Fix Every Instance

OWASP NZ Day 2022

Who is this person?

Tim Goddard

Principal Security Consultant @ CyberCX

Ex-Software Developer, Focus on AppSec and Code Review.

Thank You to Our Sponsors and Hosts!

OWASP NEW ZEALAND owasp.org.nz

QUANTUM SECURITY

AUT

CyberCX

DATACOM

snyk

Auth0

Checkmarx

HCL AppScan

kordia®

LATERAL SECURITY

MICRO FOCUS®

Pulse Security www.pulsesecurity.co.nz

RedShield

Flux

SEQA™ Information Security

Cobalt

LACEWORK

SecureFlag

Without them, OWASP New Zealand Day couldn't happen

So your project had a pentest…

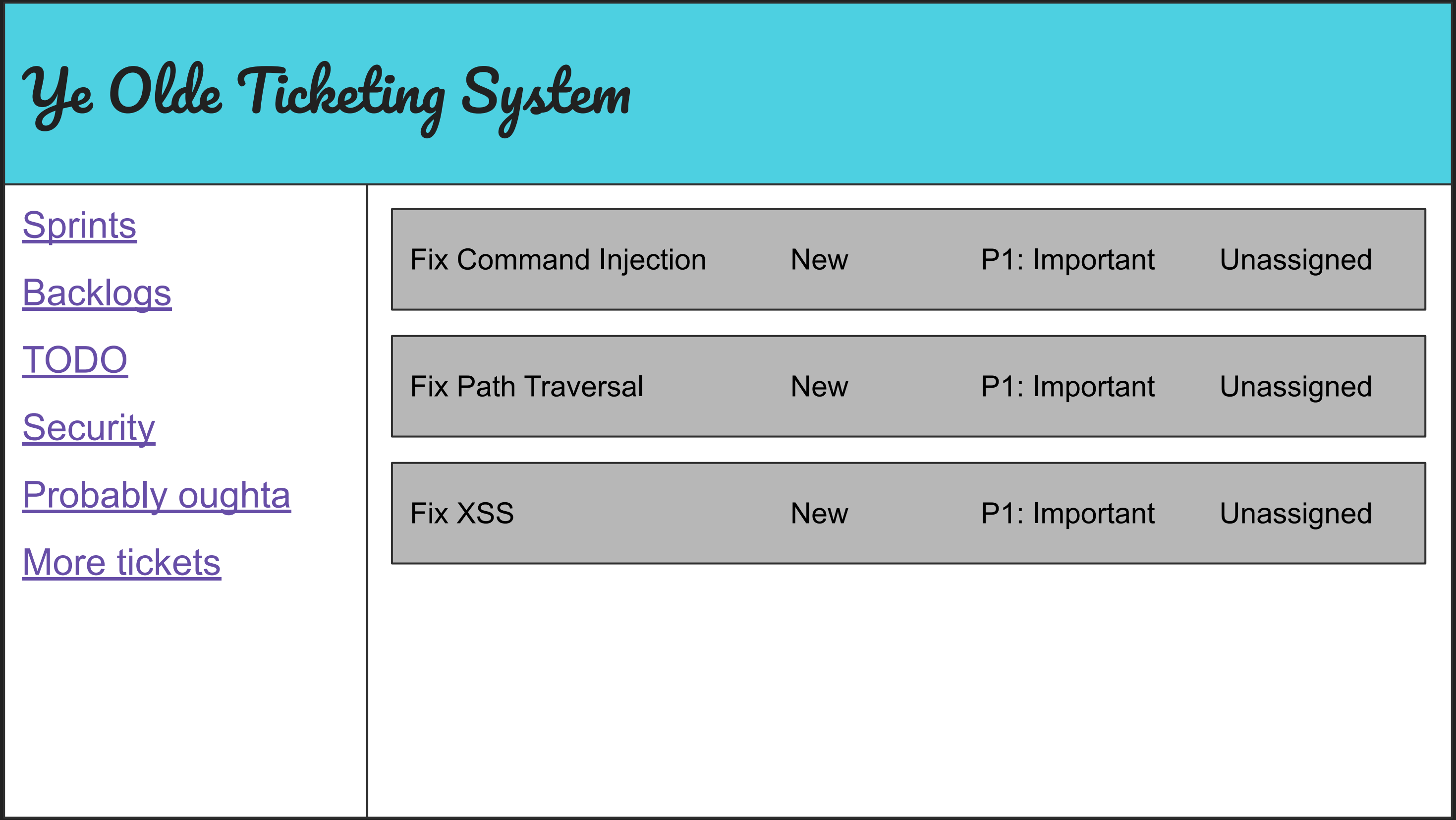
Summary

OMG THE WORLD IS BURNING!!!!!!

