OWASP 2021
>irtual
APPSEC

PRESENTED BY: Craig Francis

Ending Injection Vulnerabilities

- Injection Vulnerabilities
- Taint Checking... or not
- A special type of String
- Handling some oddities
- Examples in Go, Node (Javascript), Java, and PHP
- The Future
- And in ~10 years time.

"Distinguishing strings from a trusted developer, from strings that may be attacker controlled"

Mike Samuel - 27th March 2019

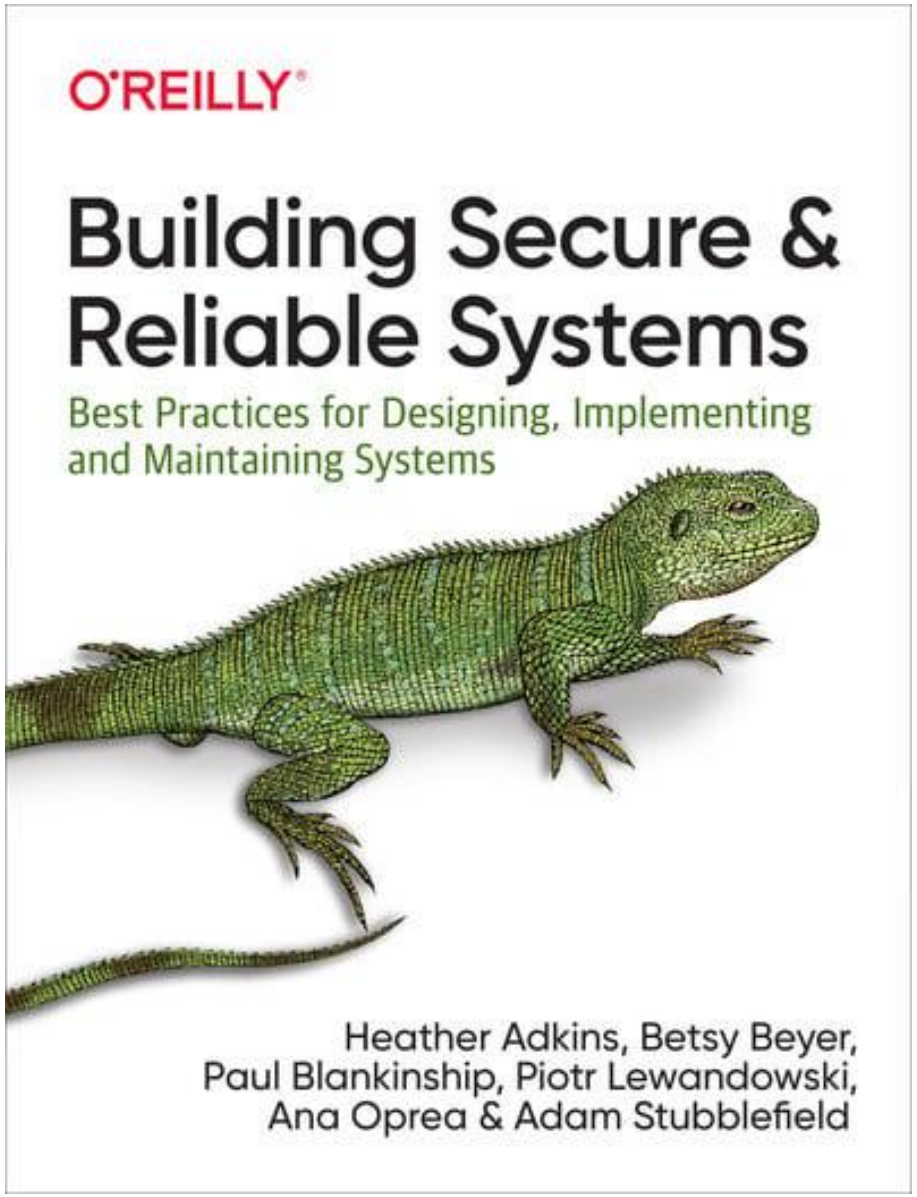
Christoph Kern

Preventing Security Bugs through Software Design

USENIX Security 2015

AppSec California 2016

https://youtu.be/ccfEu-Jj0as



Building Secure and Reliable Systems

March 2020

ISBN 9781492083078

Common Security Vulnerabilities

Page 266

Thanks to Toby Fox, for our lead actors:

Undyne,
Defender

Spamton,
Attacker

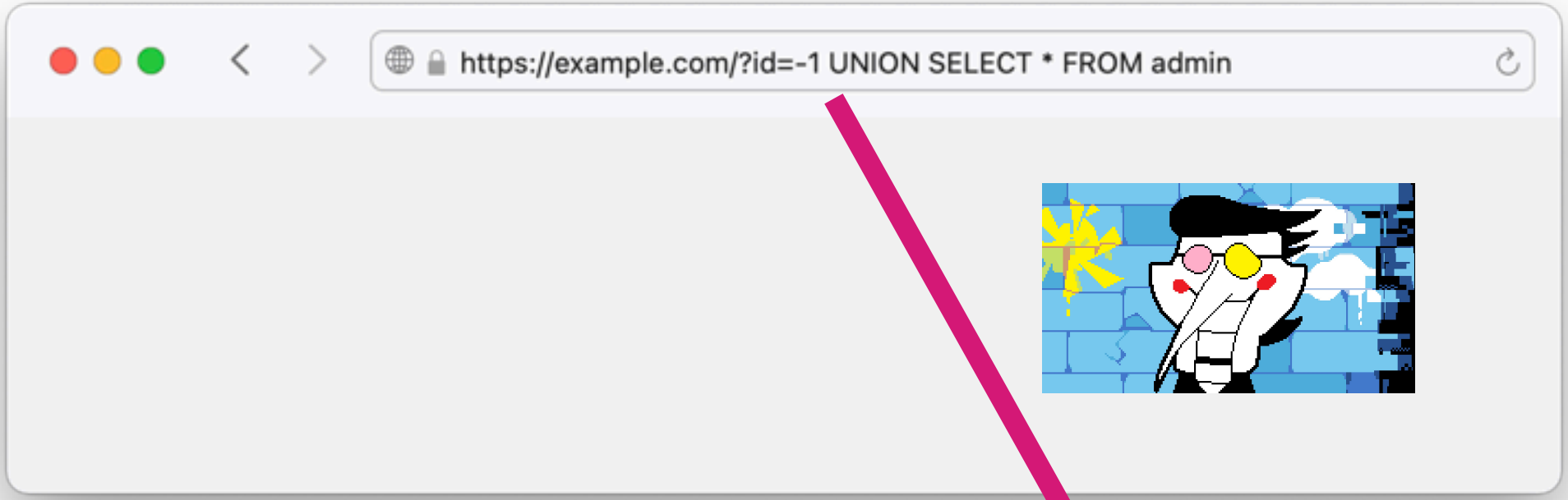
From Undertale & Deltarune

Injection Vulnerabilities


```
$sql = 'SELECT * FROM user WHERE id = ' . $id;
```

$sql = 'SELECT * FROM user WHERE id = ' . $id;

SELECT * FROM user WHERE id = 123



$sql = 'SELECT * FROM user WHERE id = ' . $id;

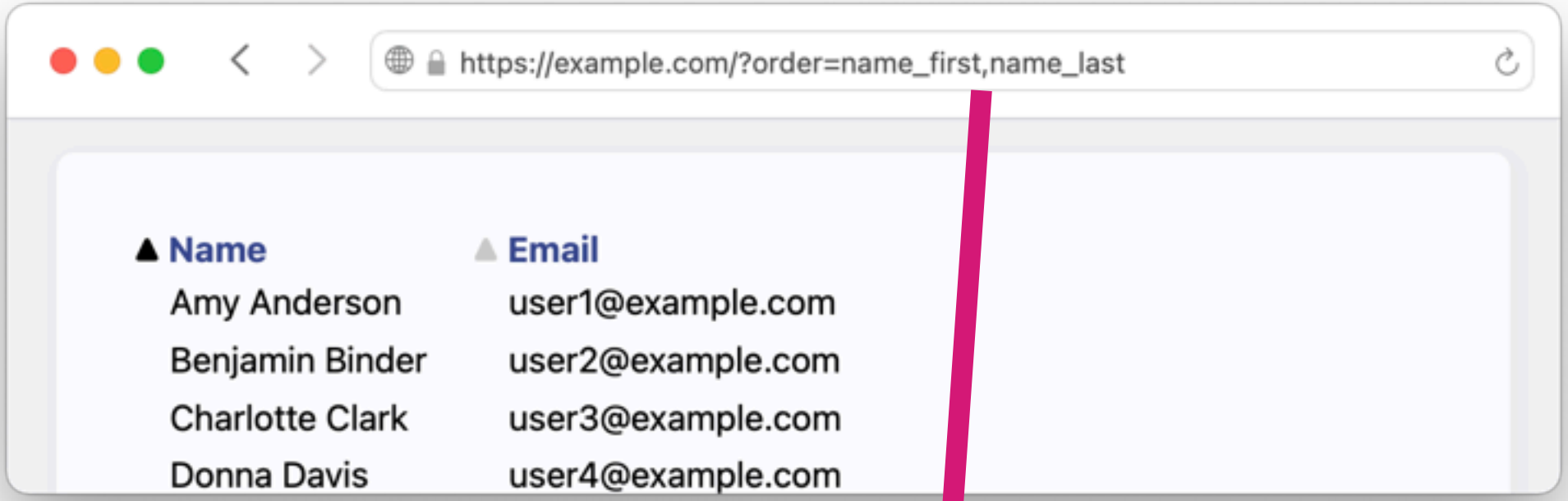
SELECT * FROM user WHERE id = -1 UNION SELECT * FROM admin


```
$sql = 'SELECT * FROM user WHERE id = ?';

$db->query($sql, $id);
```

```
$articles->where('author_id', $id);

$articles->where('author_id IS NULL');

$articles->where('DATE(published)', $date);
```

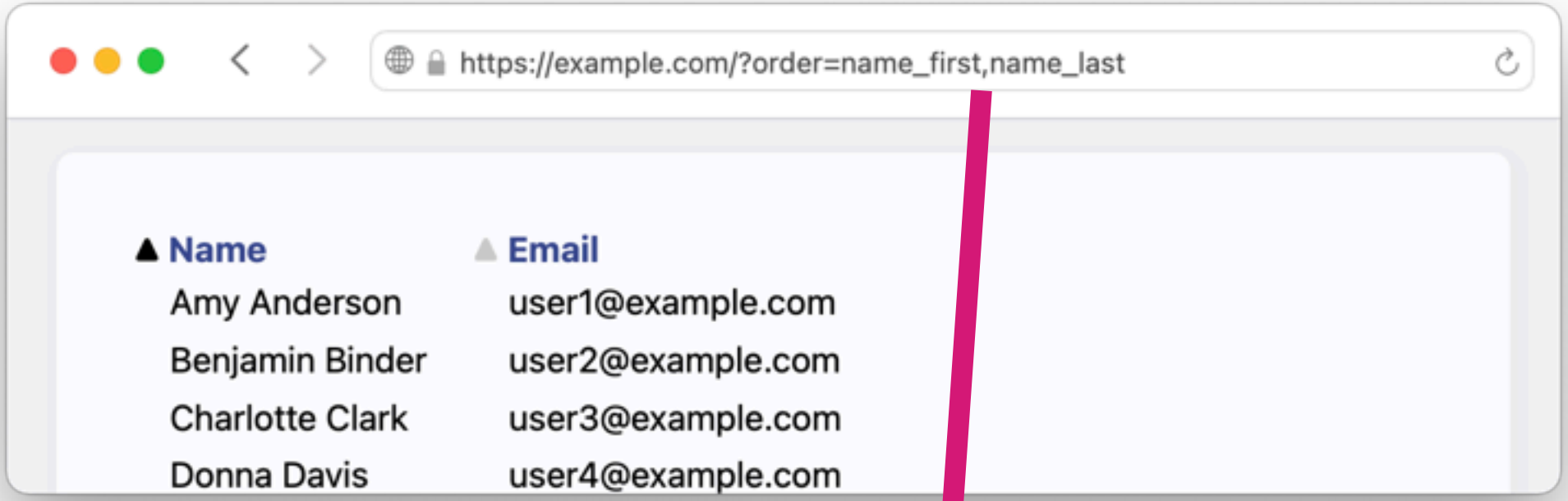
```
$articles->where('author_id', $id);

$articles->where('author_id IS NULL');

$articles->where('DATE(published)', $date);

$articles->where('word_count > 1000');

$articles->where('word_count > ', $count);

$articles->where('word_count > ' . $count);
```

'word_count > word_count UNION SELECT * FROM admin'




```
$users->order($order);
```

... ORDER BY name_first, name_last

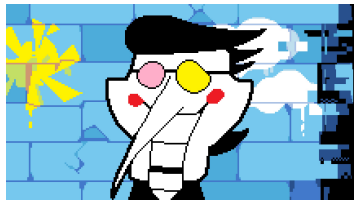



```
$users->order($order);
```

```
... ORDER BY id = (SELECT 1 FROM admin WHERE id = 6 AND pass LIKE "a%")
... ORDER BY id = (SELECT 1 FROM admin WHERE id = 6 AND pass LIKE "b%")
... ORDER BY id = (SELECT 1 FROM admin WHERE id = 6 AND pass LIKE "ba%")
```

```
$html = '<p>Hi ' . $name . '</p>';
```

```
'<p>Hi <script>alert();</script></p>'
```

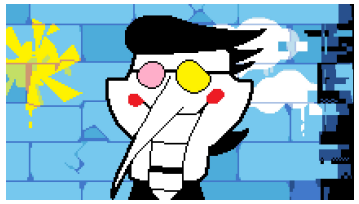


```
$html = '<p>Hi {{ name }}</p>';

$template->render($html, ['name' => $name]);
```

