



Finding bugs and scaling your security program with Semgrep

BSides Milano 2023

Pieter De Cremer & Claudio Merloni

Security researchers at Semgrep

semgrep



Pieter De Cremer
0xDC0DE



Claudio Merloni
p4p3r



Let's get to know each other



- **What's your current role?**
 - Security vs non security
 - Full Stack, Backend, Frontend, DevOps, QA, other?
- **What are you most excited to learn?**
 - Enforcing coding standards in your organization
 - Finding security bugs
 - Continuous code scanning in CI
 - Security
 - Other
- **What code-related thing has given you the biggest headache recently?**

A friendly ask



Please send us example code snippets and what you want to find!

We'll write rules together live



User: nh

Password: wifi

<https://semgrep.dev/playground/new>



Finding bugs and scaling your security program with Semgrep

BSides Milano 2023

Pieter De Cremer & Claudio Merloni

Scaling your security program with Semgrep

Semgrep

Customizing rules

Remediation guidance

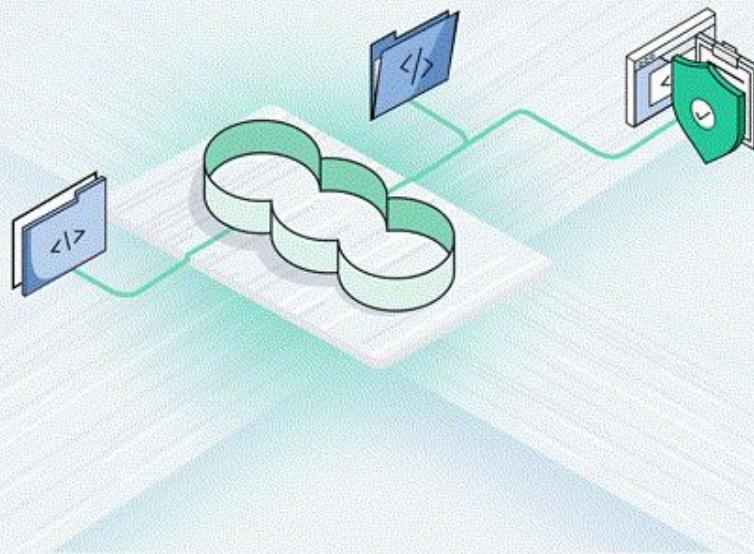
More analysis features

Bring your own code

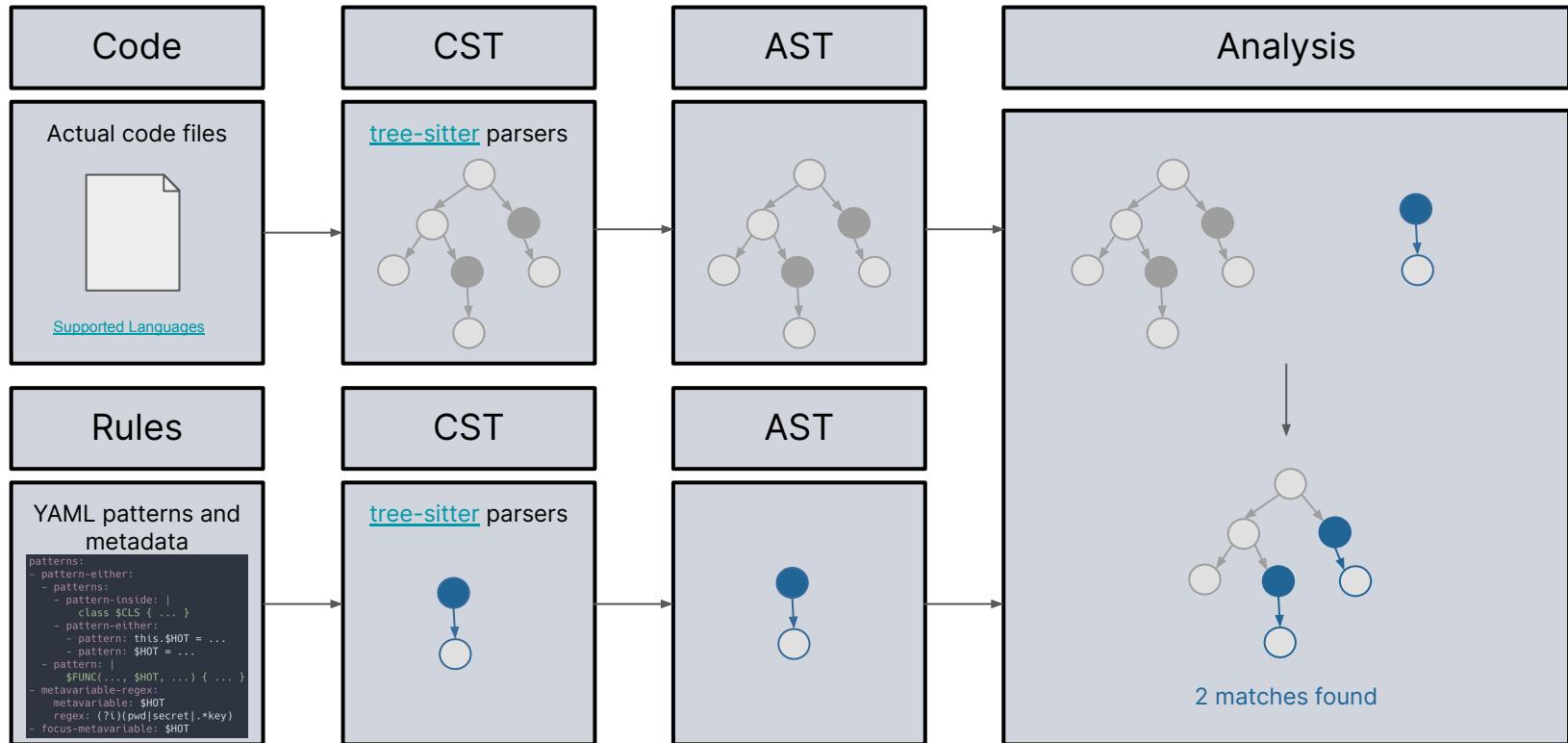
POWERED BY SEMGREP OPEN SOURCE

Code Analysis at Ludicrous Speed

Find bugs and dependency vulnerabilities, run security scans in CI, and enforce standards across your organization.

[Start scanning for free](#)[Book a demo](#)

Analysis architecture



Semgrep is open source

<https://github.com/returntocorp/semgrep>

returntocorp / semgrep Public

Edit Pins Watch 77 Fork 374 Star 7.1k

Code Issues 391 Pull requests 13 Actions Security Insights

develop 110 branches 144 tags Go to file Add file Code

nmote chore(java): Upgrade semgrep-java (#6087) ... e718782 10 hours ago 4,073 commits

.circleci Reduce memory usage when not necessary (#5730) 2 months ago

.devcontainer Set up GitHub codespaces (#5131) 4 months ago

.github use the semgrep ci bot instead of PAT for E2E checks (#6032) 7 days ago

.vscode Add more pre-commit hooks (#4943) 5 months ago

changelog.d chore(java): Upgrade semgrep-java (#6087) 10 hours ago

cli Guard removal of rules and targets files (#6084) 10 hours ago

doc Remove documentation from semgrep/doc (#3700) 13 months ago

dockerfiles Add a 'homebrew-setup' target to the main makefile (#5955) 21 days ago

interfaces Reduce size of targets file (#5748) 2 months ago

parsing-stats sacred software engineering tradition (#6017) 12 days ago

perf fix(python): Parse tuples in array accesses (#6004) 14 days ago

scripts Fix bad comment syntax in setup-m1-builder.sh (#6027) 11 days ago

semgrep-core chore(java): Upgrade semgrep-java (#6087) 10 hours ago

toy-matcher Install atdpy (#4844) 6 months ago

.dockereignore create a real release instead of a draft (#5689) 2 months ago

.gitattributes Stop using union-merge for CHANGELOG.md (#5577) 3 months ago

.gitignore create a real release instead of a draft (#5689) 2 months ago

.gitmodules Add semgrep-jsonnet (#6042) 8 days ago

.ocamlformat Install atdpy (#4844) 6 months ago

.pre-commit-config.yaml Revert "Simplify Dockerfile and fix Arch Linux failure (#6019)" (#6028) 11 days ago

.pre-commit-hooks.yaml Set default pre-commit flags (#4395) 9 months ago

.semgrepignore Rename semgrep dir to cli (#5531) 3 months ago

About

Lightweight static analysis for many languages. Find bug variants with patterns that look like source code.

semgrep.dev

javascript ruby python c java go typescript static-code-analysis static-analysis sast r2c semgrep

Readme View license Code of conduct 7.1k stars 77 watching 374 forks

Releases 136

Release v0.112.1 Latest 5 days ago + 135 releases

Used by 155

Contributors 107

+ 96 contributors

<https://github.com/returntocorp/semgrep>

Semgrep rules are open source

<https://github.com/returntocorp/semgrep-rules>

Code Issues Pull requests Discussions Actions Security Insights Settings

develop 176 branches 0 tags Go to file Add file Code

3 authors Trigger updated argo workflow for PRs (#2402) ... fdac95 5 minutes ago 2,382 commits