Issues

💥💥💥	Command Injection (Remote Code Execution)
🔥🔥🔥	Arbitrary File Read (Path Traversal)
🔥🔥🔥	Cross-Site Scripting

⋮



Ye Olde Ticketing System

Sprints

Backlogs

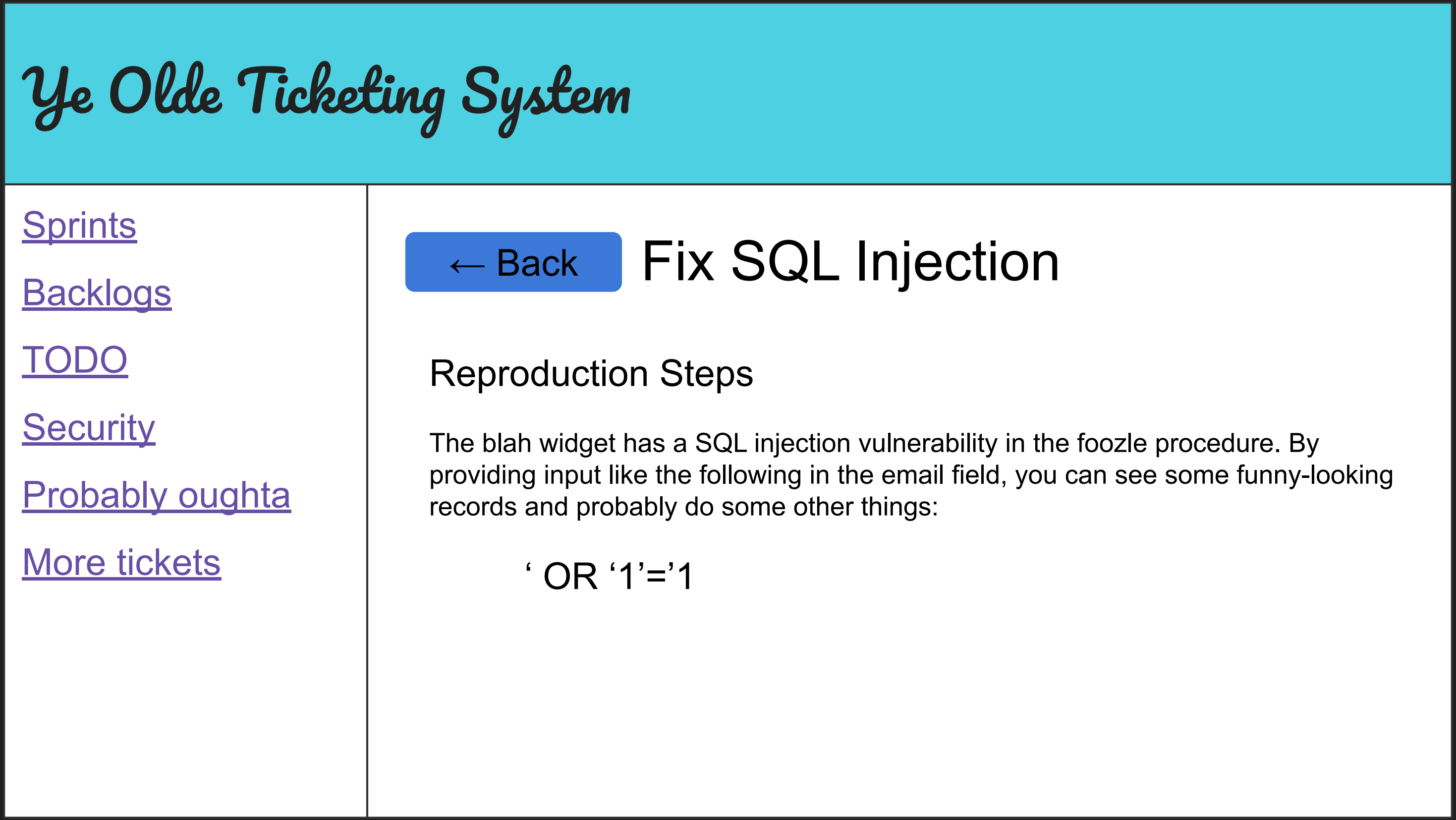
TODO

Security

Probably oughta

More tickets

Fix Command Injection	New	P1: Important	Unassigned
Fix Path Traversal	New	P1: Important	Unassigned
Fix XSS	New	P1: Important	Unassigned



Ye Olde Ticketing System

Sprints

Backlogs

TODO

Security

Probably oughta

More tickets

← Back

Fix SQL Injection

Reproduction Steps

The blah widget has a SQL injection vulnerability in the foozle procedure. By providing input like the following in the email field, you can see some funny-looking records and probably do some other things:

```
‘ OR ‘1’=’1
```

Dev team closed all the tickets… so we’re done, right?

