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

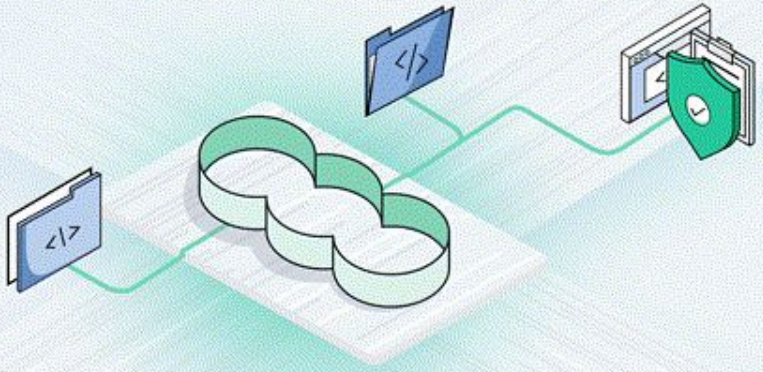
Bring your own code

POWERED BY SEMGREP OPEN SOURCE

Code Analysis at Ludicrous Speed

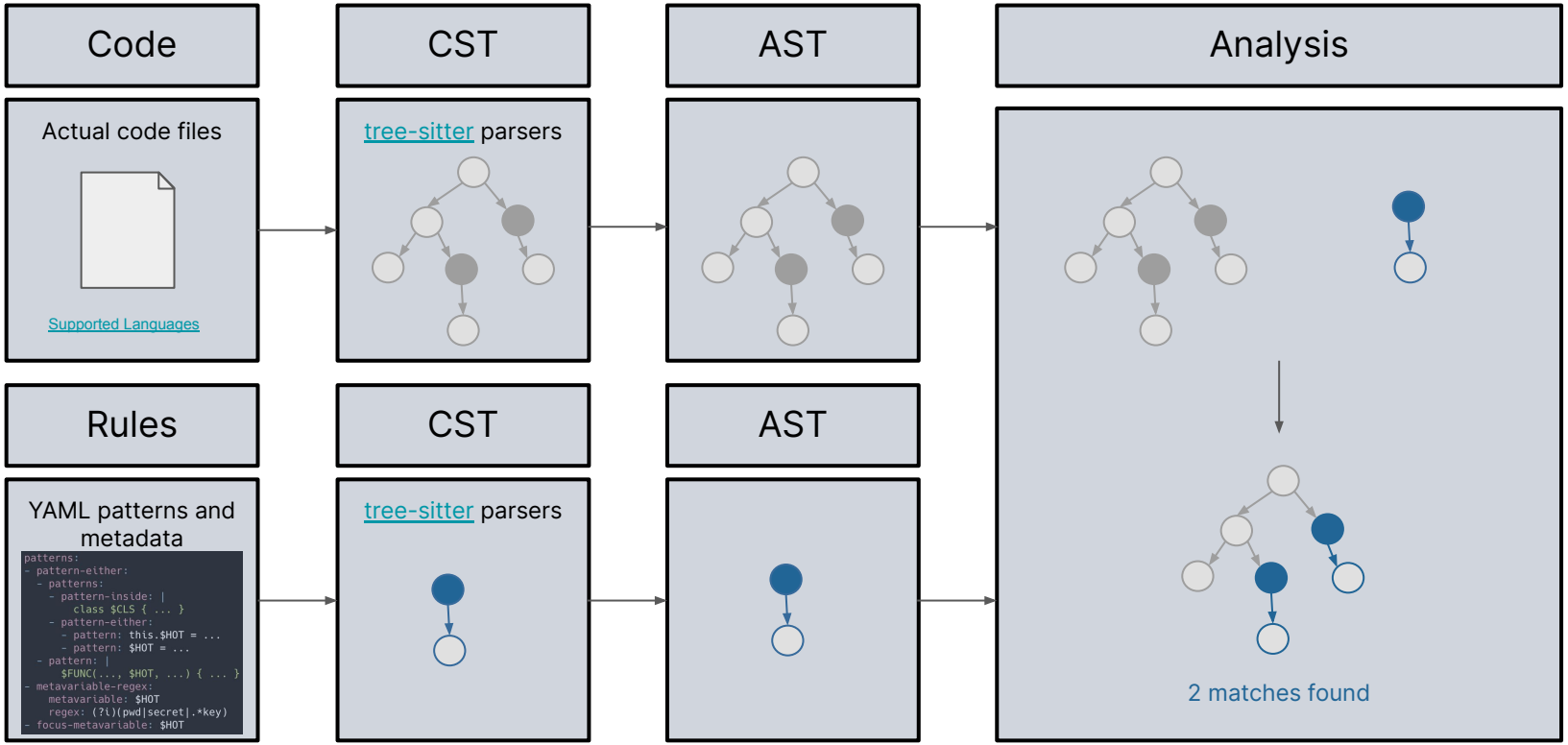
Find bugs and dependency vulnerabilities, run security scans in CI, and enforce standards across your organization.