File	Description	Time Ago
.github	Trigger updated argo workflow for PRs (#2402)	5 minutes ago
bash	Reduce findings span - part 1 (#2081)	3 months ago
c/lang	Optimize the rule use-after-free (#2403)	7 days ago
contrib	Update express sources (#2310)	last month
csharp	merge develop	15 days ago
dockerfile	add dockerfile metadata	6 hours ago
fingerprints	Add older file exts for PHP fingerprint	8 months ago
generic	adjust failing fixtures	14 days ago
go	Merge branch 'develop' into dockerfile-metadata	6 hours ago
html	remove test	5 hours ago
java	set test to be todoid and remove asvs	2 days ago
javascript	add eol	2 days ago
json	Update package-dependencies-check.yml	3 months ago
kotlinlang/security	increase severity of command injection rules (#1877)	7 months ago
ocaml/lang	Add OCaml rule 'bad-reraise' (#2142)	3 months ago
php	fix meta for echoed-request rule	21 days ago
problem-based-packs/insecure-tra...	Add missing semicolons that confuse the semgrep parser	last month
python	add todoid	5 days ago
ruby	Update ruby-eval to catch other access to params + cookies (#2412)	5 days ago
scala	Filter logger-style statements from tainted-sql-string (#2101)	4 months ago
scripts	Fix 'make test' (#2406)	6 days ago

About Semgrep rules registry semgrep.dev/registry security static-analysis program-analysis security-scanner grep-like semgrep semgrep-rules semgrep-registry Readme View license Code of conduct 427 stars 24 watching 223 forks

Releases No releases published Create a new release

Packages No packages published Publish your first package

Contributors 159 + 148 contributors

<https://github.com/returntocorp/semgrep-rules>

Semgrep's philosophy

TL;DR

- Free & OSS
- Local & self-contained
- Runs everywhere, analyses everything
- User-friendly: get started using it and writing rules in minutes

<https://semgrep.dev/docs/contributing/semgrep-philosophy/>

Getting started with Semgrep

Installation

- brew install semgrep
- pip install semgrep

Usage

- semgrep --config=<rule folder/file> <target folder/file>

Alternatively use docker

- docker run --rm -v "\${PWD}:/src" returntocorp/semgrep semgrep --config=<rules>

Can be used in CLI, pre-commit, CI workflows, etc.

<https://semgrep.dev/docs/getting-started/>

Scaling your security program with Semgrep

Semgrep

Customizing rules

Remediation guidance

More analysis features

Bring your own code

Guidelines will be different for different code bases

app/controllers/widget_controller.rb

```
85 +  
86 +      before_action :ensure_user
```



You 3 days ago Member



Hey this looks great, but we've stopped using `ensure_user()`, could you use `ensure_logged_in()` instead?

Maturity frameworks recommend adapting rules to your organization

BSIMM



SSDL Touchpoints

Architecture Analysis (AA)

- Define and use AA process. [AA2.1]
- Standardize architectural descriptions (including data flow). [AA2.2]
- Make SSG available as AA resource or mentor. [AA2.3]

Code Review (CR)

- Assign tool mentors. [CR2.5]
- Use automated tools with tailored rules. [CR2.6]
- Use a top N bugs list (real data preferred). [CR2.7]

<https://www.bsimm.com/content/dam/bsimm/datasheets/BSIMM-activities-at-a-glance.pdf>

OWASP SAMM

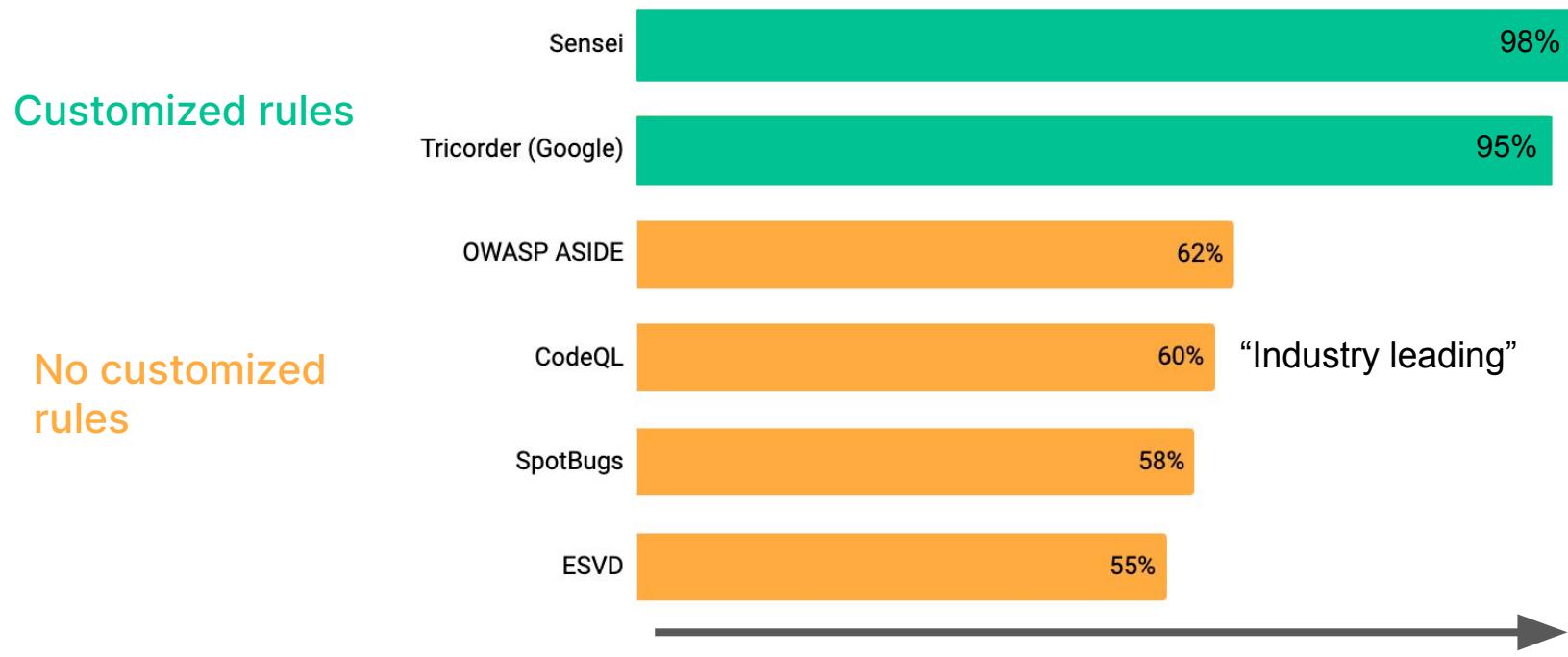
Model | Verification | Security Testing

The Security Testing (ST) practice leverages the fact that, while automated security testing is fast and scales well to numerous applications, in-depth testing based on good knowledge of an application and its business logic is often only possible via slower, manual expert security testing. Each stream therefore has one approach at its core.

The first stream focuses on establishing a common security baseline to automatically detect so-called "low hanging fruit". Progressively customize the automated tests for each application and increase their frequency of execution to detect more bugs and regressions earlier, as close as possible to their inception. The more bugs the automated processes can detect, the more time experts have to use their knowledge and creativity to focus on more complex attack vectors and ensure in-depth application testing in the second stream. As manual review is slow and hard to scale, reviewers prioritize testing components based on their risk, recent relevant changes, or upcoming major releases. Organizations can also access external expertise by participating in bug bounty programs, for example.

<https://owaspSAMM.org/model/verification/security-testing/>

Research shows that customized rules increase developer interaction



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

Semgrep data shows that customized rules increase developer interaction



Customized rules 50% higher fix rate
compared to generally applicable rules

Demo

Marking banned, deprecated or dangerous functions

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

```
1 # Use of exec() is completely banned. Find all calls to exec().
2
3 import exec as safe_function
4
5 # ruleid: python-exec
6 safe_function(user_input)
7
8 # ruleid: python-exec
9 exec()
10
11 # ruleid: python-exec
12 exec("ls")
```

Solution: <https://semgrep.dev/s/B1yx>

Writing rules with Semgrep



 go.semgrep.dev/slack

#bsides-milano-2023-workshop

Link to these slides: go.semgrep.dev/bsides_milano_2023

Playground: <https://semgrep.dev/playground/new>

Rule syntax: <https://semgrep.dev/docs/writing-rules/rule-syntax/>

Pattern syntax: <https://semgrep.dev/docs/writing-rules/pattern-syntax/>

Exercise

Find calls to exec() in Ruby

<https://semgrep.dev/playground/s/xE3j>

Solution: <https://semgrep.dev/playground/s/wEy6>

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

False positive (FP)

security perspective

secure code marked as insecure

Effective False Positive (EFP)

developer perspective

any marking a developer won't fix

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

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

Exercise

Find calls to exec() in Ruby

Find calls with only 1 argument

Find calls where the argument is not a string literal

<https://semgrep.dev/s/Ox1p>

Need help?

[Hint](#)

[Hint 2](#)

Exercise

Order of API calls must be enforced

<https://semgrep.dev/playground/s/J8WR>

```
public void base_ok(Transaction t) {  
    // OK: verify called before make  
    verify_transaction(t);  
    make_transaction(t);  
}  
  
// ruleid:find-unverified-transactions  
public void late_verify(Transaction t){  
    // BAD: transaction verified after being made  
    make_transaction(t);  
    verify_transaction(t);  
}
```

Need help?

[Hint 1](#)

[Hint 2](#)

Solution: <https://semgrep.dev/s/qNjn>

Exercise

Enforcing controller conventions

<https://semgrep.dev/s/8Jy7>

```
7 # Inherits from ApplicationController: great!
8 class ProtectedController < ApplicationController
9 end
10
11 # Doesn't inherit from ApplicationController: warn!
12 class BarController < OtherController
13 end
14
15 # Doesn't inherit from anything: warn!
16 class ZomgController
17 end
```

Need help?

