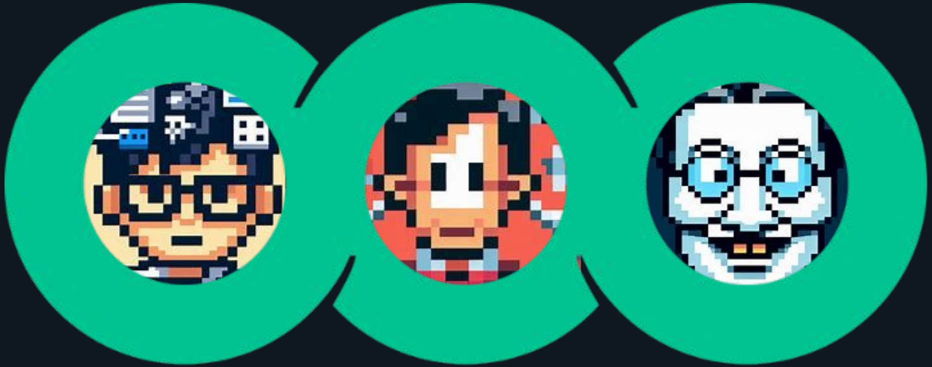
Testing Handbook: Semgrep

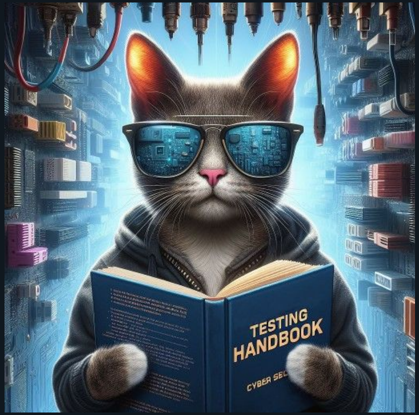
appsec.guide



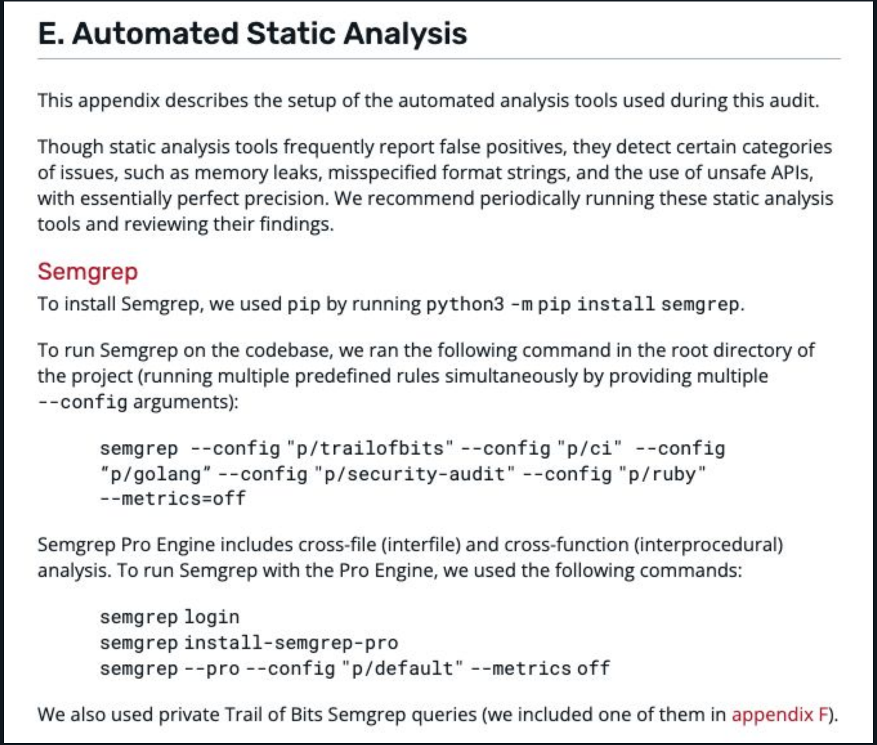
Maciej Domański ~ Matt Schwager ~ Spencer Michaels

Trail of Bits Testing Handbook

- Available at
  - https://appsec.guide
  - https://github.com/trailofbits/testing-handbook
- Written by ToB engineers, peer-reviewed & professionally edited
- Based on actual tools used on security audits
- References high-quality resources & papers
- Created with support from tool developers
- Contains guidance on CI/CD integration
- The repo is public - we appreciate feedback



Semgrep complements our audits



E. Automated Static Analysis

This appendix describes the setup of the automated analysis tools used during this audit.

Though static analysis tools frequently report false positives, they detect certain categories of issues, such as memory leaks, misspecified format strings, and the use of unsafe APIs, with essentially perfect precision. We recommend periodically running these static analysis tools and reviewing their findings.

Semgrep

To install Semgrep, we used `pip` by running `python3 -m pip install semgrep`.

To run Semgrep on the codebase, we ran the following command in the root directory of the project (running multiple predefined rules simultaneously by providing multiple `--config` arguments):

```
semgrep --config "p/trailofbits" --config "p/ci" --config
"p/golang" --config "p/security-audit" --config "p/ruby"
--metrics=off
```

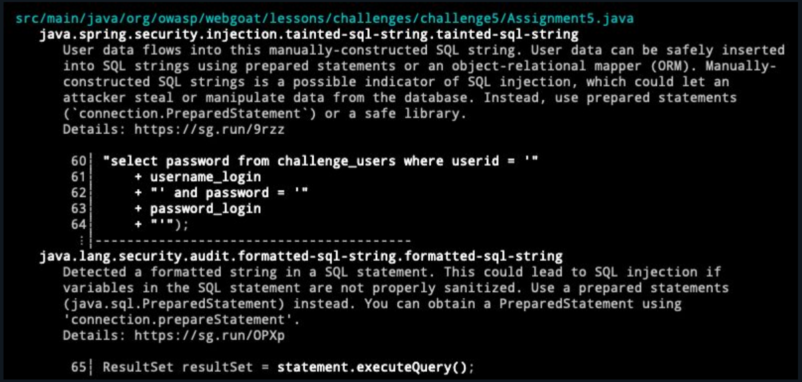
Semgrep Pro Engine includes cross-file (interfile) and cross-function (interprocedural) analysis. To run Semgrep with the Pro Engine, we used the following commands:

```
semgrep login
semgrep install-semgrep-pro
semgrep --pro --config "p/default" --metrics off
```

We also used private Trail of Bits Semgrep queries (we included one of them in appendix F).