The templating engine does need to be context aware

```
$exec = 'grep "' . $search . '" /path/to/file';
```

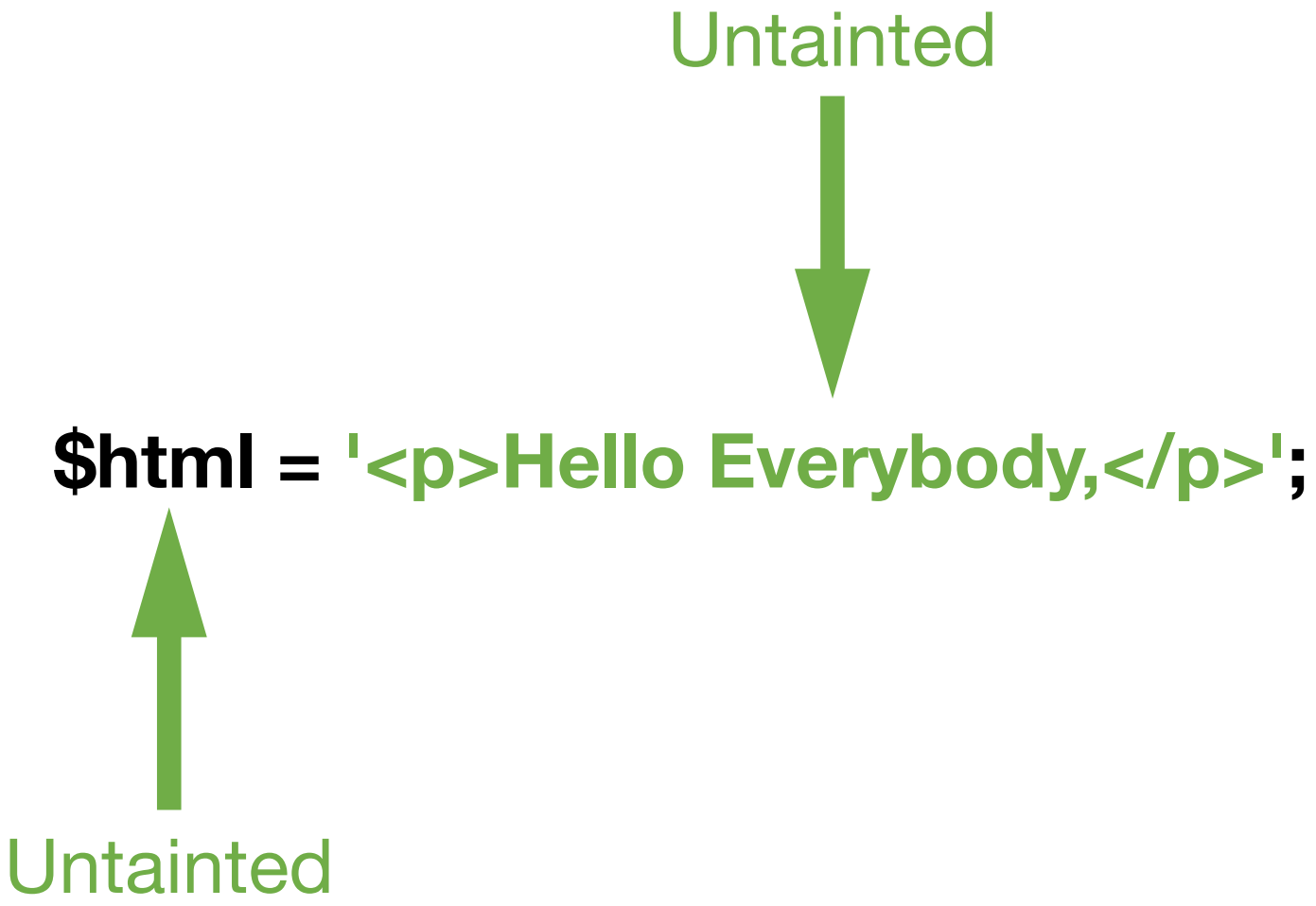
```
grep "" /path/to/secrets; # " /path/to/file
```



How do we stop mistakes?

Maybe Taint Checking?

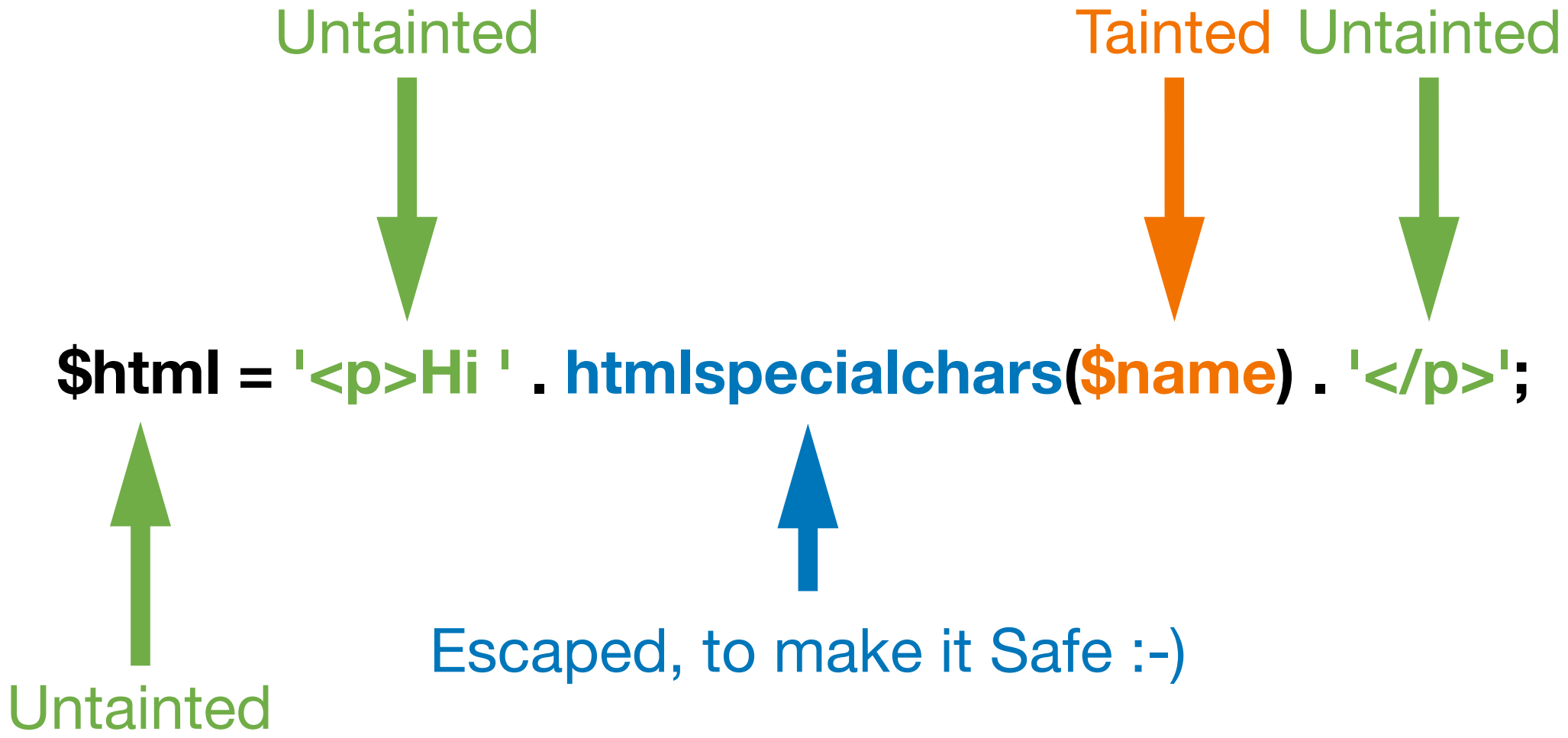
Where variables note if they are Tainted, or Untainted.


Untainted
$html = ' Hello Everybody, ';
Untainted

Untainted Tainted Untainted

```
$html = '<p>Hi ' . $name . '</p>';
```

Tainted


Untainted
Tainted
Untainted
$html = ' Hi ' . htmlspecialchars($name) . ' ';
Untainted
Escaped, to make it Safe :-)
Hello <script>alert()</script>

Unfortunately Taint Checking incorrectly assumes escaping makes a value “safe” for *any* context.

Untainted Tainted Untainted

```
$html = "<a href='" . htmlspecialchars($url) . "'>Link</a>";
```

Escapes & ' " < > ... Is this Safe?

```
<a href='javascript:alert()'>Link</a>
```

Untainted

Tainted Untainted

```
$html = "<img src=" . htmlspecialchars($url) . " />";
```

Missing Quotes

Escapes & ' " < > ... Is this Safe?

Untainted

Tainted

Untainted

```
$html = "<img src='" . htmlspecialchars($url) . "' />";
```

Before PHP 8.1, single quotes were not encoded by default :-)

Is this Safe?

Untainted

Tainted

```
$sql = "WHERE id = " . mysqli_real_escape_string($link, $id);
```

Missing Quotes

Escaped ... Is this Safe?

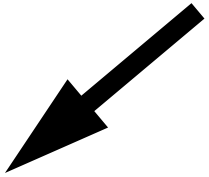
WHERE id = -1 UNION SELECT * FROM admin

Taint Checking is close,
but escaping should be done by a 3rd party library.

Instead, we can simplify it,
by looking for "strings from a trusted developer"...

Safe* vs Unsafe

When talking about Injection Vulnerabilities



Safe*

A string defined by the programmer.
(in the source code)

Unsafe

Everything else.

Safe*

```
$sql = 'SELECT * FROM user WHERE id = ?';
```

```
$db->query($sql, $id);
```

Unsafe

Safe*

Unsafe

```
$sql = 'SELECT * FROM users WHERE id = ' . $id;
```

```
$db->query($sql);
```


```
$db->query($sql);
```

Unsafe

```
$sql = 'SELECT * FROM users WHERE id = ' . $id;
```

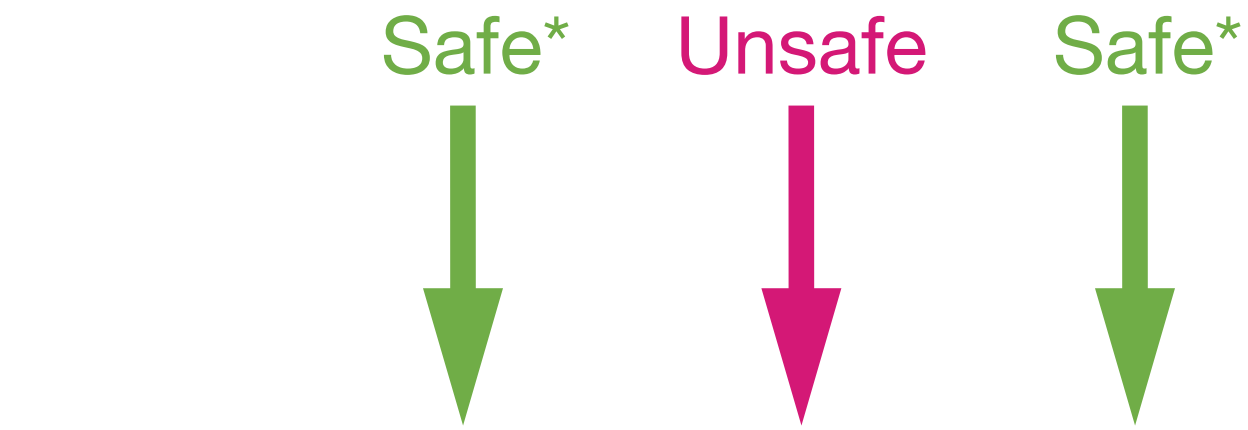
```
$db->query($sql);
```

Safe*

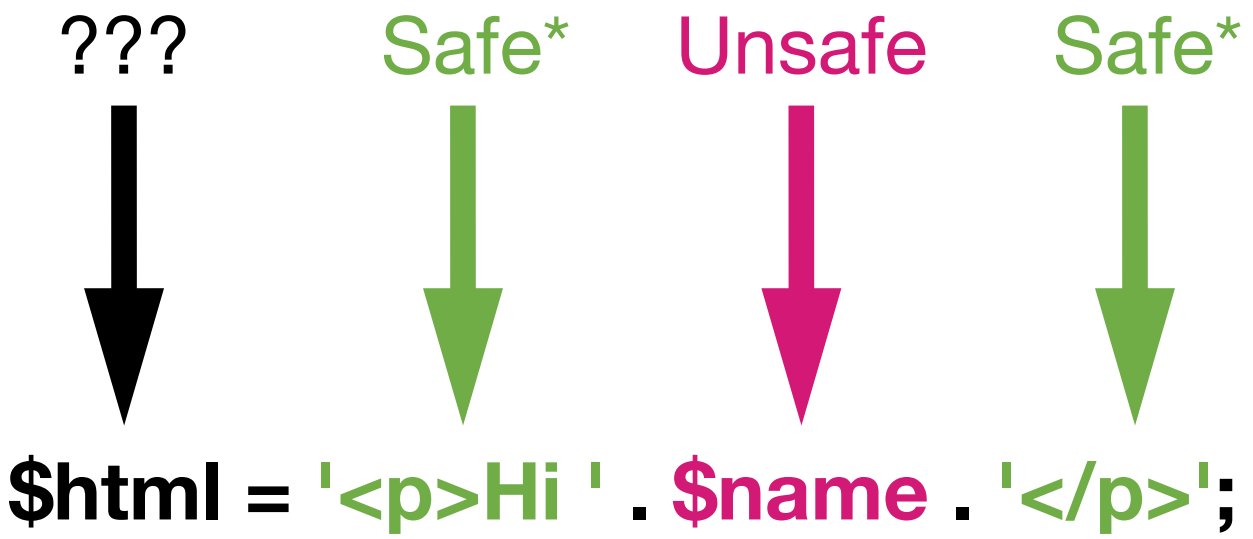
```
$html = '<p>Hi {{ name }}</p>';
```

```
$template->render($html, ['name' => $name]);
```

