



OWASP 2021  
**>virtual**  
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**

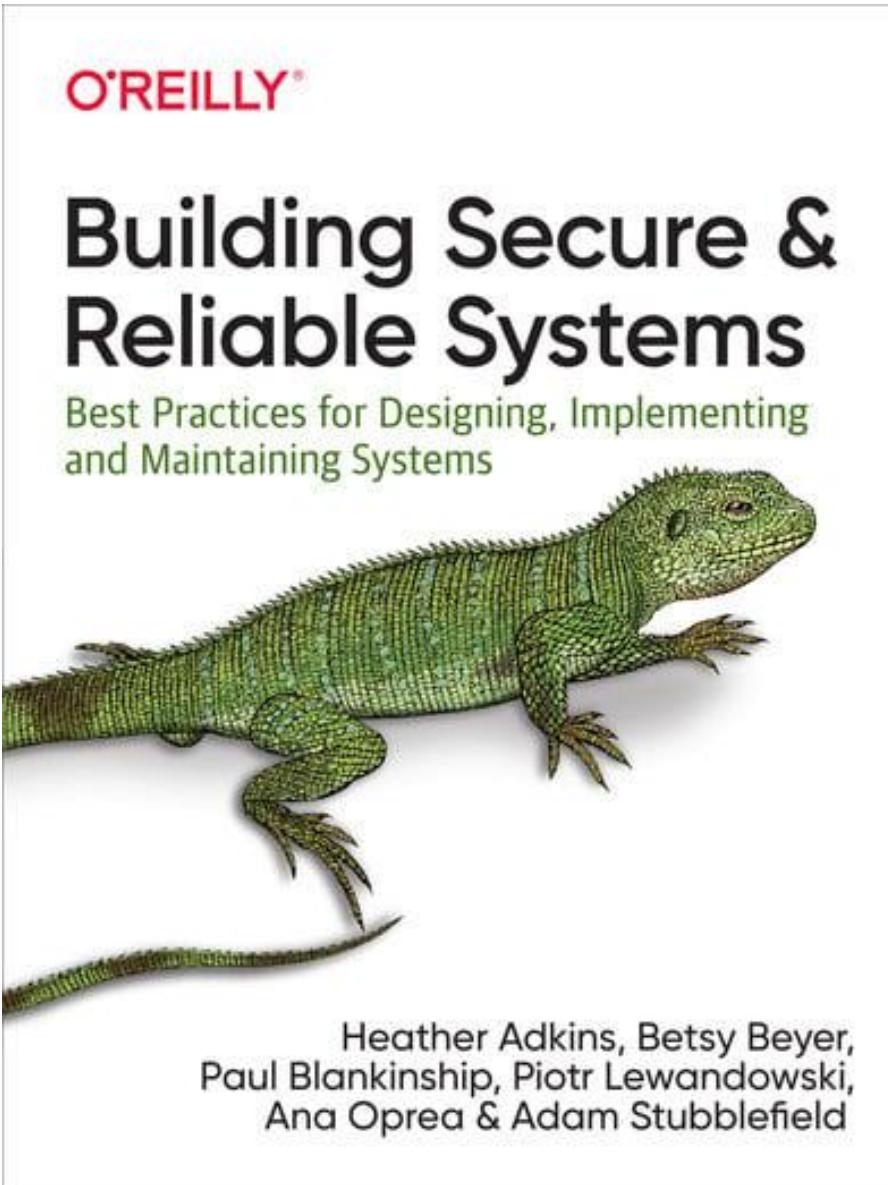
# Christoph Kern

## 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'
```





A screenshot of a web browser window displaying a user list table. The URL in the address bar is `https://example.com/?order=name_first,name_last`. The table has two columns: Name and Email. The data is:

Name	Email
Amy Anderson	user1@example.com
Benjamin Binder	user2@example.com
Charlotte Clark	user3@example.com
Donna Davis	user4@example.com

A large pink arrow points from the URL in the browser to the database query code at the bottom of the slide.

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

... ORDER BY **name\_first, name\_last**



A screenshot of a web browser displaying a table of users. The table has two columns: 'Name' and 'Email'. The data is as follows:

Name	Email
Amy Anderson	user1@example.com
Benjamin Binder	user2@example.com
Charlotte Clark	user3@example.com
Donna Davis	user4@example.com

The browser address bar shows the URL: `https://example.com/?order=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
```





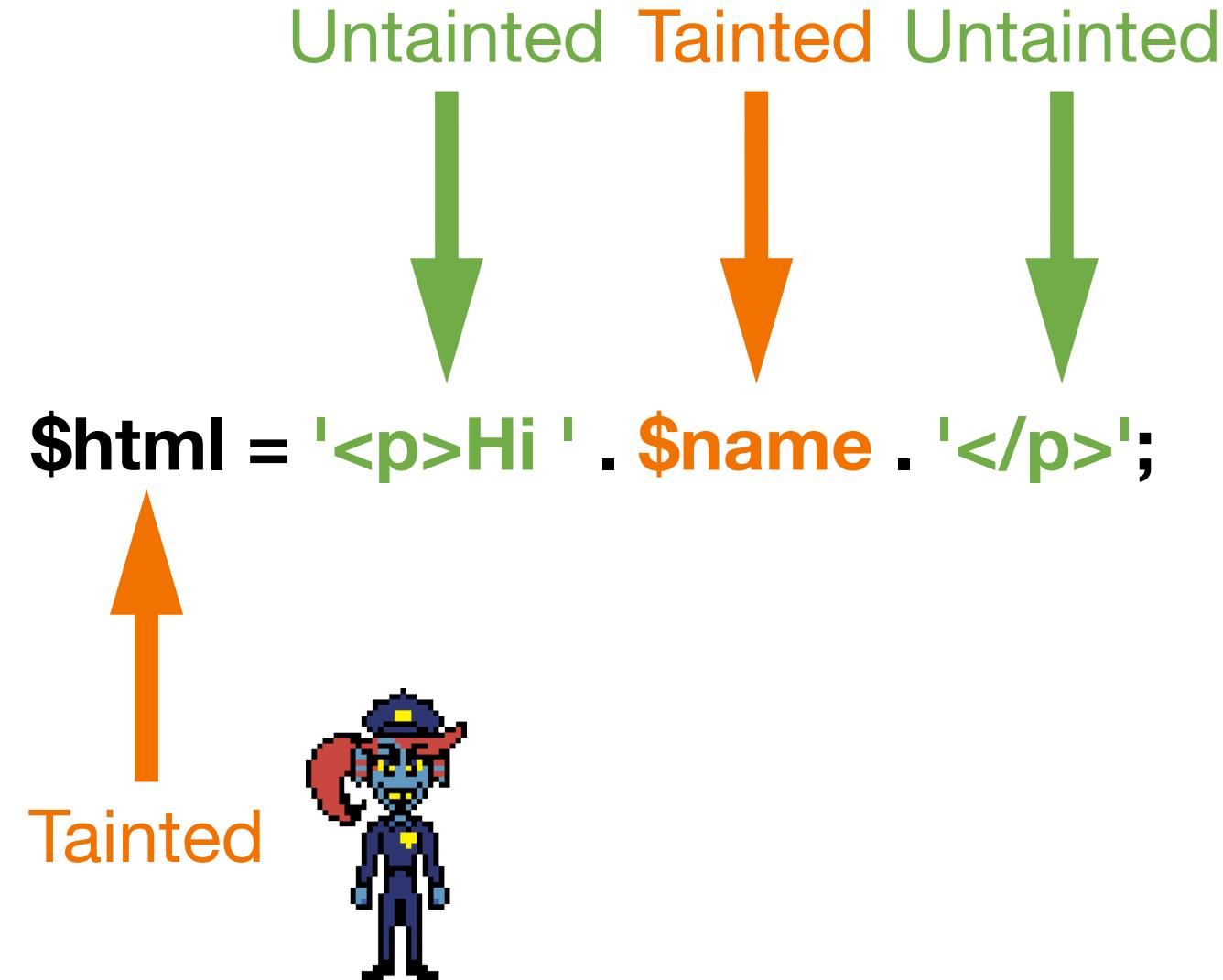
# How do we stop mistakes?

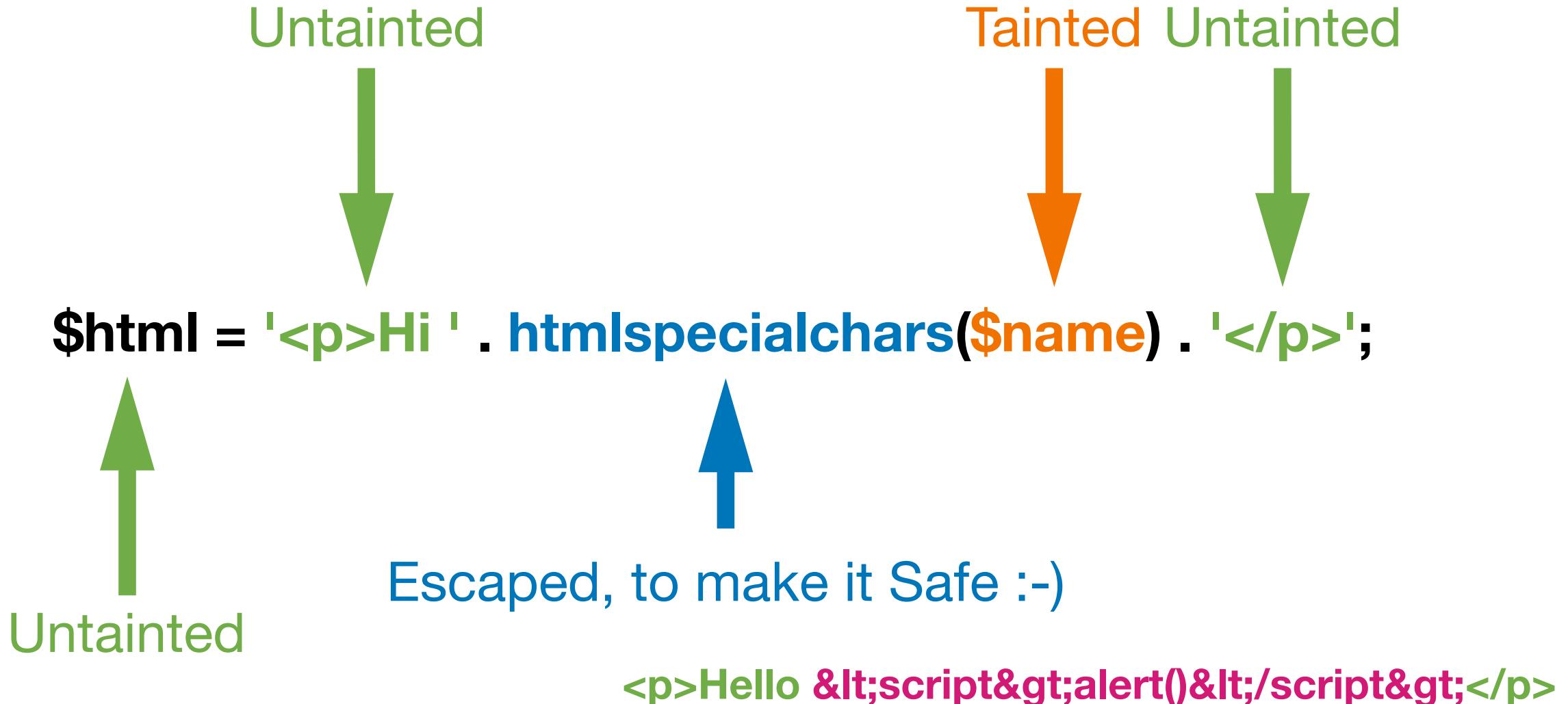


# Maybe Taint Checking?

Where variables note if they are **Tainted**, or **Untainted**.

