# Secure defaults
## developer-friendly security

Pieter De Cremer & Claudio Merloni

Who here has heard of secure defaults?

Who is already sold on this idea?

# Secure defaults is NOT just...

# Secure defaults is NOT just...

...having developers fix all security bugs

# Secure defaults is NOT just...

...having developers fix all security bugs

...only fixing high priority issues

# Secure defaults

make it easy to write secure code
make it hard to write insecure code

# Dweilen met de kraan open

## English: mopping while the tap is still running

# Security researchers at Semgrep


Pieter De Cremer
0xDC0DE

Claudio Merloni
p4p3r


# Early adopters are doing this already

Netflix

https://www.youtube.com/watch?v=HIdexRqjpWc

Meta / Facebook

https://about.fb.com/news/2019/01/designing-security-for-billions/

Microsoft

https://www.acsac.org/2007/workshop/Howard.pdf

Google

https://sre.google/books/building-secure-reliable-systems/

Snowflake

https://semgrep.dev/blog/2021/appsec-development-keeping-it-all-together-at-scale

Semgrep

https://semgrep.dev/blog/2020/fixing-leaky-logs-how-to-find-a-bug-and-ensure-it-never-returns

And many more

# Secure defaults

| | |
|---|---|
| WHY | Security must scale |
| WHAT | The secure way, the easy way |
| WHO | Success stories |
| HOW | Think long term, high impact |

# Secure defaults

| | |
|---|---|
| WHY | Security must scale |
| WHAT | The secure way, the easy way |
| WHO | Success stories |
| HOW | Think long term, high impact |

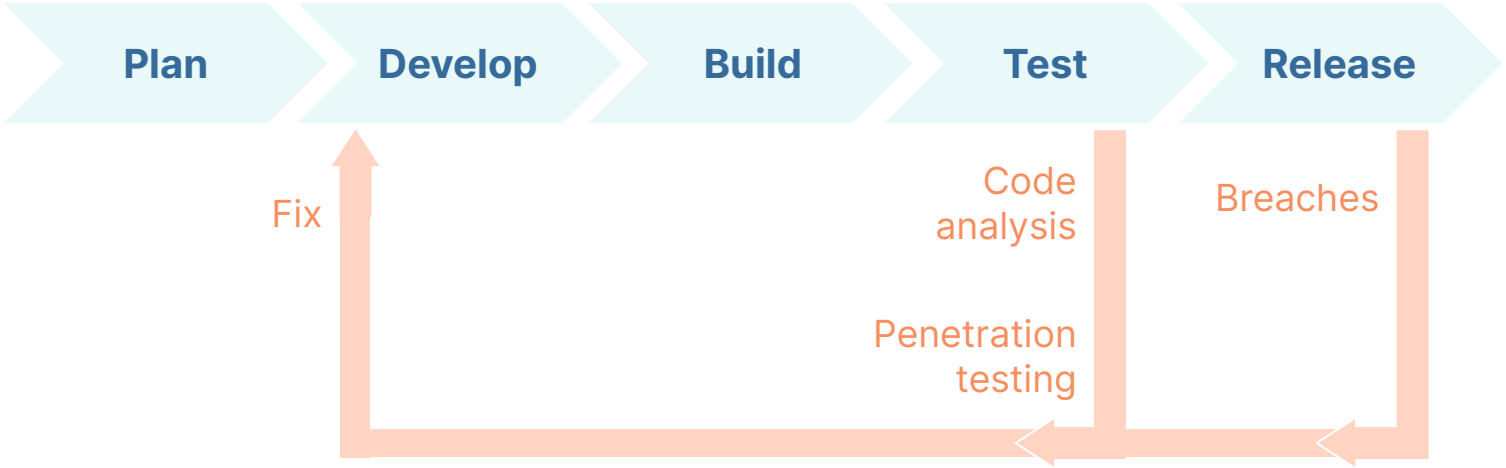
# Despite security automations, vulnerabilities are still prevalent

**Every application**
suffers from security issues throughout its lifetime

**Underlying code**
is where vulnerabilities lie, in most cases

**Nothing new under the sun**
as vulnerabilities evolve just slowly

# Traditional security tools were designed to be part of software testing



# The development team and security team historically had an adversarial relationship

## Two separate worlds

with different priorities and perspectives

## Not working in tandem

instead pushing around large lists of potential issues

## No empathy, little collaboration

caught up in a system that doesn't scale

AppSecCali 2019 - A Pragmatic Approach for Internal Security Partnerships - Netflix

# Modern development practices require security teams to adapt

## Automation is a key element

to increase speed of finding and fixing vulnerabilities

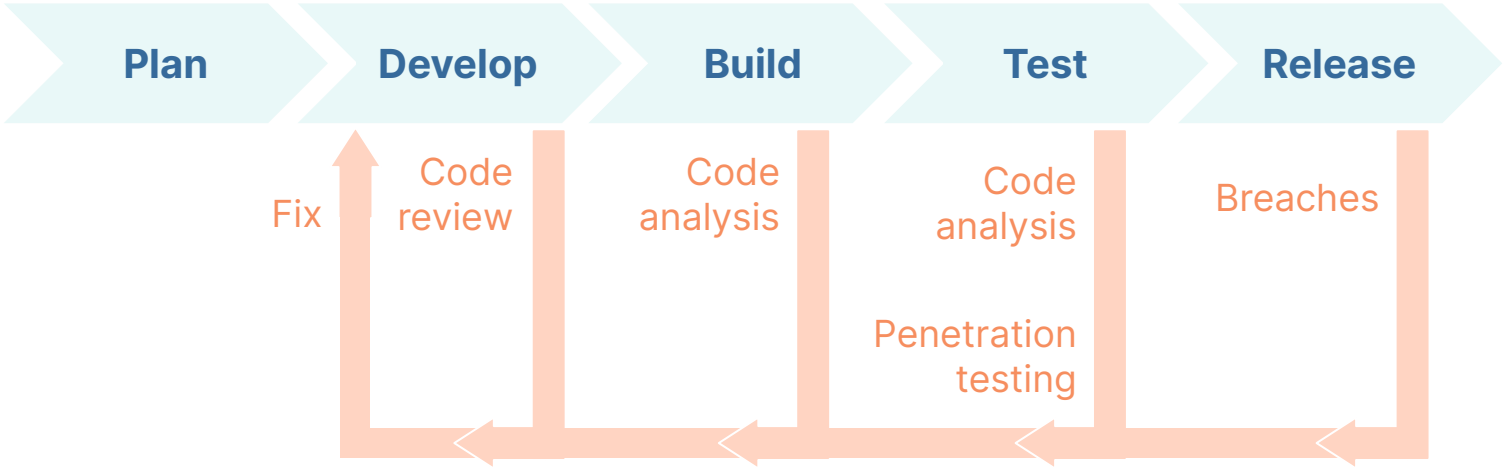
## Context switching kills productivity

and same goes for security outside of dev workflows

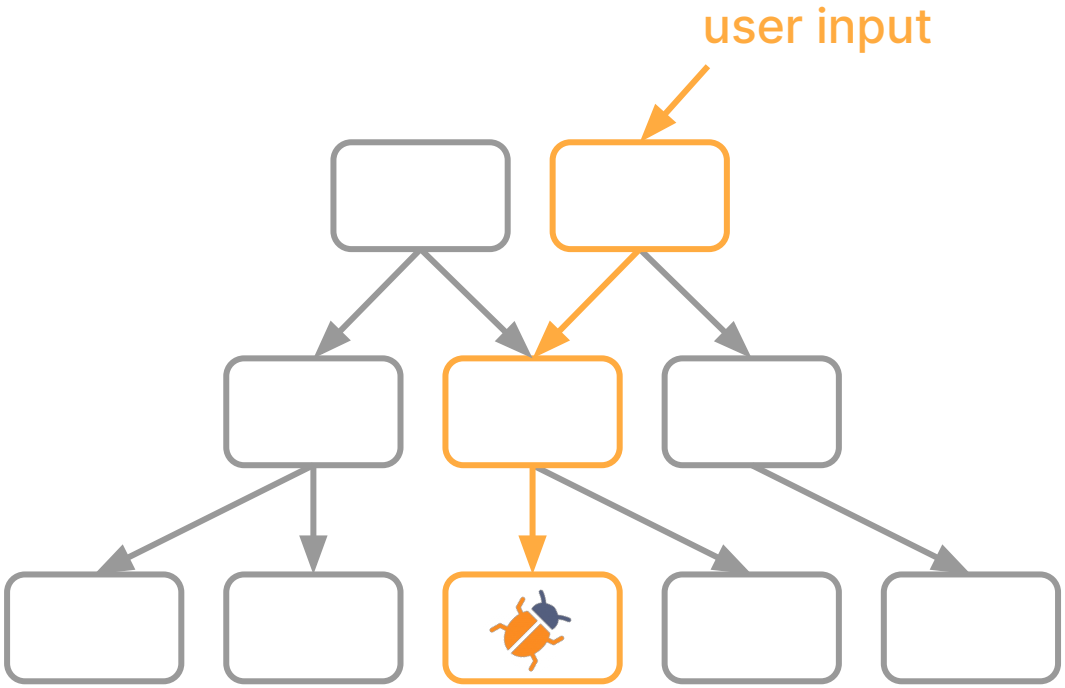
## Testing speed is fundamental

to avoid friction and make security “seamless”

