



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**

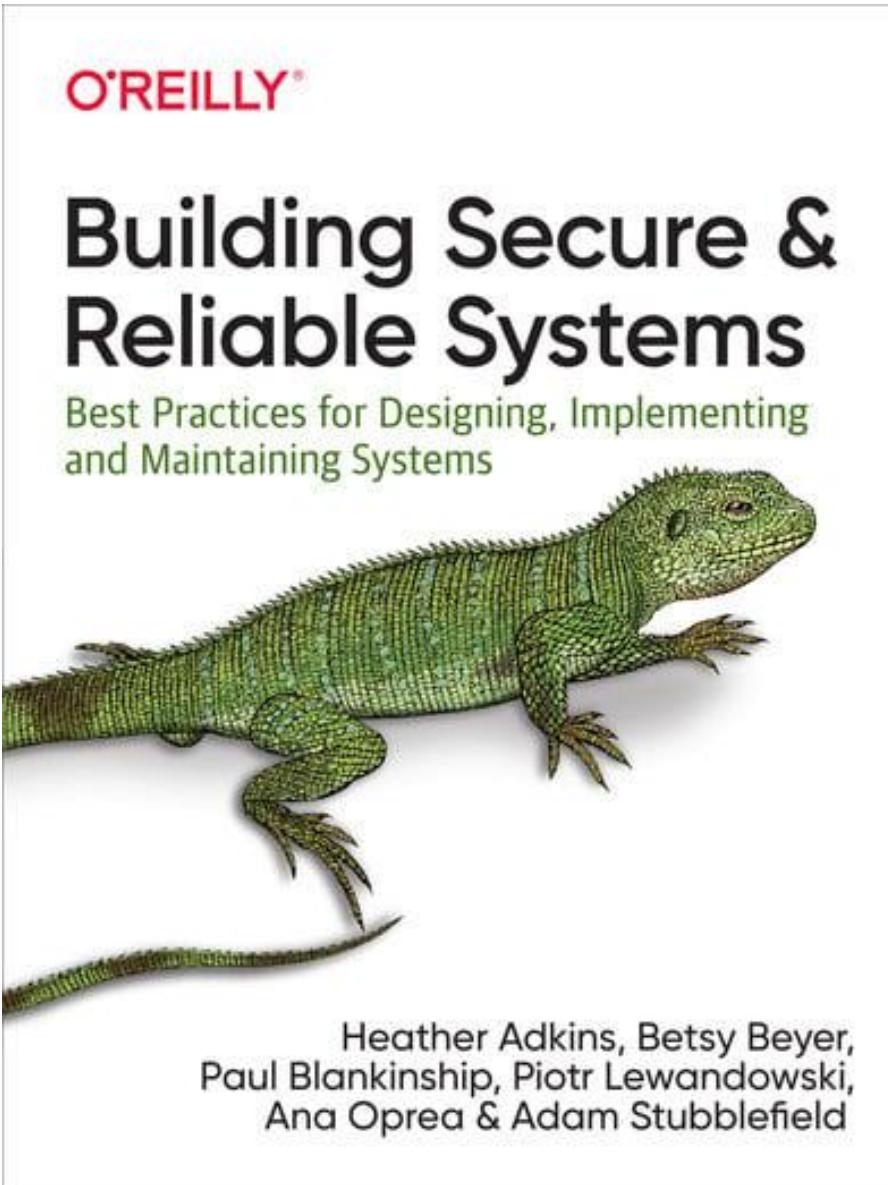
# Christoph Kern

## 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 table of user information. The table has two columns: 'Name' and 'Email'. The 'Name' column contains four rows: Amy Anderson, Benjamin Binder, Charlotte Clark, and Donna Davis. The 'Email' column contains four rows: user1@example.com, user2@example.com, user3@example.com, and user4@example.com. The browser address bar shows the URL `https://example.com/?order=name_first,name_last`. A large pink arrow points from the bottom of the table down to the corresponding SQL query.

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

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

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



A screenshot of a web browser window displaying a user list table. The table has two columns: 'Name' and 'Email'. The 'Name' column contains four rows: Amy Anderson, Benjamin Binder, Charlotte Clark, and Donna Davis. The 'Email' column contains four rows: user1@example.com, user2@example.com, user3@example.com, and user4@example.com. A pink arrow points from the bottom of the table to the code below.

