



# Semgrep

A polyglot customizable  
bug-finding tool

Yoann Padioleau | pad@r2c.dev

 [@r2cdev](https://twitter.com/r2cdev)

Strumenta Meetup, January 2023

# who is?

**me:**

Yoann Padioleau, software engineer @ r2c

ex-Facebook dev (started Test Engineering, AppSec, and Program Analysis teams),  
ex-academia (coccinelle)

**r2c**

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



# tl;dr

- Semgrep is 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  Star **1.1k**

Python	JavaScript	Go	Java	C	JSON	Ruby	OCaml	TypeScript	PHP
								Coming...	Coming...

# Outline

1. **Background** - `grep` and Abstract Syntax Trees (ASTs)
2. **Demo** - How do I use it?
3. **Ecosystem**: Registry, CI/CD, WebApp
4. **Language Engineering**: tree-sitter, generic AST, parsing/naming

# `grep` and Abstract Syntax Trees (ASTs)

# grep, ASTs, and Semgrep

```
exec("ls")

exec(some_var)

exec (arg)

exec(
    bar
)

other_exec(foo)

// exec(foo)

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

✓ Easy - `exec\ (`

✓ Easy - `exec\ (`

⚠ Handle whitespace `exec\s*\ (`

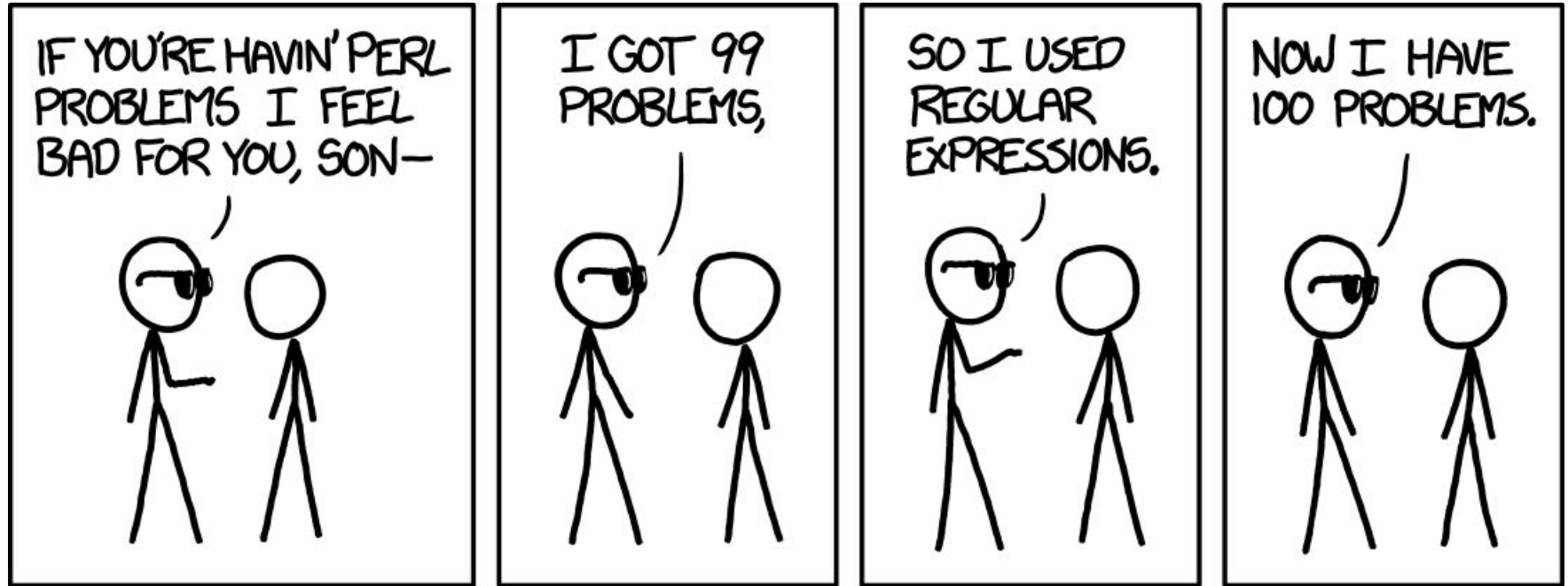
⚠ 😊 Handle whitespace/newlines

🛑 😊😊 Method suffix matches `exec`

🛑 😊😊 Is this a comment?

🛑 😊😊 Is this a string literal?

# xkcd 1171



# Code is not a string, it's a tree



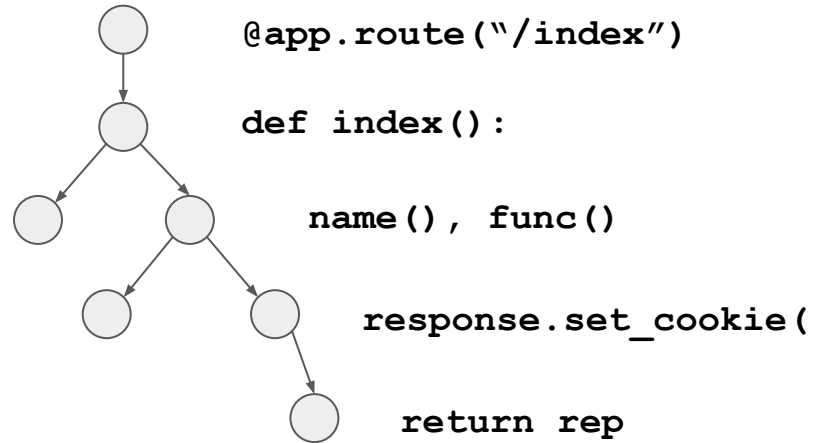
**string**

```
@app.route("/index")
def index():
    rep = response.set_cookie(name(),
secure=False, s=func())
    return rep
```

**!=**



**tree**

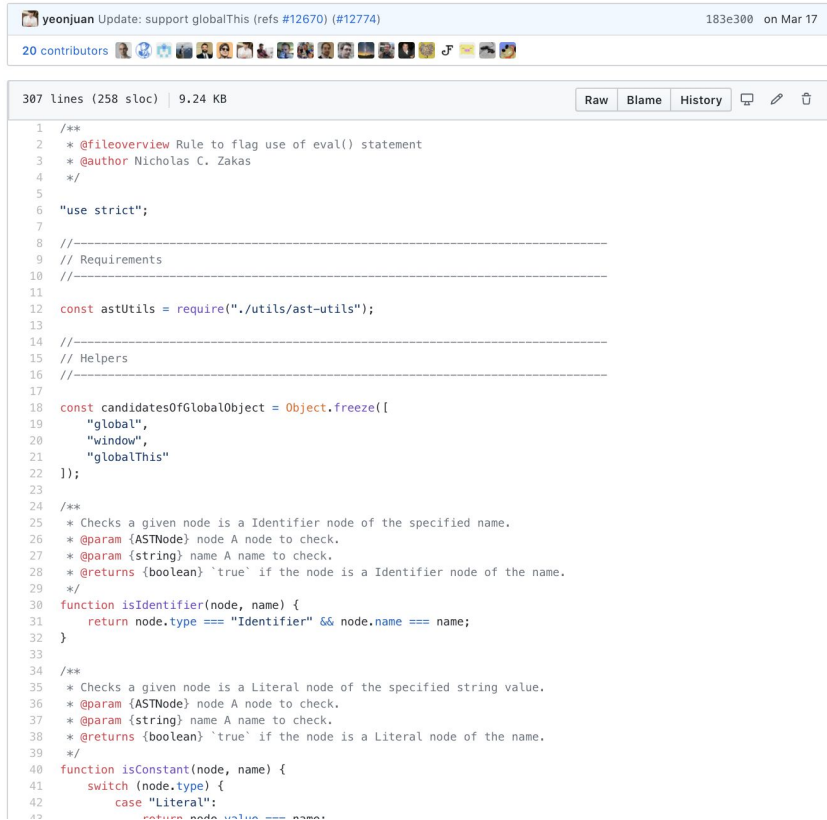




# Tree Matching

- Many tree matching tools: Bandit, Dlint, ESLint, Flake8, Golint, Gosec, Pylint, RuboCop, TSLint, and more!
- Have to become an **expert in every AST syntax** for every language your team uses
- Need **programming language expertise** to cover all idioms: languages have “more than one way to do it”
- **Commercial SAST tools?**
  - Complicated
  - Slow (not CI friendly)
  - Expensive

Find calls to `eval()`  
in only 307 LOC 👍