Findings are rarely, if ever, complete

Penetration tests are not exhaustive.

Some bugs require conditions which did not, or cannot occur.

Pentesting != Code Review

Fix bug classes, not bugs

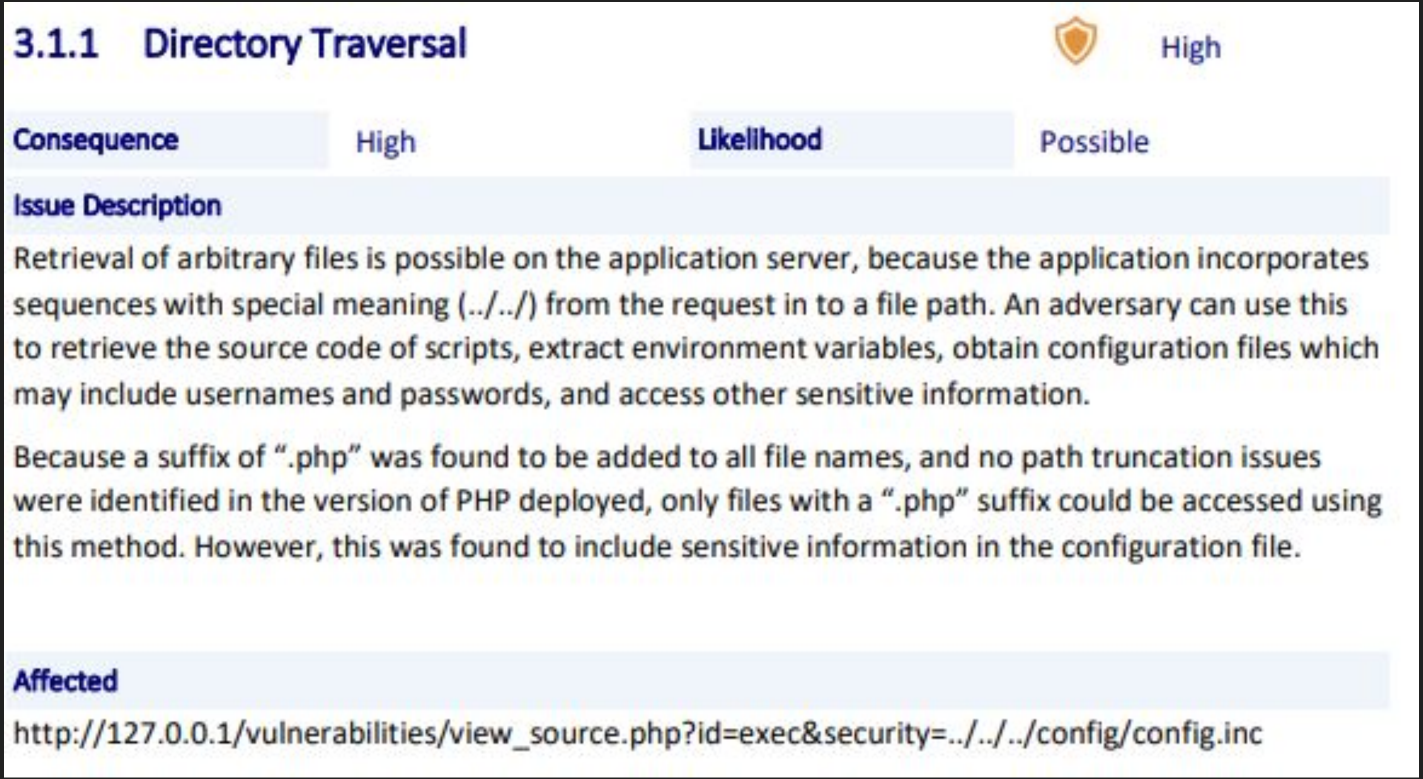
Bugs occur in repeating classes.

Whenever you see a report, ask whether this could represent a recurring pattern.

Search for more instances of the same pattern.

Fix them all in a consistent way.

