



Eradicating Vulnerability Classes

by Embracing Secure Defaults and Invariants

Clint Gibler | r2c.dev

 [@clintgibler](https://twitter.com/clintgibler)

Slides: <https://go.r2c.dev/bssg2020>



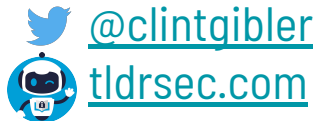
🤔 A Different Way to Approach Security

- Killing **bug classes**: scalable, systematic, long-term wins
- Enabling developers to move **fast** *and* **securely**
 - Security team as business **enablers**, not another point of friction
- **How** to do this at your company
 - Using **free**, open source tools

who is?

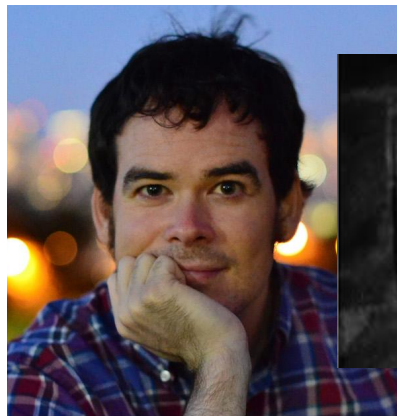
me:

Clint Gibler, Head of Security Research @ r2c
Formerly: NCC Group, UC Davis PhD



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
4. Security + Framework Collaborations

1. Why Bug-Finding Isn't The Answer

Software Development has Changed

...thus Security Teams must too

In many companies:

- Security teams can hard block engineering rarely, if ever
- Security testing must be continuous, not point in time
- Focus on building, not just breaking
- Embedded or partnered closely with dev teams

We need to re-visit our prior assumptions

Massive Shifts in Tech and Security

Waterfall development

Dev, Ops

On prem

Agile development

DevOps

Cloud



Before



After

Massive Shifts in Tech and Security

Waterfall development

Dev, Ops

On prem

Finding vulnerabilities

Agile development

DevOps

Cloud

Secure defaults and invariants



Before



After

Invariant

A property that must either
always or
never be true

Key Insight

No context needed to
make a decision

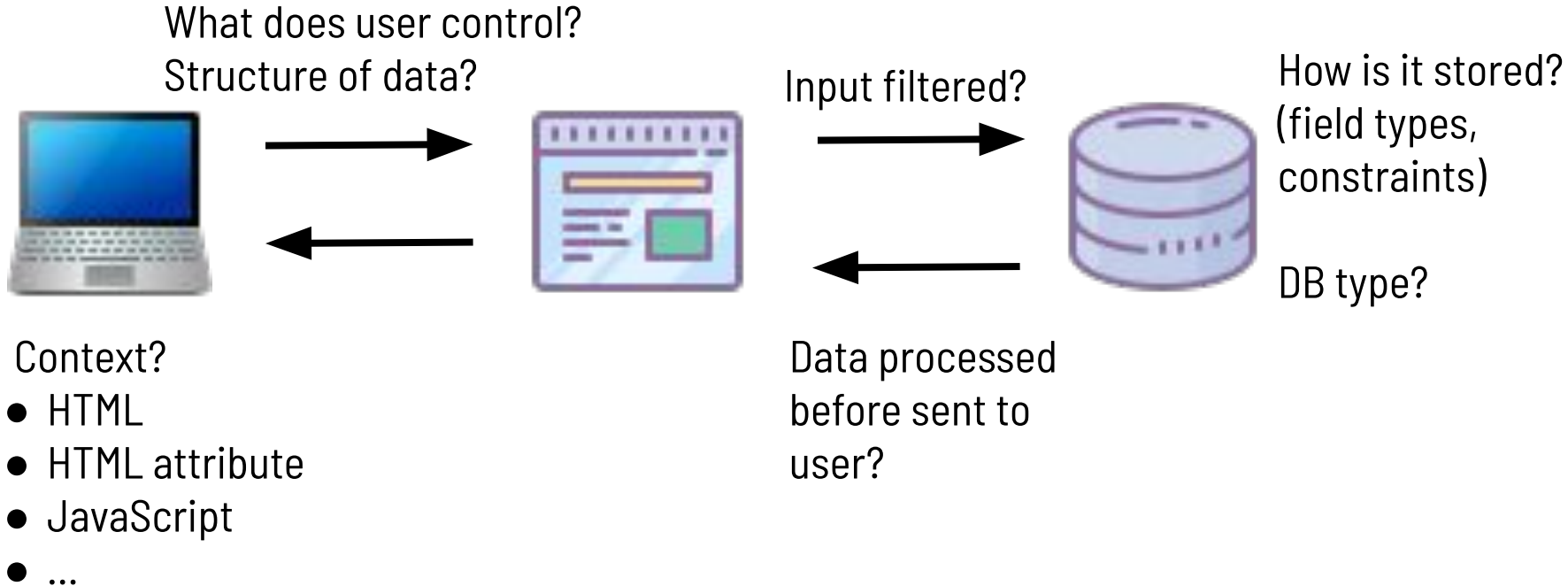
==

No operational time for
the security team

Quiz: Does this app have XSS?