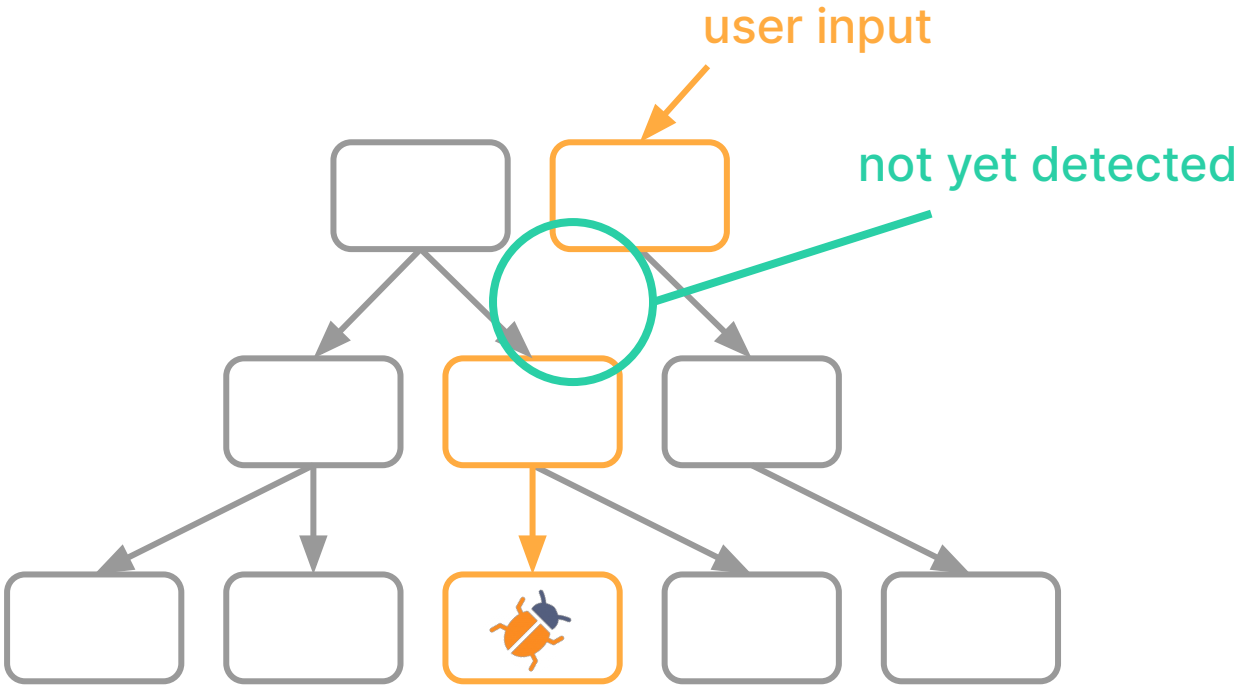
# A shift-left movement is ongoing to address security earlier in development



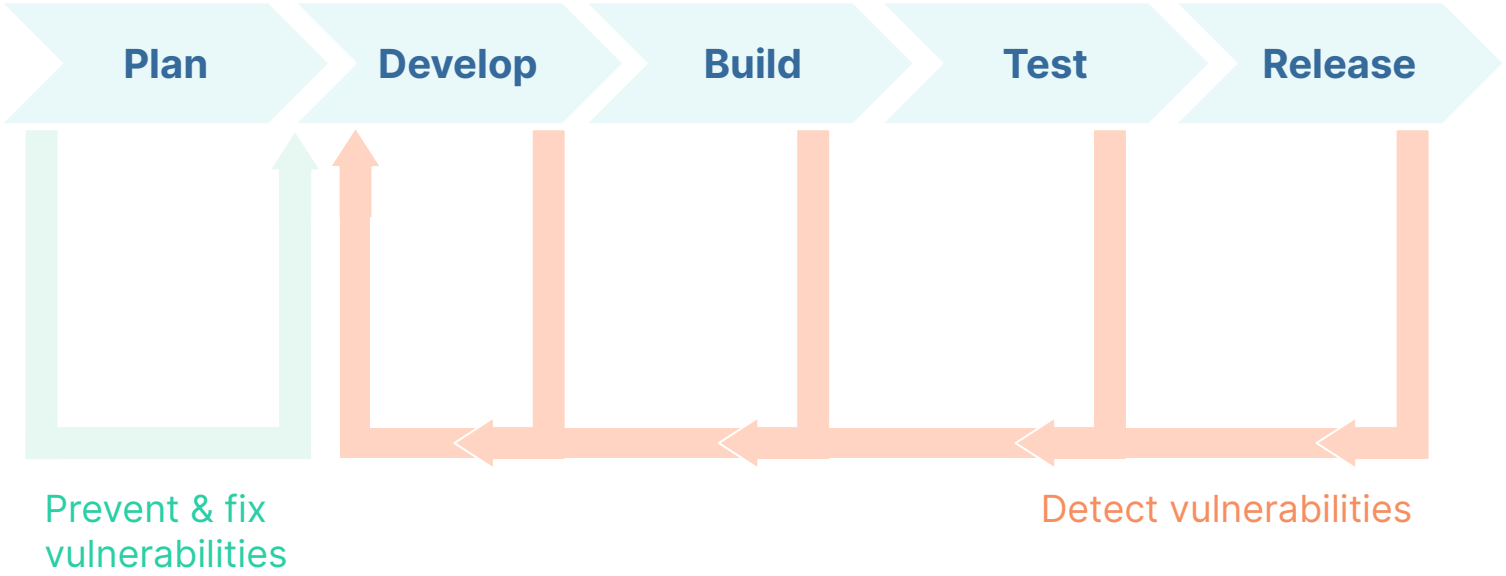
# Traditional security tools use a reactive approach



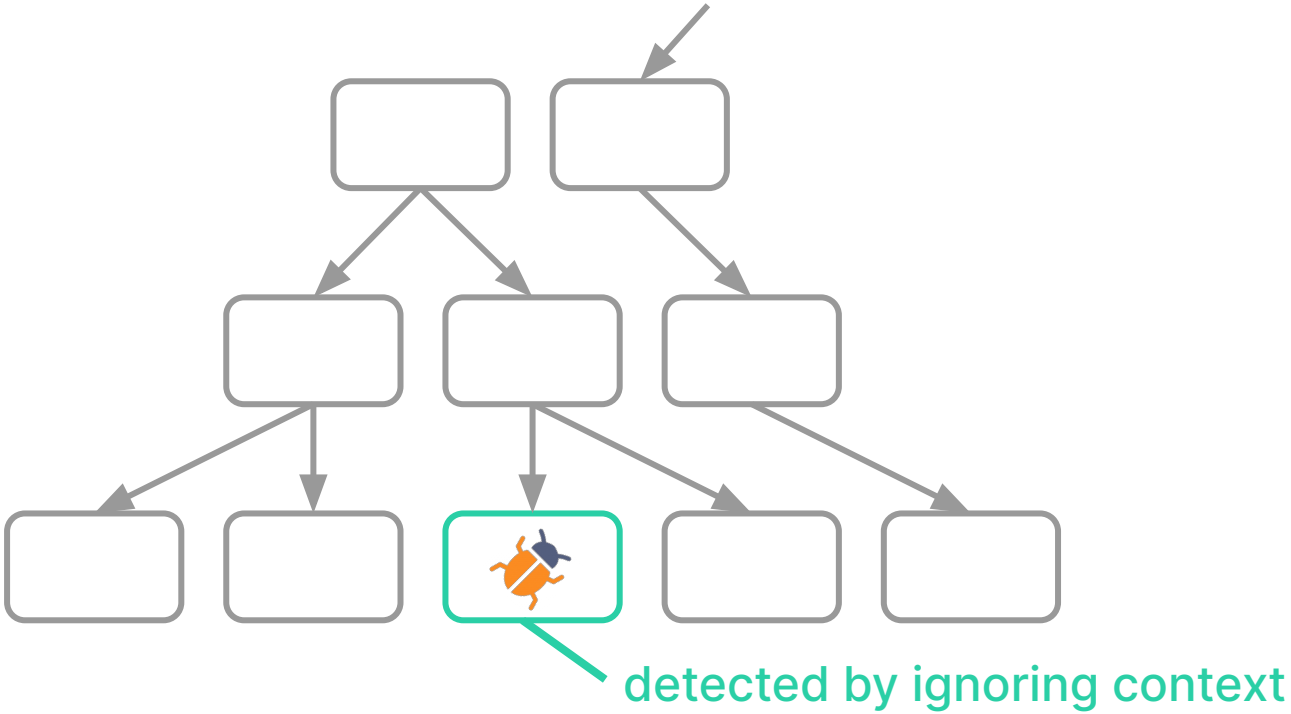
# Traditional security tools use a reactive approach



# Security teams should be enablers



# With secure defaults
# we can be more proactive



# They should provide developers with role-specific tools

**Relevant**
to the developer's work

**Efficient**
in meeting the developer's needs

**Usable**
and well-integrated into the developer's workflow

# Don’t just take our word for it

## Owasp Top 10

*“If we genuinely want to "move left" as an industry, we need more threat modeling, secure design patterns and principles, and reference architectures.”*

