# Eradicate Vulnerability Classes

## With Secure Defaults & Lightweight Enforcement

Adam Berman | r2c.dev

Slides are posted at http://bit.ly/2021Berman-OWASP-Denver

# whois?

**me:**
Adam Berman, lead engineer @ r2c
Formerly: eng lead for Meraki's analytics product, Georgia Tech

**r2c:**
We're an SF based static analysis startup on a mission to profoundly improve software security and reliability.

# Outline

1. **Why Bug-Finding Isn't The Answer**
2. How to Eradicate Vulnerability Classes
3. Tools & Techniques To Make It Real

# Massive Shifts in Tech and Security

| Before | After |
|---|---|
| Waterfall development | Agile development |
| Dev, Ops | DevOps |
| On prem | Cloud |

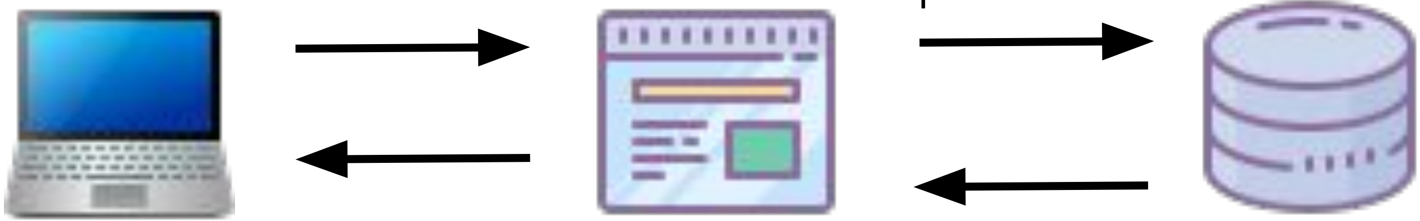
# Massive Shifts in Tech and Security

| | |
|---|---|
| Waterfall development | Agile development |
| Dev, Ops | DevOps |
| On prem | Cloud |
| Finding vulnerabilities | **Secure defaults** |
| ← Before | After → |

# Quiz: Does this app have XSS?

Icons by Icons8

# Quiz: Does this app have XSS?



What does user control?
Structure of data?

Input filtered?

How is it stored?
(field types,
constraints)

DB type?

Context?

- HTML
- HTML attribute
- JavaScript
- ...

Data processed
before sent to
user?

Icons by Icons8

# Quiz: Does this app have XSS?

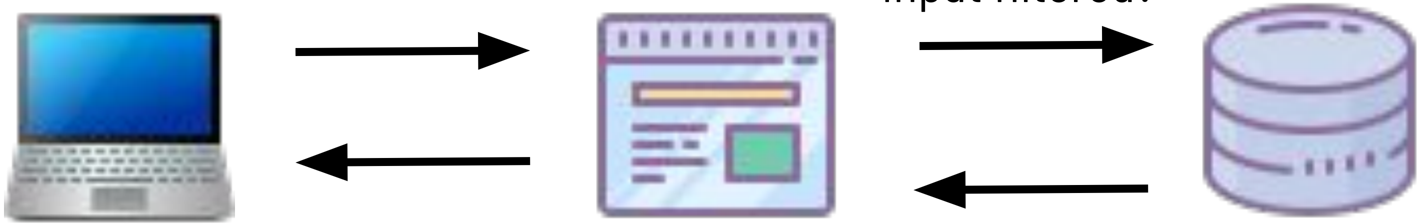
***Guardrail: Frontend is React, banned dangerouslySetInnerHTML***

What does user control?
Structure of data?

Input filtered?

How is it stored?
(field types,
constraints)

DB type?



Context?

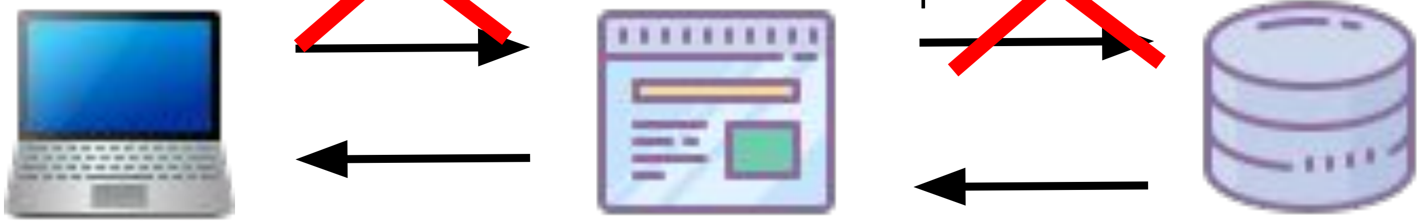
- HTML
- HTML attribute
- JavaScript
- ...

Data processed before sent to user?

Icons by Icons8

# Quiz: Does this app have XSS?

***Guardrail: Frontend is React, banned dangerouslySetInnerHTML***



~~What does user control?~~
~~Structure of data?~~

~~Input filtered?~~

~~How is it stored?~~
~~(field types,~~
~~constraints)~~

~~DB type?~~

Context?

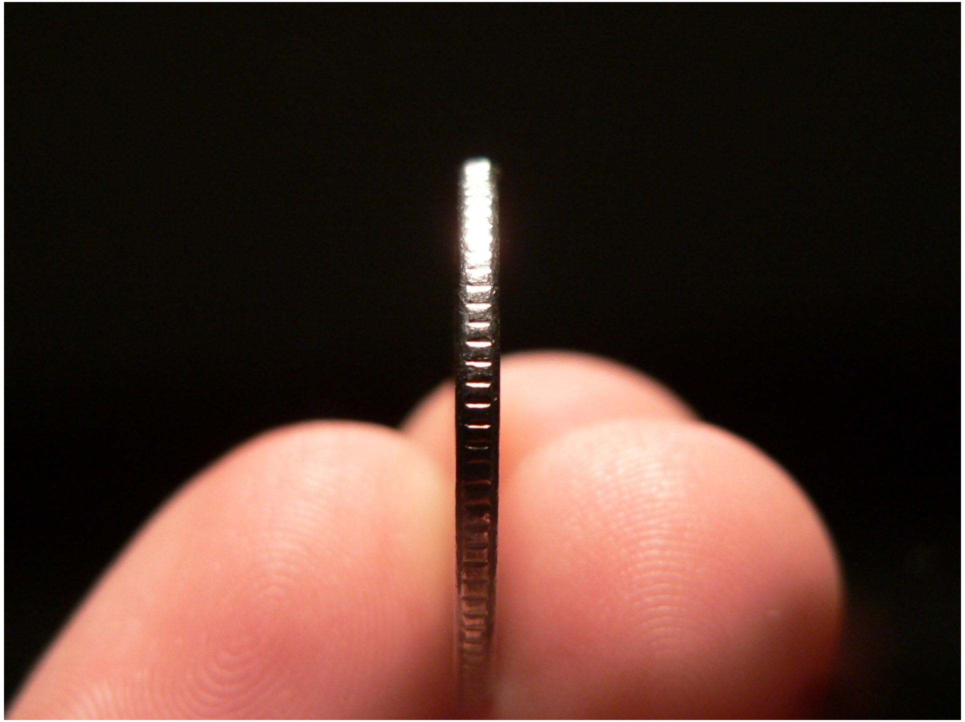
- ~~HTML~~
- ~~HTML attribute~~
- ~~JavaScript~~
- ...

~~Data processed~~
~~before sent to~~
~~user?~~

Icons by Icons8

Finding Bugs

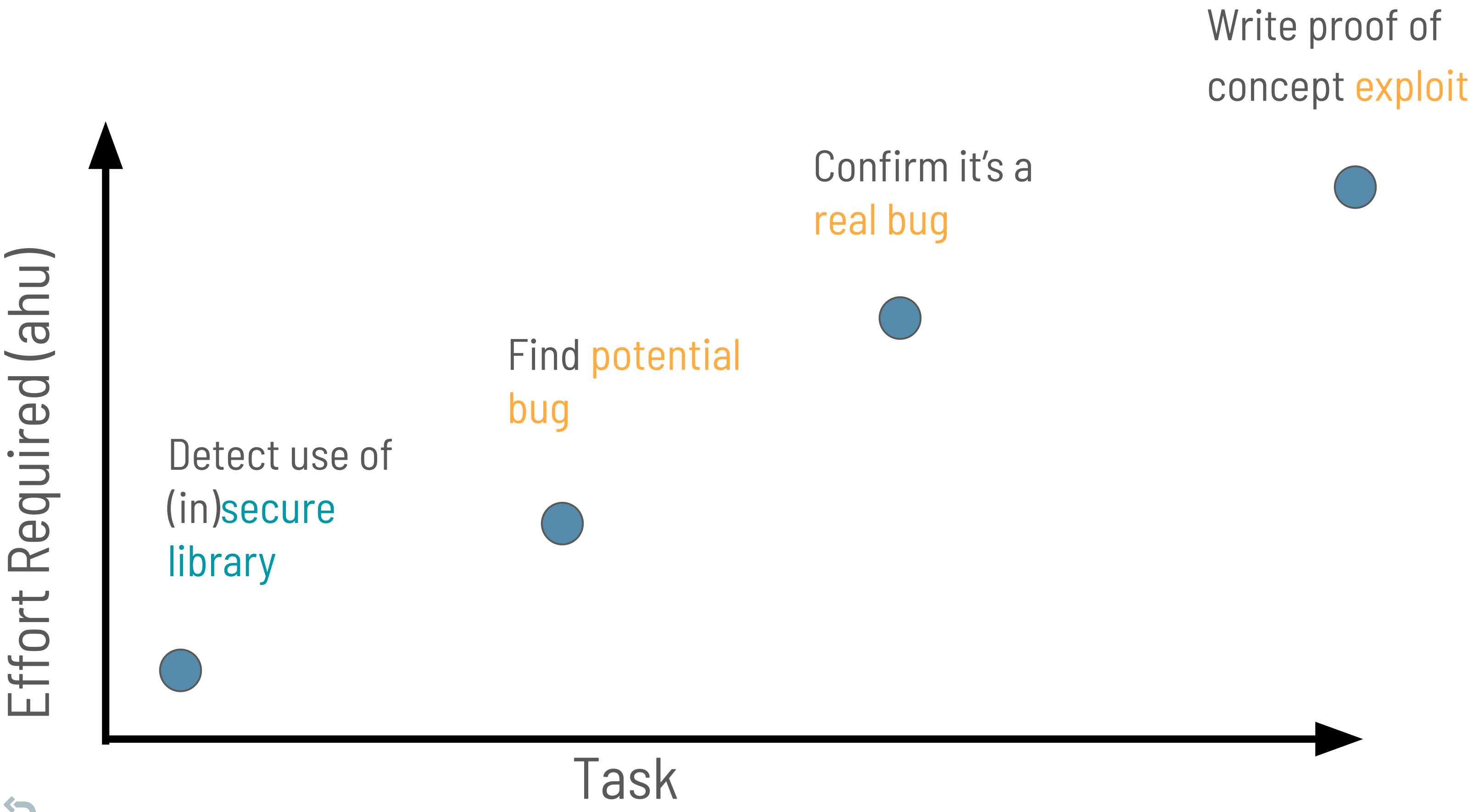
Only using the “safe” way



# Let's Solve the "Easy" Version of the Problem

- This app could have been incredibly complex, with millions of LOC
- With some strong **secure defaults**, we significantly reduced its **risk**
- We did this without fancy tools:
  - DAST that can handle single page apps, GraphQL, modern frontends...
  - SAST tracking attacker input flowing across dozens of files
  - Fuzzing
  - Symbolic execution
  - Formal methods ("proving" correctness)