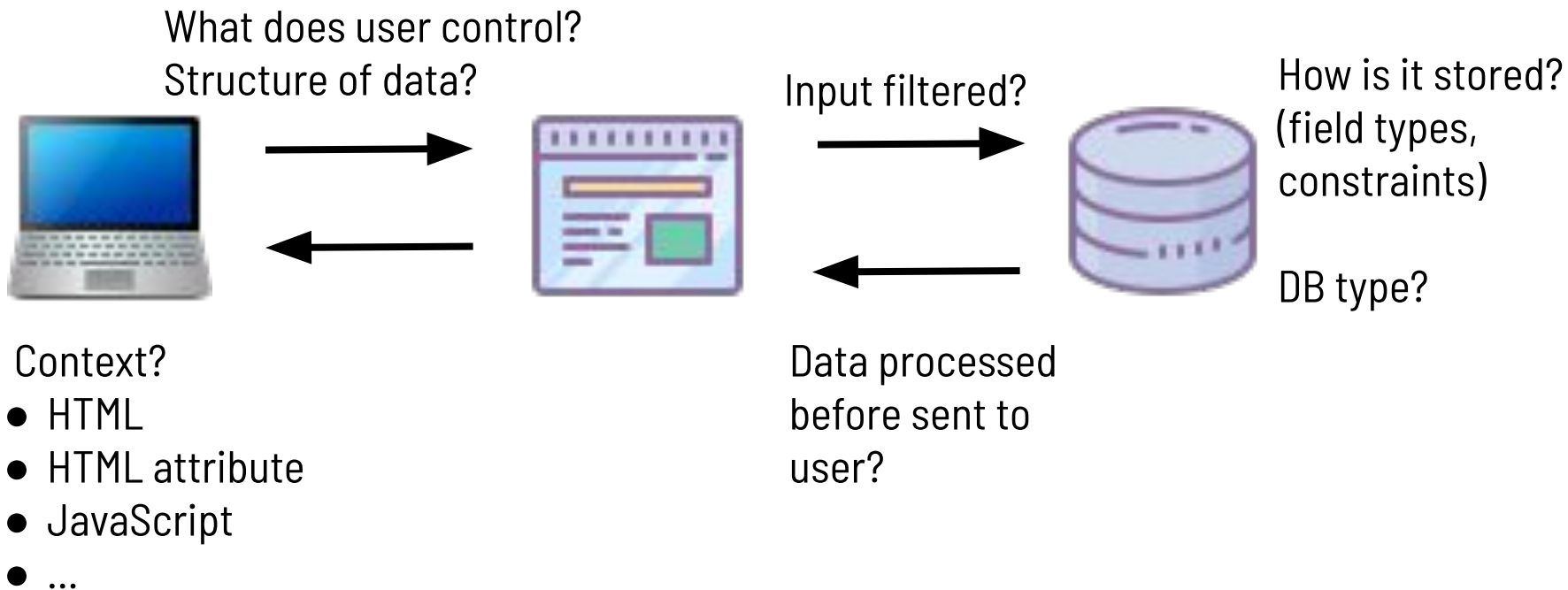


Quiz: Does this app have XSS?



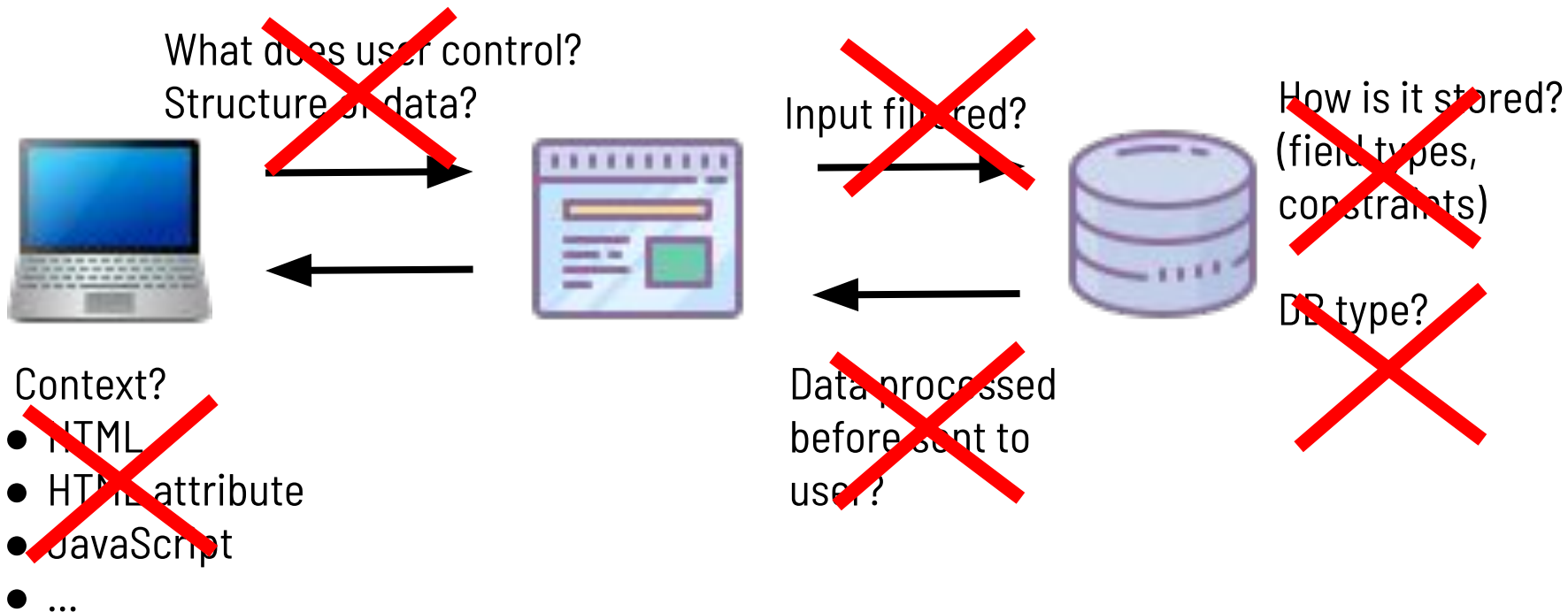
Quiz: Does this app have XSS?