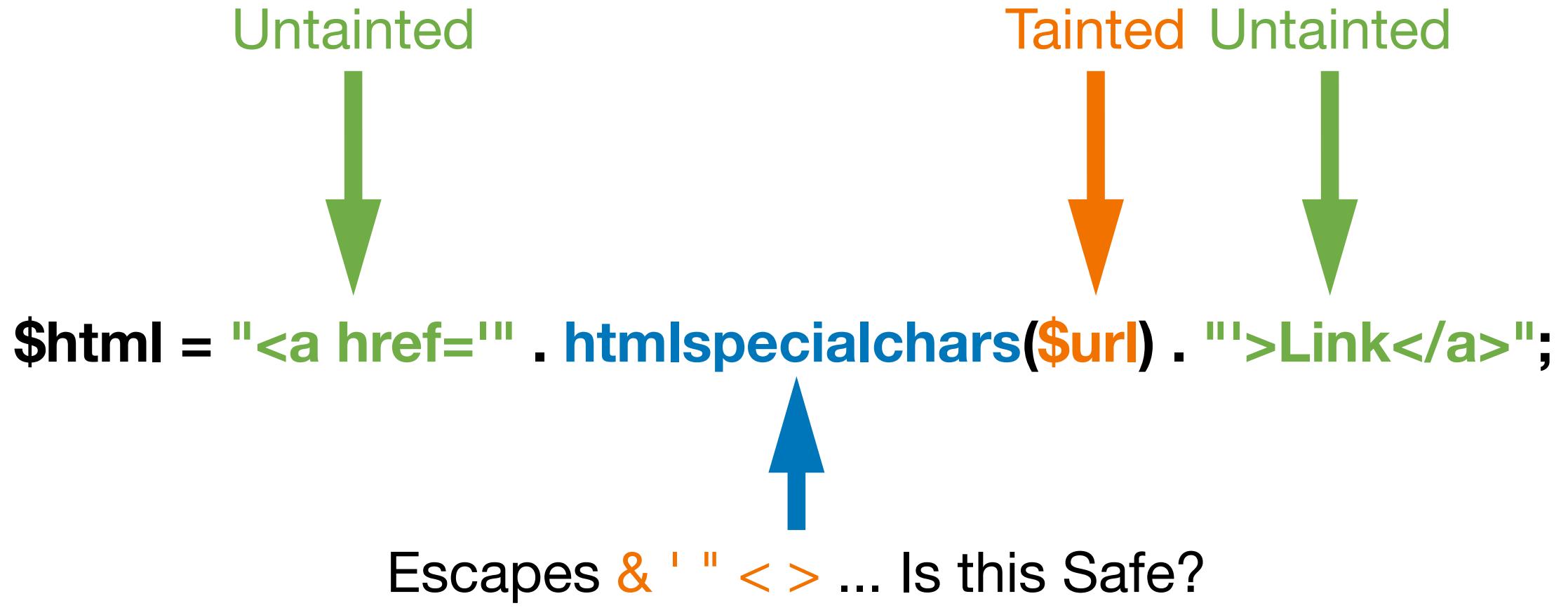






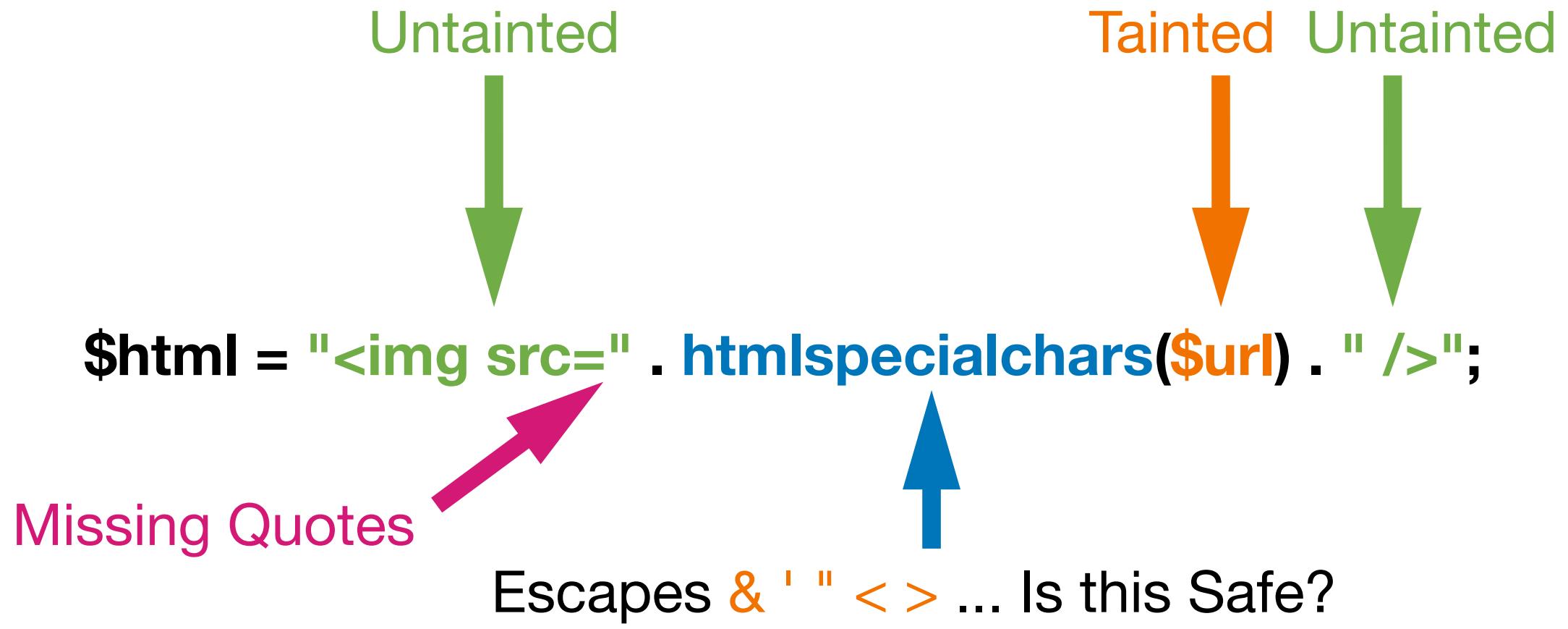


**Unfortunately Taint Checking incorrectly assumes  
escaping makes a value “safe” for *any* context.**



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





<img src=/ onerror=alert() />





Untainted



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

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



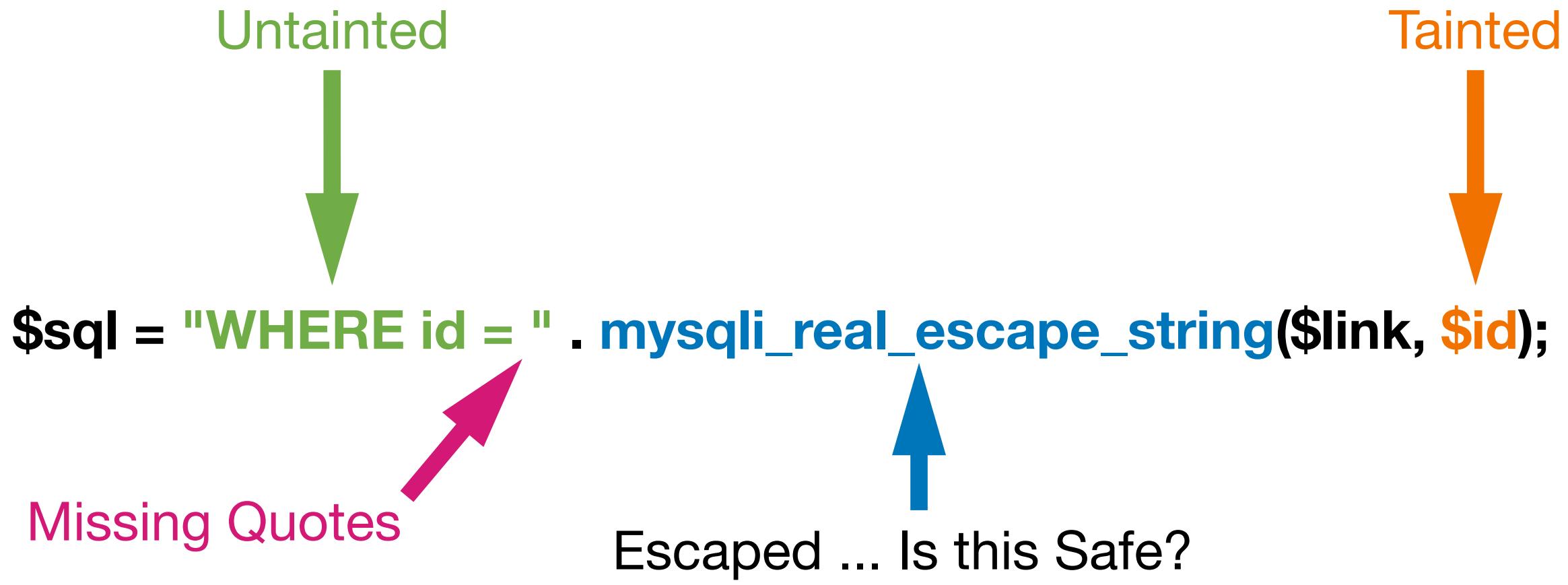
Is this Safe?

Tainted Untainted



```
<img src='/' onerror='alert()' />
```





WHERE id = -1 UNION SELECT \* FROM admin





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



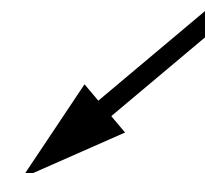
Instead, we can simplify it,  
by looking for "strings from a trusted developer"...

**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);**



???



Safe\*



Unsafe



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

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



Unsafe



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

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





Safe\*



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

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



Unsafe



Safe\*



Unsafe



Safe\*



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

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



???      Safe\*      Unsafe      Safe\*



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

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



Unsafe



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

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





Safe\*



**\$command = 'grep ? /path/to/file';**

**shell\_exec(\$command, [\$search]);**



Unsafe





Safe\*



Unsafe



Safe\*



```
$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.

```
$path = "/";  
rm -rf /
```



Safe\*



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

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



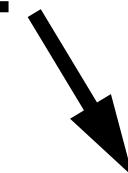
Unsafe





# How to deal with Special Cases

Did you remember to  
ensure all were integers?



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

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



```
'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);
```

```
'WHERE id IN (?, ?, ?)'
```





```
$sql = 'WHERE id IN (' . in_parameters(count($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?