[Hint 1](#)

[Hint 2](#)

[Hint 3](#)

Solution: <https://semgrep.dev/s/2DYB>

Exercise

Match only GET, PUT and POST methods
<https://semgrep.dev/s/328e>

```
25 // ruleid: express-xss-two
26 app.put('/bar', function (req, res) { // different argument name
27   var foo = req.query.email;
28   console.log("PUT route");
29   res.write('Response</br>');
30 });
31
32 // ruleid: express-xss-two
33 app.post('/bar', function (request, res) {
34   var foo = request.query.email;
35   console.log("POST route");
36   res.write('Response</br>');
37 });
38
39 // ok
40 app.delete('/bar', function (request, res) {
41   var foo = request.query.email;
42   console.log("DELETE route");
43   res.write('Response</br>');
44 });
```

Need help?

[Hint 1](#)

Solution: <https://semgrep.dev/s/4Llx>

Exercise

Check RSA encryption key strength < 2048
<https://semgrep.dev/s/RLzv>

```
1 package main
2
3 import (
4     "crypto/rand"
5     "crypto/rsa"
6     "fmt"
7 )
8
9 func main() {
10    // ruleid: use-of-weak-rsa-key
11    pvk, err := rsa.GenerateKey(rand.Reader, 1024)
12    if err != nil {
13        fmt.Println(err)
14    }
15    fmt.Println(pvk)
16
17    // ok: use-of-weak-rsa-key
18    pvk, err := rsa.GenerateKey(rand.Reader, 2048)
19    if err != nil {
20        fmt.Println(err)
21    }
22    fmt.Println(pvk)
23 }
```

Need help?

[Hint 1](#)

Solution: <https://semgrep.dev/s/BWP7>

Scaling your security program with Semgrep

Semgrep

Customizing rules

Remediation guidance

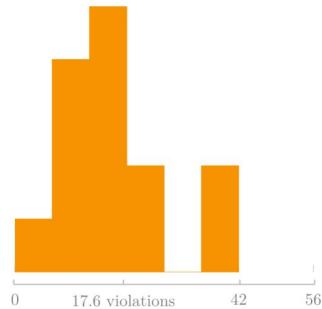
More analysis features

Bring your own code

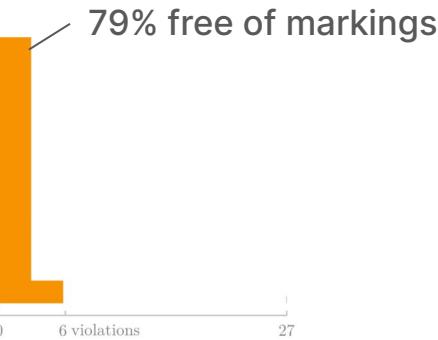
Remediation guidance improves fix rate

Remediation
guidance

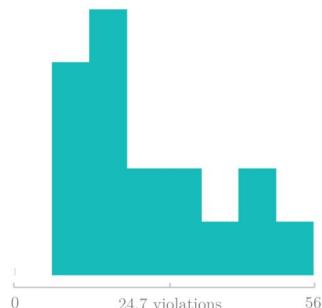
during assignment



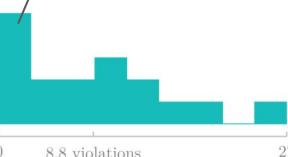
when finished



No remediation
guidance



6% free of markings



Remediation guidance improves development speed

Subject group	Mean remediation time
Test group	19.10 s
Control group	129 s (= 2.15 min)

Providing remediation guidance greatly
reduces **effective false positives**



Rules with autofix have 50% higher fix rate
compared to rules without autofix

Rules with an autofix create pull request comments in CI

A screenshot of a GitHub pull request interface. At the top, it shows a review from "semgrep-app bot" 16 minutes ago. Below the review, there is a code diff in a file named "frontend/src/components/Globals.tsx". The code contains several lines of TypeScript code, with lines 11 through 15 highlighted in green, indicating they have been modified by an autofix. A comment from "semgrep-app bot" 16 minutes ago suggests moving all hooks into a "hooks" directory. The comment includes a link to the specific lines (Lines 11-15) and instructions on how to ignore the finding. The status of the pull request is "status open".

semgrep-app bot reviewed 16 minutes ago

View changes

frontend/src/components/Globals.tsx

```
11 + function useQuery() {  
12 +   const { search } = useLocation();  
13 +  
14 +   return React.useMemo(() => new URLSearchParams(search), [search]);  
15 + }
```

semgrep-app bot 16 minutes ago

All hooks must be placed in a `hooks` directory. Please move this hook into `frontend/src/hooks/useQuery.tsx` or `frontend/src/pages/your-page/hooks/useQuery.tsx` ([Lines 11-15](#))

Reply with `/semgrep ignore <reason>` to ignore the finding created by `react-hooks-in-hooks-directory`.

status open

Semgrep App allows you to scan your Git repositories

The screenshot shows the Semgrep Cloud Platform login interface. On the left, a white sidebar on a light green background lists four features: "Deploy in CI with the click of a button", "Custom rules: Save, share, use in CI", "Manage rules across all your projects", and "See results in PR comments or Slack". The main area features the Semgrep logo (three overlapping circles) and the text "Semgrep Cloud Platform". It includes a "Sign in or sign up" link, two prominent buttons ("Sign in with GitHub" in blue and "Sign in with GitLab" in orange), and a "Use SSO" link. At the bottom, a note states: "By signing in you accept our [terms of service](#) and [privacy policy](#)".

<https://semgrep.dev/login>

Developers prefer fixing code in the IDE

Tool	Quick-fixes	Fixes applied
Sensei	IDE	73%
Tricorder (Google)	Review tool	20%

Semgrep has a VS Code Extension



A screenshot of the VS Code interface showing a Semgrep analysis results panel. The panel displays code from a file named 'pyramid-cookies.py' with several annotations:

- Line 4: A comment block containing 'true positives' is annotated with a yellow exclamation mark.
- Line 9: The parameter 'httponly=False' in the call to 'AuthTktCookieHelper' is underlined with a yellow wavy line, indicating a potential issue.
- Line 13: The parameter 'httponly=False' in the call to 'AuthTktCookieHelper' is underlined with a yellow wavy line, indicating a potential issue.
- Line 16: The variable 'myBoolean' is annotated with a yellow exclamation mark.

```
pyramid-cookies.py 4, U X
pyramid-cookies.py > ⚡ bad1
4  ##### true positives #####
5
6
7  def bad1():
8      # ruleid: pyramid-cookies
9      authhtkt = AuthTktCookieHelper(secret="test", httponly=False)
10
11 def bad2():
12     # ruleid: pyramid-cookies
13     authhtkt = AuthTktCookieHelper(secret="test", httponly=False, name="a name")
14
15 def bad3():
16     myBoolean = False
17     # ruleid: pyramid-cookies
```

Demo

Writing an autofix: secure cookies

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

```
7 def bad1():
8     # ruleid: pyramid-cookies
9     authkt = AuthTktCookieHelper(secret="test")
```

Matches

↑ Line 9

Semgrep found a match

Apply autofix

```
authkt = AuthTktCookieHelper(secret="test", httponly=True)
```

Solution: <https://semgrep.dev/s/Aqv0>

Exercise

Secure cookies part 2: replace httponly=False

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

Only the boolean is marked!

```
7 def bad1():
8     # ruleid: pyramid-cookies
9     authktkt = AuthTktCookieHelper(secret="test", httponly=False)
10
11 def bad2():
12     # ruleid: pyramid-cookies
13     authktkt = AuthTktCookieHelper(secret="test", httponly=False, name="a name")
14
15 def bad3():
16     myBoolean = False
17     # ruleid: pyramid-cookies
18     authktkt = AuthTktCookieHelper(secret="test", httponly=myBoolean)
```

Need help?

[Hint 1](#) [Hint 2](#)
[Hint 3](#)

A possible solution: <https://semgrep.dev/s/XPBw>

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)  
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

```
AuthTktCookieHelper(secret="test", httponly=False)
```

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)  
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

```
AuthTktCookieHelper(secret="test", httponly=False)
```

\$... BEFORE →
\$... AFTER →

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

```
AuthTktCookieHelper(secret="test", httponly=False)
```

\$... BEFORE → secret="test"
\$... AFTER → []

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

```
AuthTktCookieHelper(secret="test", httponly=False)
```

\$... BEFORE → secret="test"
\$... AFTER → []

```
AuthTktCookieHelper($ ... BEFORE, $ ... AFTER)
```

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

```
AuthTktCookieHelper(secret="test", httponly=False)
```

\$... BEFORE → secret="test"
\$... AFTER → []

```
AuthTktCookieHelper(secret="test", $... AFTER)
```

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

```
AuthTktCookieHelper(secret="test", httponly=False)
```

\$... BEFORE → secret="test"
\$... AFTER → []

```
AuthTktCookieHelper(secret="test", )
```