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



Quiz: Does this app have XSS?

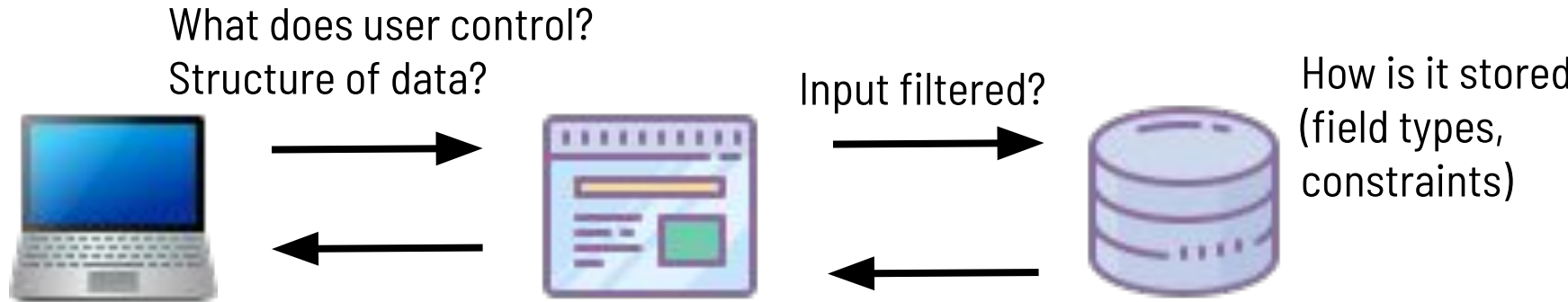
*Invariant: Frontend is **React**, banned **dangerouslySetInnerHTML***



Quiz: Does this app have RCE?



Quiz: Does this app have RCE?

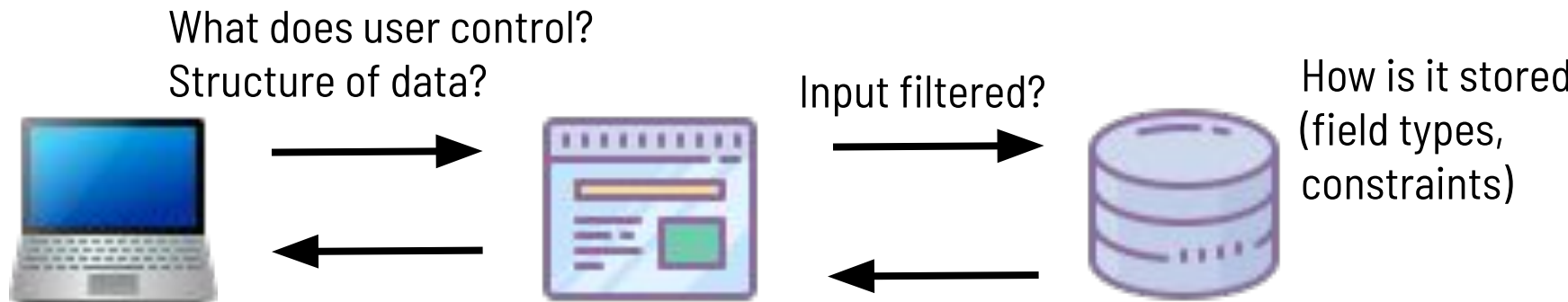


Does the app?

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

Quiz: Does this app have RCE?

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

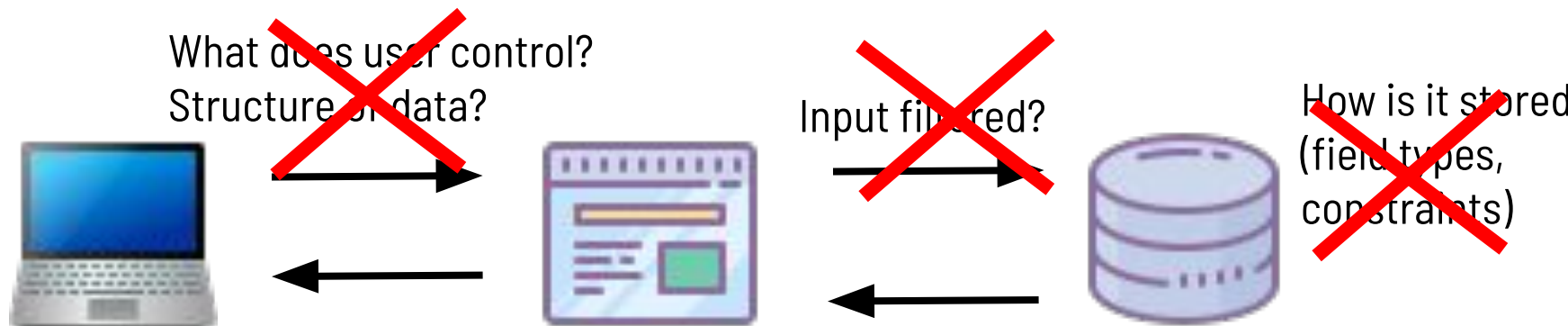


Does the app?