Semgrep tutorial in 30s

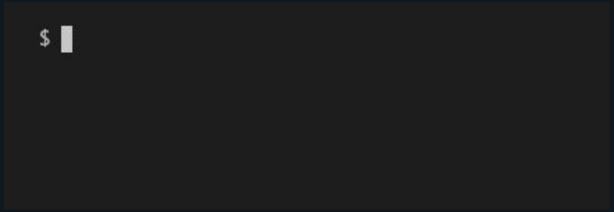
1. Install
   - `$ pip3 install semgrep`
   - Also available via `brew` and `docker`
2. Run rulesets
   - `$ semgrep --config auto /path/to/code`
3. Triage bugs 👾



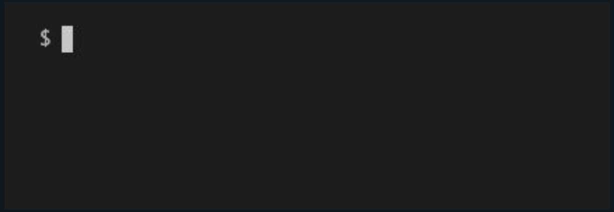
```
src/main/java/org/owasp/webgoat/lessons/challenges/challenge5/Assignment5.java
   java.spring.security.injection.tainted-sql-string.tainted-sql-string
      User data flows into this manually-constructed SQL string. User data can be safely inserted
      into SQL strings using prepared statements or an object-relational mapper (ORM). Manually-
      constructed SQL strings is a possible indicator of SQL injection, which could let an
      attacker steal or manipulate data from the database. Instead, use prepared statements
      (`connection.PreparedStatement`) or a safe library.
      Details: https://sg.run/9rzz

         "select password from challenge_users where userid = '"
             + username_login
             + "' and password = '"
             + password_login
             + "'");
          ⋮┆----------------------------------------
   java.lang.security.audit.formatted-sql-string.formatted-sql-string
      Detected a formatted string in a SQL statement. This could lead to SQL injection if
      variables in the SQL statement are not properly sanitized. Use a prepared statements
      (java.sql.PreparedStatement) instead. You can obtain a PreparedStatement using
      'connection.prepareStatement'.
      Details: https://sg.run/OPXp

         ResultSet resultSet = statement.executeQuery();
```

A quick introduction to Semgrep

- Open-source engine and rules
  - Scan code without sharing it with third parties
- Easy to use
  - One-line installation
  - Custom rules written in the target language
  - Scanning usually takes seconds/minutes (not hours/days)
- Focuses on a single file
  - Cross-file support in Semgrep Pro Engine



```
$ ▮
```



```
$ ▮
```

A quick introduction to Semgrep

No need to build the target code	● Great for security audits on proprietary products
Large repository of existing rulesets	● **Free for general auditing and CI/CD** ● 3rd party rules - universal rules ● “Noisy rules” - “manual” security research
Supports many languages	● **Supports over 30 languages** (C#, Go, Java, Javascript, Kotlin, Ruby, Rust, PHP, C, and more ...) ● Implementation has varied maturity (Generally Available/Beta/Experimental)
Can track data flow	● **Constant propagation** - tracks whether a variable has a constant value at the particular point in the program ● **Taint tracking** - useful for catching injection bugs, such as cross-site scripting because of lack of sanitization
Semgrep Pro Engine (paid)	● **Cross-file support** - for bugs spread across several files ● Pro rules - high confidence rules ● Enterprise languages support (e.g. Apex, Elixir)

A quick introduction to Semgrep

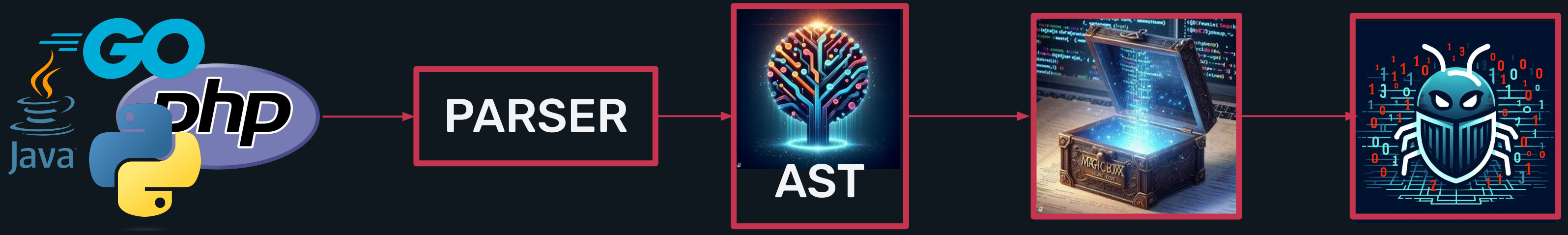
- No need to build the target code
- Large repository of existing rules
- Can track data flow
- Find more *ideal use cases* in our Testing Handbook
    - https://appsec.guide/docs/static-analysis/semgrep/#ideal-use-case

Supported technologies

- Languages
  - Generally, Semgrep is very good for C#, Go, Java, JS, TS, Kotlin, Ruby, PHP
  - Mostly, we don't rely on Semgrep when auditing
    - Solidity
    - C/C++
- Generic
  - Matches generic patterns in languages it does not support (yet)
- Structured data
  - JSON
  - YAML
    - Good for GitHub actions, Docker Compose, Kubernetes config, etc.