```
1 /**
2  * @fileoverview Rule to flag use of eval() statement
3  * @author Nicholas C. Zakas
4  */
5
6 "use strict";
7
8 -----
9 // Requirements
10 -----
11
12 const astUtils = require("../utils/ast-utils");
13
14 -----
15 // Helpers
16 -----
17
18 const candidatesOfGlobalObject = Object.freeze([
19   "global",
20   "window",
21   "globalThis"
22 ]);
23
24 /**
25  * Checks a given node is a Identifier node of the specified name.
26  * @param {ASTNode} node A node to check.
27  * @param {string} name A name to check.
28  * @returns {boolean} `true` if the node is a Identifier node of the name.
29  */
30 function isIdentifier(node, name) {
31   return node.type === "Identifier" && node.name === name;
32 }
33
34 /**
35  * Checks a given node is a Literal node of the specified string value.
36  * @param {ASTNode} node A node to check.
37  * @param {string} name A name to check.
38  * @returns {boolean} `true` if the node is a Literal node of the name.
39  */
40 function isConstant(node, name) {
41   switch (node.type) {
42     case "Literal":
43     
```

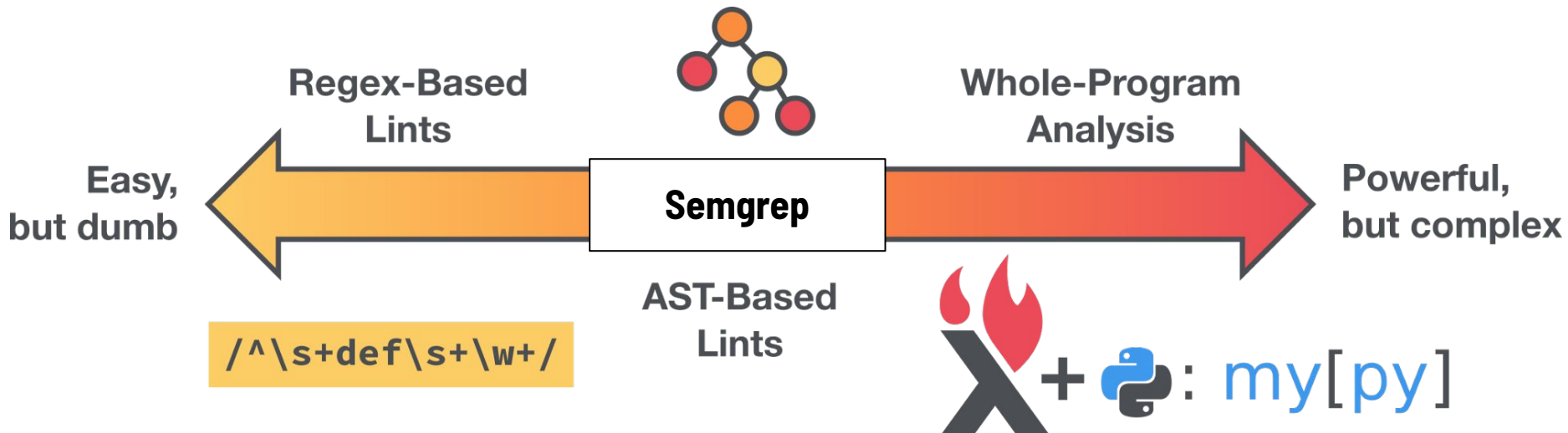
<https://github.com/eslint/eslint/blob/master/lib/rules/no-eval.js>

# Static Analysis at Scale: An Instagram Story



Benjamin Woodruff [Follow](#)

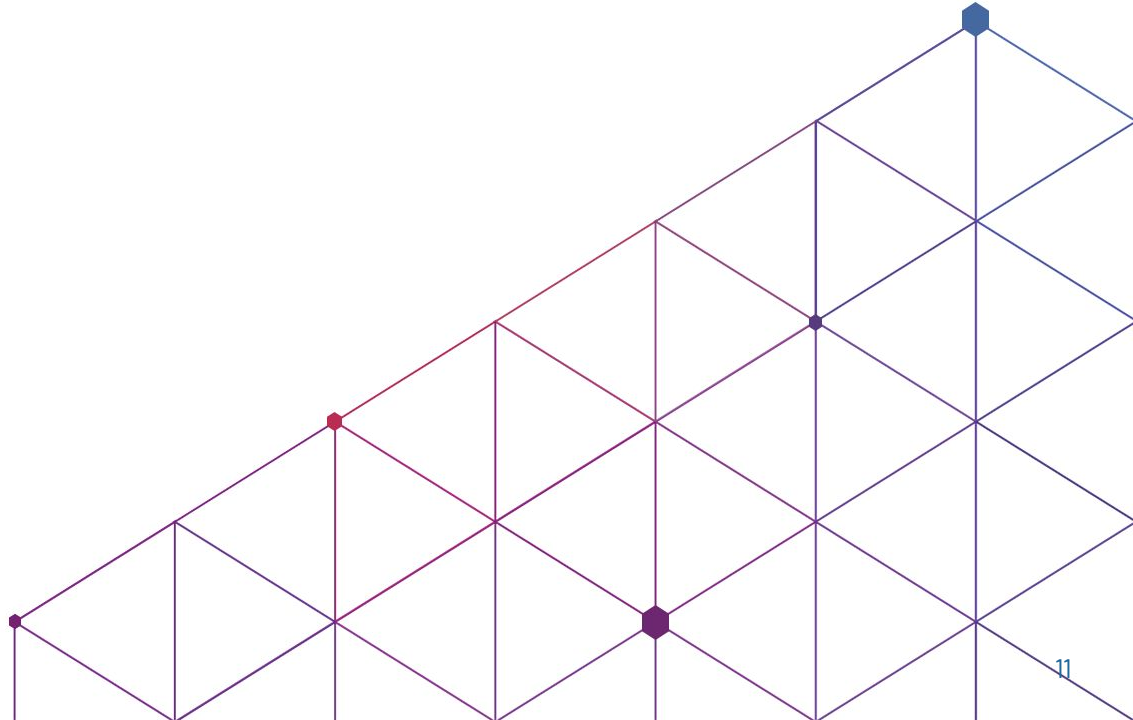
Aug 15, 2019 · 13 min read



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

# Demo

1. Ellipsis ("...") operator
2. Metavariables
3. Advanced Features



# Finding Banned, Deprecated, or Dangerous Functions

```
exec("ls")
```

⇒ <https://semgrep.dev/s/Ew0P>

Full Solution: <https://semgrep.dev/s/7KGk>

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

# Hard-coded Secrets, Constant String Arguments

```
s4 = boto3.client(  
    's3',  
    aws_secret_access_key = "jWnyeKHgaSRZVdX7ZQRATpoCkEsvPLRKNZCYRXRL",  
    aws_access_key_id = "AKIAIOSFODNN7652GQNB")
```

⇒ <https://semgrep.dev/s/RG08/>

Full Solution: <https://semgrep.dev/s/A89w/>

# Semantic Equivalences

Semgrep (Python) patterns

