Application Security
in the Age of Agentic Engineering

Jan Brennenstuhl
Principal Software Engineer

HTW Berlin · Guest Lecture · 2026

Disclaimer

These slides reflect my own views and experience. I am not speaking for my employer or any company I have worked for in the past. I encourage sharing and discussing this material, but please identify me without employer affiliation.

About Me

Jan Brennenstuhl — Principal Software Engineer, Application Security
janbrennenstuhl.eu | @jbspeakr@infosec.exchange

- Application security at a large European e-commerce platform
- ~2,000 engineers, thousands of microservices, hundreds of AWS accounts
- Focus: OAuth/OIDC, zero-trust, platform security, developer experience for security tooling

For twenty years, AppSec optimized one thing:

Finding Vulnerabilities.

It Worked Because...

- We built scanners for code, containers, pipelines, cloud
- We integrated them earlier into the lifecycle
- We called it **shift-left**

IT WORKED BECAUSE HUMANS CONTROLLED THE PACE OF CHANGE

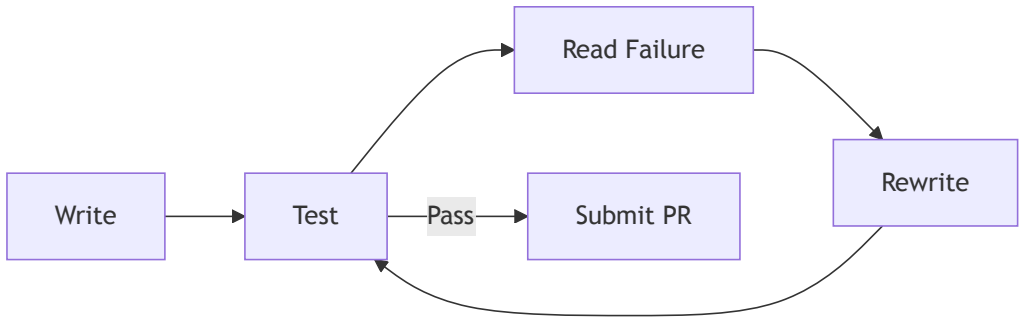
The Numbers (Early 2026)

Developers using AI agents regularly	**85%**
Using them daily	**51%**
New code that is AI-generated	**~41%**

THIS IS NOT A PILOT. THIS IS THE NEW NORMAL — AND HUMANS NO LONGER SET THE PACE

Source: addyosmani.com/blog/new-sdlc-vibe-coding

Agents Don’t Wait for Review



- Autonomous loops: write → test → rewrite → verify
- Implementation that took a team **days** collapses into **minutes**
- Findings then pile up faster than any team can **triage**

SCANNING KEEPS UP. THE FINDING-AND-FIXING LOOP DOESN’T

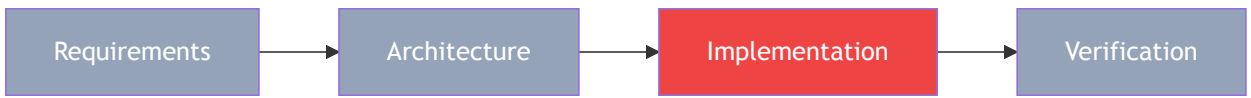
Detection is solved. Mitigation is not.

THAT GAP IS THE CENTRAL CHALLENGE FOR APPSEC TODAY

The New Lifecycle

What Agentic Engineering Actually Compresses

Uneven Compression



- **Implementation** drops from weeks to minutes
- Requirements, architecture, verification **stay slow** — judgment work

THE BOTTLENECK MOVES FROM WRITING CODE TO SPECIFYING AND VERIFYING

Agent = Model + Harness

Judgment work won't compress — so the leverage is in *how the agent is built*.

The model contributes **~10%** of the system. The harness is everything else:

- Instructions and rule files
- Tools and MCP servers
- Sandboxes and isolation
- Orchestration logic, guardrails, observability

MOST AGENT FAILURES ARE CONFIGURATION FAILURES — NOT MODEL FAILURES

Where to Intervene

❌ IMPROVING THE MODEL

- Marginal gains
- Outside your control
- Vendor dependency

✅ IMPROVING THE HARNESS

- Instructions & rule files
- Tools & MCP servers
- Constraints & guardrails
- **This is where AppSec builds**

BUILD SECURITY INTO THE HARNESS — NOT ONTO THE OUTPUT

The Economics

The structure that makes agents safe and secure is the same structure that makes them cheap.