So we have a finding…



3.1.1 Directory Traversal High

Consequence	High	**Likelihood**	Possible

Issue Description

Retrieval of arbitrary files is possible on the application server, because the application incorporates sequences with special meaning (../../) from the request in to a file path. An adversary can use this to retrieve the source code of scripts, extract environment variables, obtain configuration files which may include usernames and passwords, and access other sensitive information.

Because a suffix of ".php" was found to be added to all file names, and no path truncation issues were identified in the version of PHP deployed, only files with a ".php" suffix could be accessed using this method. However, this was found to include sensitive information in the configuration file.

Affected

http://127.0.0.1/vulnerabilities/view_source.php?id=exec&security=../../../config/config.inc

Ye Olde Ticketing System

Sprints

Backlogs

TODO

Security

Probably oughta

More tickets

← Back

Find and Fix Directory Traversal Cases

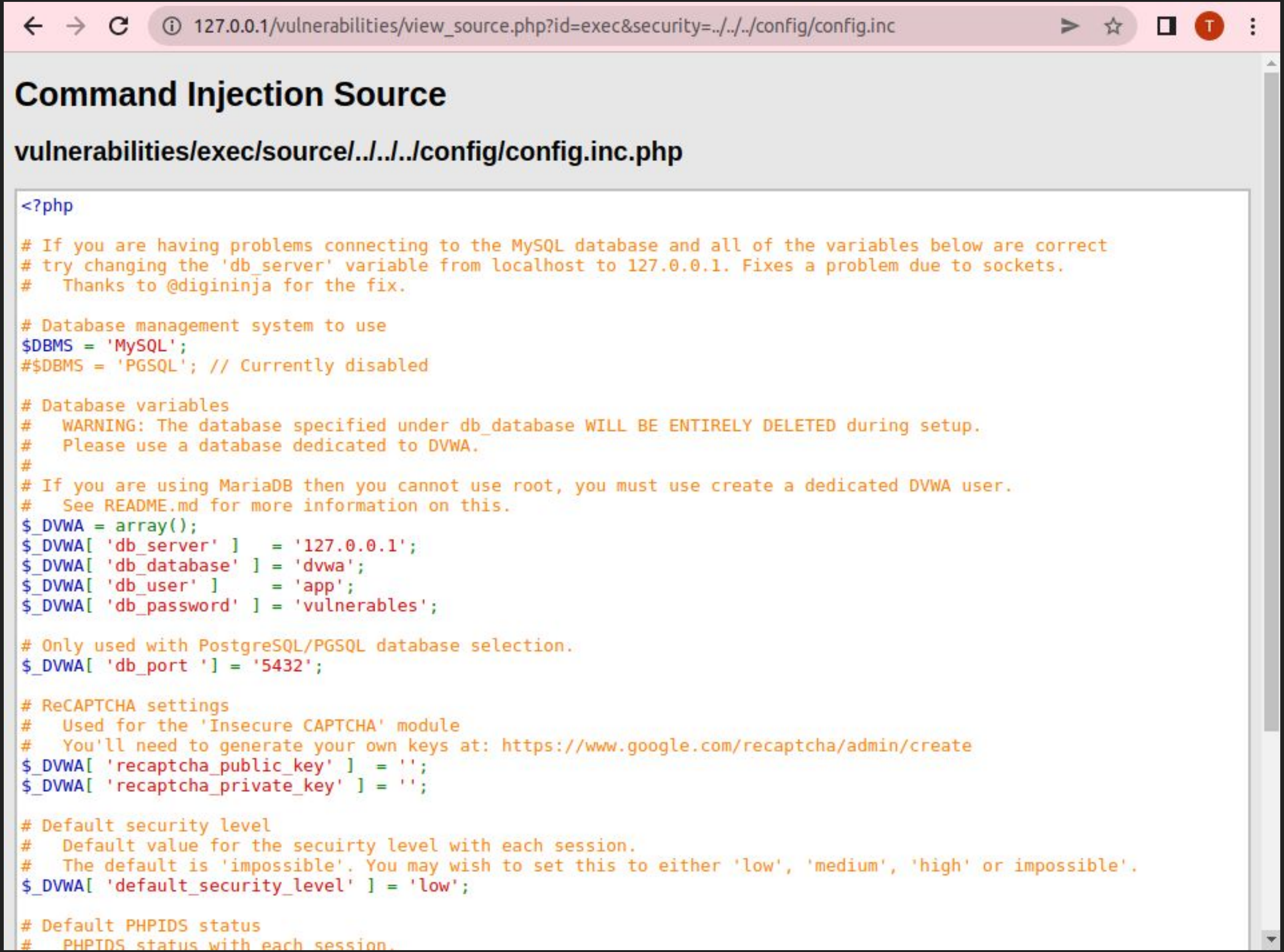
Identify the cause of this issue, find and fix any similar issues in the codebase.