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

**\$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
```





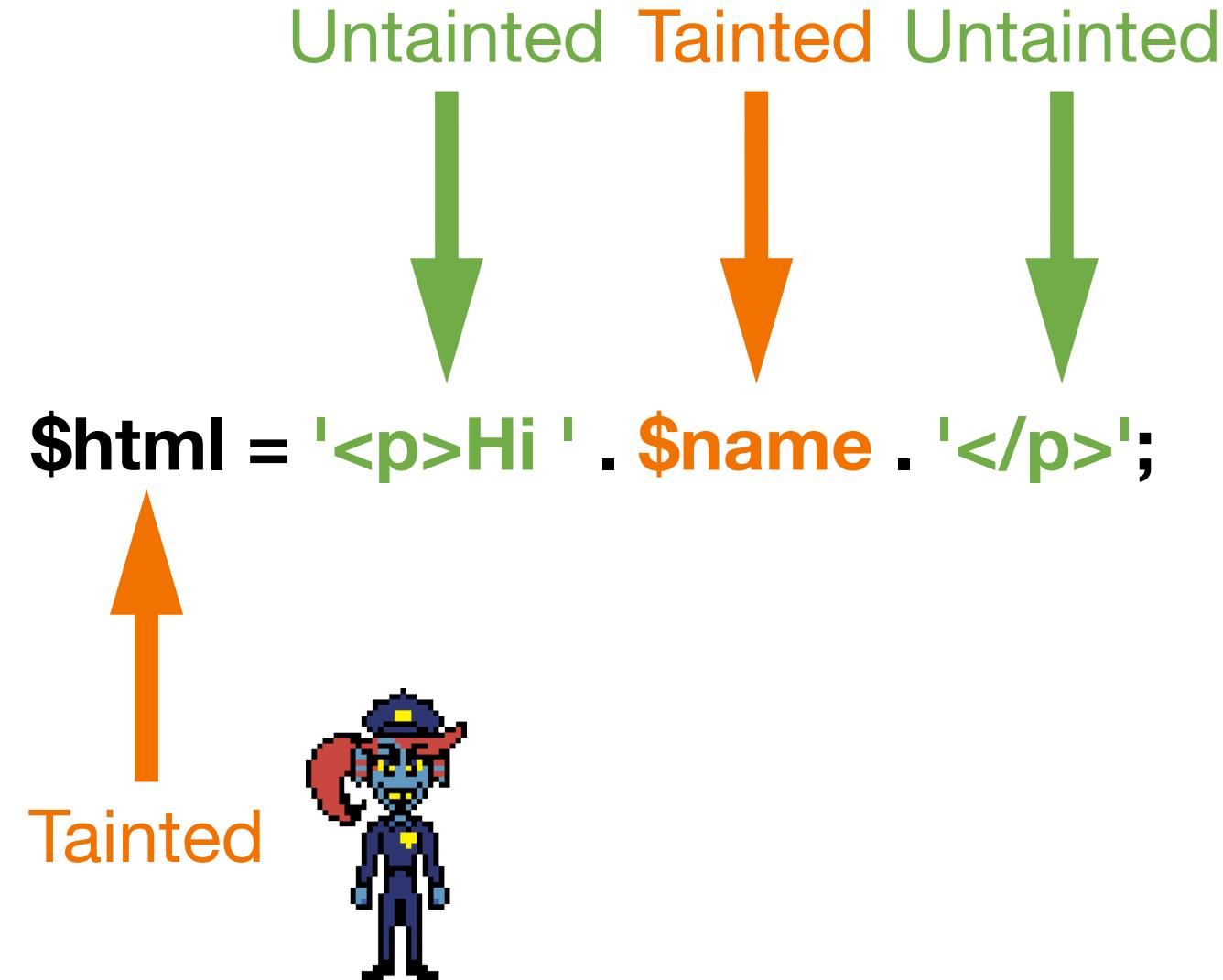
# How do we stop mistakes?

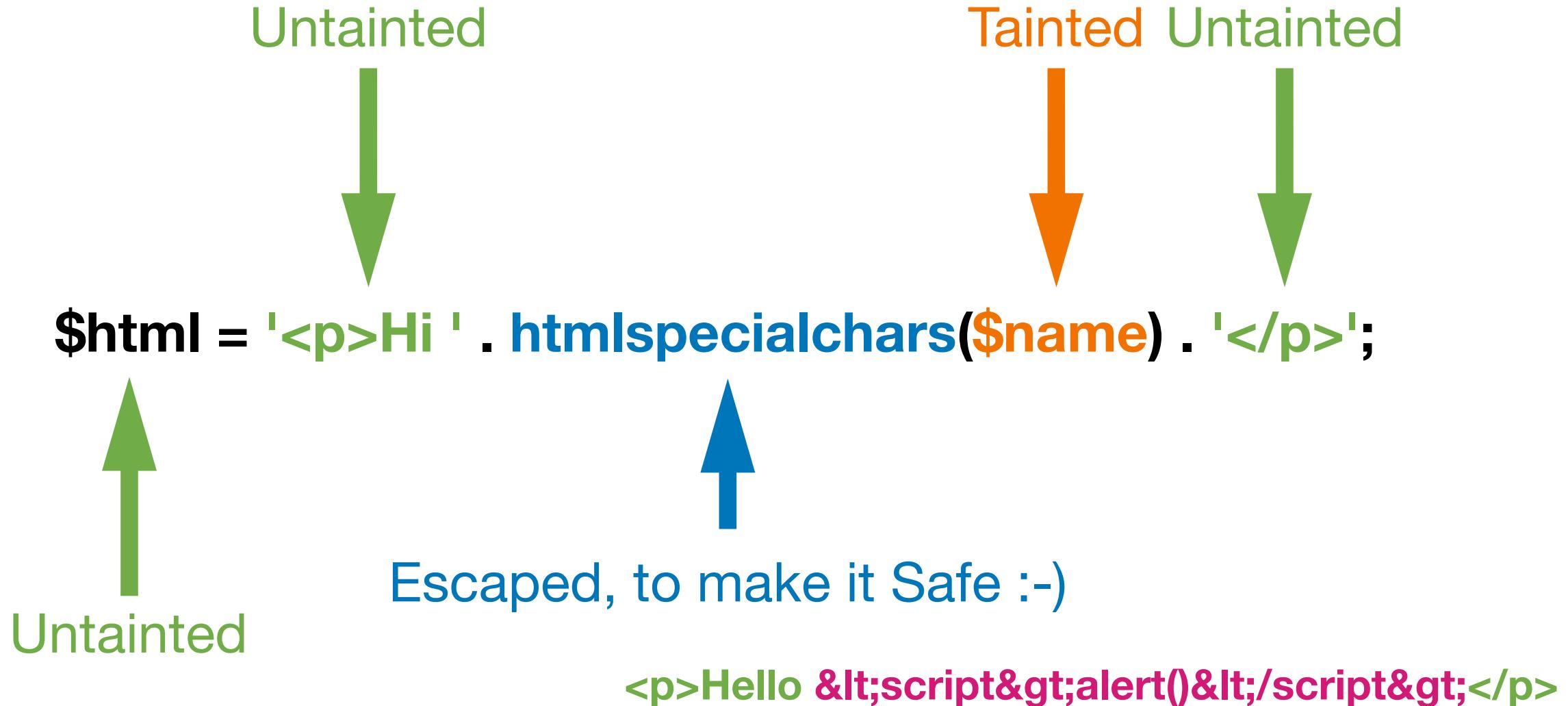


# Maybe Taint Checking?

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









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

Untainted

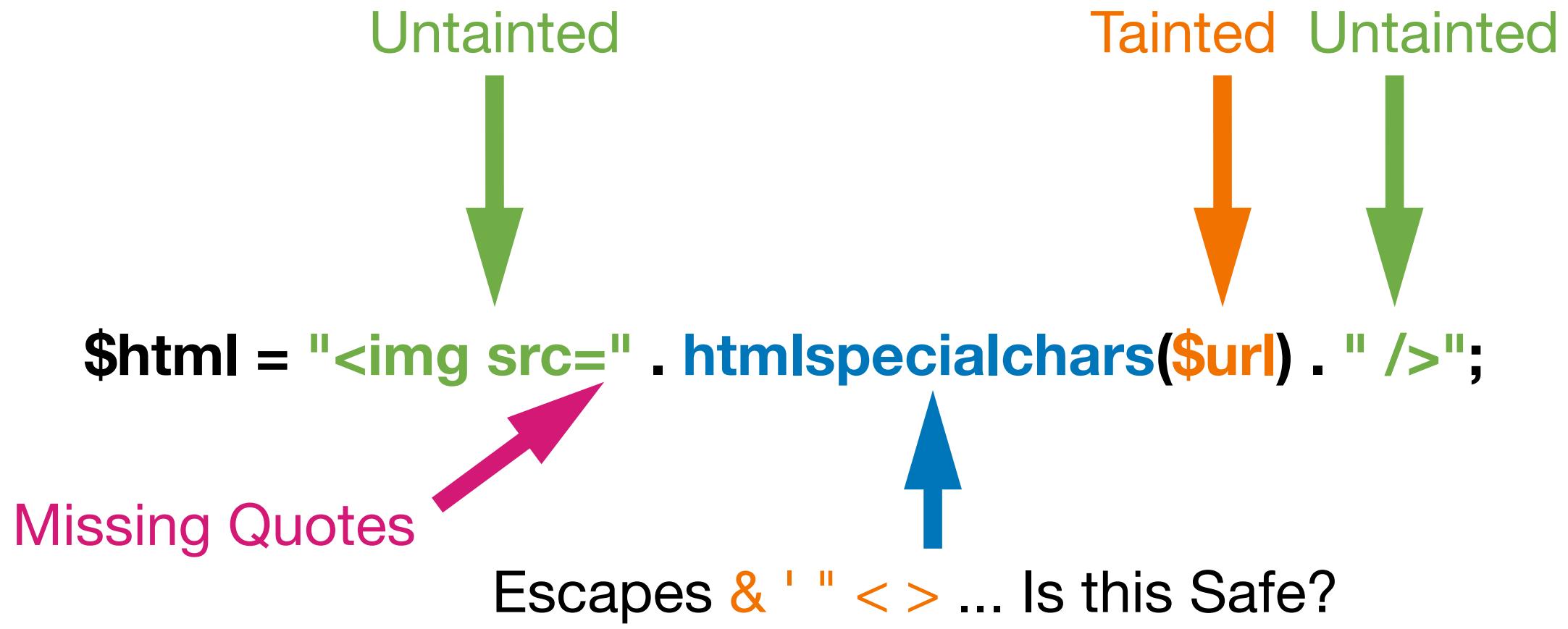
Tainted Untainted

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

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