```
foo(kwd1=1,  
    kwd2=2,  
    ...)
```

Will  
match

```
subprocess.open(...)
```

Will  
match

```
foo(1)
```

Will  
match

Target (Python) code

```
foo(kwd2=2,  
    kwd1=1,  
    kwd3=3)
```

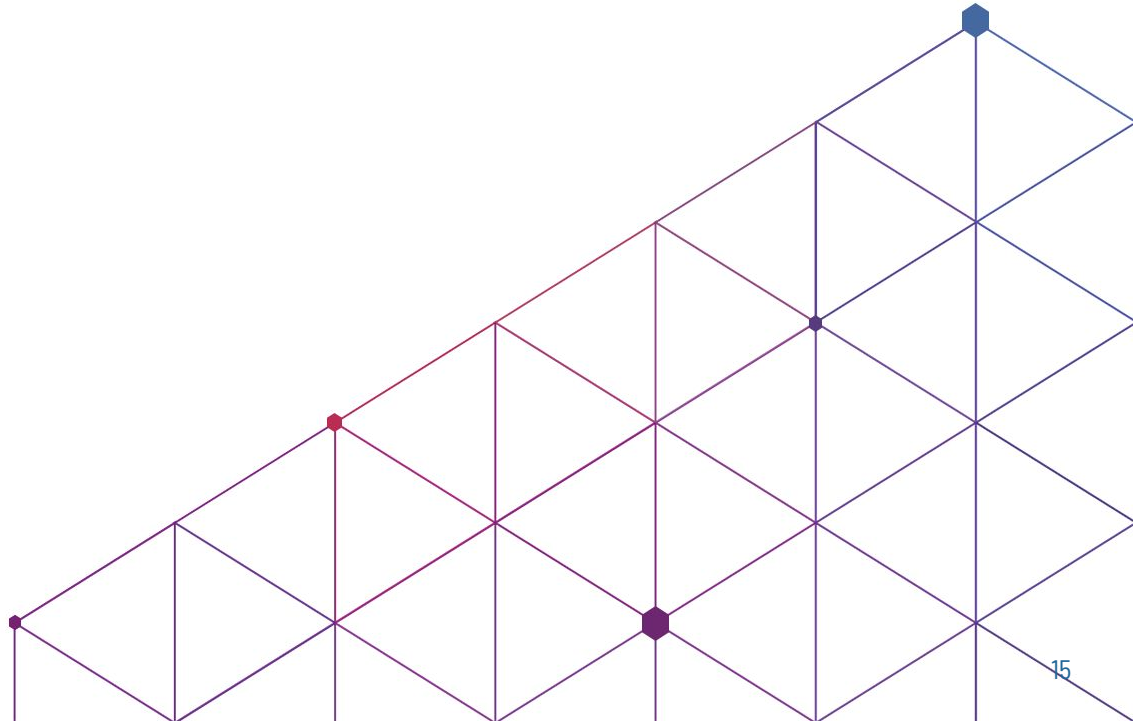
```
from subprocess import  
open as sub_open  
res = sub_open("ls")
```

```
x = 1  
bar()  
foo(x)
```

- Semgrep knows about the semantic of the languages
  - Keyword arguments ordering
  - Import aliasing
  - Constant propagation
  - Symbolic expressions propagation
  - Dataflow-based taint propagation
  - associative/commutative operations
  - ...

# Demo

1. Ellipsis ("...") operator
2. Metavariables
3. Advanced Features



## Matching Comparisons with **Metavariables**

```
5 == 5
```

```
7 == 8
```

```
if "cat" == "cat":  
    print("equal pets")
```

⇒ <https://semgrep.dev/s/61o>

Full Solution: <https://semgrep.dev/s/oB9>



# Order of API Calls Must be Enforced

```
/*
 * A financial trading application in which every
 * transaction MUST be verified (verify_transaction())
 * before it is made (make_transaction())
 */
public class TransactExample {
    public void base_ok(Transaction t) {
        // OK: verify called before make
        verify_transaction(t);
        make_transaction(t);
    }

    public void no_verify(Transaction t) {
        // BAD: transaction isn't verified
        make_transaction(t);
    }
}
```

<https://semgrep.dev/s/LNX>



# The Rule

```
- id: egeq-is-bad
  patterns:
    - pattern-not-inside: |
      def __eq__(...):
        ...
    - pattern-not-inside: assert(...)
    - pattern-not-inside: assertTrue(...)
    - pattern-not-inside: assertFalse(...)
    - pattern-either:
      - pattern: $X == $X
      - pattern: $X != $X
    - pattern-not: 1 == 1
  message: "useless comparison operation"
  languages: [python]
  severity: ERROR
```

- Boolean composition of patterns
- ID, Message, severity, etc.
- More features:
  - Analyze embedded languages
  - Tainting mode
  - Metavariable comparisons
  - ...

```
$ semgrep --config 'myrules.yml' /my/project
```

## Autofix - Use TLS

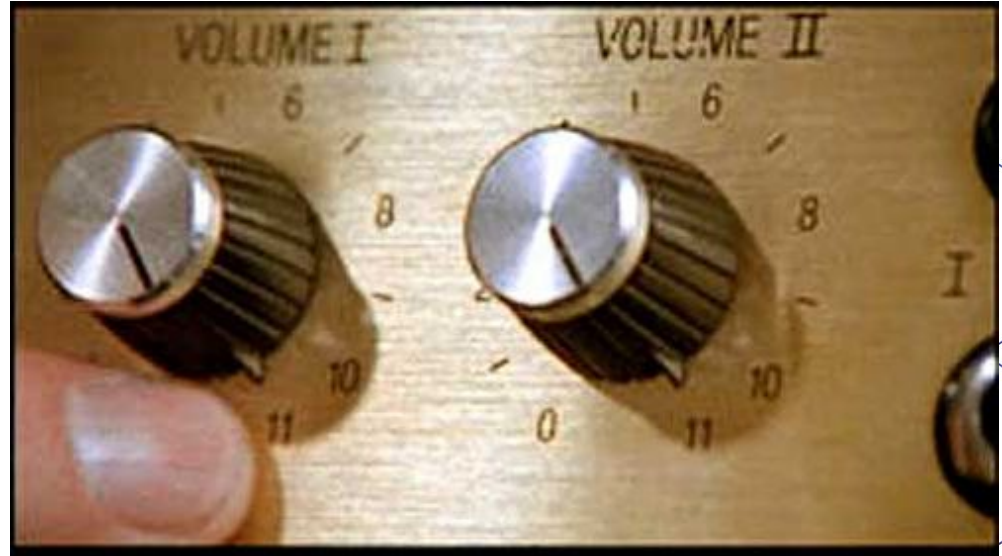
```
func main() {  
    http.HandleFunc("/index", Handler)  
    // ruleid: use-tls  
    http.ListenAndServe(":80", nil)  
}
```

<https://semgrep.live/clintgibler:use-listenAndServeTLS-try>

Solution: <https://semgrep.dev/s/clintgibler:use-listenAndServeTLS>

# Tutorials

1. Ellipsis ("...") operator
2. Metavariables
3. **Advanced Features**



# Combining Semgrep with Regex - pattern-regex

```
# Internal network
```

```
boto3.client(host="192.168.1.125")
```

```
# Okay
```

```
boto3.client(host="https://bucket.s3.amazonaws.com")
```

```
patterns:
```

- pattern-inside: boto3.client(host="...")
- pattern-regex: '192.168\.\d{1,3}\.\d{1,3}'

<https://semgrep.live/clintgibler:boto3-host-regex-try>

Solution: <https://semgrep.live/clintgibler:boto3-host-regex>

# Typed Patterns - Find calls to `exec()` on `Runtime` objects