## Owasp SAMM

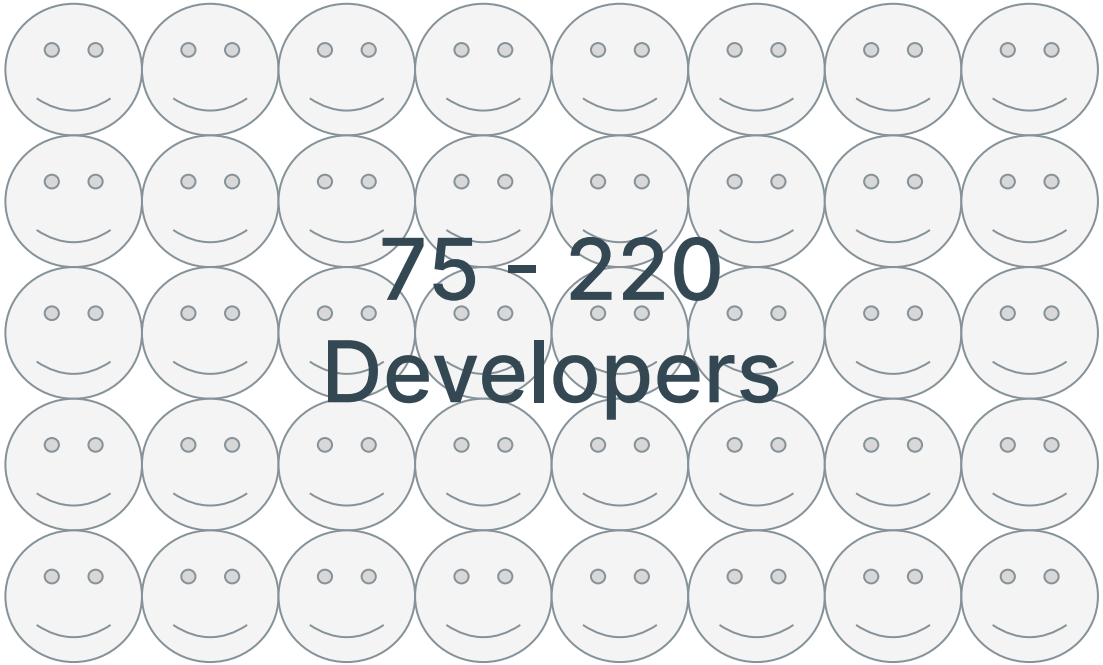
*Security Architecture - Level 2: “Direct the software design process toward known secure services and secure-by-default designs.”*

# Secure defaults

| | |
|---|---|
| WHY | Security must scale |
| WHAT | The secure way, the easy way |
| WHO | Success stories |
| HOW | Think long term, high impact |

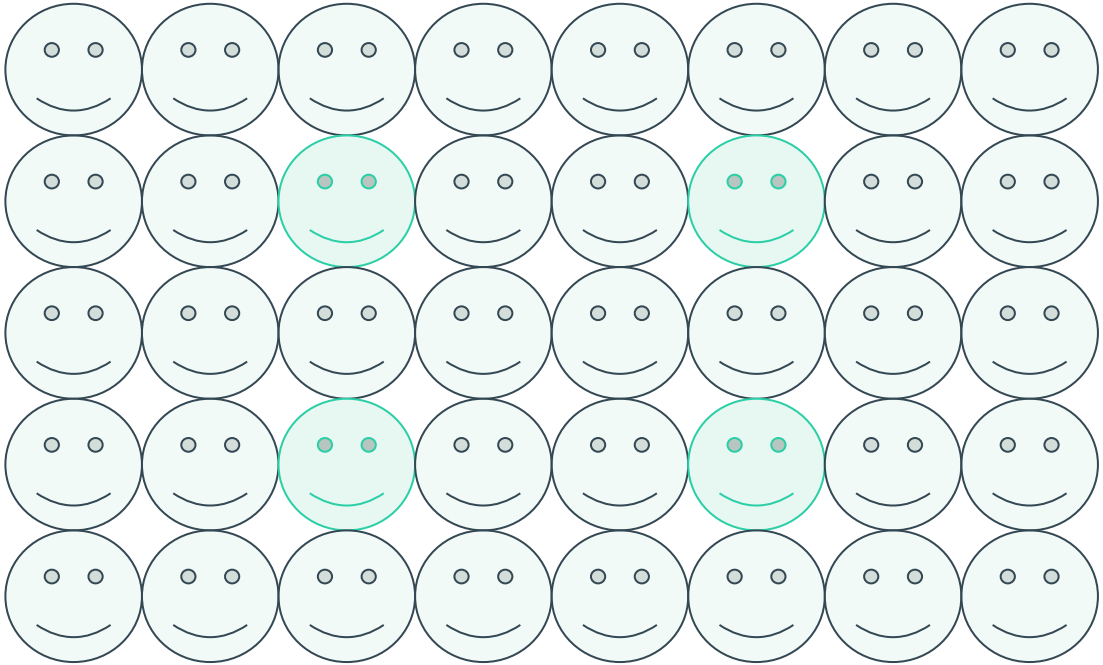
# The security team is responsible for finding vulnerabilities in the software

1
AppSec




APPSEC Cali 2018 - We Come Bearing Gifts: Enabling Product Security

# Security should become a shared responsibility



# Shared responsibility means shared goals

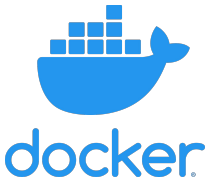
**Ship features fast**
what developers care about

**Prevent and fix vulnerabilities**
what security people care about

Improving one at the detriment of the other
is not real improvement

Security is not special
Plan and scope it with the rest of the work

# To make secure code more scalable, we can learn from the DevOps movement




## Before: Operators responsible

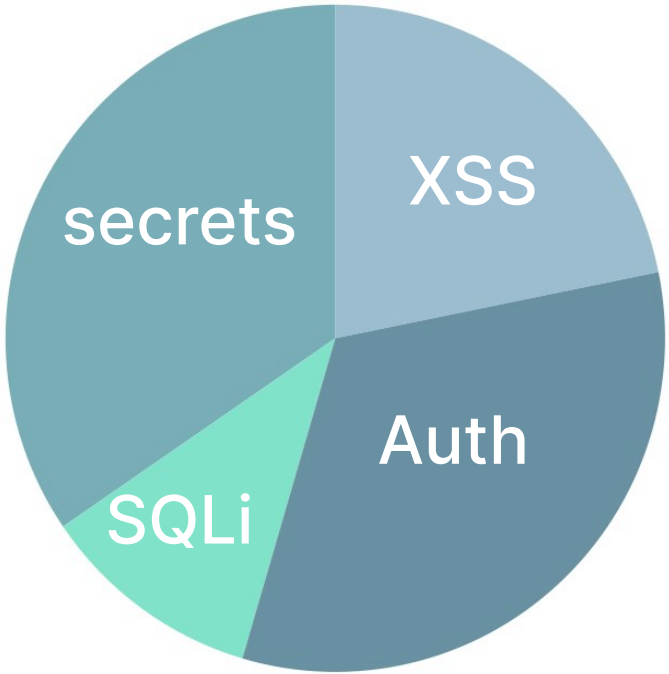
developers throw finished code over the wall

## After: Self-service deployment

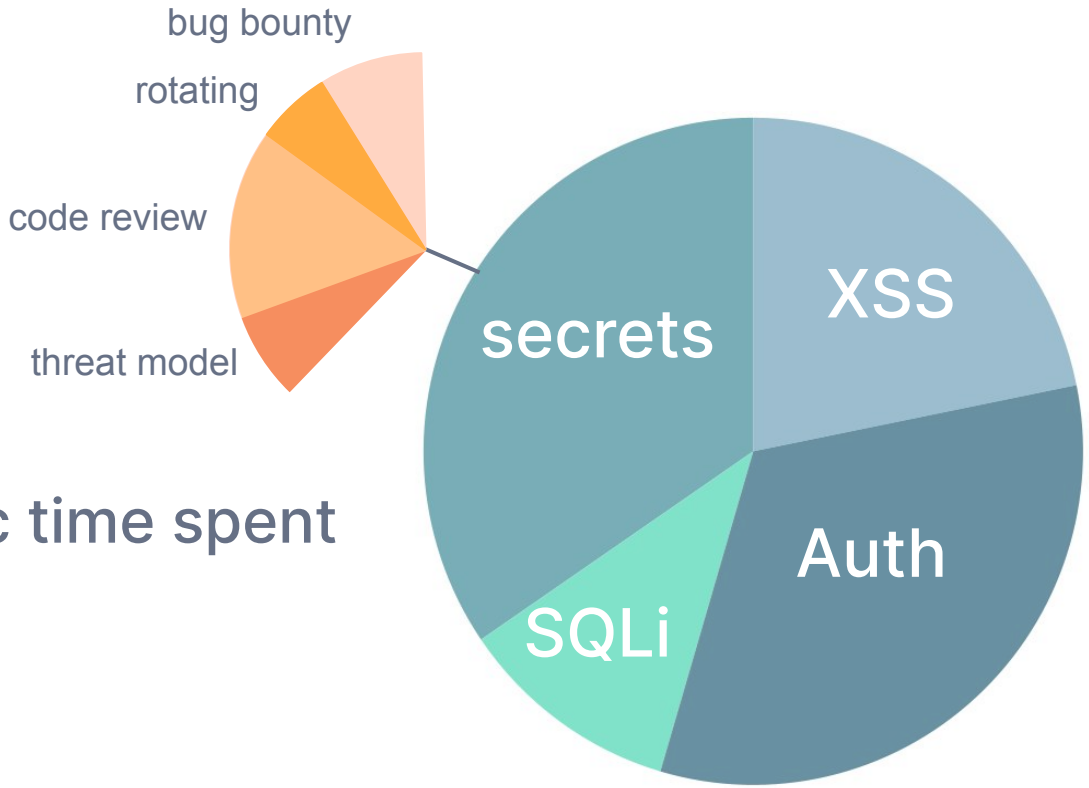
with CICD pipeline and infrastructure as code