```
<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 :-)



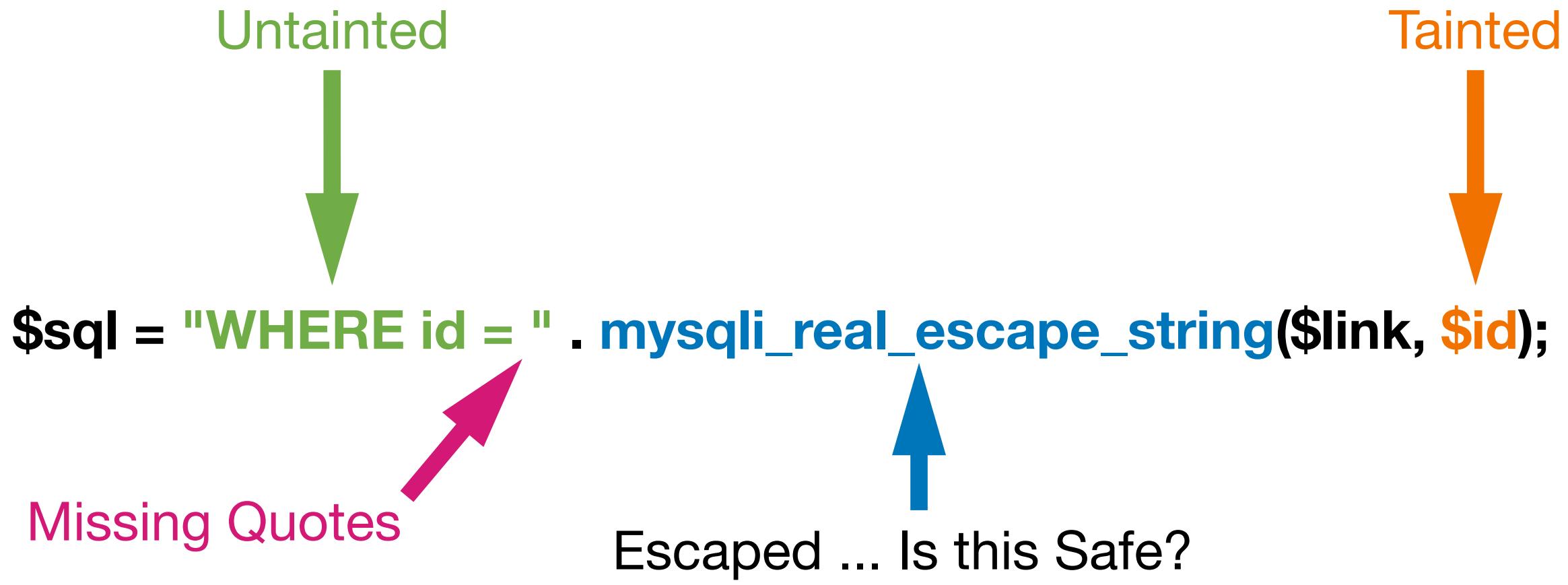
Is this Safe?

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.**



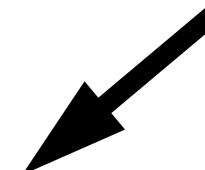
Instead, we can simplify it,  
by looking for "strings from a trusted developer"...

**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

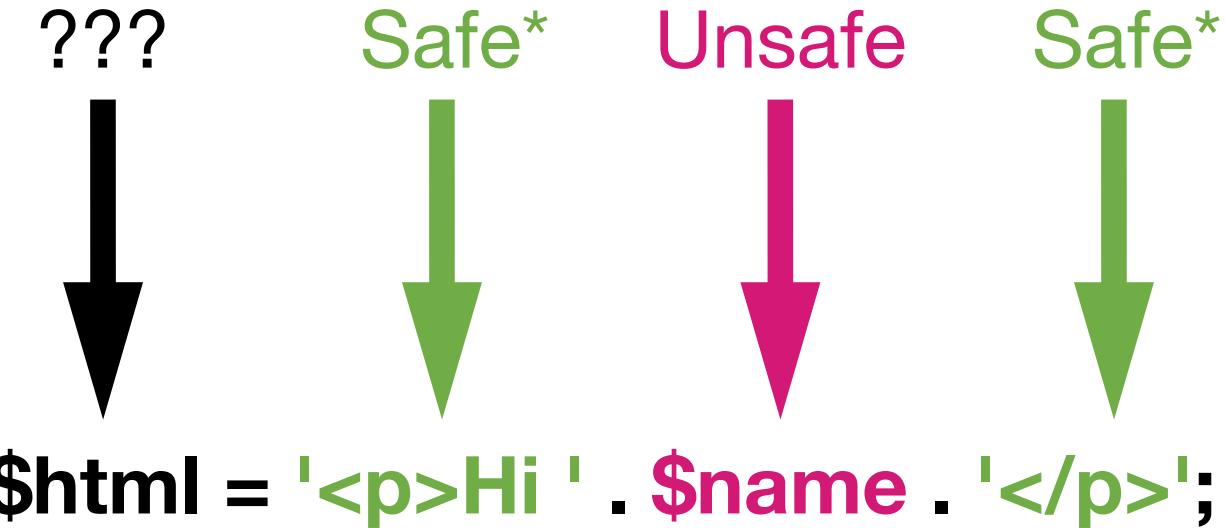


Safe\*      Unsafe      Safe\*



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

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



**\$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);
```



```
'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**



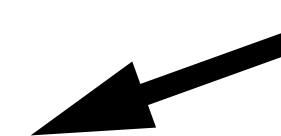
Could try to escape the field...  
But should *any* field be allowed?



```
$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`.

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

Annotations with arrows point from the text boxes to specific parts of the code:

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

Text boxes with annotations:

- "`stringConstant` type; not exported"
- "OnlyAcceptsStringConstant" function; is exported
- "`stringConstant` type; required for input"