Start scanning for free | Book a demo

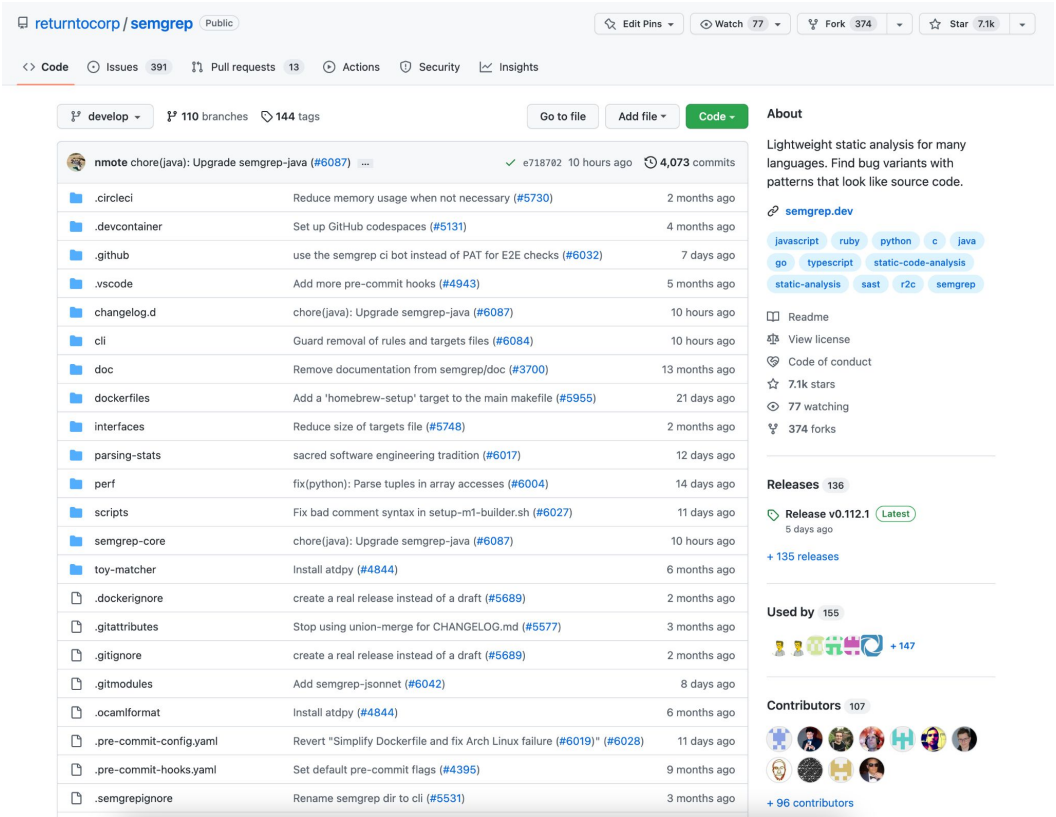


one medical PLAID HashiCorp GitLab slack

Analysis architecture

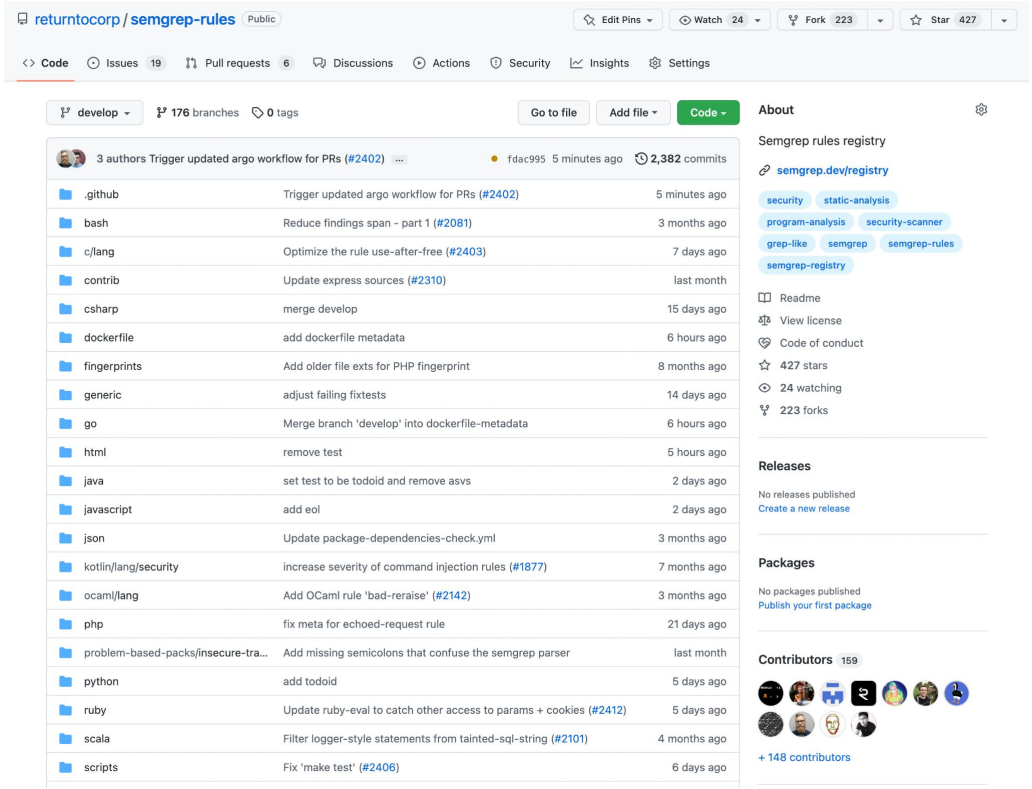


Semgrep is open source



https://github.com/returntocorp/semgrep

Semgrep rules are open source



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.

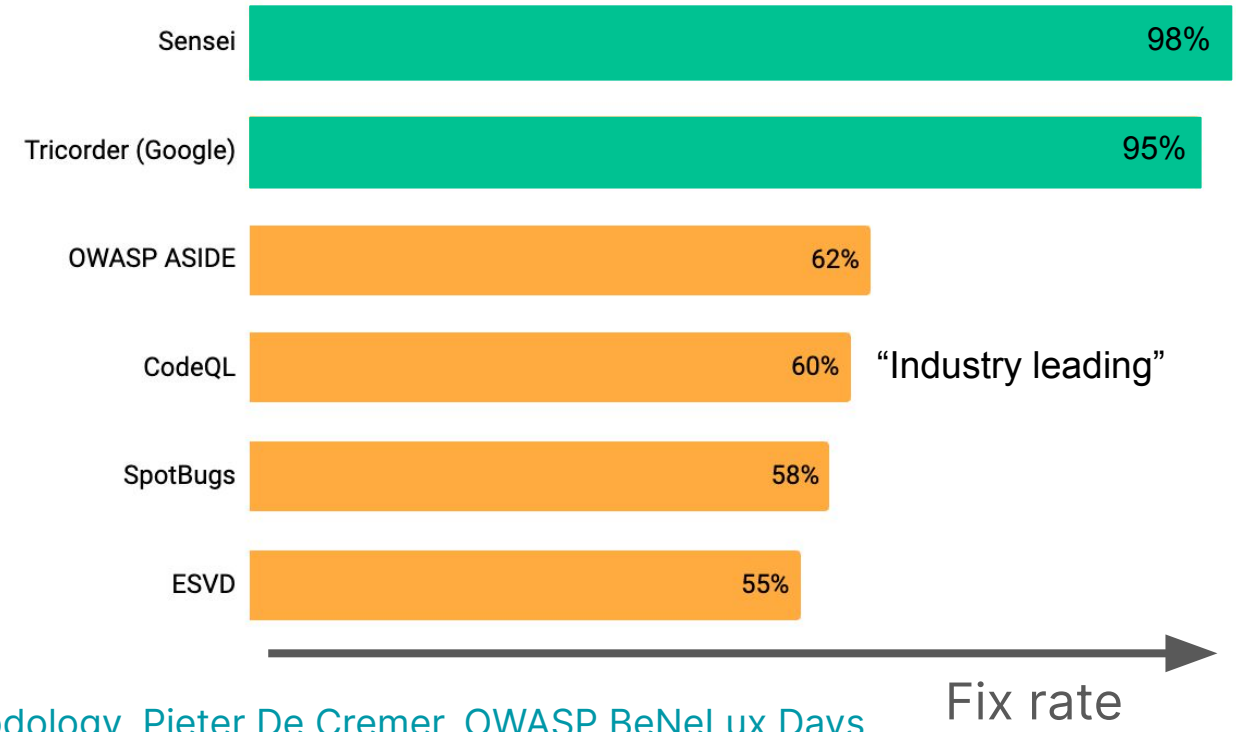
The first stream focuses on establishing a common security baseline to automatically detect so-called "low hanging fruit". Progressively customize the automated tests for each application and increase their frequency of execution to detect more bugs and regressions earlier, as close as possible to their 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

Customized rules

No customized rules



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

Tricorder: Building a Program Analysis Ecosystem, Sadowski et. al, Google

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

Solution: https://semgrep.dev/s/DK5y

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/gNJn

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 inhereit 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/4LIx

Exercise

Check RSA encryption key strength < 2048

https://semgrep.dev/s/RLzv

```go
package main

import (
    "crypto/rand"
    "crypto/rsa"
    "fmt"
)

func main() {
    // ruleid: use-of-weak-rsa-key
    pvk, err := rsa.GenerateKey(rand.Reader, 1024)
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println(pvk)

    // ok: use-of-weak-rsa-key
    pvk, err := rsa.GenerateKey(rand.Reader, 2048)
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println(pvk)
}
```

Need help?