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

Quiz: Does this app have RCE?

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



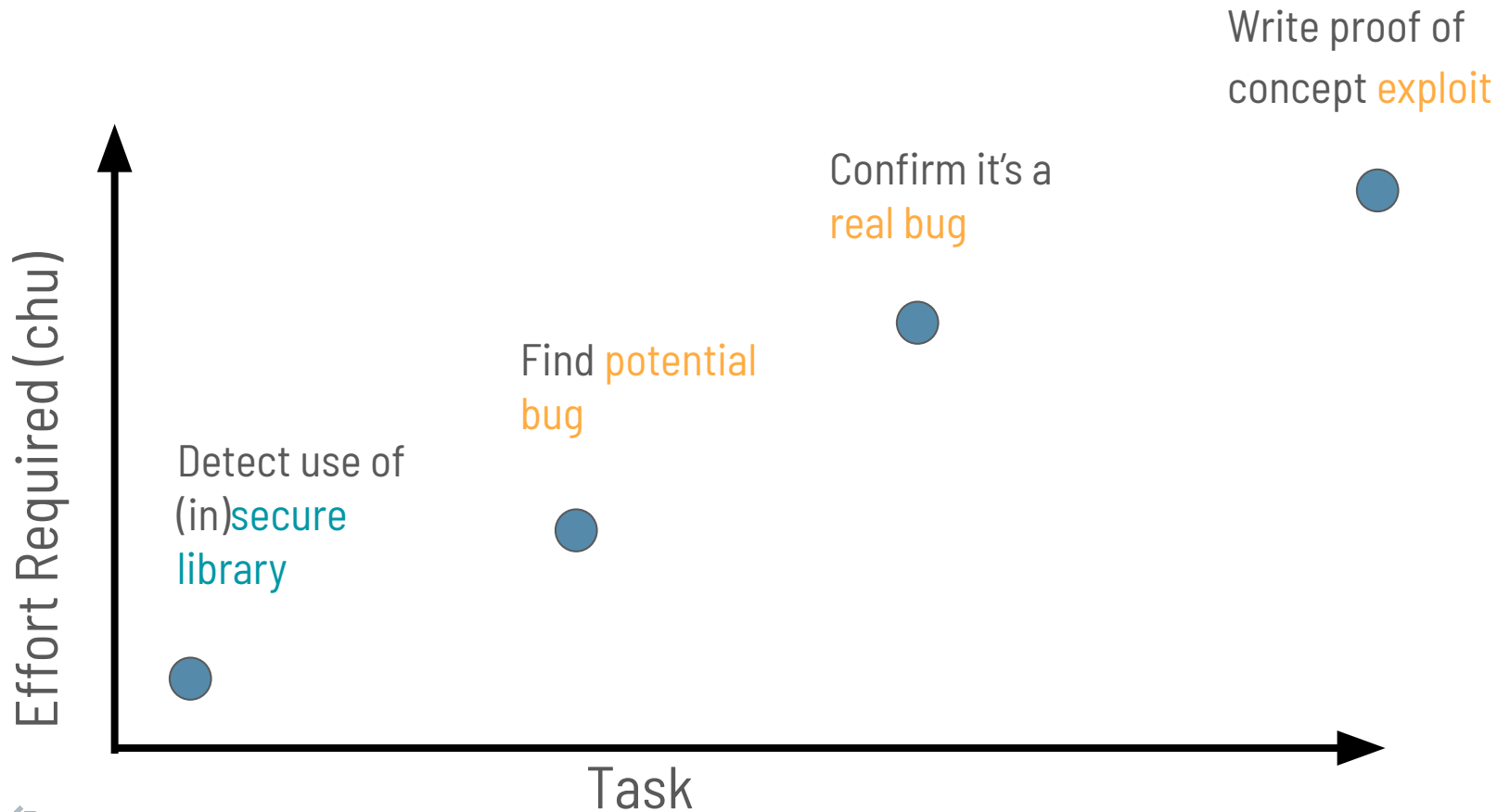
Does the app?