```
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



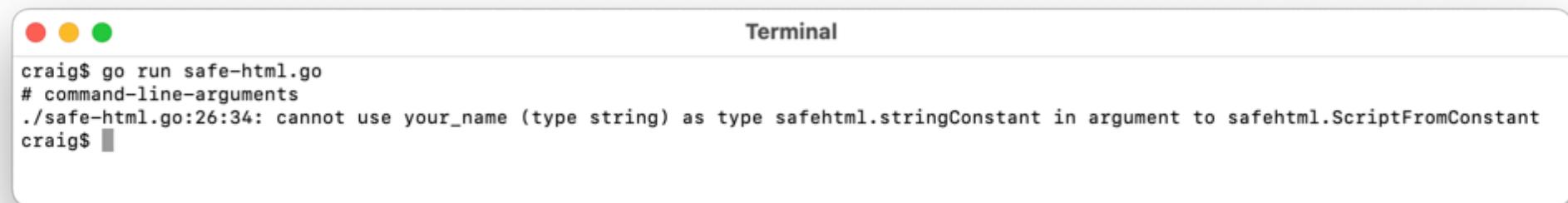
## How this is used by "go-safe-html"



**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 Mac OS X terminal window titled "Terminal". The window has three red, yellow, and green 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$
```



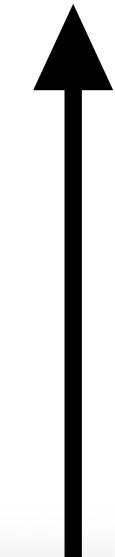
```
a := template.MustParseAndExecuteToHTML("<p>Hello ")  
b := safehtml.HTMLEscaped(your_name)  
c := template.MustParseAndExecuteToHTML("</p>")  
  
para := safehtml.HTMLConcat(a, b, c)
```



<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)
```



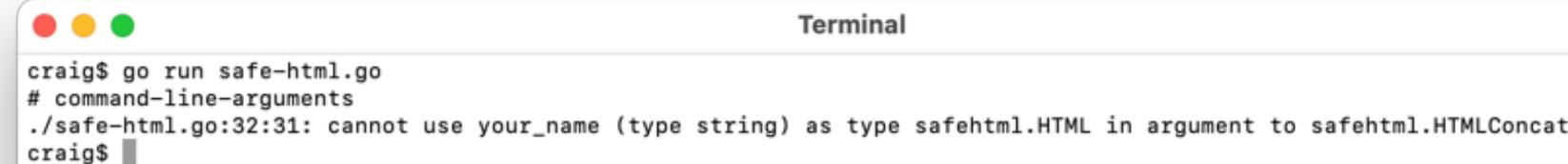
```
craig$ go run safe-html.go  
# command-line-arguments  
.safe-html.go:29:41: cannot use your_name (type string) as type "github.com/google/safehtml/template".stringConstant in  
argument to "github.com/google/safehtml/template".MustParseAndExecuteToHTML  
craig$
```



```
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

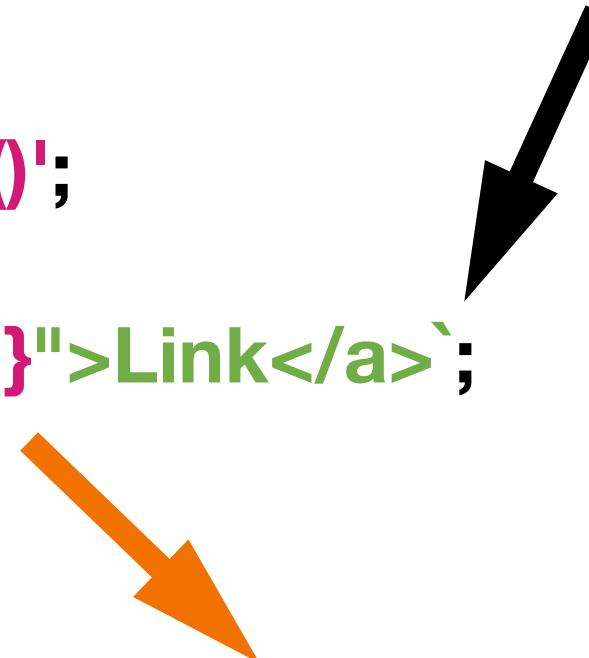
## With Node... and JavaScript (soon)



## Template Literal

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

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

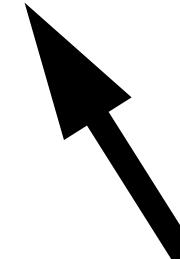


```
'<a href="javascript:alert()">Link</a>'
```



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

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

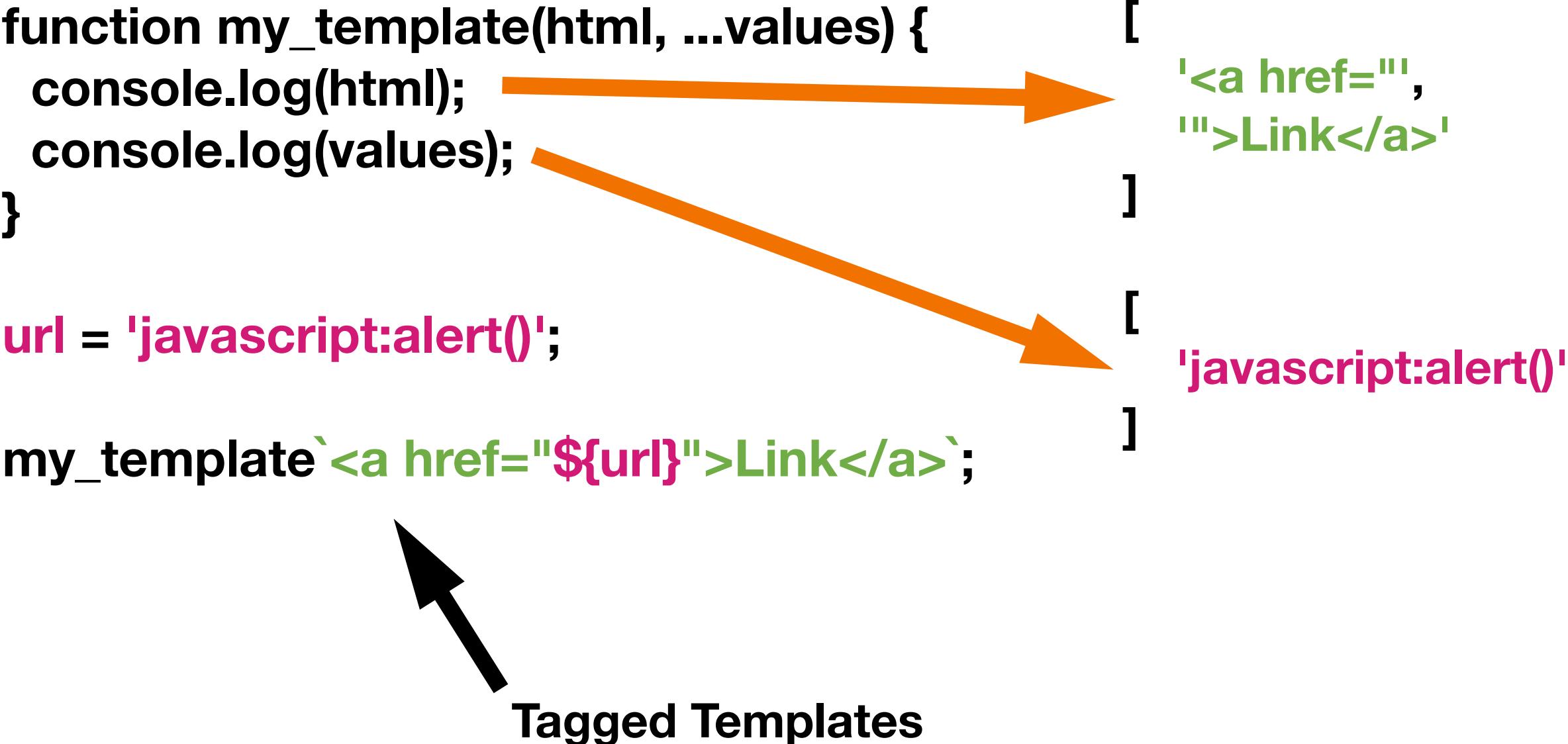


Tagged Templates