Hint 1

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

Scaling your security program with Semgrep

Semgrep

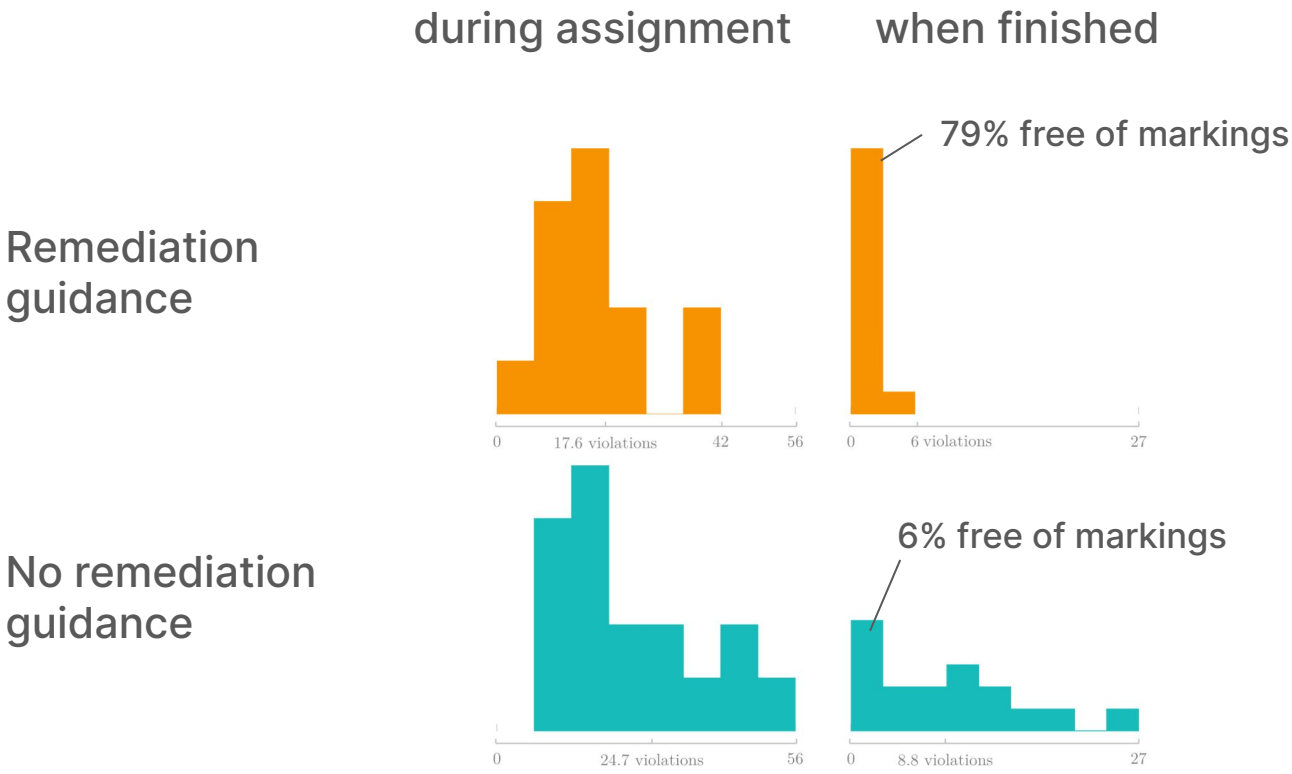
Customizing rules

Remediation guidance

More analysis features

Bring your own code

Remediation guidance improves fix rate



Remediation guidance improves development speed

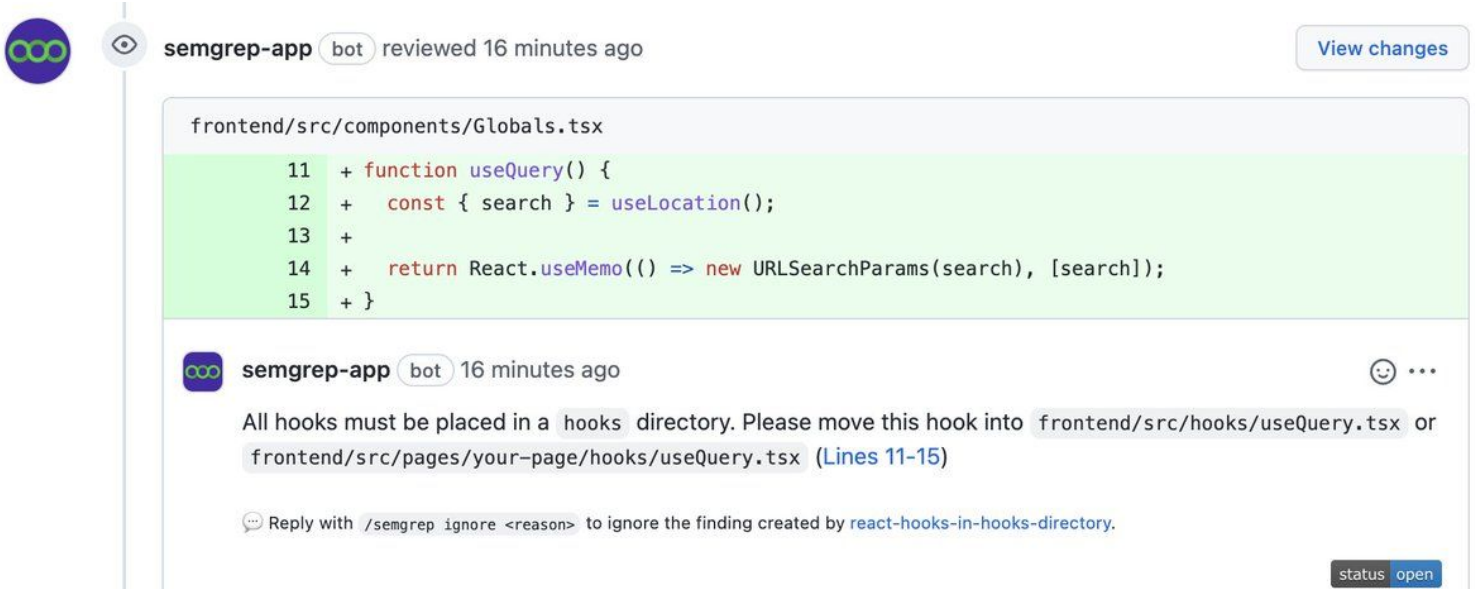
Subject group	**Mean remediation time**
Test group	19.10 s
Control group	129 s (= 2.15 min)

Providing remediation guidance greatly reduces effective false positives

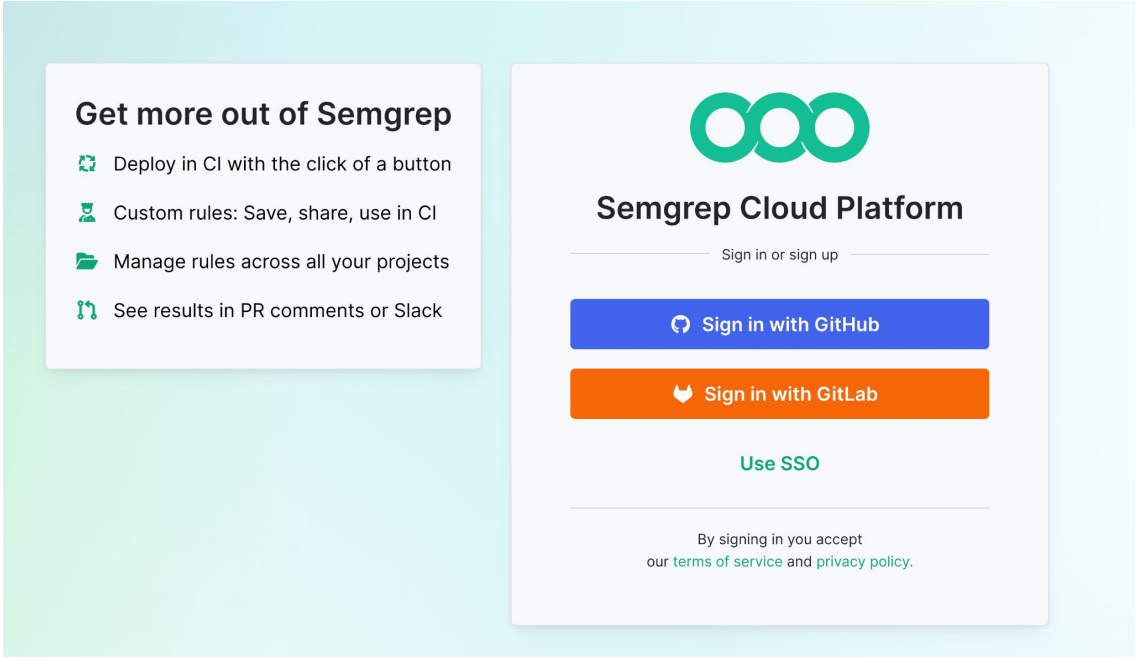
Rules with autofix have 50% higher fix rate

compared to rules without autofix

Rules with an autofix create pull request comments in CI



Semgrep App allows you to scan your Git repositories



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

https://semgrep.dev/blog/2023/semgrep-vscode-extension

Demo

Writing an autofix: secure cookies

https://semgrep.dev/s/Rvov

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

Matches

⬆ Line 9

Semgrep found a match

Apply autofix

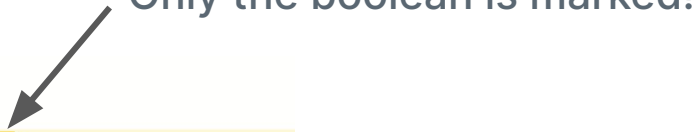
```
authtkt = AuthTktCookieHelper(secret="test", httponly=True)
```

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

Exercise

Secure cookies part 2: replace `httponly=False`

https://semgrep.dev/s/pDxJ




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

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

Semgrep has AST-based autofix in Python and JavaScript

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

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

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

Semgrep has AST-based autofix in Python and JavaScript

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

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

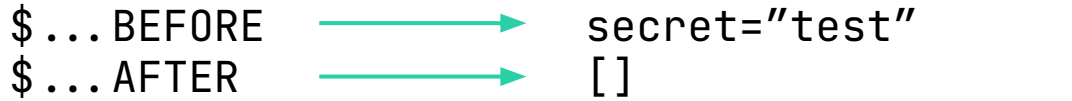
```
$...BEFORE  ⟶
$...AFTER   ⟶
```

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

Semgrep has AST-based autofix in Python and JavaScript

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

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

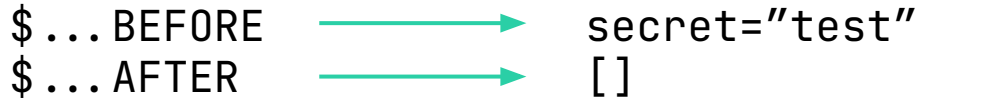


https://semgrep.dev/blog/2022/autofixing-code-with-semgrep

Semgrep has AST-based autofix in Python and JavaScript

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

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



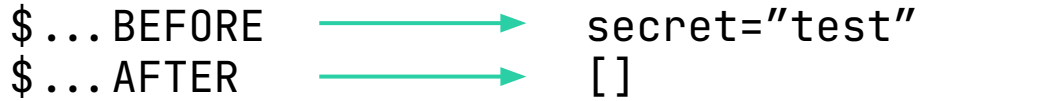

```
AuthTktCookieHelper($...BEFORE, $...AFTER)
```