# Task vs Effort Required



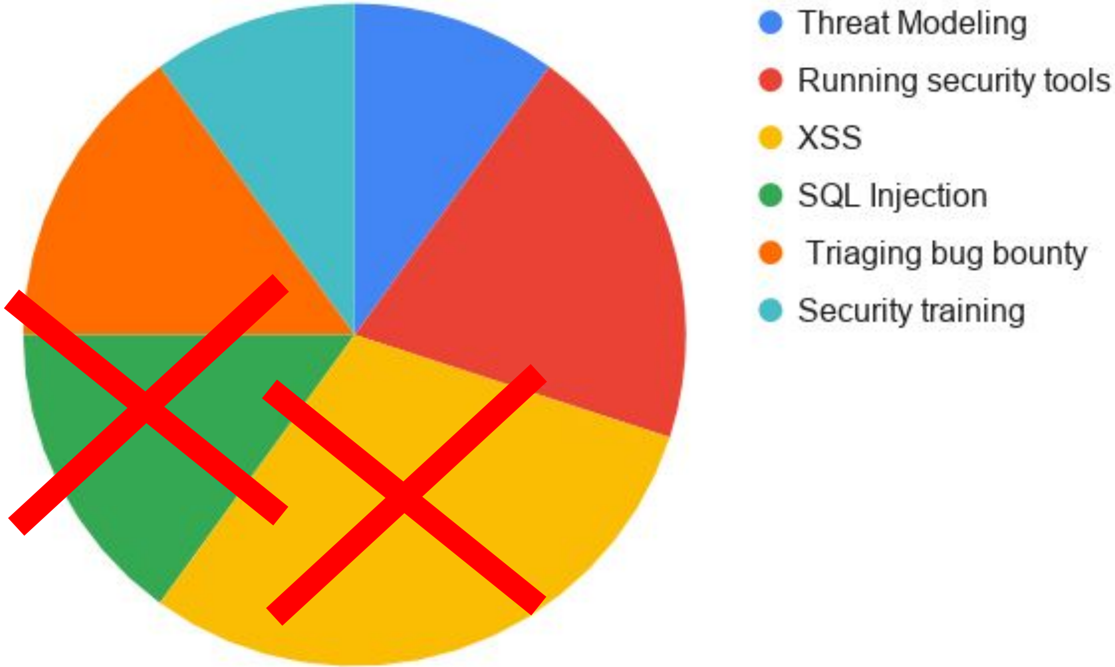
Detecting (lack of) use of

**secure defaults**

is **much easier** than

finding **bugs**

# Compounding Effects of Killing Bug Classes

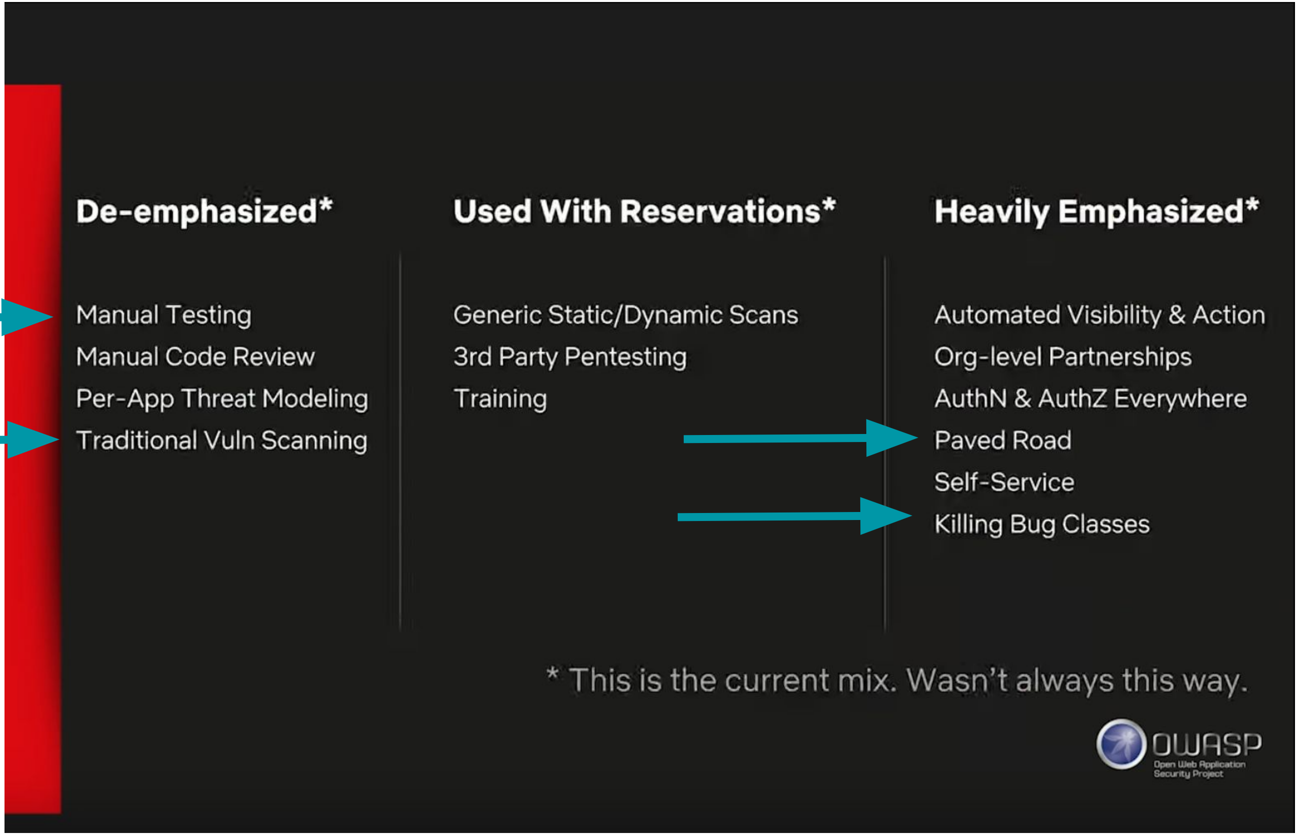


# Your Internal Dialogue?

- "All you've shown me is some hand-wavy diagrams"
- The security industry has focused on bug finding for decades
    - SAST, DAST, pen tests, bug bounty

# We Come Bearing Gifts: Enabling Prod Security w/ Culture & Cloud

AppSec Cali '18, Patrick Thomas, Astha Singhal



**De-emphasized***

- Manual Testing
- Manual Code Review
- Per-App Threat Modeling
- Traditional Vuln Scanning

**Used With Reservations***

- Generic Static/Dynamic Scans
- 3rd Party Pentesting
- Training

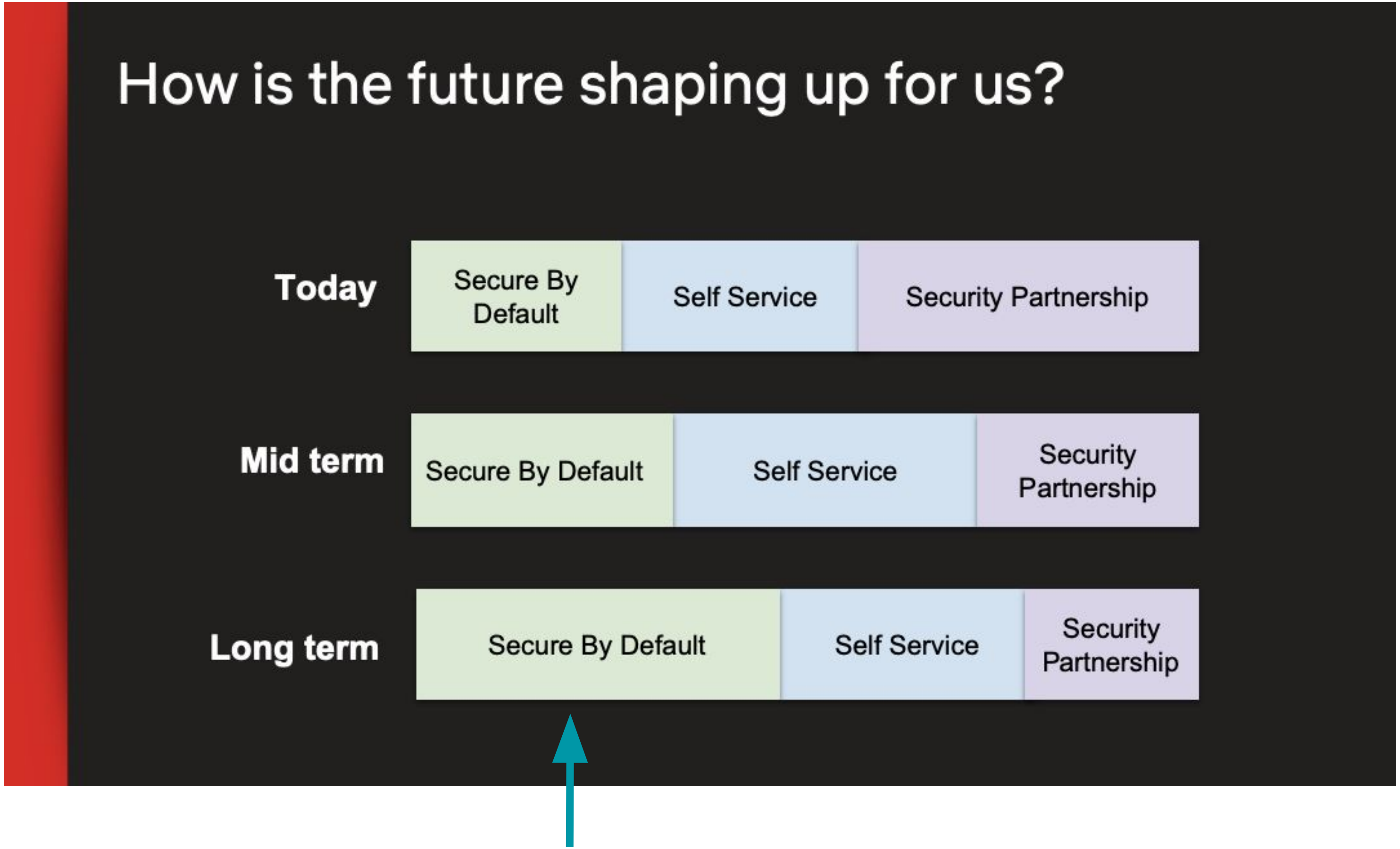
**Heavily Emphasized***

- Automated Visibility & Action
- Org-level Partnerships
- AuthN & AuthZ Everywhere
- Paved Road
- Self-Service
- Killing Bug Classes

* This is the current mix. Wasn't always this way.

# A Pragmatic Approach for Internal Security Partnerships

AppSec Cali ‘19, Scott Behrens, Esha Kanekar



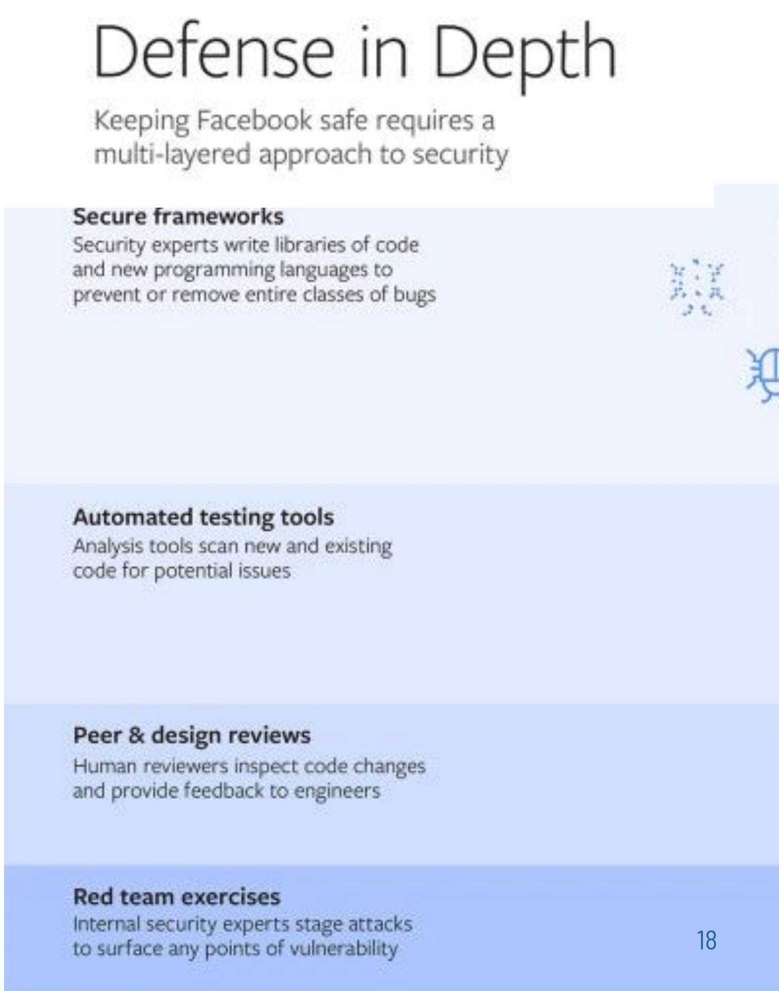
## How is the future shaping up for us?

| | | | |
|---|---|---|---|
| **Today** | Secure By Default | Self Service | Security Partnership |
| **Mid term** | Secure By Default | Self Service | Security Partnership |
| **Long term** | Secure By Default | Self Service | Security Partnership |

# Facebook:

"We invest heavily in building frameworks that help engineers prevent and remove entire classes of bugs when writing code."

*Designing Security For Billions* by Facebook



## Defense in Depth

Keeping Facebook safe requires a multi-layered approach to security

**Secure frameworks**
Security experts write libraries of code and new programming languages to prevent or remove entire classes of bugs

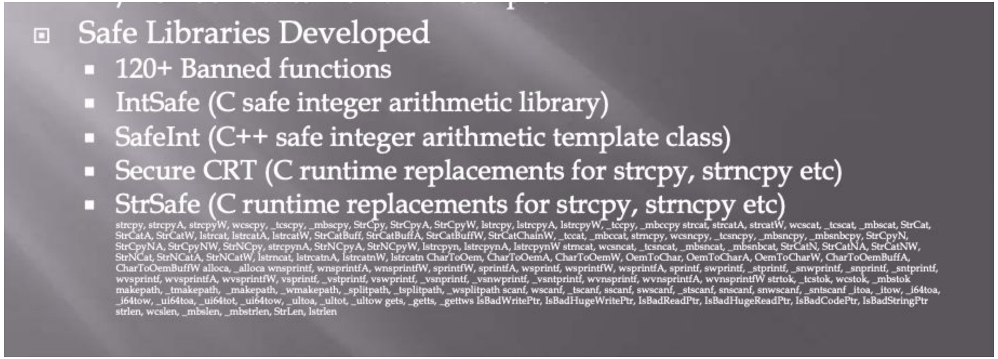
**Automated testing tools**
Analysis tools scan new and existing code for potential issues

**Peer & design reviews**
Human reviewers inspect code changes and provide feedback to engineers

**Red team exercises**
Internal security experts stage attacks to surface any points of vulnerability

# How Valuable Can Banning Functions Be?

41% of vulnerability reduction from XP → Vista from banning *strcpy* and friends



- Safe Libraries Developed
  - 120+ Banned functions
  - IntSafe (C safe integer arithmetic library)
  - SafeInt (C++ safe integer arithmetic template class)
  - Secure CRT (C runtime replacements for strcpy, strncpy etc)
  - StrSafe (C runtime replacements for strcpy, strncpy etc)

strcpy, strcpyA, strcpyW, wcscpy, _tcscpy, _mbscpy, StrCpy, StrCpyA, StrCpyW, lstrcpy, lstrcpyA, lstrcpyW, _tccpy, _mbccpy strcat, strcatA, strcatW, wcscat, _tcscat, _mbscat, StrCat, StrCatA, StrCatW, lstrcat, lstrcatA, lstrcatW, StrCatBuff, StrCatBuffA, StrCatBuffW, StrCatChainW, _tccat, _mbccat, strncpy, wcsncpy, _tcsncpy, _mbsncpy, _mbsnbcpy, StrCpyN, StrCpyNA, StrCpyNW, StrNCpy, strcpynA, StrNCpyA, StrNCpyW, lstrcpyn, lstrcpynA, lstrcpynW strncat, wcsncat, _tcsncat, _mbsncat, _mbsnbcat, StrCatN, StrCatNA, StrCatNW, StrNCat, StrNCatA, StrNCatW, lstrncat, lstrcatnA, lstrcatnW, lstrcatn CharToOem, CharToOemA, CharToOemW, OemToChar, OemToCharA, OemToCharW, CharToOemBuffA, CharToOemBuffW alloca, _alloca wnsprintf, wnsprintfA, wnsprintfW, sprintfW, sprintfA, wsprintf, wsprintfW, wsprintfA, sprintf, swprintf, _stprintf, _snwprintf, _snprintf, _sntprintf, wvsprintf, wvsprintfA, wvsprintfW, vsprintf, _vstprintf, vswprintf, _vsnprintf, _vsnwprintf, _vsntprintf, wvnsprintf, wvnsprintfA, wvnsprintfW strtok, _tcstok, wcstok, _mbstok makepath, _tmakepath, _makepath, _wmakepath, _splitpath, _tsplitpath, _wsplitpath scanf, wscanf, _tscanf, sscanf, swscanf, _stscanf, snscanf, snwscanf, _sntscanf _itoa, _itow, _i64toa, _i64tow, _ui64toa, _ui64tot, _ui64tow, _ultoa, _ultot, _ultow gets, _getts, _gettws IsBadWritePtr, IsBadHugeWritePtr, IsBadReadPtr, IsBadHugeReadPtr, IsBadCodePtr, IsBadStringPtr strlen, wcslen, _mbslen, _mbstrlen, StrLen, lstrlen