```
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!'; }
```

...

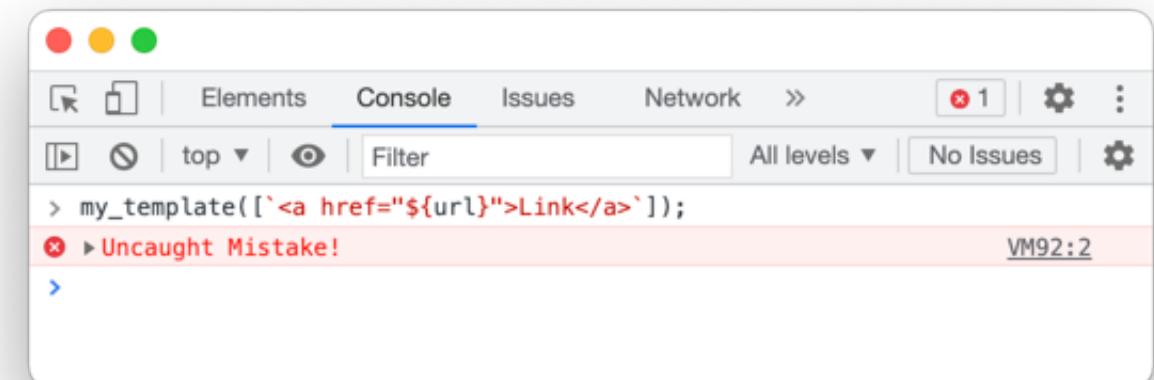
}

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
17
```

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');

**Require Polyfill,  
from Mike Samuel**

**Call  
isTemplateObject**



```
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**

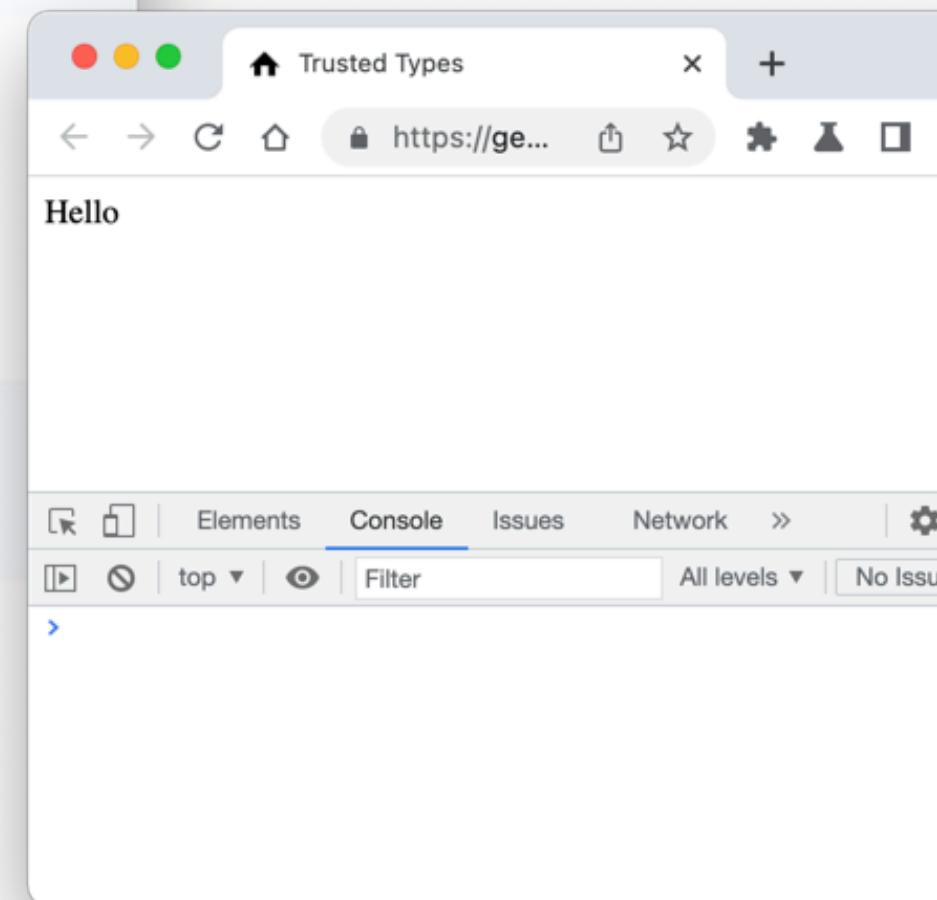
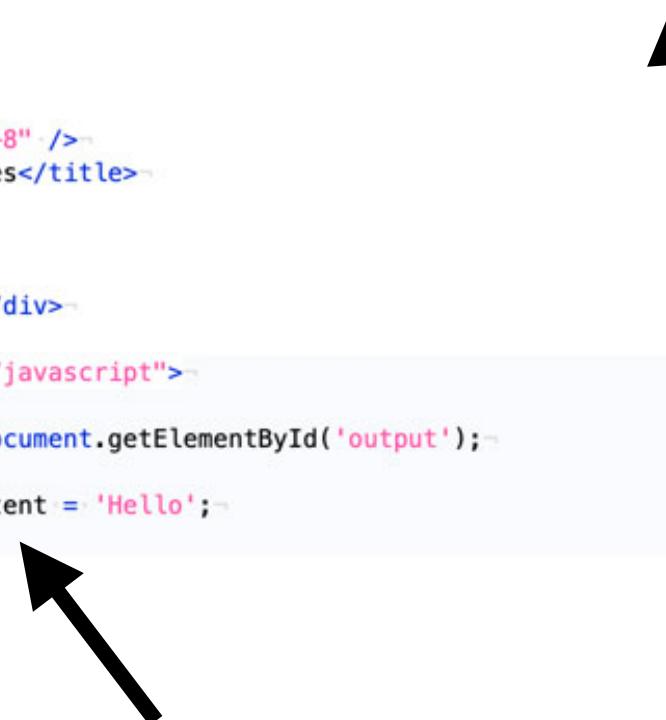


**or JavaScript and Trusted Types... Hopefully Soon™**

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

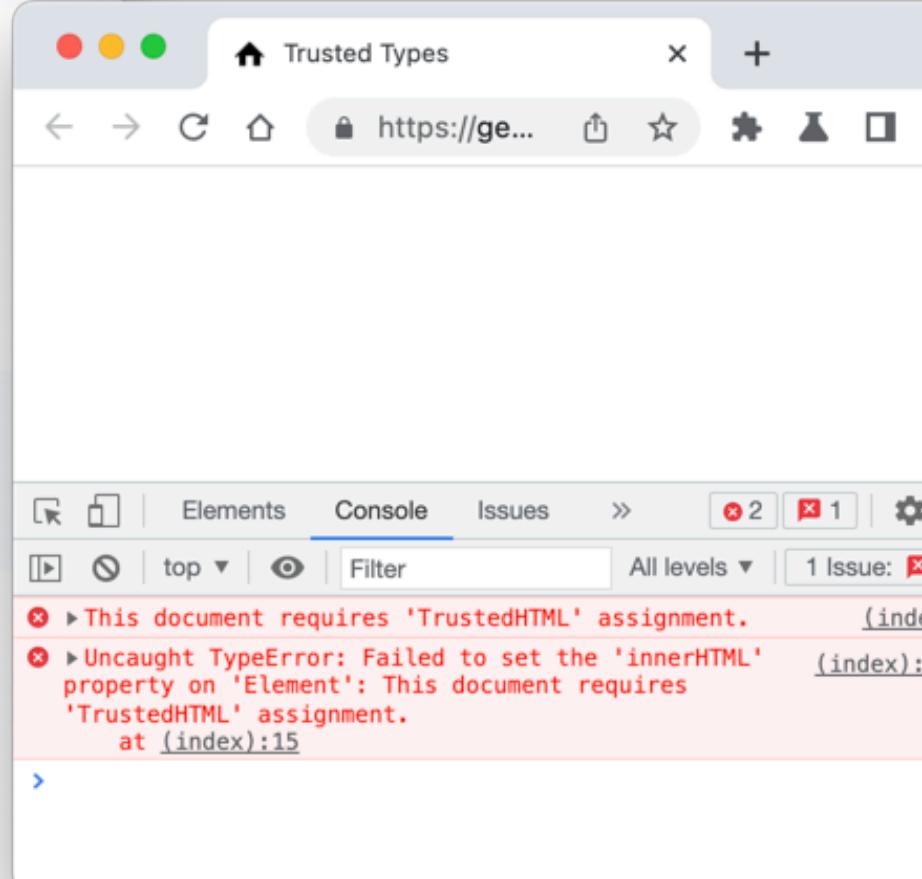
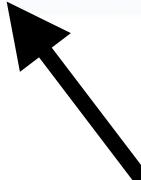