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

Semgrep has AST-based autofix in Python and JavaScript

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

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




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

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

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”, )
```

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

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

Semgrep has AST-based autofix in Python and JavaScript

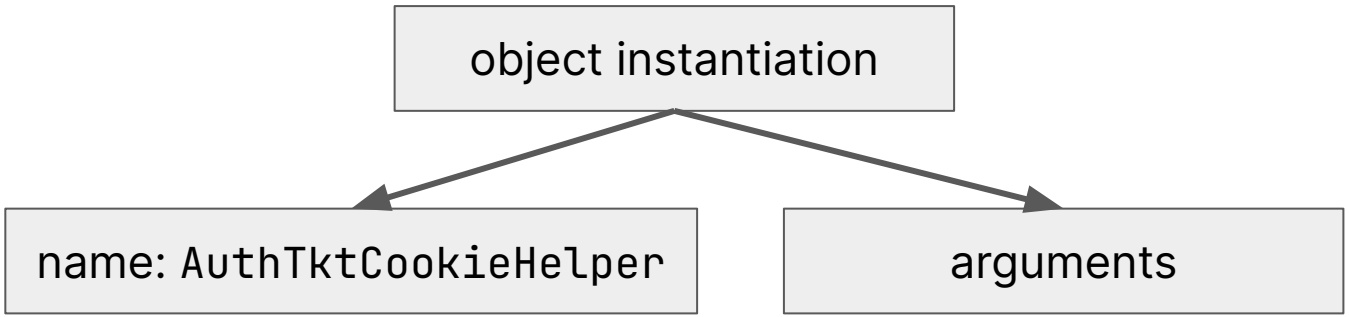
```
pattern: AuthTktCookieHelper($...BEFORE, httponly=False, $...AFTER)
fix: AuthTktCookieHelper($...BEFORE, $...AFTER)
```

object instantiation

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

Semgrep has AST-based autofix in Python and JavaScript

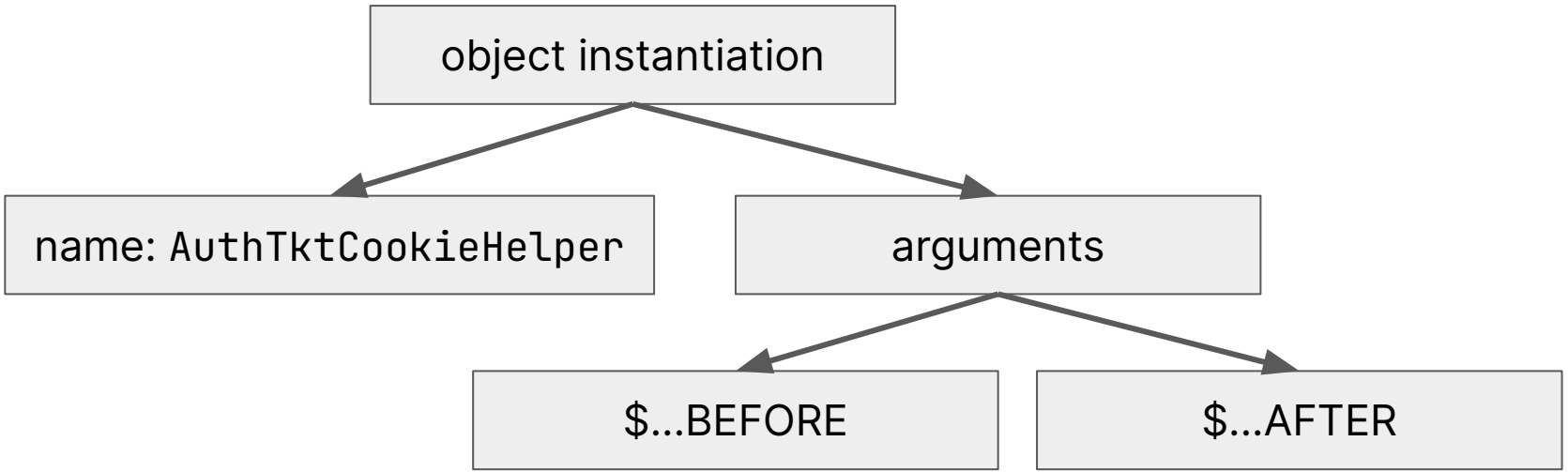
```
pattern: AuthTktCookieHelper($...BEFORE, httponly=False, $...AFTER)
fix: AuthTktCookieHelper($...BEFORE, $...AFTER)
```



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

Semgrep has AST-based autofix in Python and JavaScript

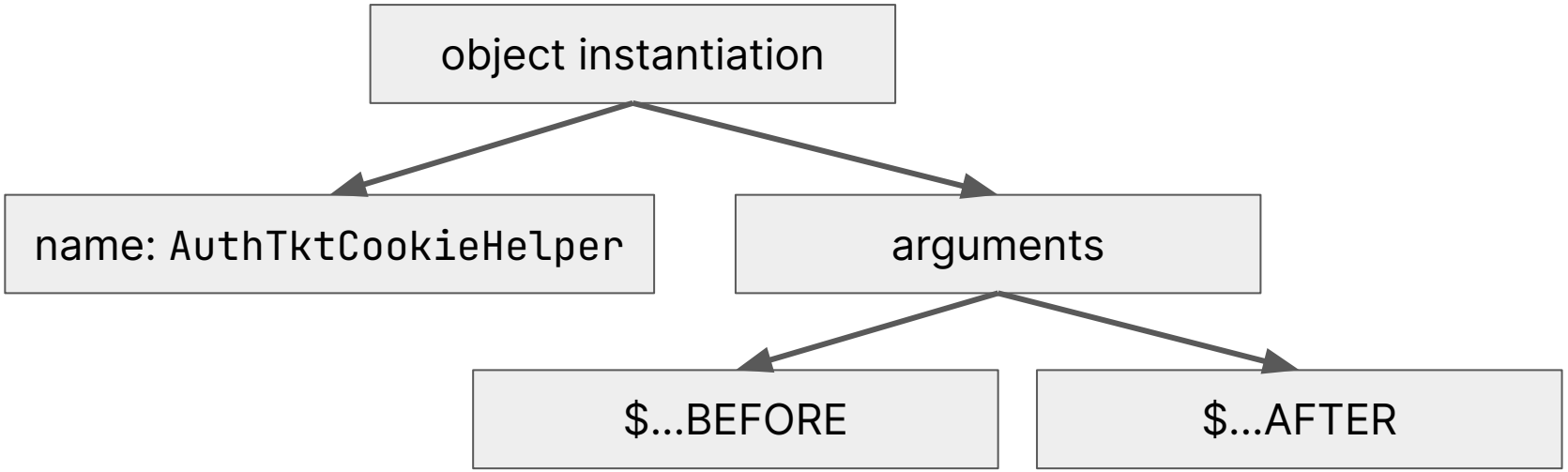
```
pattern: AuthTktCookieHelper($...BEFORE, httponly=False, $...AFTER)
fix: AuthTktCookieHelper($...BEFORE, $...AFTER)
```



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

Semgrep has AST-based autofix in Python and JavaScript

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

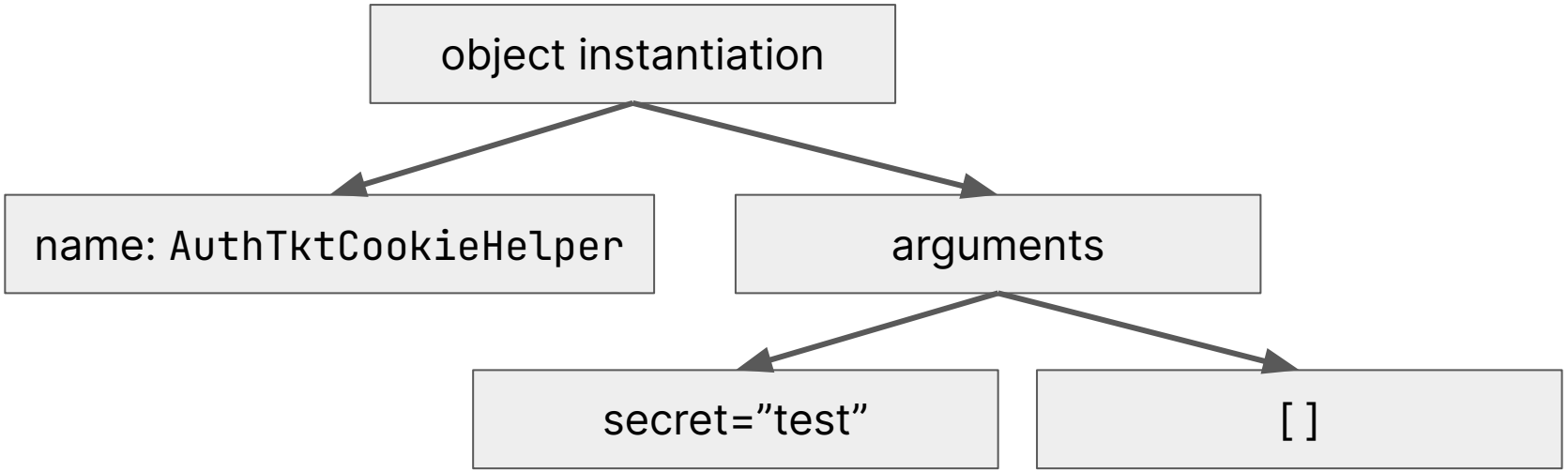


https://semgrep.dev/blog/2022/autofixing-code-with-semgrep