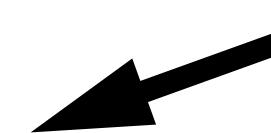


```
$sql = 'ORDER BY ' . $order_field;
```



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

List of Allowed fields



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

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



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



The screenshot shows a Go code editor with a file named `example_package.go`. The code defines a package named `example_package` and contains a type alias `stringConstant` which is an alias for the `string` type. It also contains a function named `OnlyAcceptsStringConstant` which takes a `stringConstant` value and returns a `bool`, specifically returning `true`.

Annotations with arrows point to specific parts of the code:

- An arrow points from the text box "**"stringConstant" type; not exported**" to the `stringConstant` type definition.
- An arrow points from the text box "**"OnlyAcceptsStringConstant" function; is exported**" to the `OnlyAcceptsStringConstant` function definition.
- An arrow points from the text box "**"stringConstant" type; required for input**" to the parameter of the `OnlyAcceptsStringConstant` function.

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

Line: 1 Go



```
example_code.go
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7
8     "example.com/example_package"
9 )
10
11 func main() {
12
13     reader := bufio.NewReader(os.Stdin)
14     fmt.Print("Your Name: ")
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 }
```

```
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 | Tab Size: 4 | Symbols

Get name (untrusted data)

An "Untyped String"

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



The image shows a Go development environment with two code editors and a terminal window.

**Left Editor:** example\_code.go

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

**Right Editor:** example\_package.go

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

**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$
```

A callout box points from the error message in the terminal to the text "Cannot be converted to a stringConstant".

Cannot be converted to a  
stringConstant



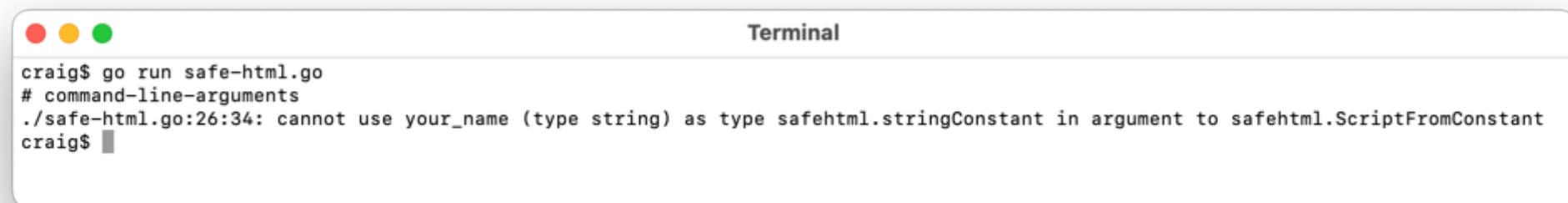
## How this is used by "go-safe-html"



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

a := safehtml.ScriptFromConstant("var my\_script = 'example';")

b := safehtml.ScriptFromConstant(your\_name)



A screenshot of a macOS terminal window titled "Terminal". The window has three red, yellow, and green circular close buttons at the top left. The terminal output shows:

```
craig$ go run safe-html.go
# command-line-arguments
./safe-html.go:26:34: cannot use your_name (type string) as type safehtml.stringConstant in argument to safehtml.ScriptFromConstant
craig$
```



```
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)
```



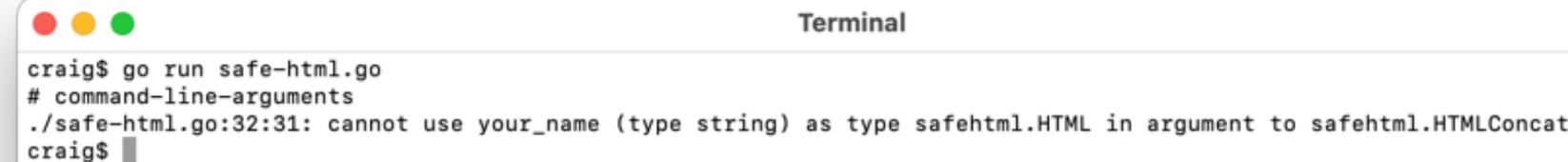
```
craig$ go run safe-html.go  
# command-line-arguments  
.safe-html.go:29:41: cannot use your_name (type string) as type "github.com/google/safehtml/template".stringConstant in  
argument to "github.com/google/safehtml/template".MustParseAndExecuteToHTML  
craig$
```



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

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

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



```
craig$ go run safe-html.go
# command-line-arguments
./safe-html.go:32:31: cannot use your_name (type string) as type safehtml.HTML in argument to safehtml.HTMLConcat
craig$
```





# Ending Injection Vulnerabilities

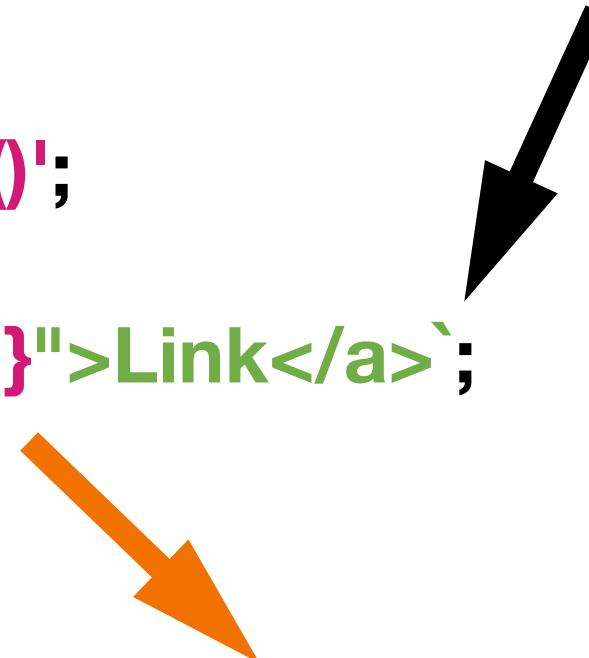
## With Node... and JavaScript (soon)



## Template Literal

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

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

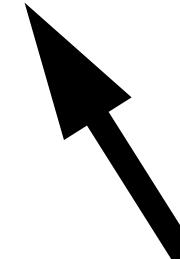


```
'<a href="javascript:alert()">Link</a>'
```



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

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

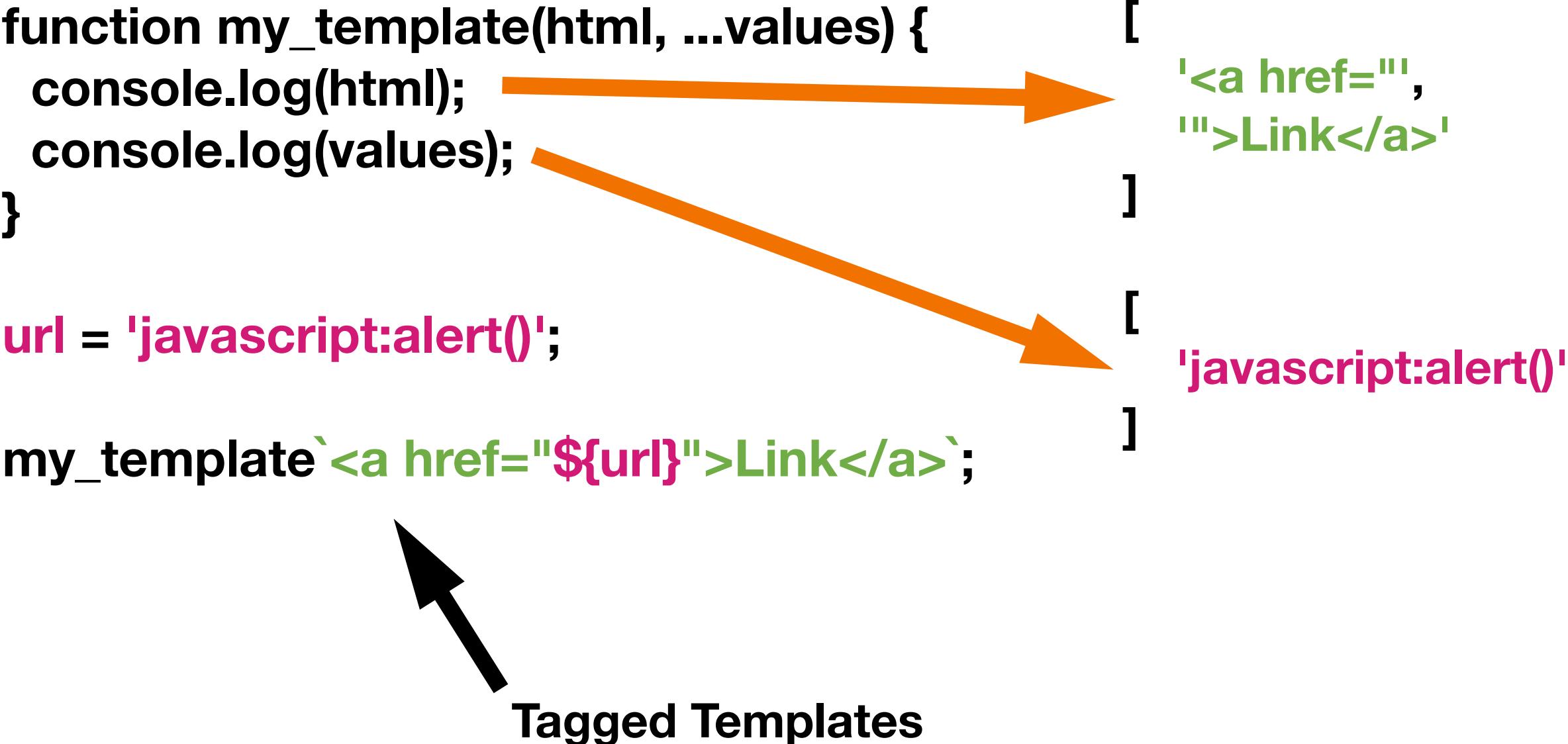


Tagged Templates

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

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

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



Tagged Templates

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

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

```
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!'; }
```

...