"VIBE CODING"	STRUCTURED AGENTIC ENGINEERING
Casual prompts, no tests, no context	Schemas, tests, curated context
↓	↓
Token burn + maintenance debt + security cleanup	Higher upfront cost, flat per-feature cost

PAST THE CROSSOVER: UNSTRUCTURED APPROACHES COST 3–10× MORE PER FEATURE

Shift-Left

Necessary, But No Longer Sufficient

Shift-Left’s Original Answer

"Who should own security?" → Everyone.

- Teams run their own SAST
- Teams manage their own dependencies
- Teams configure their own infrastructure

THE RESULT: HUNDREDS OF TEAMS MAKING THOUSANDS OF SECURITY DECISIONS INDEPENDENTLY

Drift at Agent Speed

At human speed, drift accumulates over **months**. Organizations notice, schedule remediation.

At agent speed:

- Every non-standard pattern becomes **context for generation**
- Every local workaround becomes **a template**
- Every exception becomes **infinitely scalable**

THE RISK IS NOT INSECURE CODE IN ISOLATION.
IT IS ARCHITECTURAL ENTROPY MULTIPLIED ACROSS THE FLEET

You cannot scan your way out of a problem that regenerates faster than you fix it

Shift-Down

Security as Platform Architecture

A Different Question

SHIFT-LEFT

"Did this team implement it correctly?"

SHIFT-DOWN

"**Which decisions should exist at all?**"

Platform Absorbs Complexity

Traditional	Platform
Build a secure CI/CD pipeline	Standard pipeline: verify, test, scan, sign, deploy **by default**
Manage IAM policies	Short-lived credentials bound to workloads **automatically**
Harden containers	Managed base images with secure defaults **always**

SECURITY OUTCOMES IMPROVE WHEN PLATFORMS ABSORB COMPLEXITY
THAT INDIVIDUAL TEAMS PREVIOUSLY HANDLED ALONE

The Authentication Example

SHIFT-LEFT WORLD

1. Agent generates auth code
2. Scanner flags weaknesses
3. Dev reviews, fixes, rescans
4. **3 iterations, 48 hours to fix**

SHIFT-DOWN WORLD

1. Platform provides auth as abstraction
2. Agent calls an authenticated API
3. Team never implements auth itself
4. **No insecure auth path to write**

Custom Platform Resources

A team declares a `WebService`. The platform operator provisions:

- Hardened container from managed base image
- mTLS sidecars injected
- Scoped credentials via workload identity
- Egress policies configured
- Observability wired

NO DOCKERFILE. NO PIPELINE YAML. NO SECURITY CONTEXT TO MISCONFIGURE

Agents follow the path of least resistance

WHEN THE SECURE PATH IS THE EASIEST PATH, AGENTS NATURALLY PRODUCE SECURE CODE

The Paradox

Less Freedom, More Productivity

Constraints ≠ Lack of Trust

Modern cloud engineers have **far less** infrastructure freedom than sysadmins 20 years ago.

- Cannot choose hypervisor, network stack, or storage firmware
- Yet they are **dramatically more productive**
- Few miss managing physical servers

CONSTRAINTS DON’T LIMIT PRODUCTIVITY. THEY REDIRECT IT TOWARD DIFFERENTIATION

The New AppSec

From Scanners to Systems

1. From Detection to Mitigation

TRADITIONAL LOOP

Scan → Ticket → Wait → Human Fix → Repeat

❌ Breaks at agent speed

AGENTIC LOOP

Code → Scan → Feedback → Agent Rewrites → Verified

✅ Same context, no handoff

DESIGNING TOOLING FOR THE AGENTIC LOOP IS THE NEW APPSEC DAY JOB

The Security Harness

- **Security skills & instructions** consumed before and during code generation
- **Local scanning** (SAST, DAST via CLI/MCP) for agent self-checking
- **Review agents** enforcing org-specific policies
- Idea: **Shared review ledger** in SARIF format

2. From Tickets to Campaigns

Campaign-based remediation targets entire vulnerability classes at once:

- Dependency upgrades across hundreds of services
- IaC misconfigurations fleet-wide
- Repeated insecure patterns at scale

AppSec **designs** the campaign. Agents **execute** the remediation PRs.

