



7 strategies for scaling product security



QCon 2018 – New York City
Angelo Prado, Senior Director
Jet.com | Walmart



about me

12+ years of experience in software development and **Leading Product Security** teams at Jet.com, Salesforce and Microsoft

4 times **Black Hat Speaker**, co-author of **10+ CVEs** including the **BREACH** attack (SSL Side Channel)

Currently leading a product security team across two continents, assistant professor in Spain at Comillas University, advising security startups and non-profits

earlier career attempts...



what is
product security



Product Security teams are the **guardians of customer data**, fixing and preventing security vulnerabilities. Inclusive of much **more than just code**. Product Security covers the **full service** and how your customers use and interact with it securely. It goes **beyond securing the underlying software** and includes operational responsibilities.



why do we need
Product Security?

core mission



prevent vulnerabilities



build effective automation



perform security reviews



harden the product



who needs

product security?



you do.



we do.

Security is reflected in how products are **built and operated**.
Product Security should be **engaged with customers** and partners.
Engineering teams must have a **consistent interpretation** of the
security posture and **secure development lifecycle**.

7 strategies to scale

Building Product Security from the ground up

**prioritize relationships
and establish a non-
blocking function**

01

SERVICE CATALOG



design
reviews



automation
services



security
testing



vulnerability
management



training &
research



Product Security should be a **lean, effective, non-blocking** technical assessment function

rules of engagement



prioritize relationships **over bugs**

The number of teams and individuals you interact with will keep growing – In connecting with other human beings, align priorities and exercise empathy



be thoughtful about **prioritization and risk**

Security isn't always #1 - If you want to build a relationship with someone, you need to know their priorities. Develop a narrative that resonates with them



be pragmatic and **solicit feedback**

Security should not block shipping, and it shouldn't be reactive. We triage vulnerabilities based on severity, but not all bugs are considered equal. Listen to the teams you support and proactively seek improvement opportunities



Even the most professional, security-conscious developers take it personally occasionally. It's not their fault. A regular drumbeat of "you're doing it wrong" will discourage anyone. **Developers usually want to do the right thing** - Promote thoughtful solutions that scale and balance technical capabilities with product **usability**

the hacker mindset

breaker mindset

substantial knowledge of application-level attacks and flaws

builder mindset

strong knowledge of software development, browsers, cloud services, network, crypto and defense strategies



aptitude

open source contributions, research, publications and bug bounty recognitions

soft skills

effective communication skills and the ability to influence and communicate with engineers



“

Run security like a business:

Sorry, Mr. Hacker, this just isn't working out...

02

**invest in vulnerability
management, metrics
and reporting**

vuln management

ship it!

the fix is released to production and required comms are handled

deliver bug

a vulnerability is found, an issue is created and assigned to the team backlog

verify fix

the fix is validated in an staging environment, including different variants

work on a fix

the engineering team works out a fix, assisted by the security contact



agile workflows

proactive signoff

product teams are notified of any security issues and provided with hardening recommendations



continuous testing

security owners deploy automation and perform gray-box testing



threat modeling

security owners identify weaknesses and mitigations



security owner

each product security engineer owns a portfolio of applications



design review

security owners are responsible for attending design reviews

vulnerability notifications

usability is a key

the priority, description of the vulnerability, and the remediation target date should be emphasized

make it actionable

there should be a clear call to action on any vulnerability, indicating proposed remediation

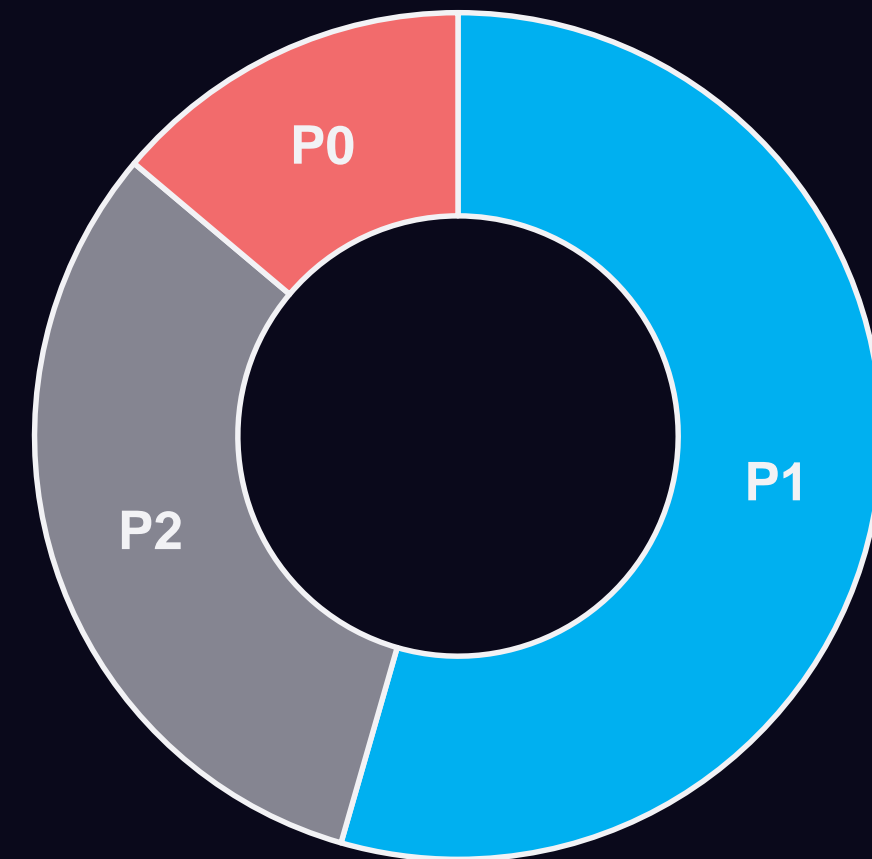
make it relevant

ensure the right engineering team and security owner receive notifications for their products



prioritize
responsibly

- Critical Priority (P0) – 7 days SLA
- Medium Priority (P1) – 30 days SLA
- Low Priority (P2) – 60 days SLA



SLA process

starts on **delivery**

only after the right product team has been identified
and their engineers notified

resets if misrouted

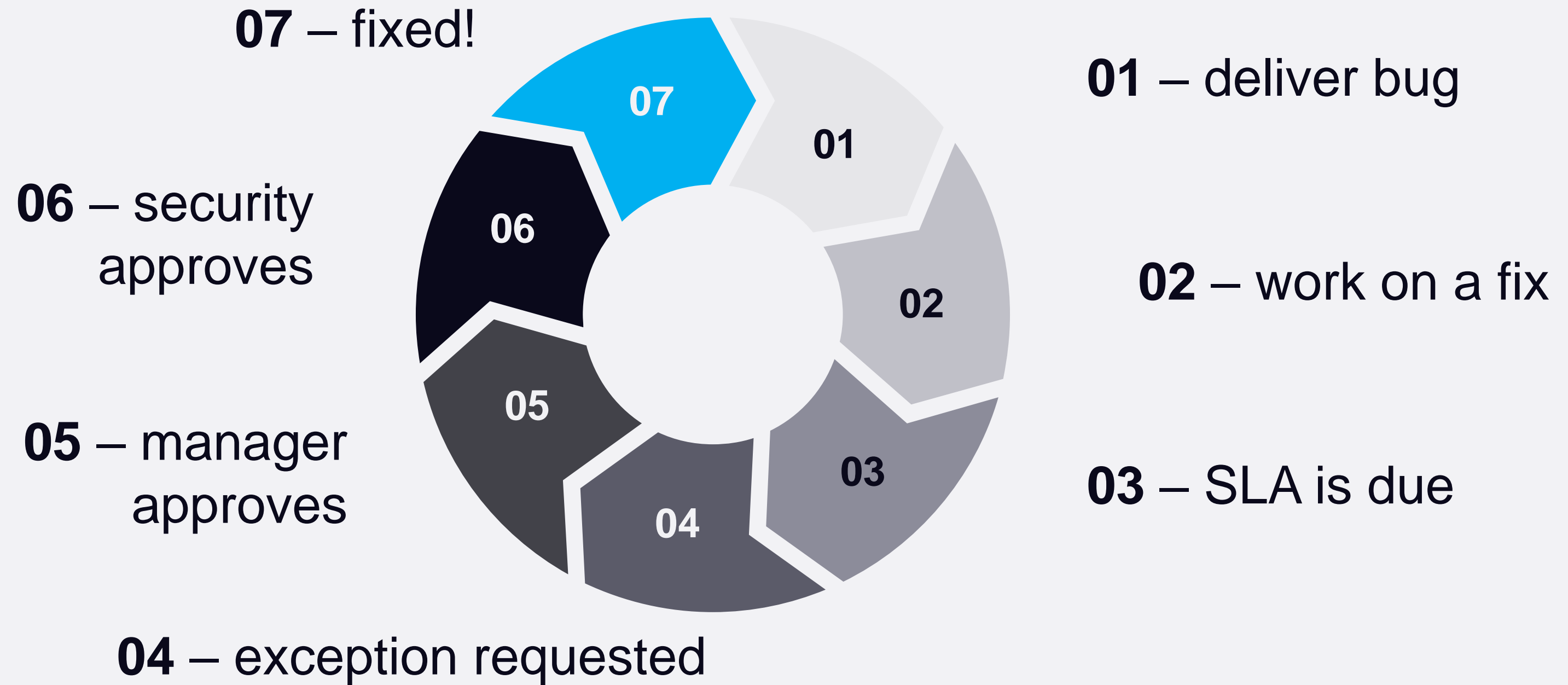
teams should not be penalized for incorrect delivery

requires **exception workflow**

engineering manager and security manager **approval is required**
if a security issue cannot be remediated within the agreed SLA