# Eliminate bug classes one at a time

AppSec time spent

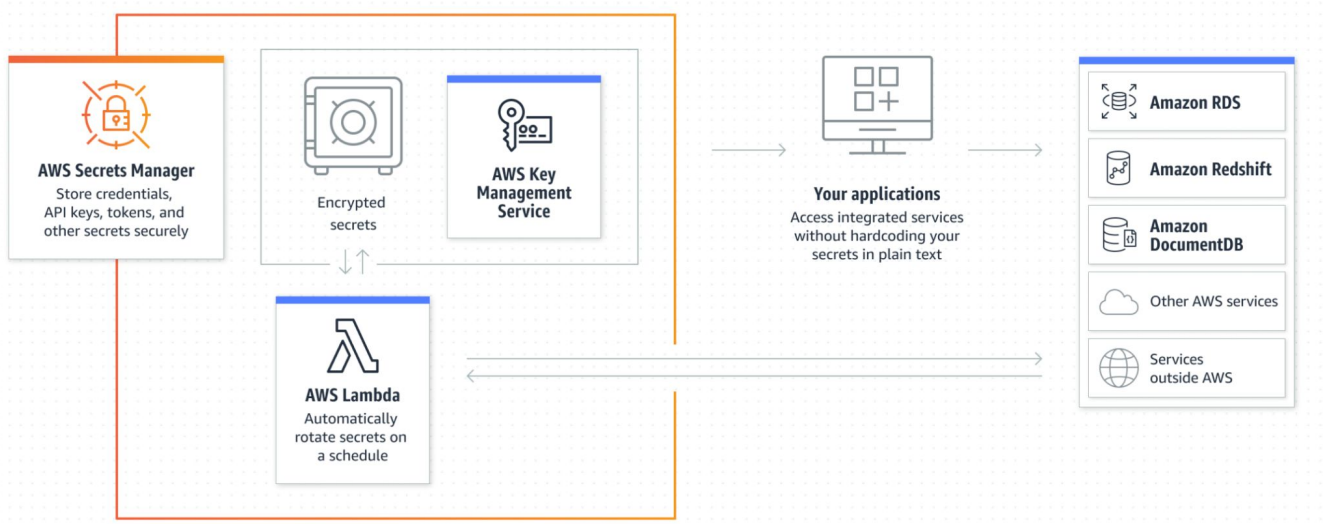


# Eliminate bug classes one at a time




AppSec time spent

# Example 1: secrets must be stored in AWS



# Example 1: secrets must be stored in AWS

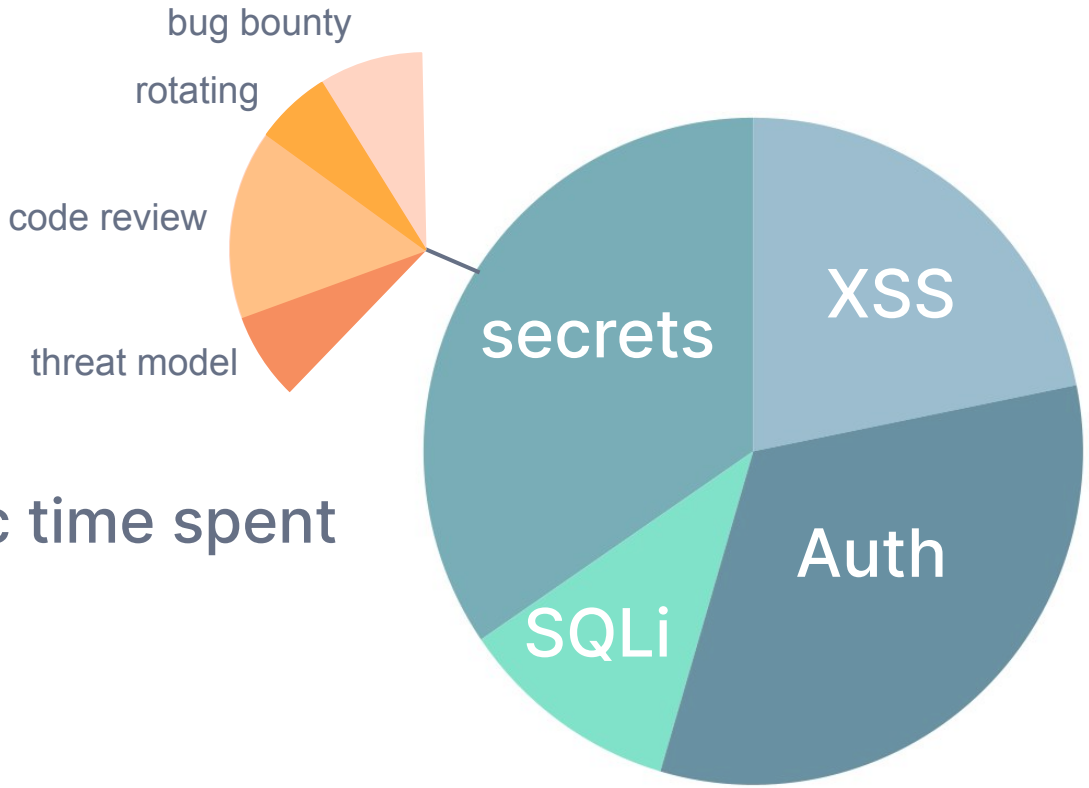
## Python

```
response = client.get_secret_value(
    SecretId='MyTestDatabaseSecret',
)
print(response)
```

## Java

```
private final SecretCache cache  = new SecretCache();

@Override public String handleRequest(String secretId,  Context c) {
final String secret  = cache.getSecretString(secretId);
    System.out.println(secret);
}
```

# Eliminate bug classes one at a time




AppSec time spent

# Killing bug classes leads to compounding effects to leverage your time better

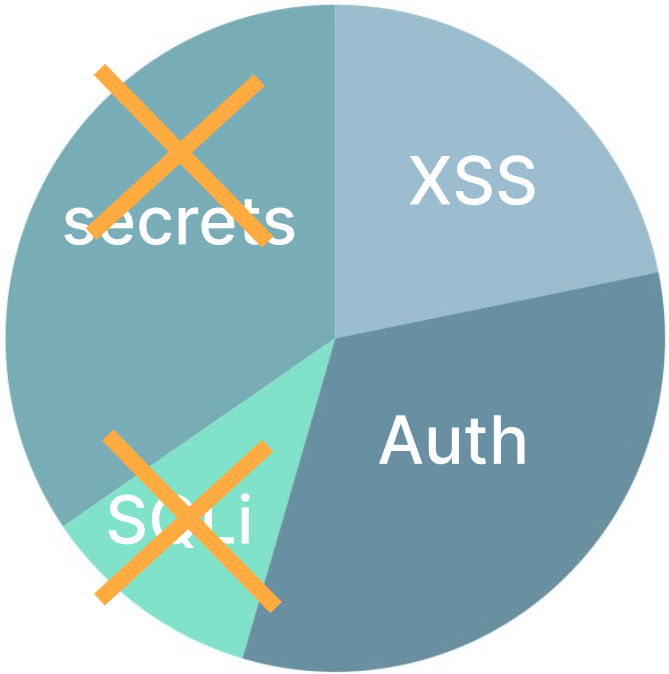
AppSec time spent

# Example 2: queries must be parameterized

```
1  import java.sql.Connection;
2
3  public class WorkshopDemo{
4
5      public ResultSet getBeer(Connection conn, String beerName){
6          String query = "SELECT brand, brewery, aclohol, price FROM beer WHERE name = " + beerName;
7          Statement stmt = conn.createStatement();
8          ResultSet rs = stmt.executeQuery(query);
9          return rs;
10     }
11
12     public ResultSet getBeerSecurely(Connection conn, String beerName){
13         String query = "SELECT brand, brewery, aclohol, price FROM beer WHERE name = ?";
14         PreparedStatement stmt = conn.prepareStatement(query);
15         stmt.setString(beerName);
16         ResultSet rs = conn.executeQuery();
17         return rs;
18     }
19
20 }
```

# Killing bug classes leads to compounding effects to leverage your time better

AppSec time spent



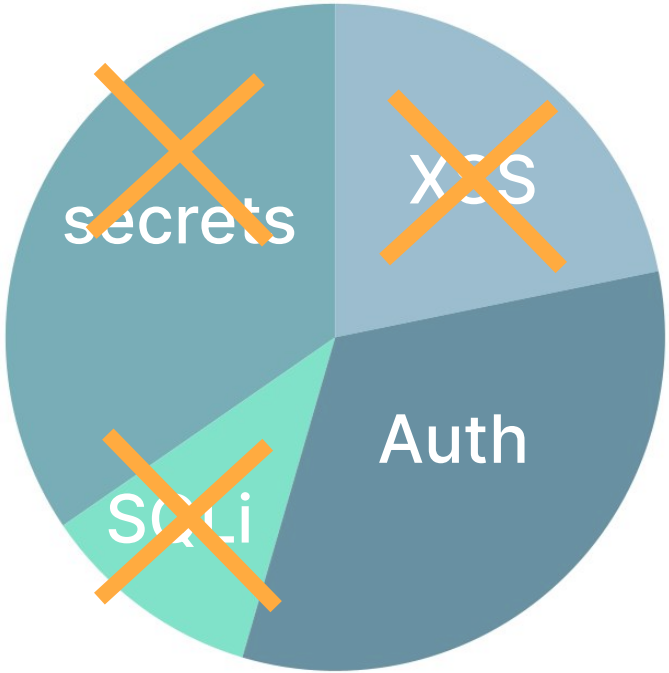
# Example 3: no direct response writer

```
29  @WebServlet(value="/xss-04/BenchmarkTest02229")
30  public class BenchmarkTest02229 extends HttpServlet {
31
32      private static final long serialVersionUID = 1L;
33
34      @Override
35      public void doPost(HttpServletRequest request, HttpServletResponse response)
36              throws ServletException, IOException {
37          response.setContentType("text/html;charset=UTF-8");
38
39          String results = doSomething(request.getParameter(“param”));
40
41          response.setHeader("X-XSS-Protection", "0");
42          response.getWriter().printf(“Results are: %s”, results);
43      }
```