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

Let's Solve the “Easy” Version of the Problem

- These apps could have been incredibly complex, with millions of LOC
- With some strong **invariants**, we significantly reduced their **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**



#Broke

Finding every vulnerability



#Woke

Preventing **classes** of vulnerabilities

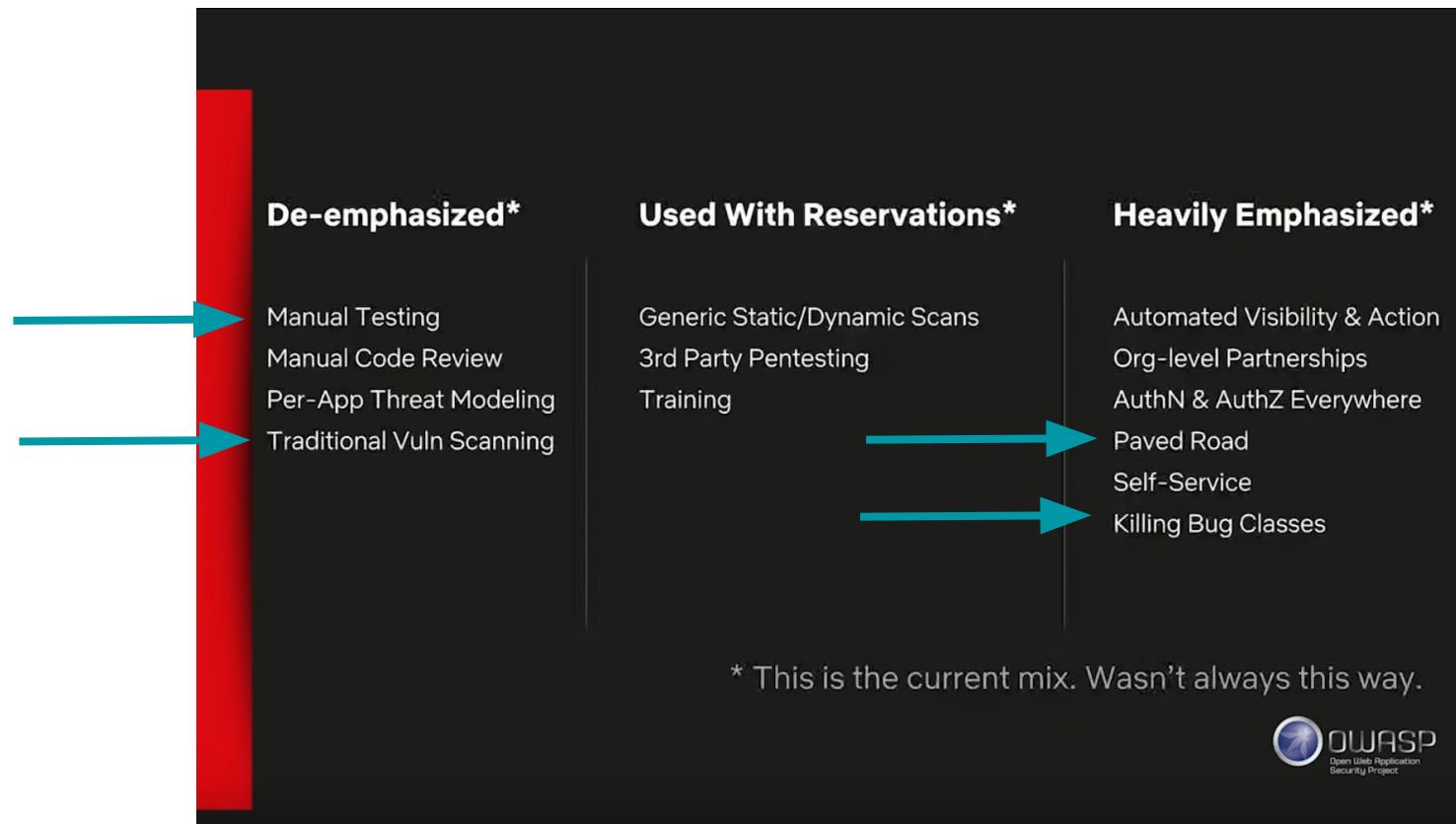
Your Internal Dialogue?

- “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

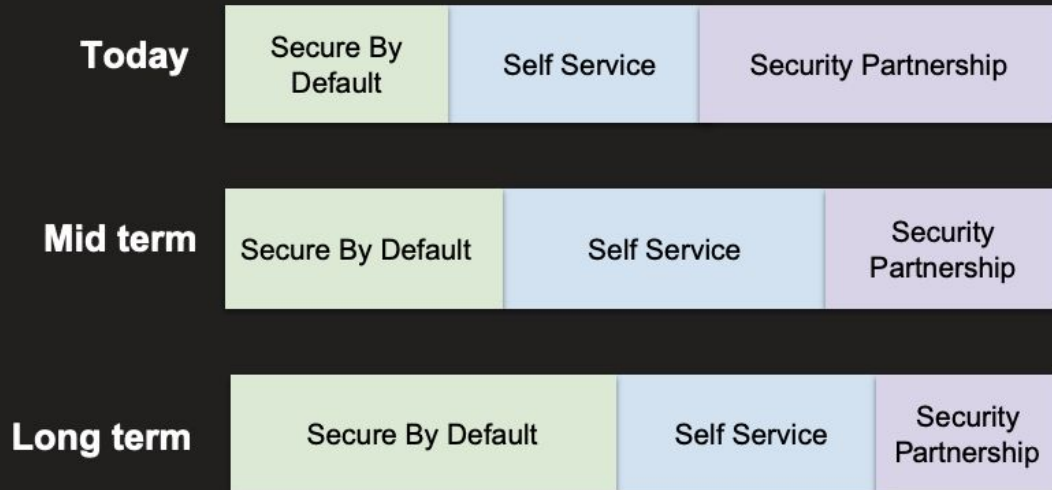
AppSec Cali '18, [Patrick Thomas](#), [Astha Singhal](#)



A Pragmatic Approach for Internal Security Partnerships

AppSec Cali '19, [Scott Behrens](#), [Esha Kanekar](#)

How is the future shaping up for us?