vulnerability management



track release progress



open bugs

these are bugs where no action has been taken



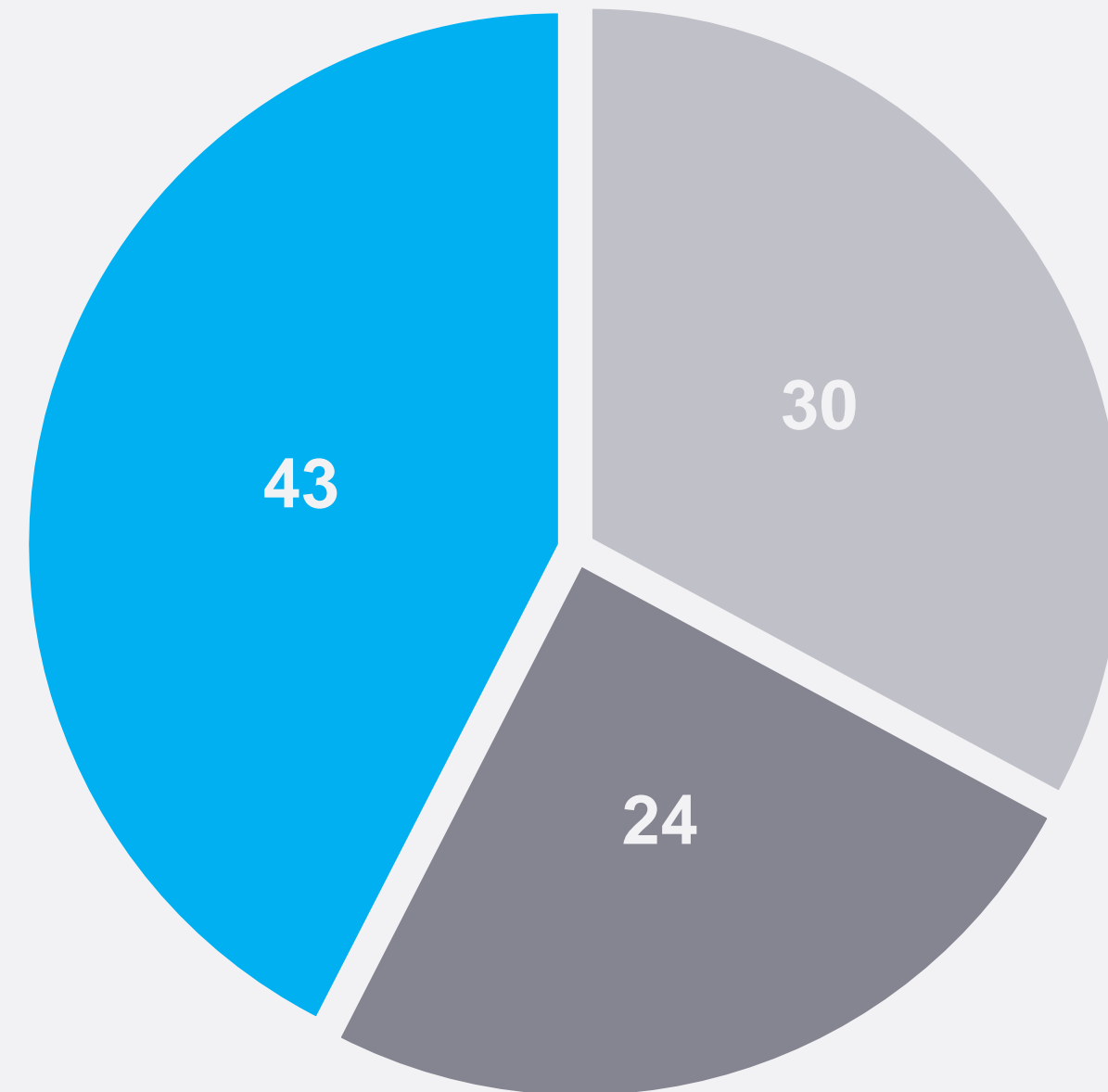
in progress

bugs actively worked on

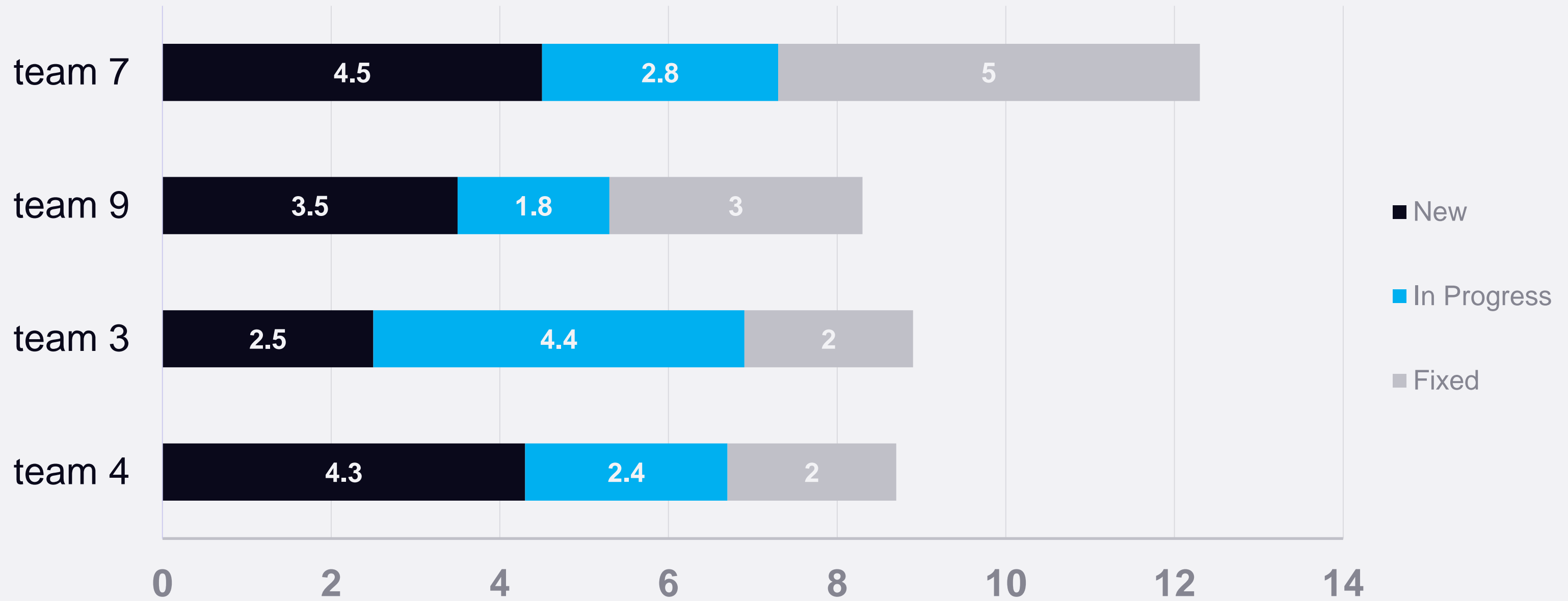


resolved

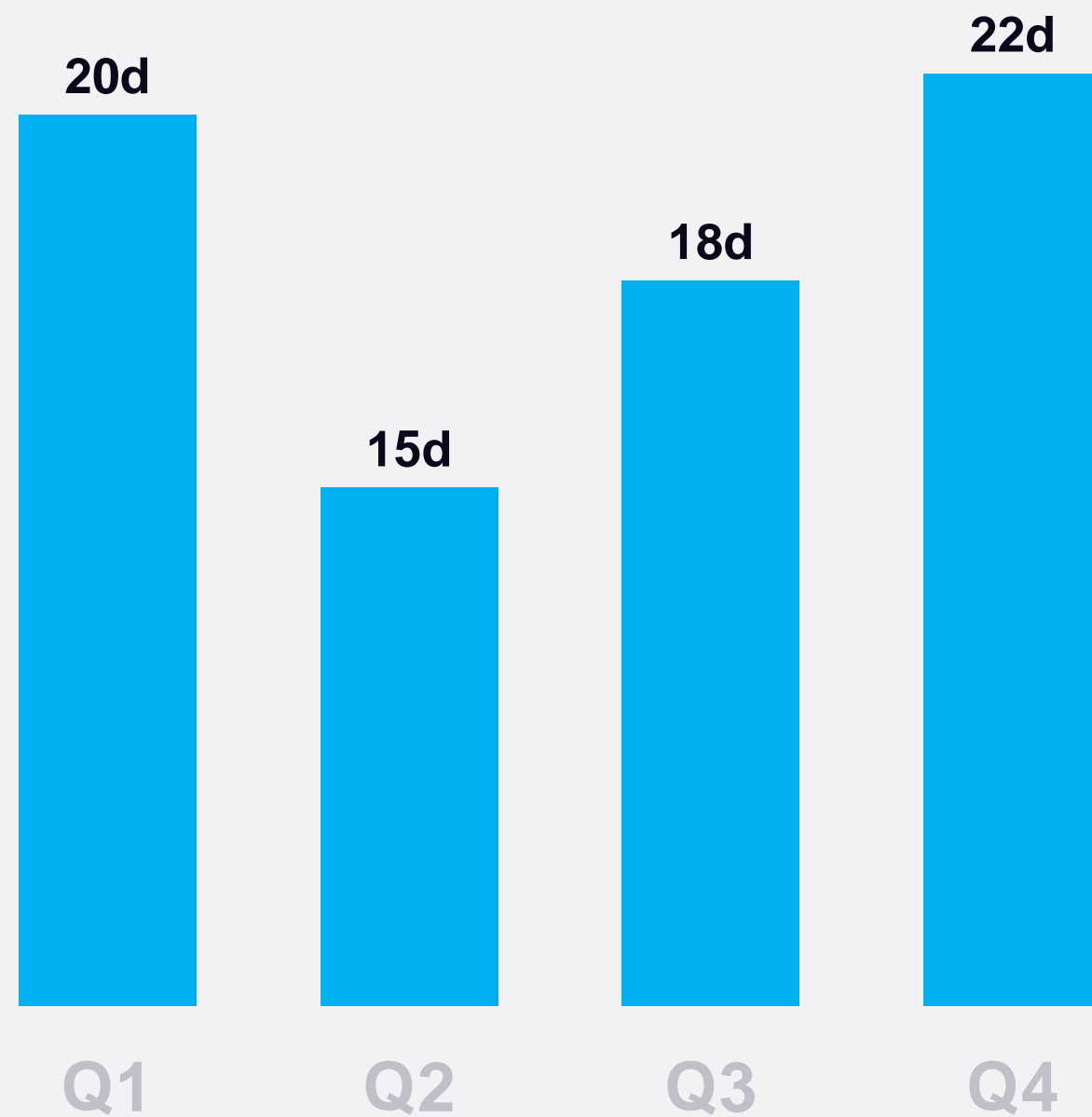
fixed & verified



intake time / time to resolution



vulnerability lifetime in production



measures time since a team starts working on a bug until a fix is deployed

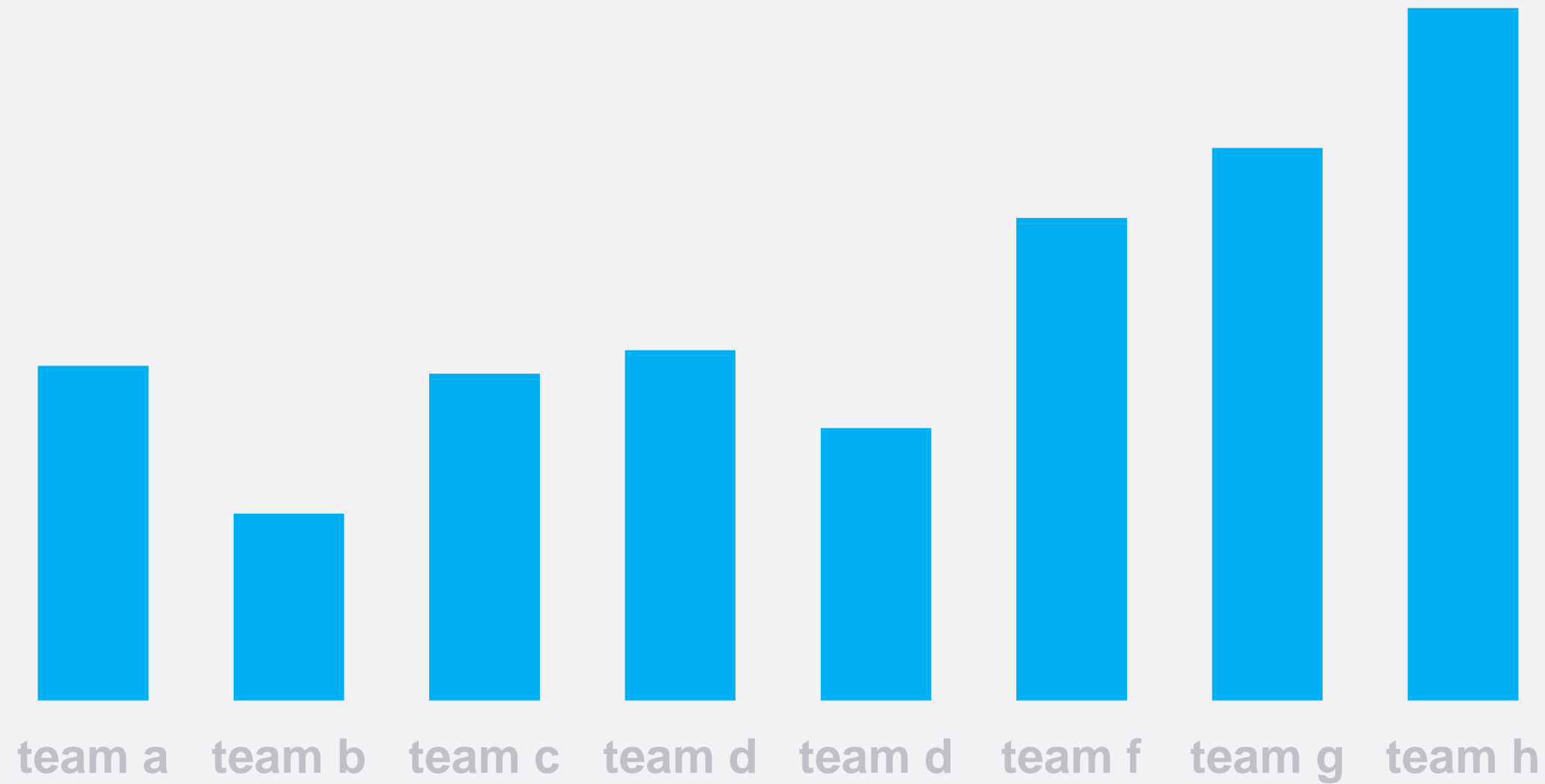


cross-referenced with pull request size, it can help understand complexity and exposure



starts when a vulnerability is introduced in production, at deployment – this metric measures the effectiveness of your product security program.