Analysis of 63 buffer-related security bugs that affect Windows XP, Windows Server 2003 or Windows 2000 but not Windows Vista: 82% removed through SDL process

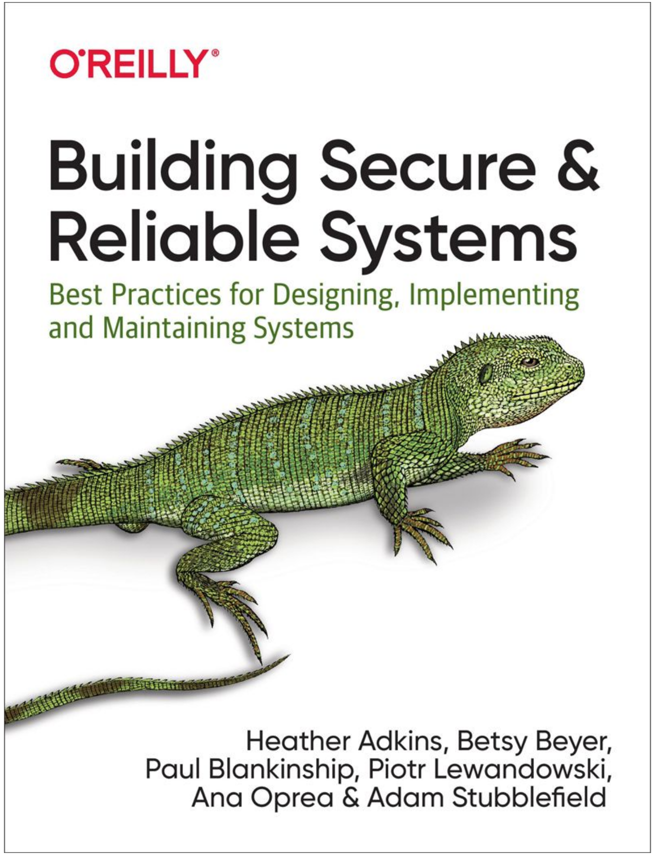
- 27 (43%) found through use of SAL (Annotations)
- **26 (41%) removed through banned API removal**

*"Security Improvements in Windows Vista", Michael Howard*

# Google:

- “It’s **unreasonable** to expect any developer to be an expert in all these subjects, or to constantly maintain vigilance when writing or reviewing code.
- A better approach is to handle security and reliability in **common frameworks**, **languages**, and **libraries**. Ideally, libraries only expose an interface that makes **writing code with common classes of security vulnerabilities impossible**.”

Building Secure and Reliable Systems, by Google



O’REILLY®

Building Secure & Reliable Systems

Best Practices for Designing, Implementing and Maintaining Systems

Heather Adkins, Betsy Beyer, Paul Blankinship, Piotr Lewandowski, Ana Oprea & Adam Stubblefield

# “But I’m not Google”

Framework / tech choices matter

- Mitigate classes of vulnerabilities

Examples:

- Using modern web frameworks
- DOMPurify - output encoding
- re2 - regexes
- tink - crypto

*Web security before modern frameworks*

# Outline

1. Why Bug-Finding Isn't The Answer
2. **How to Eradicate Vulnerability Classes**
3. Tools & Techniques To Make It Real

# How to Eradicate Vulnerability Classes

1. Select a vulnerability class
2. Determine the right approach to find/fix it at scale
3. Select a safe pattern and make it the default
4. Train developers to use the safe pattern
5. Use tools to enforce the safe pattern

# 1. Evaluate which vulnerability class to focus on

## Common selection criteria

Bug classes that are:

1. The most prevalent
2. The highest impact / risk
3. Easiest to tackle (organizationally, technically)
4. Organizational priorities
5. Weighted: `f(prevalent, severe, feasible, org)`

# 1. Evaluate which vulnerability class to focus on

## Vulnerability Management (more)

Know your current state and if your future efforts actually work

**Track**: Severity, vulnerability class, source code responsible, ...

# 1. Evaluate which vulnerability class to focus on

## Vulnerability Management (more)

Know your current state and if your future efforts actually work

**Track**: Severity, vulnerability class, source code responsible, ...

## Build a List of Prior Vulnerabilities to Review

**From**: Issue trackers, commit history, tool or pen test reports, ...

# 1. Evaluate which vulnerability class to focus on

## Vulnerability Management (more)

Know your current state and if your future efforts actually work

**Track**: Severity, vulnerability class, source code responsible, ...

## Build a List of Prior Vulnerabilities to Review

**From**: Issue trackers, commit history, tool or pen test reports, ...

## Review Prior Vulns for Trends

**Within a bug class:** Do the vulnerable code look similar?

# 1. Evaluate which vulnerability class to focus on

## Common selection criteria

Bug classes that are:

1. The most prevalent
2. The highest impact / risk
3. Easiest to tackle (organizationally, technically)
4. Organizational priorities
5. Weighted: `f(prevalent, severe, feasible, org)`

### Ideal World

Choose a vulnerability class that is:

- Widespread across teams/repos
- High Risk
- Feasible to get devs to fix
- Aligns with company priorities
- Always broken in the same way

# 2. How to Find/Fix at Scale?

Big picture, architectural flaws → Threat Modeling

Cloud misconfigurations → IaaC scanning, Cartography, BB

Complex business logic bugs → Pen tests, bug bounty

Protect vulns until they're patched → WAF, RASP

**Known good/known bad code** → **Lightweight static analysis**

# 3. Select a Safe Pattern and Make it the Default

- Based on internal coding guidelines, standards, your expertise, ...

Application Security Verification Standard 4.0

Final

# 3. Select a Safe Pattern and Make it the Default

## Update all internal coding guidelines (security & dev)

- READMEs, developer documentation, wiki pages, FAQs

## Work with developer productivity team

- Secure version should have an even better dev UX than the old way
  - How can we increase dev productivity *and* security?
- Integrate security at the right points (e.g. new project starter templates) to get automatic, widespread adoption
- "Hitch your security wagon to dev productivity." - Astha Singhal

# 4. Help Developers Use the Safe Pattern

## Making Communications Successful

- What and why something is insecure should be clear
  - Use terms developers understand, no security jargon
- Convey impact in terms devs care about
  - Risk to the business, damaging user trust, reliability, up time
- How to fix it should be *concise* and *clear*
  - Link to additional docs and resources with more info

# 5. Use Tools to Enforce the Safe Pattern

Use lightweight static analysis (grep, linting) to ensure the safe patterns are used

# Outline

1. Why Bug-Finding Isn't The Answer
2. How to Eradicate Vulnerability Classes
3. **Tools & Techniques To Make It Real**

# How to Eradicate Vulnerability Classes

1. Evaluate which vulnerability class to focus on
2. Determine the best approach to find/prevent it at scale

→ **How to set up continuous code scanning**

3. Select a safe pattern and make it the default
4. Train developers to use the safe pattern
5. Use tools to enforce the safe pattern

→ **Checking for escape hatches in secure frameworks**

# Continuous Scanning: Related Work

AppSec USA:

- Put Your Robots to Work: Security Automation at Twitter | ‘12
- Providence: rapid vuln prevention (blog, code) | '15
- Cleaning Your Applications’ Dirty Laundry with Scumblr (code) | '16
- Scaling Security Assessment at the Speed of DevOps | '16
- SCORE Bot: Shift Left, at Scale! | ‘18

# Continuous Scanning: Related Work

Salus: How Coinbase Sales Security Automation (blog, code)
*DevSecCon London '18*

Orchestrating Security Tools with AWS Step Functions (slides)
*DeepSec '18*

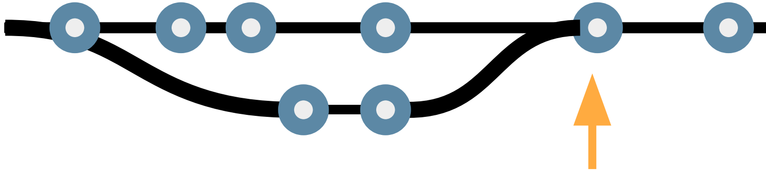
A Case Study of our Journey in Continuous Security (code)
*DevSecCon London '19*

Dracon- Knative Security Pipelines (code)
*Global AppSec Amsterdam '19*

# Continuous Scanning: Best Practices

## Scan Pull Requests

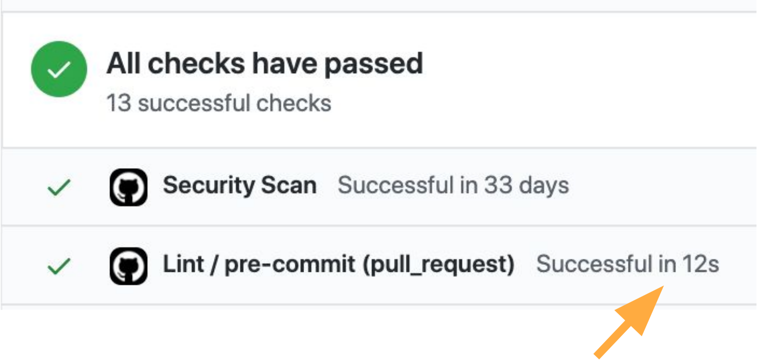
every commit is too noisy, e.g. WIP commits



## Scan Fast (<5min)

feedback while context is fresh

can do longer / more in depth scans daily or weekly



All checks have passed

13 successful checks

✓ Security Scan Successful in 33 days

✓ Lint / pre-commit (pull_request) Successful in 12s

## Two Scanning Workflows

audit (sec team, visibility), blocking (devs, pls fix)

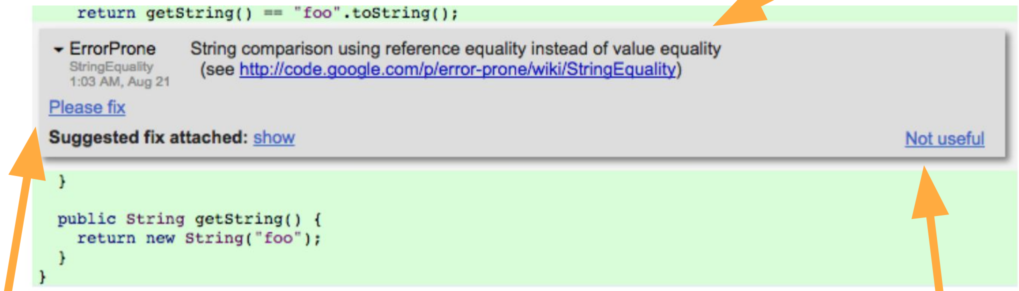
## Make Adjustment Easy

Make it cheap to add/remove tools and new rules

# Continuous Scanning: Best Practices

Show tool findings within dev systems
(e.g. on PR as a comment)

Clear, actionable, with link
to more info



```
    return getString() == "foo".toString();
```

ErrorProne
StringEquality
1:03 AM, Aug 21

String comparison using reference equality instead of value equality
(see http://code.google.com/p/error-prone/wiki/StringEquality)

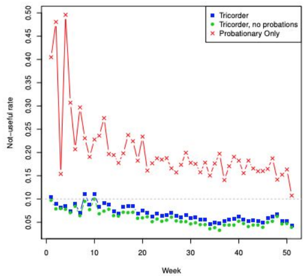
Please fix

**Suggested fix attached:** show

Not useful

```
  }

  public String getString() {
    return new String("foo");
  }
}
```

(Screenshot from Google's, Tricorder: Building a Program Analysis Ecosystem)



Capture metrics about check types, scan runtime, and false positive rates

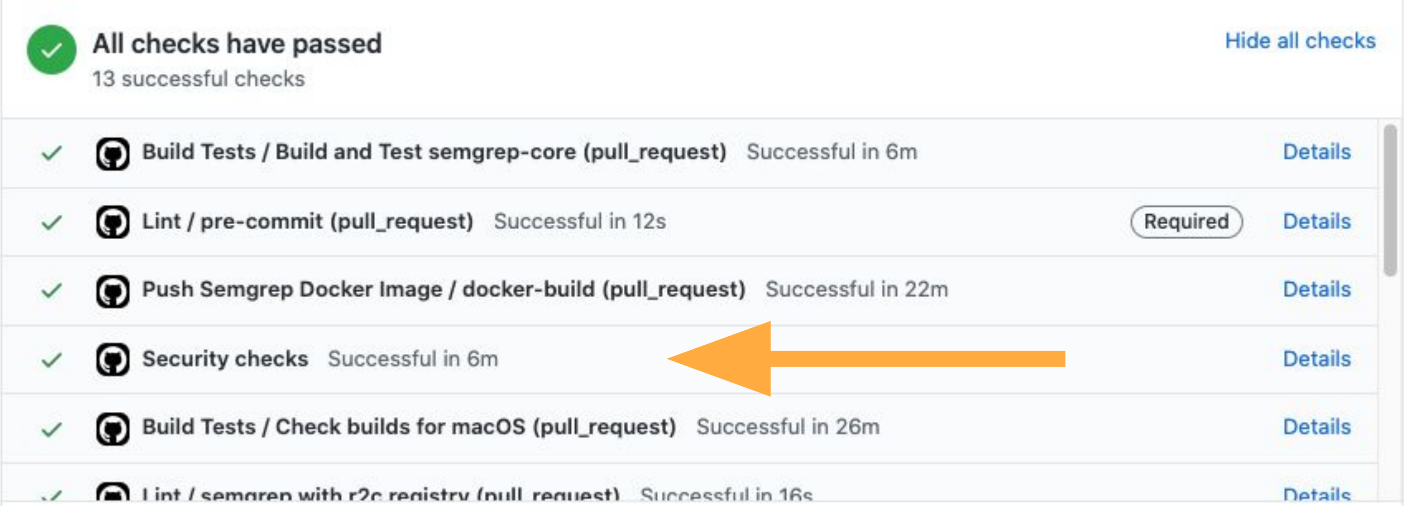
Track & evict low signal checks: keep only +95% true positives