Example instance

The blah widget includes user input to identify which file to load its description from. The “id” parameter is directly included in the filename, which can result in loading files from a different directory, by including an expression like the following:

```
../../../../etc/passwd%00
```

How?


127.0.0.1/vulnerabilities/view_source.php?id=exec&security=../../../config/config.inc

Command Injection Source

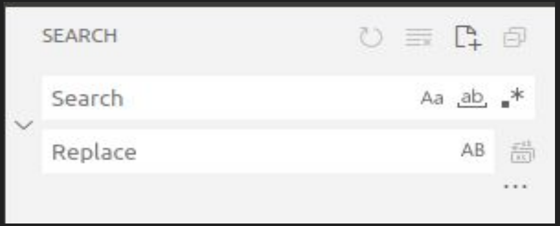
vulnerabilities/exec/source/../../../config/config.inc.php

```
<?php

# If you are having problems connecting to the MySQL database and all of the variables below are correct
# try changing the 'db_server' variable from localhost to 127.0.0.1. Fixes a problem due to sockets.
#   Thanks to @digininja for the fix.

# Database management system to use
$DBMS = 'MySQL';
#$DBMS = 'PGSQL'; // Currently disabled

# Database variables
#   WARNING: The database specified under db_database WILL BE ENTIRELY DELETED during setup.
#   Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a dedicated DVWA user.
#   See README.md for more information on this.
$_DVWA = array();
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'app';
$_DVWA[ 'db_password' ] = 'vulnerables';

# Only used with PostgreSQL/PGSQL database selection.
$_DVWA[ 'db_port '] = '5432';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
#   You'll need to generate your own keys at: https://www.google.com/recaptcha/admin/create
$_DVWA[ 'recaptcha_public_key' ]  = '';
$_DVWA[ 'recaptcha_private_key' ] = '';

# Default security level
#   Default value for the secuirty level with each session.
#   The default is 'impossible'. You may wish to set this to either 'low', 'medium', 'high' or impossible'.
$_DVWA[ 'default_security_level' ] = 'low';

# Default PHPIDS status
#   PHPIDS status with each session.
```

Locate the code in question

```
if (array_key_exists ("id", $_GET) && array_key_exists ("security", $_GET)) {
        $id       = $_GET[ 'id' ];
        $security = $_GET[ 'security' ];

        switch ($id) {
                case "fi" :
                        $vuln = 'File Inclusion';
                        Break;

                  ...

                default:
                        $vuln = "Unknown Vulnerability";
        }

        $source = @file_get_contents( DVWA_WEB_PAGE_TO_ROOT .
"vulnerabilities/{$id}/source/{$security}.php" );
```

How to find similar instances?



Commercial

SAST Scanner

Semgrep

Semgrep = “Semantic Grep”

Search for vulnerabilities (or other things) in code.

Core tool is open source.

Understands the structure of the code.

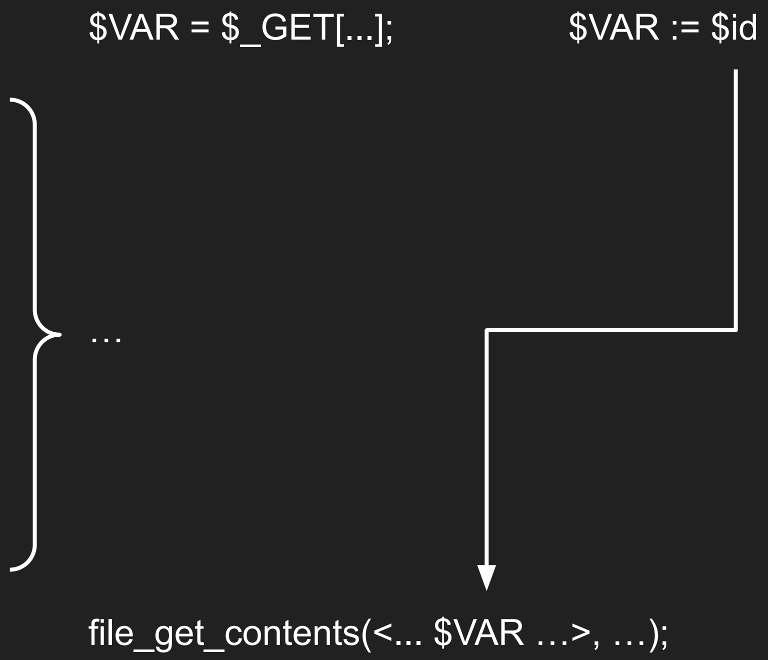
Search patterns look like code (mostly).