A quick introduction to static analysis

- Analyze code without running it
- Usually walks an Abstract Syntax Tree (AST)
  - Intermediate representation
  - May or may not require building the code
- Great for:
  - Bug and error catching
  - Improving code quality
  - Continuous bug prevention (e.g. CI/CD integration)

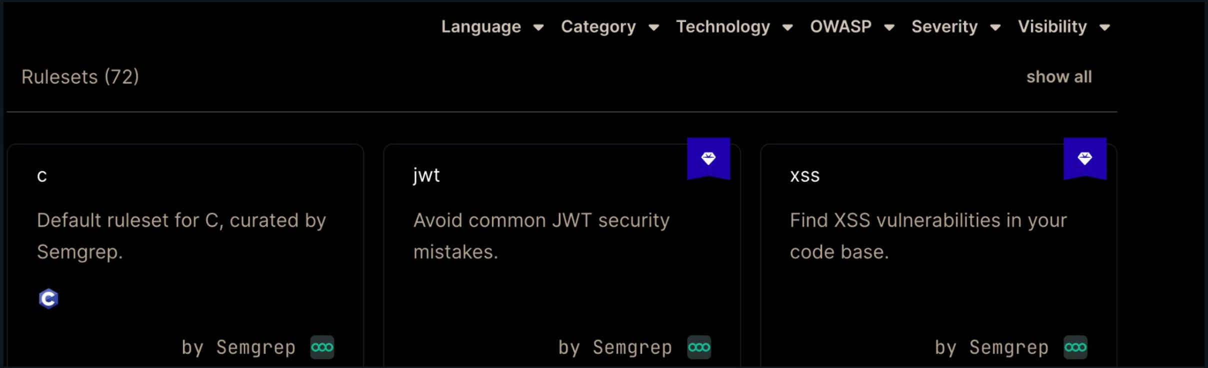


Key Takeaways #1

- Finds bugs
- Just: `$ semgrep --config auto path/`
- Generally finds errors within a single file
- No need to build a code

How we use Semgrep at TRAIL OF BITS

- Quickly identify low-hanging fruits with prepackaged rules
  - Using standard rulesets
    - `$ semgrep --config p/default`
    - `$ semgrep --config p/javascript`
    - See more in Semgrep Registry → https://semgrep.dev/explore

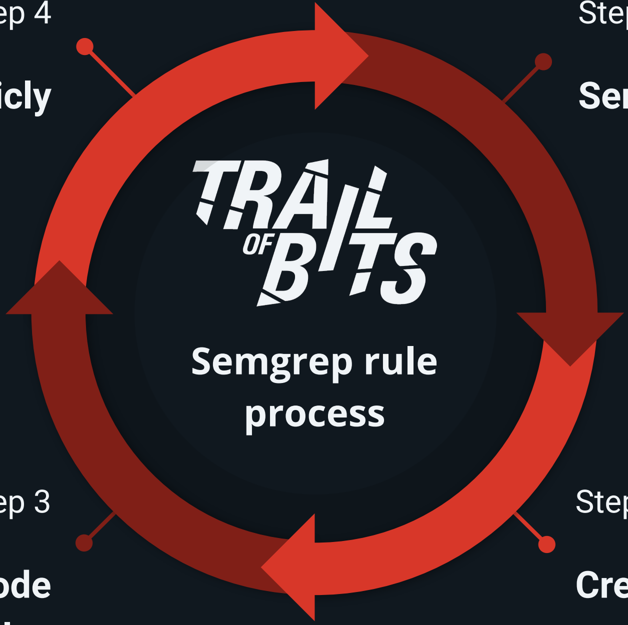


Rulesets (72)

c
Default ruleset for C, curated by Semgrep.
by Semgrep

jwt
Avoid common JWT security mistakes.
by Semgrep

xss
Find XSS vulnerabilities in your code base.
by Semgrep



Step 1

Semgrep rule idea!

Step 2

Create private Trail of Bits rule

Step 3

Refine rule during our code audit process, catch false positives and negatives

Step 4

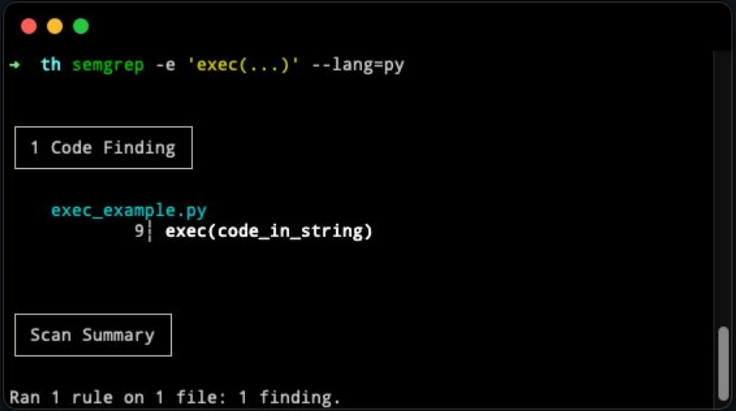
Release rule publicly

How we use Semgrep at TRAIL OF BITS

- Expanding private Semgrep library
  - Allows us to identify bugs at scale
  - We gather ideas for new rules (found both *in the wild* and during audits)
  - Snowball-potential initiative ❄️ → allows us to find bugs more and more effectively
  - Some of the rules become public
    - Semgrep Registry: https://semgrep.dev/p/trailofbits
    - Original Repository: https://github.com/trailofbits/semgrep-rules
    - `$ semgrep --config p/trailofbits`

How we use Semgrep at TRAIL OF BITS

- Triaging
  - SARIF
    - `$ semgrep --sarif p/default`
    - SARIF Viewer extension in Visual Studio Code
    - *We are releasing the SARIF Explorer soon!*
      - *Become a beta tester - send us an e-mail!* [*webinar-sarif@trailofbits.com*](mailto:webinar-sarif@trailofbits.com)
  - Fast false positive filtering
- Ephemeral rules
  - When manually auditing code
  - Alternative to the (rip)grep & weggli
  - `$ semgrep -e 'exec(...)' --lang=py`



```
→ th semgrep -e 'exec(...)' --lang=py


 1 Code Finding

    exec_example.py
              9┆ exec(code_in_string)


 Scan Summary

Ran 1 rule on 1 file: 1 finding.
```