```
$...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

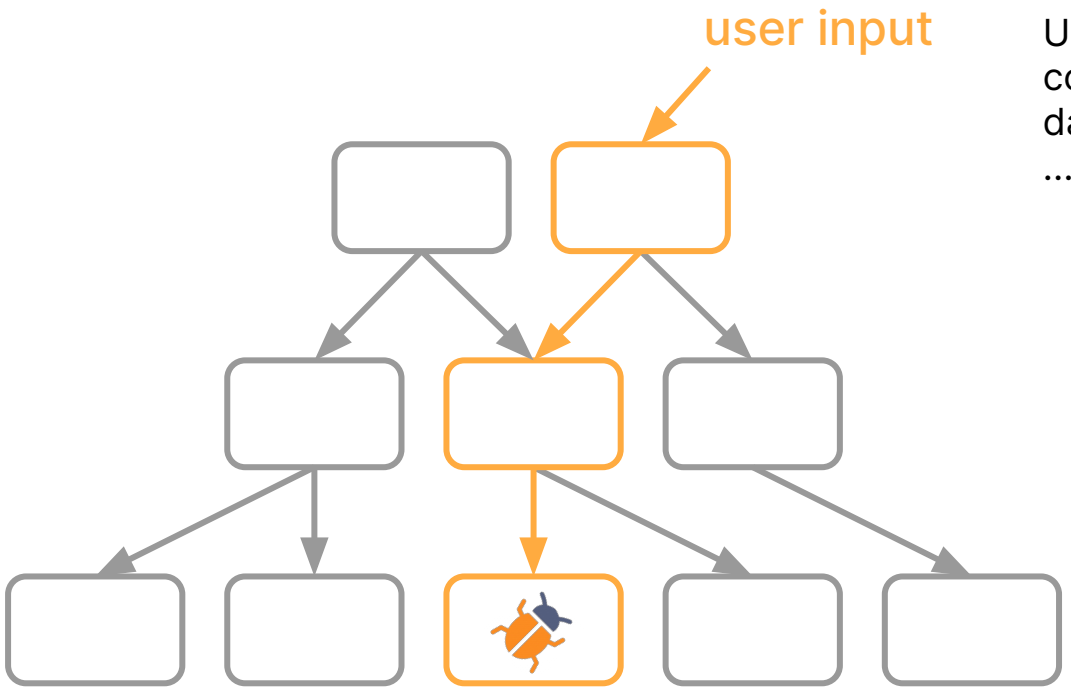
Customizing rules

Remediation guidance

More analysis features

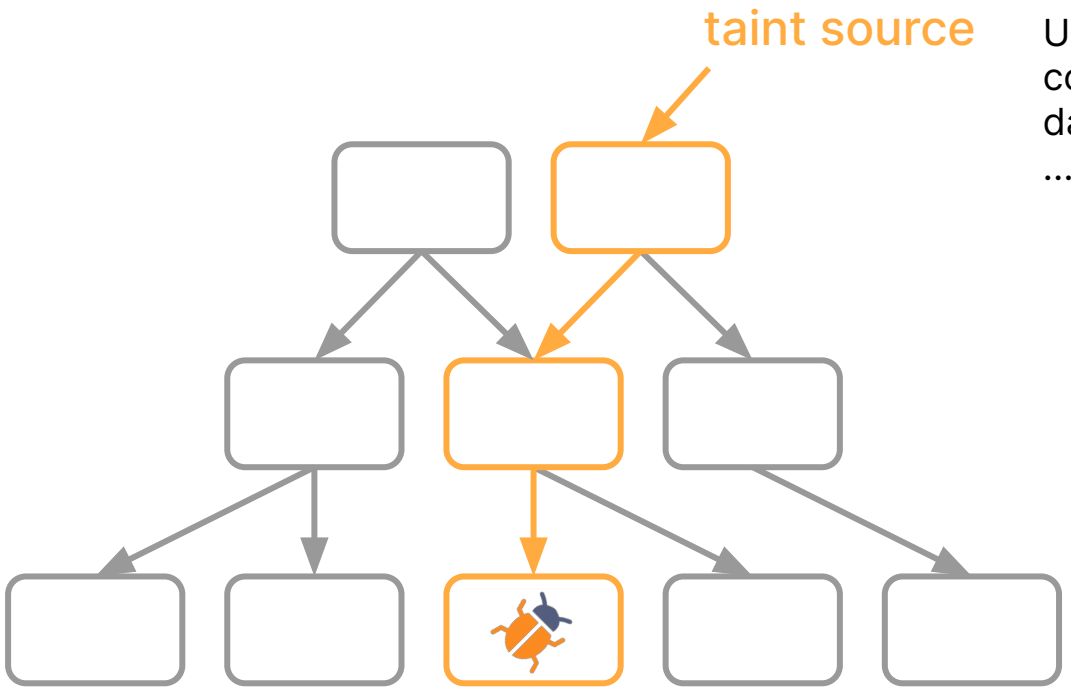
Bring your own code

Dataflow analysis terminology



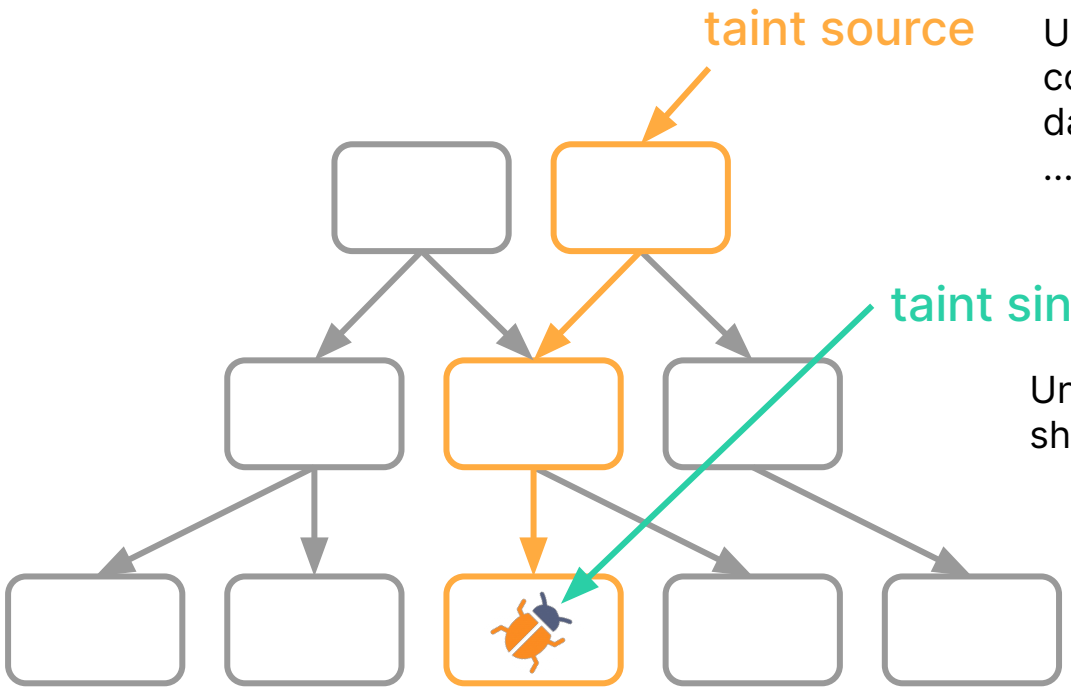
URL parameters
cookies
data from a third-party service
...

Dataflow analysis terminology



URL parameters
cookies
data from a third-party service
...

Dataflow analysis terminology



URL parameters
cookies
data from a third-party service
...