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

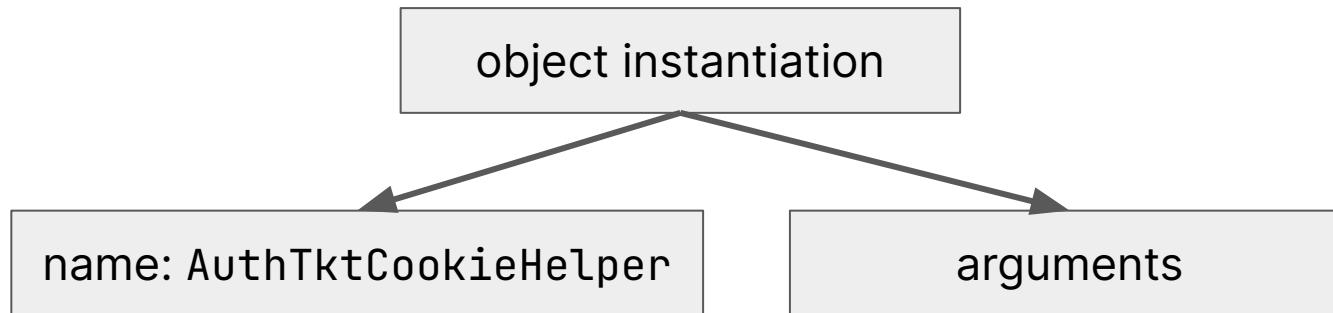
Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)  
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

object instantiation

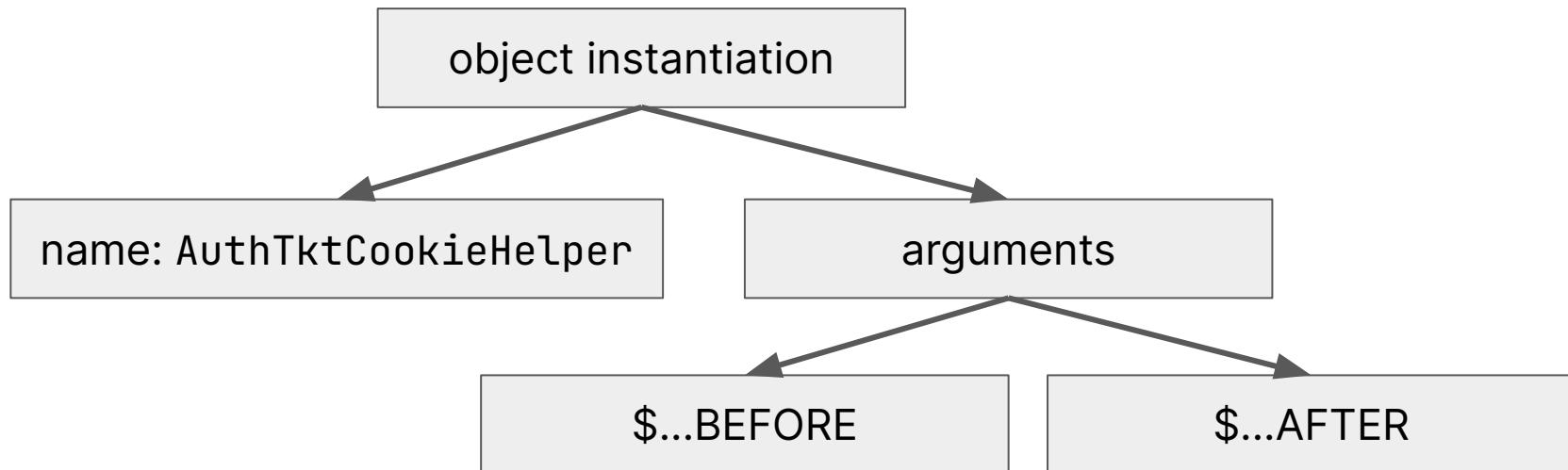
Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)  
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```



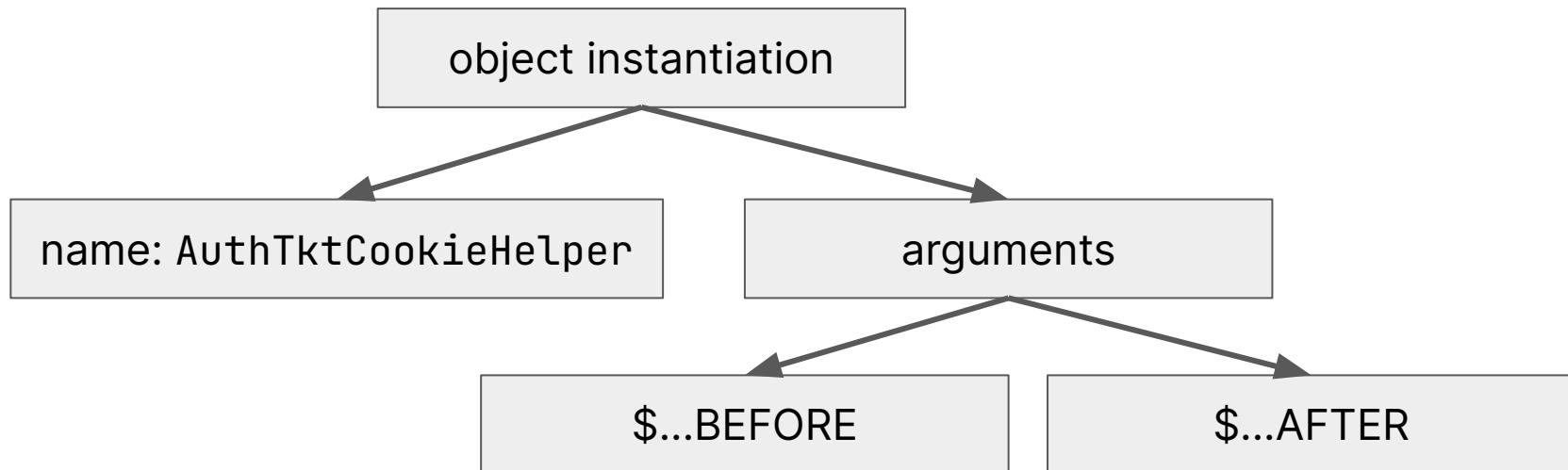
Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)  
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```



Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)  
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```

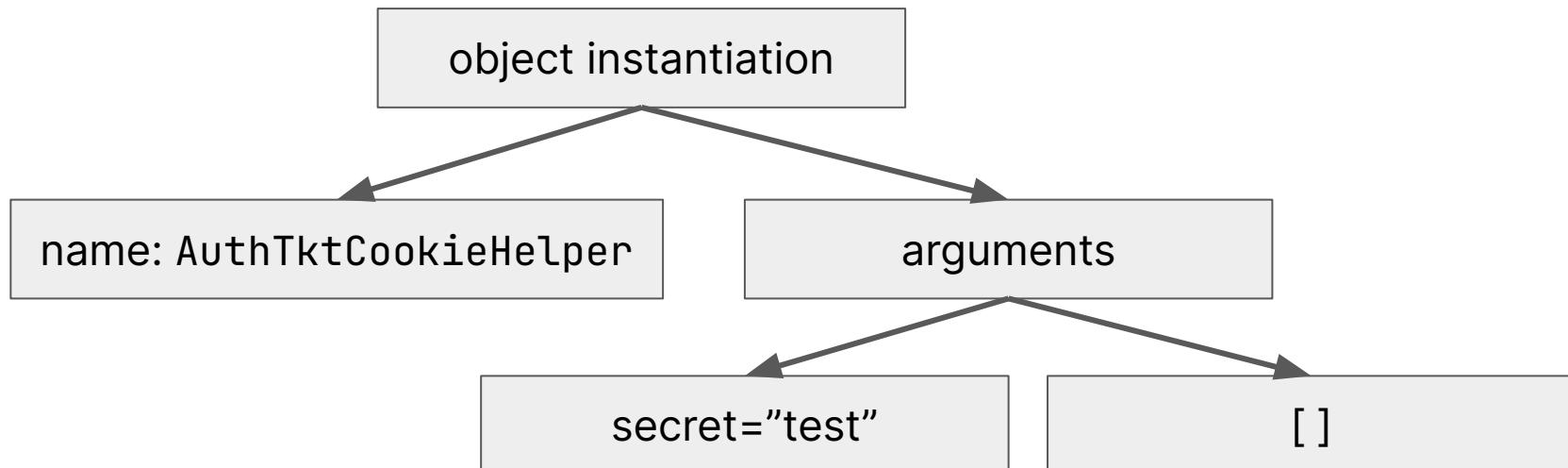


<https://semgrep.dev/blog/2022/autofixing-code-with-semgrep>

\$... BEFORE → secret="test"
\$... AFTER → []

Semgrep has AST-based autofix in Python and JavaScript

```
pattern: AuthTktCookieHelper($... BEFORE, httponly=False, $... AFTER)  
fix: AuthTktCookieHelper($... BEFORE, $... AFTER)
```



<https://semgrep.dev/blog/2022/autofixing-code-with-semgrep>

\$... BEFORE → secret="test"
\$... AFTER → []