SLA adherence benchmarks



recognizes teams that **prioritize security**



highlights teams **requiring assistance**

SLA trends over time



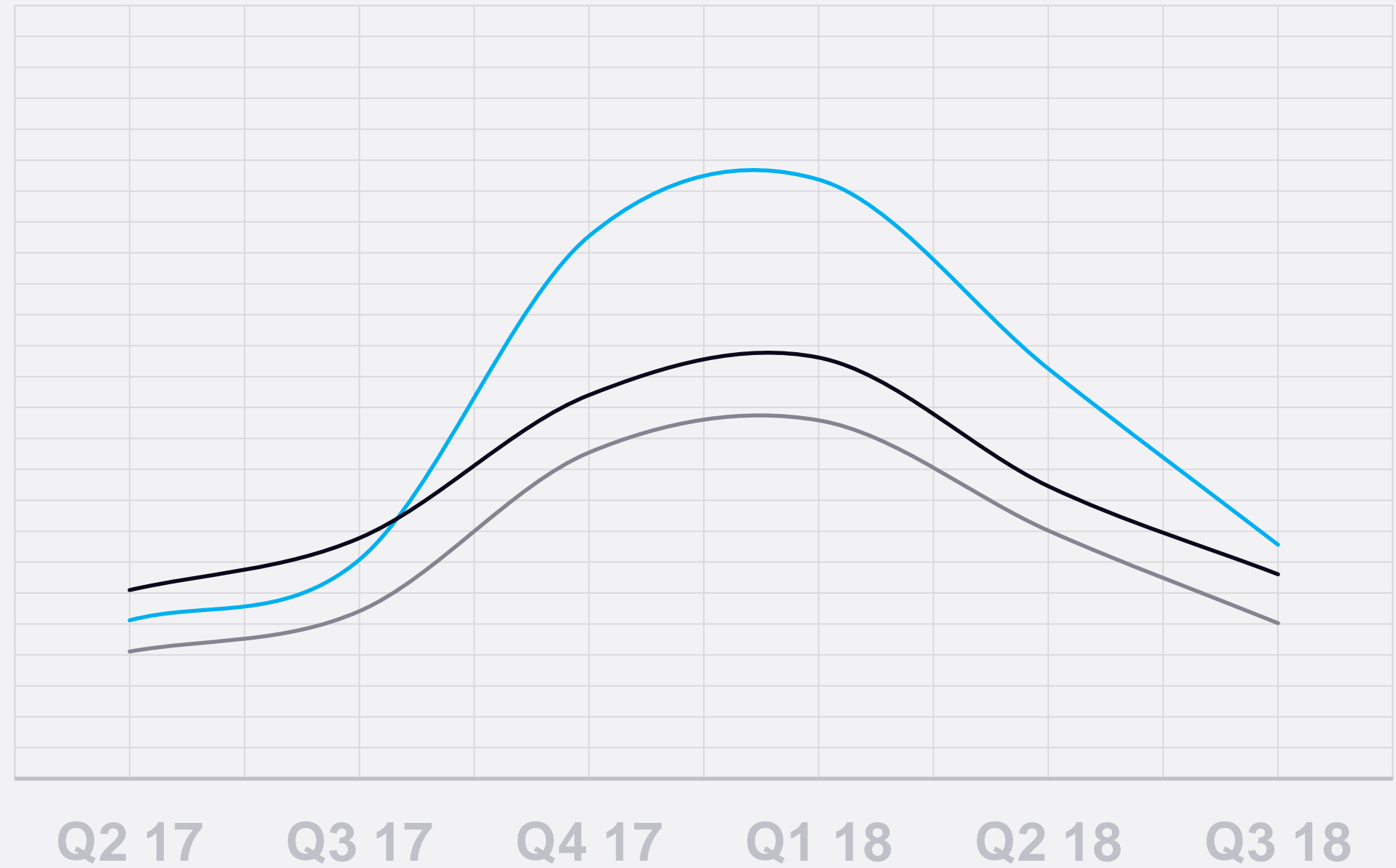
critical issues



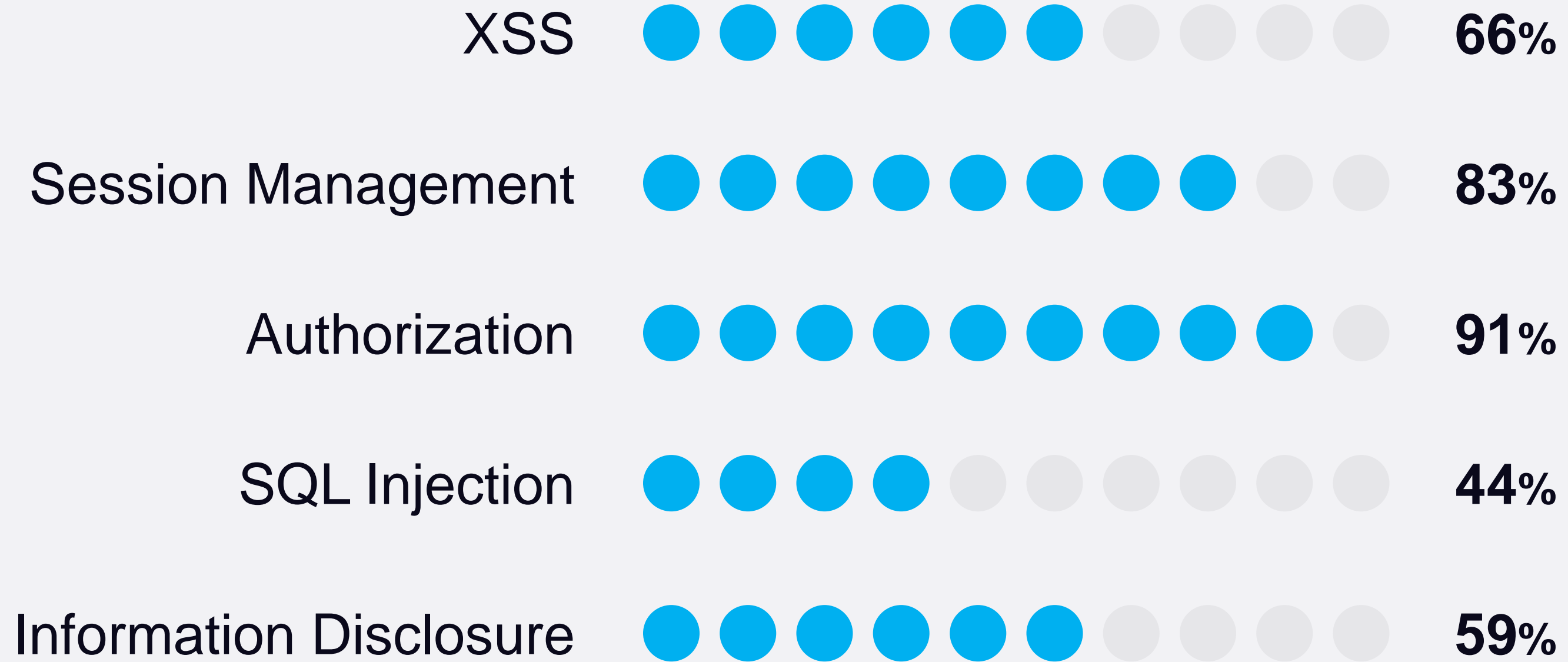
all issues



low priority



Benchmark by vulnerability type



90%

developers received
security training

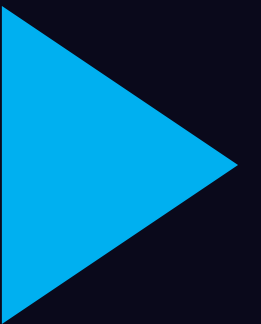
73%


teams have automated coverage
SCA | RTA | DAST



03

**automate
all the things**

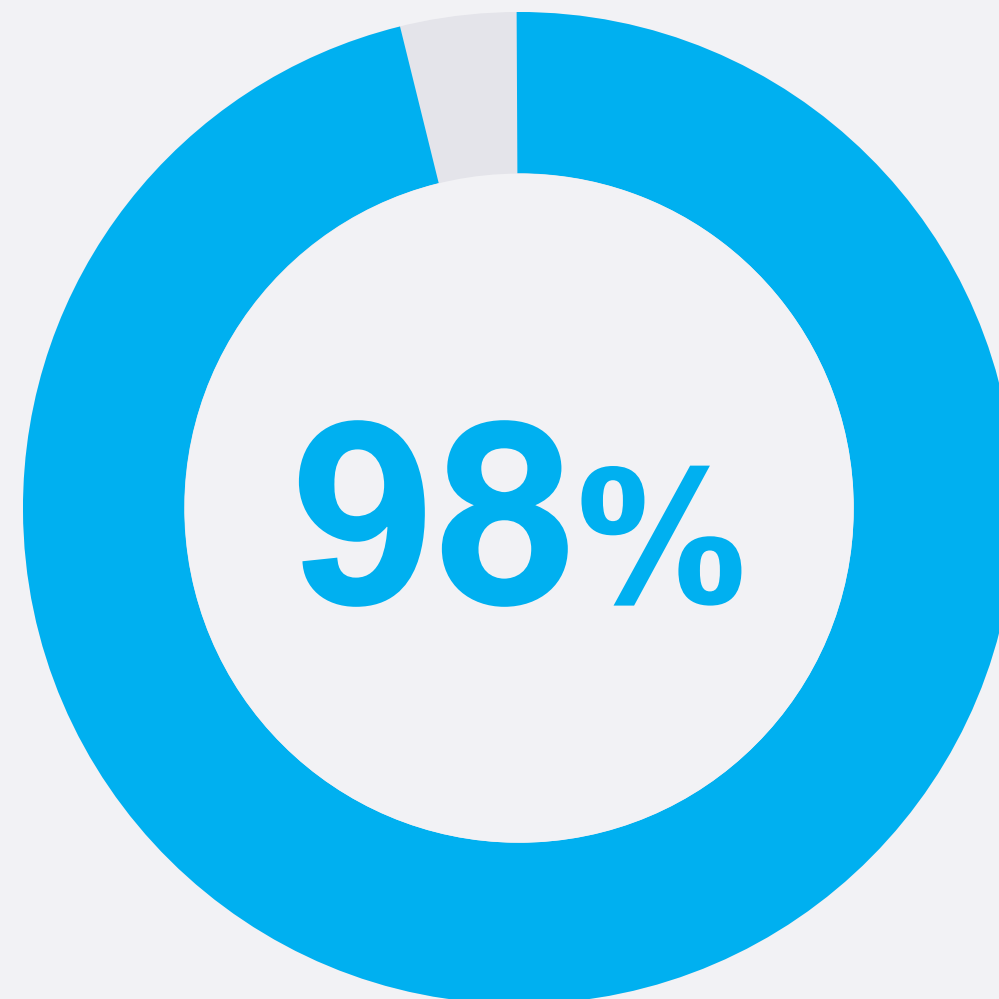




**“ Complexity is the enemy of security:
*Secure by default or die not actually trying***