IMPACT PER ENGINEER-HOUR GOES UP BY AN ORDER OF MAGNITUDE

3. From Enforcement to Architecture

Before	After
50-page security standards	Platform defaults
PDF guidelines	Service templates
Manual review gates	Policy-as-code rules

Traditional AppSec **reviews** decisions made elsewhere. Shift-down AppSec **decides** which decisions should exist.

CODING AGENTS DON’T READ PDFS. THEY CONSUME CLIS, MCPS, AND CONSTRAINTS

Risk Budgets

Security Meets SRE

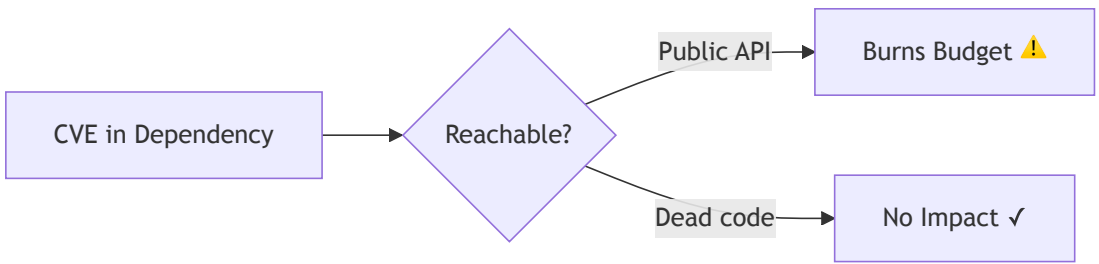
Risk Budgets

Engineering teams already manage **error budgets** for availability. Security can join that model.

- A **risk budget** caps exposure time for high-risk, reachable vulnerabilities
- Like SLOs: measurable, actionable, owned by business
- Fix what burns budget, defer what doesn’t

WITHOUT QUANTITATIVE TARGETS, NOTHING GETS PRIORITIZED. RISK BUDGETS GIVE AGENTS A DECISION RULE

Reachability Analysis



- Same CVE in dead code vs. behind a public API = **completely different risk**
- Risk budgets only work if we distinguish **real exposure** from noise

SECURITY BECOMES A SHARED OPERATIONAL CONCERN WITH MEASURABLE TARGETS

Agent Security

The Tool Is Also an Attack Surface

The Agent as Attack Surface

Every agent task is an **untrusted workload** running arbitrary code.

Agents consume external inputs — each a potential prompt injection vector:

- Dependency metadata
- Issue descriptions
- PR comments
- Documentation

RESULT: A LARGE INTERNAL REMOTE CODE EXECUTION SURFACE THAT MOST ORGANIZATIONS HAVEN'T ADDRESSED

Sandboxing & Identity

SANDBOXING

Micro-VM: Firecracker, Docker Sandboxes — disposable environments, full kernel isolation

Kernel-level: nono, Landlock/Seatbelt — irrevocable filesystem allow-lists, zero overhead

IDENTITY

Agents should **never** hold:

- Persistent IAM roles
- Long-lived API keys

An **identity broker** issues short-lived, task-scoped credentials with automatic revocation.

SHIP BOTH AS PAVED-ROAD DEFAULTS. DON'T LEAVE TEAMS TO FIGURE THIS OUT ALONE

AppSec Roles

From Pentesting to Platform Engineering

AppSec Through the Ages

Era	AppSec Identity
2005	Penetration testers
2015	Scanner operators
2020	Shift-left consultants
2025	Security engineers
2030	**Platform engineers & risk systems designers**

SECURITY EXPERTISE BECOMES SOFTWARE

What This Means for You

1. **Software engineering fundamentals are non-negotiable**

   Build, ship, and operate software. Distributed systems, IAM, observability, IaC are core skills.
2. **Learn to think in platforms**

   Design systems where broken policies **cannot be expressed**.

   Ask "why does this decision exist?" before "was this decision made correctly?"
3. **Get comfortable with the economics of risk**

   Error budgets, reachability analysis, exposure windows connect security to business decisions.

Shift-left taught us that everyone owns security

Shift-down teaches us that not everything should remain a choice

Key Takeaways

1. **Detection is solved; mitigation at agent speed is the challenge**
2. **Agent = Model + Harness** — build security into the harness
3. **Shift-down** removes decisions, not just catches mistakes
4. **Risk budgets** make security measurable and actionable
5. **Agents are attack surfaces** — sandbox and broker identity
6. **AppSec engineers become platform engineers** who ship security as software

Thank You

Questions?