Unparameterized SQL query,
shell exec()

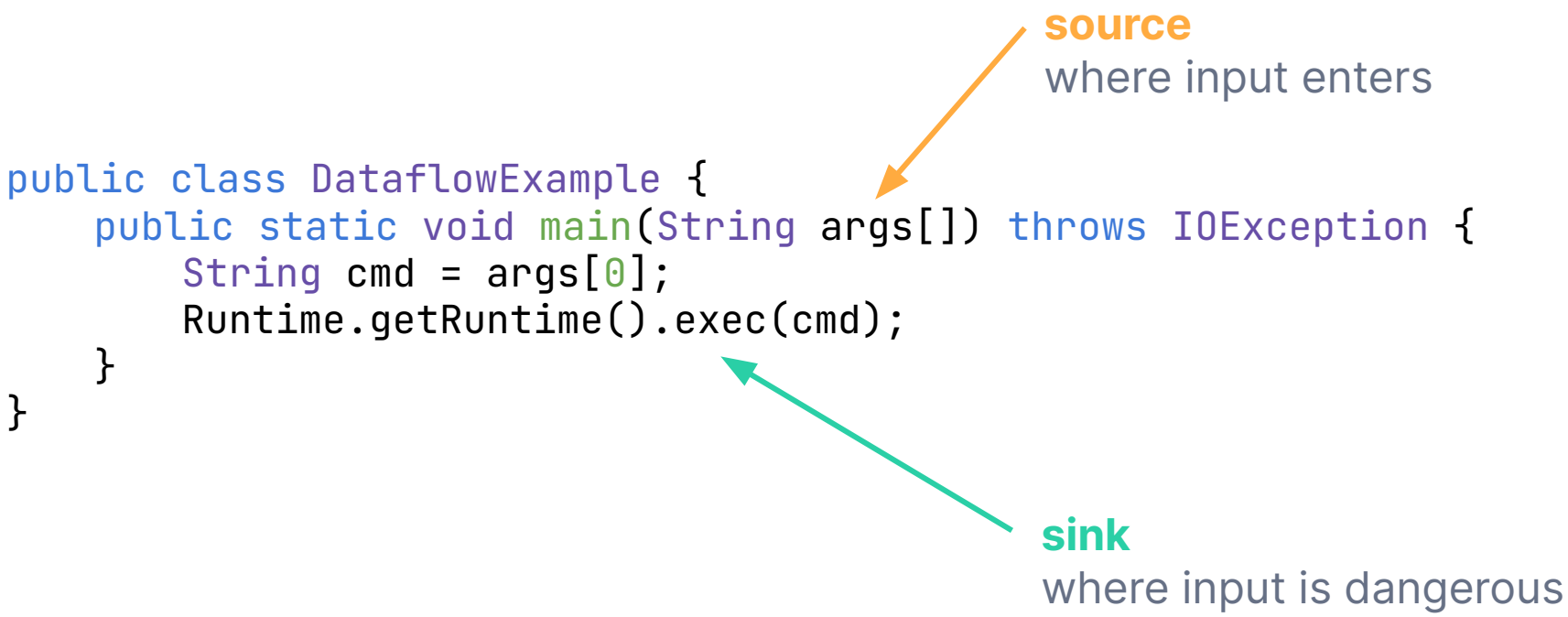
Tainted data flows from source to sink

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



Examples of sources and sinks

Sources

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

Sinks

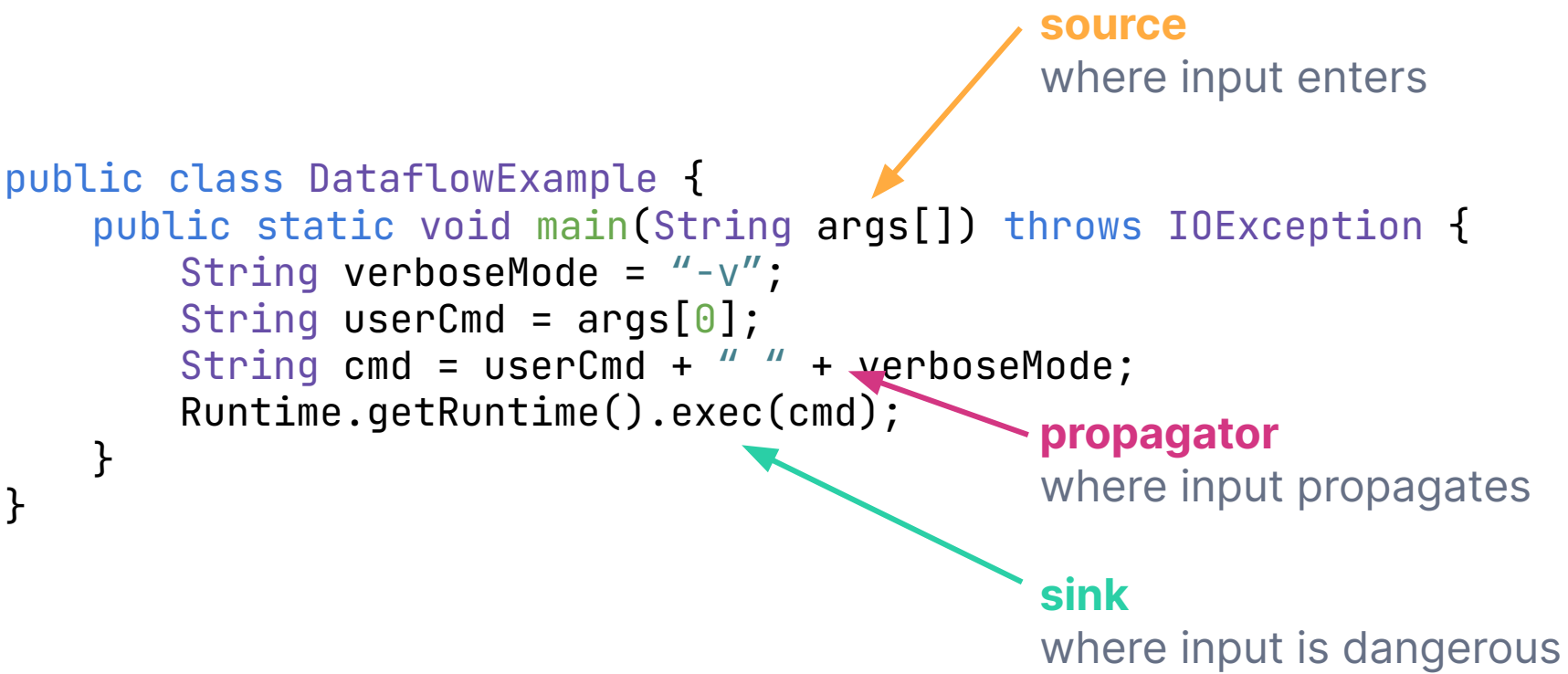
- Unparameterized SQL queries (SQLi)
- Web server sending html content (XSS)
- File names or path names(arbitrary file read/write)
- Shell exec’d commands (command injection)

“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



Tainted data can be sanitized before it reaches the sink

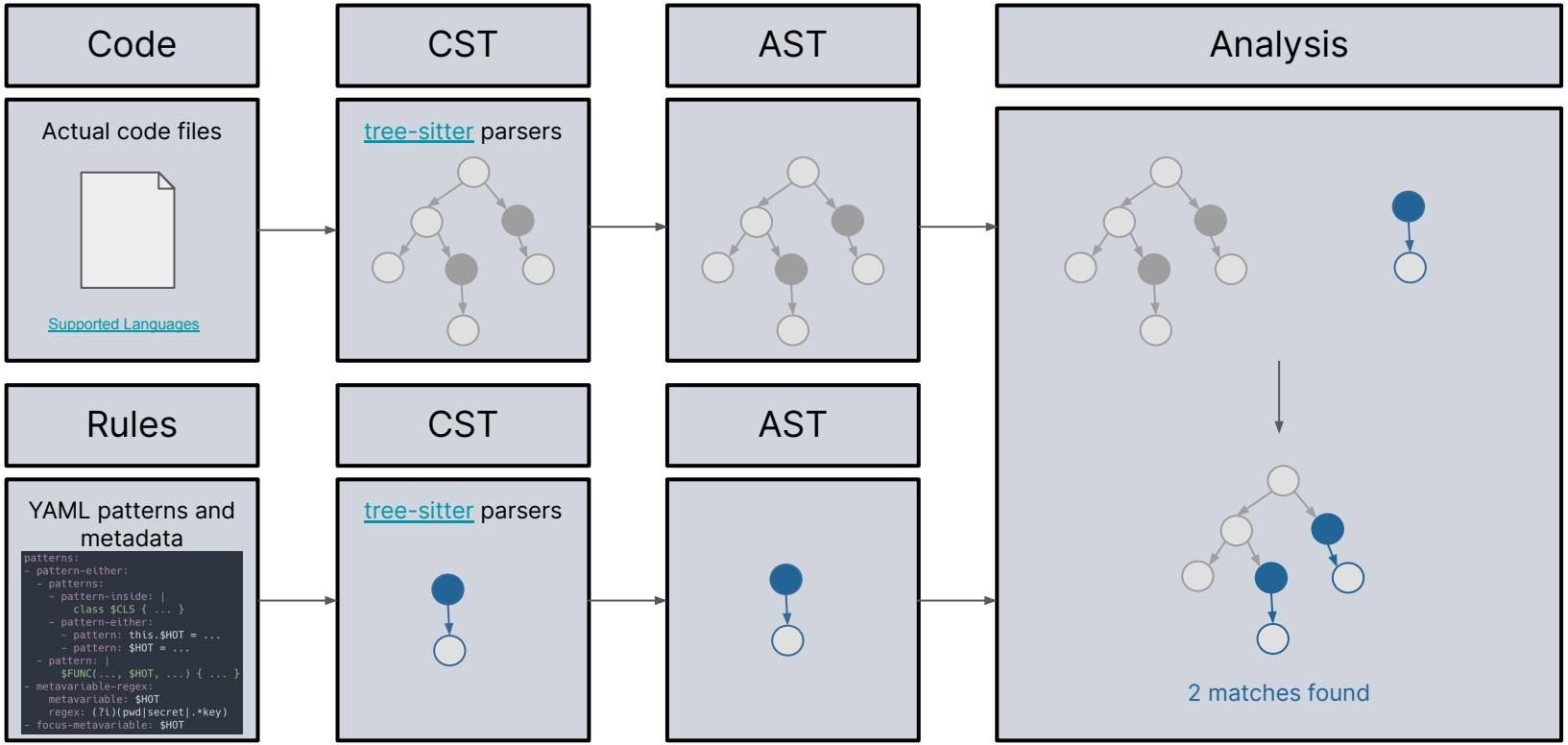
```
public class DataflowExample {
    public static void main(String args[]) throws IOException {
        String baseCmd = “rm”;
        String sanitizedPath = filterFilePath(args[0])
        String cmd = baseCMD + “ “ + sanitizedPath;
        Runtime.getRuntime().exec(cmd);
    }
}
```