**scaling source
code reviews**

**we cannot
review**



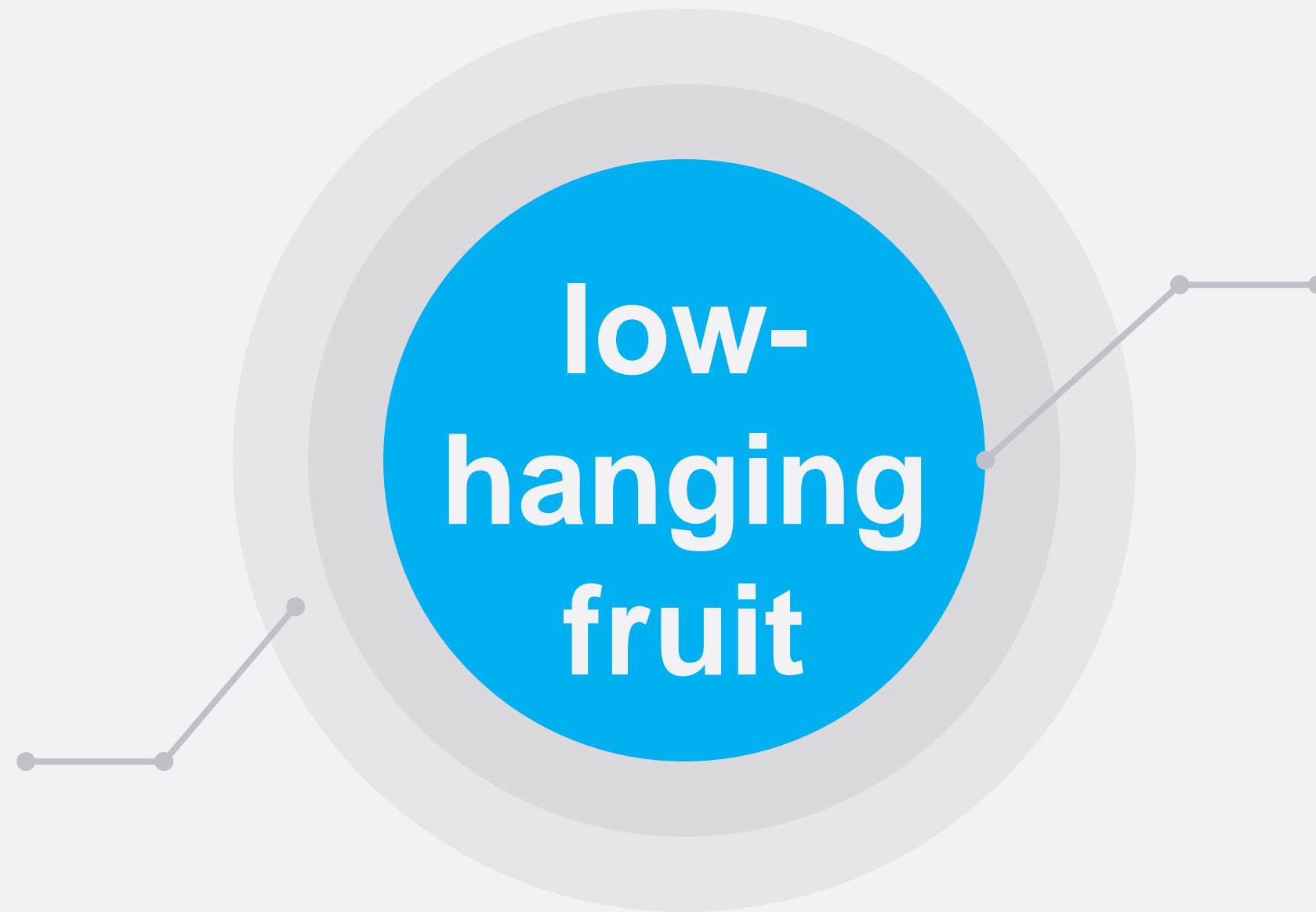
**of check-
ins**

「40%+」

of security vulnerabilities
can be **automatically**
detected

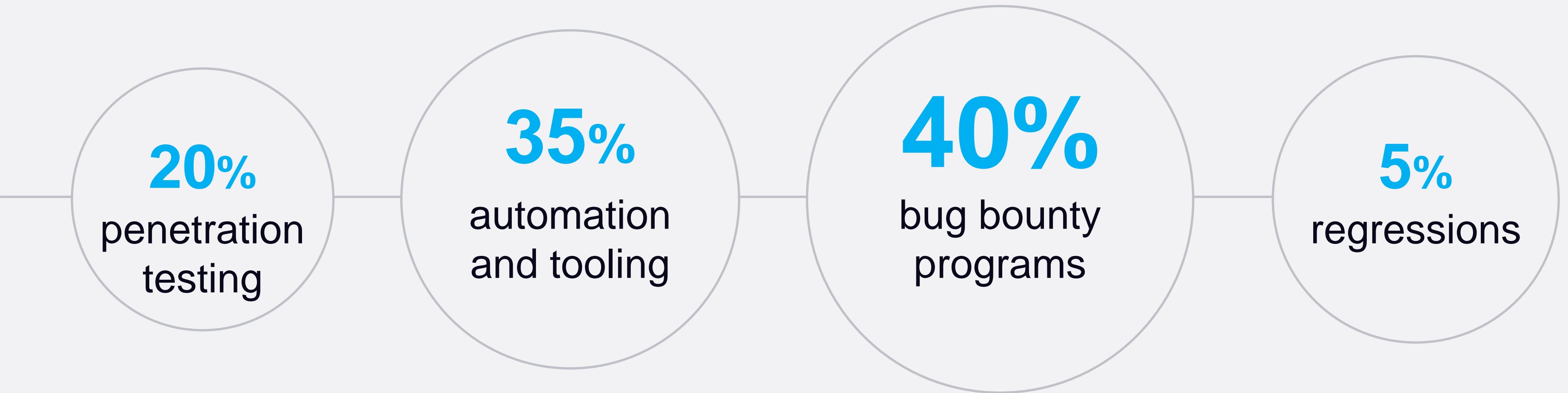
vulnerability
demographics

manual
testing **required**

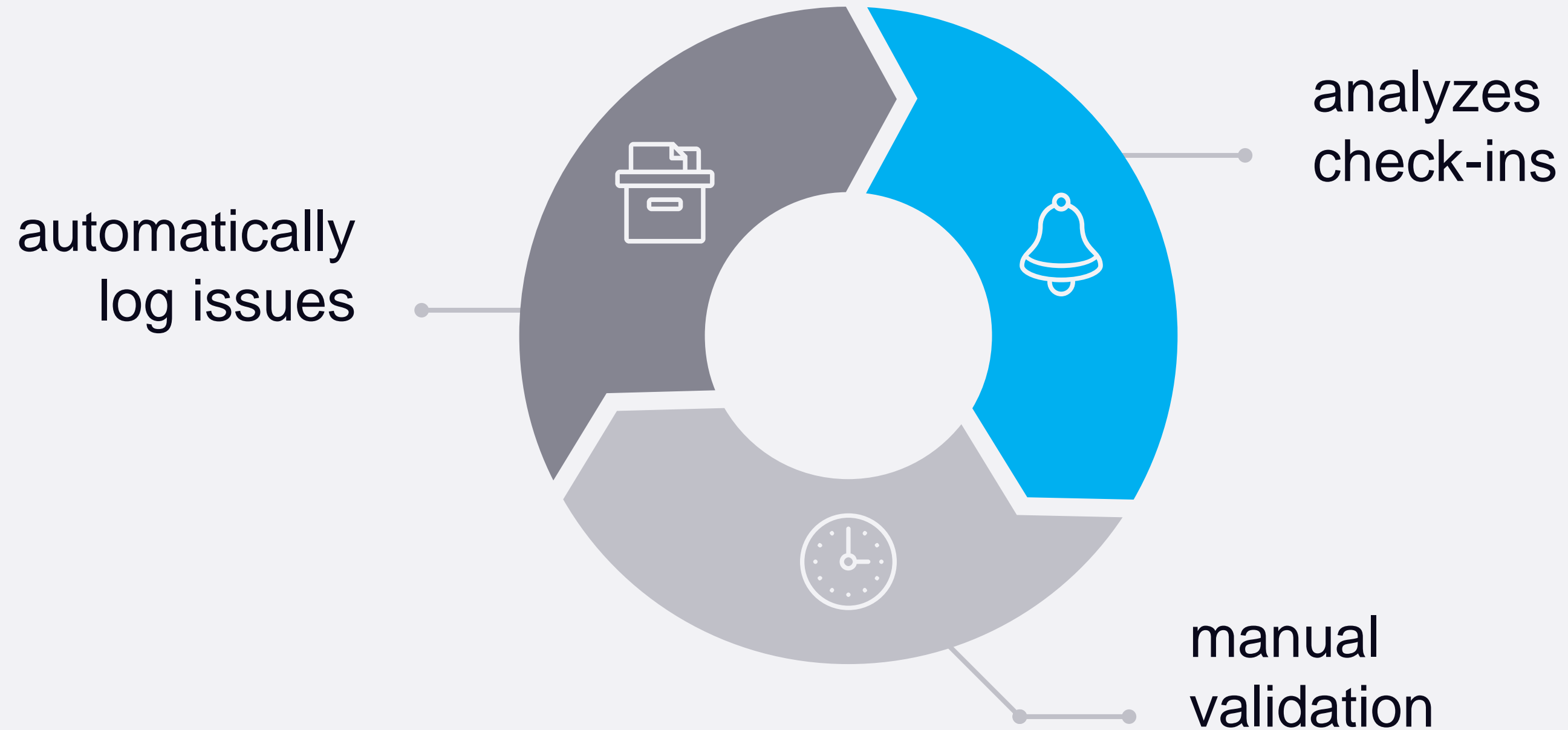


auto
discovery
possible

vuln sources



CI/CD integration



types of automation

static code analysis

analyzes source code flows and incremental check-ins with known rules

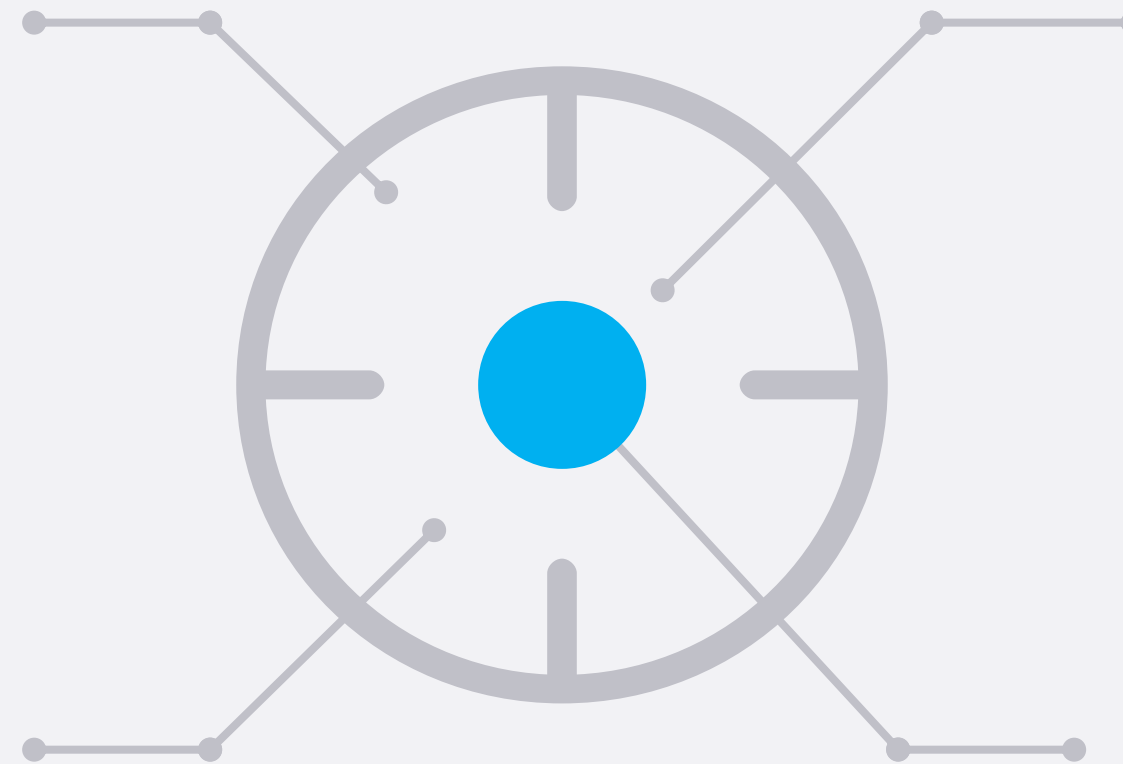
dynamic analysis

capable of testing web service and application endpoints in production