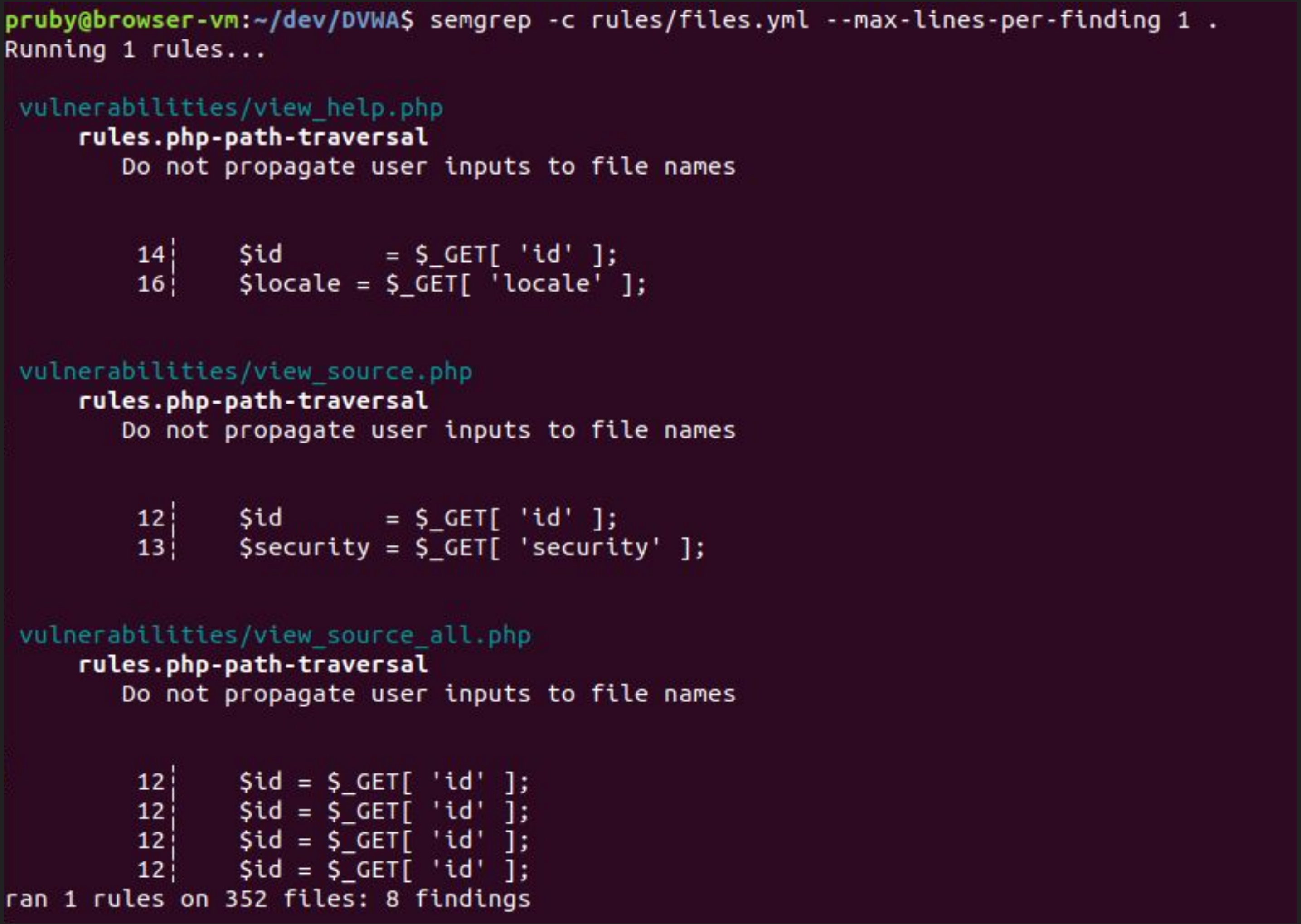
Write a rule to match

```yaml
rules:
 - id: php-path-traversal
   message: Do not propagate user inputs to file names
   severity: ERROR
   languages:
     - php
   patterns:
     - pattern: |
         $VAR = $_GET[...];
         ...
         file_get_contents(<... $VAR ...>, ...);
```