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

- ## 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)

[illegible]

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**

Tools and Training Help, but are Not Enough



We need a safer systems programming language

Security Research & Defense / By MSRC Team / July 18, 2019 / Memory Safety, Rust, Safe Systems Programming Languages, Secure Development

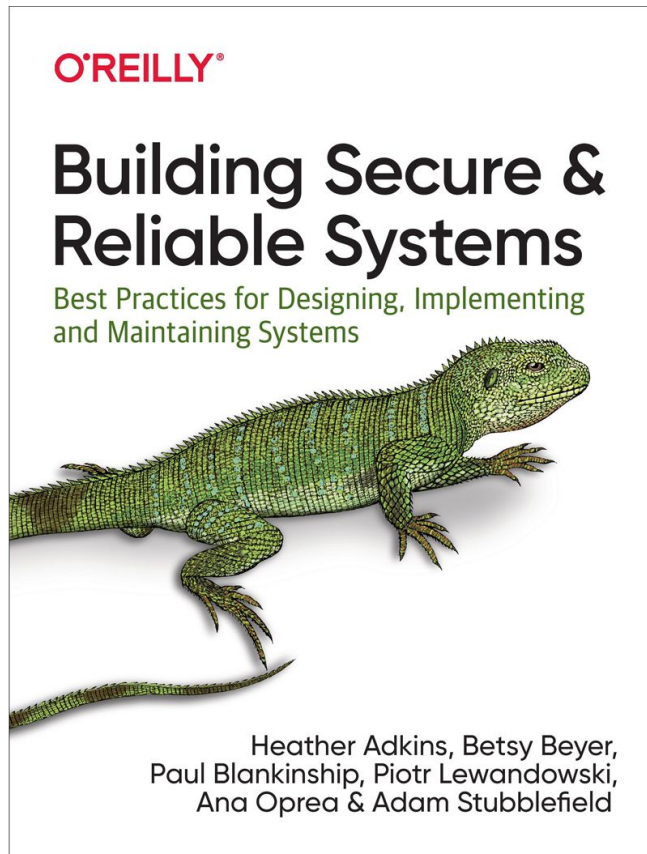
From the Microsoft Security Response Center [blog](#):

- “**Tools** and **guidance** are **demonstrably not preventing** this class of vulnerabilities; memory safety issues have represented **almost the same proportion of vulnerabilities** assigned a CVE **for over a decade**.”

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



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



“But I’m not Google”

Framework / tech choices **matter**

- Mitigate classes of vulnerabilities

Examples:

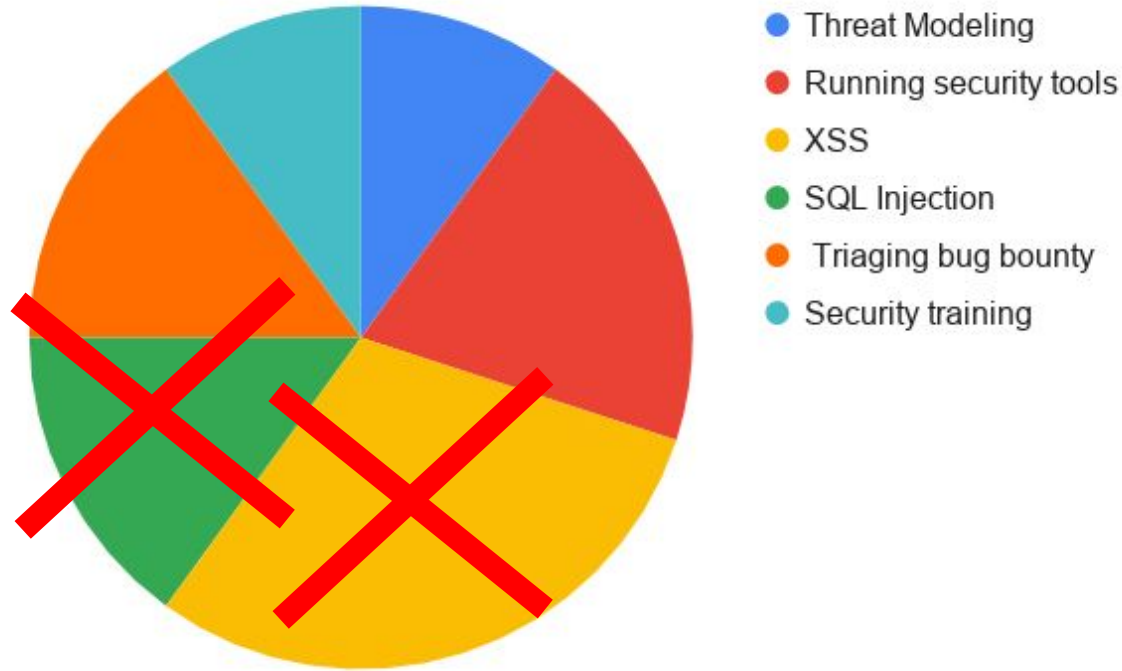
- Using modern web frameworks
- [DOMPurify](#) - output encoding
- [re2](#) - regexes
- [tink](#) - crypto