Otherwise causes ill will with devs + too much security team operational cost

# Continuous Scanning: Scan Fast

## Don't come in last!

Security checks should not be **the slowest check blocking developer from merging**



All checks have passed

13 successful checks

Hide all checks

✓ Build Tests / Build and Test semgrep-core (pull_request) Successful in 6m Details

✓ Lint / pre-commit (pull_request) Successful in 12s Required Details

✓ Push Semgrep Docker Image / docker-build (pull_request) Successful in 22m Details

✓ Security checks Successful in 6m Details

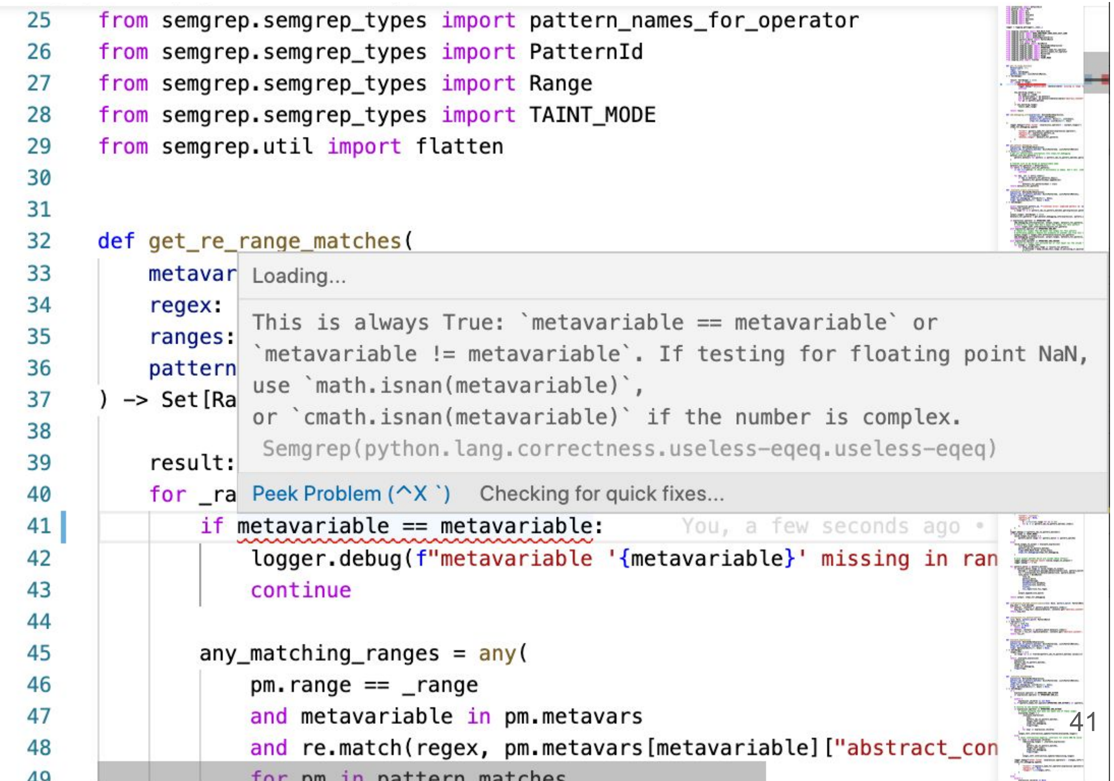
✓ Build Tests / Check builds for macOS (pull_request) Successful in 26m Details

✓ Lint / semgrep with r2c registry (pull_request) Successful in 16s Details

# Continuous Scanning: Keep context fresh

Report violations as early as possible, ideally in the editor.

Also enforce in CI so that it can't be ignored.



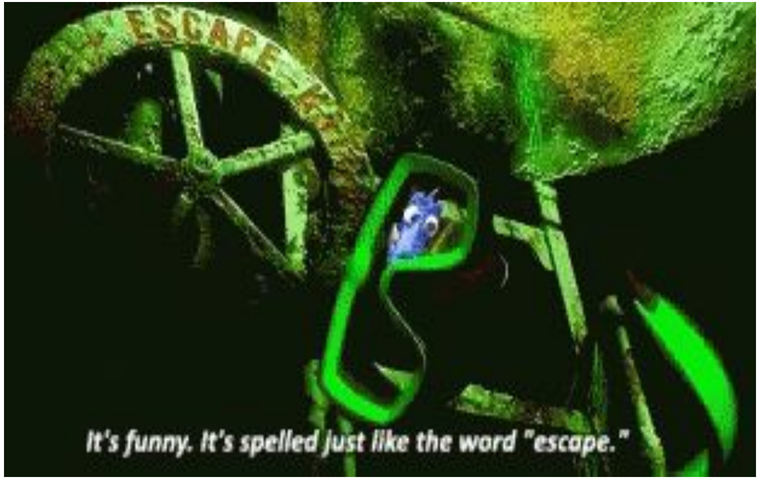
```
from semgrep.semgrep_types import pattern_names_for_operator
from semgrep.semgrep_types import PatternId
from semgrep.semgrep_types import Range
from semgrep.semgrep_types import TAINT_MODE
from semgrep.util import flatten


def get_re_range_matches(
    metavar
    regex:
    ranges:
    pattern
) -> Set[Ra

    result:
    for _ra
        if metavariable == metavariable:      You, a few seconds ago •
            logger.debug(f"metavariable '{metavariable}' missing in ran
            continue

        any_matching_ranges = any(
            pm.range == _range
            and metavariable in pm.metavars
            and re.match(regex, pm.metavars[metavariable]["abstract_con
            for pm in pattern_matches
```

```
Loading...
This is always True: `metavariable == metavariable` or
`metavariable != metavariable`. If testing for floating point NaN,
use `math.isnan(metavariable)`,
or `cmath.isnan(metavariable)` if the number is complex.
 Semgrep(python.lang.correctness.useless-eqeq.useless-eqeq)
```

# Continuously Finding: Escape Hatches

If we use secure frameworks that maintain secure defaults, all we need to do is detect the functions that let you "escape" from those secure defaults. For instance:

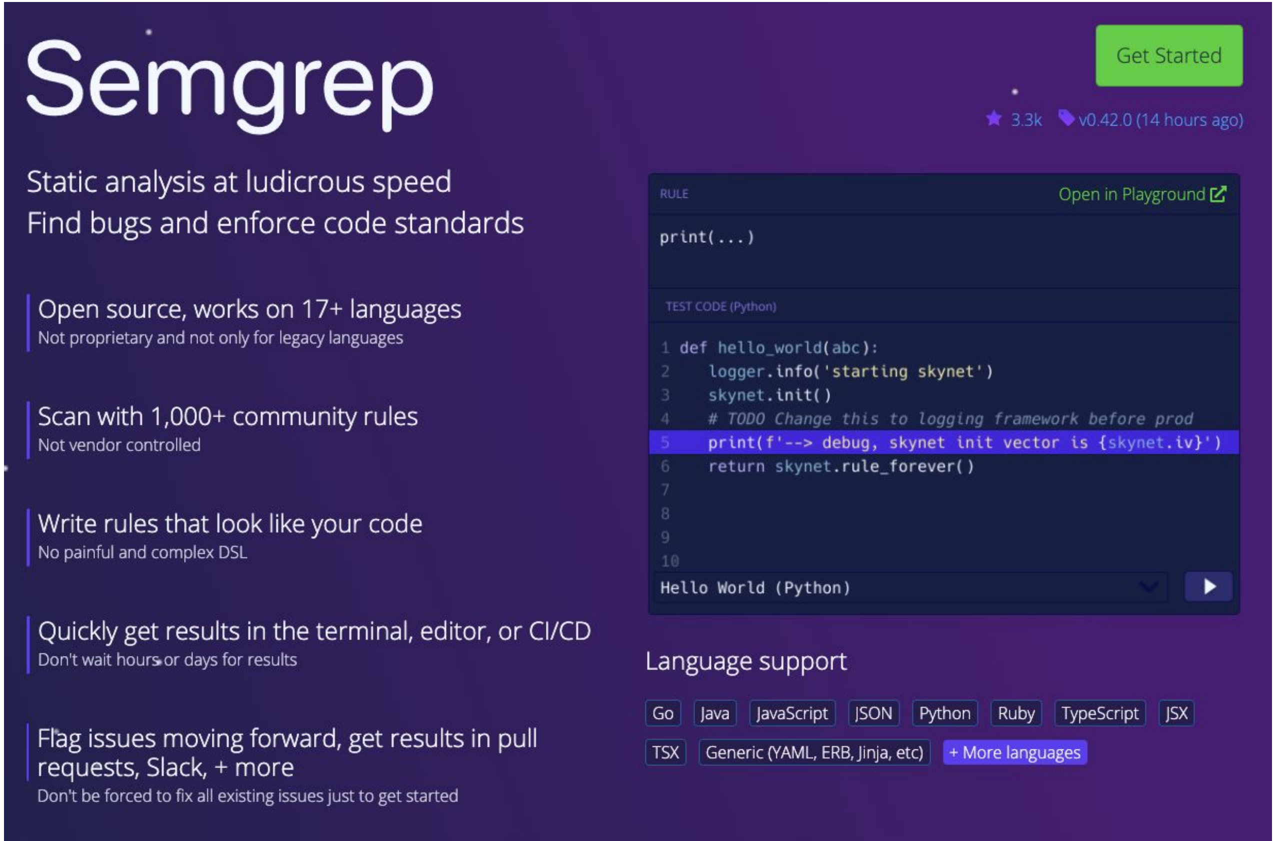
- `dangerouslySetInnerHTML`
- `exec`
- `rawSQL(...)`
- `myorg.make_superuser`



# How to find them?

- **Grep**
  - **Pro**: easy to use, interactive, fast
  - **Con**: line-oriented, mismatch with program structure (ASTs)
- **Code-Aware Linter**
  - **Pro**: robust, precise (handles whitespace, comments, ...)
  - **Con**: Each parser represents ASTs differently; have to learn each syntax
- **Anything else?**

# What we do



## Semgrep

Static analysis at ludicrous speed
Find bugs and enforce code standards

**Open source, works on 17+ languages**
Not proprietary and not only for legacy languages

**Scan with 1,000+ community rules**
Not vendor controlled

**Write rules that look like your code**
No painful and complex DSL

**Quickly get results in the terminal, editor, or CI/CD**
Don't wait hours or days for results

**Flag issues moving forward, get results in pull requests, Slack, + more**
Don't be forced to fix all existing issues just to get started

```
print(...)
```

```python
def hello_world(abc):
   logger.info('starting skynet')
   skynet.init()
   # TODO Change this to logging framework before prod
   print(f'--> debug, skynet init vector is {skynet.iv}')
   return skynet.rule_forever()
```

Language support

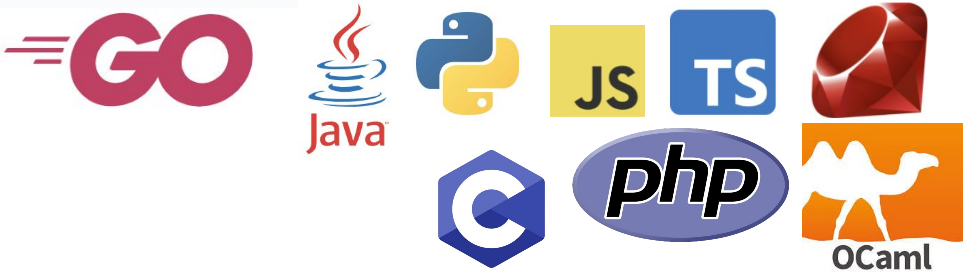
Go | Java | JavaScript | JSON | Python | Ruby | TypeScript | JSX | TSX | Generic (YAML, ERB, Jinja, etc) | + More languages

# Semgrep.dev

- Open source

returntocorp / semgrep LGPL-2.1 License

- Supports many languages



- >1000 out of the box rules
- Does **not** require buildable source code
- 🔥 **No painful DSL, patterns look like the code you’re targeting**

# How to Eradicate Vulnerability Classes

1. Select a vulnerability class
2. Select a safe pattern and make it the default
3. Train developers to use the safe pattern
4. Use tools to enforce the safe pattern

# 1. Select a vulnerability class

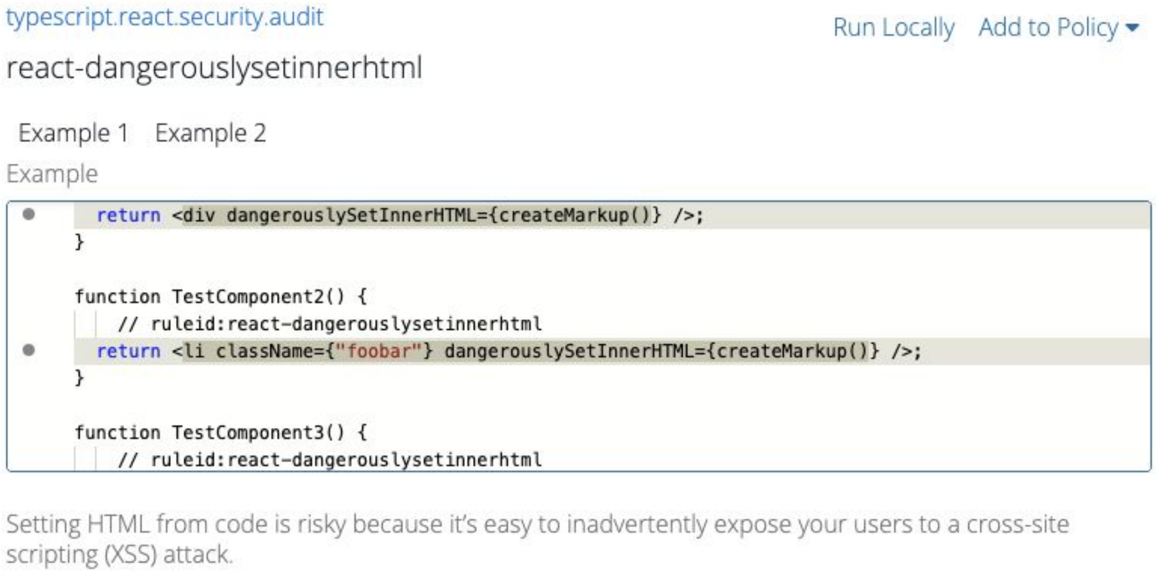
- r2c is young
  - Two (2) primary codebases
  - Limited vulnerability history