Key Takeaways #2

- There are lots of rulesets

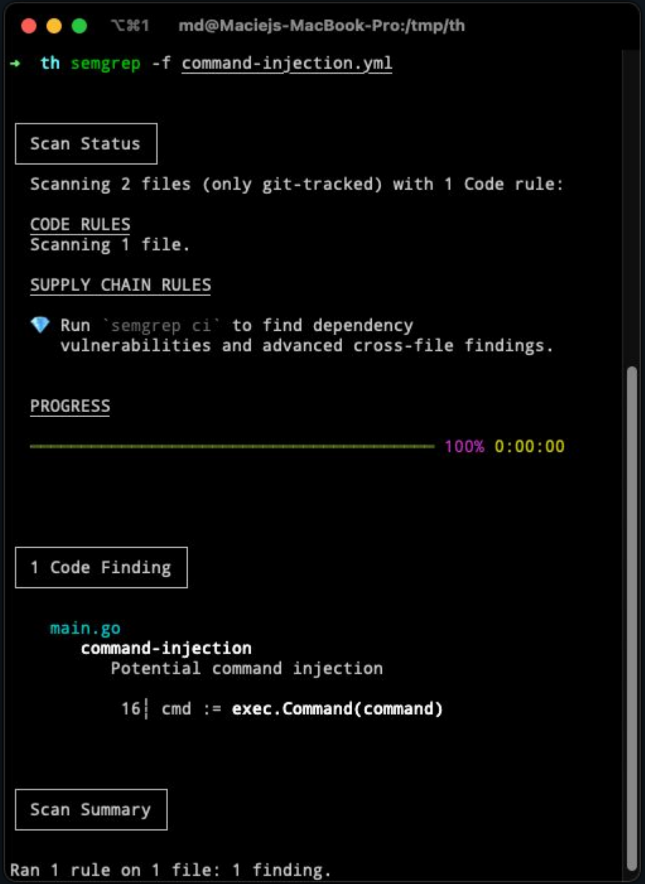
  -
    ```
    $ semgrep --config p/default
    $ semgrep --config p/xss
    $ semgrep --config p/trailofbits
    ```
- Find out more rulesets in Semgrep Registry
  - https://semgrep.dev/explore
- Use SARIF
- Build your private rule portfolio

Example custom rule

- Just a YAML file
- The simplest rule:

```yaml
rules:
 - id: command-injection
   pattern: exec.Command(...)
   message: Potential command injection
   languages: [go]
   severity: ERROR
```

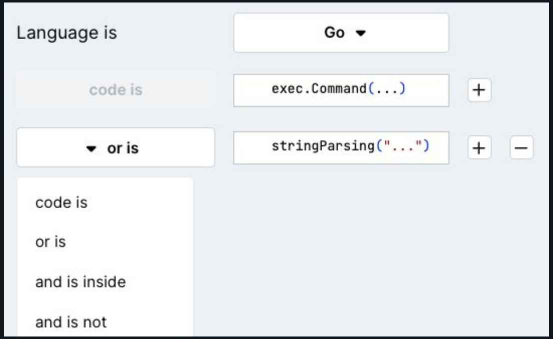
command-injection.yml



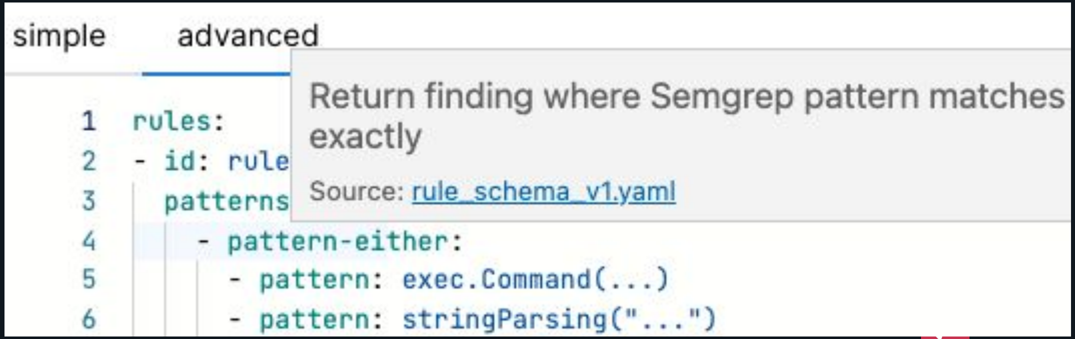
```
⌥⌘1    md@Maciejs-MacBook-Pro:/tmp/th
→ th semgrep -f command-injection.yml


┌─────────────┐
│ Scan Status │
└─────────────┘
  Scanning 2 files (only git-tracked) with 1 Code rule:

  CODE RULES
  Scanning 1 file.

  SUPPLY CHAIN RULES

  💎 Run `semgrep ci` to find dependency
     vulnerabilities and advanced cross-file findings.


  PROGRESS

  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00



┌────────────────┐
│ 1 Code Finding │
└────────────────┘

    main.go
       command-injection
          Potential command injection

           16┆ cmd := exec.Command(command)


┌──────────────┐
│ Scan Summary │
└──────────────┘

Ran 1 rule on 1 file: 1 finding.
```

How to write custom rules easily

- Use Semgrep Playground! → https://semgrep.dev/playground
  - “IDE in a browser” for writing custom Semgrep rules
  - Gives immediate feedback: a rule inspection, underlines errors, etc.
  - Share button to get unique link for a written rule with code
  - Two modes of a rule creation: simple and advanced



```
exec.Command(...)
stringParsing("...")
```



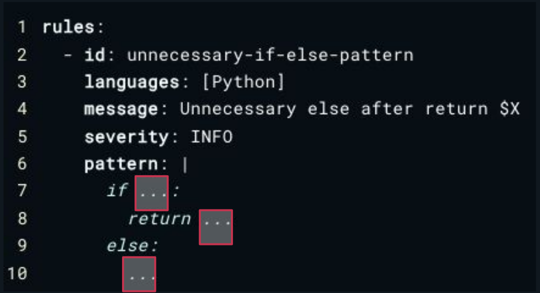
```
rules:
- id: rule
  patterns
  - pattern-either:
    - pattern: exec.Command(...)
    - pattern: stringParsing("...")
```

Return finding where Semgrep pattern matches exactly

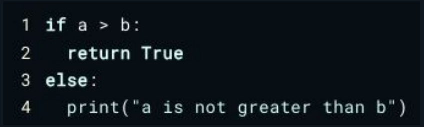
Source: rule_schema_v1.yaml

Basic syntax

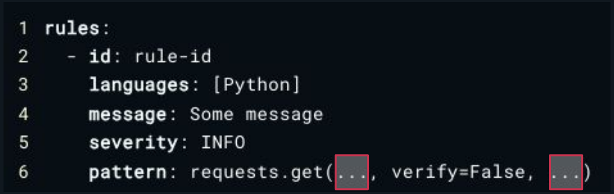
- **Ellipses → . . .**
  - Allow for flexible pattern matching
  - Match zero or more arguments, statements, parameters, etc.



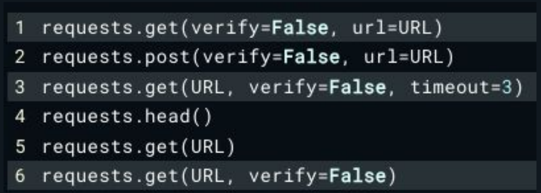
```
rules:
  - id: unnecessary-if-else-pattern
    languages: [Python]
    message: Unnecessary else after return $X
    severity: INFO
    pattern: |
      if ...:
        return ...
      else:
        ...
```