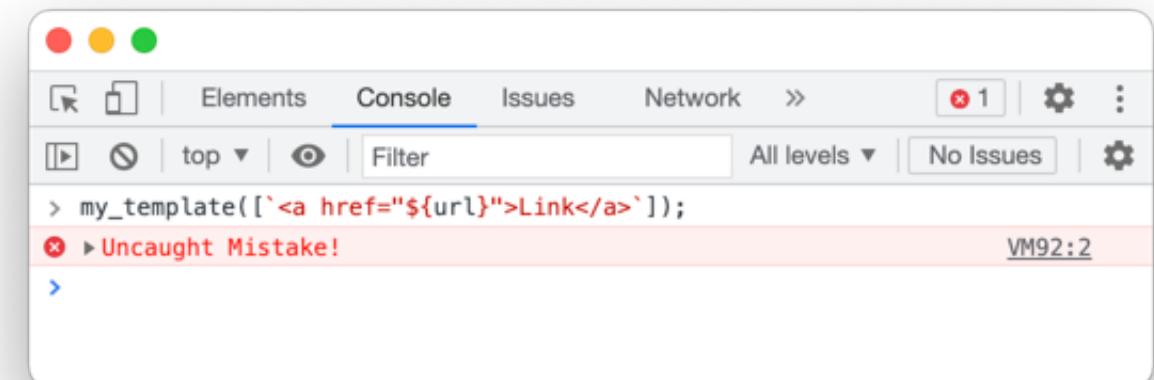
}

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
```



The screenshot shows a code editor window titled "index.js". The code is as follows:

```
1  isTemplateObject = require('is-template-object');
2
3
4  function template(html, ...values) {
5    console.log(isTemplateObject(html), html, values);
6  }
7
8 //-----
9
10 var username = "<script>alert()</script>"; // From evil user
11
12 template`<p>Hi ${username}<p>`;
13
14 template(['<p>Hi ${username}<p>']); // Wrong
15
16 template(['<p>Hi ', username, '<p>']); // Wrong
```

A callout box with a black arrow points from the text "Call isTemplateObject" to the line "console.log(isTemplateObject(html), html, values);". Another callout box with a black arrow points from the text "Require Polyfill, from Mike Samuel" to the line "isTemplateObject = require('is-template-object');

**Call  
isTemplateObject**

**Require Polyfill,  
from Mike Samuel**



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

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

Get name  
(untrusted data)

Use  
`goog.string.Const.from()`



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

Terminal
```

```
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-v20211006.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("<>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**

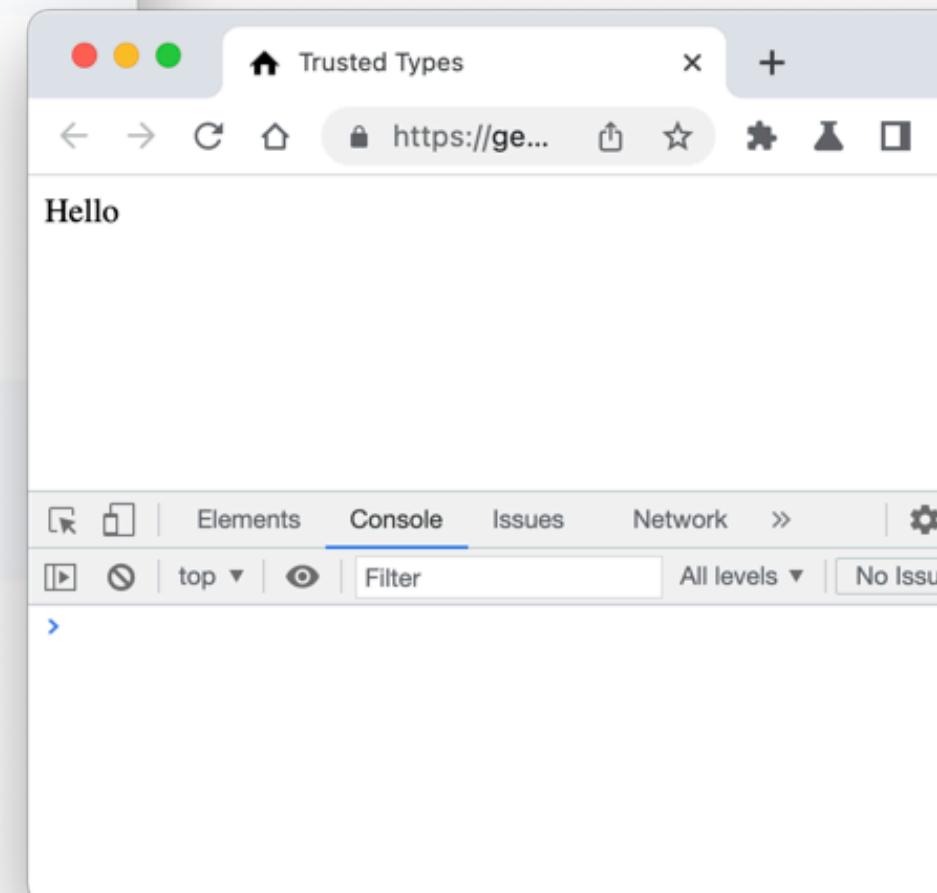
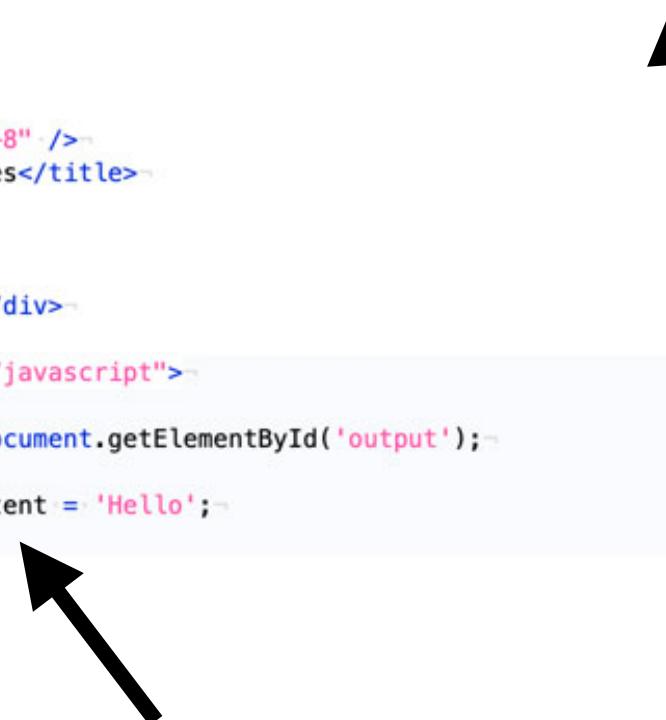


**or JavaScript and Trusted Types... Hopefully Soon™**

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

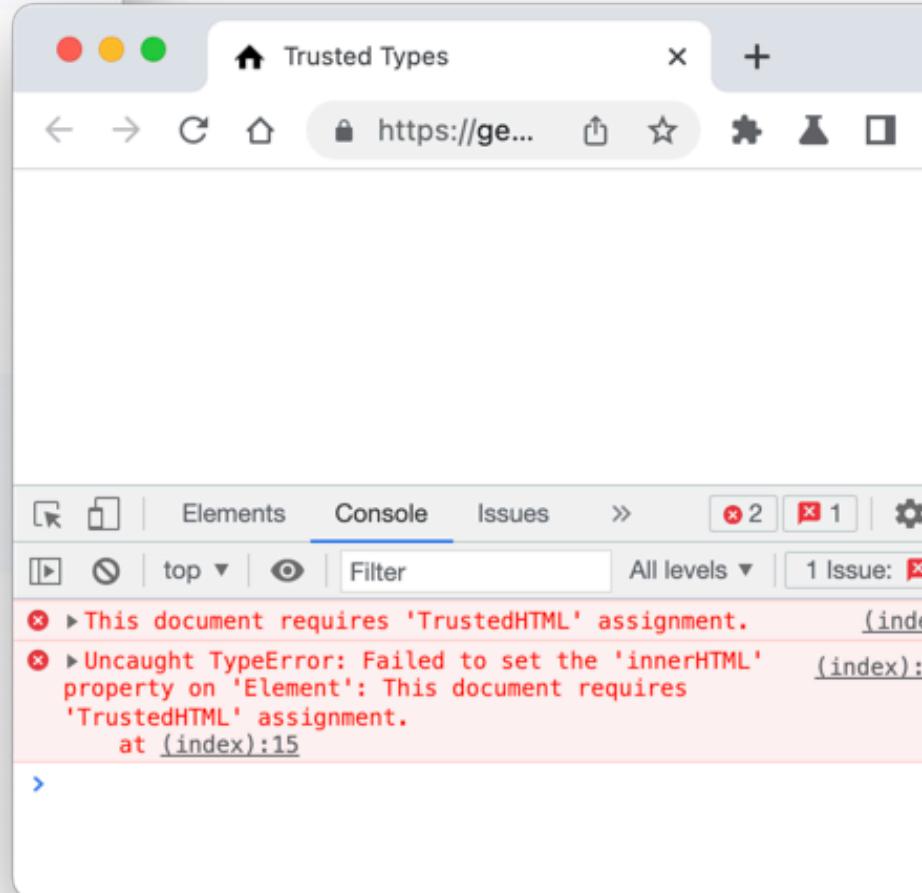
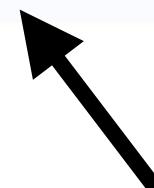


```
1 <?php
2
3     header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types 'none';");
4
5 ?>
6 <!DOCTYPE html>
7 <html lang="en-GB">
8 <head>
9     <meta charset="UTF-8" />
10    <title>Trusted Types</title>
11 </head>
12 <body>
13
14     <div id="output"></div>
15
16     <script type="text/javascript">
17
18         var output = document.getElementById('output');
19
20         output.textContent = 'Hello';
21
22     </script>
23
24 </body>
25 </html>
```





```
1 <?php
2
3     header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types 'none';");
4
5 ?>
6 <!DOCTYPE html>
7 <html lang="en-GB">
8 <head>
9     <meta charset="UTF-8" />
10    <title>Trusted Types</title>
11 </head>
12 <body>
13
14     <div id="output"></div>
15
16     <script type="text/javascript">
17
18         var output = document.getElementById('output');
19
20         output.innerHTML = 'Hello';
21
22     </script>
23
24 </body>
25 </html>
```



The browser developer tools console displays the following errors:

- `✖ This document requires 'TrustedHTML' assignment.` ([index:15](#))
- `✖ Uncaught TypeError: Failed to set the 'innerHTML' property on 'Element': This document requires 'TrustedHTML' assignment.` `at (index):15`



```
1 <?php
2
3     header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");
4
5 ?>
6
7 <!DOCTYPE html>
8 <html lang="en-GB">
9 <head>
10    <meta charset="UTF-8" />
11    <title>Trusted Types</title>
12 </head>
13 <body>
14
15    <div id="output"></div>
16
17    <script type="text/javascript">
18
19        var my_trusted_type = {
20            'createHTML': function(s) {
21                return s; // Dangerous
22            }
23        };
24
25        if (windowtrustedTypes) {
26            my_trusted_type = windowtrustedTypes.createPolicy('my_trusted_type', my_trusted_type);
27        }
28
29        var output = document.getElementById('output');
30
31        output.innerHTML = my_trusted_type.createHTML('Hello');
32
33    </script>
```

Our Trusted Type.

Use DOMPurify,  
"Sanitizer" API, etc.

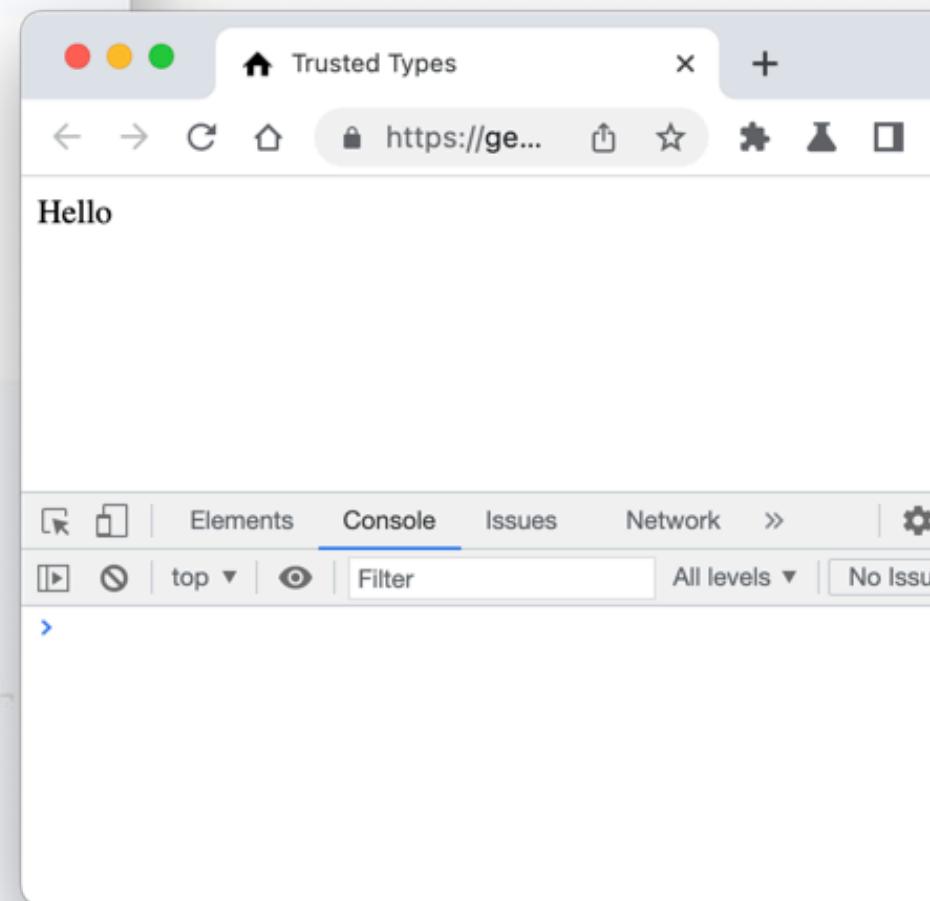