Unsafe




```
$html = '<p>Hi ' . $name . '</p>';

$template->render($html);
```




```
$template->render($html);
```

Unsafe

```
$html = '<p>Hi ' . $name . '</p>';
```

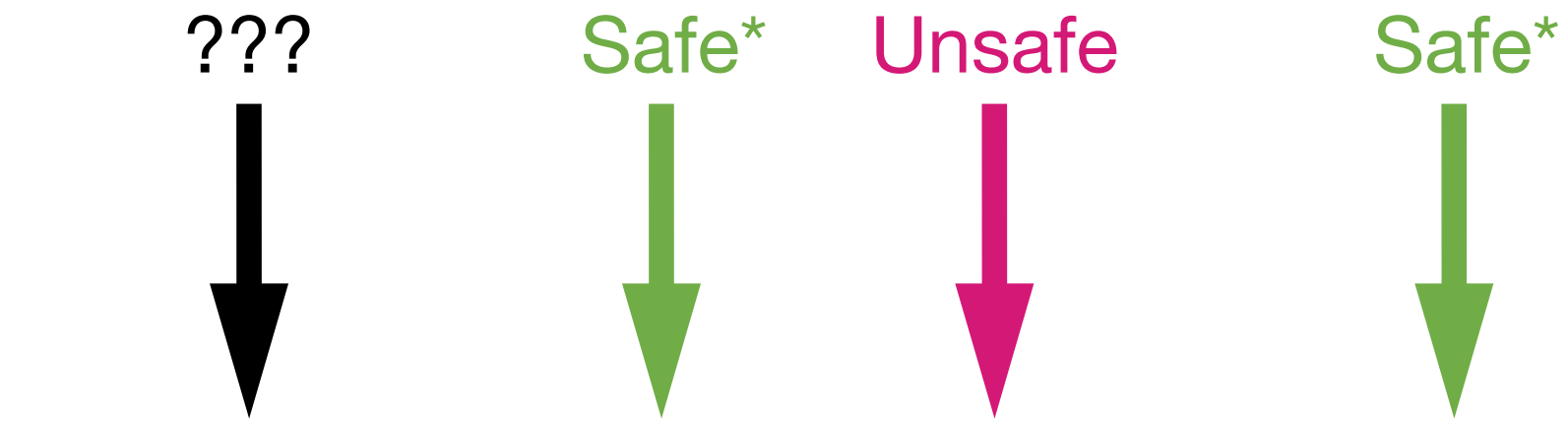
```
$template->render($html);
```

Safe*

```
$command = 'grep ? /path/to/file';

shell_exec($command, [$search]);
```

Unsafe


```
$command = 'grep "' . $search . '" /path/to/file';

shell_exec($command);
```




```
$command = 'grep "' . $search . '" /path/to/file';

shell_exec($command);
```

Unsafe

```
$command = 'grep "' . $search . '" /path/to/file';
```

```
shell_exec($command);
```

Remember, only "Safe" when talking about Injection Vulnerabilities.

Safe*

Unsafe

$value = "0000-00-00";

WHERE published > "0000-00-00"

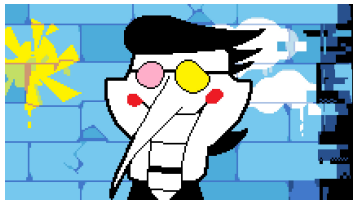
```
$articles->where('published > ', $date);
```

Remember, only "Safe" when talking about Injection Vulnerabilities.

Safe*

$path = "/";

rm -rf /



```
$command = 'rm -rf ?';

shell_exec($command, [$path]);
```

Unsafe

How to deal with Special Cases

Did you remember to
ensure all were integers?

```
$sql = 'WHERE id IN (' . implode(',', $ids) . ')';

$db->query($sql);
```

'WHERE id IN (1, 7, 9)'

WHERE id = (-1) UNION SELECT * FROM admin WHERE id IN (2)

```
$sql = 'WHERE id IN (' . in_parameters(count($ids)) . ')';

$db->query($sql, $ids);
```

```
$sql = 'WHERE id IN (' . in_parameters(count($ids)) . ')';

function in_parameters($count) {
    $sql = '?';
    for ($k = 1; $k < $count; $k++) {
        $sql .= ',?';
    }
    return $sql;
}
```

```
$sql = 'WHERE id IN (' . in_parameters(count($ids)) . ')';
```

```
function in_parameters($count) {
    return join(',', array_fill(0, $count, '?'));
}
```

```
$sql = 'WHERE id IN (' . in_parameters(count($ids)) . ')';
```

Be careful with no $ids

Could try to escape the field...
But should *any* field be allowed?

```
$sql = 'ORDER BY ' . $order_field;
```

```
$order_fields = [
    'name',
    'email',
    'created',
  ];

$order_id = array_search($order_field, $order_fields);

$sql = 'ORDER BY ' . $order_fields[$order_id];
```

List of Allowed fields

Use array for the trusted "programmer defined strings"

What about config values?

e.g. table names, set in an INI/JSON/YAML file.

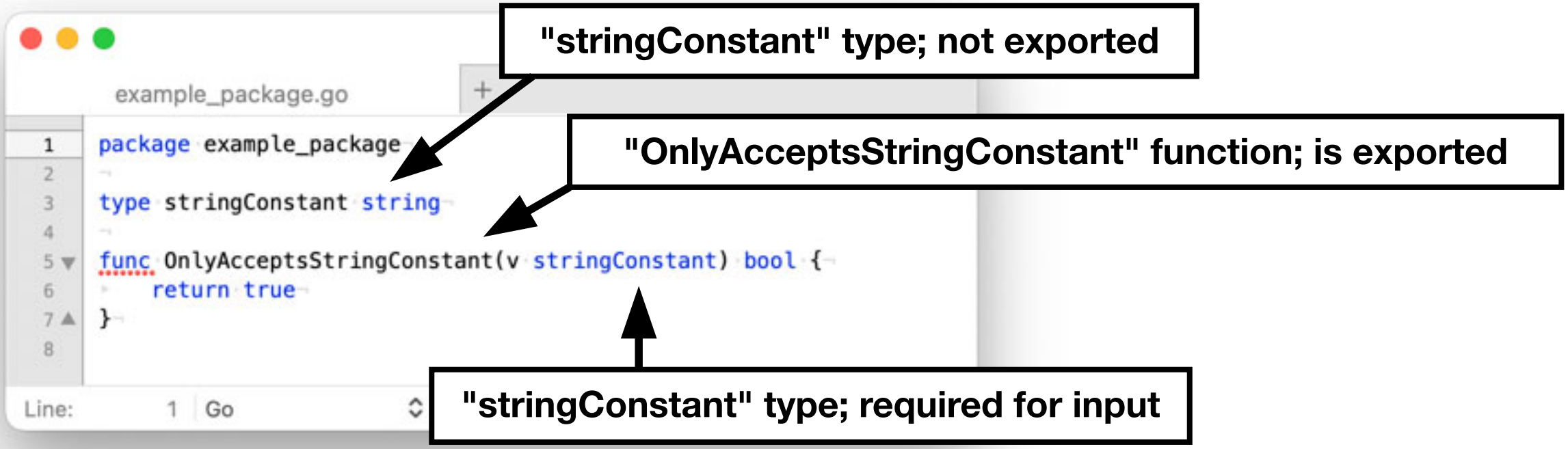
This will be covered after the next section :-)

But in short, the library needs to handle these (safely).

Ending Injection Vulnerabilities

In Go

Thanks to Dima Kotik and Roberto Clapis


"stringConstant" type; not exported
"OnlyAcceptsStringConstant" function; is exported
"stringConstant" type; required for input
example_package.go
1 package example_package
2
3 type stringConstant string
4
5 func OnlyAcceptsStringConstant(v stringConstant) bool {
6 return true
7 }
8
Line: 1 Go

example_code.go

```go
package main

import (
    "bufio"
    "fmt"
    "os"

    "example.com/example_package"
)

func main() {

    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Your Name: ")
    your_name, _ := reader.ReadString('\n')
    your_name = your_name[:len(your_name)-1]

    a := example_package.OnlyAcceptsStringConstant("<p>Hello ")

    b := example_package.OnlyAcceptsStringConstant(your_name)

}
```

example_package.go

```go
package example_package

type stringConstant string

func OnlyAcceptsStringConstant(v stringConstant) bool {
    return true
}
```

Line: 1 Go Tab Size: 4 Symbols

Get name (untrusted data)

An "Untyped String"

Go uses Type Conversion, turning this into a "stringConstant"

example_code.go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7
8      "e
9  )
10
11 func m
12
13     re
14     fm
15     your_name, _ := reader.ReadString('\n')
16     your_name = your_name[:len(your_name)-1]
17
18     a := example_package.OnlyAcceptsStringConstant("<p>Hello ")
19
20     b := example_package.OnlyAcceptsStringConstant(your_name)
21
22 }
23
```

example_package.go

```
1  package example_package
2
3  type stringConstant string
4
5  func OnlyAcceptsStringConstant(v stringConstant) bool {
6      return true
7  }
8
```

Terminal

```
craig$ go run example_code.go
# command-line-arguments
./example_code.go:20:48: cannot use your_name (type string) as type example_package.stringConstant
in argument to example_package.OnlyAcceptsStringConstant
craig$
```

Cannot be converted to a stringConstant

How this is used by "go-safe-html"

How this is used by "github.com/google/safehtml"

```
a := safehtml.ScriptFromConstant("var my_script = 'example';")
```

```
b := safehtml.ScriptFromConstant(your_name)
```

```
a := template.MustParseAndExecuteToHTML("<p>Hello ")
b := safehtml.HTMLEscaped(your_name)
c := template.MustParseAndExecuteToHTML("</p>")

para := safehtml.HTMLConcat(a, b, c)
```

```
<p>Hello &lt;script&gt;alert()&lt;/script&gt;</p>
```

```
a := template.MustParseAndExecuteToHTML("<p>Hello ")
b := template.MustParseAndExecuteToHTML(your_name)
c := template.MustParseAndExecuteToHTML("</p>")

para := safehtml.HTMLConcat(a, b, c)
```

```
a := template.MustParseAndExecuteToHTML("<p>Hello ")

c := template.MustParseAndExecuteToHTML("</p>")

para := safehtml.HTMLConcat(a, your_name, c)
```

Ending Injection Vulnerabilities

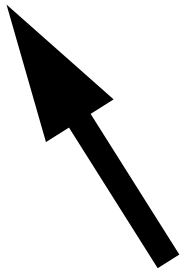
With Node... and JavaScript (soon)

Template Literal

```
url = 'javascript:alert()';

html = `<a href="${url}">Link</a>`;
```

```
'<a href="javascript:alert()">Link</a>'
```

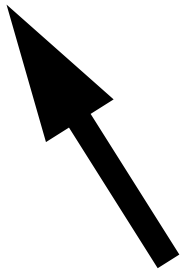
```
url = 'javascript:alert()';

my_template`<a href="${url}">Link</a>`;
```



Tagged Templates

```
function my_template(html, ...values) {
  console.log(html);
  console.log(values);
}

url = 'javascript:alert()';

my_template`<a href="${url}">Link</a>`;
```

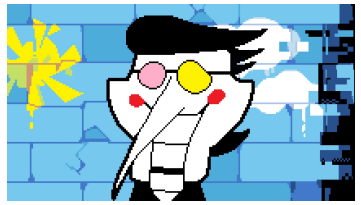
```
[
  '<a href="',
  '">Link</a>'
]

[
  'javascript:alert()'
]
```



Tagged Templates

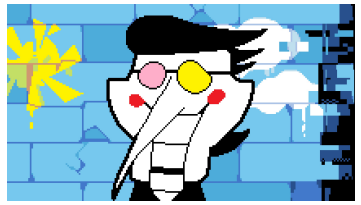
```
function my_template(html, ...values) {
  console.log(html);
  console.log(values);
}

url = 'javascript:alert()';

my_template`<a href="${url}">Link</a>`;

my_template([`<a href="${url}">Link</a>`]);
```

```
['<a href="javascript:alert()">Link</a>']
[ ]
```



```
function my_template(html, ...values) {
  console.log(html);
  console.log(values);
}

url = 'javascript:alert()';

my_template`<a href="${url}">Link</a>`;

my_template([`<a href="${url}">Link</a>`]);