```
if a > b:
  return True
else:
  print("a is not greater than b")
```



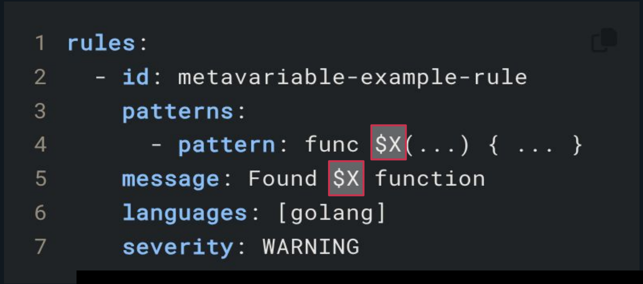
```
rules:
  - id: rule-id
    languages: [Python]
    message: Some message
    severity: INFO
    pattern: requests.get(..., verify=False, ...)
```



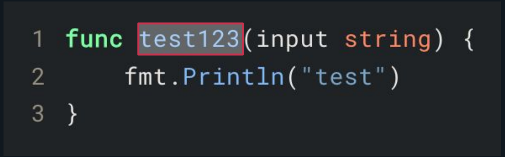
```
requests.get(verify=False, url=URL)
requests.post(verify=False, url=URL)
requests.get(URL, verify=False, timeout=3)
requests.head()
requests.get(URL)
requests.get(URL, verify=False)
```

Basic syntax

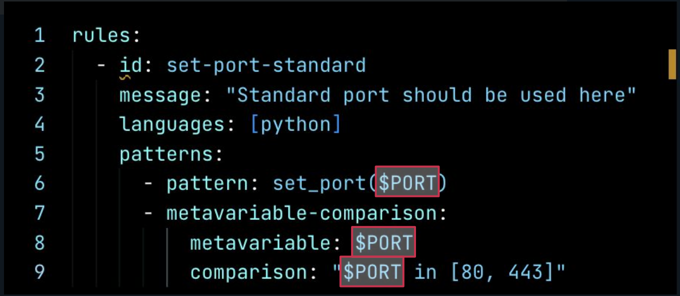
- **Metavariables** → `$X $Y $Z $WHATEVER`
  - Match any code element, such as variables, functions, arguments, classes, etc.
  - Capture and track the use of values across a code scope
  - Can be interpolated into other parts of a rule, including other patterns



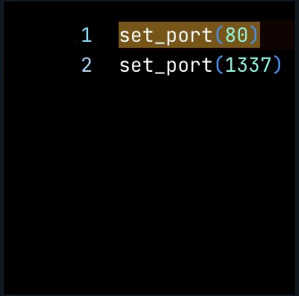
```yaml
rules:
  - id: metavariable-example-rule
    patterns:
      - pattern: func $X(...) { ... }
    message: Found $X function
    languages: [golang]
    severity: WARNING
```



```go
func test123(input string) {
    fmt.Println("test")
}
```



```yaml
rules:
  - id: set-port-standard
    message: "Standard port should be used here"
    languages: [python]
    patterns:
      - pattern: set_port($PORT)
      - metavariable-comparison:
          metavariable: $PORT
          comparison: "$PORT in [80, 443]"
```



```python
set_port(80)
set_port(1337)
```

Basic syntax

- **Operators**
  - Allow you to combine different patterns, for example, logically (OR, AND, NOT)
  - *"Semgrep, inform me if this specific line is in the code BUT don't raise an error if this one line of code exist too"*

`pattern`	Find code matching this expression
`patterns`	Logical AND of multiple patterns
`pattern-either`	Logical OR of multiple patterns
`pattern-regex`	Find code matching this PCRE-compatible pattern in multiline mode
`pattern-not`	Logical NOT - remove findings matching this expression
`pattern-not-inside`	Keep findings that do not lie inside this pattern
`...`	● *https://appsec.guide/docs/static-analysis/semgrep/advanced/* ● *https://semgrep.dev/docs/writing-rules/rule-syntax/*

Key Takeaways #3

- Semgrep patterns mimic the target code
  - `for (...)`
  - `var xyz = $WHATEVER`
- Use patterns to combine logic together
  - *pattern, patterns, pattern-either, pattern-regex, pattern-inside, metavariable-regex, metavariable-pattern, metavariable-comparison, pattern-not, pattern-not-inside, pattern-not-regex …*
- Leverage Semgrep Playground with simple & advanced mode
- Docs: https://semgrep.dev/docs/writing-rules/rule-syntax/

Basic syntax - Example #1

Semgrep, please inform me when…

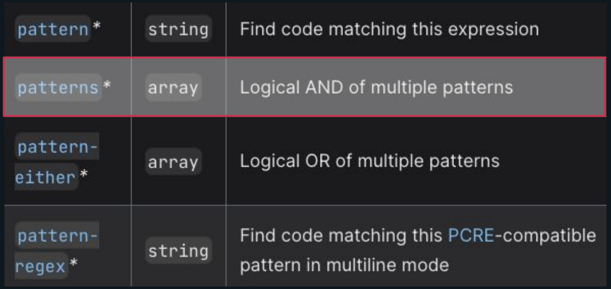
Two of those line codes exist:

`if (1 < 2)`

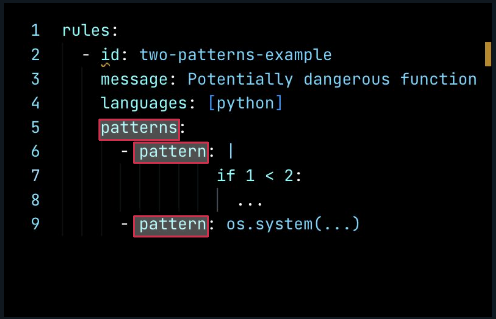
AND

`os.system()` with any argument

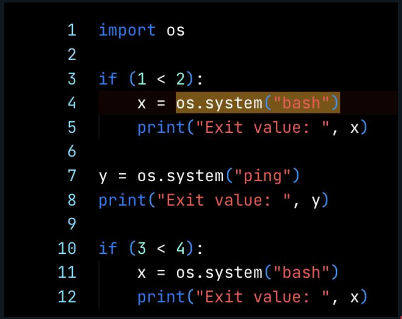
So, I need a rule that requires two of the patterns to be present in the code (logical AND)



pattern*	string	Find code matching this expression
patterns*	array	Logical AND of multiple patterns
pattern-either*	array	Logical OR of multiple patterns
pattern-regex*	string	Find code matching this PCRE-compatible pattern in multiline mode