```
1 <?php
2
3     header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");
4
5 ?>
6 <!DOCTYPE html>
7 <html lang="en-GB">
8 <head>
9     <meta charset="UTF-8" />
10    <title>Trusted Types</title>
11 </head>
12 <body>
13
14 <div id="output"></div>
15
16 <script type="text/javascript">
17
18     var my_trusted_type = {
19         'createHTML': function(s) {
20             return s; // Dangerous
21         }
22     };
23
24     if (windowtrustedTypes) {
25         my_trusted_type = windowtrustedTypes.createPolicy('my_trusted_type', my_trusted_type);
26     }
27
28     var output = document.getElementById('output');
29
30     output.innerHTML = my_trusted_type.createHTML('Hello');
31
32 </script>
```

Tell browser to trust



```
1 <?php
2
3     header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");
4
5 ?>
6 <!DOCTYPE html>
7 <html lang="en-GB">
8 <head>
9     <meta charset="UTF-8" />
10    <title>Trusted Types</title>
11 </head>
12 <body>
13
14     <div id="output"></div>
15
16     <script type="text/javascript">
17
18         var my_trusted_type = {
19             'createHTML': function (s) {
20                 return s; // Dangerous
21             }
22         };
23
24         if (windowtrustedTypes) {
25             my_trusted_type = windowtrustedTypes.createPolicy('my_trusted_type', my_trusted_type);
26         }
27
28         var output = document.getElementById('output');
29
30         output.innerHTML = my_trusted_type.createHTML('Hello');
31
32     </script>
```



Use it



```
1 <?php
2
3 header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");
4
5 ?>
6 <!DOCTYPE html>
7 <html lang="en-GB">
8 <head>
9   <meta charset="UTF-8" />
10  <title>Trusted Types</title>
11 </head>
12 <body>
13
14 <div id="output"></div>
15
16 <script type="text/javascript">
17
18   var output = document.getElementById('output');
19
20   output.innerHTML = window.trustedHTML.fromLiteral`Hello`;
21
22 </script>
23
24 </body>
25 </html>
```



Hopefully one day :-)



# 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



pom.xml vs. pom.xml

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.1</version>
    <scope>test</scope>
</dependency>
</dependencies>

<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <artifactId>maven-clean-plugin</artifactId>
                <version>3.1.0</version>
            </plugin>
            <plugin>
                <artifactId>maven-resources-plugin</artifactId>
                <version>3.0.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.0</version>
                <configuration>
                    <source>8</source>
                    <target>8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.1</version>
            </plugin>
            <plugin>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.0.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-install-plugin</artifactId>
                <version>2.5.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-deploy-plugin</artifactId>
                <version>2.8.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-site-plugin</artifactId>
                <version>3.7.1</version>
            </plugin>
            <plugin>
```

1

```
</dependency>
<dependency>
    <groupId>com.google.errorprone</groupId>
    <artifactId>error_prone_annotations</artifactId>
    <version>2.0.8</version>
</dependency>
</dependencies>

<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <artifactId>maven-clean-plugin</artifactId>
                <version>3.1.0</version>
            </plugin>
            <plugin>
                <artifactId>maven-resources-plugin</artifactId>
                <version>3.0.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.0</version>
                <configuration>
                    <source>8</source>
                    <target>8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
                <fork>true</fork>
                <compilerArgs>
                    <arg>-XDcompilePolicy=simple</arg>
                    <arg>-Xplugin:ErrorProne</arg>
                    <arg>-J--add-exports=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-exports=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-exports=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-exports=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-exports=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-exports=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-exports=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-exports=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-opens=jdk.compiler/com.sun.tools.javac</arg>
                    <arg>-J--add-opens=jdk.compiler/com.sun.tools.javac</arg>
                </compilerArgs>
                <annotationProcessorPaths>
                    <path>
                        <groupId>com.google.errorprone</groupId>
                        <artifactId>error_prone_core</artifactId>
                        <version>2.9.0</version>
                    </path>
                </annotationProcessorPaths>
            </configuration>
```

2

```
</configuration>
```



**There are a few  
dependencies**



The diagram illustrates the flow of code execution in a Java file named index.java. It starts with the import statement, followed by the declaration of a class named index. Inside the class, there is a static void main method that prints "Your Name:", creates a Scanner object from System.in, reads a string from the user, and then closes the scanner. It then calls two functions: print\_constant("Compile Time Constant") and print\_constant(your\_name). The print\_constant function is annotated with @CompileTimeConstant, which is highlighted in red. The code also uses System.out.println to output the value of the variable value.

```
index.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 +  
1 import java.util.Scanner;  
2 import com.google.errorprone.annotations.CompileTimeConstant;  
3  
4 public class index {  
5  
6     private static void print_constant(@CompileTimeConstant String value) {  
7         System.out.println(value);  
8     }  
9 }
```

### 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] -----
```

```
17  
18     print_constant("Compile Time Constant");  
19  
20     print_constant(your_name);  
21  
22 }  
23  
24 }  
25
```





# 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://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**



```
[craig] composer require --dev vimeo/palm
Using version ^4.12 for vimeo/palm
./composer.json has been created
Running composer update vimeo/palm
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Locking amphp/amp [v2.6.1]
  - Locking amphp/byte-stream [v1.8.1]
  - Locking composer/package-versions-deprecated [1.11.99.4]
  - Locking composer/composer [v2.3.0]
  - Locking composer/xdebug-handler [v2.8.2]
  - Locking dnogei/php-xdg-base-dir [v0.1.1]
  - Locking felixb Becker/advanced-json-rpc [v3.2.1]
  - Locking felixb Becker/language-server-protocol [1.5.1]
  - Locking felixb Becker/jasmine [v1.5.0]
  - Locking nikic/php-parser [v4.13.1]
  - Locking openl3/lib-arrayxml [1.0.0]
  - Locking phpdocum ent/reflection-common [2.2.0]
  - Locking phpdocum ent/reflection-docblock [5.3.0]
  - Locking phpdocum ent/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/crossref [v5.15.0]
  - 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-php72 [v1.23.0]
  - Locking symfony/polyfill-php80 [v1.23.1]
  - Locking symfony/service-contract [v2.4.0]
  - Locking symfony/string [v5.3.18]
  - Locking vimeo/palm [4.12.0]
  - Locking webmozart/assert [v1.18.0]
  - 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 vimeo/palm [v4.12.0]
  - Downloading phpdocum ent/reflection-docblock [5.3.0]
  - Downloading psr/container [1.1.2]
  - Downloading symfony/string [v5.3.18]
  - Downloading doctrine/consolidation [v2.0.18]
  - Downloading nikic/php-parser [v4.13.1]
  - Downloading composer/sever [3.2.6]
  - Downloading vimeo/palm [v4.12.0]
  - Installing composer/package-versions-deprecated [1.11.99.4]: Extracting archive
  - Installing dnogei/php-xdg-base-dir [v0.1.1]: Extracting archive
  - Installing sebastian/assert [1.18.0]: Extracting archive
  - Installing phpdocum ent/reflection-common [2.2.0]: Extracting archive
  - Installing phpdocum ent/type-resolver [1.5.1]: Extracting archive
  - Installing phpdocum ent/reflection-docblock [5.3.0]: Extracting archive
  - Installing symfony/deprecation-contracts [v2.4.0]: Extracting archive
  - Installing symfony/polyfill-ctype [v1.23.0]: Extracting archive
  - Installing psr/container [1.1.2]: Extracting archive
  - Installing symfony/service-contract [v2.4.0]: 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 [v5.3.18]: Extracting archive
  - Installing webmozart/assert [v1.18.0]: Extracting archive
  - Installing webmozart/path-util [2.3.0]: Extracting archive
  - Installing sebastian/diff [4.0.4]: Extracting archive
  - Installing openl3/lib-arrayxml [1.0.0]: Extracting archive
  - Installing nikic/php-parser [v4.13.1]: Extracting archive
  - Installing felixb Becker/language-server-protocol [1.5.1]: Extracting archive
  - Installing felixb Becker/advanced-json-rpc [v3.2.1]: Extracting archive
  - Installing dnogei/php-xdg-base-dir [v0.1.1]: Extracting archive
  - Installing amphp/amp [v2.6.1]: Extracting archive
  - Installing amphp/byte-stream [v1.8.1]: Extracting archive
  - Installing vimeo/palm [4.12.0]: Extracting archive
  - Installing vimeo/palm [4.12.0]: Extracting archive
  - Installation successful, but composer suggest' to see details.
Package suggestion was added by new dependencies, use 'composer suggest' instead.
[craig] composer install 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
10 packages you are using are looking for funding.
Use the "composer fund" command to find out more!
```



**./vendor/bin/psalm --init**



**Check Psalm is at level 3 or stricter.  
(level 1 is the most strict)**



```
1 <?php
2
3 $id = (string)($_GET['id'] ?? '');
4
5 class db {
6
7     /**
8      * @psalm-param literal-string $sql
9      */
10
11    public function query(string $sql, array $parameters = []): void {
12
13        // Send $sql and $parameters to the database.
14
15    }
16
17 }
18
19 $db = new db();
20
21 $db->query('SELECT * FROM user WHERE id = ?', [$id]);
22
23 $db->query('SELECT * FROM user WHERE id = ' . $id);
```

Use 'literal-string'  
type for \$sql