How the code is matched…

```
if (array_key_exists ("id", $_GET) && array_key_exists
("security", $_GET)) {
       $id       = $_GET[ 'id' ];
       $security = $_GET[ 'security' ];

       switch ($id) {
               case "fi" :
                       $vuln = 'File Inclusion';
                       Break;

                 …

               default:
                       $vuln = "Unknown Vulnerability";
       }

       $source = @file_get_contents(
DVWA_WEB_PAGE_TO_ROOT .
"vulnerabilities/{$id}/source/{$security}.php" );
```





```
pruby@browser-vm:~/dev/DVWA$ semgrep -c rules/files.yml --max-lines-per-finding 1 .
Running 1 rules...

 vulnerabilities/view_help.php
     rules.php-path-traversal
        Do not propagate user inputs to file names


         14┆    $id       = $_GET[ 'id' ];
         16┆    $locale = $_GET[ 'locale' ];


 vulnerabilities/view_source.php
     rules.php-path-traversal
        Do not propagate user inputs to file names


         12┆    $id       = $_GET[ 'id' ];
         13┆    $security = $_GET[ 'security' ];


 vulnerabilities/view_source_all.php
     rules.php-path-traversal
        Do not propagate user inputs to file names


         12┆    $id = $_GET[ 'id' ];
         12┆    $id = $_GET[ 'id' ];
         12┆    $id = $_GET[ 'id' ];
         12┆    $id = $_GET[ 'id' ];
ran 1 rules on 352 files: 8 findings
```

What if our code were this?

```
$id       = $_GET[ 'id' ];
$security = $_GET[ 'security' ];

...

$filename = DVWA_WEB_PAGE_TO_ROOT . "vulnerabilities/{$id}/source/{$security}.php";
$source = @file_get_contents( $filename );
```

Deal with intermediate vars in “taint” mode

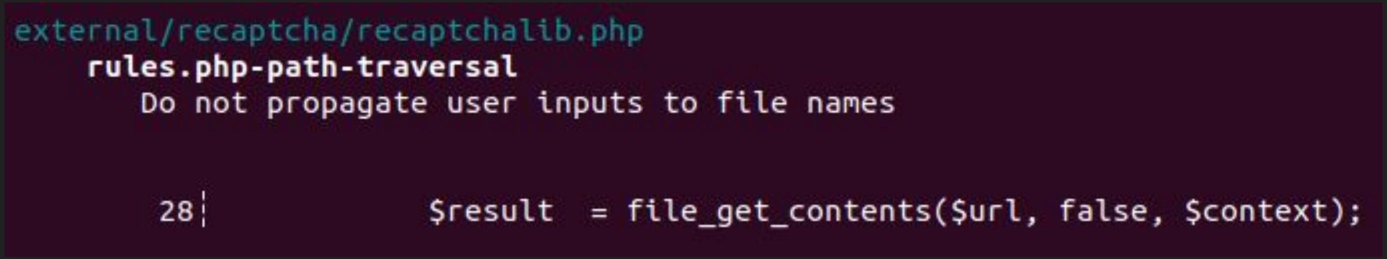
```yaml
rules:
 - id: php-path-traversal
   message: Do not propagate user inputs to file names
   severity: ERROR
   languages:
       - php
   mode: taint
   pattern-sources:
       - pattern: $_GET[...]
   pattern-sinks:
       - pattern: file_get_contents(...)
```

NB: Taint mode does not trace taint between functions, or understand conditions. Very simple rules.

Expand to cover other options…

```yaml
rules:
 - id: php-path-traversal
   message: Do not propagate user inputs to file names
   severity: ERROR
   languages:
       - php
   mode: taint
   pattern-sources:
       - pattern: $_GET[...]
       - pattern: $_POST[...]
       - pattern: $_REQUEST[...]
       - pattern: $_COOKIE[...]
       - pattern: $_SERVER[...]
   pattern-sinks:
       - pattern: file(...)
       - pattern: file_get_contents(...)
       - patterns:
           - pattern-inside: file_put_contents($FILE, ...)
           - pattern: $FILE
```

As we expand, there will be false positives



```
external/recaptcha/recaptchalib.php
    rules.php-path-traversal
       Do not propagate user inputs to file names

       28┆            $result  = file_get_contents($url, false, $context);
```

```php
function CheckCaptcha($key, $response) {

   try {
       $url = 'https://www.google.com/recaptcha/api/siteverify';
       $dat = array(
           'secret'   => $key,
           'response' => urlencode($response),
           'remoteip' => urlencode($_SERVER['REMOTE_ADDR'])
       );
```

Exclude criteria that identify the safe instances

```
pattern-sinks:
    - pattern: file(...)
    - pattern: file_get_contents(...)
    - patterns:
        - pattern-inside: file_put_contents($FILE, ...)
        - pattern: $FILE
pattern-sanitizers:
    - pattern: urlencode(...)
```

Note urlencode is not a complete sanitizer for file names. In our code, however, it might be a good heuristic.