Scaling your security program with Semgrep

Semgrep

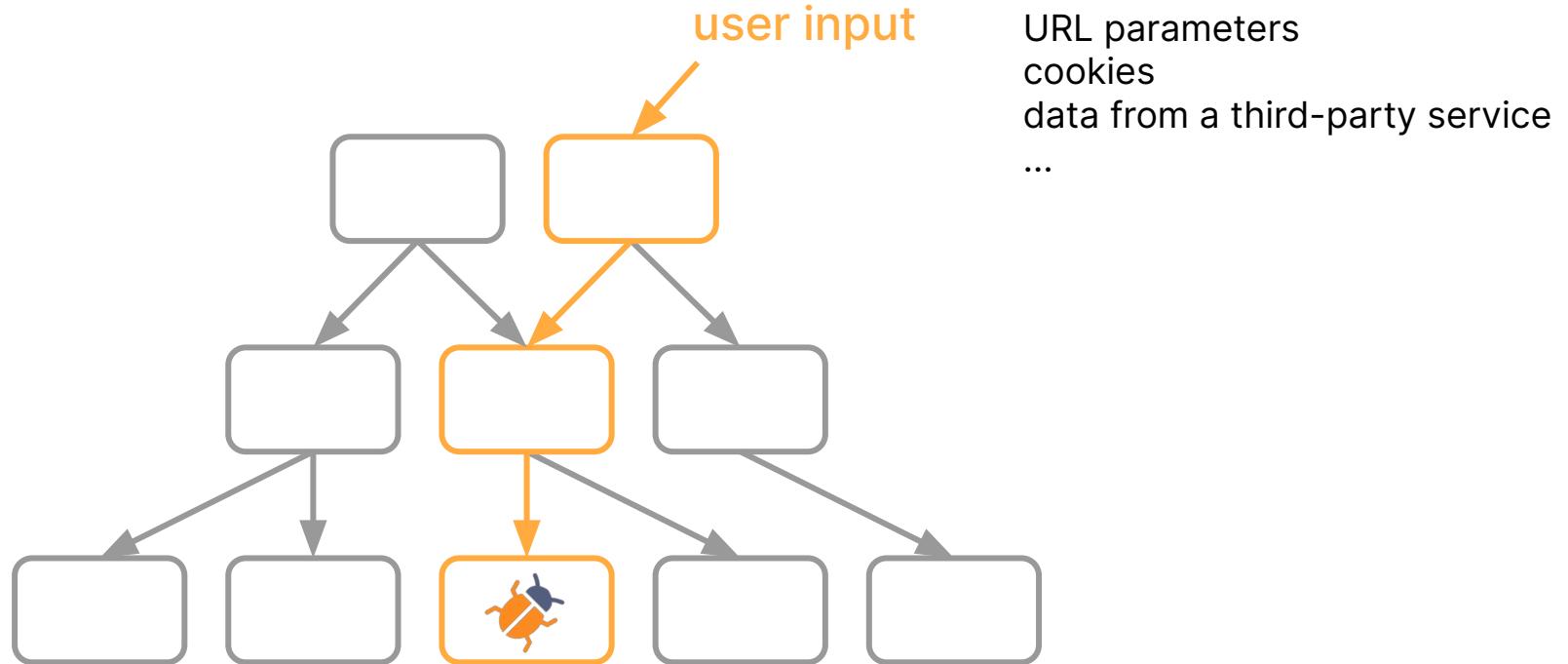
Customizing rules

Remediation guidance

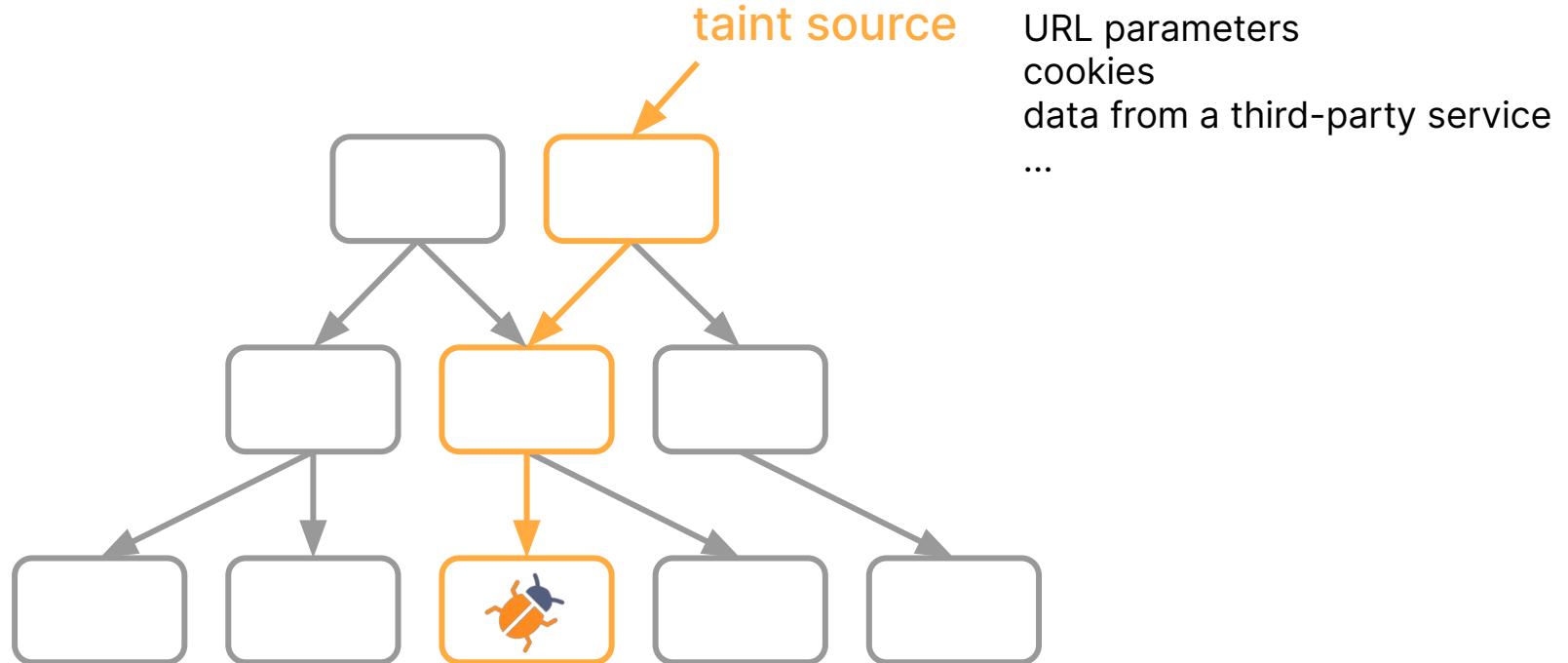
More analysis features

Bring your own code

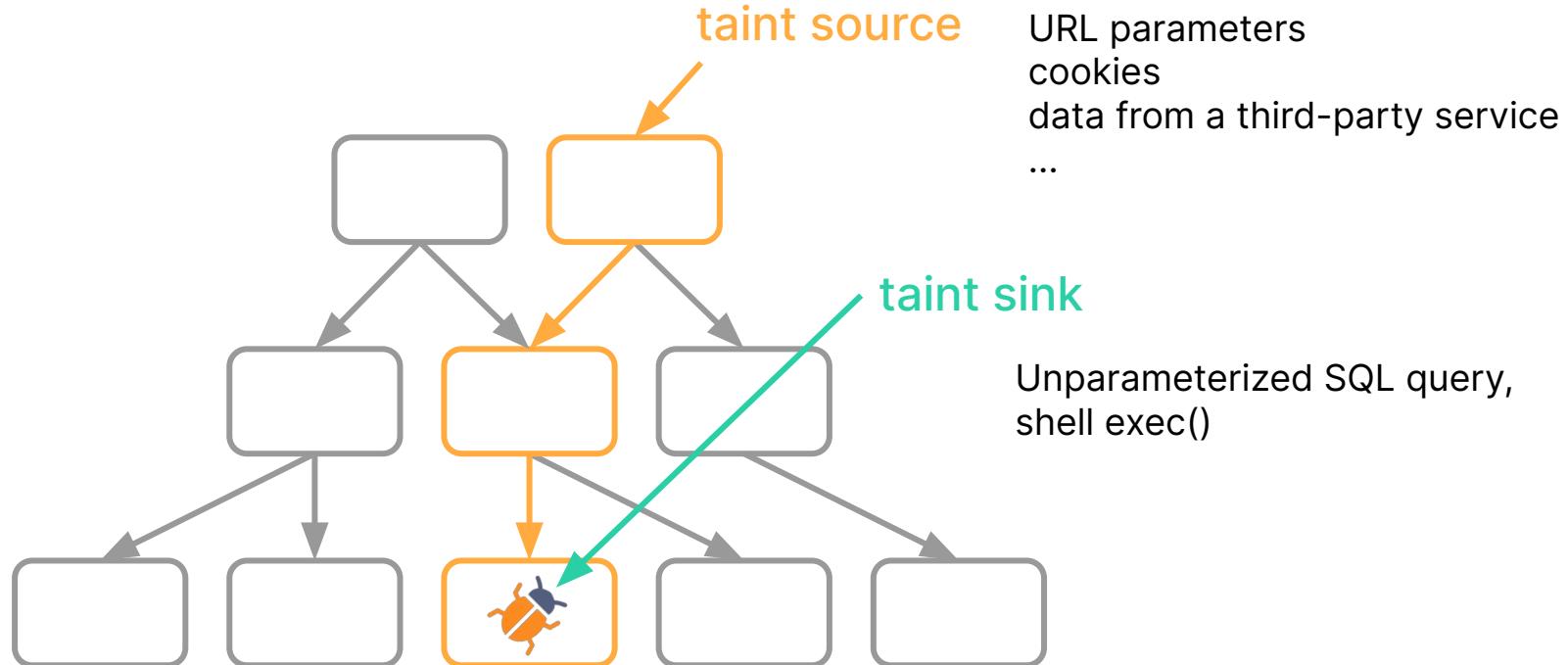
Dataflow analysis terminology



Dataflow analysis terminology



Dataflow analysis terminology



Tainted data flows from **source** to **sink**

source

where input enters

```
public class DataflowExample {  
    public static void main(String args[]) throws IOException {  
        String cmd = args[0];  
        Runtime.getRuntime().exec(cmd);  
    }  
}
```

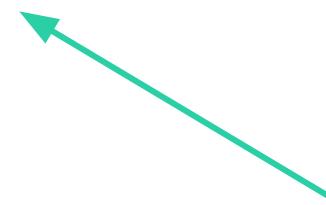
sink

where input is dangerous

Tainted data flows from **source** to **sink**

```
public class DataflowExample {  
    public static void main(String args[]) throws IOException {  
        String cmd = args[0];  
        Runtime.getRuntime().exec(cmd);  
    }  
}
```

source
where input enters



sink
where input is dangerous

Examples of sources and sinks

Sources

- URL / form params
- Headers
- Environment variables
- Reading from file or a database
- Command line arguments

Sinks

- Unparameterized SQL queries (SQLi)
- Web server sending html content (XSS)
- File names or path names(arbitrary file read/write)
- Shell exec'd commands (command injection)

“Hm, it seems like many security issues are basically data flow issues.” – Astute listener

“Hm, it seems like many security issues are basically data flow issues.” – Astute listener

Yes!