```
1 <?php
2
3 $id = (string) ($_GET['id'] ?? '');
4

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$ 
```

A large black arrow points downwards to the line of code '\$db->query('SELECT \* FROM user WHERE id = ' . \$id);' in the terminal output.





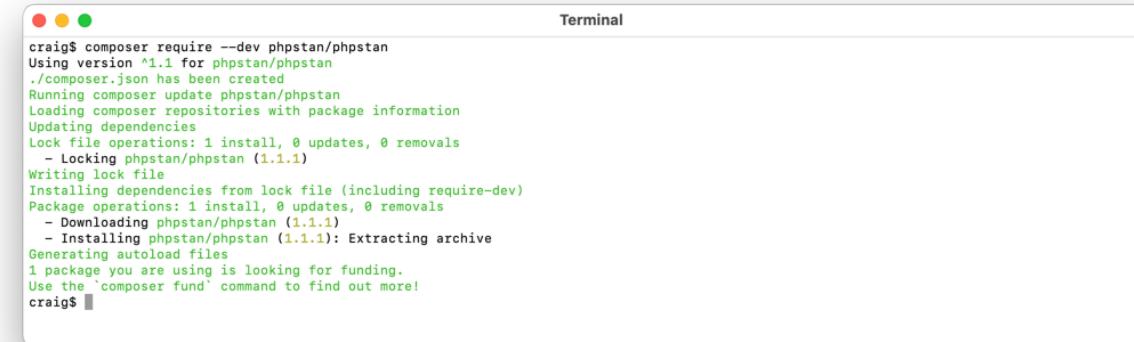
# Using PHPStan

## Static Analysis 2

Thanks to Ondřej Mirtes



**composer require --dev phpstan/phpstan**



A screenshot of a Mac OS X terminal window titled "Terminal". The window has three colored window control buttons (red, yellow, green) at the top left. The terminal itself is white with black text. It displays the following command and its execution:

```
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)**



```
1 <?php
2
3 $id = (string)($_GET['id'] ?? '');
4
5 ▼ class db {
6
7     /**
8      * @phpstan-param literal-string $sql
9      * @phpstan-param array<int, string> $parameters
10     */
11
12    public function query(string $sql, array $parameters = []): void {
13        // Send $sql and $parameters to the database.
14
15    }
16 ▲
17
18 ▲ }
19
20 $db = new db();
21
22 $db->query('SELECT * FROM user WHERE id = ?', [$id]);
23
24 $db->query('SELECT * FROM user WHERE id = ?', $id);
```

Use 'literal-string'  
type for \$sql



```
1 <?php
2
3 $id = (string) ($_GET['id'] ?? '');
4
5 class db {
6
7     public function query($sql, $params = [])
8     {
9         // ...
10    }
11
12    public function prepare($sql, $params = [])
13    {
14        // ...
15    }
16
17    public function execute($sql, $params = [])
18    {
19        // ...
20    }
21
22    public function fetchAll($sql, $params = [])
23    {
24        // ...
25    }
26
27    public function close()
28    {
29        // ...
30    }
31
32    public function __destruct()
33    {
34        // ...
35    }
36
37    public function __construct()
38    {
39        // ...
40    }
41
42    public function __call($method, $args)
43    {
44        // ...
45    }
46
47    public function __get($name)
48    {
49        // ...
50    }
51
52    public function __set($name, $value)
53    {
54        // ...
55    }
56
57    public function __toString()
58    {
59        // ...
60    }
61
62    public function __wakeup()
63    {
64        // ...
65    }
66
67    public function __sleep()
68    {
69        // ...
70    }
71
72    public function __destruct()
73    {
74        // ...
75    }
76
77    public function __call($method, $args)
78    {
79        // ...
80    }
81
82    public function __get($name)
83    {
84        // ...
85    }
86
87    public function __set($name, $value)
88    {
89        // ...
90    }
91
92    public function __toString()
93    {
94        // ...
95    }
96
97    public function __wakeup()
98    {
99        // ...
100   }
101
102  public function __destruct()
103  {
104      // ...
105  }
106
107  public function __call($method, $args)
108  {
109      // ...
110  }
111
112  public function __get($name)
113  {
114      // ...
115  }
116
117  public function __set($name, $value)
118  {
119      // ...
120  }
121
122  public function __toString()
123  {
124      // ...
125  }
126
127  public function __wakeup()
128  {
129      // ...
130  }
131
132  public function __destruct()
133  {
134      // ...
135  }
136
137  public function __call($method, $args)
138  {
139      // ...
140  }
141
142  public function __get($name)
143  {
144      // ...
145  }
146
147  public function __set($name, $value)
148  {
149      // ...
150  }
151
152  public function __toString()
153  {
154      // ...
155  }
156
157  public function __wakeup()
158  {
159      // ...
160  }
161
162  public function __destruct()
163  {
164      // ...
165  }
166
167  public function __call($method, $args)
168  {
169      // ...
170  }
171
172  public function __get($name)
173  {
174      // ...
175  }
176
177  public function __set($name, $value)
178  {
179      // ...
180  }
181
182  public function __toString()
183  {
184      // ...
185  }
186
187  public function __wakeup()
188  {
189      // ...
190  }
191
192  public function __destruct()
193  {
194      // ...
195  }
196
197  public function __call($method, $args)
198  {
199      // ...
200  }
201
202  public function __get($name)
203  {
204      // ...
205  }
206
207  public function __set($name, $value)
208  {
209      // ...
210  }
211
212  public function __toString()
213  {
214      // ...
215  }
216
217  public function __wakeup()
218  {
219      // ...
220  }
221
222  public function __destruct()
223  {
224      // ...
225  }
226
227  public function __call($method, $args)
228  {
229      // ...
230  }
231
232  public function __get($name)
233  {
234      // ...
235  }
236
237  public function __set($name, $value)
238  {
239      // ...
240  }
241
242  public function __toString()
243  {
244      // ...
245  }
246
247  public function __wakeup()
248  {
249      // ...
250  }
251
252  public function __destruct()
253  {
254      // ...
255  }
256
257  public function __call($method, $args)
258  {
259      // ...
260  }
261
262  public function __get($name)
263  {
264      // ...
265  }
266
267  public function __set($name, $value)
268  {
269      // ...
270  }
271
272  public function __toString()
273  {
274      // ...
275  }
276
277  public function __wakeup()
278  {
279      // ...
280  }
281
282  public function __destruct()
283  {
284      // ...
285  }
286
287  public function __call($method, $args)
288  {
289      // ...
290  }
291
292  public function __get($name)
293  {
294      // ...
295  }
296
297  public function __set($name, $value)
298  {
299      // ...
300  }
301
302  public function __toString()
303  {
304      // ...
305  }
306
307  public function __wakeup()
308  {
309      // ...
310  }
311
312  public function __destruct()
313  {
314      // ...
315  }
316
317  public function __call($method, $args)
318  {
319      // ...
320  }
321
322  public function __get($name)
323  {
324      // ...
325  }
326
327  public function __set($name, $value)
328  {
329      // ...
330  }
331
332  public function __toString()
333  {
334      // ...
335  }
336
337  public function __wakeup()
338  {
339      // ...
340  }
341
342  public function __destruct()
343  {
344      // ...
345  }
346
347  public function __call($method, $args)
348  {
349      // ...
350  }
351
352  public function __get($name)
353  {
354      // ...
355  }
356
357  public function __set($name, $value)
358  {
359      // ...
360  }
361
362  public function __toString()
363  {
364      // ...
365  }
366
367  public function __wakeup()
368  {
369      // ...
370  }
371
372  public function __destruct()
373  {
374      // ...
375  }
376
377  public function __call($method, $args)
378  {
379      // ...
380  }
381
382  public function __get($name)
383  {
384      // ...
385  }
386
387  public function __set($name, $value)
388  {
389      // ...
390  }
391
392  public function __toString()
393  {
394      // ...
395  }
396
397  public function __wakeup()
398  {
399      // ...
400  }
401
402  public function __destruct()
403  {
404      // ...
405  }
406
407  public function __call($method, $args)
408  {
409      // ...
410  }
411
412  public function __get($name)
413  {
414      // ...
415  }
416
417  public function __set($name, $value)
418  {
419      // ...
420  }
421
422  public function __toString()
423  {
424      // ...
425  }
426
427  public function __wakeup()
428  {
429      // ...
430  }
431
432  public function __destruct()
433  {
434      // ...
435  }
436
437  public function __call($method, $args)
438  {
439      // ...
440  }
441
442  public function __get($name)
443  {
444      // ...
445  }
446
447  public function __set($name, $value)
448  {
449      // ...
450  }
451
452  public function __toString()
453  {
454      // ...
455  }
456
457  public function __wakeup()
458  {
459      // ...
460  }
461
462  public function __destruct()
463  {
464      // ...
465  }
466
467  public function __call($method, $args)
468  {
469      // ...
470  }
471
472  public function __get($name)
473  {
474      // ...
475  }
476
477  public function __set($name, $value)
478  {
479      // ...
480  }
481
482  public function __toString()
483  {
484      // ...
485  }
486
487  public function __wakeup()
488  {
489      // ...
490  }
491
492  public function __destruct()
493  {
494      // ...
495  }
496
497  public function __call($method, $args)
498  {
499      // ...
500  }
501
502  public function __get($name)
503  {
504      // ...
505  }
506
507  public function __set($name, $value)
508  {
509      // ...
510  }
511
512  public function __toString()
513  {
514      // ...
515  }
516
517  public function __wakeup()
518  {
519      // ...
520  }
521
522  public function __destruct()
523  {
524      // ...
525  }
526
527  public function __call($method, $args)
528  {
529      // ...
530  }
531
532  public function __get($name)
533  {
534      // ...
535  }
536
537  public function __set($name, $value)
538  {
539      // ...
540  }
541
542  public function __toString()
543  {
544      // ...
545  }
546
547  public function __wakeup()
548  {
549      // ...
550  }
551
552  public function __destruct()
553  {
554      // ...
555  }
556
557  public function __call($method, $args)
558  {
559      // ...
560  }
561
562  public function __get($name)
563  {
564      // ...
565  }
566
567  public function __set($name, $value)
568  {
569      // ...
570  }
571
572  public function __toString()
573  {
574      // ...
575  }
576
577  public function __wakeup()
578  {
579      // ...
580  }
581
582  public function __destruct()
583  {
584      // ...
585  }
586
587  public function __call($method, $args)
588  {
589      // ...
590  }
591
592  public function __get($name)
593  {
594      // ...
595  }
596
597  public function __set($name, $value)
598  {
599      // ...
600  }
601
602  public function __toString()
603  {
604      // ...
605  }
606
607  public function __wakeup()
608  {
609      // ...
610  }
611
612  public function __destruct()
613  {
614      // ...
615  }
616
617  public function __call($method, $args)
618  {
619      // ...
620  }
621
622  public function __get($name)
623  {
624      // ...
625  }
626
627  public function __set($name, $value)
628  {
629      // ...
630  }
631
632  public function __toString()
633  {
634      // ...
635  }
636
637  public function __wakeup()
638  {
639      // ...
640  }
641
642  public function __destruct()
643  {
644      // ...
645  }
646
647  public function __call($method, $args)
648  {
649      // ...
650  }
651
652  public function __get($name)
653  {
654      // ...
655  }
656
657  public function __set($name, $value)
658  {
659      // ...
660  }
661
662  public function __toString()
663  {
664      // ...
665  }
666
667  public function __wakeup()
668  {
669      // ...
670  }
671
672  public function __destruct()
673  {
674      // ...
675  }
676
677  public function __call($method, $args)
678  {
679      // ...
680  }
681
682  public function __get($name)
683  {
684      // ...
685  }
686
687  public function __set($name, $value)
688  {
689      // ...
690  }
691
692  public function __toString()
693  {
694      // ...
695  }
696
697  public function __wakeup()
698  {
699      // ...
700  }
701
702  public function __destruct()
703  {
704      // ...
705  }
706
707  public function __call($method, $args)
708  {
709      // ...
710  }
711
712  public function __get($name)
713  {
714      // ...
715  }
716
717  public function __set($name, $value)
718  {
719      // ...
720  }
721
722  public function __toString()
723  {
724      // ...
725  }
726
727  public function __wakeup()
728  {
729      // ...
730  }
731
732  public function __destruct()
733  {
734      // ...
735  }
736
737  public function __call($method, $args)
738  {
739      // ...
740  }
741
742  public function __get($name)
743  {
744      // ...
745  }
746
747  public function __set($name, $value)
748  {
749      // ...
750  }
751
752  public function __toString()
753  {
754      // ...
755  }
756
757  public function __wakeup()
758  {
759      // ...
760  }
761
762  public function __destruct()
763  {
764      // ...
765  }
766
767  public function __call($method, $args)
768  {
769      // ...
770  }
771
772  public function __get($name)
773  {
774      // ...
775  }
776
777  public function __set($name, $value)
778  {
779      // ...
780  }
781
782  public function __toString()
783  {
784      // ...
785  }
786
787  public function __wakeup()
788  {
789      // ...
790  }
791
792  public function __destruct()
793  {
794      // ...
795  }
796
797  public function __call($method, $args)
798  {
799      // ...
800  }
801
802  public function __get($name)
803  {
804      // ...
805  }
806
807  public function __set($name, $value)
808  {
809      // ...
810  }
811
812  public function __toString()
813  {
814      // ...
815  }
816
817  public function __wakeup()
818  {
819      // ...
820  }
821
822  public function __destruct()
823  {
824      // ...
825  }
826
827  public function __call($method, $args)
828  {
829      // ...
830  }
831
832  public function __get($name)
833  {
834      // ...
835  }
836
837  public function __set($name, $value)
838  {
839      // ...
840  }
841
842  public function __toString()
843  {
844      // ...
845  }
846
847  public function __wakeup()
848  {
849      // ...
850  }
851
852  public function __destruct()
853  {
854      // ...
855  }
856
857  public function __call($method, $args)
858  {
859      // ...
860  }
861
862  public function __get($name)
863  {
864      // ...
865  }
866
867  public function __set($name, $value)
868  {
869      // ...
870  }
871
872  public function __toString()
873  {
874      // ...
875  }
876
877  public function __wakeup()
878  {
879      // ...
880  }
881
882  public function __destruct()
883  {
884      // ...
885  }
886
887  public function __call($method, $args)
888  {
889      // ...
890  }
891
892  public function __get($name)
893  {
894      // ...
895  }
896
897  public function __set($name, $value)
898  {
899      // ...
900  }
901
902  public function __toString()
903  {
904      // ...
905  }
906
907  public function __wakeup()
908  {
909      // ...
910  }
911
912  public function __destruct()
913  {
914      // ...
915  }
916
917  public function __call($method, $args)
918  {
919      // ...
920  }
921
922  public function __get($name)
923  {
924      // ...
925  }
926
927  public function __set($name, $value)
928  {
929      // ...
930  }
931
932  public function __toString()
933  {
934      // ...
935  }
936
937  public function __wakeup()
938  {
939      // ...
940  }
941
942  public function __destruct()
943  {
944      // ...
945  }
946
947  public function __call($method, $args)
948  {
949      // ...
950  }
951
952  public function __get($name)
953  {
954      // ...
955  }
956
957  public function __set($name, $value)
958  {
959      // ...
960  }
961
962  public function __toString()
963  {
964      // ...
965  }
966
967  public function __wakeup()
968  {
969      // ...
970  }
971
972  public function __destruct()
973  {
974      // ...
975  }
976
977  public function __call($method, $args)
978  {
979      // ...
980  }
981
982  public function __get($name)
983  {
984      // ...
985  }
986
987  public function __set($name, $value)
988  {
989      // ...
990  }
991
992  public function __toString()
993  {
994      // ...
995  }
996
997  public function __wakeup()
998  {
999      // ...
1000 }
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892