source
where input enters

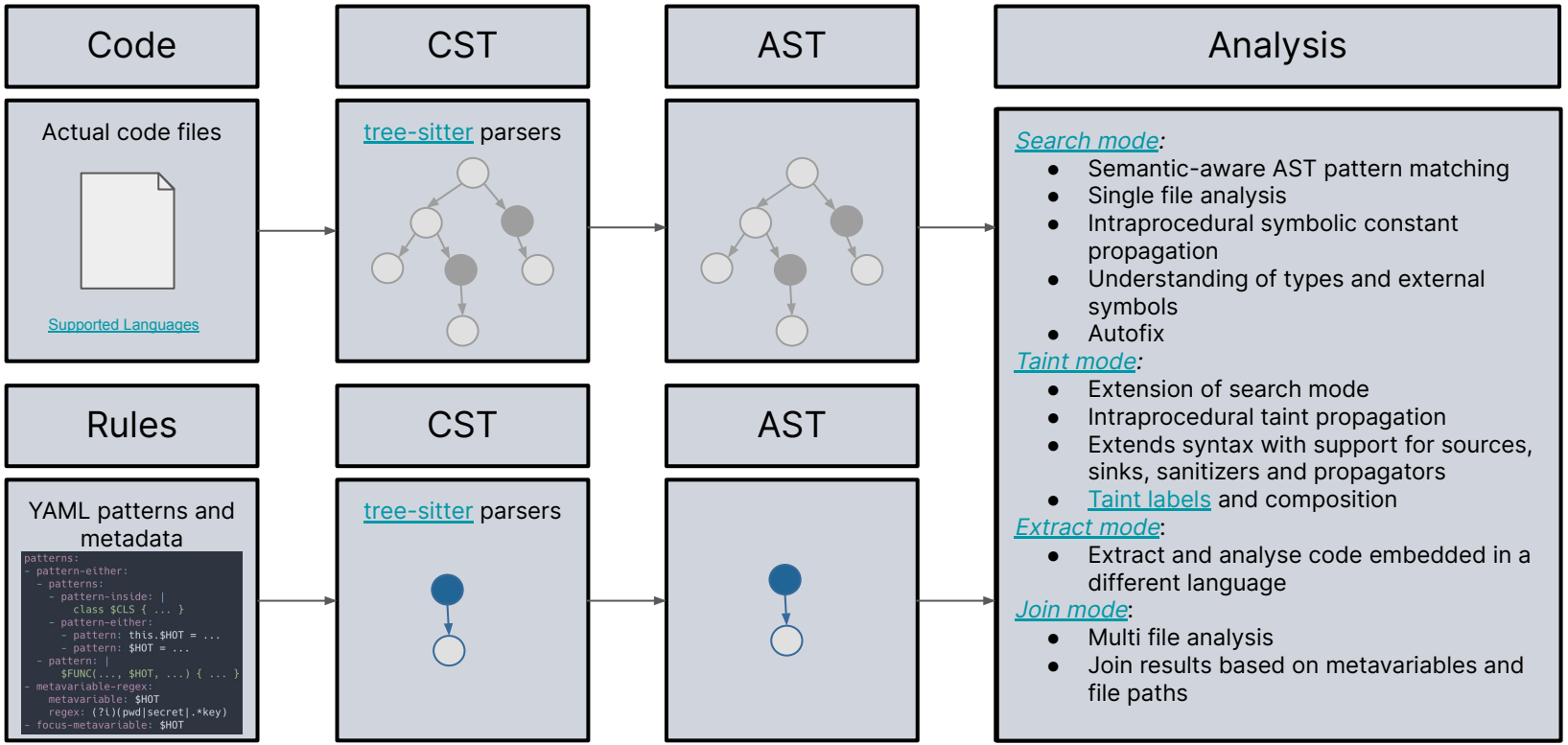
sanitizer
where input is sanitized

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
```

Solution: https://semgrep.dev/playground/s/KPO9

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/8Qgq

```
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 });
```

A possible solution: https://semgrep.dev/s/Pygp

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)
```

A possible solution: https://semgrep.dev/playground/s/PA3e

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)
21          # ok: tainted-ruamel
22          data = yml.load(s)
23
24          yml = ruamel.yaml.YAML(typ='safe')
25          # ok: tainted-ruamel
26          data = yml.load(tainted)
27
28          # The default loader (typ='rt') is a direct derivative of the safe loader
29          yml = ruamel.yaml.YAML(typ='rt')
30          # ok: tainted-ruamel
31          data = yml.load(tainted)
32
33          yml = ruamel.yaml.YAML()
34          # ok: tainted-ruamel
35          data = yml.load(tainted)
```

Solution: https://semgrep.dev/playground/s/WlLj

New simplified rule syntax

```
patterns
pattern-either
pattern-inside
pattern-regex
```

```
all
any
inside
regex
```

https://youtu.be/dZUPjFvknnI

Top level `match` or `taint` keyword

```yaml
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: $SHELL
              regex: ^(sh|bash|ksh|csh|tcsh|zsh)$
```

```python
    # ruleid:subprocess-injection
    subprocess.run("ping "+ ip)

def b(request):
    host = request.headers["HOST"]
    # ruleid:subprocess-injection
    subprocess.run("echo {} > log".format(host))

def d(request):
    cmd = request.POST.get("cmd")
    ip = request.POST.get("ip")
    command = [cmd, ip]
    # ruleid:subprocess-injection
    subprocess.capture_output(command)

def e(request):
    event = json.loads(request.body)
    cmd = event['id'].split()
    # ruleid:subprocess-injection
    subprocess.call([cmd[0], cmd[1], "some", "args"])

def f(request):
    event = json.loads(request.body)
    # ruleid:subprocess-injection
    subprocess.run(["bash", "-c", event['id']], shell=True)

def g(request):
    event = request.body
    python_file = f"""
        print("What is your name?")
```

https://semgrep.dev/playground/s/ZNbo

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.humanativaspa.it/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

```yaml
id: taint-rule
languages:
  - js
message: Careful with taint!
mode: taint
pattern-sanitizers:
  - not_conflicting: true
    patterns:
      - pattern: $F(...)
      - pattern-not: some_unsafe_function(...)
pattern-sinks:
  - pattern: sink(...)
pattern-sources:
  - pattern: source
severity: WARNING
```

```javascript
var x = some_safe_function(source);
// ok: taint-rule
sink(x);

var y = some_unsafe_function(source);
// ruleid: taint-rule
sink(y)
```

Symbolic propagation

```yaml
id: test
options:
  symbolic_propagation: false
pattern: pandas.DataFrame(...).index.set_value(...)
message: Semgrep found a match
languages:
  - python
severity: WARNING
```

```python
import pandas

def test1():
    # ruleid: test
    pandas.DataFrame(x).index.set_value(a, b, c)

def test2():
    df = pandas.DataFrame(x)
    ix = df.index
    # ruleid: test
    ix.set_value(a, b, c)
```

Symbolic propagation

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

```yaml
id: test
options:
  symbolic_propagation: false
pattern: pandas.DataFrame(...).index.set_value(...)
message: Semgrep found a match
languages:
  - python
severity: WARNING
```

```python
import pandas

def test1():
    # ruleid: test
    pandas.DataFrame(x).index.set_value(a, b, c)

def test2():
    df = pandas.DataFrame(x)
    ix = df.index
    # ruleid: test
    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.

```yaml
id: test
options:
  symbolic_propagation: true
pattern: pandas.DataFrame(...).index.set_value(...)
message: Semgrep found a match
languages:
  - python
severity: WARNING
```

```python
import pandas

def test1():
    # ruleid: test
    pandas.DataFrame(x).index.set_value(a, b, c)

def test2():
    df = pandas.DataFrame(x)
    ix = df.index
    # ruleid: test
    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
8   def g():
9       foo(3)
10      meh(2)
11      baz(1)
12
13  # OK:test-mvp-dots-mvar
14  def h():
15      baz(1)
16      bar(2)
17      foo(3)
```

```
patterns:
  - pattern: |
      def $F():
          $...BODY
  - metavariable-pattern:
      metavariable: $...BODY
      patterns:
          - pattern: |
              foo(...)
              ...
              baz(...)
```