- Prioritize based on common problems for the **type** of application:
  - Web application → XSS
  - Command line interface → Code and Command injection

# 2. Select a safe pattern and make it the default

React

Flask

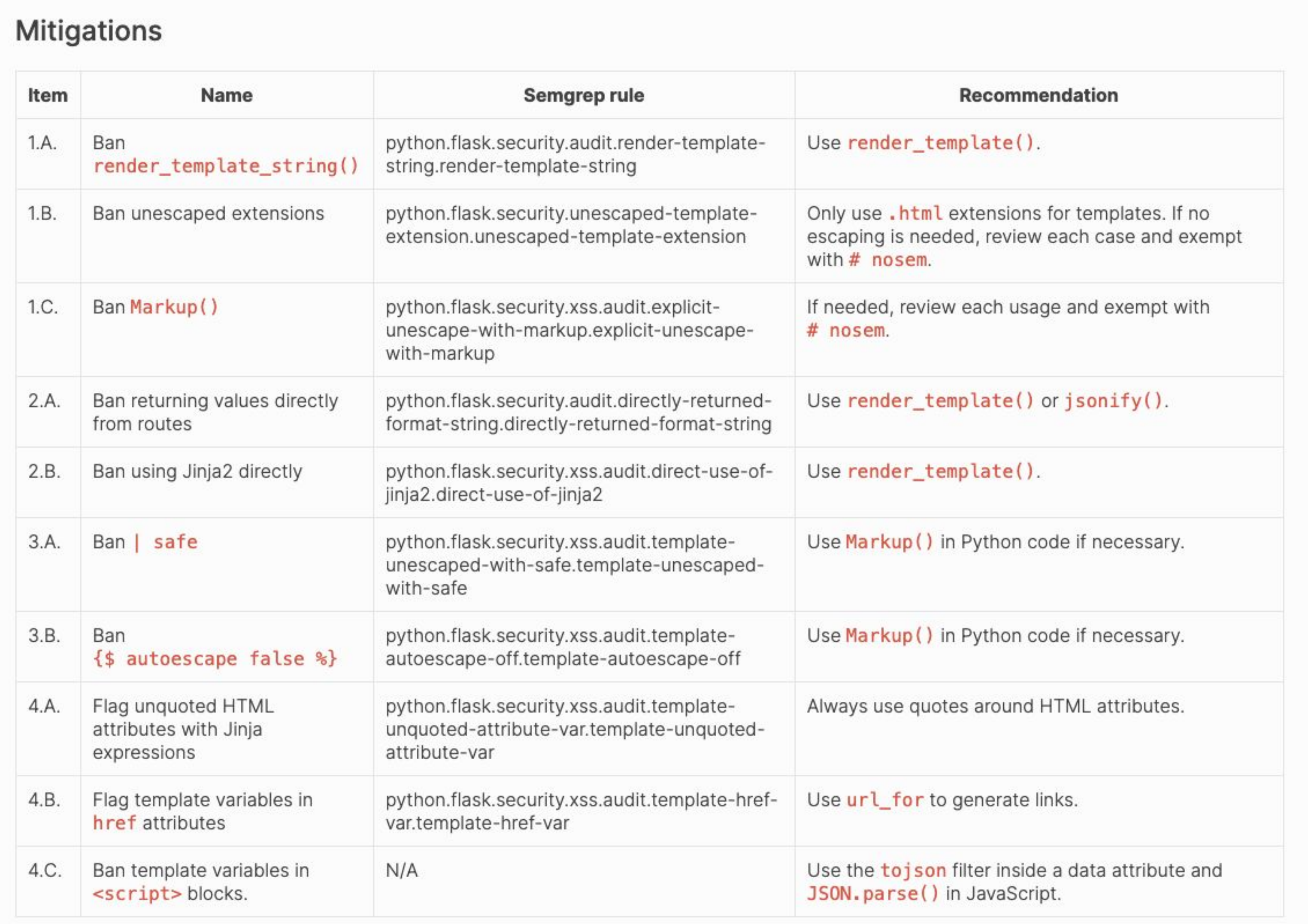


react-dangerouslysetinnerhtml

Example

```
● return <div dangerouslySetInnerHTML={createMarkup()} />;
  }

  function TestComponent2() {
      // ruleid:react-dangerouslysetinnerhtml
● return <li className={"foobar"} dangerouslySetInnerHTML={createMarkup()} />;
  }

  function TestComponent3() {
      // ruleid:react-dangerouslysetinnerhtml
```

Setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site scripting (XSS) attack.

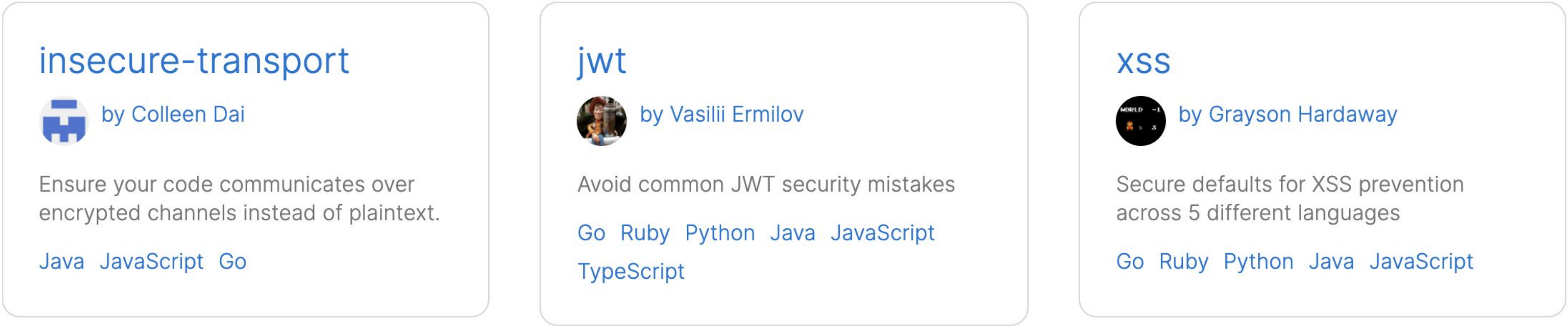


## Mitigations

| Item | Name | Semgrep rule | Recommendation |
|---|---|---|---|
| 1.A. | Ban `render_template_string()` | python.flask.security.audit.render-template-string.render-template-string | Use `render_template()`. |
| 1.B. | Ban unescaped extensions | python.flask.security.unescaped-template-extension.unescaped-template-extension | Only use `.html` extensions for templates. If no escaping is needed, review each case and exempt with `# nosem`. |
| 1.C. | Ban `Markup()` | python.flask.security.xss.audit.explicit-unescape-with-markup.explicit-unescape-with-markup | If needed, review each usage and exempt with `# nosem`. |
| 2.A. | Ban returning values directly from routes | python.flask.security.audit.directly-returned-format-string.directly-returned-format-string | Use `render_template()` or `jsonify()`. |
| 2.B. | Ban using Jinja2 directly | python.flask.security.xss.audit.direct-use-of-jinja2.direct-use-of-jinja2 | Use `render_template()`. |
| 3.A. | Ban `\| safe` | python.flask.security.xss.audit.template-unescaped-with-safe.template-unescaped-with-safe | Use `Markup()` in Python code if necessary. |
| 3.B. | Ban `{$ autoescape false %}` | python.flask.security.xss.audit.template-autoescape-off.template-autoescape-off | Use `Markup()` in Python code if necessary. |
| 4.A. | Flag unquoted HTML attributes with Jinja expressions | python.flask.security.xss.audit.template-unquoted-attribute-var.template-unquoted-attribute-var | Always use quotes around HTML attributes. |
| 4.B. | Flag template variables in `href` attributes | python.flask.security.xss.audit.template-href-var.template-href-var | Use `url_for` to generate links. |
| 4.C. | Ban template variables in `<script>` blocks. | N/A | Use the `tojson` filter inside a data attribute and `JSON.parse()` in JavaScript. |

# Making Secure Defaults Easier

https://semgrep.dev/explore



insecure-transport

by Colleen Dai

Ensure your code communicates over encrypted channels instead of plaintext.

Java JavaScript Go

jwt

by Vasilii Ermilov

Avoid common JWT security mistakes

Go Ruby Python Java JavaScript TypeScript

xss

by Grayson Hardaway

Secure defaults for XSS prevention across 5 different languages

Go Ruby Python Java JavaScript

SECURITY CHEAT SHEETS

Django XSS

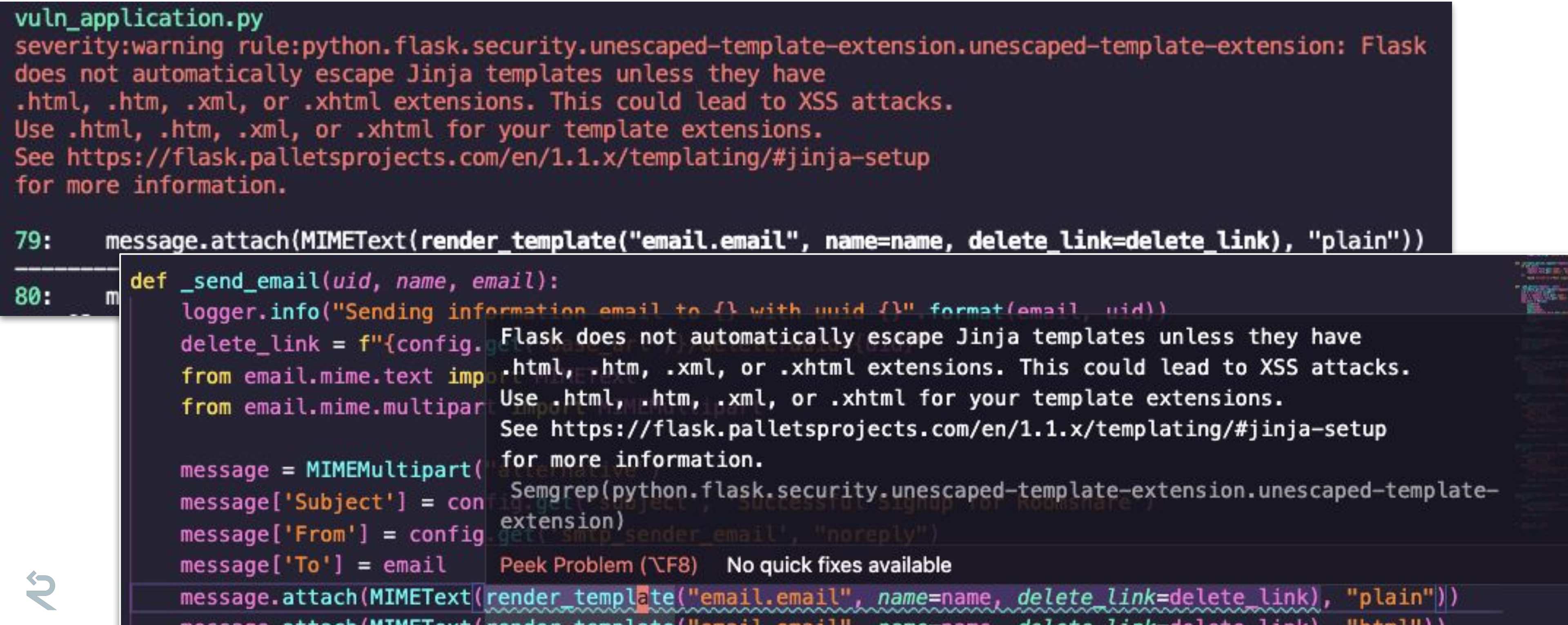
Flask XSS

Java/JSP XSS

Rails XSS

https://semgrep.dev/docs/cheat-sheets/django-xss/

# 3. Train developers to use the safe pattern



```
vuln_application.py
severity:warning rule:python.flask.security.unescaped-template-extension.unescaped-template-extension: Flask
does not automatically escape Jinja templates unless they have
.html, .htm, .xml, or .xhtml extensions. This could lead to XSS attacks.
Use .html, .htm, .xml, or .xhtml for your template extensions.
See https://flask.palletsprojects.com/en/1.1.x/templating/#jinja-setup
for more information.

79:    message.attach(MIMEText(render_template("email.email", name=name, delete_link=delete_link), "plain"))
--------
80:    m
```

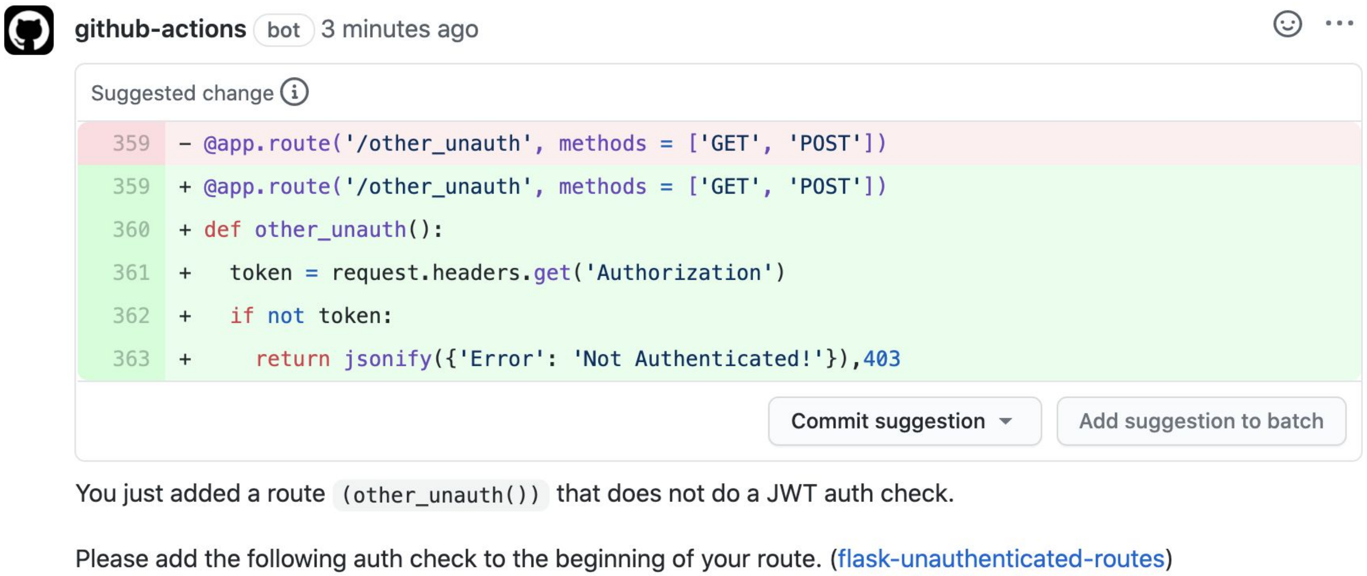
```
def _send_email(uid, name, email):
    logger.info("Sending information email to {} with uuid {}".format(email, uid))
    delete_link = f"{config.
    from email.mime.text imp
    from email.mime.multipar

    message = MIMEMultipart(
    message['Subject'] = con
    message['From'] = config
    message['To'] = email
    message.attach(MIMEText(render_template("email.email", name=name, delete_link=delete_link), "plain"))
```

```
Flask does not automatically escape Jinja templates unless they have
.html, .htm, .xml, or .xhtml extensions. This could lead to XSS attacks.
Use .html, .htm, .xml, or .xhtml for your template extensions.
See https://flask.palletsprojects.com/en/1.1.x/templating/#jinja-setup
for more information.
 Semgrep(python.flask.security.unescaped-template-extension.unescaped-template-
extension)
```

# Autofix

Make security fixes fast and easy.