```
public void foo(Runtime arg) {  
    Runtime rt = Runtime.getRuntime();  
    rt.exec("ls");  
  
    arg.exec("rm /");  
  
    Other other = new Other();  
    other.exec("wrong exec");  
}
```

```
pattern: |  
| (Runtime $CLASS).exec(...);
```

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

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

# Taint Analysis

```
def foo():  
    a = source1()  
    b = sanitize(a)  
    sink1(b)  
    sink(b)
```

```
def bar():  
    a = source1()  
    sanitize()  
    eval(a)  
    sink(a)
```

```
pattern-sources:  
- source(...)  
- source1(...)  
pattern-sinks:  
- sink(...)  
- sink1(...)  
- eval(...)  
pattern-sanitizers:  
- sanitize(...)  
- sanitize1(...)
```

Try it: <https://semgrep.live/ievans:tainting>

# JSON

```
{
  "Version": "2012-10-17",
  "Id": "S3-Account-Permissions",
  "Statement": [{
    "Sid": "1",
    "Effect": "Allow",
    "Principal": {"AWS": ["arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:root"]},
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3::mybucket",
      "arn:aws:s3::mybucket/*"
    ]
  }]
}
```

```
{ "Id": "S3-Account-Permissions",
  "Statement": [
    { "Effect": "Allow",
      Resource: [..., "=~/arn.*/", ... ]
    } /* notice non-quoted literal Resource, and use of internal regex */
  ]
}
```

Try it: <https://semgrep.live/clintgibler:s3-account-permissions-try>  
Solution: <https://semgrep.live/clintgibler:s3-account-permissions>





# Semgrep Ecosystem



# The Registry and the Ruleset

A screenshot of the Semgrep Registry search results for the query 'java eqeq'. The page shows a search bar with the query, a 'Use in CI' button, and filter dropdowns for Language, Category, and Technology. Below the filters, it indicates 'Rules (7)'. The first rule shown is 'java.lang.correctness.eqeq.eqeq' with an 'error' severity. The description for this rule is: '\$X == \$X` or `\$X != \$X` is always true. (Unless the value compare...'. The second rule shown is 'java.lang.correctness.no-string-eqeq.no-string-eqeq' with a 'warning' severity. The description for this rule is: 'Strings should not be compared with `==`! This is a reference compa...

A screenshot of the Semgrep Registry 'Explore' page. The page features a 'Popular' section with a red line graph icon and the text: 'Check out [ci](#), the most popular ruleset'. Below this is a 'Getting Started' section with the text: 'These rulesets cover a wide range of use cases. Start here to get up...'. At the bottom, there are two featured rulesets: 'owasp-top-ten' with icons for Go, Java, JavaScript, Python, Ruby, and TypeScript, and 'r2c-ci' with icons for Go, Java, JavaScript, Python, and Ruby. The description for 'owasp-top-ten' is 'The OWASP Top 10 is an...' and for 'r2c-ci' is 'Scan for runtime errors,'.

```
$ semgrep --config 'p/owasp-top-ten' /my/project
```



# Integrations

- Enforce secure defaults + secure frameworks at CI time
  - Easy to add to CI as either a Docker container or Linux binary
  - JSON output

Use in CI

[gitpre-commit](#) [GitHub](#) [GitLab](#) [CircleCI](#) [AppVeyor](#) [Travis](#)

Add this snippet in your `.github/workflows/semgrep.yml`:

```
name: Semgrep
on: [push, pull_request]
jobs:
  semgrep:
    runs-on: ubuntu-latest
    name: Check
    steps:
      - uses: actions/checkout@master
```

Show Advanced ➔

# Pull Request (PR) comments and Autofix

The screenshot shows a GitHub Pull Request interface. At the top, the browser address bar displays `github.com/returntocorp/semgrep/pull/4944/files`. The page title is "Misc comments #4944" with a "Review changes" button. A file filter is set to "File filter" and "Conversations" is visible. The left sidebar shows a file tree with folders like "semgrep-core/src", "core/il", "naming", and "semgrep/semgrep", and files like "IL.ml", "Naming\_AST.ml", and "core\_runner.py". The main content area shows a diff for `semgrep/semgrep/core_runner.py`. The diff highlights a change on line 250: `subprocess.call("touch /tmp")` is being replaced with `subprocess.check_call(...)`. Below the diff, a comment from the bot "semgrep-app" states: "This is not checking the return value of this subprocess call; if it fails no exception will be raised. Consider subprocess.check\_call() instead". A "Suggested change" box shows the proposed code change. At the bottom, there is a "Not helpful" button and a note: "This finding does not block your pull request." and "From python.lang.correctness.unchecked-returns.unchecked-subprocess-call."