Solution: Use framework like JavaServer Faces (JSF) instead

# Killing bug classes leads to compounding effects to leverage your time better

AppSec time spent



# Secure defaults

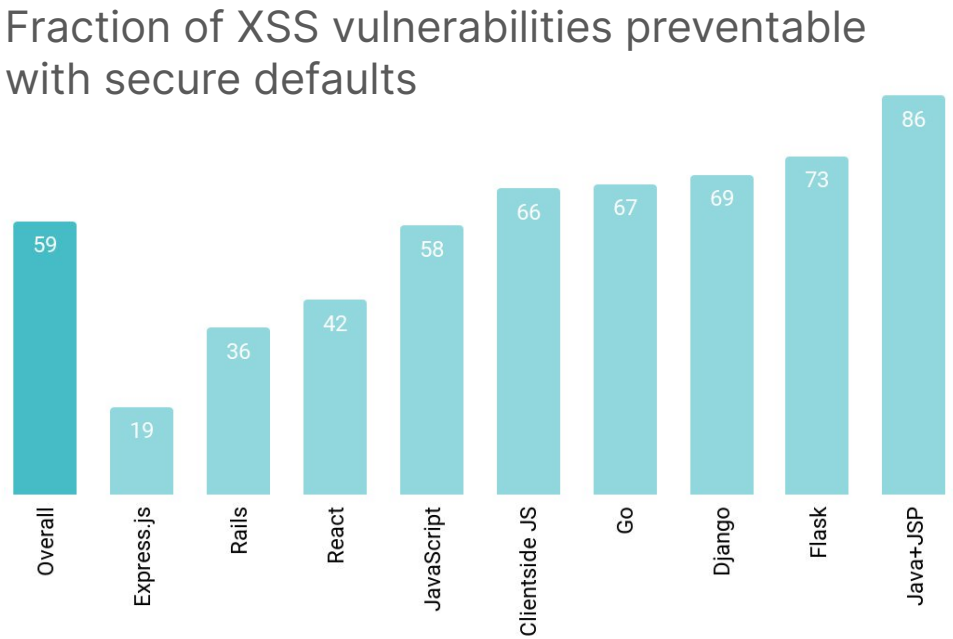
WHY Security must scale

WHAT The secure way, the easy way

WHO Success stories

HOW Think long term, high impact

# 59% of XSS vulnerabilities could have been prevented with secure defaults




[The power of Guardrails, Colleen Dai, Grayson Hardaway, BSides San Francisco]

# What does success look like?

Classes of security risk eliminated

Average time to find and fix reduced

Average severity reduced

Bug bounty costs reduced

# How Netflix does secure defaults

Netflix Culture Meets Product Security | by Bryan D. Payne | Medium
The Paved Road at Netflix
APPSEC Cali 2018 - We Come Bearing Gifts: Enabling Product Security
Scaling Appsec at Netflix. By Astha Singhal
AppSecCali 2019 - A Pragmatic Approach for Internal Security Partnerships
The Show Must Go On: Securing Netflix Studios At Scale
Scaling Appsec at Netflix (Part 2) | by Netflix Technology Blog

# How Netflix does secure defaults

## In-house consulting

no long-term relationships, no clear priorities

## Per-app assessment does not scale

actionable self-service is important

Scaling Appsec at Netflix. By Astha Singhal

# How Netflix does secure defaults

## Context, not control

not required, recommended

## Partnerships

invest in paved road together with the consuming team

[APPSEC Cali 2018 - We Come Bearing Gifts: Enabling Product Security with Culture and Cloud]

# How Netflix does secure defaults

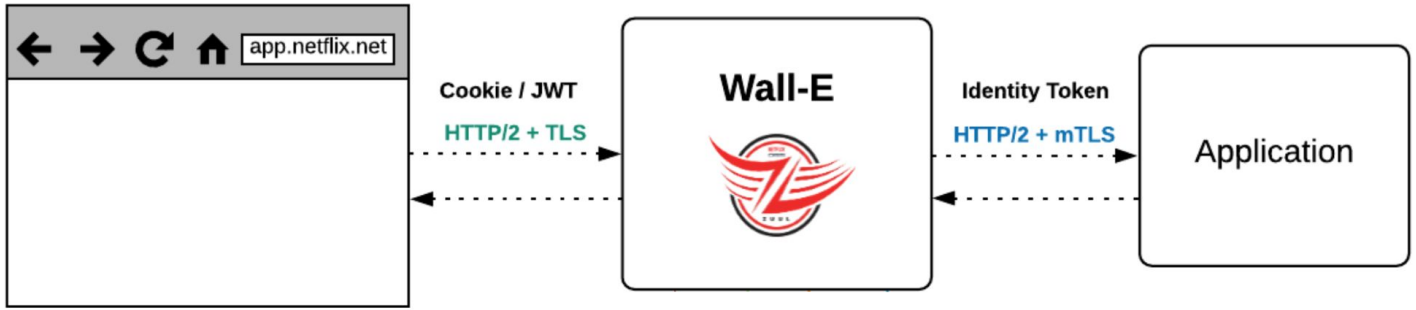
1. Engagement Identification
2. Discovery meeting
3. Security Review
4. Alignment and Document priorities
5. Sync regularly

The Show Must Go On: Securing Netflix Studios At Scale

# How Netflix does secure defaults

## Missing or incomplete authentication

most critical type of issue they regularly faced

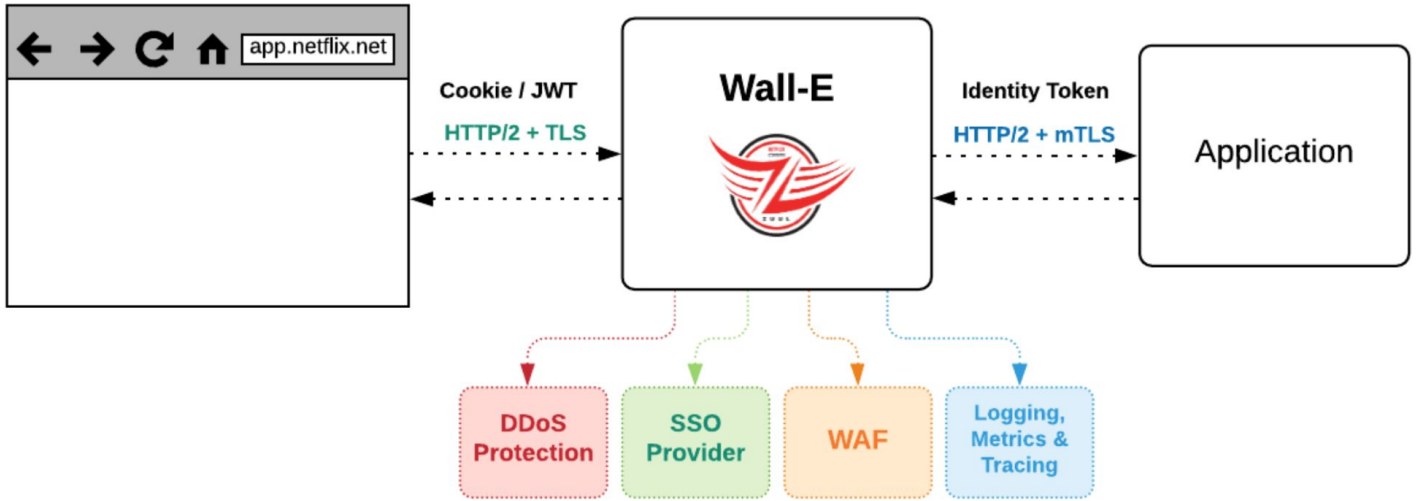



[The Show Must Go On: Securing Netflix Studios At Scale]

# How Netflix does secure defaults

## No organic adoption

until other features were added




[The Show Must Go On: Securing Netflix Studios At Scale]

# How Netflix does secure defaults

## Paved road simplifies reviews

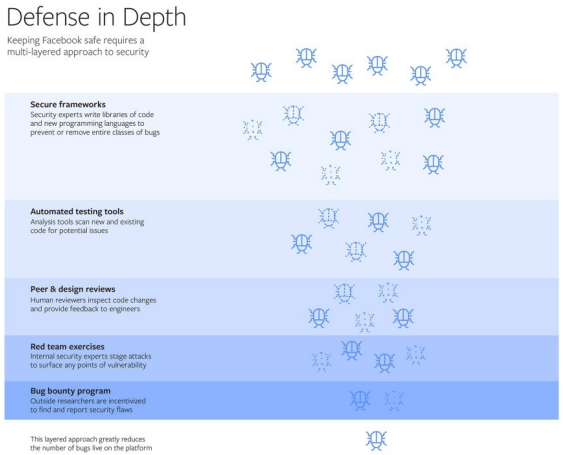
are you using it or not?