Even an imperfect suggestion is better than nothing!

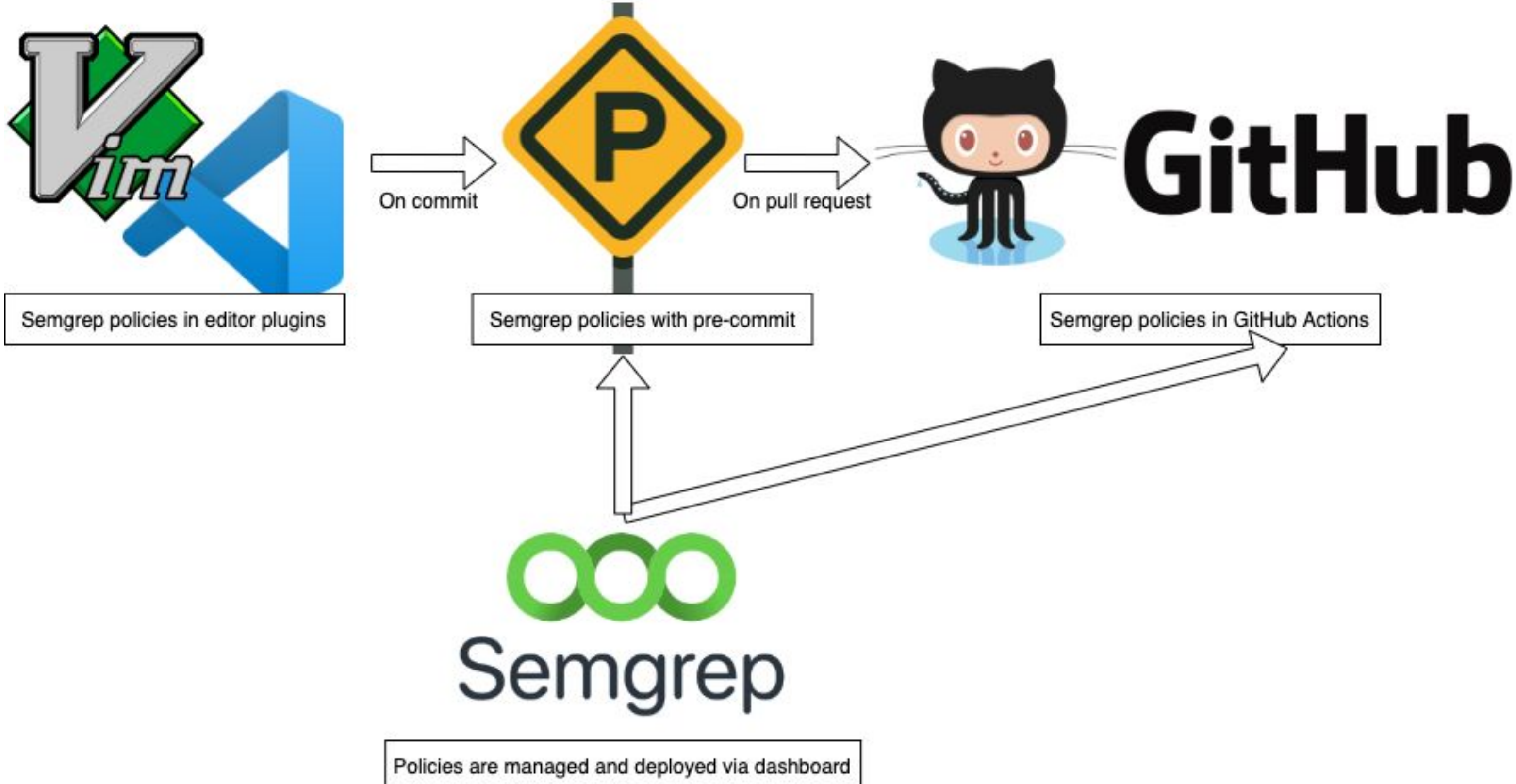


```
- @app.route('/other_unauth', methods = ['GET', 'POST'])
+ @app.route('/other_unauth', methods = ['GET', 'POST'])
+ def other_unauth():
+   token = request.headers.get('Authorization')
+   if not token:
+     return jsonify({'Error': 'Not Authenticated!'}),403
```

You just added a route `(other_unauth())` that does not do a JWT auth check.

Please add the following auth check to the beginning of your route. (flask-unauthenticated-routes)

# 4. Use tools to enforce the safe pattern

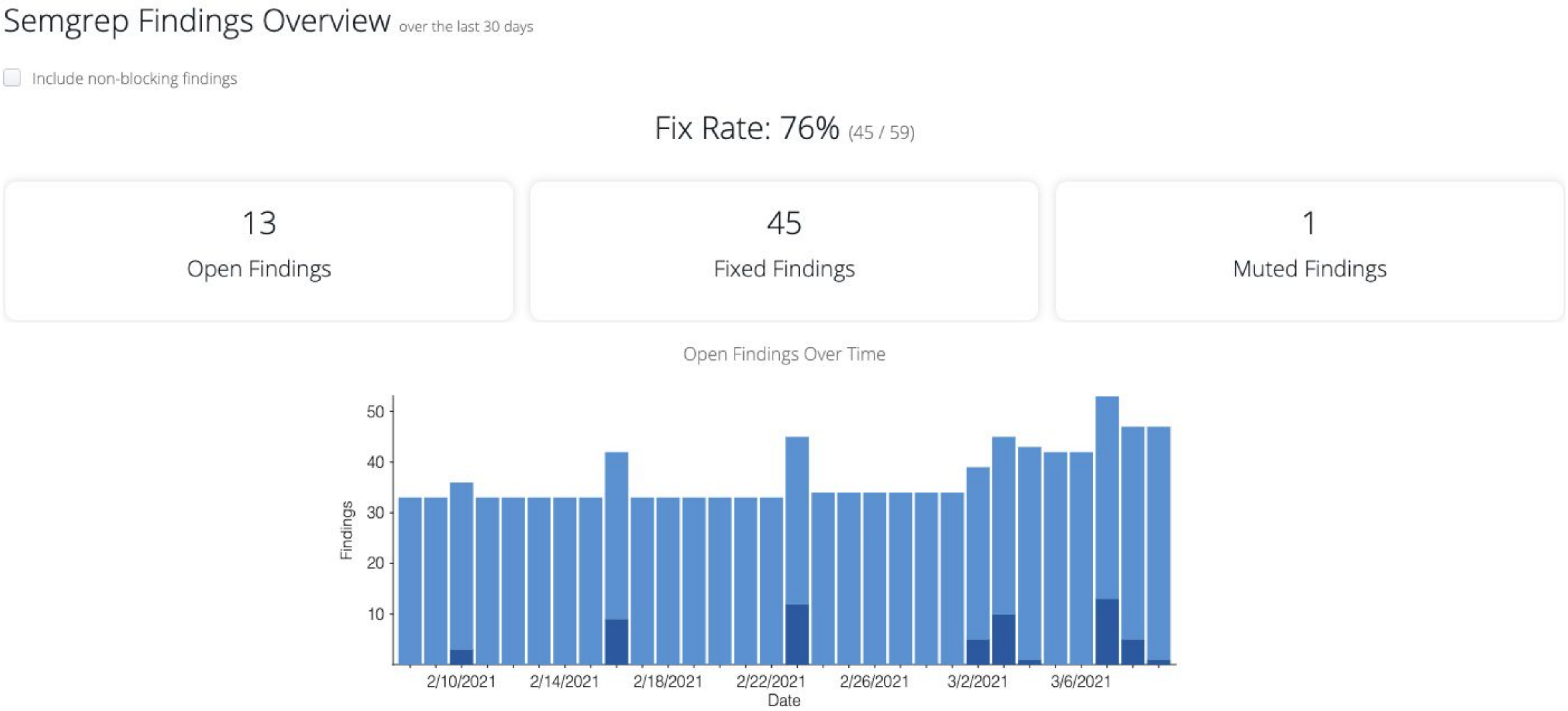


Semgrep policies in editor plugins

Semgrep policies with pre-commit

Semgrep policies in GitHub Actions

Policies are managed and deployed via dashboard



## Semgrep Findings Overview over the last 30 days

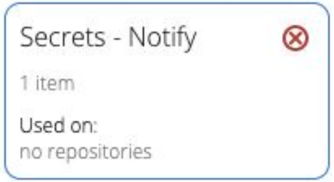
□ Include non-blocking findings

Fix Rate: 76% (45 / 59)

| 13 | 45 | 1 |
|---|---|---|
| Open Findings | Fixed Findings | Muted Findings |

Open Findings Over Time

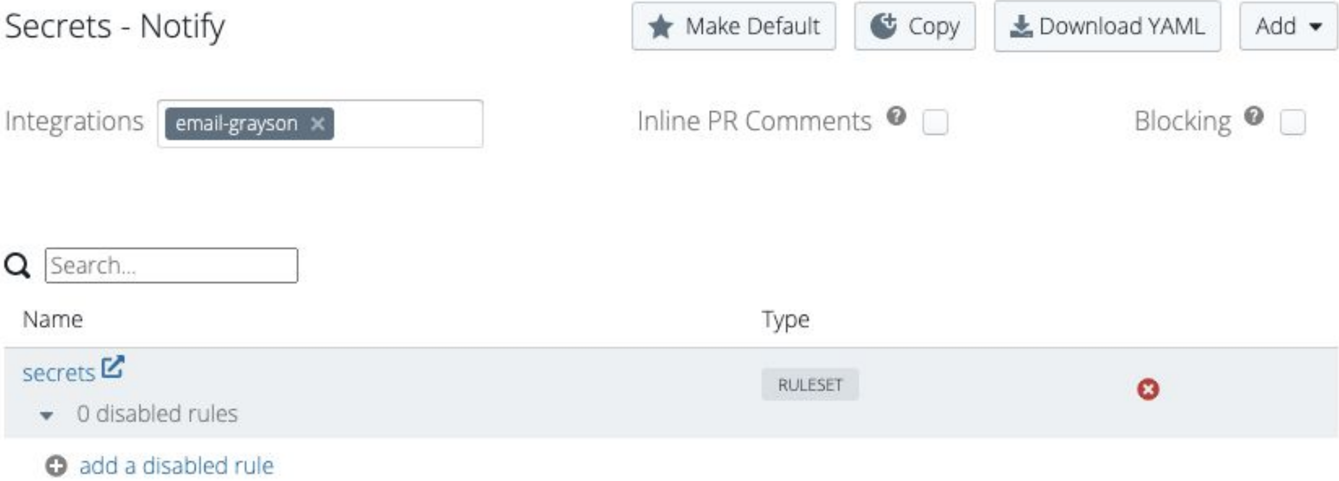
# BONUS: Quietly monitor new policies



Secrets - Notify ⊗

1 item

Used on:
no repositories



Secrets - Notify

Integrations: email-grayson ×

Inline PR Comments ☐

Blocking ☐

Search...

| Name | Type |
|---|---|
| secrets<br>▾ 0 disabled rules | RULESET |

⊕ add a disabled rule

# Conclusion

- Secure defaults are the best way to scalably raise your security bar
  - **Not** finding bugs (bug whack-a-mole)
- Killing bug classes makes your AppSec team more leveraged
- Define safe pattern → educate / roll out → enforce continuously
  - Fast & lightweight (e.g. semgrep), focus on dev UX

**Slides:** http://bit.ly/2021Berman-OWASP-Denver

Adam Berman

# Outline

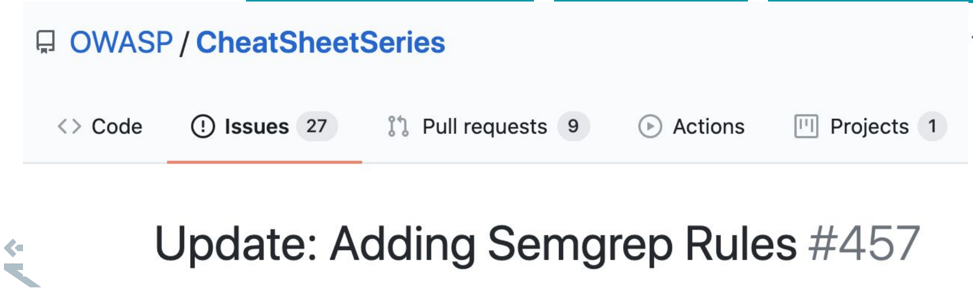
1. Why Bug-Finding Isn't The Answer
2. How to Eradicate Vulnerability Classes
3. Tools & Techniques To Make It Real
4. **Community Collaboration**