my_template(['<a href="', url, '">Link</a>']);
```

```
['<a href="', 'javascript:alert()', '">Link</a>']
[ ]
```

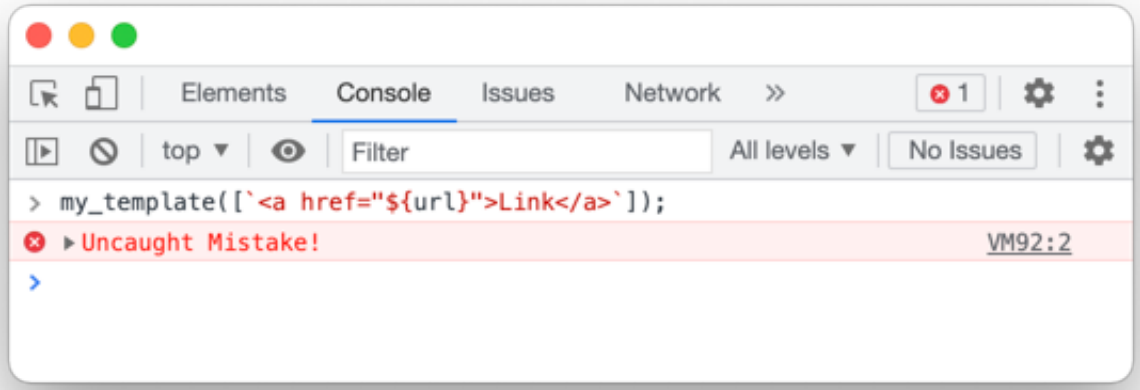


Array.isTemplateObject();

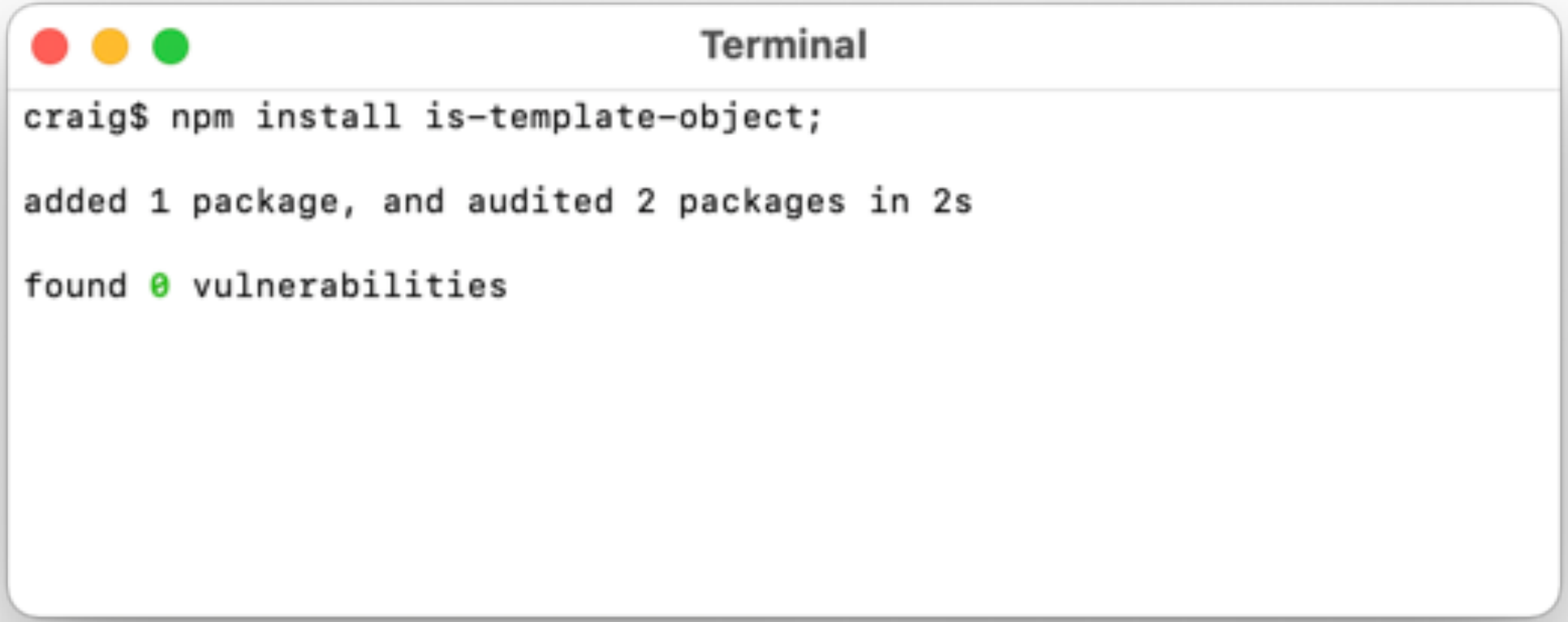
```
function my_template(html, ...values) {
  if (!isTemplateObject(html)) { throw 'Mistake!'; }
  ...
}

url = 'javascript:alert()';

my_template`<a href="${url}">Link</a>`;
```

```
my_template([`<a href="${url}">Link</a>`]);

my_template(['<a href="', url, '">Link</a>']);
```

```
function my_template(html, ...values) {
  if (!isTemplateObject(html)) { throw 'Mistake!'; }
  ...
}

url = 'javascript:alert()';

my_template`<a href="${url}">Link</a>`;

my_template([`<a href="${url}">Link</a>`]);

my_template(['<a href="', url, '">Link</a>']);
```



How this works in Node...


Terminal
craig$ npm install is-template-object;
added 1 package, and audited 2 packages in 2s
found 0 vulnerabilities


Require Polyfill, from Mike Samuel
Call isTemplateObject
index.js
isTemplateObject = require('is-template-object');
function template(html, ...values) {
console.log(isTemplateObject(html), html, values);
}
//
var username = "<script>alert()</script>"; // From evil user
template`<p>Hi ${username}<p>`;
template([`<p>Hi ${username}<p>`]); // Wrong
template(['<p>Hi ', username, '<p>']); // Wrong

```
craig$ node index.js;
true [ '<p>Hi ', '<p>' ] [ '<script>alert()</script>' ]
false [ '<p>Hi <script>alert()</script><p>' ] []
false [ '<p>Hi ', '<script>alert()</script>', '<p>' ] []
```

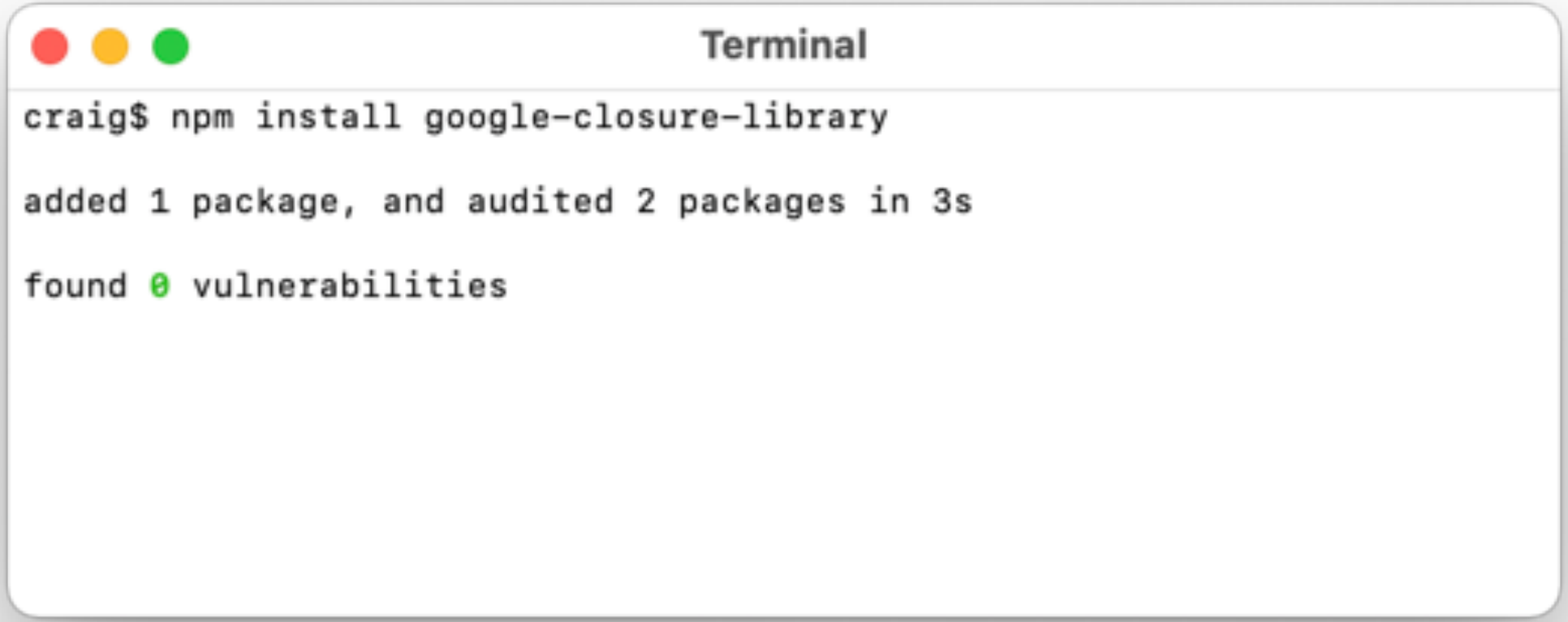
Coming to JavaScript... Soon™

https://github.com/tc39/proposal-array-is-template-object

Thanks to Krzysztof Kotowicz and Mike Samuel

Or a different approach,

Node with Google's Closure Library...


Terminal
craig$ npm install google-closure-library
added 1 package, and audited 2 packages in 3s
found 0 vulnerabilities

```
1
2   require("google-closure-library");
3
4   goog.require("goog.string.Const");
5
6   console.log("Your Name: ");
7
8   process.stdin.once('data', (chunk) => {
9
10      var your_name = chunk.toString();
11
12      var a = goog.string.Const.from("<p>Hello, ");
13
14      var b = goog.string.Const.from(your_name);
15
16      console.log(a.toString());
17      console.log(b.toString());
18
19  });
```

Require "Closure Library", and "goog.string.Const"

Get name (untrusted data)

Use goog.string.Const.from()

Terminal

```
craig$ node index.js
Transpiled debug/error.js in 0.14 seconds
Your Name:
Craig
Const{<p>Hello, }
Const{Craig
}
```

```
11
12      var a = goog.string.Const.from("<p>Hello, ");
13
14      var b = goog.string.Const.from(your_name);
15
16      console.log(a.toString());
17      console.log(b.toString());
18
19  });
```

Terminal

```
craig$ curl https://repo1.maven.org/maven2/com/google/javascript/closure-compiler/v20211006/closure-compiler-v2
0211006.jar --output closure-compiler.jar;
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 12.5M  100 12.5M    0     0  32.0M      0 --:--:-- --:--:-- --:--:-- 31.9M
craig$
```

```
11
12      var a = goog.string.Const.from("<b>Hello, ");
13
14      var b = goog.string.Const.from(your_name);
15
16      console.log(a.toString());
17      console.log(b.toString());
18
19  });
```

Terminal

```
craig$ java -jar closure-compiler.jar --js ./node_modules/google-closure-library --js index.js --js_output_file
 index-compiled.js

index.js:14:32: ERROR - [JSC_CONSTANT_NOT_STRING_LITERAL_ERROR] Function argument is not a string literal or a
constant assigned from a string literal or a concatenation of these.
  14|   var b = goog.string.Const.from(your_name);
                                       ^^^^^^^^^

1 error(s), 0 warning(s)
```

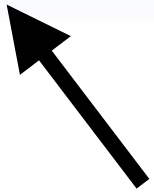
```
11
12    var a = goog.string.Const.from("<b>Hello, ");
13
14    var b = goog.string.Const.from(your_name);
15
16    console.log(a.toString());
17    console.log(b.toString());
18
19 });
```

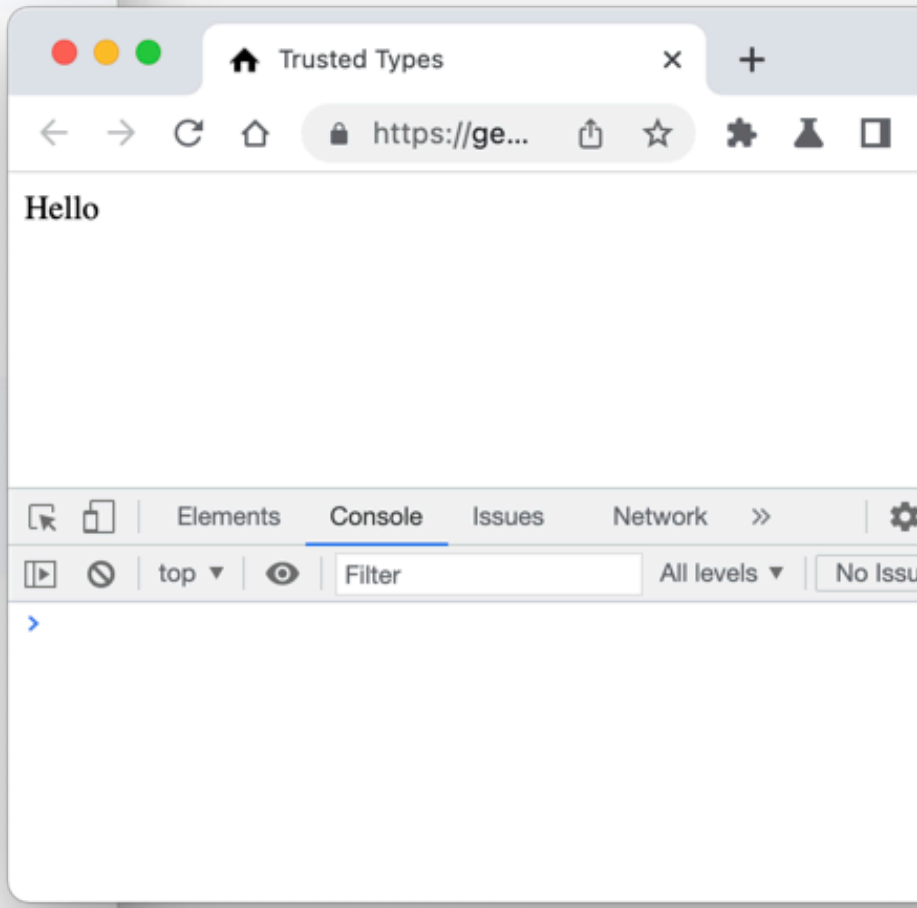
For DOM XSS Protection

or JavaScript and Trusted Types... Hopefully Soon™

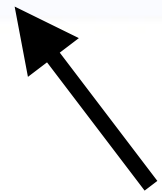
https://github.com/w3c/webappsec-trusted-types/issues/347

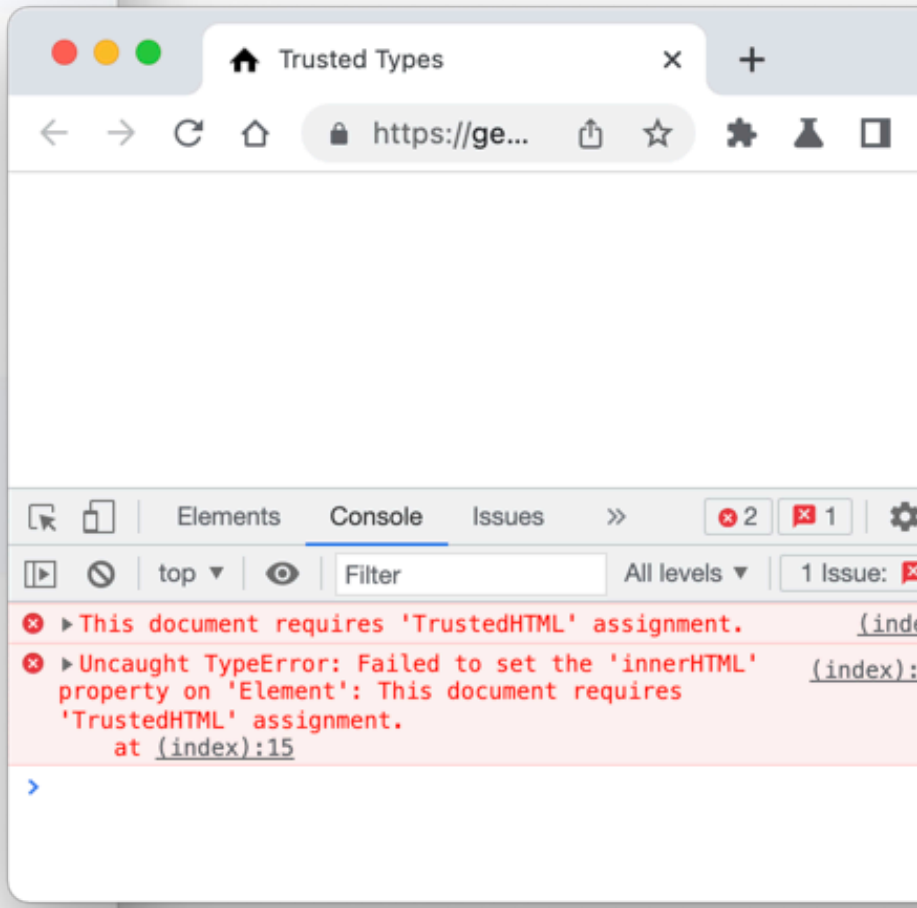
```php
<?php