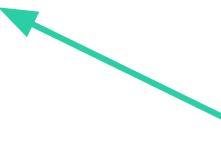
Tainted data propagates from source to sink

```
public class DataflowExample {  
    public static void main(String args[]) throws IOException {  
        String verboseMode = "-v";  
        String userCmd = args[0];  
        String cmd = userCmd + " " + verboseMode;  
        Runtime.getRuntime().exec(cmd);  
    }  
}
```

source
where input enters



propagator
where input propagates



sink
where input is dangerous

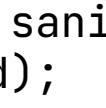
Tainted data can be sanitized before it reaches the sink

```
public class DataflowExample {  
    public static void main(String args[]) throws IOException {  
        String baseCmd = "rm";  
        String sanitizedPath = filterFilePath(args[0])  
        String cmd = baseCMD + " " + sanitizedPath;  
        Runtime.getRuntime().exec(cmd);  
    }  
}
```

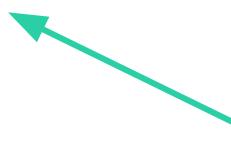
source
where input enters



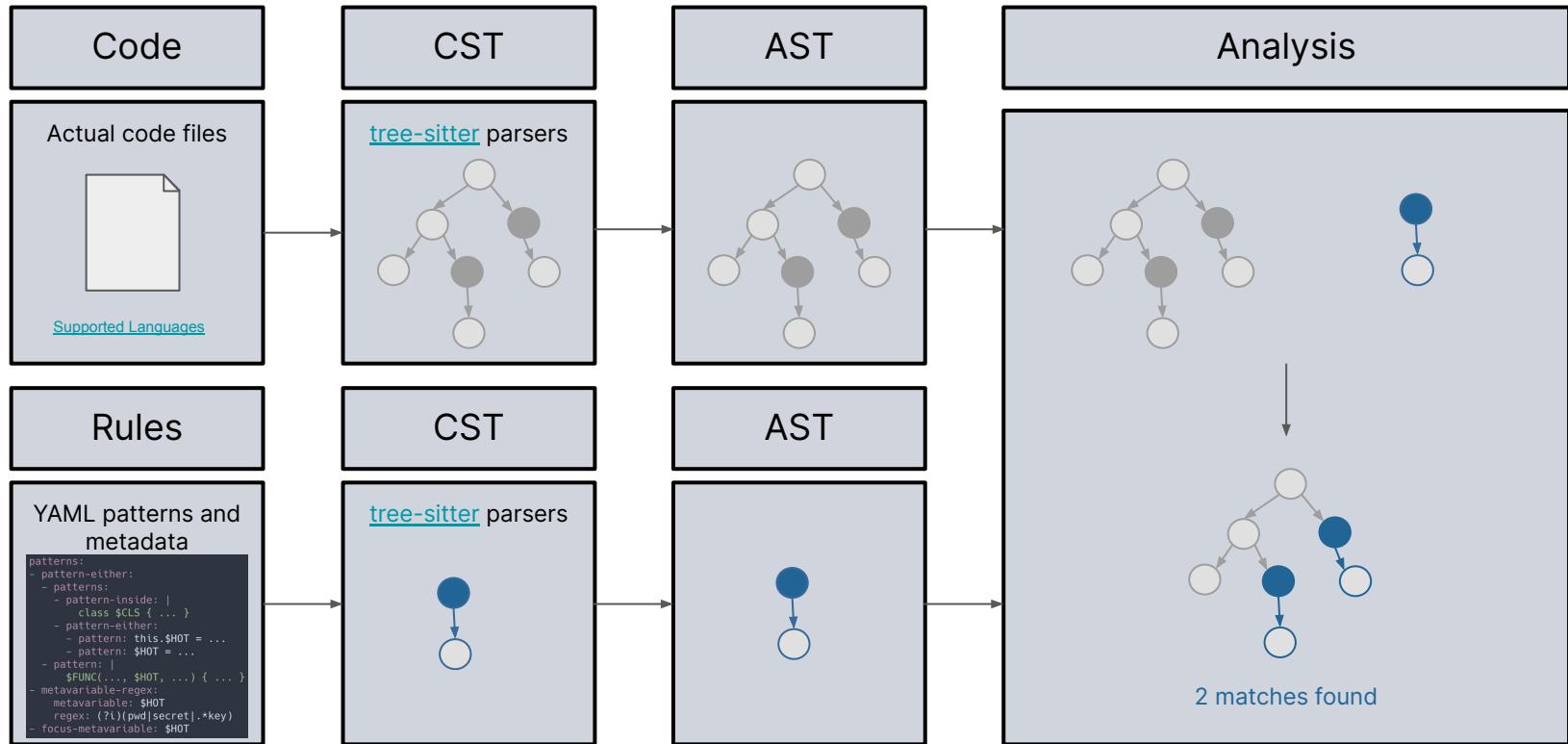
sanitizer
where input is sanitized



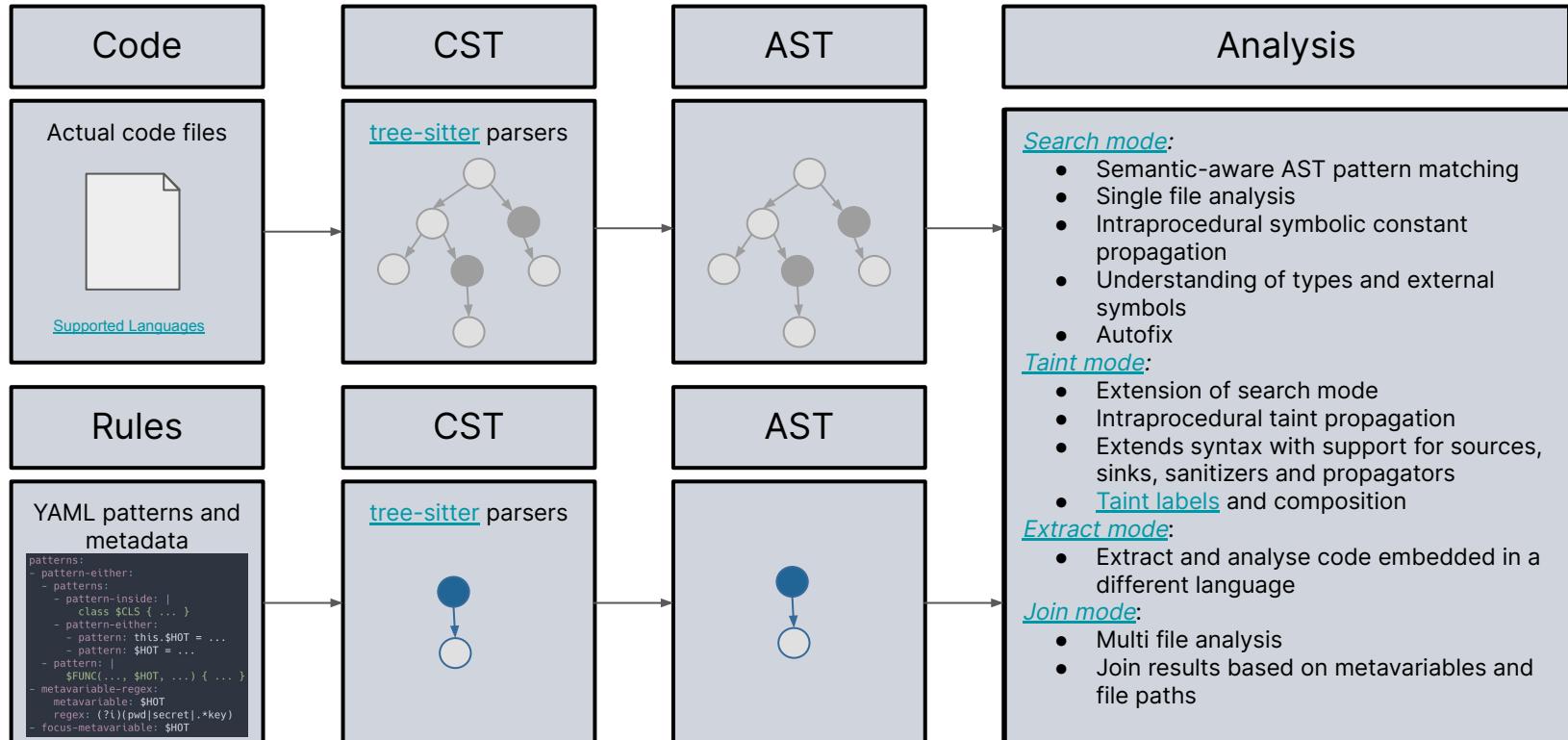
sink
where input is dangerous



Analysis architecture



Analysis architecture



Demo

Writing a taint mode rule

<https://semgrep.dev/playground/s/4WGL>

```
1 def f1():
2     s = source()
3     # ruleid: taint-demo
4     sink(s)
5
6 def f2():
7     s = source()
8     s2 = "string concat propagates already" + s
9     # ruleid: taint-demo
10    sink(s2)
11
```

Exercise

Do not pass request parameters to Runtime.exec
<https://semgrep.dev/s/EopL>

```
5  public void bad(HttpServletRequest req){
6      String cmd = req.getParam("cmd");
7
8      Runtime rt = Runtime.getRuntime();
9      //ruleid: find-runtime-exec-taint
10     rt.exec(cmd);
11 }
12
13 public void ok(HttpServletRequest req){
14     String cmd = req.getParam("cmd");
15
16     Other rt = new Other();
17     //ok: find-runtime-exec-taint
18     rt.exec(cmd);
19 }
20
21 public void ok2(HttpServletRequest req, String cmd){
22     Runtime rt = Runtime.getRuntime();
23     //ok: find-runtime-exec-taint
24     rt.exec(cmd);
25 }
```

Need help?