# Partnering with OWASP

- Partnership between Semgrep + OWASP ASVS, Cheat Sheets
- **Goal**: Out of the box support for:
  - Verifying if your code is compliant with ASVS Level 1
  - Finding code that violates Cheat Sheets best practice recommendations

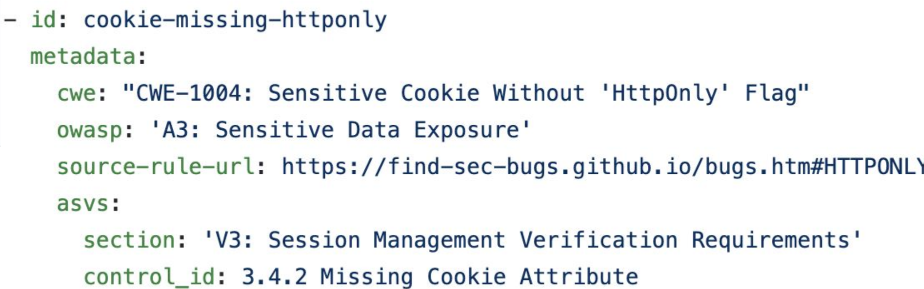
Want to get involved? Let's talk! 🙌

Thanks to Daniel Cuthbert, Joe Bollen, Rohit Salecha, and more



OWASP / CheatSheetSeries

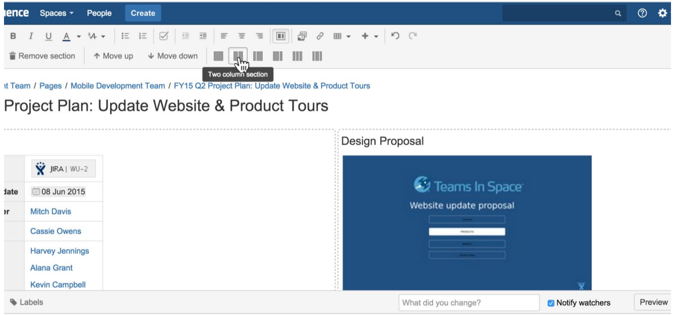
Update: Adding Semgrep Rules #457



```
- id: cookie-missing-httponly
  metadata:
    cwe: "CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag"
    owasp: 'A3: Sensitive Data Exposure'
    source-rule-url: https://find-sec-bugs.github.io/bugs.htm#HTTPONLY
    asvs:
      section: 'V3: Session Management Verification Requirements'
      control_id: 3.4.2 Missing Cookie Attribute
```

# Why Semgrep is 😍 for AppSec Engineers & Developers

## Coding Standards ➡️ Enforce Continuously



t Team / Pages / Mobile Development Team / FY15 Q2 Project Plan: Update Website & Product Tours

Project Plan: Update Website & Product Tours

| | |
|---|---|
| | JIRA \| WU-2 |
| date | 08 Jun 2015 |
| er | Mitch Davis |
| | Cassie Owens |
| | Harvey Jennings<br>Alana Grant<br>Kevin Campbell |

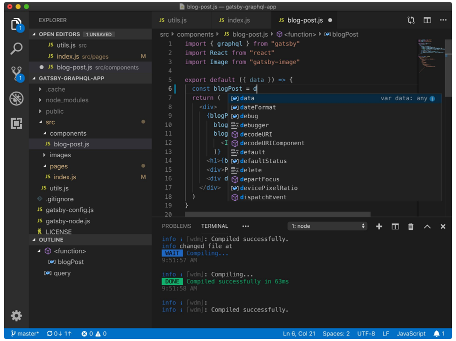
Design Proposal

Teams In Space

Website update proposal

What did you change?

☑ Notify watchers

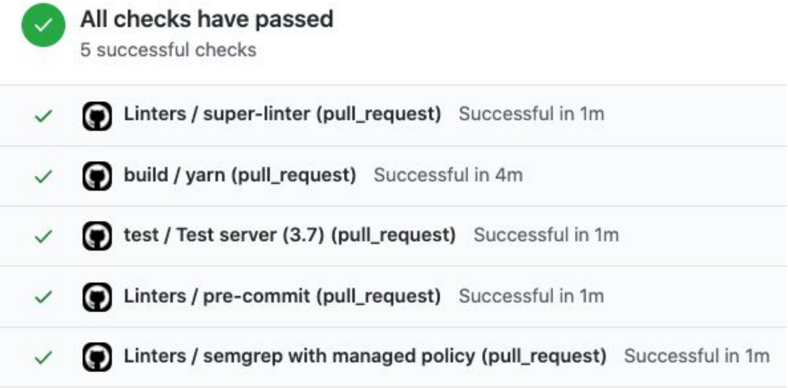


src › components › blog-post.js › <function> › blogPost

```
import { graphql } from "gatsby"
import React from "react"
import Image from "gatsby-image"

export default ({ data }) => {
  const blogPost = d
  return (
    <div>
      {blogP
        blog
        blog
          <I
        )}
      <h1>{b
      <div>P
      <div d
    </div>
  )
}
```

```
data                  var data: any
dateFormat
debug
debugger
decodeURI
decodeURIComponent
default
defaultStatus
delete
departFocus
devicePixelRatio
dispatchEvent
```

```
info ｉ ｢wdm｣: Compiled successfully.
info changed file at
 WAIT  Compiling...
9:51:57 AM

info ｉ ｢wdm｣: Compiling...
 DONE  Compiled successfully in 63ms
9:51:58 AM

info ｉ ｢wdm｣:
info ｉ ｢wdm｣: Compiled successfully.
```



All checks have passed

5 successful checks

✓ Linters / super-linter (pull_request) Successful in 1m

✓ build / yarn (pull_request) Successful in 4m

✓ test / Test server (3.7) (pull_request) Successful in 1m

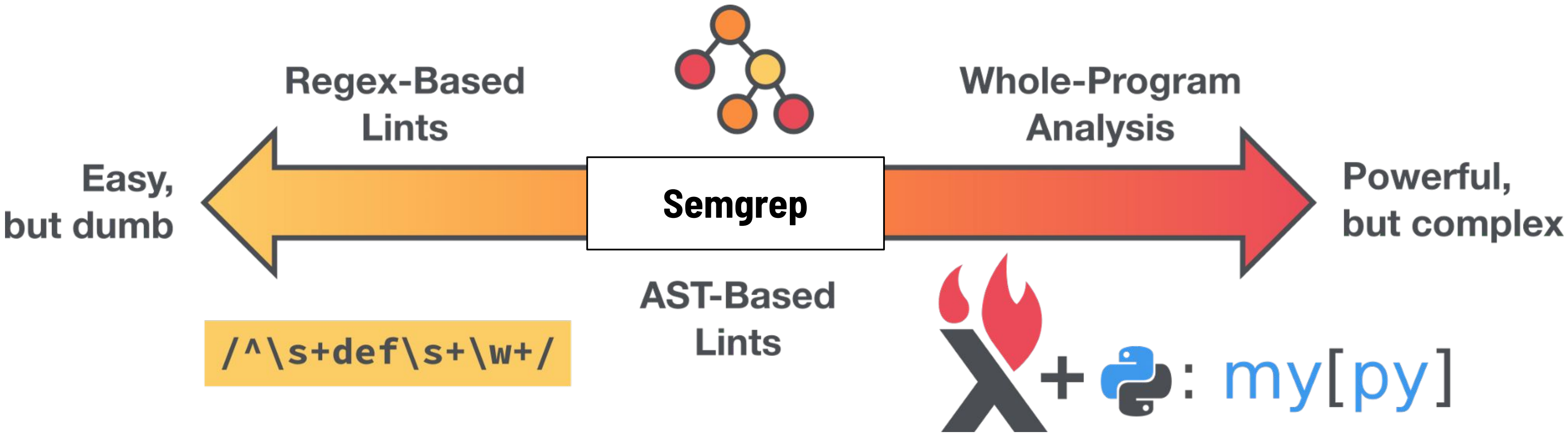
✓ Linters / pre-commit (pull_request) Successful in 1m

✓ Linters / semgrep with managed policy (pull_request) Successful in 1m

Static Analysis at Scale: An Instagram Story

Benjamin Woodruff

Aug 15, 2019 · 13 min read



Easy, but dumb

Regex-Based Lints

```
/^\s+def\s+\w+/
```

Semgrep

AST-Based Lints

Whole-Program Analysis

Powerful, but complex

https://instagram-engineering.com/static-analysis-at-scale-an-instagram-story-8f498ab71a0c

# Our Worldview

- **Speed matters** - scan in minutes, not hours/days
- **False Negatives > False Positives**
- **Ease of use is key**
  - Huge value in org-specific and code base specific checks
  - Heavily prioritize first time user experience, “average” users
  - Accessible to developers, not just security professionals
- **Enforcing secure defaults > bug finding** (more)

# Design Decisions

Given:

- Speed matters
- False Negatives > False Positives
- Ease of use is key
- Enforcing secure defaults > bug finding

Semgrep:

- **Focuses on single file / localized analysis**
  - Interprocedural data flow analysis is slow/imprecise
  - Almost always sufficient for enforcing secure defaults
  - Doesn’t require buildable source, fast
- **Has rules that look like source code (can’t express everything)**

# Popular SAST Vendors

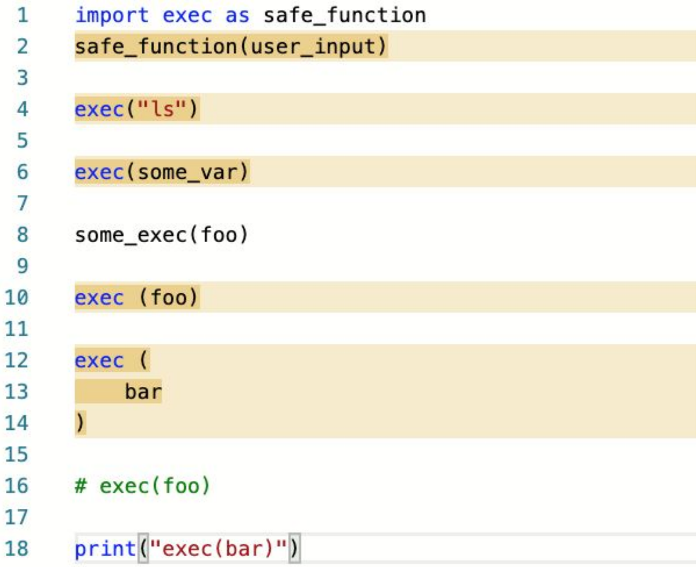
| | | | Semgrep |
|---|---|---|---|
| Open s | | | ✅ |
| Open s | | | ✅ |
| Freely<br>**source** | | | ✅ |
| Open source **SaaS app** | ❌ | ❌ | ❌ |

# How to find them?

- **Grep**
  - **Pro**: easy to use, interactive, fast
  - **Con**: line-oriented, mismatch with program structure (ASTs)
- **Code-Aware Linter**
  - **Pro**: robust, precise (handles whitespace, comments, ...)
  - **Con**: Each parser represents ASTs differently; have to learn each syntax
- **Semgrep**
  - **Pro**: Handles languages with “more than one way to do it”
  - **Pro**: Single tool for multiple languages, simple pattern language
  - **Con**: Slower than grep, not all languages supported

# Finding *exec*

```
$ semgrep -e 'exec(...)' -lang py exec.py
```

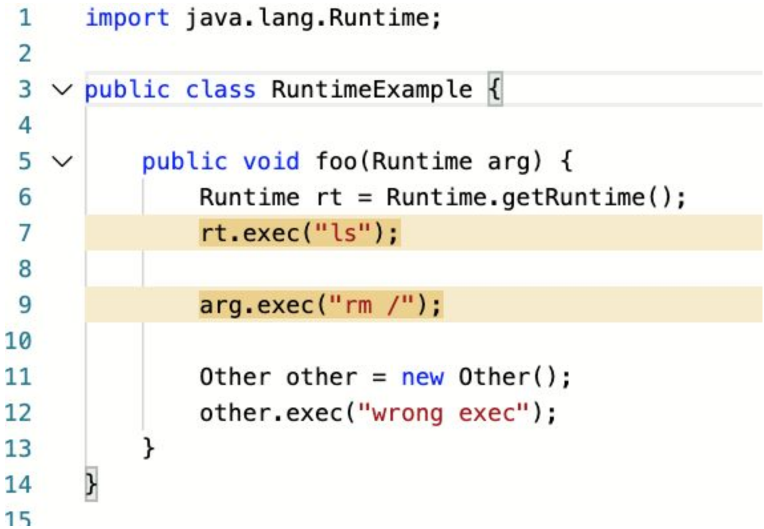


```python
import exec as safe_function
safe_function(user_input)

exec("ls")

exec(some_var)

some_exec(foo)

exec (foo)

exec (
    bar
)

# exec(foo)

print("exec(bar)")
```

Try it: [https://semgrep.dev/ievans:python-exec](https://semgrep.dev/ievans:python-exec)

# Secure defaults + types

```
$ semgrep -e '(Runtime $X).exec(...);' -lang java test.java
```



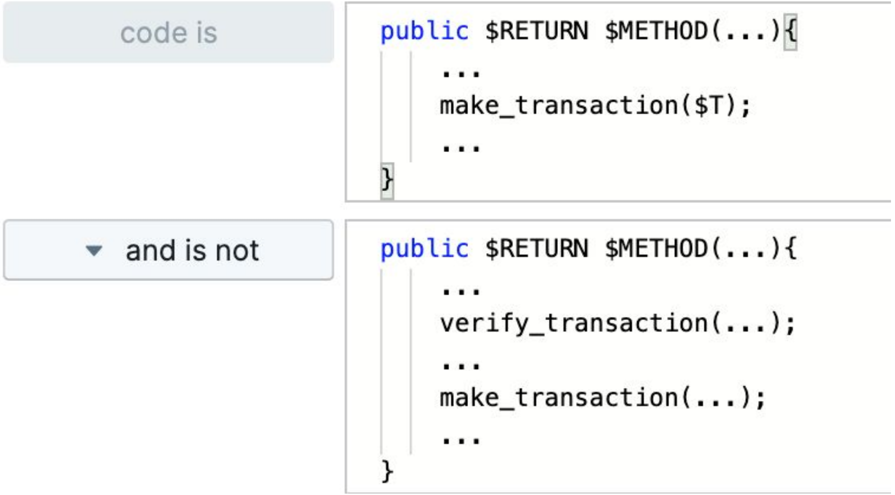
```java
import java.lang.Runtime;

public class RuntimeExample {

    public void foo(Runtime arg) {
        Runtime rt = Runtime.getRuntime();
        rt.exec("ls");

        arg.exec("rm /");

        Other other = new Other();
        other.exec("wrong exec");
    }
}