## Security was not the main motivation

the secure default allowed developers to move faster

[The Show Must Go On: Securing Netflix Studios At Scale]

# How Meta / Facebook does secure defaults

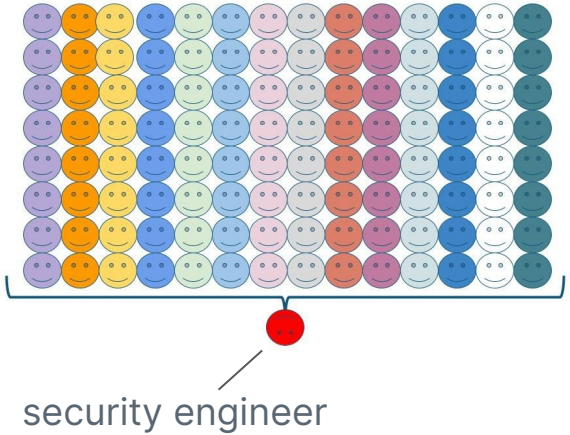



## Defense in Depth

*Secure frameworks* to reduce programming errors
*Automated testing* tools to analyze code non-stop, automatically and at scale

[Designing Security for Billions - Facebook]

# How Snowflake does secure defaults

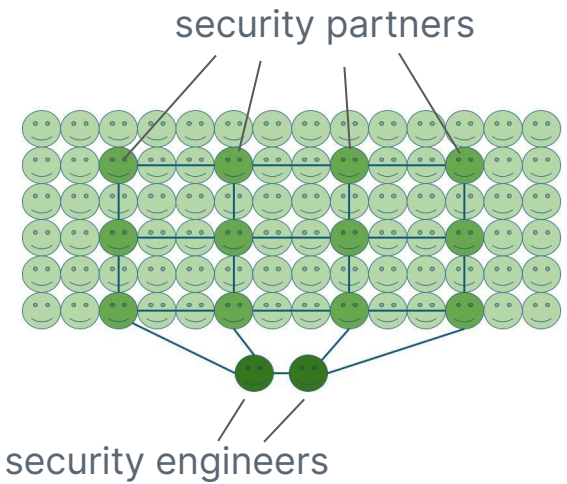



## Unscalable security reviews

performed by security engineers

[Appsec Development: Keeping it all together at scale - Jacob Salassi]

# How Snowflake does secure defaults

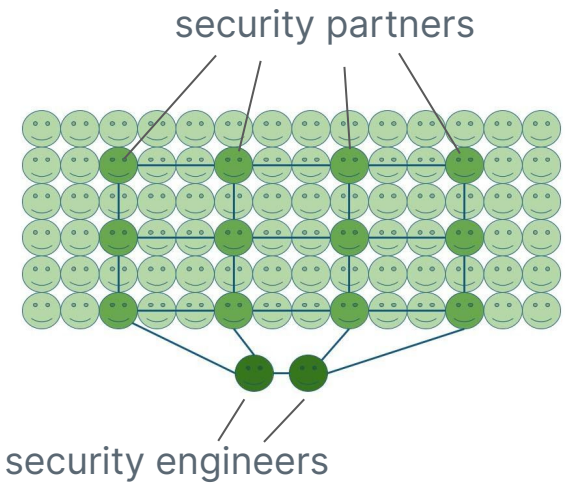



## Self-service threat modeling

by security partners
a big long questionnaire

[Appsec Development: Keeping it all together at scale - Jacob Salassi]

# How Snowflake does secure defaults




## Self-service threat modeling

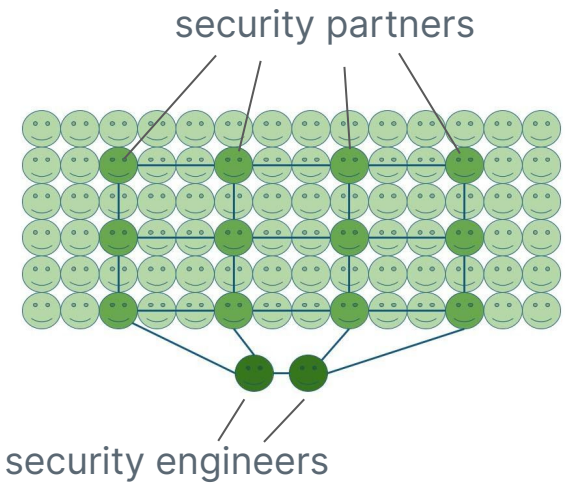
by security partners

## Risk assessment

to determine if threat modeling can be skipped
6 questions to determine if it is Low, Med, or High risk

Appsec Development: Keeping it all together at scale - Jacob Salassi

# How Snowflake does secure defaults




## Self-service threat modeling

by security partners

## Risk assessment

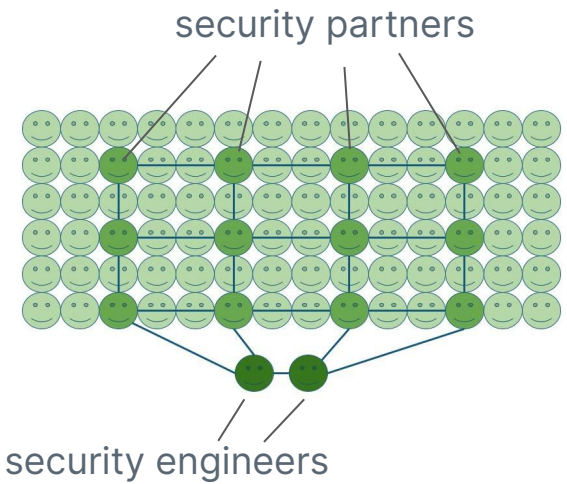
to determine if threat modeling can be skipped

## Security impact assessment

to filter “no security impact” work

Appsec Development: Keeping it all together at scale - Jacob Salassi

# How Snowflake does secure defaults




## Self-service threat modeling
by security partners

## Risk assessment
to determine if threat modeling can be skipped

## Security impact assessment
to filter “no security impact” work

## Project risk impact assessment
to manage timeline risk

Appsec Development: Keeping it all together at scale - Jacob Salassi

# How Snowflake does secure defaults

## Never threat model the same thing twice

create re-usable secure defaults

## Speed up reviews

block anti-patterns with Semgrep

Appsec Development: Keeping it all together at scale - Jacob Salassi

# How Semgrep does secure defaults

**Self-service DevSec**
without security team

**Faster resolution**
solved in minutes

**Security can focus on high-impact work**
not fixing devs latest XSS mistake

Fixing leaky logs: how to find a bug and ensure it never returns - Nathan Brahms

# How Semgrep does secure defaults

## Found tokens being logged

1. **Mitigate**
   Revert logging change
2. **The secure default**
   Replace `str` param with `ObfuscatedStr`
3. **Enforcement**

[Fixing leaky logs: how to find a bug and ensure it never returns - Nathan Brahms]

# How Semgrep does secure defaults

## 3. Enforcement

Block commits to SQLAlchemy models for security review

Yearly training on the pitfalls of logging sensitive data

Audit logs weekly

File an issue with your SAST provider, demanding they add checks to catch sensitively logged data!

[Fixing leaky logs: how to find a bug and ensure it never returns - Nathan Brahms]

# How Semgrep does secure defaults

## 3. Enforcement

~~Block commits to SQLAlchemy models for security review~~

~~Yearly training on the pitfalls of logging sensitive data~~

~~Audit logs weekly~~

~~File an issue with your SAST provider, demanding they add checks to catch sensitively logged data!~~

[Fixing leaky logs: how to find a bug and ensure it never returns - Nathan Brahms]

# How Semgrep does secure defaults

## 3. Enforcement

Enforce an invariant with Semgrep

```
rules:
- id: obfuscate-sensitive-string-columns
  patterns:
    - pattern: |
        $COLUMN = db.Column(db.String, ...)
    - metavariable-regex:
        metavariable: $COLUMN
        regex: '.*(?<![A-Za-z])(token|key|email|secret)(?![A-RT-Za-rt-z]).*'
  message: |
    '$COLUMN' may expose sensitive information in logs and exceptions. Use
    'db.ObfuscatedString' instead of 'db.String'.
  severity: WARNING
```

[Fixing leaky logs: how to find a bug and ensure it never returns - Nathan Brahms]

# Secure defaults

| | |
|---|---|
| WHY | Security must scale |
| WHAT | The secure way, the easy way |
| WHO | Success stories |
| HOW | Think long term, high impact |

# Think long term,

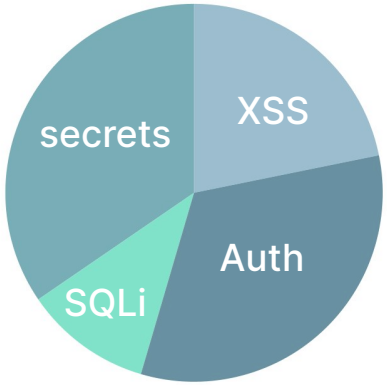
# think high impact

# Think long term, high impact

1. Select vulnerability class
2. Build a scalable solution and make it the default
3. Measure adoption
4. Drive organic adoption

# 1. Select vulnerability class

AppSec time spent




## Focus on best ROI

maximize impact, minimize ongoing time requirements

## Reduce risk, ensure a baseline

don’t try to find and fix every bug

## Eliminate bug classes

find and prevent at scale for compound effect

BSidesSF 2020 - How to 10X Your Company’s Security (Without a Series D) - Clint Gibler