    header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types 'none';");

?>
<!DOCTYPE html>
<html lang="en-GB">
<head>
    <meta charset="UTF-8" />
    <title>Trusted Types</title>
</head>
<body>

    <div id="output"></div>

    <script type="text/javascript">

        var output = document.getElementById('output');

        output.textContent = 'Hello';

    </script>

</body>
</html>
```





```php
<?php

    header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types 'none';");

?>
<!DOCTYPE html>
<html lang="en-GB">
<head>
    <meta charset="UTF-8" />
    <title>Trusted Types</title>
</head>
<body>

    <div id="output"></div>

    <script type="text/javascript">

        var output = document.getElementById('output');

        output.innerHTML = 'Hello';

    </script>

</body>
</html>
```



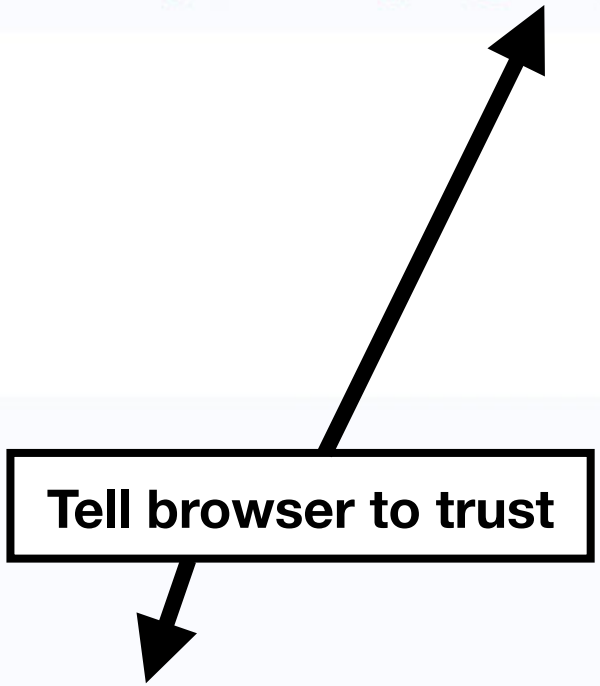


```php
<?php

    header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");

?>
<!DOCTYPE html>
<html lang="en-GB">
<head>
    <meta charset="UTF-8" />
    <title>Trusted Types</title>
</head>
<body>

    <div id="output"></div>

    <script type="text/javascript">

        var my_trusted_type = {
            'createHTML': function (s) {
                return s; // Dangerous
            }
        };

        if (window.trustedTypes) {
            my_trusted_type = window.trustedTypes.createPolicy('my_trusted_type', my_trusted_type);
        }

        var output = document.getElementById('output');

        output.innerHTML = my_trusted_type.createHTML('Hello');

    </script>
```

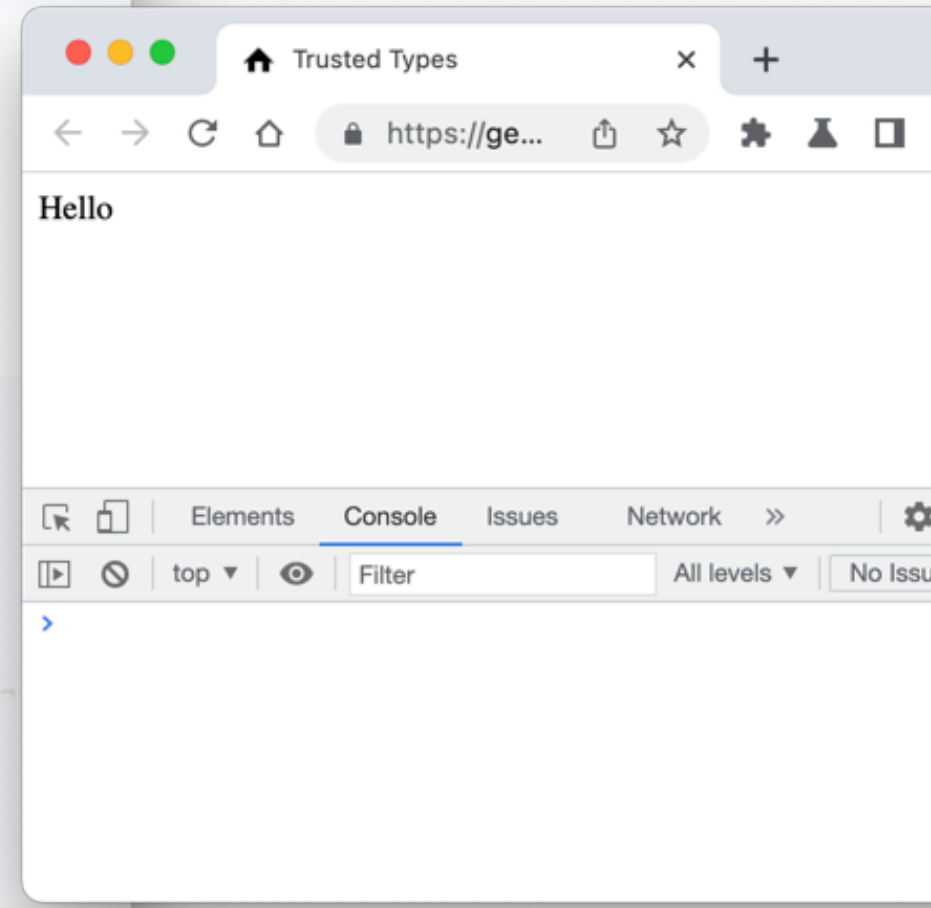
Our Trusted Type.

Use DOMPurify, "Sanitizer" API, etc.

```php
<?php

    header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");

?>
<!DOCTYPE html>
<html lang="en-GB">
<head>
    <meta charset="UTF-8" />
    <title>Trusted Types</title>
</head>
<body>

    <div id="output"></div>

    <script type="text/javascript">

        var my_trusted_type = {
            'createHTML': function (s) {
                return s; // Dangerous
            }
        };

        if (window.trustedTypes) {
            my_trusted_type = window.trustedTypes.createPolicy('my_trusted_type', my_trusted_type);
        }

        var output = document.getElementById('output');

        output.innerHTML = my_trusted_type.createHTML('Hello');

    </script>
```



```php
<?php

    header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");

?>
<!DOCTYPE html>
<html lang="en-GB">
<head>
    <meta charset="UTF-8" />
    <title>Trusted Types</title>
</head>
<body>

    <div id="output"></div>

    <script type="text/javascript">

        var my_trusted_type = {
            'createHTML': function (s) {
                return s; // Dangerous
            }
        };

        if (window.trustedTypes) {
            my_trusted_type = window.trustedTypes.createPolicy('my_trusted_type', my_trusted_type);
        }

        var output = document.getElementById('output');

        output.innerHTML = my_trusted_type.createHTML('Hello');

    </script>
```



Use it

```php
<?php

    header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");

?>
<!DOCTYPE html>
<html lang="en-GB">
<head>
    <meta charset="UTF-8" />
    <title>Trusted Types</title>
</head>
<body>

    <div id="output"></div>

    <script type="text/javascript">

        var output = document.getElementById('output');

        output.innerHTML = window.trustedHTML.fromLiteral`Hello`;

    </script>

</body>
</html>
```

Ending Injection Vulnerabilities

In Java

Can be done in Java with Error Prone

https://errorprone.info

It runs extra checks at Compile Time

Adding to Maven

There are a few dependencies

index.java

```java
import java.util.Scanner;
import com.google.errorprone.annotations.CompileTimeConstant;

public class index {

    private static void print_constant(@CompileTimeConstant String value) {
        System.out.println(value);
    }

    public static void main(String[] args) {

        System.out.print("Your Name: ");
        Scanner scanner = new Scanner(System.in);
        String your_name = scanner.next();
        scanner.close();

        print_constant("Compile Time Constant");

        print_constant(your_name);

    }

}
```

Import

Type Annotation Check

Get name (untrusted data)

Call Sensitive Function

index.java

```java
import java.util.Scanner;
import com.google.errorprone.annotations.CompileTimeConstant;

public class index {

    private static void print_constant(@CompileTimeConstant String value) {
        System.out.println(value);
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String your_name = in.nextLine();

        print_constant("Compile Time Constant");

        print_constant(your_name);

    }

}
```

Terminal

```
[INFO] -------------------------------------------------------------
[ERROR] COMPILATION ERROR :
[INFO] -------------------------------------------------------------
[ERROR] index.java:[20,17] error: [CompileTimeConstant] Non-compile-time constant expression passed to parameter with
        @CompileTimeConstant type annotation.
[INFO] 1 error
[INFO] -------------------------------------------------------------
```

Java could also use Google GWT?

SafeHtml.fromSafeConstant()

http://www.gwtproject.org/javadoc/latest/com/google/gwt/safehtml/shared/SafeHtmlUtils.html#fromSafeConstant-java.lang.String-

HTTP, not HTTPS, abandoned project?

Ending Injection Vulnerabilities

In C++

"Use a template constructor that depends on each character value in the string."

- Building Secure and Reliable Systems

The following C++ example creates a TrustedResourceUrl from a compile-time constant:

```
TrustedResourceUrl url = TrustedResourceUrl::FromConstant(
    "http://www.google-analytics.com/analytics.js");
```

The compiler will enforce that FromConstant() is only called with a compile-time constant.

- Safe HTML Types Overview, Google

¯_(ツ)_/¯

Ending Injection Vulnerabilities

In PHP

Using Psalm

Static Analysis 1

Thanks to Matthew Brown

composer require --dev vimeo/psalm


Terminal
craig$ composer require --dev vimeo/psalm
Using version ^4.12 for vimeo/psalm
./composer.json has been created
Running composer update vimeo/psalm
Loading composer repositories with package information
Updating dependencies
Lock file operations: 30 installs, 0 updates, 0 removals
- Locking amphp/amp (v2.6.1)
- Locking amphp/byte-stream (v1.8.1)
- Locking composer/package-versions-deprecated (1.11.99.4)
- Locking composer/semver (3.2.6)
- Locking composer/xdebug-handler (2.0.2)
- Locking dnoegel/php-xdg-base-dir (v0.1.1)
- Locking felixfbecker/advanced-json-rpc (v3.2.1)
- Locking felixfbecker/language-server-protocol (1.5.1)
- Locking netresearch/jsonmapper (v4.0.0)
- Locking nikic/php-parser (v4.13.1)
- Locking openlss/lib-array2xml (1.0.0)
- Locking phpdocumentor/reflection-common (2.2.0)
- Locking phpdocumentor/reflection-docblock (5.3.0)
- Locking phpdocumentor/type-resolver (1.5.1)
- Locking psr/container (1.1.2)
- Locking psr/log (2.0.0)
- Locking sebastian/diff (4.0.4)
- Locking symfony/console (v5.3.10)
- Locking symfony/deprecation-contracts (v2.4.0)
- Locking symfony/polyfill-ctype (v1.23.0)
- Locking symfony/polyfill-intl-grapheme (v1.23.1)
- Locking symfony/polyfill-intl-normalizer (v1.23.0)
- Locking symfony/polyfill-mbstring (v1.23.1)
- Locking symfony/polyfill-php73 (v1.23.0)
- Locking symfony/polyfill-php80 (v1.23.1)
- Locking symfony/service-contracts (v2.4.0)
- Locking symfony/string (v5.3.10)
- Locking vimeo/psalm (4.12.0)
- Locking webmozart/assert (1.10.0)
- Locking webmozart/path-util (2.3.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 30 installs, 0 updates, 0 removals
- Downloading webmozart/assert (1.10.0)
- Downloading phpdocumentor/reflection-docblock (5.3.0)
- Downloading psr/container (1.1.2)
- Downloading symfony/string (v5.3.10)
- Downloading symfony/console (v5.3.10)
- Downloading nikic/php-parser (v4.13.1)
- Downloading composer/semver (3.2.6)
- Downloading vimeo/psalm (4.12.0)
- Installing composer/package-versions-deprecated (1.11.99.4): Extracting archive
- Installing symfony/polyfill-ctype (v1.23.0): Extracting archive
- Installing webmozart/assert (1.10.0): Extracting archive
- Installing phpdocumentor/reflection-common (2.2.0): Extracting archive
- Installing phpdocumentor/type-resolver (1.5.1): Extracting archive
- Installing phpdocumentor/reflection-docblock (5.3.0): Extracting archive
- Installing psr/log (2.0.0): Extracting archive
- Installing symfony/deprecation-contracts (v2.4.0): Extracting archive
- Installing symfony/polyfill-php73 (v1.23.0): Extracting archive
- Installing psr/container (1.1.2): Extracting archive
- Installing symfony/service-contracts (v2.4.0): Extracting archive
- Installing symfony/polyfill-php80 (v1.23.1): Extracting archive
- Installing symfony/polyfill-mbstring (v1.23.1): Extracting archive
- Installing symfony/polyfill-intl-normalizer (v1.23.0): Extracting archive
- Installing symfony/polyfill-intl-grapheme (v1.23.1): Extracting archive
- Installing symfony/string (v5.3.10): Extracting archive
- Installing webmozart/path-util (2.3.0): Extracting archive
- Installing symfony/console (v5.3.10): Extracting archive
- Installing sebastian/diff (4.0.4): Extracting archive
- Installing openlss/lib-array2xml (1.0.0): Extracting archive
- Installing nikic/php-parser (v4.13.1): Extracting archive
- Installing netresearch/jsonmapper (v4.0.0): Extracting archive
- Installing felixfbecker/language-server-protocol (1.5.1): Extracting archive
- Installing felixfbecker/advanced-json-rpc (v3.2.1): Extracting archive
- Installing dnoegel/php-xdg-base-dir (v0.1.1): Extracting archive
- Installing composer/xdebug-handler (2.0.2): Extracting archive
- Installing composer/semver (3.2.6): Extracting archive
- Installing amphp/amp (v2.6.1): Extracting archive
- Installing amphp/byte-stream (v1.8.1): Extracting archive
- Installing vimeo/psalm (4.12.0): Extracting archive
5 package suggestions were added by new dependencies, use `composer suggest` to see details.
Package webmozart/path-util is abandoned, you should avoid using it. Use symfony/filesystem instead.
Generating autoload files
composer/package-versions-deprecated: Generating version class...
composer/package-versions-deprecated: ...done generating version class
16 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
craig$

./vendor/bin/psalm --init

Check Psalm is at level 3 or stricter.

(level 1 is the most strict)

```php
<?php

$id = (string) ($_GET['id'] ?? '');

class db {

    /**
     * @psalm-param literal-string $sql
     */

    public function query(string $sql, array $parameters = []) : void {

        // Send $sql and $parameters to the database.

    }

}

$db = new db();

$db->query('SELECT * FROM user WHERE id = ?', [$id]);

$db->query('SELECT * FROM user WHERE id = ' . $id);
```

Use 'literal-string' type for $sql

```php
<?php

$id = (string) ($_GET['id'] ?? '');
```

Terminal

```
craig$ ./vendor/bin/psalm
Scanning files...
Analyzing files...


ERROR: ArgumentTypeCoercion - public/index.php:23:12 - Argument 1 of db::query expects literal-string, parent type non-empty-string provided
(see https://psalm.dev/193)
$db->query('SELECT * FROM user WHERE id = ' . $id);


------------------------------
1 errors found
------------------------------

Checks took 0.00 seconds and used 4.375MB of memory
No files analyzed
Psalm was able to infer types for 100% of the codebase
craig$
```

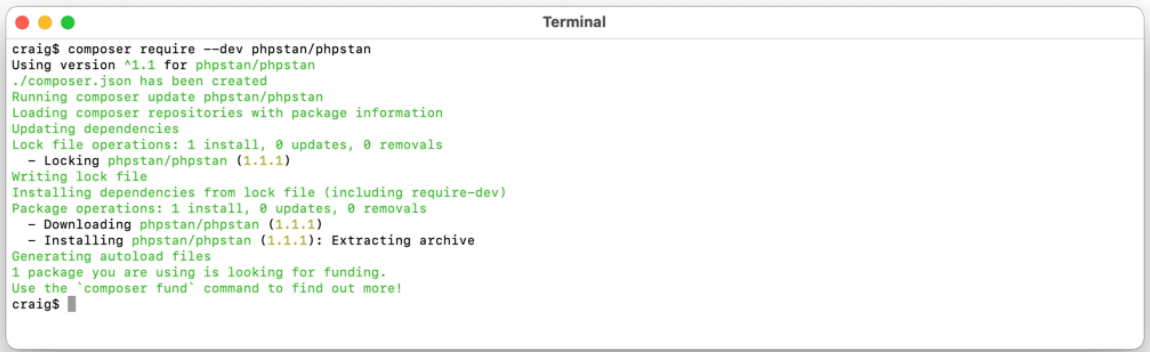
```php
$db->query('SELECT * FROM user WHERE id = ?', [$id]);

$db->query('SELECT * FROM user WHERE id = ' . $id);
```

Using PHPStan

Static Analysis 2

Thanks to Ondřej Mirtes

composer require --dev phpstan/phpstan


Terminal
craig$ composer require --dev phpstan/phpstan
Using version ^1.1 for phpstan/phpstan
./composer.json has been created
Running composer update phpstan/phpstan
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking phpstan/phpstan (1.1.1)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Downloading phpstan/phpstan (1.1.1)
- Installing phpstan/phpstan (1.1.1): Extracting archive
Generating autoload files
1 package you are using is looking for funding.
Use the `composer fund` command to find out more!
craig$

Check PHPStan is at level:

5 or stricter when an argument uses a single type.
7 or stricter when an argument uses multiple types.

(level 9 is the most strict)


```php
<?php

$id = (string) ($_GET['id'] ?? '');

class db {

    /**
     * @phpstan-param literal-string $sql
     * @phpstan-param array<int, string> $parameters
     */

    public function query(string $sql, array $parameters = []) : void {

        // Send $sql and $parameters to the database.

    }

}

$db = new db();

$db->query('SELECT * FROM user WHERE id = ?', [$id]);

$db->query('SELECT * FROM user WHERE id = ' . $id);
```

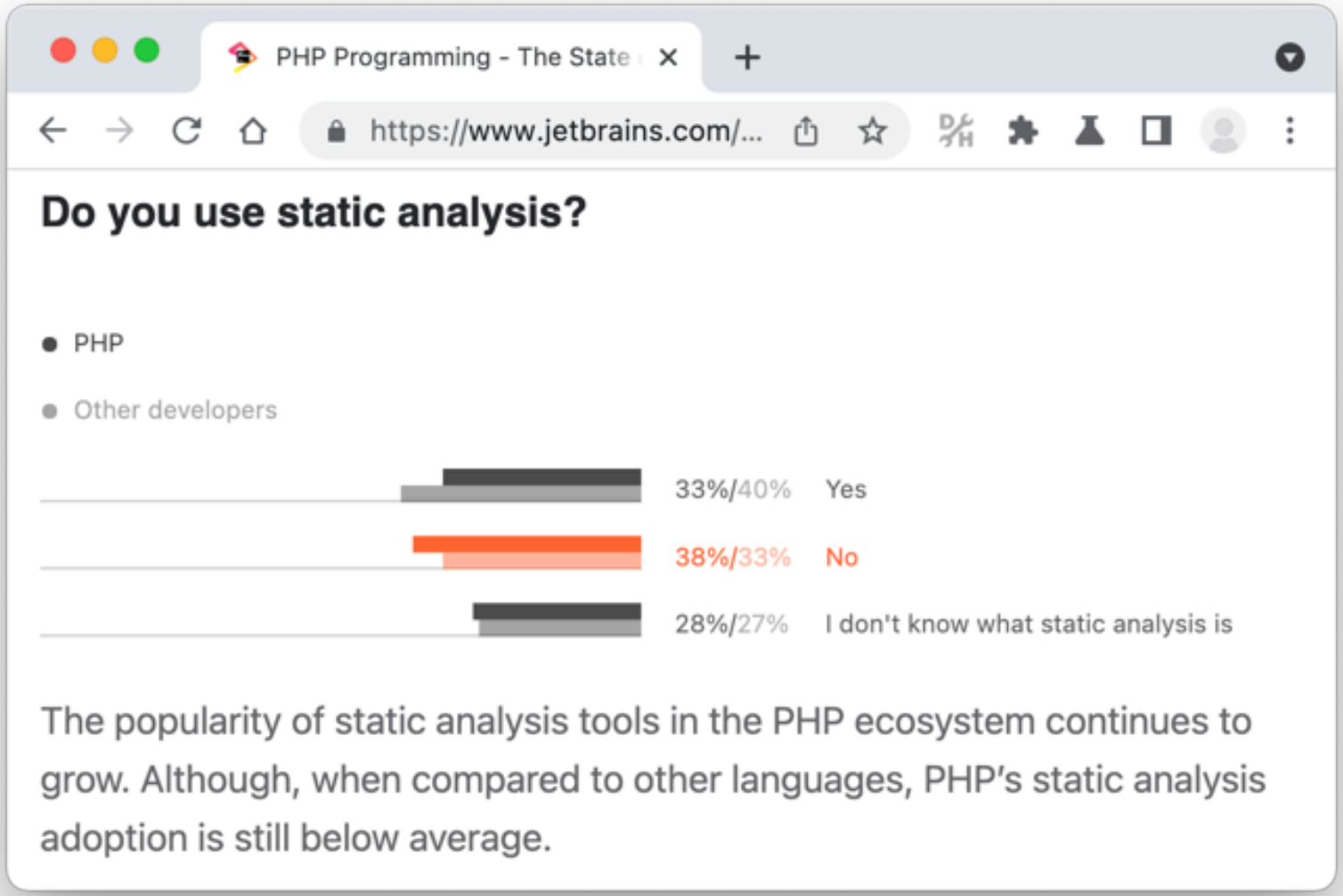
```php
<?php

$id = (string) ($_GET['id'] ?? '');

class db {

$db = new db();

$db->query('SELECT * FROM user WHERE id = ?', [$id]);

$db->query('SELECT * FROM user WHERE id = ' . $id);
```

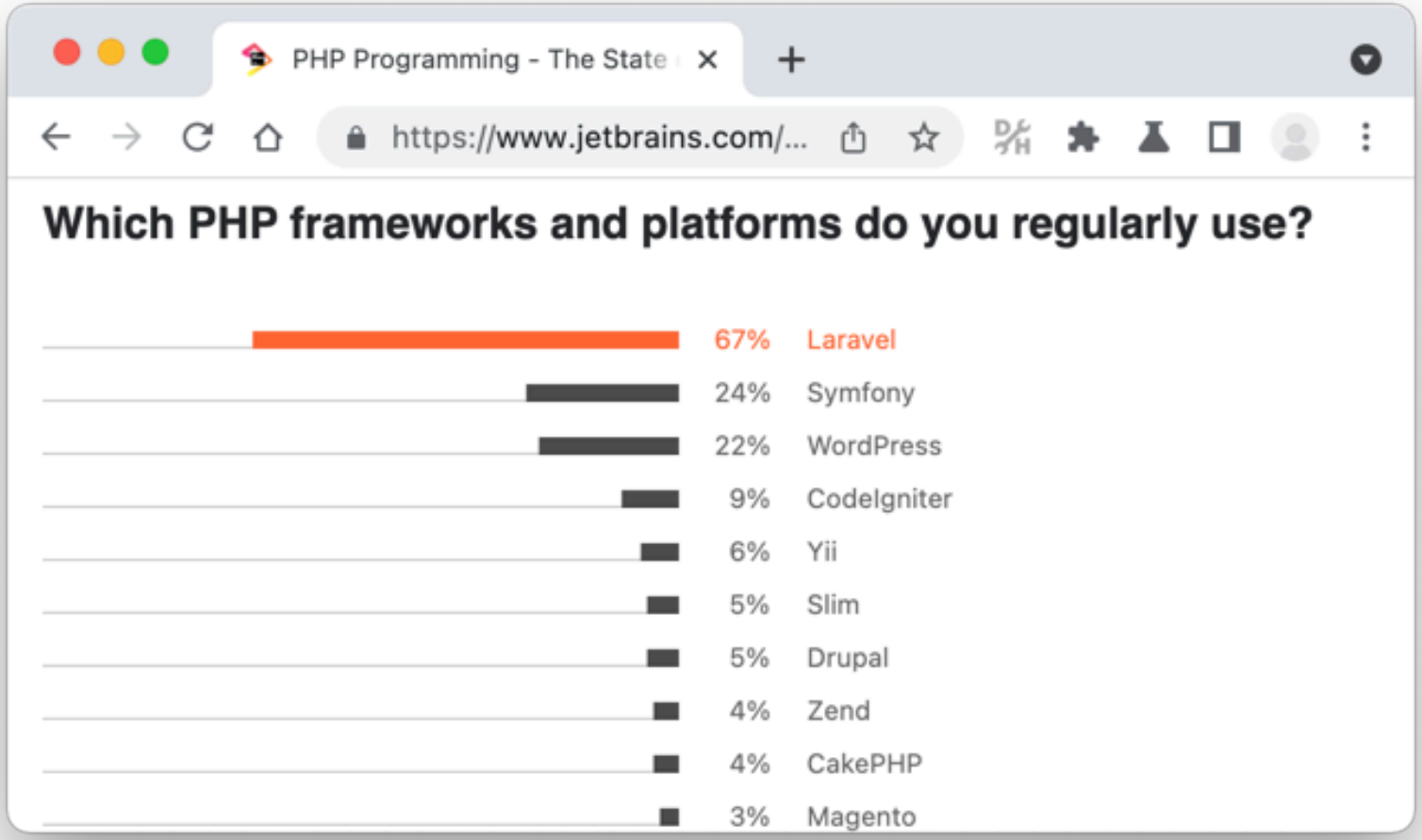
Terminal