```yaml
rules:
  - id: two-patterns-example
    message: Potentially dangerous function
    languages: [python]
    patterns:
      - pattern: |
          if 1 < 2:
            ...
      - pattern: os.system(...)
```



```python
import os

if (1 < 2):
    x = os.system("bash")
    print("Exit value: ", x)

y = os.system("ping")
print("Exit value: ", y)

if (3 < 4):
    x = os.system("bash")
    print("Exit value: ", x)
```

Solution: **patterns**

Basic syntax - Example #2

Semgrep, please inform me when...

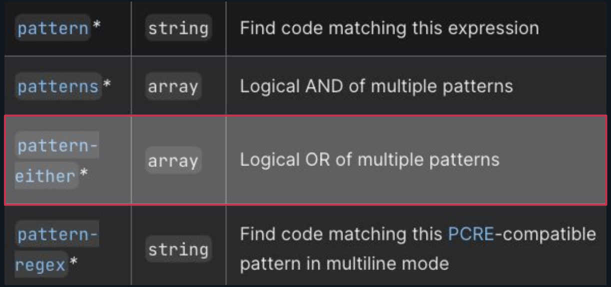
One of the potentially dangerous functions is used:

```
os.system()
```

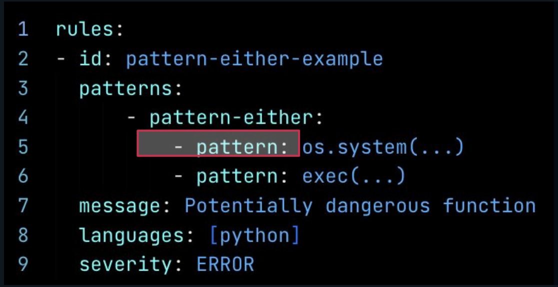
OR

```
exec()
```

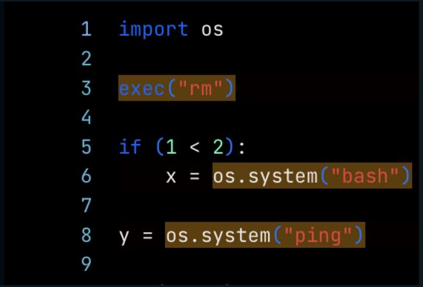
So, I need a rule that requires one of the patterns to be present in the code (logical OR)



`pattern`*	`string`	Find code matching this expression
`patterns`*	`array`	Logical AND of multiple patterns
`pattern-either`*	`array`	Logical OR of multiple patterns
`pattern-regex`*	`string`	Find code matching this PCRE-compatible pattern in multiline mode



```yaml
rules:
- id: pattern-either-example
  patterns:
      - pattern-either:
          - pattern: os.system(...)
          - pattern: exec(...)
  message: Potentially dangerous function
  languages: [python]
  severity: ERROR
```



```python
import os

exec("rm")

if (1 < 2):
    x = os.system("bash")

y = os.system("ping")

```

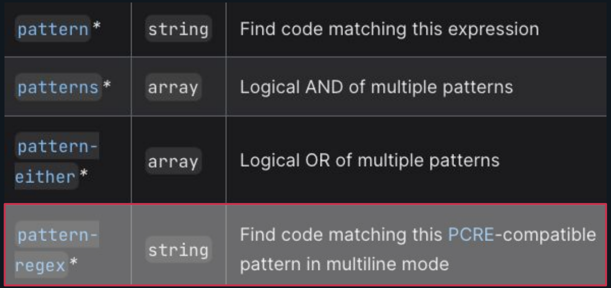
Solution: **pattern-either**

Basic syntax - Example #3

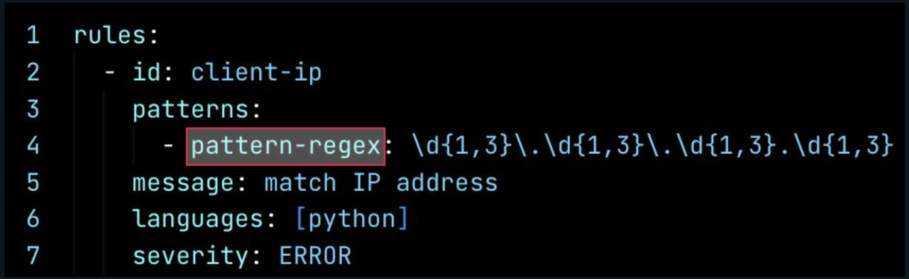
Semgrep, please inform me when…

You match any IP address:
`192.168.0.1`, `1.1.1.1`, *etc.*

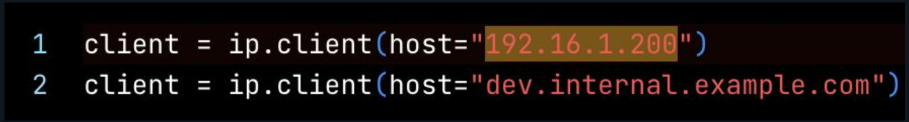
So, I need a rule that matches specific regex pattern to be present in the code



`pattern`*	`string`	Find code matching this expression
`patterns`*	`array`	Logical AND of multiple patterns
`pattern-either`*	`array`	Logical OR of multiple patterns
`pattern-regex`*	`string`	Find code matching this PCRE-compatible pattern in multiline mode