```
es.  
id: unchecked-subprocess-call  
patterns:  
- pattern: subprocess.call(...)  
- pattern-not-inside: $$ = subprocess.c  
- pattern-not-inside: subprocess.call(  
- pattern-not-inside: return subprocess  
fix: subprocess.check_call(...)  
message: This is not checking the return
```

# The SAAS App and the Rule Board

The screenshot displays the Semgrep Rule Board interface in a web browser. The browser's address bar shows the URL `semgrep.dev/orgs/returntocorp/board`. The interface features a dark purple sidebar on the left with navigation links: `returntocorp`, `Tour`, `Dashboard`, `Projects`, `Rule board` (highlighted), `Findings (1.7k)`, `SCA beta`, `Editor beta`, `Settings`, `Support`, `Registry`, `Playground`, and `Docs`. At the bottom of the sidebar is the Semgrep logo and version `0.86.5`.

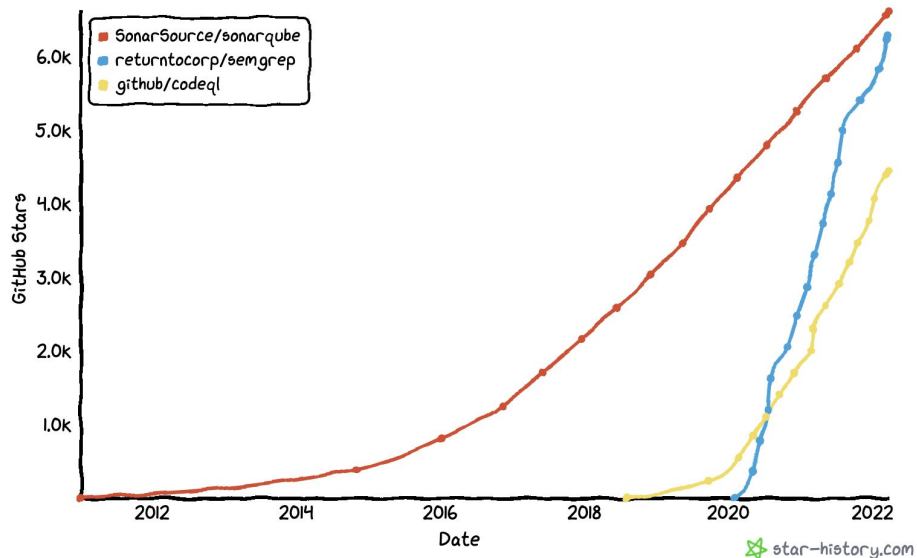
The main content area is titled `Rule board` and includes a `Filter...` dropdown and a `Config active` indicator. It is organized into three columns:

- Audit**: Contains two rule sets: `drewdennison's snipp...` and `returntocorp's snipp...`.
- PR/MR Comments**: Contains three rule sets: `github-actions ruleset` (3 of 3 rules included), `react ruleset` (12 of 14 rules included), and `terraform ruleset` (15 of 15 rules included). The `terraform ruleset` is expanded to show two sub-rules: `s3 public read bucket` (with `terraform lang` and `security` tags, and a toggle switch) and `all origins allowed` (with `terraform lang` and `security` tags, and a toggle switch).
- Block**: Contains five rule sets: `ocaml ruleset` (31 of 31 rules included), `ci ruleset` (135 of 135 rules included), `xss ruleset` (78 of 79 rules included), `eslint-plugin-security ruleset` (6 of 7 rules included), and `bandit ruleset` (56 of 61 rules included).

Each rule set card includes a copy icon and a trash icon. The `PR/MR Comments` column also includes a sub-rule `s3 cors all origins` and a partially visible `unencrypted ebs vol...` rule.

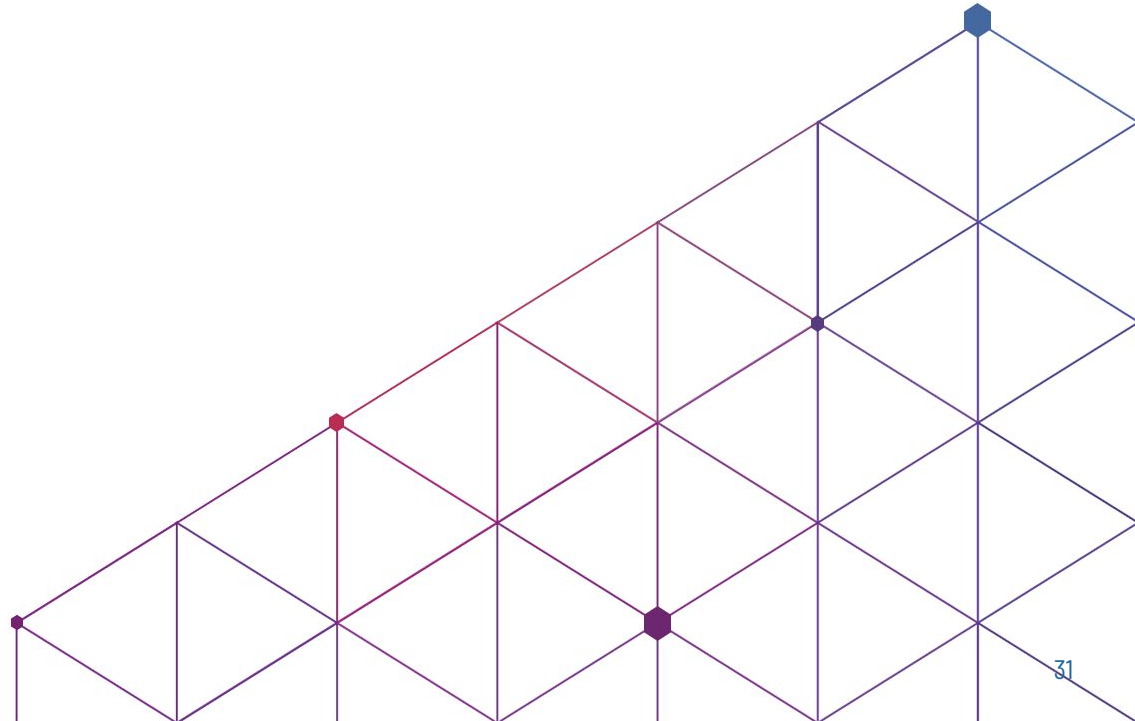
# Why do people like Semgrep?

- Fast
  - Easy to parallelize (analyze files separately)
  - Does not take long in CI; can run in CI (vs Coverity, ...)
  - Can even be used in editor (developer's workflow)
- Support most (popular) languages
  - Python, Javascript, Java, Go, C++, OCaml, Scala, ...
  - Takes few weeks to add a language (harder for CodeQL)
  - Config files (IaC) too: Docker, Terraform, ...
- Easy to setup
  - does not require buildable code (vs CodeQL, ...)
  - Easy to configure with Web App
- Easy to customize
  - readable rules
  - "Learn principles once, apply to many languages"



star-history.com

# Language Engineering



# ocaml-tree-sitter

Internally Semgrep relies on the [tree-sitter](#) library to parse code:

- Developed at Github (powers code highlighting in github.com, Atom, Neovim, some of VSCode plugins)
- GLR parser generator. No grammar action, generate CST from grammar (JSON program tree)
- Many bindings
- > 40 programming language grammars (C, C++, Java, Rust, Javascript, OCaml, ...)
- Small but active community (2-3 regular committer per language)

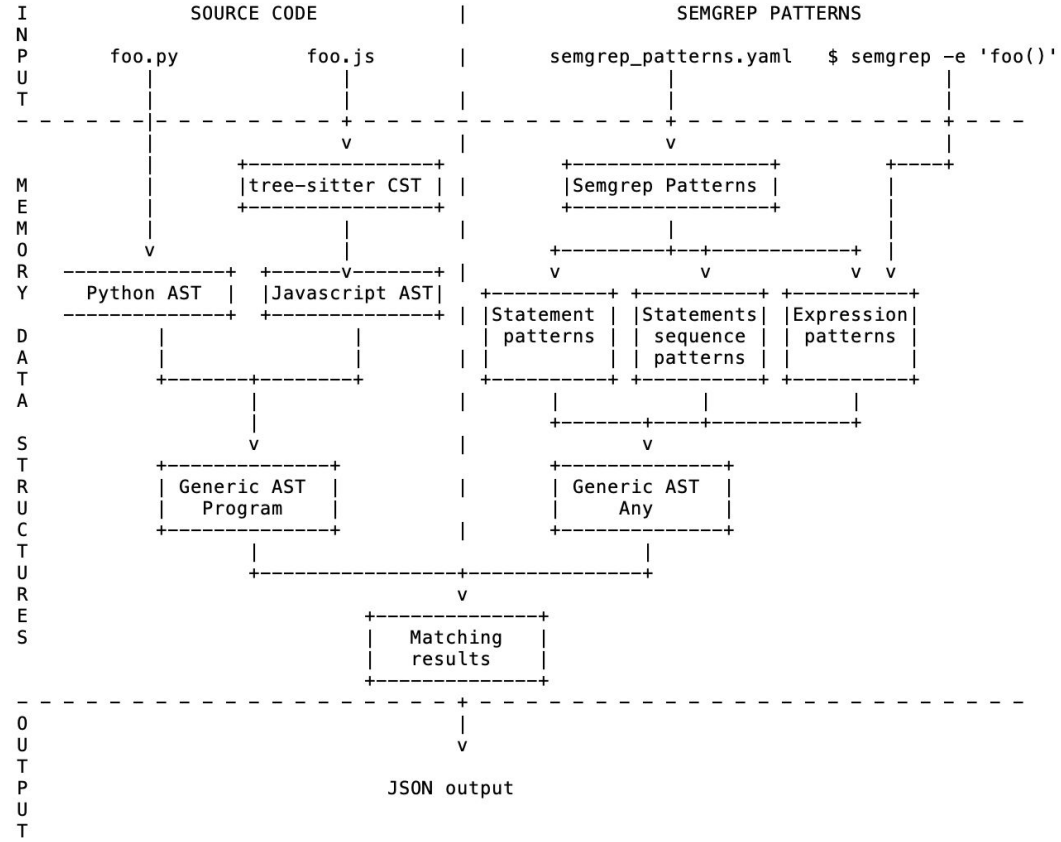
We developed [ocaml-tree-sitter](#), a tool to help generate typed AST from untyped CST

- Build on [reason-tree-sitter](#) OCaml binding to tree-sitter
- OCaml-ready parsers for many languages: <https://github.com/returntocorp/ocaml-tree-sitter-semgrep>
- OPAM packages soon for each languages



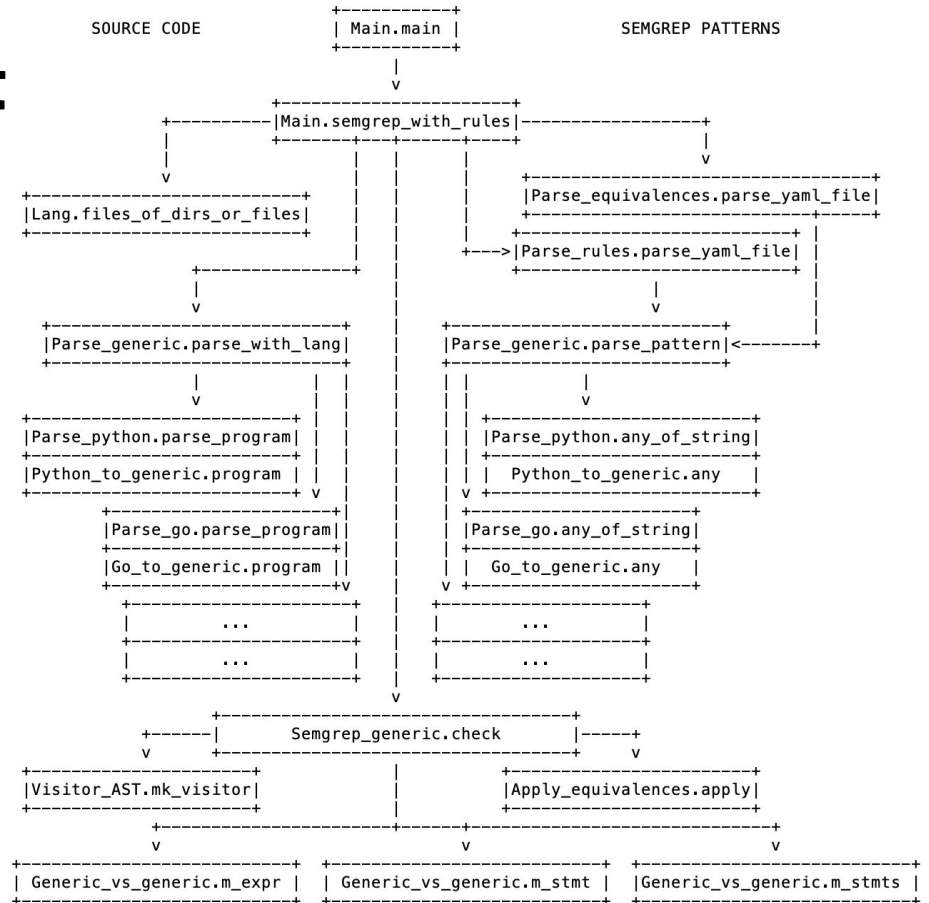
# Semgrep parsing architecture

- Pfff vs tree-sitter
- Target vs patterns



# Semgrep matching architect

- AST generic pattern vs AST generic target
- Visiting and matching
- Matching monad
- 



# Semgrep semantic analysis architecture

- Naming
  - Target: `let a = 1; function foo() { let a = 2; return a; }`
  - Pattern: `$X = 1; ... return $X;`
- Typing (declaration propagation)
- IL and CFG to support advanced features
  - Tainting (dataflow-based)
  - Constant propagation

# Future work

- DeepSemgrep
  - Same rules
  - Interfile/Interprocedural analysis (interfile constant propagation (Java), typing, tainting)
  - Slower, but less FPs/FNs
- Yaml -> Jsonnet (templating or rules, factorize rules, taint libraries, etc.)
- More languages
- More features
-

# We are hiring!



**R2C:** We're an SF based static analysis startup on a mission to profoundly improve software security and reliability.

**Join us!** Love OCaml? Passion for dev tools and/or security? Full-remote positions!

**Contact me:** [pad@r2c.dev](mailto:pad@r2c.dev)





# Semgrep

lightweight static analysis for many languages

Locally:

1. `(brew or pip) install semgrep`
2. `semgrep --config=r2c .`

Online editor:

- [semgrep.dev/playground](https://semgrep.dev/playground)

Yoann Padioleau | [pad@r2c.dev](mailto:pad@r2c.dev)

[r2c.dev](https://r2c.dev) |  [@r2cdev](https://twitter.com/r2cdev)

<https://r2c.dev/survey> ← plz :)



# Semgrep history

Academia -> Facebook -> R2C Startup:

- Coccinelle (2007):
  - Domain Specific Language (DSL) for **program transformation**
  - Just for **C** (mostly for Linux device drivers)
- Sgrep (2011):
  - Syntactical grep (trimmed down version of semantic patches), to **find bugs**
  - Just for **PHP** (for Facebook codebase)
- Semgrep (2020):
  - Semantic grep, Polyglot (**Python, Javascript, Java, Go, C, PHP, ...**)
  - An **ecosystem to improve security** (not just CLI: Playground, Web app, CI integration, ...)
  - In 2022 Semgrep is used by many companies (Dropbox, Netflix, Snowflake, Figma, Apple, ...)

“It takes 15 years for a research idea to reach the industry” - ??

# Semgrep terminology

/your/project/.semgrep.yml