```
craig$ vendor/bin/phpstan analyse --level 9 public
 1/1 [============================] 100%

 ------ ------------------------------------------------------------------------------------------
  Line   index.php
 ------ ------------------------------------------------------------------------------------------
  24     Parameter #1 $sql of method db::query() expects literal-string, non-empty-string given.
 ------ ------------------------------------------------------------------------------------------

 [ERROR] Found 1 error

craig$
```

The Future

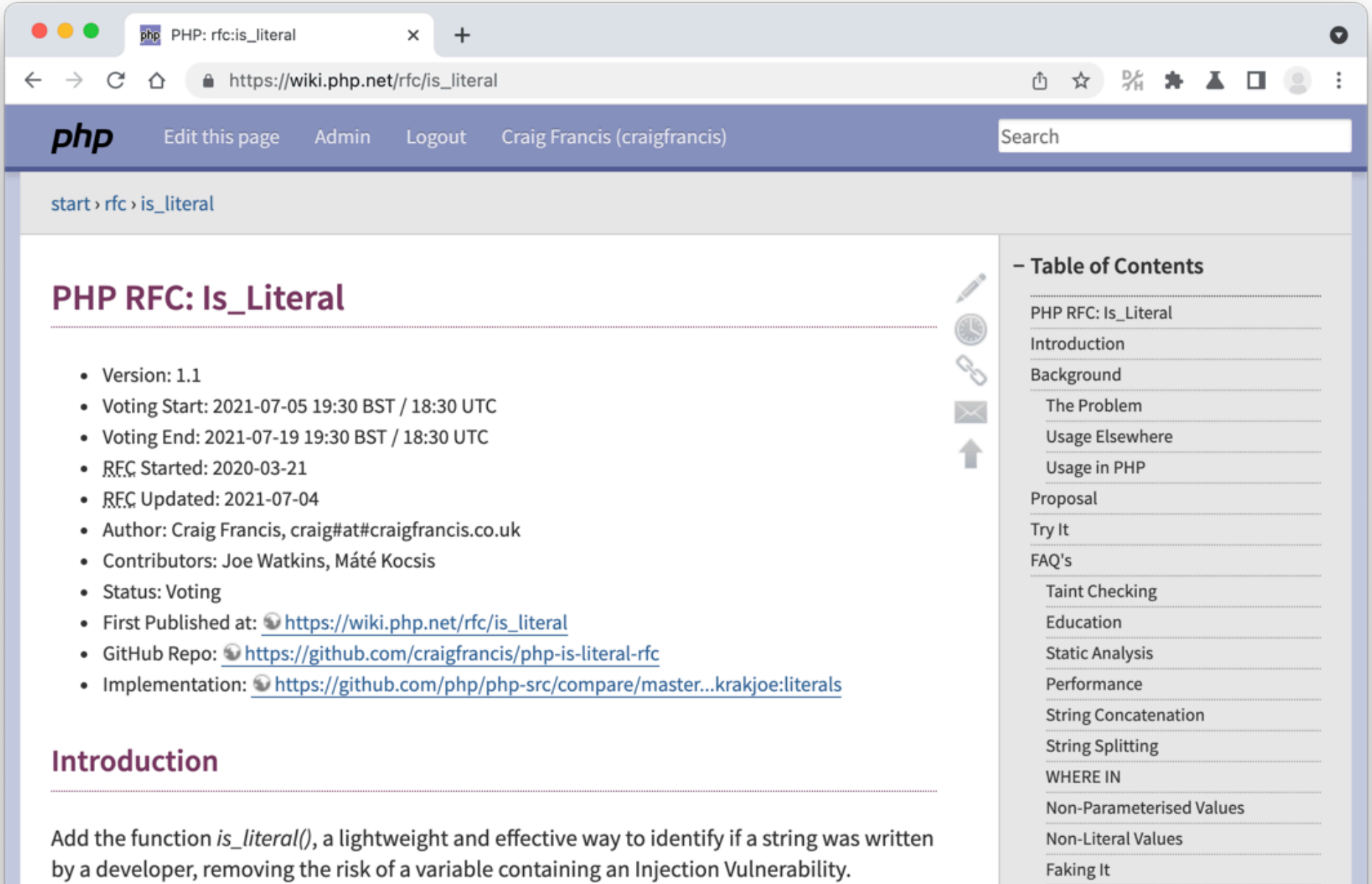
How programming languages can help


PHP Programming - The State
https://www.jetbrains.com/...
Do you use static analysis?
PHP
Other developers
33%/40%
Yes
38%/33%
No
28%/27%
I don't know what static analysis is
The popularity of static analysis tools in the PHP ecosystem continues to grow. Although, when compared to other languages, PHP's static analysis adoption is still below average.


PHP Programming - The State
https://www.jetbrains.com/...
Which PHP frameworks and platforms do you regularly use?
67% Laravel
24% Symfony
22% WordPress
9% CodeIgniter
6% Yii
5% Slim
5% Drupal
4% Zend
4% CakePHP
3% Magento

PHP, and the is_literal() RFC

Thanks to Joe Watkins and Máté Kocsis


PHP: rfc:is_literal
https://wiki.php.net/rfc/is_literal
php
Edit this page
Admin
Logout
Craig Francis (craigfrancis)
Search
start › rfc › is_literal
PHP RFC: Is_Literal
Version: 1.1
Voting Start: 2021-07-05 19:30 BST / 18:30 UTC
Voting End: 2021-07-19 19:30 BST / 18:30 UTC
RFC Started: 2020-03-21
RFC Updated: 2021-07-04
Author: Craig Francis, craig#at#craigfrancis.co.uk
Contributors: Joe Watkins, Máté Kocsis
Status: Voting
First Published at: https://wiki.php.net/rfc/is_literal
GitHub Repo: https://github.com/craigfrancis/php-is-literal-rfc
Implementation: https://github.com/php/php-src/compare/master...krakjoe:literals
Introduction
Add the function is_literal(), a lightweight and effective way to identify if a string was written by a developer, removing the risk of a variable containing an Injection Vulnerability.
Table of Contents
PHP RFC: Is_Literal
Introduction
Background
The Problem
Usage Elsewhere
Usage in PHP
Proposal
Try It
FAQ's
Taint Checking
Education
Static Analysis
Performance
String Concatenation
String Splitting
WHERE IN
Non-Parameterised Values
Non-Literal Values
Faking It

No dependencies.

Easy to use.

No *need* to use Static Analysis,

But works very well with it.

Works with existing code / libraries.

e.g. No need to re-write everything to use a query builder.

You can choose how to handle mistakes.

Log to a file/db/api, throw an exception, do nothing, etc.

Libraries won't need everyone to read/understand *all* of their documentation.

Libraries won't rely on developers never making a mistake.

Developers would be warned *as* they write their code.

(IDEs could highlight problems as they type, or as they run their code).

~0.47% Performance Impact

PHP 8.1 is roughly 30% faster than PHP 8.0 - Máté, 8th July 2021

Tested on Symfony Demo,
without connecting to the database (too much variability)

```php
<?php

class db {

    private function literal_check($var) {

        if (function_exists('is_literal') && !is_literal($var)) {
            throw new Exception('Non-literal value detected!');
        }

    }

    public function query($sql, $parameters = []) {

        $this->literal_check($sql);

        // Send $sql and $parameters to the database.

    }
```

Backwards Compatibility

```php
<?php

class db {

    private function literal_check($var) {

        if (function_exists('is_literal') && !is_literal($var)) {
            throw new Exception('Non-literal value detected!');
        }

    }

    public function query($sql, $parameters = []) {

        $this->literal_check($sql);

        // Send $sql and $parameters to the database.

    }

```

Run the check

```php
<?php

class db {

    private function literal_check($var) {

        if (function_exists('is_literal') && !is_literal($var)) {
            throw new Exception('Non-literal value detected!');
        }

    }

    public function query($sql, $parameters = []) {

        $this->literal_check($sql);

        // Send $sql and $parameters to the database.

    }

```

Too Strict?

```php
<?php

class db {

    private $protection_level = 1;
        // 0 = No checks, could be useful on the production server.
        // 1 = Just warnings, the default.
        // 2 = Exceptions, for anyone who wants to be absolutely sure.

    public function enforce_injection_protection() {
        $this->protection_level = 2;
    }

    public function unsafe_disable_injection_protection() {
        $this->protection_level = 0; // Not recommended, try `new unsafe_value('XXX')`
    }

    private function literal_check($var) {
        if (!function_exists('is_literal') || is_literal($var)) {
            // Fine - This is a programmer defined string (bingo), or not using PHP 8.X
        } else if ($var instanceof unsafe_value) {
            // Fine - Not ideal, but at least they know this one is unsafe.
        } else if ($this->protection_level === 0) {
            // Fine - Programmer aware, and is choosing to disable this check everywhere.
        } else if ($this->protection_level === 1) {
            trigger_error('Non-literal value detected!', E_USER_WARNING);
        } else {
            throw new Exception('Non-literal value detected!');
        }
    }
```

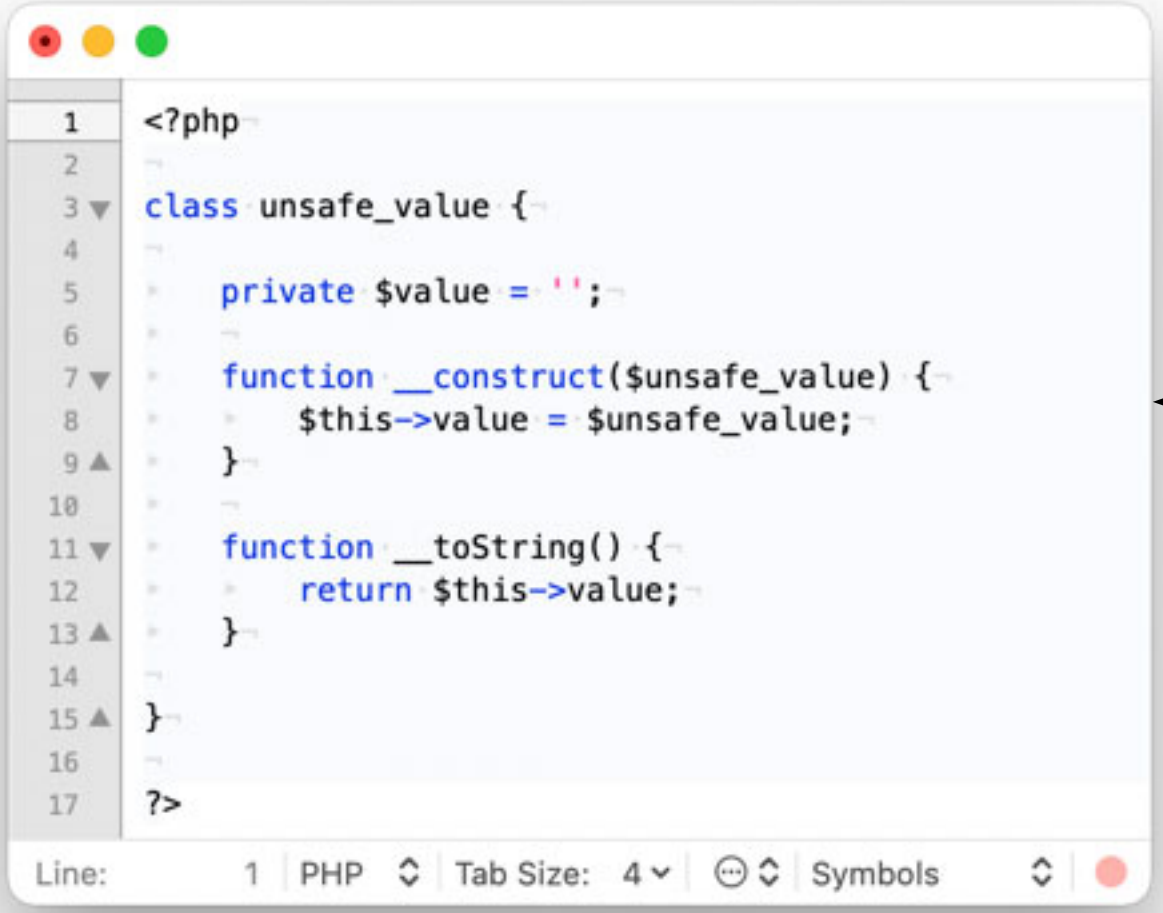
Protection Level

```php
<?php

class db {

    private $protection_level = 1;
        // 0 = No checks, could be useful on the production server.
        // 1 = Just warnings, the default.
        // 2 = Exceptions, for anyone who wants to be absolutely sure.

    public function enforce_injection_protection() {
        $this->protection_level = 2;
    }

    public function unsafe_disable_injection_protection() {
        $this->protection_level = 0; // Not recommended, try `new unsafe_value('XXX')`
    }

    private function literal_check($var) {
        if (!function_exists('is_literal') || is_literal($var)) {
            // Fine - This is a programmer defined string (bingo), or not using PHP 8.X
        } else if ($var instanceof unsafe_value) {
            // Fine - Not ideal, but at least they know this one is unsafe.
        } else if ($this->protection_level === 0) {
            // Fine - Programmer aware, and is choosing to disable this check everywhere.
        } else if ($this->protection_level === 1) {
            trigger_error('Non-literal value detected!', E_USER_WARNING);
        } else {
            throw new Exception('Non-literal value detected!');
        }
    }
```

Private function, Used by the library

```php
<?php

class db {

    private $protection_level = 1;
        // 0 = No checks, could be useful on the production server.
        // 1 = Just warnings, the default.
        // 2 = Exceptions, for anyone who wants to be absolutely sure.

    public function enforce_injection_protection() {
        $this->protection_level = 2;
    }

    public function unsafe_disable_injection_protection() {
        $this->protection_level = 0; // Not recommended, try `new unsafe_value('XXX')`
    }

    private function literal_check($var) {
        if (!function_exists('is_literal') || is_literal($var)) {
            // Fine - This is a programmer defined string (bingo), or not using PHP 8.X
        } else if ($var instanceof unsafe_value) {
            // Fine - Not ideal, but at least they know this one is unsafe.
        } else if ($this->protection_level === 0) {
            // Fine - Programmer aware, and is choosing to disable this check everywhere.
        } else if ($this->protection_level === 1) {
            trigger_error('Non-literal value detected!', E_USER_WARNING);
        } else {
            throw new Exception('Non-literal value detected!');
        }
    }
```

Special Cases?