*Web security before
modern frameworks*



2. How to Eradicate Vulnerability Classes

Compounding Effects of Killing Bug Classes



How to Eradicate Vulnerability Classes

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



IaaS 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. Train Developers to 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**

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

salesforce

[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

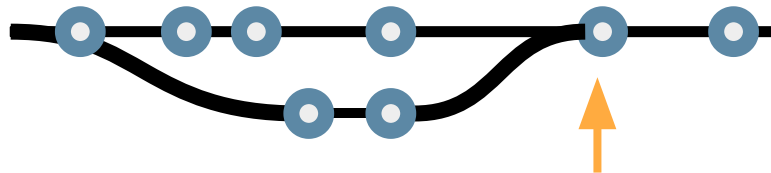
can do longer / more in depth scans daily or weekly



Two Scanning Workflows


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

Make Adjustment Easy

Make it cheap to add/remove tools and new rules



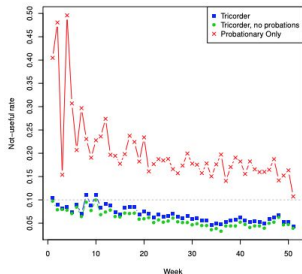
✓	All checks have passed 13 successful checks
✓	 Security Scan Successful in 33 days
✓	 Lint / pre-commit (pull_request) Successful in 12s



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

Continuously Finding: Escape Hatches

If we use secure frameworks that maintain invariants, all we need to do is **detect the functions that let you "escape" from those invariants**. 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?**

Semgrep tl;dr

- Originally developed at Facebook to encourage secure code patterns!
- A customizable, lightweight, static analysis tool for finding bugs
- Batteries included, with hundreds of existing community rules
- Combine the speed + customization of grep with the expressiveness of SAST
- Runs offline, on uncompiled code, fast and [open source](#)!
- No painful DSL, patterns look like the source code you're targeting

 [returntocorp](#) / [semgrep](#)  LGPL-2.1 License

Python	Go	Java	JavaScript	JSON	Ruby (beta)	C (alpha)	OCaml (alpha)
--------	----	------	------------	------	-------------	-----------	---------------

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
```

```
1  import exec as safe_function
2  safe_function(user_input)
3
4  exec("ls")
5
6  exec(some_var)
7
8  some_exec(foo)
9
10 exec (foo)
11
12 exec (
13     bar
14 )
15
16 # exec(foo)
17
18 print("exec(bar)")
```

Try it: <https://semgrep.dev/ievans:python-exec>

Secure defaults + types

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

```
1  import java.lang.Runtime;
2
3  public class RuntimeExample {
4
5      public void foo(Runtime arg) {
6          Runtime rt = Runtime.getRuntime();
7          rt.exec("ls");
8
9          arg.exec("rm /");
10
11          Other other = new Other();
12          other.exec("wrong exec");
13      }
14  }
15
```

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)

```
25 from semgrep.semgrep_types import pattern_names_for_operator
26 from semgrep.semgrep_types import PatternId
27 from semgrep.semgrep_types import Range
28 from semgrep.semgrep_types import TAINT_MODE
29 from semgrep.util import flatten
30
31
32 def get_re_range_matches(
33     metavar
34     regex:
35     ranges:
36     pattern
37 ) -> Set[Range]
38
39     result:
40     for _ra
41     if metavariable == metavariable:
42         logger.debug(f"metavariable '{metavariable}' missing in range")
43         continue
44
45     any_matching_ranges = any(
46         pm.range == _range
47         and metavariable in pm.metavars
48         and re.match(regex, pm.metavars[metavariable])["abstract_con
49     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)

Peek Problem (^X) Checking for quick fixes...

You, a few seconds ago



Autofix

Make security fixes fast and easy.

Even an imperfect suggestion is better than nothing!

SEMGREP RULE

Simple

Advanced

code is

TODO

Suggest +

and autofix is

TODO

Suggest

TEST CODE

```
1 import sys
2
3 def check_db(user):
4     if user is None:
5         exit(4)
6     else:
```

Run ↵

MATCHES

No Matches

Try tweaking your pattern or code or open a bug if you feel there should be a match. If you are composing multiple patterns, try enabling step-by-step debugging under "Advanced Options" next to the Run button.

The [config documentation](#) has more details on the syntax and boolean logic. Here are some common issues:

Try: <https://semgrep.dev/ievans:tlsautofix>

4. Security + Framework Collaboration

Solving OWASP Top 10 with Frameworks

Google maps every OWASP Top 10 vulnerability to a framework hardening measure

[*Building Secure & Reliable Systems*](#) by Google

OWASP top 10 vulnerability	Framework hardening measures
[SQL] Injection	TrustedSQLString (see the following section).
Broken authentication	Require authentication using a well-tested mechanism like OAuth before routing a request to the application. (See “Example: Framework for RPC Backends” on page 247.)
Sensitive data exposure	Use distinct types (instead of strings) to store and handle sensitive data like credit card numbers. This approach can restrict serialization to prevent leaks and enforce appropriate encryption. Frameworks can additionally enforce transparent in-transit protection, like HTTPS with LetsEncrypt. Cryptographic APIs such as Tink can encourage appropriate secret storage, such as loading keys from a cloud key management system instead of a configuration file.
XML external entities (XXE)	Use an XML parser without XXE enabled; ensure this risky feature is disabled in libraries that support it. ⁹
Broken access control	This is a tricky problem, because it’s often application-specific. Use a framework that requires every request handler or RPC to have well-defined access control restrictions. If possible, pass end-user credentials to the backend, and enforce an access control policy in the backend.
Security misconfiguration	Use a technology stack that provides secure configurations by default and restricts or doesn’t allow risky configuration options. For example, use a web framework that does not print error information in production. Use a single flag to enable all debug features, and set up your deployment and monitoring infrastructure to ensure this flag is not enabled for public users. The <code>environment</code> flag in Rails is one example of this approach.
Cross-site scripting (XSS)	Use an XSS-hardened template system (see “Preventing XSS: SafeHtml” on page 254).
Insecure deserialization	Use deserialization libraries that are built for handling untrusted inputs, such as Protocol Buffers .
Using components with known vulnerabilities	Choose libraries that are popular and actively maintained. Do not pick components that have a history of unfixed or slowly fixed security issues. Also see “Lessons for Evaluating and Building Frameworks” on page 256.
Insufficient logging & monitoring	Instead of relying on ad hoc logging, log and monitor requests and other events as appropriate in a low-level library. See the logging interceptor described in the previous section for an example.

Partnering with OWASP

- New 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

 [semgrep / rules-owasp-asvs](#)

 [OWASP / CheatSheetSeries](#)

<> Code

! Issues 27

🔗 Pull requests 9

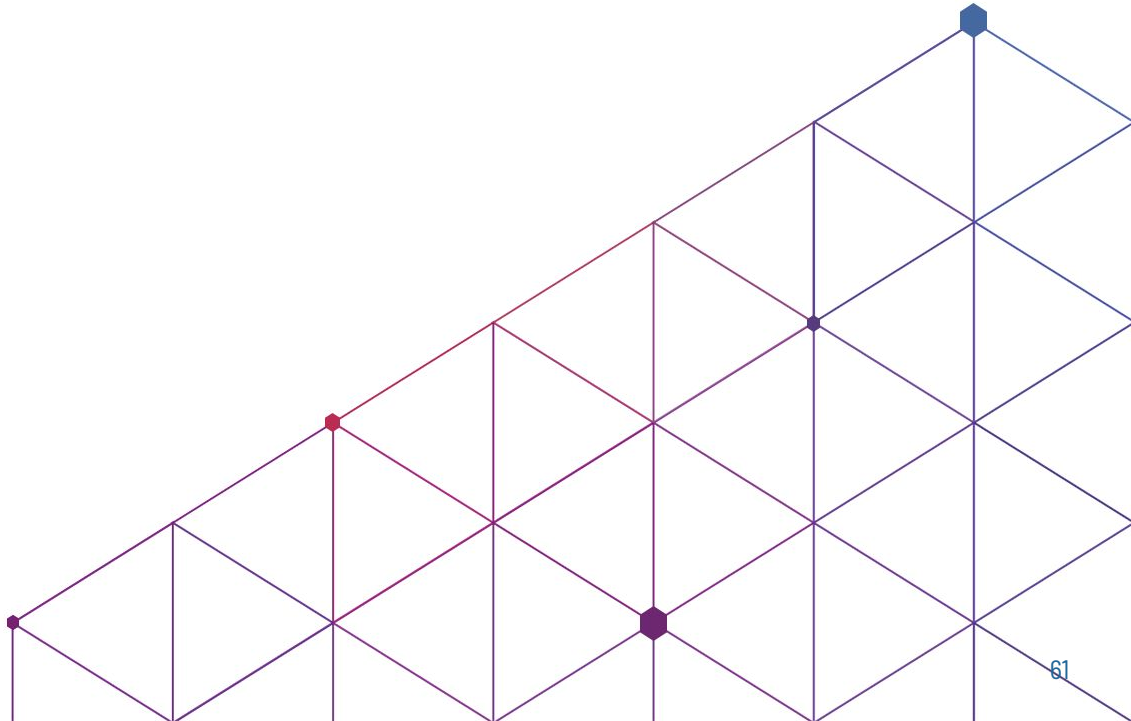
▶ Actions

📁 Projects 1

Update: Adding Semgrep Rules #457



Wrapping Up





How to 10X Your Security (Without the Series D)



Clint Gibler

 [@clintgibler](https://twitter.com/clintgibler)



Watch on Youtube



Tl;dr sec Newsletter - <https://tldrsec.com>

Talk **summaries** | **Tools & Resources** links | **Original research**



Caleb Sima • 1st



VP - Security at Databricks : Hiring in all positions - Review my experienc...
20h

As a busy exec who's heart is still deep in tech. I have found it almost impossible anymore to keep up with latest good tools/talks in infosec. I have to give a shoutout to **Clint Gibler** 's newsletter tldr;sec which gives me a weekly email that is a curated view of the best stuff. It is absolute gold - keep up the good work Clint. I highly recommend people sign up:

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: <https://go.r2c.dev/bssg2020>

Clint Gibler

 [@clintgibler](#) | [LinkedIn](#)
 [tldrsec.com](#)