```
rules:  
  - id: epeq-is-bad  
    patterns:  
      - pattern-not-inside: |  
          def __eq__(Pattern):  
            ...  
      - pattern-not-inside: assert(...)  
      - pattern-not-inside: assertTrue(...)  
      - pattern-not-inside: assertFalse(...)  
      - pattern-either:  
        - pattern:  $\$X == \$X$  Formula  
        - pattern:  $\$X != \$X$   
      - pattern-not: 1 == 1  
    message: "useless comparison operation `\$X == \$X` or `\$X !=  
    \$X`; if testing for floating point NaN, use `math.isnan`, or  
    `cmath.isnan` if the number is complex."  
    languages: [python]  
    severity: ERROR
```

Rule

/your/project/foo.

```
def foo():  
    return 1  
  
def bar(a, b):  
    return a + b
```

Target



# The playground (rule editor)

The screenshot shows the Semgrep playground interface. At the top, there's a navigation bar with "Semgrep" logo, "Registry", "Playground", "App", "Pricing", and "Docs". On the right, there are links for "Sign in / Sign up free", "Use in CI", and "Share". Below the navigation, there are tabs for "File", "Examples", "Tools", and "Help". The main area is titled "Untitled rule" and contains a "SEMGREP RULE" section with "Simple" and "Advanced" tabs. The "Simple" tab is active, showing a configuration for a rule: "language is" set to "Java", "code is" set to "\$X == \$X", and "and is not" set to "1 == 1". Below this is a "RULE METADATA" section and a "TEST CODE" section with a "Run" button. The test code is a Java class named "Bar" with a "main" method. Line 8 is highlighted, showing an "if (myBoolean == myBoolean) {" statement. On the right side, there is a "NEW EDITOR" section with a purple callout box containing a message about the new editor and buttons for "Hide" and "See current rule in Editor". Below that is a "MATCHES (3)" section with a white box containing the text for "Line 8": "`myBoolean == myBoolean` or `myBoolean != myBoolean` is always true. (Unless the value compared is a float or double). To test if `myBoolean` is not-a-number, use `Double.isNaN(myBoolean)`." and a partially visible "Line 13:".

Semgrep Registry Playground App Pricing Docs Sign in / Sign up free

File Examples Tools Help Use in CI Share

Untitled rule

SEMGREP RULE

Simple Advanced

language is Java

code is \$X == \$X + x

and is not 1 == 1 + x

► RULE METADATA

TEST CODE