runtime self-protection

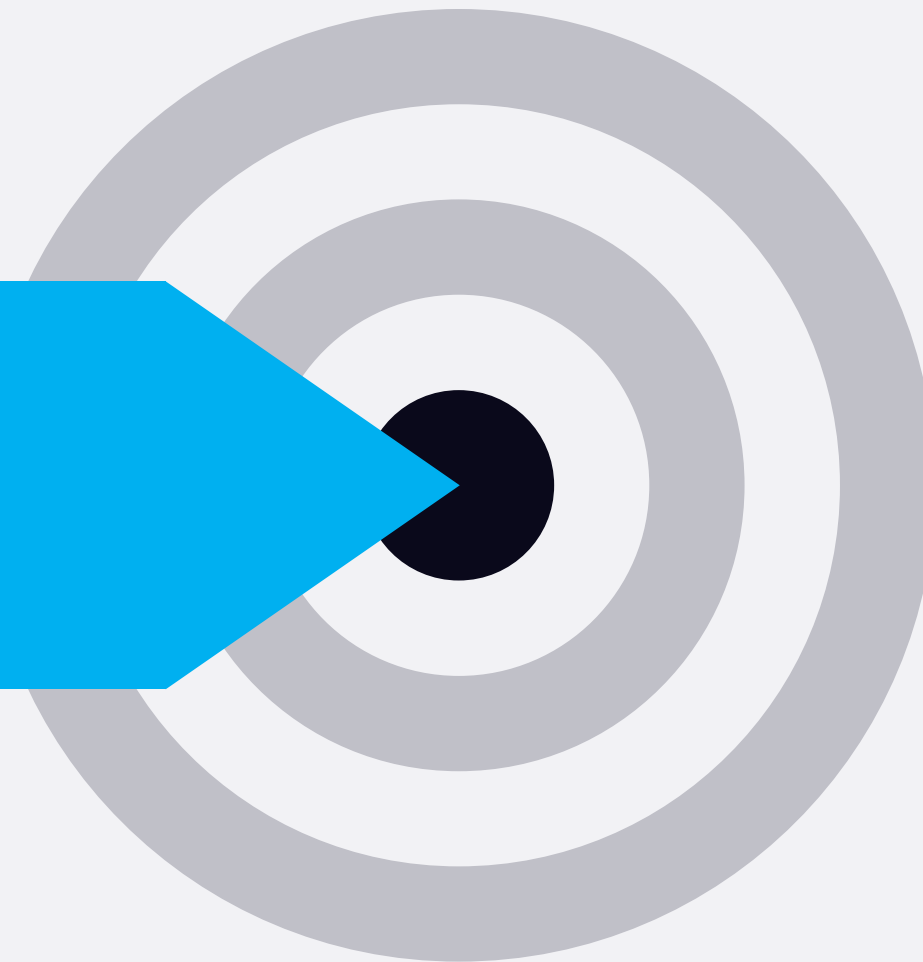
understands when an application's normal flow is being exercised by a malicious actor

actual vulnerability



Vulnerabilities in Third-Party Libraries

A solid third-party library program is required to review exploitable vulnerabilities and dependencies. Monitor CVEs and public exploits.



successful automation

false positives

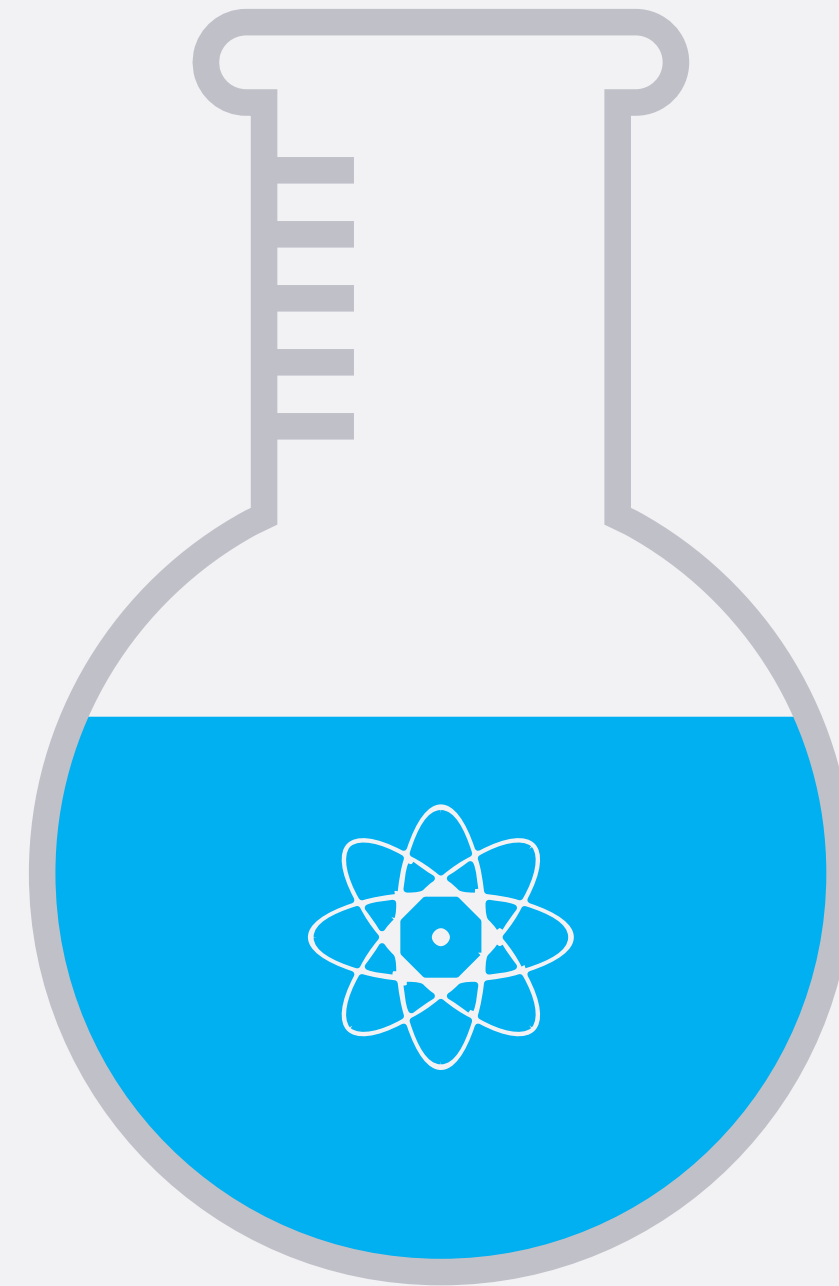
not actual vulnerabilities

acceptable risk

things that are technically valid but we are willing to live with due to mitigating controls or exploitability