```



# The Future

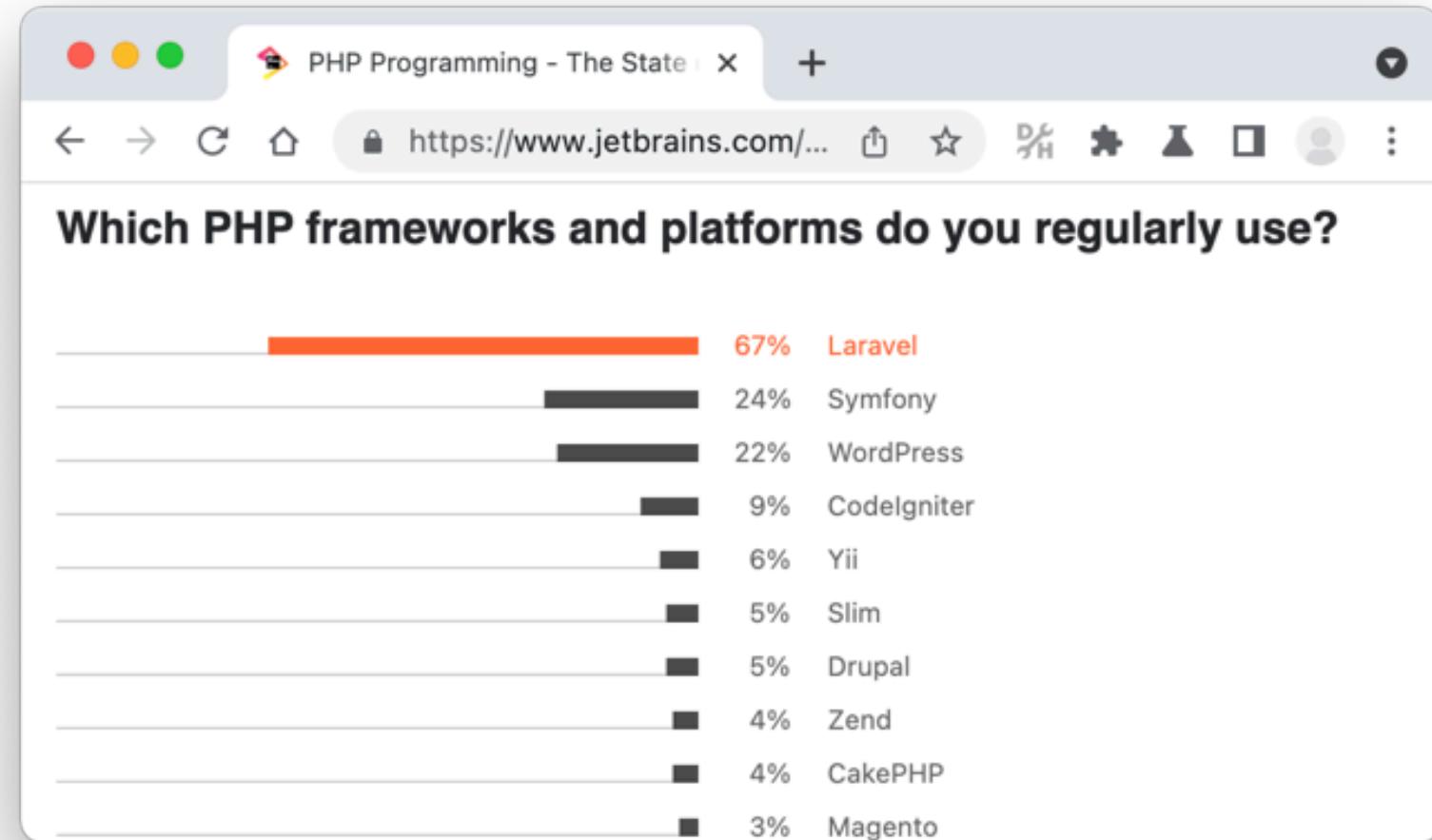
**How programming languages can help**



The screenshot shows a web browser window with the title "PHP Programming - The State" and the URL "https://www.jetbrains.com...". The main content is a survey question: "Do you use static analysis?". Below the question, there are two categories: "PHP" and "Other developers". For each category, there are three horizontal bars representing different responses. The data is summarized in the following table:

Response	PHP (%)	Other developers (%)
Yes	33%	40%
No	38%	33%
I don't know what static analysis is	28%	27%

The text below the chart states: "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, and the `is_literal()` RFC

Thanks to Joe Watkins and Máté Kocsis



The screenshot shows a web browser window with the following details:

- Title Bar:** PHP: rfc:is\_literal
- Address Bar:** https://wiki.php.net/rfc/is\_literal
- Header:** php Edit this page Admin Logout Craig Francis (craigfrancis) Search
- Breadcrumbs:** start > rfc > is\_literal
- Section Header:** PHP RFC: Is\_Literal
- List of Information:**
  - 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](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>
- 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)**



## Backwards Compatibility

```
1 <?php
2
3 class db {
4
5     private function literal_check($var) {
6
7         if (function_exists('is_literal') && !is_literal($var)) {
8             throw new Exception('Non-literal value detected!');
9         }
10    }
11
12
13     public function query($sql, $parameters = []) {
14
15         $this->literal_check($sql);
16
17         // Send $sql and $parameters to the database.
18
19    }
20
```



```
1 <?php
2
3 class db {
4
5     private function literal_check($var) {
6
7         if (function_exists('is_literal') && !is_literal($var)) {
8             throw new Exception('Non-literal value detected!');
9         }
10    }
11
12    public function query($sql, $parameters = []) {
13
14        $this->literal_check($sql); ← Run the check
15
16        // Send $sql and $parameters to the database.
17
18    }
19
20 }
```



```
1 <?php
2
3 class db {
4
5     private function literal_check($var) {
6
7         if (function_exists('is_literal') && !is_literal($var)) {
8             throw new Exception('Non-literal value detected!');
9         }
10    }
11
12    public function query($sql, $parameters = []) {
13        $this->literal_check($sql);
14
15        // Send $sql and $parameters to the database.
16
17    }
18
19}
```

Too Strict?



```
1 <?php
2
3 class db {
4
5     private $protection_level = 1;
6     // 0 = No checks, could be useful on the production server.
7     // 1 = Just warnings, the default.
8     // 2 = Exceptions, for anyone who wants to be absolutely sure.
9
10    public function enforce_injection_protection() {
11        $this->protection_level = 2;
12    }
13
14    public function unsafe_disable_injection_protection() {
15        $this->protection_level = 0; // Not recommended, try `new unsafe_value('XXX')`!
16    }
17
18    private function literal_check($var) {
19        if (!function_exists('is_literal') || is_literal($var)) {
20            // Fine - This is a programmer defined string (bingo), or not using PHP 8.X
21        } else if ($var instanceof unsafe_value) {
22            // Fine - Not ideal, but at least they know this one is unsafe.
23        } else if ($this->protection_level === 0) {
24            // Fine - Programmer aware, and is choosing to disable this check everywhere.
25        } else if ($this->protection_level === 1) {
26            trigger_error('Non-literal value detected!', E_USER_WARNING);
27        } else {
28            throw new Exception('Non-literal value detected!');
29        }
30    }