```
1 <?php
2
3     header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types 'none';");
4
5 ?>
6 <!DOCTYPE html>
7 <html lang="en-GB">
8 <head>
9     <meta charset="UTF-8" />
10    <title>Trusted Types</title>
11 </head>
12 <body>
13
14     <div id="output"></div>
15
16     <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.

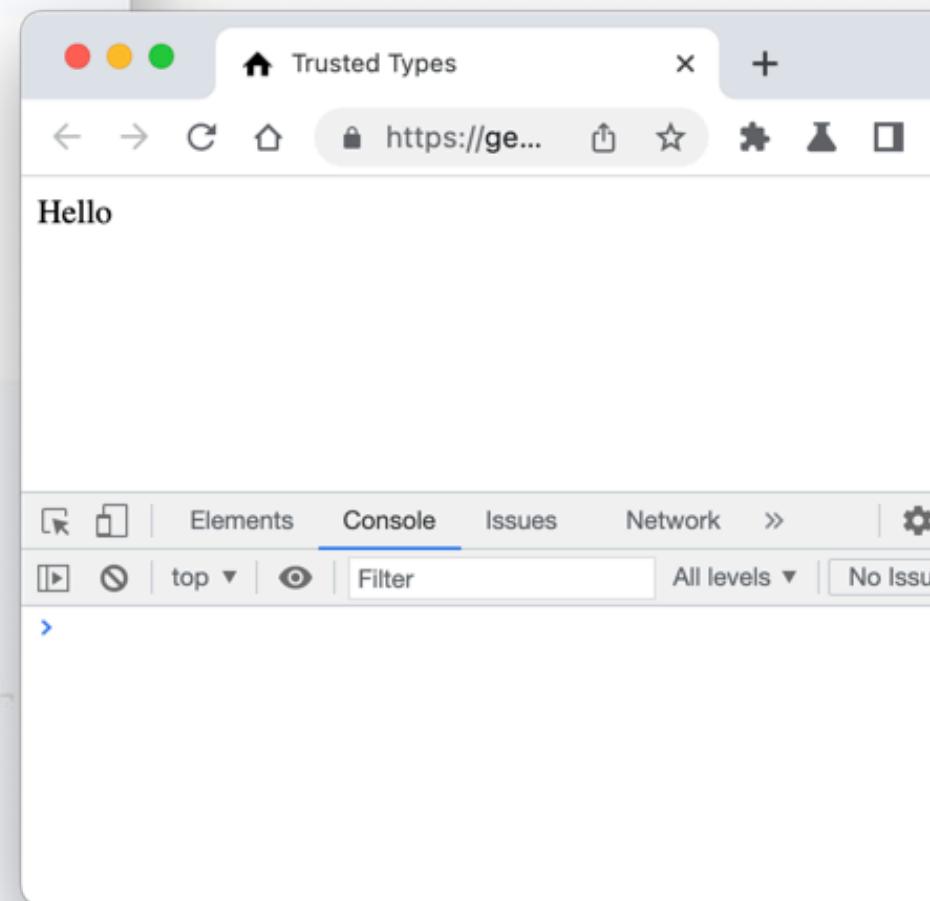


```
1 <?php
2
3     header("Content-Security-Policy: require-trusted-types-for 'script'; trusted-types my_trusted_type;");
4
5 ?>
6 <!DOCTYPE html>
7 <html lang="en-GB">
8 <head>
9     <meta charset="UTF-8" />
10    <title>Trusted Types</title>
11 </head>
12 <body>
13
14 <div id="output"></div>
15
16 <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>
```

```
</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 →

2 →

```
</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-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>
```



There are a few  
dependencies