issues we care about

Important, exploitable vulnerabilities



◀ 04

**Invest in
product hardening**



「awkwardness」

That period with an API
after you know what you
can do but before you
know what you should do

The Kaminsky Dictionary

nailing the fundamentals

01 HSTS & CSP
HTTP Strict Transport Security
and Content Security Policy

02 Device Fingerprinting
Stopping account take-over attempts and
using second-factor Auth smartly

03 Secret Management
Storing secrets securely

04 Proactive Controls
Providing users and admins with
management controls and visibility

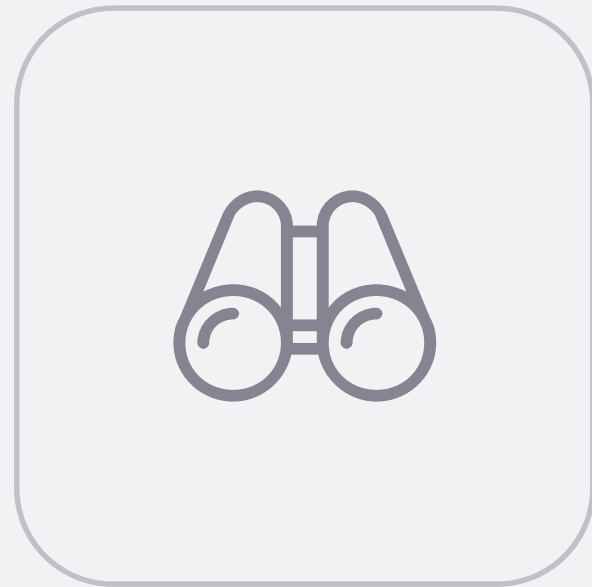
reducing the attack surface



HSTS, CSP & Expect-CT

Ensuring that all requests are done with strict transport security and that rogue certificates are not being used (certificate transparency). Content Security Policy enables us to filter out insecure content, avoid referrer leakage and in general block malicious JavaScript from executing

secret management



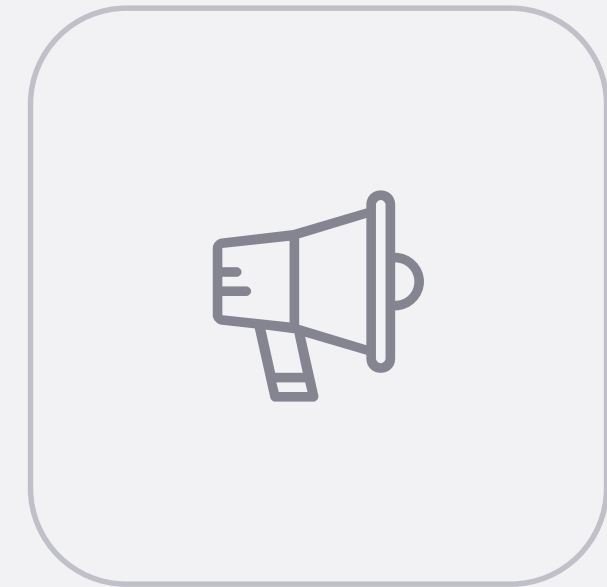
identify secrets

use rules & regular expressions
implement automatic validation



store securely

key management system
(key vault with HSM)



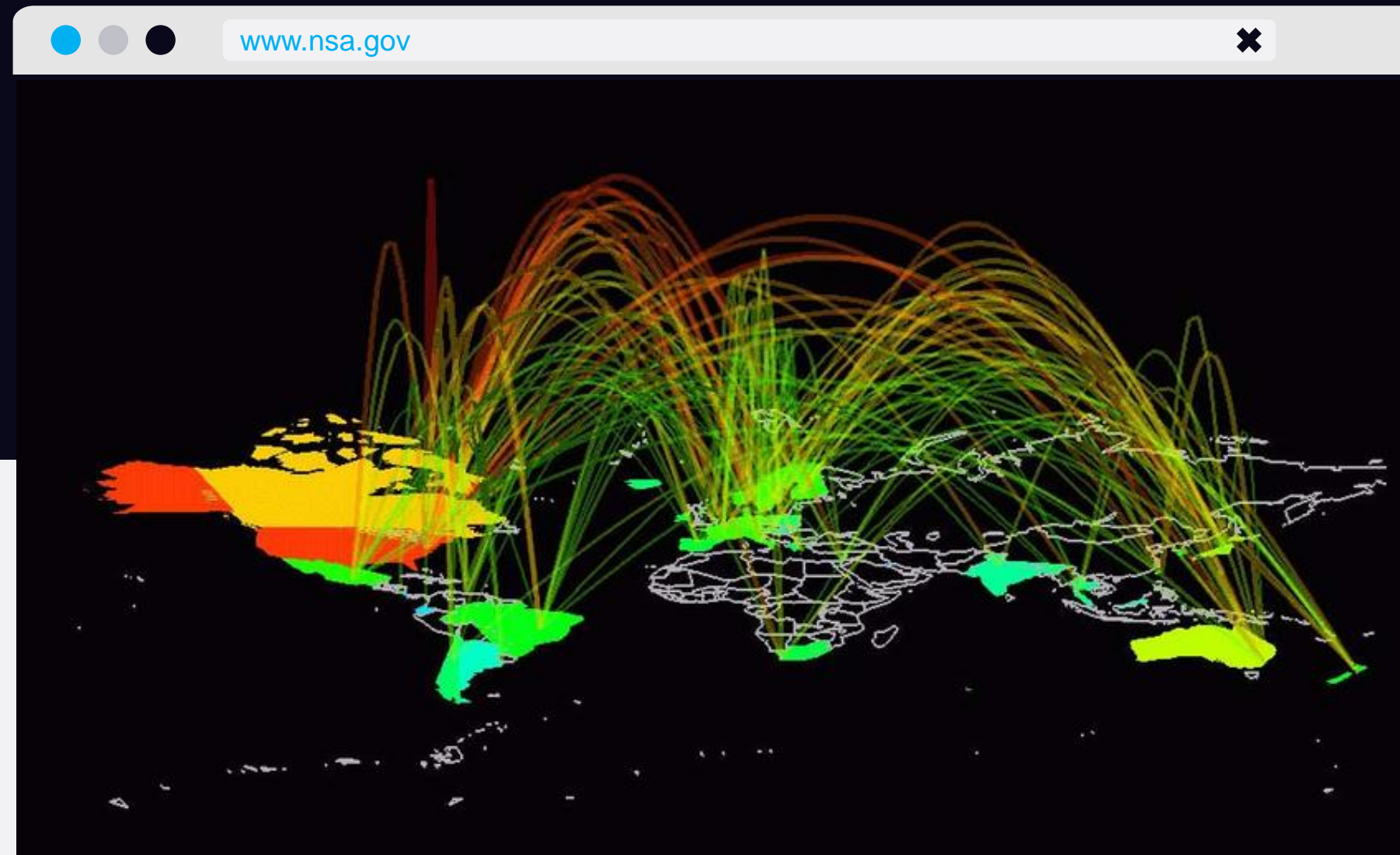
rotate secrets

automatically perform key rotation


session management

Login 
History

Device & 
Location



 Apps / OAuth

 Active Sessions

device fingerprinting



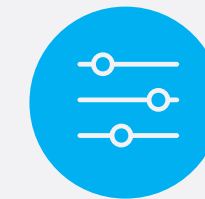
linked to the user

fingerprints are stored over time and attached to a given user identity



unique

prioritize features with a higher weight, more specific to your users



adaptive

understand that certain capabilities for the user-agent can change

smart and effective implementation

Proper device fingerprinting combined with behavioral and geolocation analytics enables you to perform contextual **two-factor authentication** via SMS or one-time links / tokens via email, reducing false negatives and false positives

proactive

Define Security Requirements



Leverage Security Frameworks



Secure Database Access



Validate Inputs & Escape Data



controls

Enforce Access Controls



Protect Data at Rest & in Transit



Implement Secure Logging



Handle Errors & Exceptions

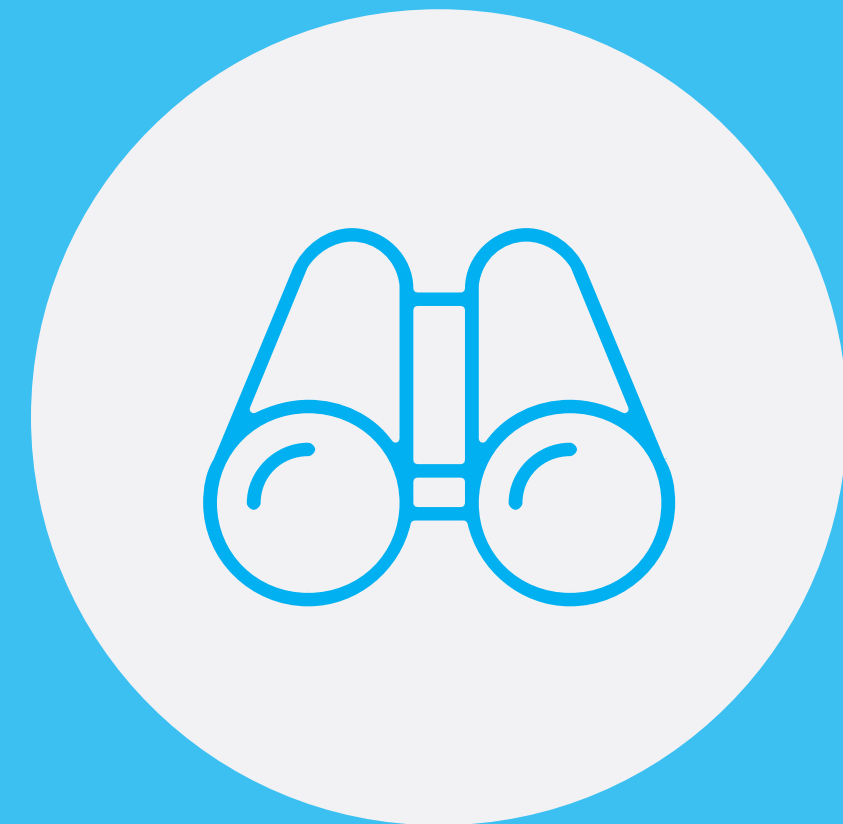


05

**create a mature
education & awareness
program**

threat modeling

Learn to think like a hacker and identify threats and security objectives. Identify flows, mitigations and make informed decisions about residual risk.