Metavariable-pattern can be nested

```
1  # ruleid:test-mvp-nested
2  foo(bar(a + b), 2, 3)
3  # OK:test-mvp-nested
4  foo(a + b, 2, 3)
5  # OK:test-mvp-nested
6  bar(a + b)
```

```
patterns:
  - pattern: foo($P, ...)
  - metavariable-pattern:
      metavariable: $P
      patterns:
          - pattern: |
              bar($X)
          - metavariable-pattern:
              metavariable: $X
              patterns:
                - pattern: ... + ...
```

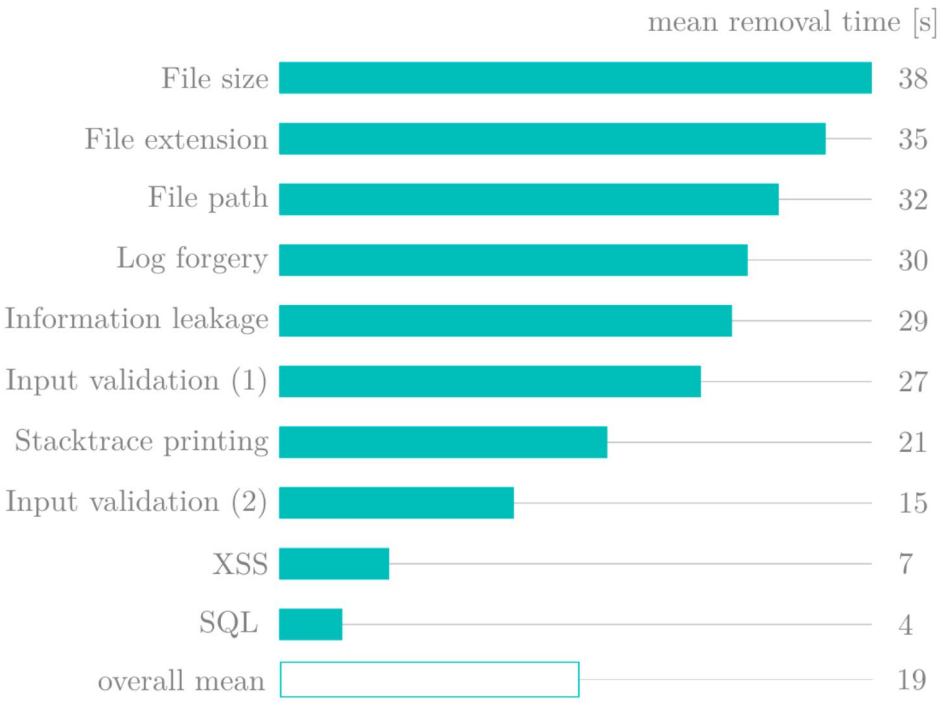
Metavariable-pattern allows to scan polyglot files 💪

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


mean removal time [s]
File size
38
File extension
35
File path
32
Log forgery
30
Information leakage
29
Input validation (1)
27
Stacktrace printing
21
Input validation (2)
15
XSS
7
SQL
4
overall mean
19

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(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:

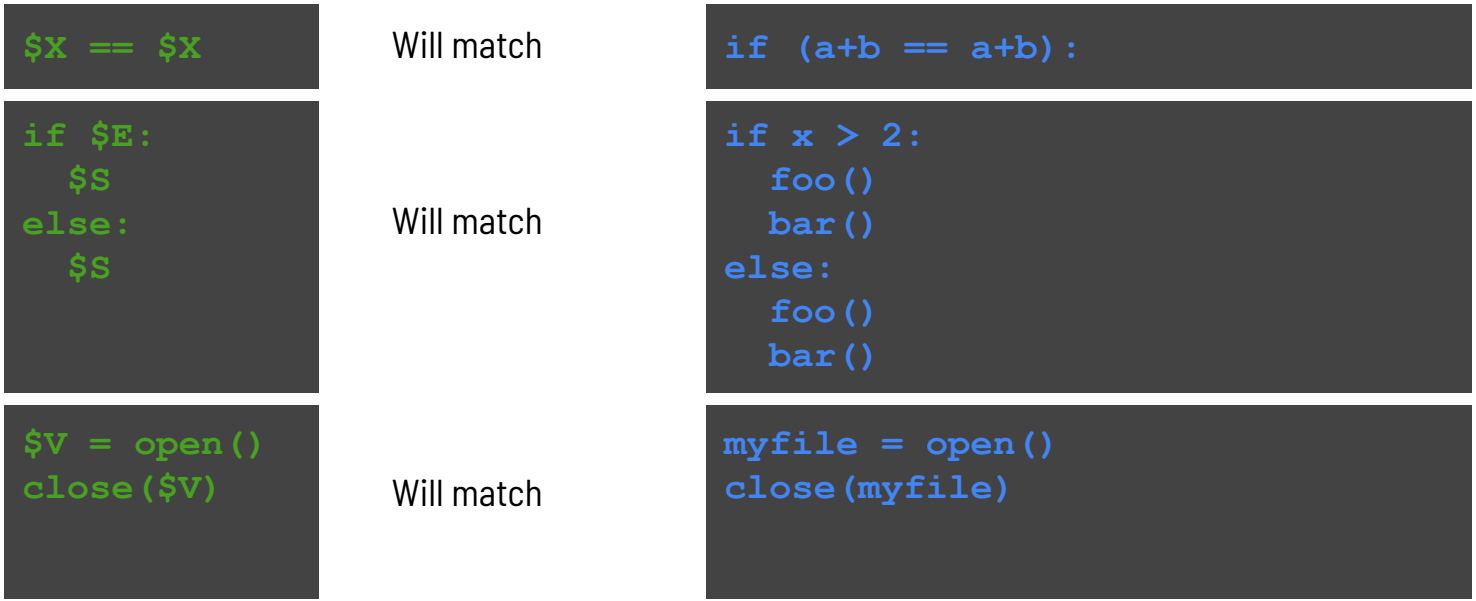
- Arguments, parameters
- Characters
- Statements

#3 Metavariables (part 1)

`foo($X,2)`	Will match	`foo(1,2)`
`if $E:` `  foo()`	Will match	`if x > 2:` `  foo()`
`if $X > $Y:` `  $S`	Will match	`if var > 2:` `  return 1`
`$F(1,2)`	Will match	`foo(1,2)`

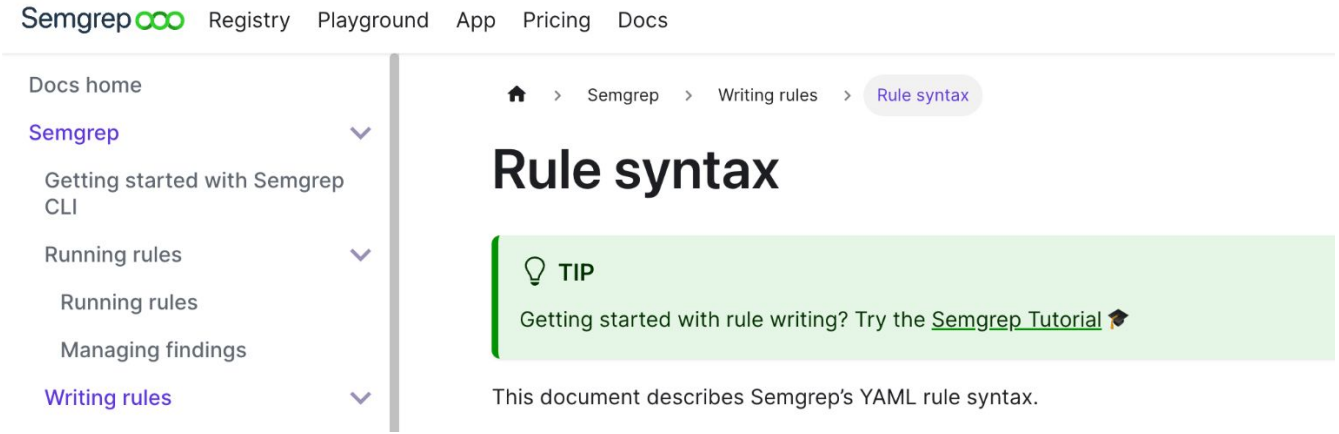
- **Metavariables** start with a $ ($X, $Y, $WHATEVER) **,** contain uppercase ASCII characters
- **Matches:**
  - Expressions (including arguments)
  - Statements
  - Names (functions, fields, etc.)

#3 Metavariables (part 2)



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

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`