… choose the assumptions you’re comfortable with …

```
pattern-sources:
    - pattern: $_GET[...]
    - pattern: $_POST[...]
    - pattern: $_REQUEST[...]
    - patterns:
        - pattern: $_SERVER[...]
        - pattern-not: $_SERVER['REMOTE_ADDR']
    - pattern: $_COOKIE[...]
```

We’re making a different assumption here - that “REMOTE_ADDR” can only be set to safe values.

Fix everything!

If it’s a bad pattern, why prove vulnerability?

Deploy as a pipeline test…

Reuse for similar apps.

Ask not what is bad,
ask what is good.

Choose your own standards

Chances are you already have an agreed standard/convention for dealing with database queries, HTML generation, URL construction, etc.

You can write a strict rule, which enforces this convention.

Some conventions are better than others, but nearly any convention is better than none.

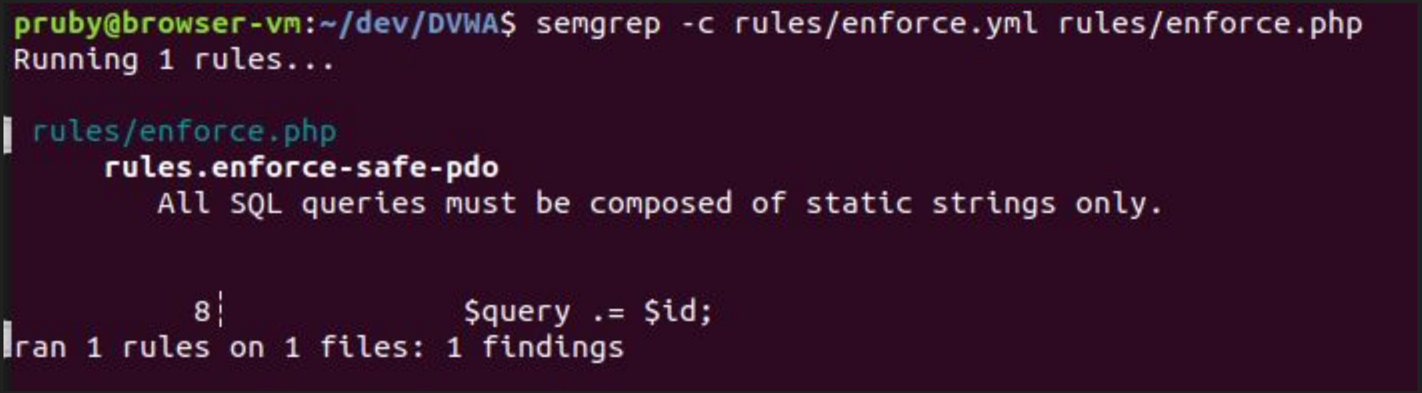
If you would like to parametrise all SQL queries, turn to page 15.

If you would like to sanitise queries by hand, turn to page 97.

E.g. All SQL queries must be composed of static strings

```yaml
rules:
 - id: enforce-safe-pdo
   message: All SQL queries must be composed of static strings only.
   severity: WARNING
   languages:
     - php
   patterns:
     - pattern-inside: |
         ...
         $DBO->query(<... $QUERY  ...>, ...);
     - pattern-either:
         - pattern: $QUERY = $NONSTATIC;
         - pattern: $QUERY .= $NONSTATIC;
     - metavariable-pattern:
         metavariable: $NONSTATIC
         patterns:
           - pattern-not: '"..."'
```

Standard breach = bug



```
pruby@browser-vm:~/dev/DVWA$ semgrep -c rules/enforce.yml rules/enforce.php
Running 1 rules...

 rules/enforce.php
     rules.enforce-safe-pdo
        All SQL queries must be composed of static strings only.


          8┆             $query .= $id;
ran 1 rules on 1 files: 1 findings
```

Key Points

- Fix bug classes, not bugs.
- Fix every instance you can find of a bug.

  … even in other applications that may have been built similarly.

- Use rules to prevent re-introduction of issues in CI/CD.
- Ideally, choose one safe way, rather than trying to detect all possible bad ways.

Questions → Whoova Session Q&A