self-guided training

deliver **secure coding guidelines** that are relevant to the our organization's languages and frameworks

at a minimum, common attack patterns, secure storage, cloud security and secure feature design should be covered



classroom training

- Clear **secure coding guidelines**
- **Real-life libraries & frameworks**
- Previous **vulnerability examples**
- Actionable code snippets

Keep it **relevant!** i.e. NodeJS developers don't need to know about XML injection and heap overflow exploitation

security champions

shared accountability

programs like this help you scale as engineering organizations outnumber security engineers

Recognize and reward good behavior across all roles



06

**leverage the collective
skills of the research
community**

**why do I need a
Bug Bounty Program**



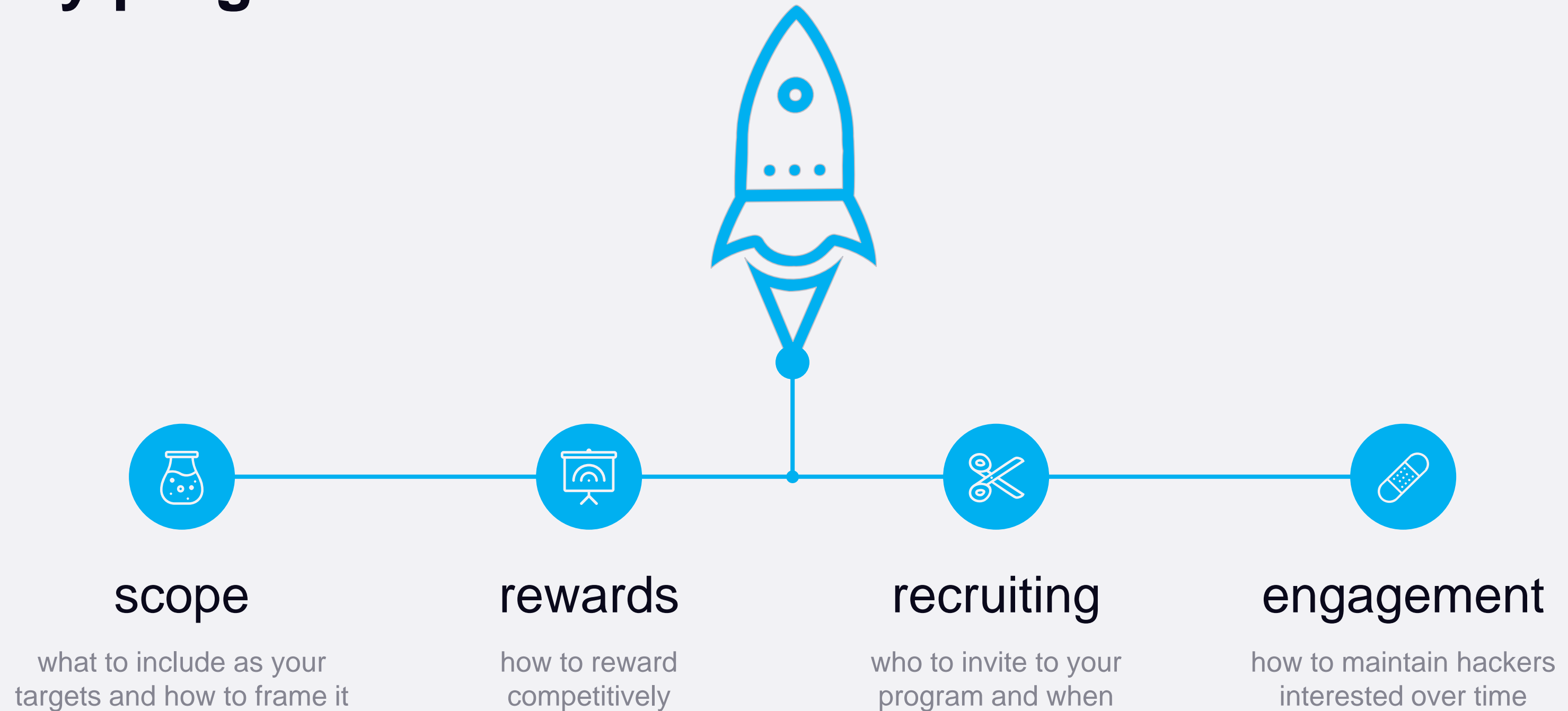


Everything fails.
Even things that
make everything
fail.

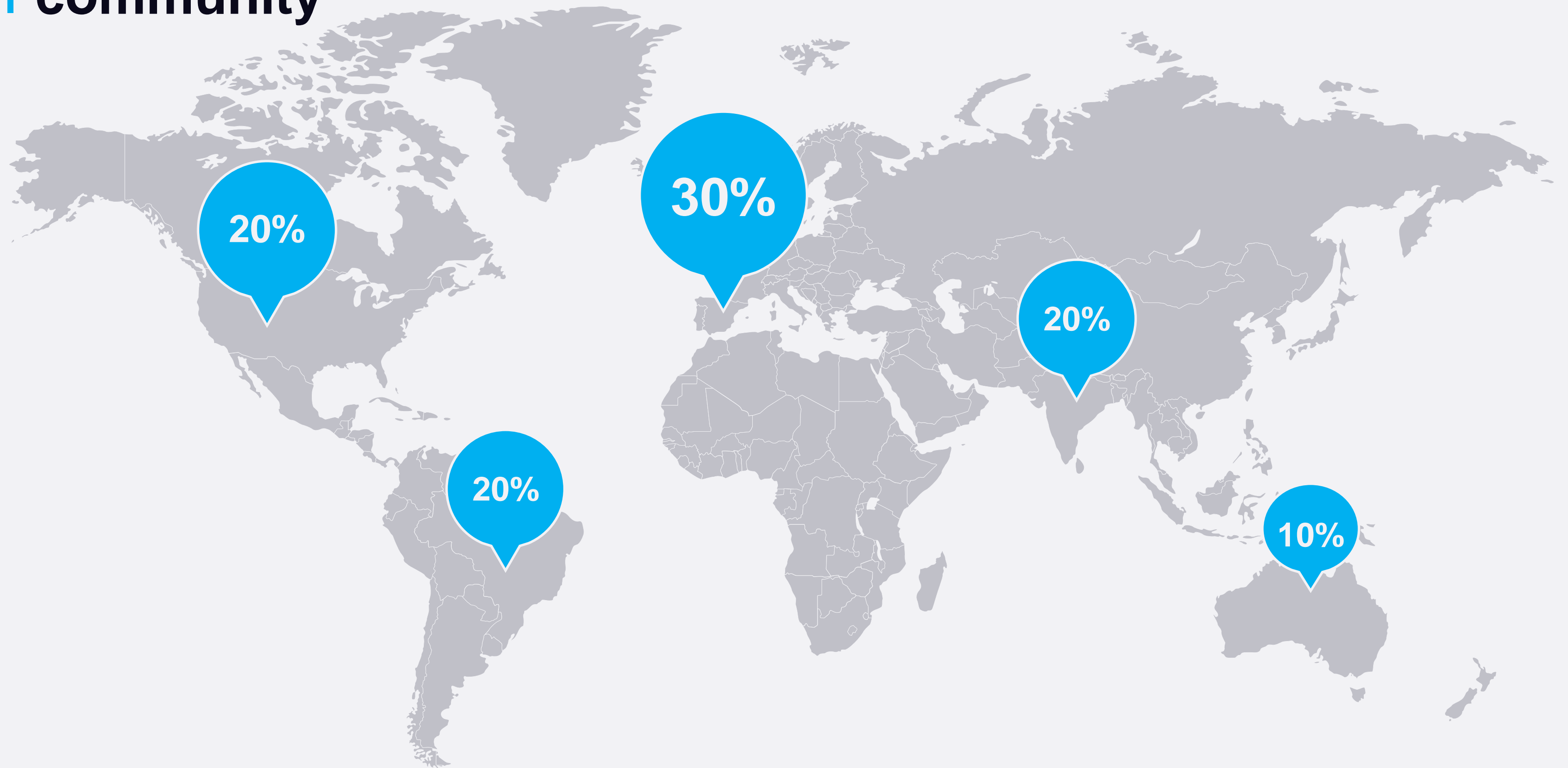
Dan Kaminsky

”

launching a bounty program



| a global
community





Hackerone



Over 170,000 hackers participating
Over 70,000 vulnerabilities found
Over \$30 million paid in bounties

Data as of June 2018

Source: HackerOne

engage your top researchers

Fly them to Vegas and keep them hydrated. Be **transparent** and overcommunicate. Keep them happy. Fly them to your HQ. **Recruit them if necessary.** Be prompt, reasonable and technical. Run recurring **promotions and challenges.**





Private programs enable you to increase signal to noise ratio. **VIP programs drive retention.** Consider **researcher circles** for knowledge sharing. Recruit from active programs. **Reward competitively.** Defuse escalations / disclosure. Resource your program.