# 1. Select vulnerability class

AppSec time spent


## Focus on best ROI

maximize impact, minimize ongoing time requirements

## Reduce risk, ensure a baseline

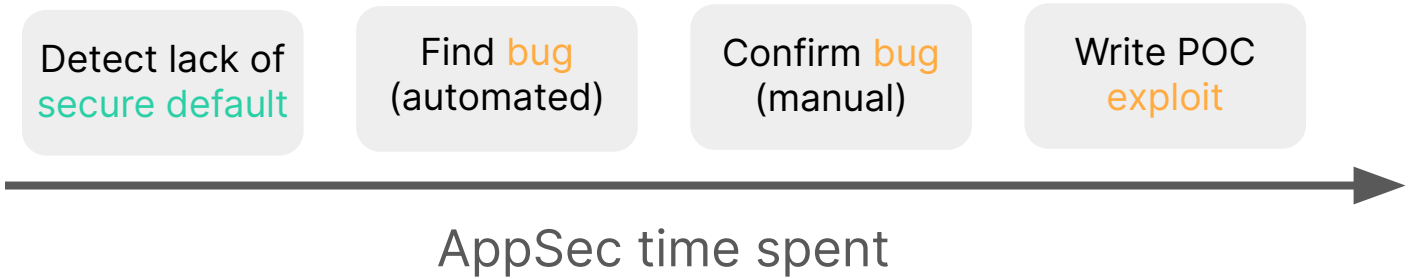
don’t try to find and fix every bug

## Eliminate bug classes

find and prevent at scale for compound effect

BSidesSF 2020 - How to 10X Your Company’s Security (Without a Series D) - Clint Gibler

# 2. Build a scalable solution and make it the default




[Scaling AppSec - sec4dev - Clint Gibler]

# 3. Measure adoption

| Team | Score |
| --- | --- |
| 1 | |
| 2 | |
| 3 | |

## Track costs and fix time

per team and per bug class

## Track adoption of secure defaults

speak to your “customers”

also provides friendly peer pressure

# 4. Drive organic adoption by productizing your secure defaults

## Integrate into existing features

make the secure way, the easy way

## Add non-security features

make it attractive to use

# 4. Drive organic adoption

## Integrate into existing features

make the secure way, the easy way

## Add non-security features

make it attractive to use

## Automate checks

to observe, and to enforce adoption

# An effective false positive is a marking where the developer chooses not to take action

## False positive (FP)

security perspective
secure code marked as insecure

## Effective False Positive (EFP)

developer perspective
any marking a developer won't fix

Tricorder: Building a Program Analysis Ecosystem, Sadowski et. al, Google

# Drive adoption with better tools

## Relevant
project-specific guidelines

## Efficient
fast scan times, well-integrated

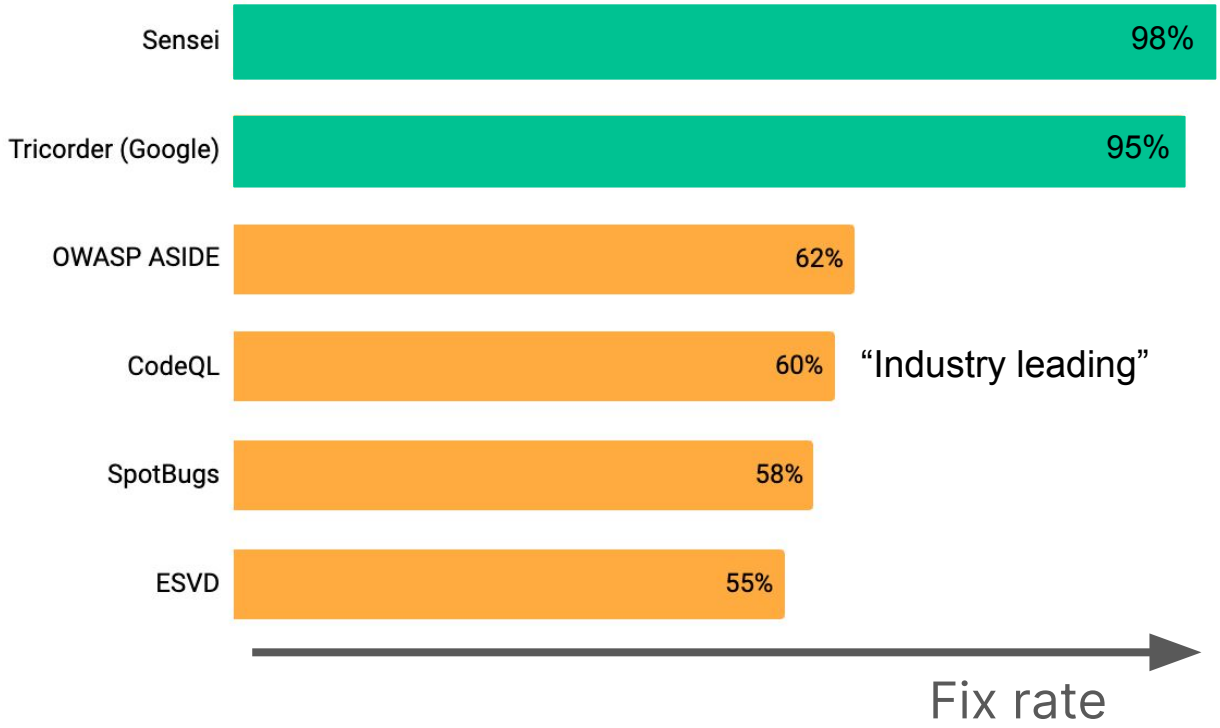
## Usable
not just detect mistakes, but help with fixing

The Paved Path Methodology, Pieter De Cremer, OWASP BeNeLux Days

# A relevant tool allows for customized rules

Customized rules

No customized rules




The Paved Path Methodology, Pieter De Cremer, OWASP BeNeLux Days
Find critical vulnerabilities and eradicate them, forever - CodeQL

# Semgrep allows for easy rule customization

```yaml
rules:
  - id: python-exec
    pattern: exec(...)
    message: Found use of banned function
    severity: WARNING
    languages:
      - python
```

```python
# Use of exec() is completely banned. Find all calls to exec().

import exec as safe_function

# ruleid: python-exec
safe_function(user_input)

# ruleid: python-exec
exec()

# ruleid: python-exec
exec("ls")

# ruleid: python-exec
exec(foo)

# ruleid: python-exec
exec(
    bar
)

# ruleid: python-exec
exec(foo, bar)

# ok: python-exec
some_exec(foo)

# ok: python-exec
# exec(foo)
```

Run ⌘↵

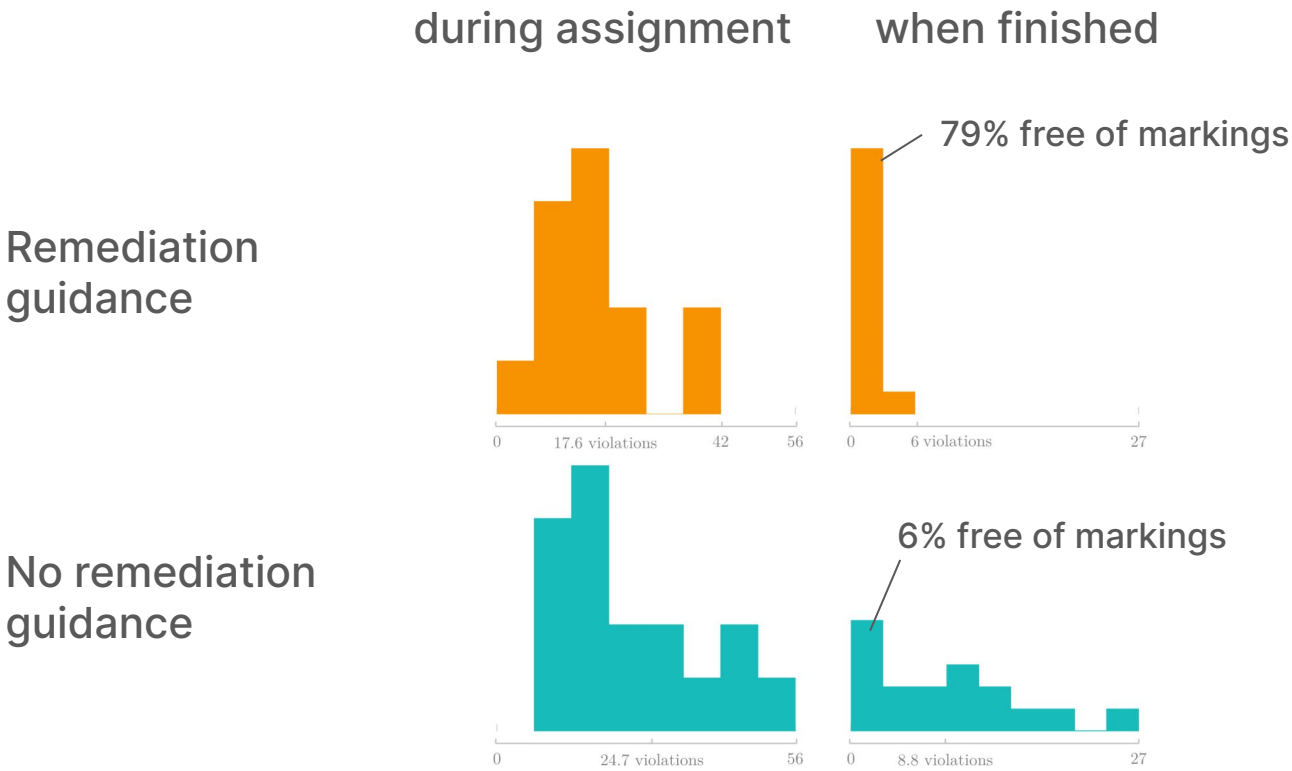
**6 matches** Semgrep v1.25.0 · in 0.4s · tests passed