```
1 class Bar {
2     void main() {
3         boolean myBoolean;
4
5         //myBoolean == myBoolean;
6
7         // ruleid:eqeq
8         if (myBoolean == myBoolean) {
9             continue;
```

Run ⌘↵

NEW EDITOR

✦ It's here! Check out the new Editor - a powerful new tool to help you create, test, and share your Semgrep rules.

Hide See current rule in Editor

MATCHES (3)

Line 8:

`myBoolean == myBoolean` or `myBoolean != myBoolean` is always true. (Unless the value compared is a float or double). To test if `myBoolean` is not-a-number, use `Double.isNaN(myBoolean)`.

Line 13:

# Configuration Files

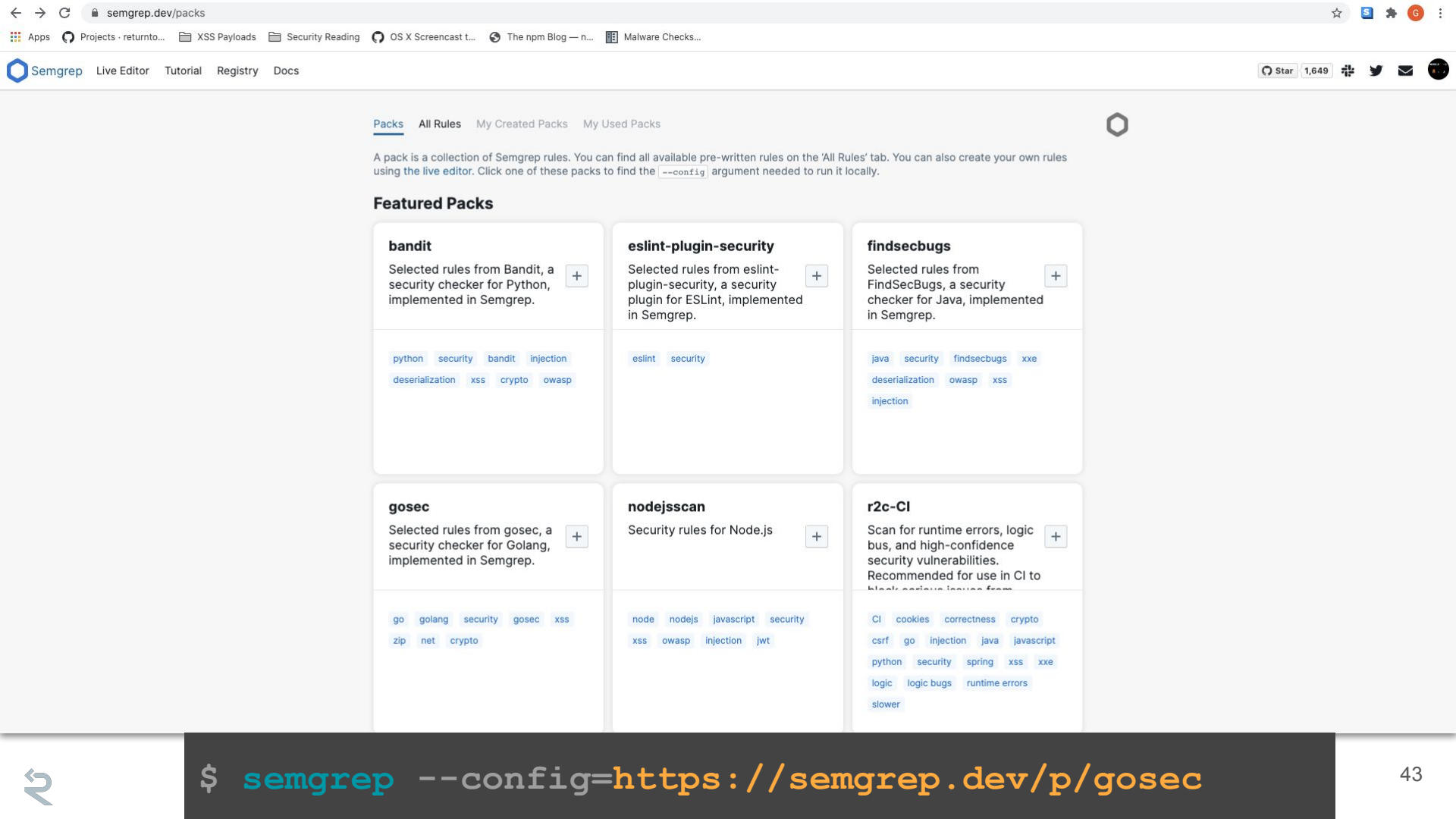
This document describes `semgrep` configuration files and provides rule examples. Configuration files are specified with the `--config` (or `-f`) flag. A single [YAML](#) file or a directory of files ending in `.yaml` or `.yml` may be specified. Each configuration file must match the [schema](#).

For more information on the `--config` flag see [other configuration options](#).

Contents:

- [Simple Example](#)
- [Other Configuration Options](#)
- [Schema](#)
- [Operators](#)
  - [pattern](#)
  - [patterns](#)
  - [pattern-either](#)
  - [pattern-regex](#)
  - [pattern-not](#)
  - [pattern-inside](#)
  - [pattern-not-inside](#)
  - [pattern-where-python](#)
- [Metavariable Matching](#)
  - [Metavariables in Logical ANDs](#)
  - [Metavariables in Logical ORs](#)
  - [Metavariables in Complex Logic](#)

<https://github.com/returntocorp/semgrep/blob/develop/docs/configuration-files.md>



[Packs](#) [All Rules](#) [My Created Packs](#) [My Used Packs](#)

A pack is a collection of Semgrep rules. You can find all available pre-written rules on the 'All Rules' tab. You can also create your own rules using the [live editor](#). Click one of these packs to find the `--config` argument needed to run it locally.

### Featured Packs

#### bandit

Selected rules from Bandit, a security checker for Python, implemented in Semgrep.



python security bandit injection  
deserialization xss crypto owasp

#### eslint-plugin-security

Selected rules from eslint-plugin-security, a security plugin for ESLint, implemented in Semgrep.



eslint security

#### findsecbugs

Selected rules from FindSecBugs, a security checker for Java, implemented in Semgrep.



java security findsecbugs xxe  
deserialization owasp xss  
injection

#### gosec

Selected rules from gosec, a security checker for Golang, implemented in Semgrep.



go golang security gosec xss  
zip net crypto

#### nodejsscan

Security rules for Node.js



node nodejs javascript security  
xss owasp injection jwt

#### r2c-CI

Scan for runtime errors, logic bus, and high-confidence security vulnerabilities. Recommended for use in CI to block serious issues from



CI cookies correctness crypto  
csrf go injection java javascript  
python security spring xss xxe  
logic logic bugs runtime errors  
slower

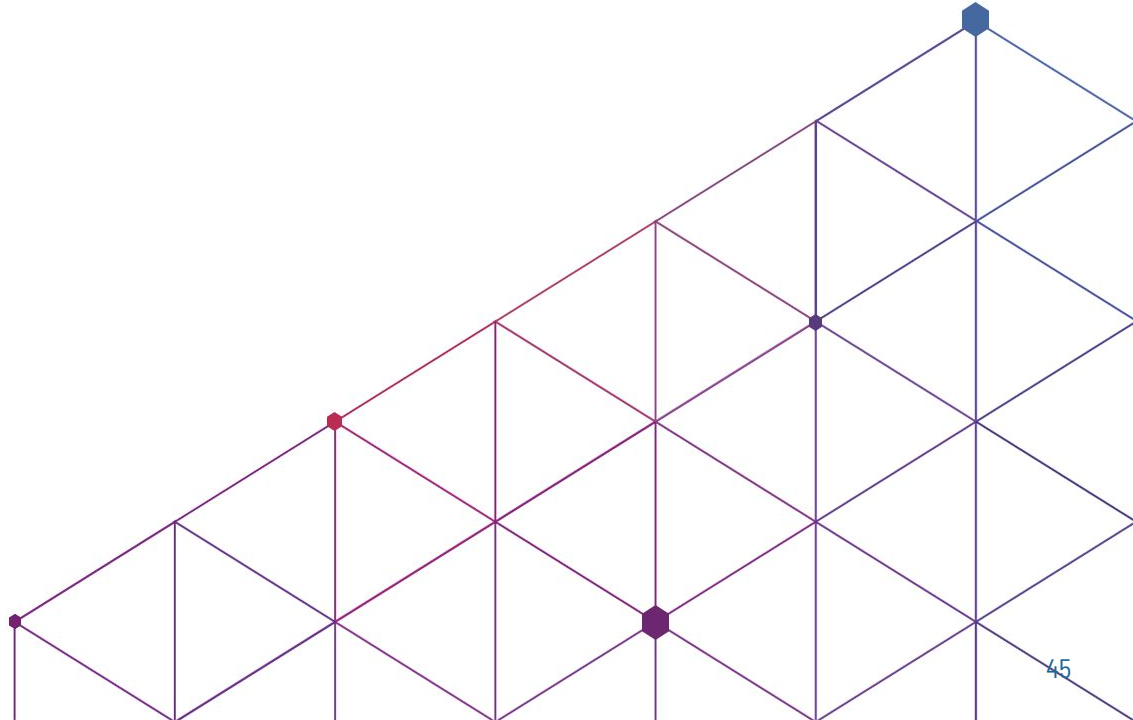
`$ semgrep --config=https://semgrep.dev/p/gosec`

# Community rule registry

[semgrep.live/registry](https://semgrep.live/registry) ⇒ [github.com/returntocorp/semgrep-rules](https://github.com/returntocorp/semgrep-rules)