07

**deploy a solid SDL
and maturity model**

six steps for a good SDL

design

Threat Modeling
Design Reviews

build

Static Code Analysis
Code Reviews

verify

Penetration Testing

release

Dynamic Testing
Bug Bounty

ownership

Patch Management
Remediation
Pen-testing

learn and refine

Retrospective
Planning





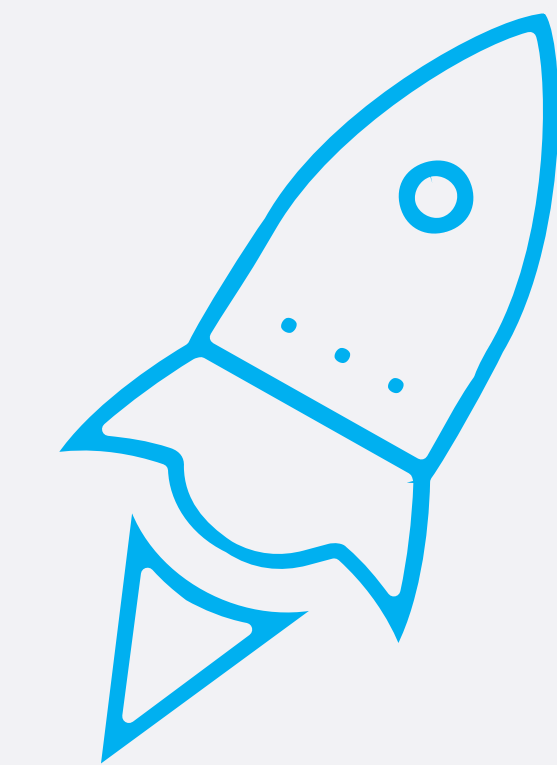
maturity model

evidence-based **framework** for evaluating the **overall security stance of a business unit** or new acquisition. Provides an authoritative and consistent roadmap for the advancement of a the organization's overall product security posture. Should be **meaningful and objective**.



Another day, another layer of abstraction

maturity model



level 1 – initial

Application Login/Admin Interface Inventory – Continuous Dynamic Application Scanning – Customer Data Inventory – HTTPS By Default – Legacy Source Code Review & Remediation – Product Security 3rd Party Assessment – Strong Password Hashing

1

Q1 Y1

Q3 Y1

Q1 Y2

Q2 Y2

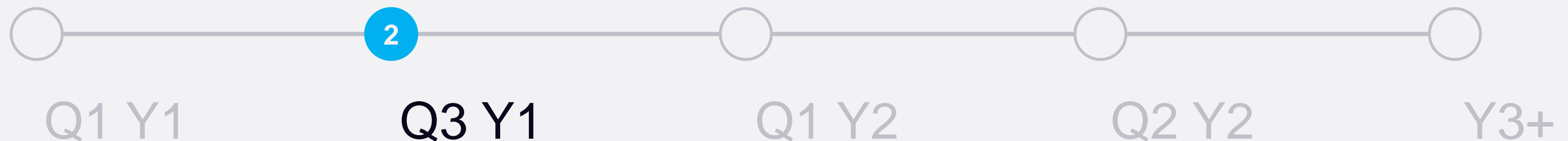
Y3+

maturity model

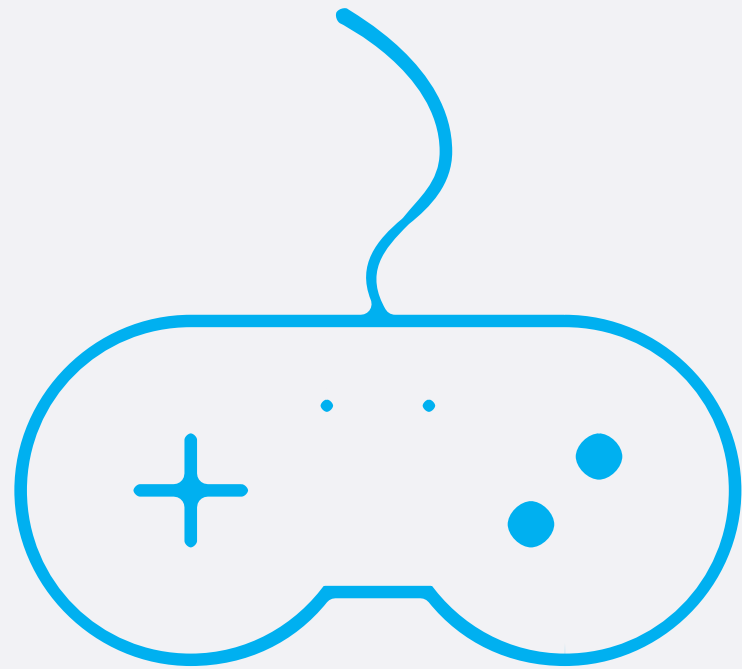


level 2 – defined

Basic Logging for Security Events – Client Software is Signed – Encryption keys not stored in source control – Security Requirements for New Features and Designs – NGWAF deployed for Web + API endpoints – In-House Manual Testing of Codebase / App – No "Roll-your-own" Cryptography – Security Tools Run Against Codebase / App On Release – Strong Session Management (AuthN/AuthZ) – Strong Encryption Standards

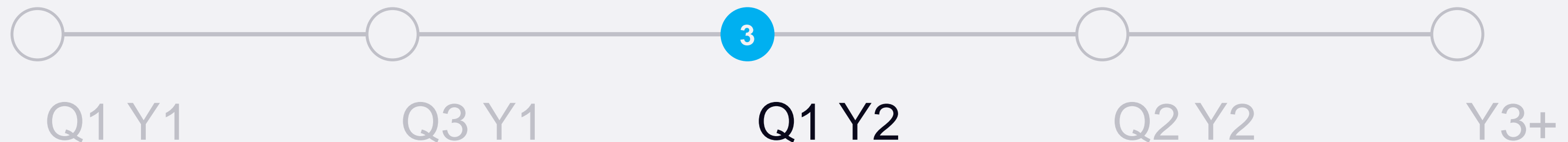


maturity model



Level 3 – managed

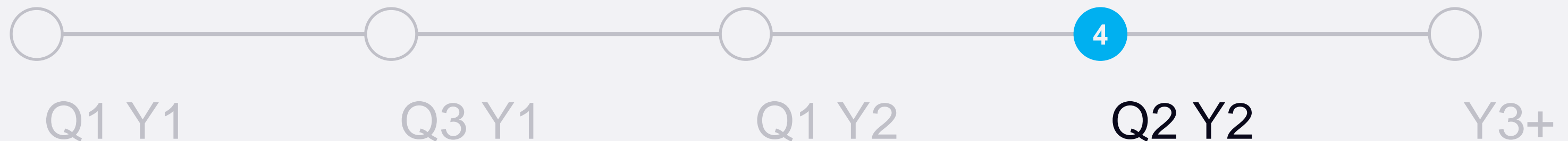
Enhanced Application Logging – HTTPS-Only (HSTS) – Inventory of open source – SLA + Signoff or Equivalent Control (90% > Adherence) – Source Code Check-in Monitoring – Strong Multitenancy Controls – Multi-factor Authentication – Strong Secrets Storage – Strong Session Authentication/Authorization – Threat Modeling of New Features – Role-Based Access Control



maturity model

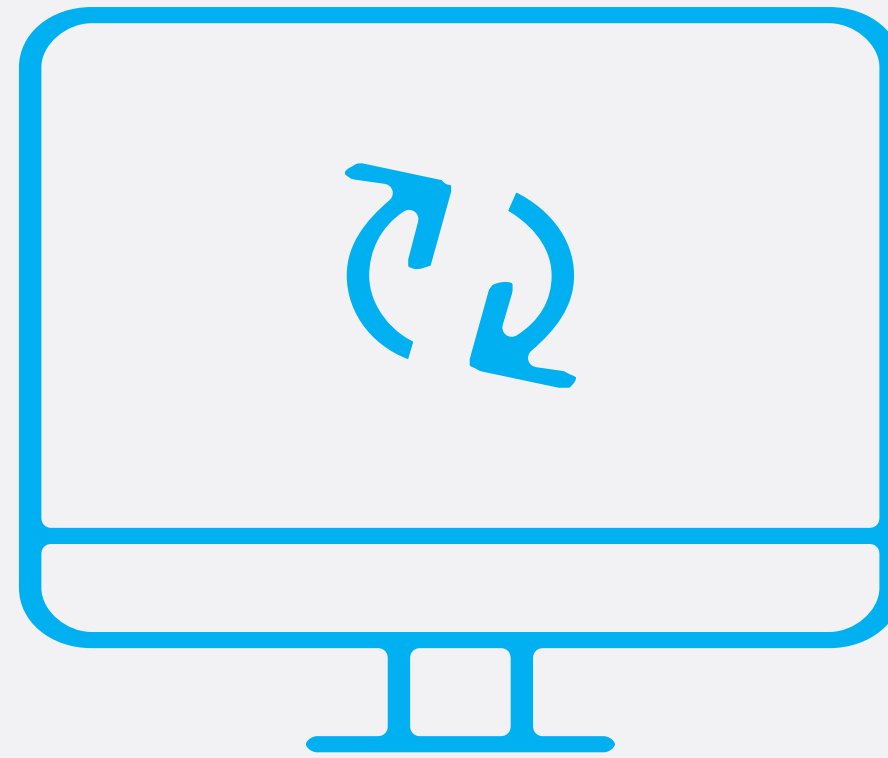
level 4 – mature and automated

Static Code Analysis at Check-in time – Runtime and Dynamic Analysis – APIs must support multi-scope tokens – Bug Bounty Program Coverage – Code Signing – Continuous External App Scanning – Field-level Authenticated Encryption – Integrated Automated Security – Testing with QA Process – Device Fingerprinting – Test Key/Credential Rotation

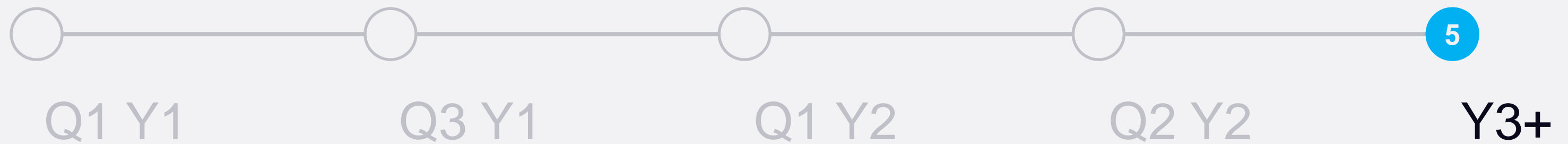


maturity model





















level 5 – optimizing



- | Usage of App Containers
- | Automated OSS Coverage
- | Built-In Honeypot / Indicators
- | HSM and Device Fingerprinting
- | Behavioral Anomaly Detection



sample scorecard

security control	initial	defined	mature	optimizing
HTTPS by default				
Strong Session Management				
Multi-Factor Authentication				
Bug Bounty Program				
Credential Rotation				



the last 0day is in captivity – the galaxy is at peace

thank



you !

* you guys were great

contact

angelpm@gmail.com

-  PradoAngelo
-  LinkedIn.com/in/angeloprado

Check out my **SSL Research**:

 [BreachAttack.com](https://breachattack.com)

333
ANGELO'S ALLEY
SSC C&C

NOT A
THROUGH
STREET

