Eradicate Vulnerability Classes
With Secure Defaults & Lightweight Enforcement

Adam Berman | r2c.dev

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

Waterfall development
Dev, Ops
On prem

After

Agile development
DevOps
Cloud
Massive Shifts in Tech and Security

Before

- Waterfall development
- Dev, Ops
- On prem
- Finding vulnerabilities

After

- Agile development
- DevOps
- Cloud
- Secure defaults
Quiz: Does this app have XSS?
Quiz: Does this app have XSS?

What does user control? Structure of data?
- HTML
- HTML attribute
- JavaScript
- ...

Input filtered?

Data processed before sent to user?

How is it stored?
(field types, constraints)

DB type?

Context?

Data processed before sent to user?

Input filtered?

How is it stored?
(field types, constraints)

DB type?

Data processed before sent to user?

Input filtered?

How is it stored?
(field types, constraints)

DB type?

Data processed before sent to user?

Input filtered?

How is it stored?
(field types, constraints)

DB type?

Data processed before sent to user?

Input filtered?

How is it stored?
(field types, constraints)

DB type?

Data processed before sent to user?

Input filtered?

How is it stored?
(field types, constraints)

DB type?

Data processed before sent to user?
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?

Data processed before sent to user?

Context?
- HTML
- HTML attribute
- JavaScript
- ...

Icons by Icons8
Quiz: Does this app have XSS?

Guardrail: Frontend is React, banned dangerouslySetInnerHTML

What does user control?
Structure of data?

Input filtered?

Data processed before sent to user?

How is it stored?
(field types, constraints)

DB type?

Context?
- HTML
- HTML attribute
- JavaScript
- ...

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

- Detect use of (in)secure library
  - Effort Required
- Find potential bug
- Confirm it's a real bug
- Write proof of concept exploit

Effort Required (ahu)

Task
Detecting (lack of) use of secure defaults is much easier than finding bugs.
Compounding Effects of Killing Bug Classes

- Threat Modeling
- Running security tools
- XSS
- SQL Injection
- Triaging bug bounty
- Security training
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
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
"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
How Valuable Can Banning Functions Be?

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

"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
“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
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: \( f(\text{prevalent, severe, feasible, org}) \)
Vulnerability Management (more)

1. Evaluate which vulnerability class to focus on

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: $f(\text{prevalent, severe, feasible, org})$

---

**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, ...
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

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

Track & evict low signal checks:
keep only +95% true positives
Otherwise causes ill will with devs + too much security team operational cost

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

(Screenshot from Google's, Tricorder: Building a Program Analysis Ecosystem)
Continuous Scanning: Scan Fast

Don't come in last!

Security checks should not be *the slowest check blocking developer from merging*
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.
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 1.7+ 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

Language support

- Go
- Java
- JavaScript
- JSON
- Python
- Ruby
- TypeScript
- JSX
- TSX
- Generic (YAML, ERB, Jinja, etc)
Semgrep.dev