[Hint 1](#)

[Hint 2](#)

Solution: <https://semgrep.dev/s/Z57p>

Exercise

Find XSS in ExpressJS

<https://semgrep.dev/s/8Qqq>

```
7  app.get('/bar', function (req, res) {  
8      const email = req.params.email;  
9      const lowercaseEmail = toLowerCase(email);  
10  
11     // ruleid: express-taint  
12     res.write('Response</br>' + lowercaseEmail);  
13 });  
..
```

Exercise

Find path traversal in Go

<https://semgrep.dev/playground/s/J4Od>

```
14 func asdf(req *http.Request) {
15     username, _, _ := req.BasicAuth()
16
17     // ruleid: net-http-path-traversal-taint
18     os.Open(username)
19
20     unsanpath := path.Join("../", username)
21     // ruleid: net-http-path-traversal-taint
22     os.Open(unsanpath)
23
24     sanpath := path.Clean(username)
25     // ok: net-http-path-traversal-taint
26     os.Open(sanpath)
```

This is ok!

Exercise

SQL injection

<https://semgrep.dev/playground/s/Gk9e>

```
24     public void getAllFields(HttpServletRequest request) throws SQLException {
25         String tableName = request.getParameter("tableName");
26         Connection c = DB.getConnection();
27         // ruleid:formatted-sql-string
28         ResultSet rs = c.createStatement().executeQuery("SELECT * FROM " + tableName);
29     }
30
31     public void getAllFields(HttpServletRequest request) throws SQLException {
32         String tableName = request.getParameter("tableName");
33         Connection c = DB.getConnection();
34         // ruleid:formatted-sql-string
35         ResultSet rs = c.createStatement().execute("SELECT * FROM " + tableName);
36     }
37
38     public void getAllFields(HttpServletRequest request) throws SQLException {
39         String tableName = request.getParameter("tableName");
40         Connection c = DB.getConnection();
41         // ruleid:formatted-sql-string
42         StringBuilder str = new StringBuilder();
43         str.append(tableName);
44
45         ResultSet rs = c.createStatement().execute("SELECT * FROM " + str);
46     }
```

A possible solution: <https://semgrep.dev/playground/s/50RW>

Demo

Taint labels

<https://semgrep.dev/playground/s/A19w>

```
16     tainted = urlparse.urlparse(self.path).query
17
18     yml = ruamel.yaml.YAML(typ='unsafe')
19     # ruleid: tainted-ruamel
20     data = yml.load(tainted) # ok: tainted-ruamel
21
22     # ok: tainted-ruamel
23     data = yml.load(s)
24
25     yml = ruamel.yaml.YAML(typ='safe')
26     # ok: tainted-ruamel
27     data = yml.load(tainted)
28
29     # The default loader (typ='rt') is a direct derivative of the safe loader
30     yml = ruamel.yaml.YAML(typ='rt')
31     # ok: tainted-ruamel
32     data = yml.load(tainted)
33
34     yml = ruamel.yaml.YAML()
35     # ok: tainted-ruamel
36     data = yml.load(tainted)
```

New simplified rule syntax



patterns
pattern-either
pattern-inside
pattern-regex



all
any
inside
regex

Top level match or taint keyword

```
taint: ←  
sources:  
- all:  
  - inside:  
    def $FUNC(..., $REQUEST, ...):  
      ...  
  where:  
    - focus: $REQUEST  
    - metavariable: $REQUEST  
  all:  
    - request  
    - not:  
      inside: request.build_absolute_uri  
sinks:  
- any:  
  - all:  
    - subprocess.$FUNC(...)  
    - not: subprocess.$FUNC("...", ...)  
    - not: subprocess.$FUNC(["...", ...], ...)  
    - not:  
      inside:  
        $CMD = ["...", ...]  
        ...  
      subprocess.$FUNC($CMD, ...)  
  - all:  
    - |  
      subprocess.$FUNC(["$SHELL", "-c", ...], ...)  
  where:  
    - metavariable: $SHELL  
    - regex: ^(sh|bash|ksh|csh|tcsh|zsh)$
```

```
8   # ruleid:subprocess-injection  
9   subprocess.run("ping "+ ip)  
10  
11 def b(request):  
12     host = request.headers["HOST"]  
13     # ruleid:subprocess-injection  
14     subprocess.run("echo {} > log".format(host))  
15  
16 def d(request):  
17     cmd = request.POST.get("cmd")  
18     ip = request.POST.get("ip")  
19     command = [cmd, ip]  
20     # ruleid:subprocess-injection  
21     subprocess.capture_output(command)  
22  
23 def e(request):  
24     event = json.loads(request.body)  
25     cmd = event['id'].split()  
26     # ruleid:subprocess-injection  
27     subprocess.call([cmd[0], cmd[1], "some", "args"])  
28  
29 def f(request):  
30     event = json.loads(request.body)  
31     # ruleid:subprocess-injection  
32     subprocess.run(["bash", "-c", event['id']], shell=True)  
33  
34 def g(request):  
35     event = request.body  
36     python_file = f"""  
37         print("What is your name?")
```

Scaling your security program with Semgrep

Semgrep

Customizing rules

Remediation guidance

More analysis features

Bring your own code

Bring **your** own code

Semgrep on decompiled code

Automating binary vulnerability discovery with Ghidra and Semgrep

TL;DR

I've released some tools to help **automate vulnerability discovery tasks via static analysis techniques**:

- **Rhabdomancer** is a simple **Ghidra** script that locates all **calls to potentially insecure API functions** in a binary. Auditors can backtrace from these candidate points to find pathways allowing access from untrusted input.
- **Haruspex** is another Ghidra script that is able to **extract all pseudo-code** generated by the Ghidra decompiler in a format that should be suitable to be imported into an IDE, such as **VS Code**, or parsed by static analysis tools, such as **Semgrep**.
- **My Semgrep rules** are specially crafted to help auditors **identify potential bugs and locate hotspots in C/C++ code** on which to focus their attention.

Semgrep is a static analysis tool that works on source code, but thanks to Haruspex we can leverage its power also against **closed source binaries**.

<https://security.humanativaspait/automating-binary-vulnerability-discovery-with-ghidra-and-semgrep/>



Finding bugs and scaling your security program with Semgrep

BSides Milano 2023

Pieter De Cremer & Claudio Merloni

More exercises and examples

[enno - YouTube](#)

[0xDC0DE - YouTube](#)

[Full day workshop with additional exercises](#)

Security researchers at Semgrep

semgrep



Pieter De Cremer
0xDC0DE



Claudio Merloni
p4p3r



Taint mode enables intra-procedural taint tracking

- *mode: taint => intra-procedural* taint tracking!
- Allow producing matches only if data flows from source to sink
- Supports definition of sanitizers and taint propagators
- Sources, sinks and sanitizers can be any pattern that Semgrep can match!
- <https://semgrep.dev/docs/writing-rules/data-flow/taint-mode/>
- *Semgrep Pro Engine*, an extension to Semgrep that enables intra-file and intra-procedural taint tracking is also available:
<https://semgrep.dev/docs/semgrep-code/semgrep-pro-engine-intro/>

An example of taint mode rule

```
1 id: taint-rule
2 languages:
3   - js
4 message: Careful with taint!
5 mode: taint
6 pattern-sanitizers:
7   - not_conflicting: true
8     patterns:
9       - pattern: $F(...)
10      - pattern-not: some_unsafe_function(...)
11 pattern-sinks:
12   - pattern: sink(...)
13 pattern-sources:
14   - pattern: source
15 severity: WARNING
```

```
1 var x = some_safe_function(source);
2 // ok: taint-rule
3 sink(x);
4
5 var y = some_unsafe_function(source);
6 // ruleid: taint-rule
7 sink(y)
```

Symbolic propagation

```
1 id: test
2 options:
3 | symbolic_propagation: false
4 pattern: pandas.DataFrame(...).index.set_value(...)
5 message: Semgrep found a match
6 languages:
7 | - python
8 severity: WARNING
```

```
1 import pandas
2
3 def test1():
4     # ruleid: test
5     pandas.DataFrame(x).index.set_value(a, b, c)
6
7 def test2():
8     df = pandas.DataFrame(x)
9     ix = df.index
10    # ruleid: test
11    ix.set_value(a, b, c)
```