# A relevant tool uses customized rules

**Customized rules 50% higher fix rate**

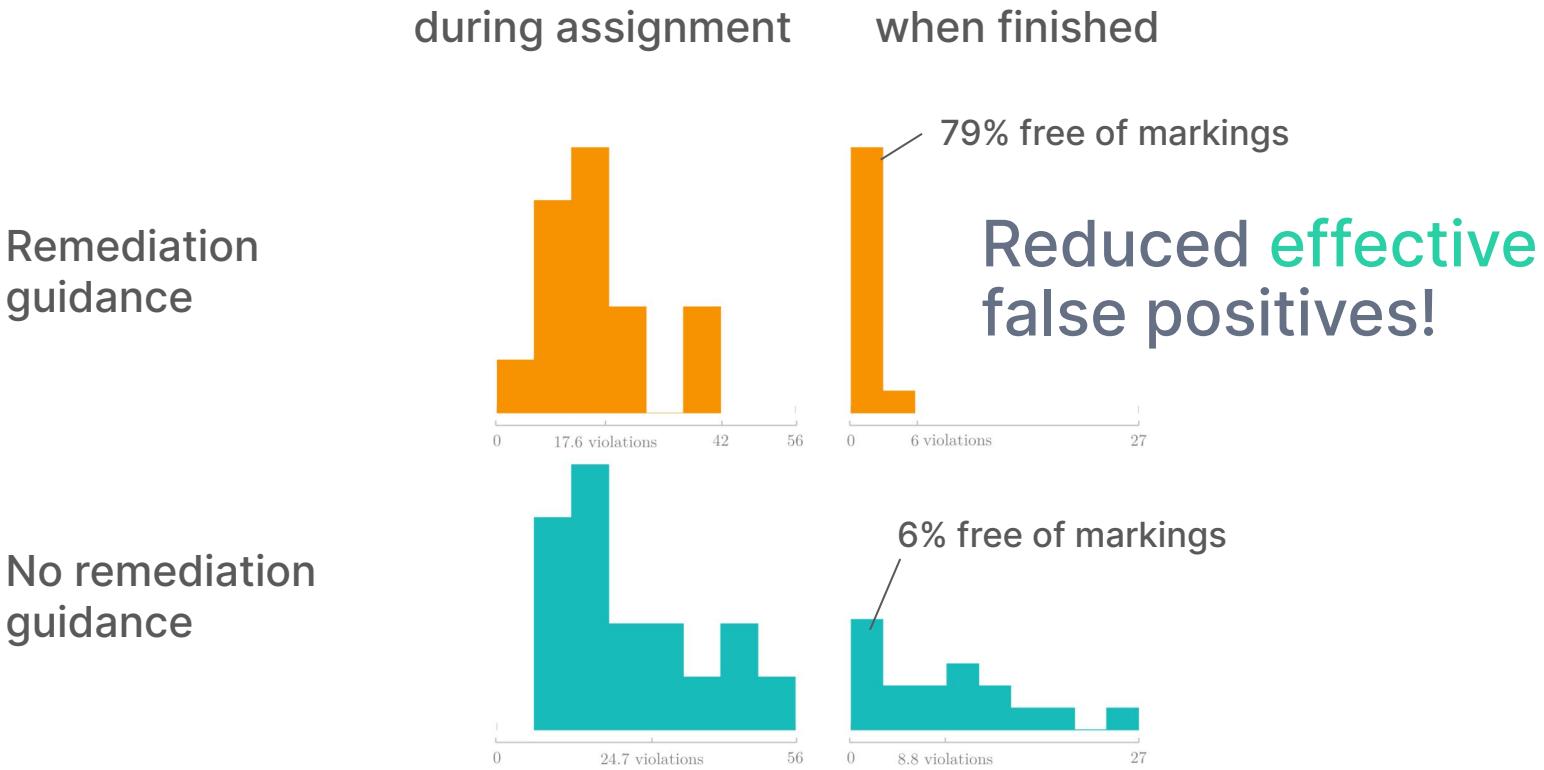
compared to generally applicable rules

# The tool should provide remediation guidance




[The Paved Path Methodology, Pieter De Cremer, OWASP BeNeLux Days]

# The tool should provide remediation guidance




[The Paved Path Methodology, Pieter De Cremer, OWASP BeNeLux Days]

# Semgrep provides remediation guidance in the form of autofixes

```yaml
rules:
  - id: python-exec
    pattern: exec($...CMD)
    fix: safe_exec($...CMD)
    message: Found use of banned function `exec($...CMD)`
    severity: WARNING
    languages:
      - python
```

```python
# Use of exec() is completely banned. Find all calls to exec().

import exec as safe_function

# ruleid: python-exec
safe_function(user_input)

# ruleid: python-exec
exec()

# ruleid: python-exec
exec("ls")

# ruleid: python-exec
exec(foo)
```

Run ⌘↵

**Matches**

Line 6

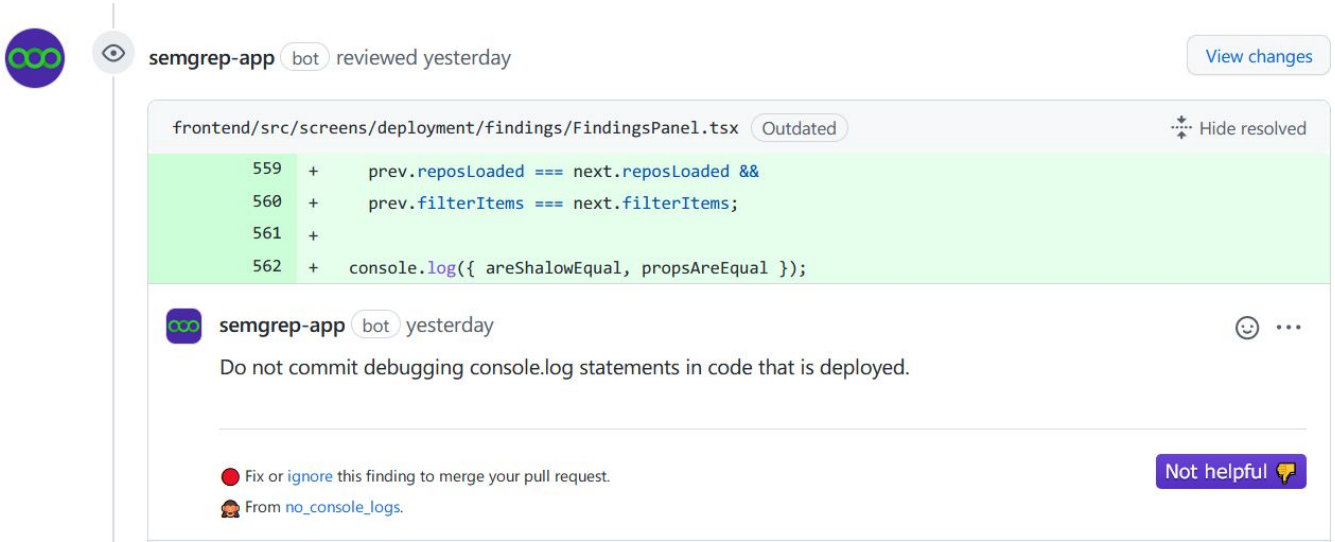
Found use of banned function `exec(user_input)`

autofix `safe_exec(user_input)`

# Semgrep provides remediation guidance in the form of autofixes

```
demo » semgrep --config rule.yaml testcode.py

6 Code Findings

    testcode.py
       python-exec
          Found use of banned function `exec(user_input)`

           ▶▶┆ Autofix ▶ safe_exec(user_input)
            6┆ safe_function(user_input)
             ⋮┆----------------------------------------
       python-exec
          Found use of banned function `exec()`

           ▶▶┆ Autofix ▶ safe_exec()
            9┆ exec()
             ⋮┆----------------------------------------
       python-exec
          Found use of banned function `exec("ls")`

           ▶▶┆ Autofix ▶ safe_exec("ls")
           12┆ exec("ls")
             ⋮┆----------------------------------------
```

# Semgrep provides remediation guidance in the form of autofixes



# Semgrep provides remediation guidance in the form of autofixes

**Rules with autofix have 50% higher fix rate**

compared to rules without autofix

# Struggles and future work

## Political and cultural resistance

the security team wants control

## Code quality takes time away from features

markings are false positives

## Siloed teams and persistent habits

no empathy or synergy, need for building partnerships

# Secure defaults

| | |
|---|---|
| WHY | Security must scale |
| WHAT | The secure way, the easy way |
| WHO | Success stories |
| HOW | Think long term, high impact |

# Secure defaults is NOT just...

...having developers fix all security bugs

...only fixing high priority issues

# Secure defaults is NOT just...

...having developers fix all security bugs
but building scalable self-service solutions

...only fixing high priority issues

# Secure defaults is NOT just...

...having developers fix all security bugs
but building scalable self-service solutions

...only fixing high priority issues
but killing high-impact bug classes

# TL;DR secure defaults

**WHY** Security must scale

speed of development has increased
security experts are understaffed

**WHAT** The secure way, the easy way
systematic fundamental solutions
productizing those solutions

**WHO** Early adopters have been successful
Netflix, Meta, Snowflake, Semgrep, and more

**HOW** Think long term, high impact
leverage your time most effectively now
to have big wins in the future
automate smart

# Secure defaults
## developer-friendly security

Pieter De Cremer & Claudio Merloni