```php
<?php

class unsafe_value {

    private $value = '';

    function __construct($unsafe_value) {
        $this->value = $unsafe_value;
    }

    function __toString() {
        return $this->value;
    }

}

?>
```

Line: 1 PHP Tab Size: 4 Symbols

A Stringable Value Object.

Should not be needed.

```
$unsafe = new unsafe_value('Something ' . $weird);
```

Easy for Auditor to find :-)

???

```
$unsafe = new unsafe_value('WHERE id ' . $comparison . ' ?');

$db->query($unsafe, [$id]);
```

What about Identifiers in SQL?

```php
<?php

class db {

    // ...

    function query($sql, $parameters = [], $identifiers = []) {

        $this->literal_check($sql);

        foreach ($identifiers as $name => $value) {
            if (!preg_match('/^[a-z0-9_]+$/', $name)) {
                throw new Exception('Invalid identifier name "' . $name . '"');
            } else if (!preg_match('/^[a-z0-9_]+$/', $value)) {
                throw new Exception('Invalid identifier value "' . $value . '"');
            } else {
                $sql = str_replace('{' . $name . '}', '`' . $value . '`', $sql);
            }
        }

        // Send $sql and $parameters to the database.

    }

```

Variable Identifiers can still be risky (see the ORDER BY example).

But this shows how special values (e.g. from INI/JSON/YAML files) can be safely used *after* checking the SQL has been written by the programmer.

```php
<?php

class db {

    // ...

    function query($sql, $parameters = [], $identifiers = []) {

        $this->literal_check($sql);

        foreach ($identifiers as $name => $value) {
            if (!preg_match('/^[a-z0-9_]+$/', $name)) {
                throw new Exception('Invalid identifier name "' . $name . '"');
            } else if (!preg_match('/^[a-z0-9_]+$/', $value)) {
                throw new Exception('Invalid identifier value "' . $value . '"');
            } else {
                $sql = str_replace('{' . $name . '}', '`' . $value . '`', $sql);
            }
        }

        // Send $sql and $parameters to the database.

    }
```

```php
<?php

class db {

    // ...

    function query($sql, $parameters = [], $identifiers = []) {

        $this->literal_check($sql);

        foreach ($identifiers as $name => $value) {
            if (!preg_match('/^[a-z0-9_]+$/', $name)) {
                throw new Exception('Invalid identifier name "' . $name . '"');
            } else if (!preg_match('/^[a-z0-9_]+$/', $value)) {
                throw new Exception('Invalid identifier value "' . $value . '"');
            } else {
                $sql = str_replace('{' . $name . '}', '`' . $value . '`', $sql);
            }
        }

        // Send $sql and $parameters to the database.

    }
```

Over to you :-)

```php
<?php

    $db = new db();

    $db->query('SELECT name FROM user WHERE id = ?', [$id]);

    $db->query('SELECT name FROM user WHERE id = ' . $id); // REJECTED

?>
```

Line: 1 PHP Tab Size: 4 Symbols

Using Identifiers

```php
<?php

    $db = new db();

    $db->query('SELECT name FROM {table}', [], ['table' => $table]);

    $db->query('SELECT name FROM ' . $table); // REJECTED

?>
```

Line: 1 PHP Tab Size: 4 Symbols

$articles->where('field', $value);

1.php — Downloads

```php
<?php

class orm {

    // ...

    function where($field, $value = NULL) {

        $this->literal_check($field);

        // ...

    }

}

?>
```

Line: 1 PHP Tab Size: 4 Symbols

Run the check

$users->order($order);

```php
<?php

class orm {

    // ...

    function order($sql) {

        $this->literal_check($sql);

        // ...

    }

}

?>
```

Line: 1 PHP Tab Size: 4 Symbols

Run the check

CLI

```php
<?php

function parameterised_exec($cmd, $parameters = []) {

  if (!is_literal($cmd)) {
    throw new Exception('The first argument must be a literal');
  }

  $offset = 0;
  $k = 0;
  while (($pos = strpos($cmd, '?', $offset)) !== false) {
    if (!isset($parameters[$k])) {
      throw new Exception('Missing parameter "' . ($k + 1) . '"');
      exit();
    }
    $par = escapeshellarg($parameters[$k]);
    $cmd = substr($cmd, 0, $pos) . $par . substr($cmd, ($pos + 1));
    $offset = ($pos + strlen($par));
    $k++;
  }
  if (isset($parameters[$k])) {
    throw new Exception('Unused parameter "' . ($k + 1) . '"');
    exit();
  }

  return exec($cmd);

}
```

Strict Check

```php
<?php

function parameterised_exec($cmd, $parameters = []) {

  if (!is_literal($cmd)) {
    throw new Exception('The first argument must be a literal');
  }

  $offset = 0;
  $k = 0;
  while (($pos = strpos($cmd, '?', $offset)) !== false) {
    if (!isset($parameters[$k])) {
      throw new Exception('Missing parameter "' . ($k + 1) . '"');
      exit();
    }
    $par = escapeshellarg($parameters[$k]);
    $cmd = substr($cmd, 0, $pos) . $par . substr($cmd, ($pos + 1));
    $offset = ($pos + strlen($par));
    $k++;
  }
  if (isset($parameters[$k])) {
    throw new Exception('Unused parameter "' . ($k + 1) . '"');
    exit();
  }

  return exec($cmd);

}
```

Escaping all values

Apply $parameters

```php
<?php

function parameterised_exec($cmd, $parameters = []) {

  if (!is_literal($cmd)) {
    throw new Exception('The first argument must be a literal');
  }

  $offset = 0;
  $k = 0;
  while (($pos = strpos($cmd, '?', $offset)) !== false) {
    if (!isset($parameters[$k])) {
      throw new Exception('Missing parameter "' . ($k + 1) . '"');
      exit();
    }
    $par = escapeshellarg($parameters[$k]);
    $cmd = substr($cmd, 0, $pos) . $par . substr($cmd, ($pos + 1));
    $offset = ($pos + strlen($par));
    $k++;
  }
  if (isset($parameters[$k])) {
    throw new Exception('Unused parameter "' . ($k + 1) . '"');
    exit();
  }

  return exec($cmd);

}
```

Run $cmd

```
parameterised_exec('grep ? /path/to/file', [$search]);
```

First argument is checked

```
parameterised_exec('grep "' . $search . '" /path/to/file');
```

First argument is checked

HTML

```
$template->render('<p>Hi {{ name }}</p>', ['name' => $name]);
```

First argument is checked

```
$template->render('<p>Hi ' . $name . '</p>');
```

First argument is checked

About 300 Lines :-)

Protection Level

Allowed tags, attributes, and attribute values

HTML Parsing... in XML mode :-)

Node walking, to find "?" for parameters

Return HTML with (checked) parameters

The `unsafe_value` object (should not be needed)

```
$html = ht('<p>Hi <span>?</span></p>', [$name]);
```

```
$template = ht('<p>Hi <span>?</span></p>');

$html_1 = $template->html([$name_1]);
$html_2 = $template->html([$name_2]);
$html_3 = $template->html([$name_3]);
```

```
$html = ht('<p>Hi ' . $name . '</p>');
```

```
$html = ht('<p>Hi ' . $name . '</p>');
```

```
$url = 'https://example.com';

$html = ht('<a href="?">Link</p>', [$url]);
```

```
$url = 'javascript:alert()';

$html = ht('<a href="?">Link</p>', [$url]);
```

```
$template = ht('
    <a href="?">
        <figure>
            <img src="?" width="?" height="?" alt="?" />
            <figcaption>?</figcaption>
        </figure>
    </a>');

$html = $template->html([
      '/example/',
      '/img/example.jpg',
      100,
      200,
      'My Image Alt',
      'My Image Caption',
 ]);
```

And in ~10 years time...

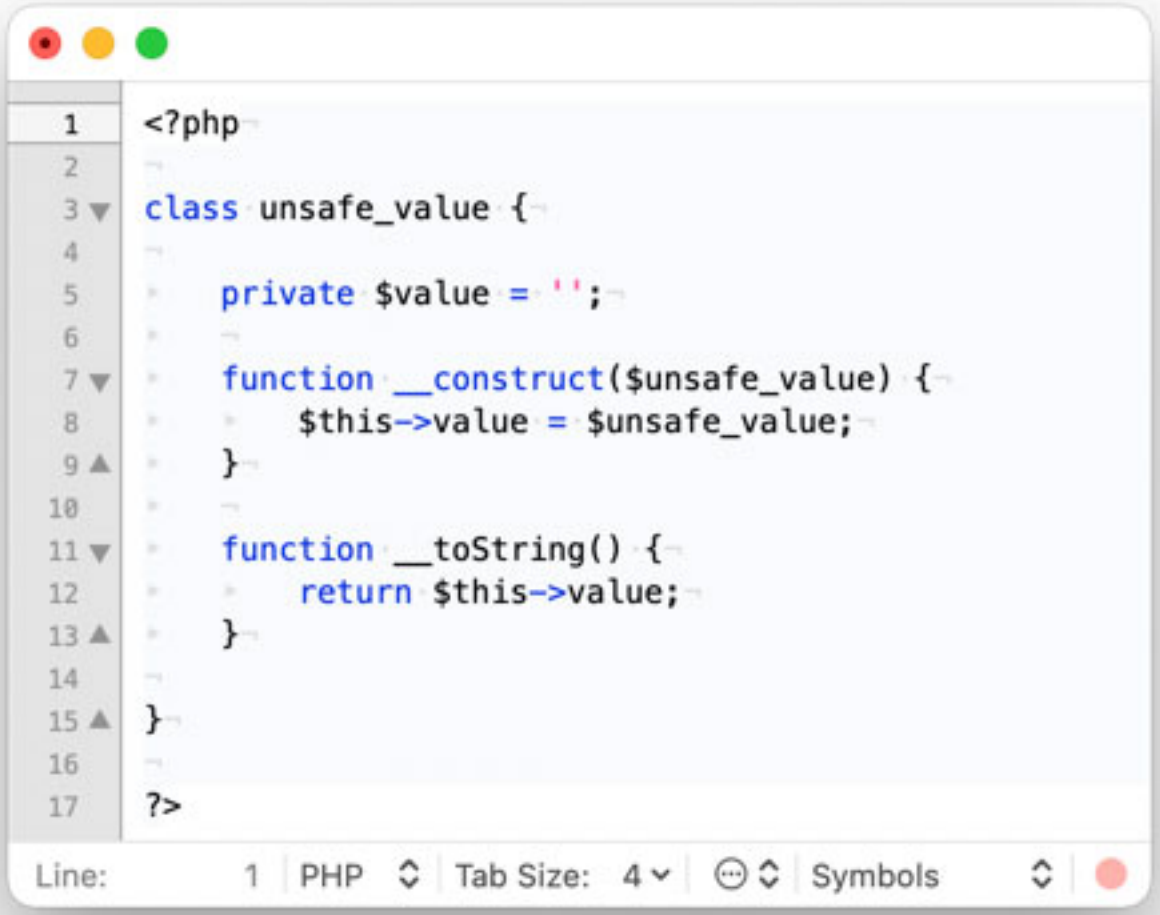
Native functions,
like mysqli_query(), and mysqli_prepare(),
can use this check.

They would accept everything,

but warn if not given a "string from a trusted developer".

And there would need to be a way for special cases to be trusted...

e.g. strings created by a library.



```php
<?php

class unsafe_value {

    private $value = '';

    function __construct($unsafe_value) {
        $this->value = $unsafe_value;
    }

    function __toString() {
        return $this->value;
    }

}

?>
```

Line: 1 PHP Tab Size: 4 Symbols

Maybe a "stringable value-object"?

Some way to trust the stringable value object

What it can be trusted for.

```php
<?php

class db_trusted_sql {

    private $sql = '';

    function __construct($sql) {
        $this->sql = $sql;
    }

    function __toString() {
        return $this->sql;
    }

}

?>
```

Line: 1 PHP Tab Size: 4 Symbols

```php
class db {

    //...

    function __construct() {

        // ...

        spl_trust_object('db_trusted_sql', 'sql');

    }

    function query($sql, $parameters = []) {

        // ...

        $trusted_sql = new db_trusted_sql($sql);

        mysqli_query($this->link, $trusted_sql);

    }
```

Accepted by the native function

There would be a way to disable the check.

At least the developer is then aware their code is unsafe.

There would be a way to enforce the check.

For developers, confident in their system, to enforce this protection.

"Distinguishing strings from a trusted developer, from strings that may be attacker controlled"

Mike Samuel - 27th March 2019

OWASP 2021

>irtual

APPSEC

THANK YOU!