```
Terminal
$ java -jar build/libs/itstlethal-1.0.jar
[INFO] Scanning for projects...
[INFO] Building itstlethal 1.0
[INFO]   [jar]
[INFO] --- com.example.itstlethal:itstlethal:1.0@jar:jar:1.0 ---
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/com/google/errorprone/error_prone_annotations/2.0.8/error_prone_annotations-2.0.8.pom (1.3 kB at 4.2 kB/s)
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/com/google/errorprone/error_prone_annotations/2.0.8/error_prone_annotations-2.0.8.jar (4.8 kB at 89 kB/s)
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/com/google/errorprone/error_prone_annotations/2.0.8/error_prone_annotations-2.0.8.pom (1.3 kB at 4.2 kB/s)
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/com/google/errorprone/error_prone_annotations/2.0.8/error_prone_annotations-2.0.8.jar (4.8 kB at 89 kB/s)
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/com/google/errorprone/error_prone_annotations/2.0.8/error_prone_annotations-2.0.8.pom (1.3 kB at 51 kB/s)
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/com/google/errorprone/error_prone_annotations/2.0.8/error_prone_annotations-2.0.8.jar (4.8 kB at 212 kB/s)
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ itstlethal ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resource@{relativePath} /WEB-INF/lib/itstlethal/others/java/etc/main/resources
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ itstlethal ---
[INFO] Compiling 1 source file to /Volumes/Websphere/Projects/php_itstlethal/others/java/etc/main/resources
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ itstlethal ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resource@{relativePath} /WEB-INF/lib/itstlethal/others/java/test/main/resources
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ itstlethal ---
[INFO] No tests to run.
[INFO] Total time: 4.227 s
[INFO] Finished at: 2023-10-01T11:48:32
[INFO] -----
[INFO] -----
[INFO] -----
```



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



OWASP 2021  
virtual  
APPSEC

Ending Injection Vulnerabilities

↖\_(ツ)\_↗



# Ending Injection Vulnerabilities In PHP



# Using Psalm

## Static Analysis 1

Thanks to Matthew Brown



**composer require --dev vimeo/psalm**



```
cris@cris-OptiPlex-5070: ~
```

cris@cris-OptiPlex-5070: ~ % 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 dependency info from cache...  
Your lock file is up to date.  
Load time: 0ms (0.0000s)  
- Locking amphp/amp (v2.6.1)  
- Locking amphp/baye-stream (v1.8.1)  
- Locking composer/package-options-deprecated (1.11.99.4)  
- Locking composer/semver (v2.1.0)  
- Locking composer/debug-handler (2.0.2)  
- Locking dnogel/php-xdg-base-dir (v0.1.1)  
- Locking felixfbecker/advanced-json-rpc (v3.2.1)  
- Locking felixfbecker/language-server-protocol (1.5.1)  
- Locking fzaninotto/faker (v1.9.0)  
- Locking nikic/php-parser (v4.13.1)  
- Locking openssl/lib-xml2xml (v1.6.0)  
- Locking phpdocumtor/reflection-common (2.2.0)  
- Locking phpdocumtor/reflection-docblock (5.3.0)  
- Locking phpdocumtor/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/string (v6.3.18)  
- Locking symfony/deprecation-contracts (v2.4.0)  
- Locking symfony/polyfill-c 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-php80 (v1.23.1)  
- Locking symfony/service-contracts (v2.4.0)  
- Locking symfony/string (v6.3.10)  
- Locking vimeo/psalm (4.12.0)  
- Locking webmozart/assert (v1.18.0)  
- Locking webmozart/path-util (2.3.6)  
Writing lock file  
Installing dependencies from lock file (including require-dev)  
Package operations: 0 installs, 0 updates, 0 removals  
- Downloading webmozart/assert (v1.18.0)  
- Downloading phpdocumtor/reflection-docblock (5.3.0)  
- Downloading psr/container (1.1.2)  
- Downloading symfony/string (v6.3.18)  
- Downloading symfony/deprecation-contracts (v2.4.0)  
- Downloading nikic/php-parser (v4.13.1)  
- Downloading composer/server (3.2.6)  
- Downloading vimeo/psalm (4.12.0)  
- Installing composer/package-versions-deprecated (1.11.99.4): Extracting archive  
- Installing dnogel/php-xdg-base-dir (v0.1.1): Extracting archive  
- Installing webmozart/assert (v1.18.0): Extracting archive  
- Installing phpdocumtor/reflection-common (2.2.0): Extracting archive  
- Installing phpdocumtor/type-resolver (1.5.1): Extracting archive  
- Installing phpdocumtor/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-intl-php73 (v1.23.0): Extracting archive  
- Installing psr/container (1.1.2): Extracting archive  
- Installing symfony/service-contracts (v2.4.0): Extracting archive  
- Installing psr/intl (v1.2.0): Extracting archive  
- Installing symfony/polyfill-intl-ctype (v1.23.1): Extracting archive  
- Installing symfony/polyfill-intl-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/polyfill-string (v6.3.18): Extracting archive  
- Installing webmozart/path-util (2.3.6): Extracting archive  
- Installing symfony/console (v6.3.18): Extracting archive  
- Installing sebastian/diff (4.0.4): Extracting archive  
- Installing openssl/lib-xml2xml (v1.6.0): Extracting archive  
- Installing amphp/amp (v2.6.1): Extracting archive  
- Installing amphp/baye-stream (v1.8.1): Extracting archive  
- Installing vimeo/psalm (4.12.0): Extracting archive  
Package suggestions have been added by new dependencies; use 'composer suggest' to see details.  
Package amphp/amp 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  
It 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.





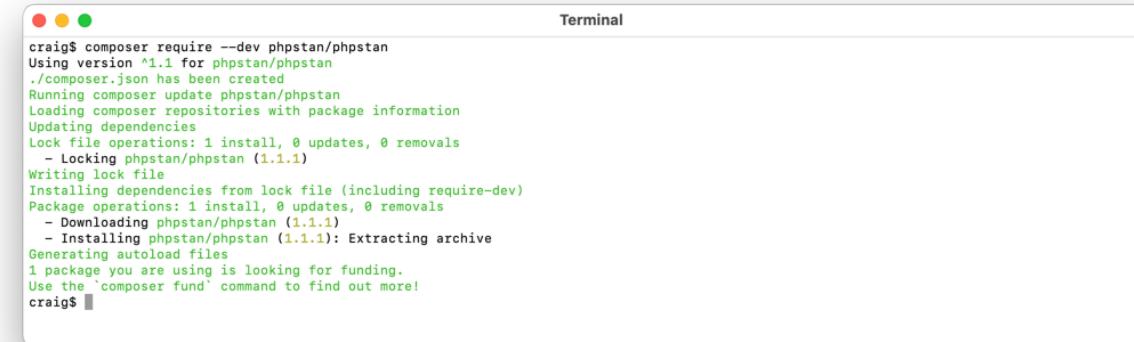
# Using PHPStan

## 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         $stmt = $this->conn->prepare($sql);
10        $stmt->execute($params);
11        $stmt->close();
12        return $stmt;
13    }
14
15    public function fetchAll($sql, $params = [])
16    {
17        $stmt = $this->query($sql, $params);
18        $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
19        $stmt->close();
20        return $rows;
21    }
22
23    public function fetchOne($sql, $params = [])
24    {
25        $stmt = $this->query($sql, $params);
26        $row = $stmt->fetch(PDO::FETCH_ASSOC);
27        $stmt->close();
28        return $row;
29    }
30
31    public function insert($table, $data)
32    {
33        $keys = array_keys($data);
34        $values = array_map(function ($value) {
35            return ':' . $value;
36        }, $data);
37
38        $sql = "INSERT INTO {$table} (" . implode(', ', $keys) . ") VALUES (" . implode(', ', $values) . ")";
39
40        $stmt = $this->query($sql);
41        $stmt->close();
42    }
43
44    public function update($table, $data, $where)
45    {
46        $set = [];
47        foreach ($data as $key => $value) {
48            $set[] = "{$key} = :{$value}";
49        }
50
51        $sql = "UPDATE {$table} SET " . implode(', ', $set) . " WHERE " . $where;
52
53        $stmt = $this->query($sql);
54        $stmt->close();
55    }
56
57    public function delete($table, $where)
58    {
59        $sql = "DELETE FROM {$table} WHERE " . $where;
60
61        $stmt = $this->query($sql);
62        $stmt->close();
63    }
64
65    private $conn;
66
67    public function __construct()
68    {
69        $this->conn = new PDO('mysql:host=' . DB_HOST . ';dbname=' . DB_NAME, DB_USER, DB_PASS);
70        $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
71    }
72}
```