```yaml
rules:
  - id: client-ip
    patterns:
      - pattern-regex: \d{1,3}\.\d{1,3}\.\d{1,3}.\d{1,3}
    message: match IP address
    languages: [python]
    severity: ERROR
```



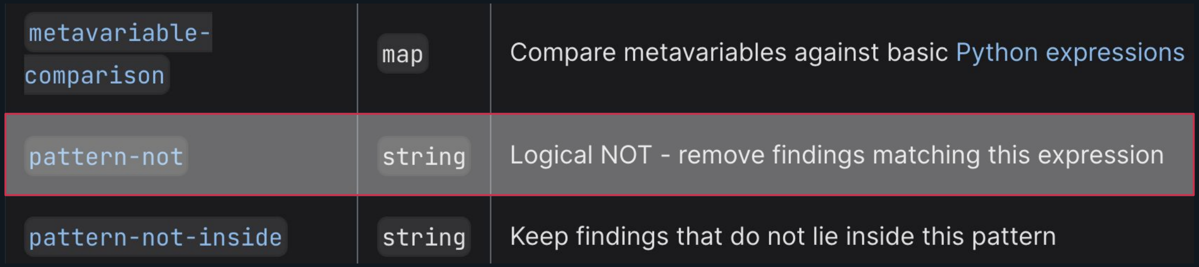
```python
client = ip.client(host="192.16.1.200")
client = ip.client(host="dev.internal.example.com")
```

Solution: **pattern-regex**

A slightly more advanced rule - Example #4

Semgrep, please inform me when…

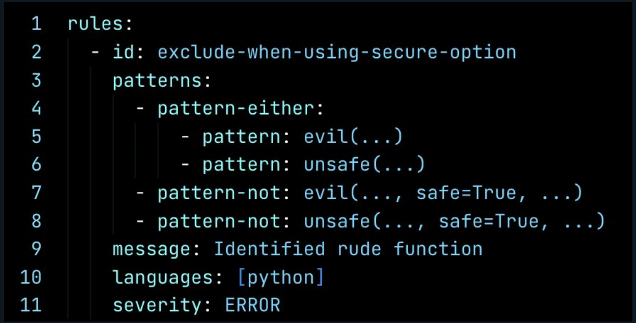
1. One of the potentially dangerous functions is used: `evil()` **OR** `unsafe()`
   - We know **`pattern-either`** from the previous example!
2. **BUT** don't throw an error if one of them has the following option: `safe=True`



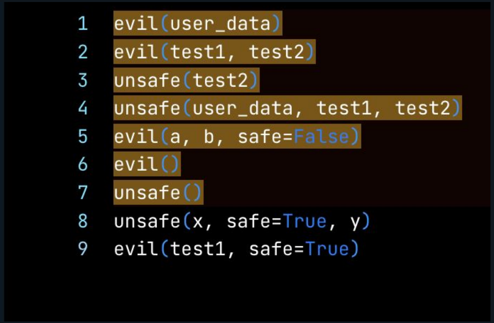
`metavariable-comparison`	`map`	Compare metavariables against basic Python expressions
`pattern-not`	`string`	Logical NOT - remove findings matching this expression
`pattern-not-inside`	`string`	Keep findings that do not lie inside this pattern

Solution: **pattern-either + pattern-not**

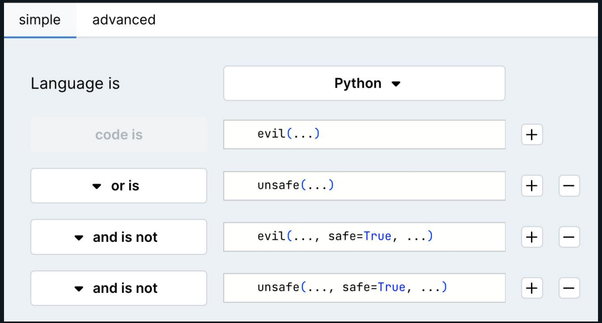
Combining operators together



```yaml
rules:
  - id: exclude-when-using-secure-option
    patterns:
      - pattern-either:
          - pattern: evil(...)
          - pattern: unsafe(...)
      - pattern-not: evil(..., safe=True, ...)
      - pattern-not: unsafe(..., safe=True, ...)
    message: Identified rude function
    languages: [python]
    severity: ERROR
```



```python
evil(user_data)
evil(test1, test2)
unsafe(test2)
unsafe(user_data, test1, test2)
evil(a, b, safe=False)
evil()
unsafe()
unsafe(x, safe=True, y)
evil(test1, safe=True)
```



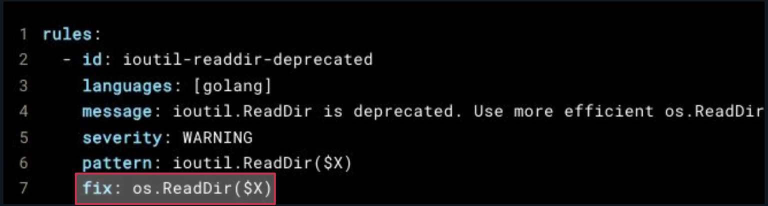
Language is Python

code is	`evil(...)`
or is	`unsafe(...)`
and is not	`evil(..., safe=True, ...)`
and is not	`unsafe(..., safe=True, ...)`

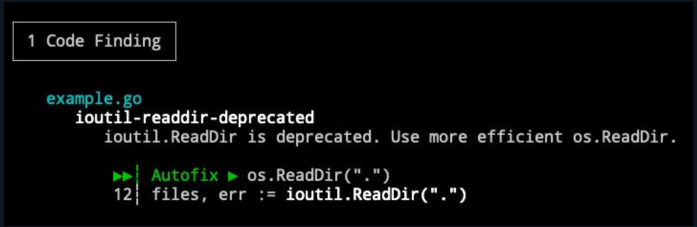
- Sometimes it's possible to create the same logic in different ways
  - Carefully test Semgrep rules to identify edge-case nuances
  - The most optimized rule usually wins. See the Optimizing Semgrep Rules in our Testing Handbook
- The order of child patterns in a `patterns` operator does not affect the final result
- Read the `patterns` operator evaluation strategy
  - https://semgrep.dev/docs/writing-rules/rule-syntax/#patterns-operator-evaluation-strategy

Auto fix

- Automatically fix bugs found by Semgrep rules
  - No worries, you can do a dry run before



```
rules:
  - id: ioutil-readdir-deprecated
    languages: [golang]
    message: ioutil.ReadDir is deprecated. Use more efficient os.ReadDir
    severity: WARNING
    pattern: ioutil.ReadDir($X)
    fix: os.ReadDir($X)
```



```
1 Code Finding

  example.go
     ioutil-readdir-deprecated
        ioutil.ReadDir is deprecated. Use more efficient os.ReadDir.