```

## Protection Level



```
1 <?php
2
3 class db {
4
5     private $protection_level = 1;
6     // 0 = No checks, could be useful on the production server.
7     // 1 = Just warnings, the default.
8     // 2 = Exceptions, for anyone who wants to be absolutely sure.
9
10    public function enforce_injection_protection() {
11        $this->protection_level = 2;
12    }
13
14    public function unsafe_disable_injection_protection() {
15        $this->protection_level = 0; // Not recommended, try `new unsafe_value('XXX')`!
16    }
17
18    private function literal_check($var) {
19        if (!function_exists('is_literal') || is_literal($var)) {
20            // Fine - This is a programmer defined string (bingo), or not using PHP 8.X
21        } else if ($var instanceof \unsafe_value) {
22            // Fine - Not ideal, but at least they know this one is unsafe.
23        } else if ($this->protection_level === 0) {
24            // Fine - Programmer aware, and is choosing to disable this check everywhere.
25        } else if ($this->protection_level === 1) {
26            trigger_error('Non-literal value detected!', E_USER_WARNING);
27        } else {
28            throw new Exception('Non-literal value detected!');
29        }
30    }
}
```

Private function,  
Used by the library



```
1 <?php
2
3 class db {
4
5     private $protection_level = 1;
6     // 0 = No checks, could be useful on the production server.
7     // 1 = Just warnings, the default.
8     // 2 = Exceptions, for anyone who wants to be absolutely sure.
9
10    public function enforce_injection_protection() {
11        $this->protection_level = 2;
12    }
13
14    public function unsafe_disable_injection_protection() {
15        $this->protection_level = 0; // Not recommended, try `new unsafe_value('XXX')`!
16    }
17
18    private function literal_check($var) {
19        if (!function_exists('is_literal') || is_literal($var)) {
20            // Fine - This is a programmer defined string (bingo), or not using PHP 8.X
21        } else if ($var instanceof unsafe_value) {
22            // Fine - Not ideal, but at least they know this one is unsafe.
23        } else if ($this->protection_level === 0) {
24            // Fine - Programmer aware, and is choosing to disable this check everywhere.
25        } else if ($this->protection_level === 1) {
26            trigger_error('Non-literal value detected!', E_USER_WARNING);
27        } else {
28            throw new Exception('Non-literal value detected!');
29        }
30    }
}
```

Special Cases?



The screenshot shows a code editor window with a dark theme. The title bar has three colored circles (red, yellow, green) and the text 'Line: 1 | PHP | Tab Size: 4 | Symbols'. The code itself is:

```
<?php

class unsafe_value {

    private $value = '';

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

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

?>
```

**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?



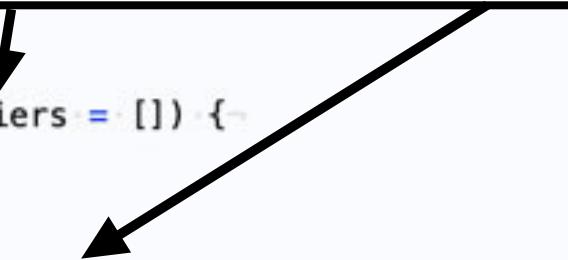
```
1  <?php
2
3  class db {
4
5      // ...
6
7      function query($sql, $parameters = [], $identifiers = []) {
8
9          $this->literal_check($sql);
10
11         foreach ($identifiers as $name => $value) {
12             if (!preg_match('/^[\w\-\_]+$/i', $name)) {
13                 throw new Exception('Invalid identifier name "' . $name . '"');
14             } else if (!preg_match('/^[\w\-\_]+$/i', $value)) {
15                 throw new Exception('Invalid identifier value "' . $value . '"');
16             } else {
17                 $sql = str_replace('{' . $name . '}', $value, $sql);
18             }
19         }
20
21         // Send $sql and $parameters to the database.
22
23     }
24
```



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.

```
1  <?php
2
3  class db {
4
5      // ...
6
7      function query($sql, $parameters = [], $identifiers = []) {
8
9          $this->literal_check($sql);
10
11         foreach ($identifiers as $name => $value) {
12             if (!preg_match('/^[_a-z0-9]+$/i', $name)) {
13                 throw new Exception('Invalid identifier name "' . $name . '"');
14             } else if (!preg_match('/^[_a-z0-9]+$/i', $value)) {
15                 throw new Exception('Invalid identifier value "' . $value . '"');
16             } else {
17                 $sql = str_replace('{' . $name . '}', $value, $sql);
18             }
19         }
20
21         // Send $sql and $parameters to the database.
22
23     }
24 }
```





```
1 <?php
2
3 class db {
4
5     // ...
6
7     function query($sql, $parameters = [], $identifiers = []) {
8
9         $this->literal_check($sql);
10
11        foreach ($identifiers as $name => $value) {
12            if (!preg_match('/^[\w\000-\010]*$/u', $name)) {
13                throw new Exception('Invalid identifier name "' . $name . '"');
14            } else if (!preg_match('/^[\w\000-\010]*$/u', $value)) {
15                throw new Exception('Invalid identifier value "' . $value . '"');
16            } else {
17                $sql = str_replace('{' . $name . '}', '' . $value . '', $sql);
18            }
19        }
20
21        // Send $sql and $parameters to the database.
22    }
23}
```



Over to you :-)



```
1 <?php
2
3     $db = new db();
4
5     $db->query( 'SELECT name FROM user WHERE id = ?' , [ $id ] );
6
7     $db->query( 'SELECT name FROM user WHERE id = ' . $id ); // REJECTED
8
9 ?>
```

Line: 1 | PHP Tab Size: 4 | Symbols





## Using Identifiers

```
<?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);

The screenshot shows a code editor window titled "1.php — Downloads". The code is a PHP class definition:<?php  
class orm {  
 // ...  
 function where(\$field, \$value = NULL) {  
 \$this->literal\_check(\$field);  
 // ...  
 }  
}  
?>Line numbers 1 through 17 are visible on the left. A black arrow points from a callout box containing the text "Run the check" to the line "\$this->literal\_check(\$field);". The code editor interface includes tabs for "Line:", "1 | PHP", "Tab Size: 4", and "Symbols".

Run the check



# \$users->order(\$order);

```
1 <?php
2
3 class orm {
4
5     // ...
6
7     function order($sql) {
8
9         $this->literal_check($sql);
10
11        // ...
12
13    }
14
15 }
16
17 ?>
```

Line: 1 | PHP | Tab Size: 4 | Symbols

Run the check



# CLI



```
1 <?php
2
3 function parameterised_exec($cmd, $parameters = []) {
4
5     if (!is_literal($cmd)) {
6         throw new Exception('The first argument must be a literal');
7     }
8
9     $offset = 0;
10    $k = 0;
11    while (($pos = strpos($cmd, '?', $offset)) !== false) {
12        if (!isset($parameters[$k])) {
13            throw new Exception('Missing parameter #' . ($k + 1));
14            exit();
15        }
16        $par = escapeshellarg($parameters[$k]);
17        $cmd = substr($cmd, 0, $pos) . $par . substr($cmd, ($pos + 1));
18        $offset = ($pos + strlen($par));
19        $k++;
20    }
21    if (isset($parameters[$k])) {
22        throw new Exception('Unused parameter #' . ($k + 1));
23        exit();
24    }
25
26    return exec($cmd);
27
28}
```

Strict Check



```
1 <?php
2
3 function parameterised_exec($cmd, $parameters = []) {
4
5     if (!is_literal($cmd)) {
6         throw new Exception('The first argument must be a literal');
7     }
8
9     $offset = 0;
10    $k = 0;
11    while (($pos = strpos($cmd, '?', $offset)) !== false) {
12        if (!isset($parameters[$k])) {
13            throw new Exception("Missing parameter " . ($k + 1));
14            exit();
15        }
16        $par = escapeShellArg($parameters[$k]);
17        $cmd = substr($cmd, 0, $pos) . $par . substr($cmd, ($pos + 1));
18        $offset = ($pos + strlen($par));
19        $k++;
20    }
21    if (isset($parameters[$k])) {
22        throw new Exception("Unused parameter " . ($k + 1));
23        exit();
24    }
25
26    return exec($cmd);
27
28 }
```

Escaping all values

Apply \$parameters



```
1 <?php
2
3 function parameterised_exec($cmd, $parameters = []) {
4
5     if (!is_literal($cmd)) {
6         throw new Exception('The first argument must be a literal');
7     }
8
9     $offset = 0;
10    $k = 0;
11    while (($pos = strpos($cmd, '?', $offset)) !== false) {
12        if (!isset($parameters[$k])) {
13            throw new Exception("Missing parameter " . ($k + 1));
14            exit();
15        }
16        $par = escapeShellArg($parameters[$k]);
17        $cmd = substr($cmd, 0, $pos) . $par . substr($cmd, ($pos + 1));
18        $offset = ($pos + strlen($par));
19        $k++;
20    }
21    if (isset($parameters[$k])) {
22        throw new Exception("Unused parameter " . ($k + 1));
23        exit();
24    }
25
26    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**





A screenshot of a terminal window displaying a massive amount of text, likely a stack trace or a large dump of memory dump. The text is mostly black on a white background, with some red and blue highlights. The terminal window has a dark theme.

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>
    
    <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.**



The screenshot shows a code editor window with the following PHP code:

```
<?php
class unsafe_value {
    private $value = '';
    function __construct($unsafe_value) {
        $this->value = $unsafe_value;
    }
    function __toString() {
        return $this->value;
    }
}
?>
```

The code defines a class named `unsafe_value` with a private attribute `$value` and two methods: `__construct` and `__toString`. The `__construct` method takes a parameter `$unsafe_value` and assigns it to `$this->value`. The `__toString` method returns the value of `$this->value`.

At the bottom of the editor, there are navigation controls: Line: 1 | PHP | Tab Size: 4 | Symbols.

**Maybe a "stringable value-object"?**



```
Line: 1 | PHP | Tab Size: 4 | Symbols | 
1 <?php
2
3 ▼ class db_trusted_sql {
4
5     private $sql = '';
6
7 ▼     function __construct($sql) {
8         $this->sql = $sql;
9     }
10
11 ▼     function __toString() {
12         return $this->sql;
13     }
14
15 ▲ }
16
17 ?>
```

Some way to trust the stringable value object

What it can be trusted for.

```
3 ▼ class db {
4
5     //...
6
7 ▼     function __construct() {
8
9     //...
10
11     spl_trust_object('db_trusted_sql', 'sql');
12
13 ▲ }
14
15 ▼     function query($sql, $parameters = []) {
16
17     //...
18
19     $trusted_sql = new db_trusted_sql($sql);
20
21     mysqli_query($this->link, $trusted_sql);
22
23 ▲ }
24
```

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  
**>virtual**  
APPSEC

THANK YOU!