### 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$
```

```
20 $db = new db();
21
22 $db->query('SELECT * FROM user WHERE id = ?' , [$id]);
23
24 $db->query('SELECT * FROM user WHERE id = ' . $id);
```





# The Future

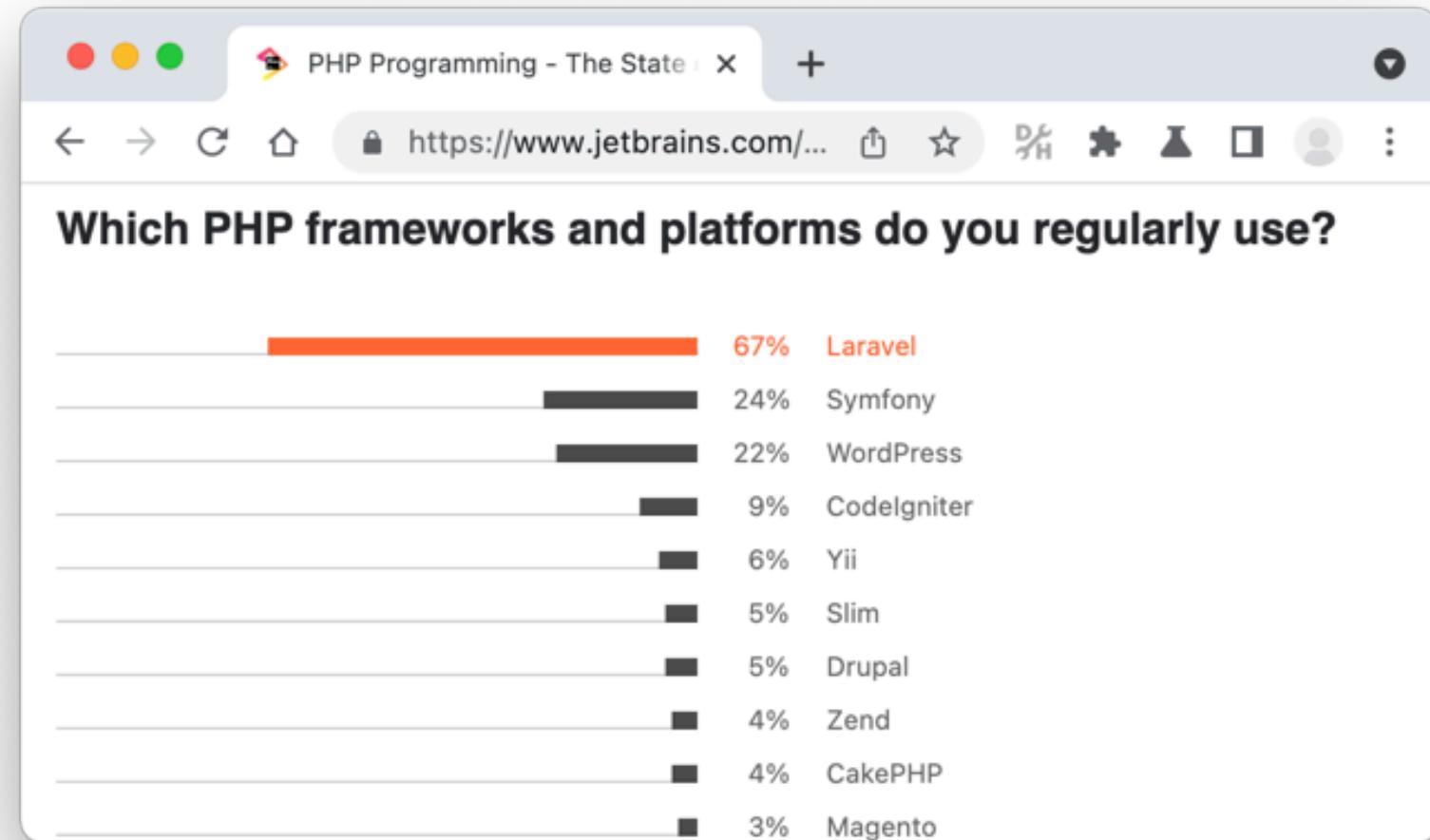
**How programming languages can help**



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: "Yes" (grey), "No" (orange), and "I don't know what static analysis is" (black).

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 Details:**
  - 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)**



```
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
```

## 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    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 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 property `$value` and two methods: `__construct` and `__toString`. The `__toString` method returns the value of `$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?



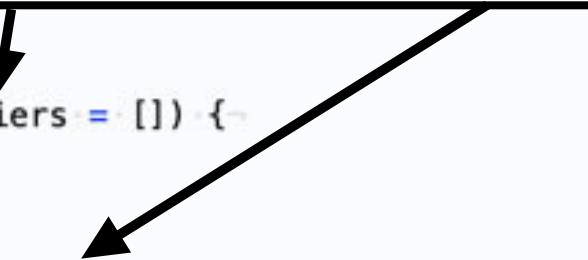
```
1  <?php
2
3  ▼ class db {
4
5      // ...
6
7  ▼     function query($sql, $parameters = [], $identifiers = []) {
8
9      $this->literal_check($sql);
10
11     foreach ($identifiers as $name => $value) {
12         if (!preg_match('/^[\w\-\_]+$/i', $name)) {
13             throw new Exception('Invalid 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 "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 log file. The text is mostly black on a white background, with some red and blue highlights. The terminal window has a dark border and a title bar at the top.

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!