Symbolic propagation

The following is (at the moment) the default analysis behavior.

```
1 id: test
2 options:
3 | symbolic_propagation: false
4 pattern: pandas.DataFrame(...).index.set_value(...)
5 message: Semgrep found a match
6 languages:
7 | - python
8 severity: WARNING
```

```
1 import pandas
2
3 def test1():
4     # ruleid: test
5     pandas.DataFrame(x).index.set_value(a, b, c)
6
7 def test2():
8     df = pandas.DataFrame(x)
9     ix = df.index
10    # ruleid: test
11    ix.set_value(a, b, c)
```

Symbolic propagation

Symbolic propagation is a generalization of constant propagation.
It allows Semgrep to perform matching even in the presence of variable assignments.

```
1 id: test
2 options:
3 | symbolic_propagation: true
4 pattern: pandas.DataFrame(...).index.set_value(...)
5 message: Semgrep found a match
6 languages:
7 | - python
8 severity: WARNING
```

```
1 import pandas
2
3 def test1():
4     # ruleid: test
5     pandas.DataFrame(x).index.set_value(a, b, c)
6
7 def test2():
8     df = pandas.DataFrame(x)
9     ix = df.index
10    # ruleid: test
11    ix.set_value(a, b, c)
```

Metavariable-pattern

- Run one or more Semgrep patterns on the code matched by metavariables

```
7 # ruleid:test-mvp-dots-mvar          patterns:  
8 def g():  
9     foo(3)  
10    meh(2)  
11    baz(1)  
12  
13 # OK:test-mvp-dots-mvar          - pattern: |  
14 def h():  
15     baz(1)  
16     bar(2)  
17     foo(3)                         def $F():  
                                         ...BODY  
                                         - metavariable-pattern:  
                                         metavariable: ...BODY  
                                         patterns:  
                                         - pattern: |  
                                         foo(...)  
                                         ...  
                                         baz(...)
```

Metavariable-pattern can be nested

```
1 # ruleid:test-mvp-nested      patterns:
2 foo(bar(a + b), 2, 3)          - pattern: foo($P, ...)
3 # OK:test-mvp-nested          - metavariable-pattern:
4 foo(a + b, 2, 3)              metavariable: $P
5 # OK:test-mvp-nested          patterns:
6 bar(a + b)                   - pattern: |
                                bar($X)
                                - metavariable-pattern:
                                metavariable: $X
                                patterns:
                                - pattern: ... + ...
```

Metavariable-pattern allows to scan polyglot files 💪

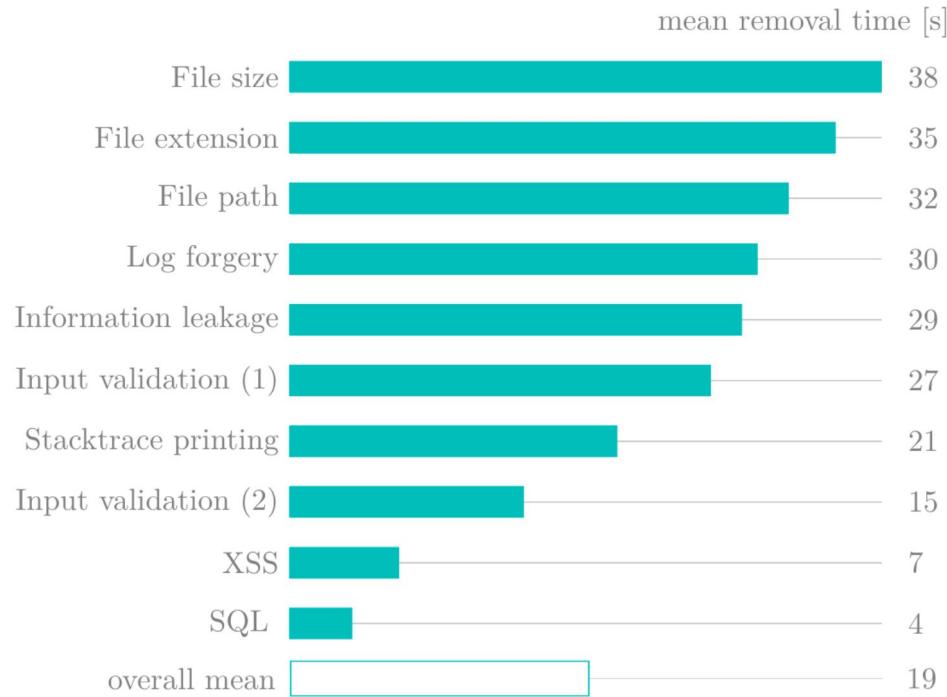
Applications: JS in HTML, template files, React/VueJS, infra as code, Dockerfiles

```
1 <html>
2
3 <script>
4 var x = "foo";
5 console.log(x);
6 </script>
7
8 </html>
```

patterns:

- pattern: <script ...>\$...JS</script>
- metavariable-pattern:
 - metavariable: \$...JS
 - language: javascript
- patterns:
 - pattern: console.log(...);

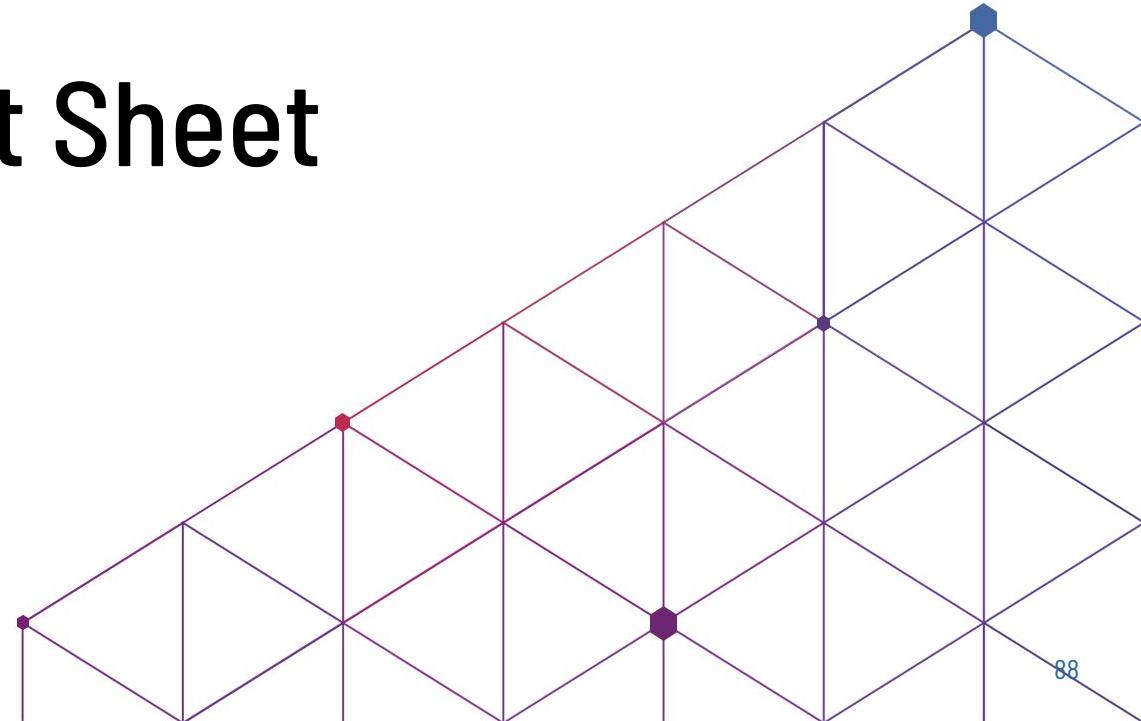
Q/A slides



Q/A slides

- **patterns:**
 - **pattern:** |
\$F(...)
 - **metavariable-pattern:**
metavariable: \$F
patterns:
 - **pattern-not-regex:** YOUR_REGEX

Semgrep Cheat Sheet



#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.dev/s/DK5q>)

#2: '...' ellipsis operator

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

Will match

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

Will match

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

Will match

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

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

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

'...' can match sequences of:

- Arguments, parameters
- Characters
- Statements

#3 Metavariables (part 1)

<code>foo(\$x, 2)</code>	Will match	<code>foo(1, 2)</code>
<code>if \$E: foo()</code>	Will match	<code>if x > 2: foo()</code>
<code>if \$x > \$y: \$s</code>	Will match	<code>if var > 2: return 1</code>
<code>\$F(1, 2)</code>	Will match	<code>foo(1, 2)</code>

- **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 $E:  
  $S  
else:  
  $S
```

Will match

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

Will match

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

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

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

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

Rule Syntax

Semgrep  Registry Playground App Pricing Docs

Docs home

Semgrep

Getting started with Semgrep
CLI

Running rules

Running rules

Managing findings

Writing rules

[Home](#) > [Semgrep](#) > [Writing rules](#) > [Rule syntax](#)

Rule syntax



TIP

Getting started with rule writing? Try the [Semgrep Tutorial](#) 

This document describes Semgrep's YAML rule syntax.

<https://semgrep.dev/docs/writing-rules/rule-syntax/>

<https://semgrep.dev/docs/writing-rules/pattern-syntax/>

Tutorial: <https://semgrep.dev/learn>

Operators

pattern

patterns

pattern-either

pattern-regex

pattern-not-regex

metavariable-regex

metavariable-pattern

Nested language

metavariable-comparison

pattern-not

pattern-inside

pattern-not-inside

pattern-where-python

Metavariable matching

Metavariables in logical ANDs

Metavariables in logical ORs

Metavariables in complex logic

options

fix

metadata

paths