          ▶▶┆ Autofix ▶ os.ReadDir(".")
          12┆ files, err := ioutil.ReadDir(".")
```

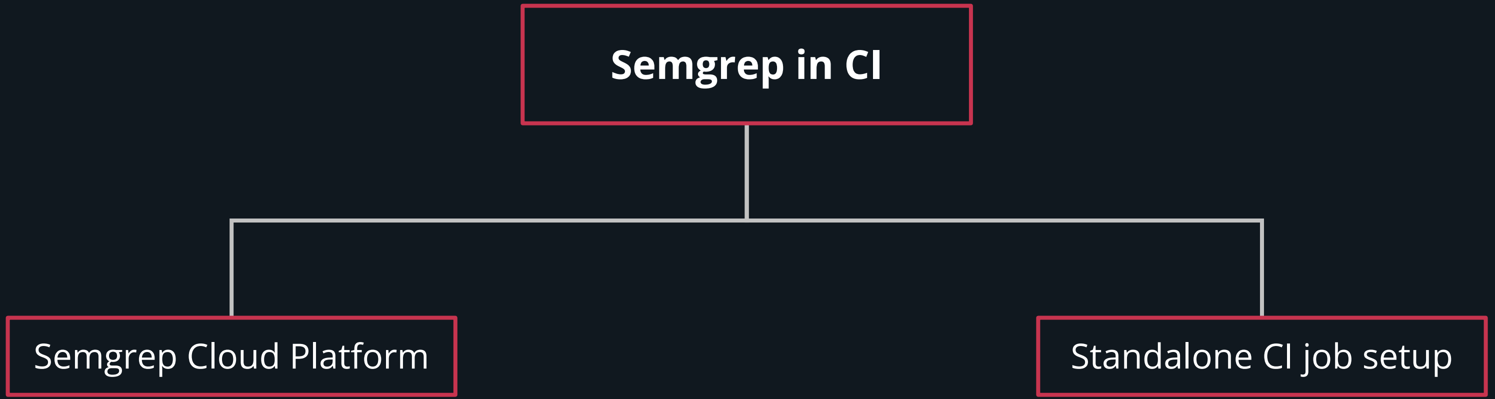
- Speeds up the process of patching vulnerabilities
- It's very informative for developers
  - Shows both the bug and how to fix it so devs learn what to avoid in the future
- The `fix-regex` variant applies regex replacements

```
fix-regex:
  regex: 'csrfPrevention:\s*false'
  replacement: "csrfPrevention: true"
```

Key Takeaways #4

- Mastering operator combinations may initially require trial and error. *This is okay!*
- Leverage Semgrep Playground for testing
- Test your rules on the real code
  - Have a couple large repositories in a specific language at hand
  - 0-days for free ↑
- Learn from existing rules and our informative Semgrep blog posts
  - https://blog.trailofbits.com/category/semgrep/
- Over time, rules improve and refine, reducing false positives and increasing true ones
- It’s impossible to create a universal rule that meets all standards. Some rules tend to be more noisy.

Semgrep in CI/CD



Semgrep Cloud Platform

- Centralized app to manage findings, configure rulesets and notifications
- Configured by clicking elegantly in GUI
- Free for up to 10 devs

Standalone CI job setup

- Highly customizable, depending on the provider
- Mainly based on the configuration file (like GitHub Workflow)
- Free

See the full comparison: https://semgrep.dev/docs/semgrep-ci/overview/#feature-comparison

How to introduce Semgrep to your CI/CD

1. Get acquainted with documentation related to your CI vendor
   - For example, with GitHub Actions
   - See official Semgrep docs and our CI chapter of the Testing Handbook
2. Incorporate incrementally - try out a *pilot test* first on a repository
   - Don't overwhelm devs with too many results
   - Use rules that provide high confidence and true positive results
   - Use comments to ignore false positives:

     `// nosemgrep: go.lang.security.audit.xss`
3. Schedule a full Semgrep scan on the main branch
4. Include a *diff-aware scanning* approach when an event triggers (e.g., a pull request)
   - Scans only changes in files on a trigger
   - Maintains efficiency
5. Configure Semgrep to block the PR pipeline with unresolved findings

Semgrep in CI/CD - More tips for your organization

1. Obtain your ideal rulesets chain
   - Check out non-security rulesets, such as best practices rules
   - Cover other aspects: shell scripts, configuration files, Dockerfiles
2. Consider writing custom rules for found bugs
   - Create an internal repository to aggregate custom Semgrep rules
   - Encourage developers to jot down ideas for Semgrep rules (e.g., on a Trello board)
3. Create a place for the team to discuss Semgrep (e.g., a Slack channel)
   - Support for writing custom rules
   - Troubleshooting

More information in our How to introduce Semgrep to your organization blog post
https://blog.trailofbits.com/2024/01/12/how-to-introduce-semgrep

Where to find support

1. Semgrep Community Slack
    - https://go.semgrep.dev/slack
    - Great place to ask for help with custom rule development or using the tool
2. Semgrep GitHub Issues
    - https://github.com/semgrep/semgrep/issues
    - Report bugs
    - Suggest new features
3. `#testing-handbook` channel in the Empire Hacking Slack
    - https://slack.empirehacking.nyc/
    - **ToB** is happy to help (:
    - Give us feedback on the Testing Handbook!

Summary

- Testing Handbook: https://appsec.guide/
- Check out our blog: https://blog.trailofbits.com/
  - *30 new Semgrep rules: Ansible, Java, Kotlin, shell scripts, and more*
  - *How to introduce Semgrep to your organization*
  - *Secure your Apollo GraphQL server with Semgrep*
  - *Secure your machine learning with Semgrep*
  - *Discovering goroutine leaks with Semgrep*
- Our public audit reports show how Semgrep behaves for real https://github.com/trailofbits/publications
- ToB public Semgrep rules: https://github.com/trailofbits/semgrep-rules