- Open source
- 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

Example 1

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

Example 2

```
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

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Semgrep rule</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.A</td>
<td>Ban <code>render_template_string()</code></td>
<td><code>python.flask.security.audit.render-template-string.render-template-string</code></td>
<td>Use <code>render_template()</code></td>
</tr>
<tr>
<td>1.B</td>
<td>Ban unescaped extensions</td>
<td><code>python.flask.security.unescaped-template-extension.unescaped-template-extension</code></td>
<td>Only use <code>.html</code> extensions for templates. If no escaping is needed, review each case and exempt with <code># nosem</code>.</td>
</tr>
<tr>
<td>1.C</td>
<td>Ban <code>Markup()</code></td>
<td><code>python.flask.security.xss.audit.explicit-unescape-with-markup.explicit-unescape-with-markup</code></td>
<td>If needed, review each usage and exempt with <code># nosem</code>.</td>
</tr>
<tr>
<td>2.A</td>
<td>Ban returning values directly from routes</td>
<td><code>python.flask.security.audit.directly-returned-format-string.directly-returned-format-string</code></td>
<td>Use <code>render_template()</code> or <code>jsonify()</code>.</td>
</tr>
<tr>
<td>2.B</td>
<td>Ban using Jinja2 directly</td>
<td><code>python.flask.security.xss.audit.direct-use-of-jinja2.direct-use-of-jinja2</code></td>
<td>Use <code>render_template()</code></td>
</tr>
<tr>
<td>4.A</td>
<td>Flag unquoted HTML attributes with Jinja expressions</td>
<td><code>python.flask.security.xss.audit.template-unquoted-attribute-var.template-unquoted-attribute-var</code></td>
<td>Always use quotes around HTML attributes.</td>
</tr>
<tr>
<td>4.C</td>
<td>Ban template variables in <code>&lt;script&gt;</code> blocks.</td>
<td>N/A</td>
<td>Use the <code>tojson</code> filter inside a data attribute and <code>JSON.parse()</code> in JavaScript.</td>
</tr>
</tbody>
</table>
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 Vasilli 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
Autofix

Make security fixes fast and easy.
Even an imperfect suggestion is better than nothing!

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

- Vim: Semgrep policies in editor plugins
- GitHub: Semgrep policies in GitHub Actions
- Semgrep: Semgrep policies with pre-commit

Policies are managed and deployed via dashboard.
Semgrep Findings Overview over the last 30 days

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

Fix Rate: 76% (45 / 59)

Open Findings Over Time
BONUS: Quietly monitor new policies

Secrets - Notify

1 item
Used on:
no repositories

Secrets - Notify

Integrations

Email

Inline PR Comments

Blocking

Search...
Name
Type

secrets

0 disabled rules

+ 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](http://bit.ly/2021Berman-OWASP-Denver)), focus on dev UX

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

---

**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.html#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

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

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Fortify</th>
<th>Checkmarx</th>
<th>CodeQL</th>
<th>Semgrep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open source</td>
<td>❌</td>
<td>❌</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Open source rules</td>
<td>❌</td>
<td>❌</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Freely test on closed source repos</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✅</td>
</tr>
<tr>
<td>Open source SaaS app</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>
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`

```
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

```java
import java.lang.Runtime;

public class RuntimeExample {

    private 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](https://semgrep.live/clintgibler:java-runtime-exec-try)
Solution: [https://semgrep.live/clintgibler:java-runtime-exec](https://semgrep.live/clintgibler:java-runtime-exec)
Beyond OWASP Top 10: Business Logic

"call verify_transaction() before "make_transaction()"

Try it:  https://semgrep.dev/ievans:make-transaction-try
Solution:  https://semgrep.dev/ievans:make-transaction
Tell me as soon as possible (ideally in editor)

IDE Integration

```python
def get_re_range_matches(
    metavar, regex: PatternId, ranges: PatternId, pattern: PatternId) -> Set[Range]:
    result = Semgrep(pattern)
    metavar = metavar
    regex = regex
    ranges = ranges
    pattern = pattern
    if metavar == metavar:
        logger.debug(f"metavar {metavar} missing in range")
        continue
    any_matching_ranges = any(
        pm.range == range
        and metavar in pm.metavars
        and re.match(regex, pm.metavars[metavar]["abstract_con"]
                    for pm in pattern.match
```
Autofix

Make security fixes fast and easy.

Even an imperfect suggestion is better than nothing!

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

semgrep-dev  bot  1 minute ago

Suggested change

342 - @app.route('/other_unauth', methods = ['GET', 'POST'])
343 - def other_unauth():
344 -     print("Calling other_unauth route")
345 -     return jsonify({'ok': 'some text'}), 204
342 + @app.route('/other_unauth', methods = ['GET', 'POST'])
343 + def other_unauth():
344 +     token = request.headers.get('Authorization')
345 +     if not token:
346 +         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?
Quiz: Does this app have RCE?

- Does the app?
  - Deserialize data
  - Run shell commands
  - Mix data and code
    - eval(), exec()
    - Metaprogramming

What does user control?
Structure of data?
Input filtered?
How is it stored?
(field types, constraints)
Quiz: Does this app have RCE?

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

Does the app?
- Deserialize data
- Run shell commands
- Mix data and code
  - `eval()`, `exec()`
  - Metaprogramming

What does user control?
Structure of data?

How is it stored?
(field types, constraints)

Input filtered?
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

Ban:
- `exec()`, `eval()`, `shell exec`, deserialization (objects, YAML, XML, JSON)
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?
2. Rolling out requires org-wide buy-in
3. Enforcing secure defaults

Docs

- Onboarding
- Coding standards
- Code quality