```
$ brew install semgrep  
$ semgrep --config=<url>
```

# Integrations





github-actions bot reviewed now

[View changes](#)

aws.py

```
1 + from boto3 import client
2 +
3 + key = "jWnyVKHgaSRZVdX7ZQRATpoCkVsvPLRKNZCYRXRL"
4 + s3 = client("s3", aws_secret_access_key=key)
```



github-actions bot now



## semgrep-action

**Severity:** Warning 🟡

**Rule id:** python.boto3.security.hardcoded-token.hardcoded-token

**Message:** Hardcoded AWS access token detected. Use environment variables to access tokens (e.g., `os.environ.get(...)`) or use non version-controlled configuration files.

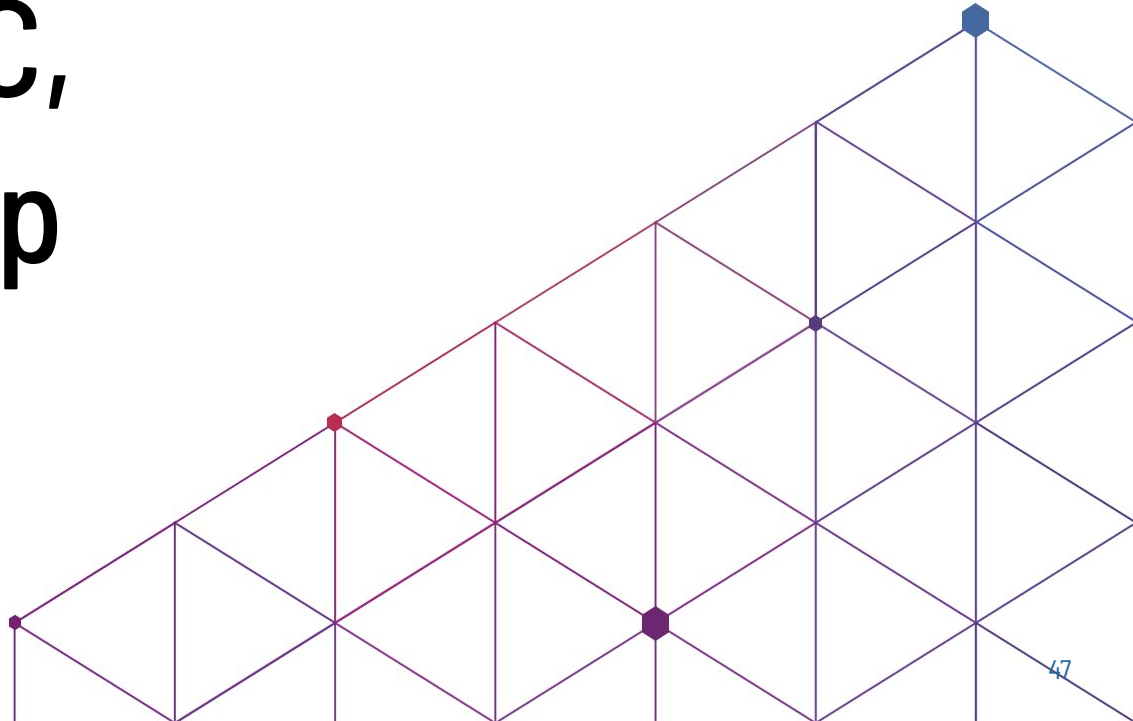
[Report an issue with this rule.](#)



Reply...

Resolve conversation

# Web App, SSC, DeepSemgrep



recap, a.k.a.  
"learn semgrep in 5 min"





# #1 Code equivalence (**semantic grep**)

```
$X == $X
```

Will match

```
(a+b != a+b) # <=> !(a+b==a+b)
```

```
foo(kwd1=1, kwd2=2, ...)
```

Will match

```
foo(kwd2=2, kwd1=1, kwd3=3)
```

```
subprocess.open(...)
```

Will match

```
from subprocess import open as  
sub_open  
  
result = sub_open("ls")
```

```
import foo.bar
```

Will match

```
from foo import bar
```

- **semgrep** knows about the semantics of the language, so one pattern can match variations of equivalent code (constant propagation! <https://semgrep.live/4K5>)

## #2: '...' ellipsis operator

```
foo(...,5)
```

Will match

```
foo(1,2,3,4,5)  
foo(5)
```

```
foo("...")
```

Will match

```
foo("whatever sequence of chars")
```

```
$V = get()  
...  
eval($V)
```

Will match

```
user_data = get()  
print("do stuff")  
foobar()  
eval(user_data)
```

'...' can match sequences of:

- Arguments, parameters
- Characters
- Statements

# #3 Metavariables (part 1)

<pre>foo(\$X,2)</pre>	Will match	<pre>foo(1,2)</pre>
<pre>if \$E:     foo()</pre>	Will match	<pre>if x &gt; 2:     foo()</pre>
<pre>if \$X &gt; \$Y:     \$S</pre>	Will match	<pre>if var &gt; 2:     return 1</pre>
<pre>\$F(1,2)</pre>	Will match	<pre>foo(1,2)</pre>

- **Metavariables** start with a \$ (\$X, \$Y, \$WHATEVER), contain uppercase ASCII characters
- **Matches:**
  - Expressions (including arguments)
  - Statements
  - Names (functions, fields, etc.)



## #3 Metavariables (part 2)

```
$X == $X
```

Will match

```
if (a+b == a+b) :
```

```
if $E:  
    $S  
else:  
    $S
```

Will match

```
if x > 2:  
    foo()  
    bar()  
else:  
    foo()  
    bar()
```

```
$V = open()  
close($V)
```

Will match

```
myfile = open()  
close(myfile)
```

You can reuse the same metavariable: **semgrep** enforces **equality constraint**