```

Try it: https://semgrep.live/clintgibler:java-runtime-exec-try

Solution: https://semgrep.live/clintgibler:java-runtime-exec

# Beyond OWASP Top 10: Business Logic

"call verify_transaction() before "make_transaction()"



code is

```
public $RETURN $METHOD(...){
    ...
    make_transaction($T);
    ...
}
```

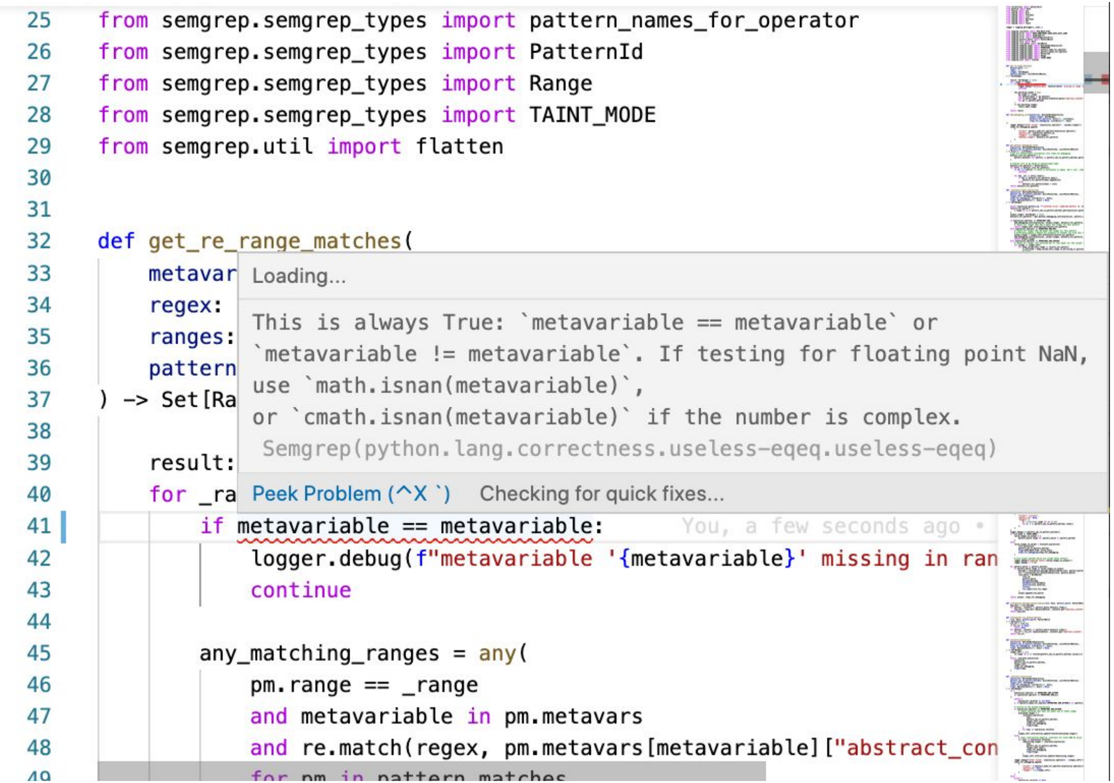
▾ and is not

```
public $RETURN $METHOD(...){
    ...
    verify_transaction(...);
    ...
    make_transaction(...);
    ...
}
```

Try it: https://semgrep.dev/ievans:make-transaction-try
Solution: https://semgrep.dev/ievans:make-transaction

# IDE Integration

Tell me as soon as possible (ideally in editor)



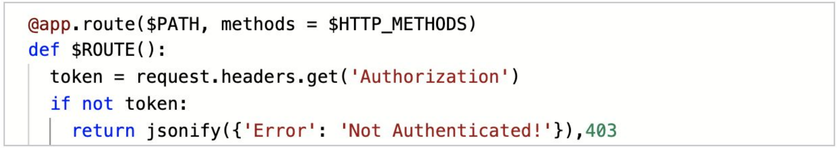
```
from semgrep.semgrep_types import pattern_names_for_operator
from semgrep.semgrep_types import PatternId
from semgrep.semgrep_types import Range
from semgrep.semgrep_types import TAINT_MODE
from semgrep.util import flatten


def get_re_range_matches(
    metavar
    regex:
    ranges:
    pattern
) -> Set[Ra

    result:
    for _ra
        if metavariable == metavariable:      You, a few seconds ago •
            logger.debug(f"metavariable '{metavariable}' missing in ran
            continue

        any_matching_ranges = any(
            pm.range == _range
            and metavariable in pm.metavars
            and re.match(regex, pm.metavars[metavariable]["abstract_con
            for pm in pattern_matches
```

Loading...

```
This is always True: `metavariable == metavariable` or
`metavariable != metavariable`. If testing for floating point NaN,
use `math.isnan(metavariable)`,
or `cmath.isnan(metavariable)` if the number is complex.
 Semgrep(python.lang.correctness.useless-eqeq.useless-eqeq)
```

# Autofix

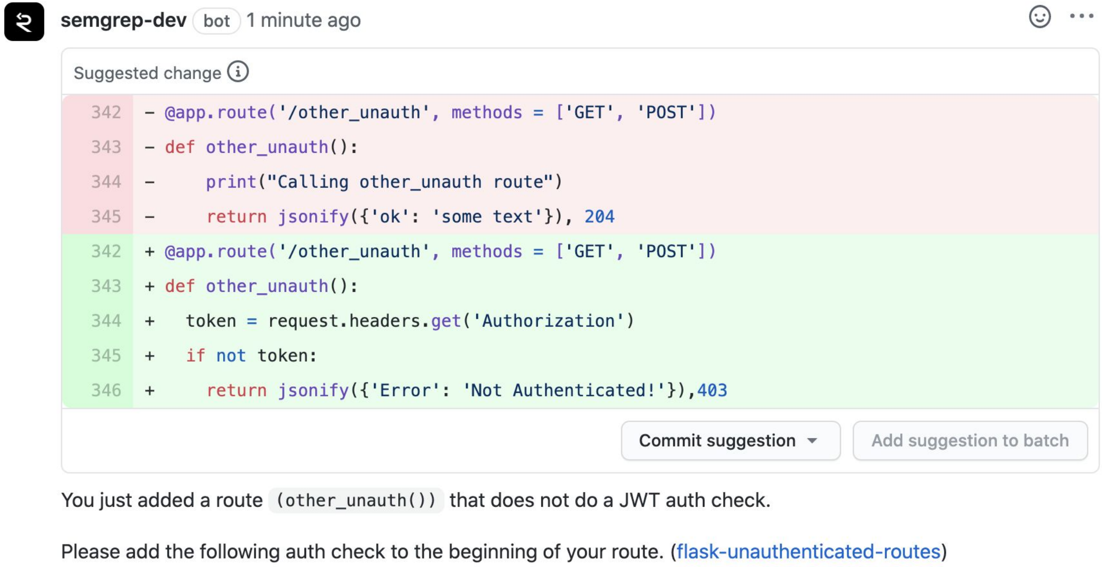
and autofix is



```
@app.route($PATH, methods = $HTTP_METHODS)
def $ROUTE():
  token = request.headers.get('Authorization')
  if not token:
    return jsonify({'Error': 'Not Authenticated!'}),403
```

Make security fixes fast and easy.

Even an imperfect suggestion is better than nothing!



```
- @app.route('/other_unauth', methods = ['GET', 'POST'])
- def other_unauth():
-     print("Calling other_unauth route")
-     return jsonify({'ok': 'some text'}), 204
+ @app.route('/other_unauth', methods = ['GET', 'POST'])
+ def other_unauth():
+   token = request.headers.get('Authorization')
+   if not token:
+     return jsonify({'Error': 'Not Authenticated!'}),403
```

You just added a route `(other_unauth())` that does not do a JWT auth check.

Please add the following auth check to the beginning of your route. (flask-unauthenticated-routes)

# Quiz: Does this app have RCE?

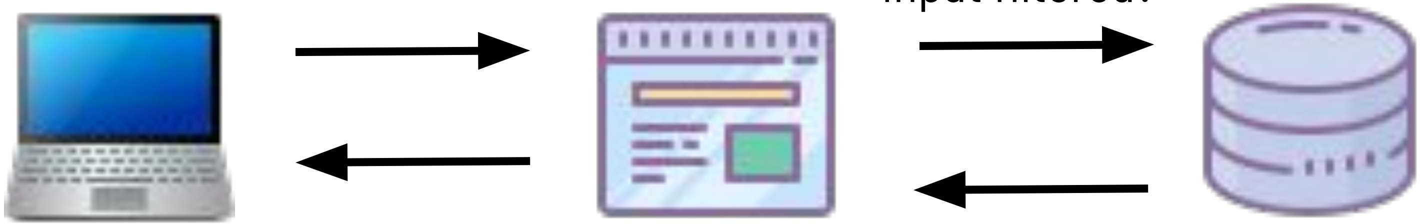
Icons by

# Quiz: Does this app have RCE?

What does user control?
Structure of data?

Input filtered?

How is it stored
(field types,
constraints)



Does the app?

- Deserialize data
- Run shell commands
- Mix data and code
  - eval(), exec()
  - Metaprogramming

Icons by

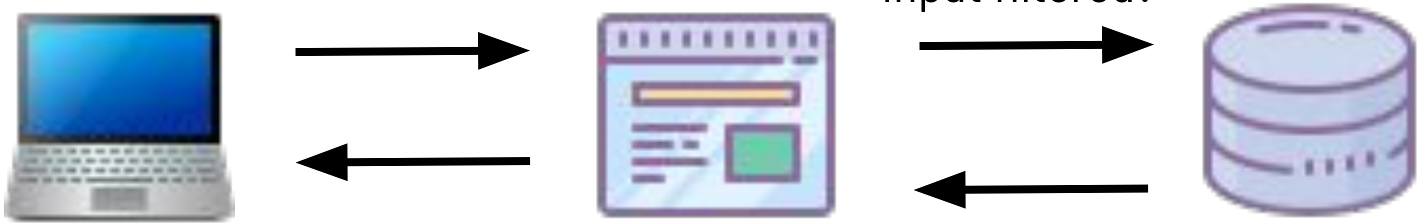
# Quiz: Does this app have RCE?

***Ban:* exec(), eval(), shell exec, deserialization *(objects, YAML, XML, JSON)***

What does user control?
Structure of data?

Input filtered?

How is it stored
(field types,
constraints)

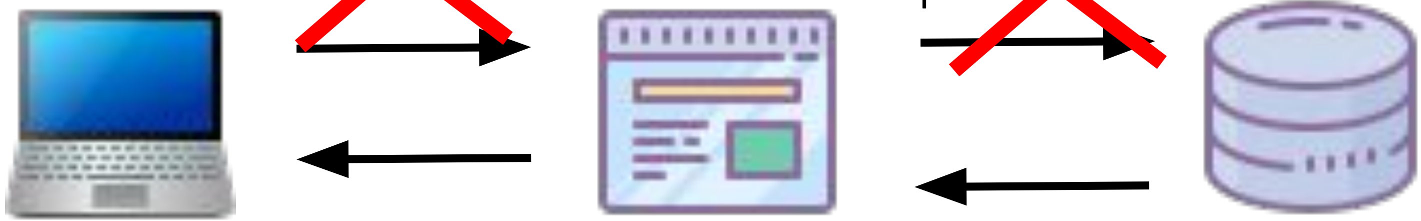


Does the app?

- Deserialize data
- Run shell commands
- Mix data and code
    - eval(), exec()
    - Metaprogramming

# Quiz: Does this app have RCE?

***Ban:* *exec()*, *eval()*, *shell exec*, *deserialization* *(objects, YAML, XML, JSON)***



~~What does user control?~~
~~Structure of data?~~

~~Input filtered?~~

~~How is it stored~~
~~(field types,~~
~~constraints)~~

Does the app?

- ~~Deserialize data~~
- ~~Run shell commands~~
- ~~Mix data and code~~
  - ~~eval(), exec()~~
  - Metaprogramming

Icons by

# Secure Defaults: Challenges in Practice

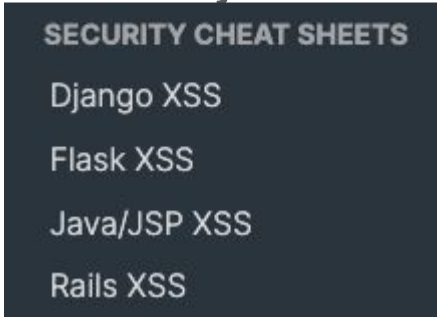
“If this is such a good idea, why ~~aren’t you rich~~ isn’t everyone doing it already?”

1. What secure defaults should I use?
2. Rolling out requires org-wide buy-in
3. Enforcing secure defaults

# Secure Defaults: Challenges in Practice

"If this is such a good idea, why ~~aren't you rich~~ isn't everyone doing it already?"

1. What secure defaults should I use? → Docs
2. Rolling out requires org-wide buy-in
3. Enforcing secure defaults



SECURITY CHEAT SHEETS

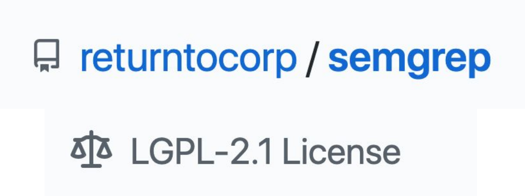
Django XSS

Flask XSS

Java/JSP XSS

Rails XSS

- Onboarding
- Coding standards
- Code quality



returntocorp / semgrep

LGPL-2.1 License

Semgrep

Static analysis at ludicrous speed
Find bugs and enforce code standards