

PHP Aspisp: Using Partial Taint Tracking To Protect Against Injection Attacks

Ioannis Papagiannis, Matteo Migliavacca, Peter Pietzuch
Department of Computing, Imperial College London

USENIX WebApps 2011
Portland, OR, USA

Injection Vulnerability Example



```
<?php
$name=$GET[ 'name' ] ;

$sql =
    "SELECT *
    FROM USERS
    WHERE user=" .
    $name ;

mysql_query ($sql) ;
?>
```



Injection Vulnerability Example

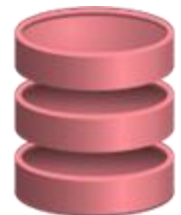
http://...?name=**Yiannis**



```
<?php
$name=$GET[ 'name' ] ;

$sql =
    "SELECT *
    FROM USERS
    WHERE user=" .
    $name ;

mysql_query($sql) ;
?>
```



Sanitisation

http://...?name=Yiannis



```
<?php
$name=$GET[ 'name' ] ;

$sql =
    "SELECT *
    FROM USERS
    WHERE user=" .
    sanitise_sql($name) ;

mysql_query($sql) ;
?>
```



Taint Tracking

taint data in entry points

1

propagate taint

2

use taint to guide sanitisation

3

```
<?php
$name=$GET[ 'name' ] ;

$sql =
    "SELECT *
    FROM USERS
    WHERE user=" .
    $name ;

mysql_query($sql) ;
?>
```

Taint Tracking in PHP

No support

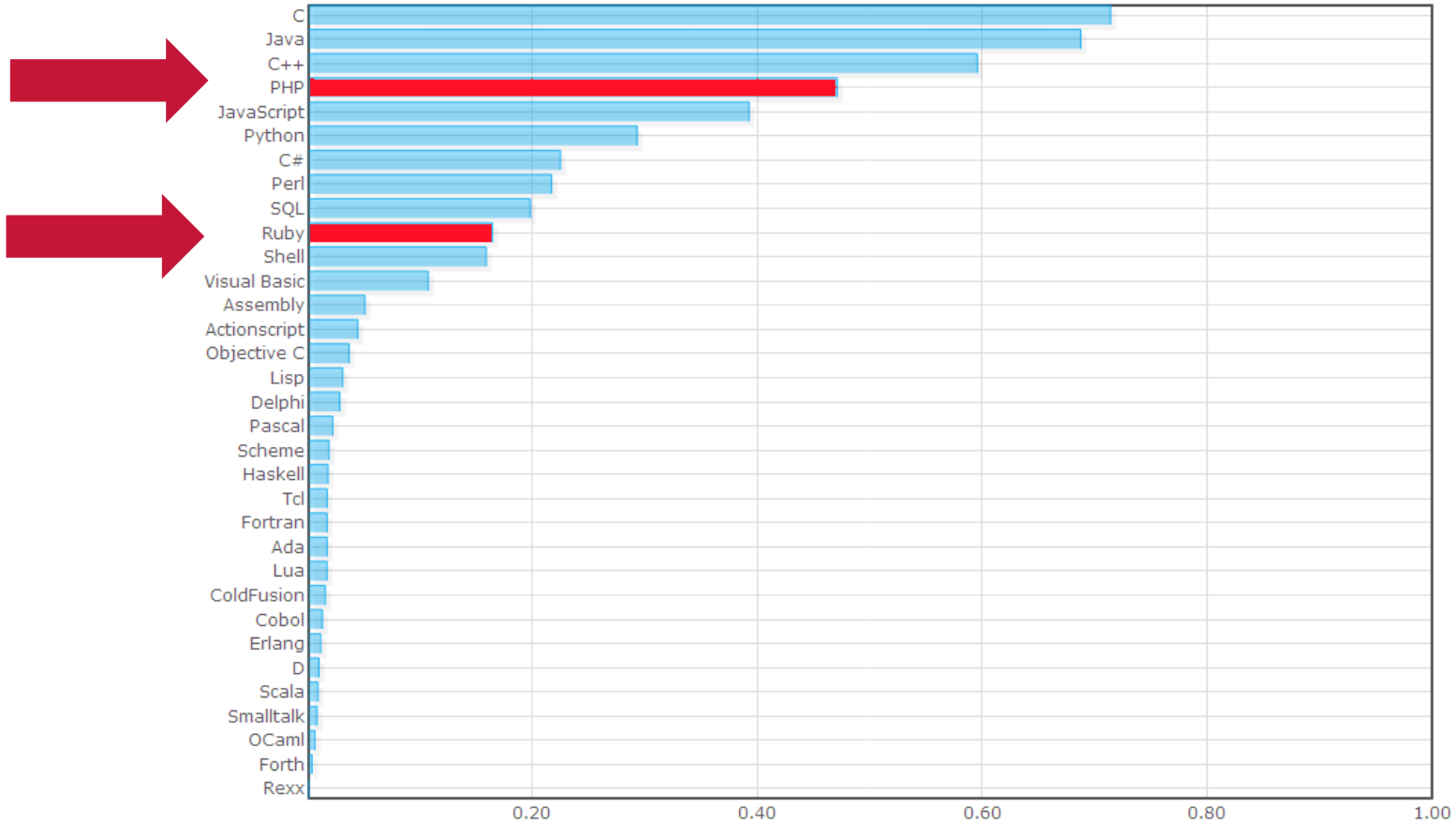
1. Suggested but ignored
[Venema06]
2. Custom research interpreters
[Yip09, Pietraszek06]

“modifications to the Zend engine should be avoided. Changes here result in incompatibilities with the rest of the world, and hardly anyone will ever adapt to specially patched Zend engines. ... Therefore, this method is generally considered bad practice”

The PHP Manual



PHP is popular



data provided by langpop.com

PHP Aspisp Contributions

1

Source-to-source transformations

2

Partial Taint Tracking



PHP Aspisp Contributions

1

Source-to-source transformations

2

Partial Taint Tracking



1 Why source transformations?

- ~~Adopt officially~~
- ~~Custom Runtime~~
- Source code transformations
 - ✓ On demand
 - ✓ Portable

1 Why source transformations?

- ~~Adopt officially~~
- ~~Custom Runtime~~
- Source code transformations
 - ✓ On demand
 - ✓ Portable

Challenges:

1. Not everything is an object
(how can you attach taint to strings?)
2. The interpreter cannot be edited
(no metaprogramming)



2 What is the performance overhead?

Partial taint tracking: Code is not equally vulnerable



Third-party plugin code

year	WordPress	WordPress Plugins
2009	2	13
2010	2	10

CVE WordPress-Platform Injection Vulnerabilities



Code that handles user data

CVE #	Functionality
2009-2851	Display user comments
2009-3891	File upload handler
2010-4257	Trackback handling
2010-4536	Display user comments

CVE WordPress-Core Injection Vulnerabilities

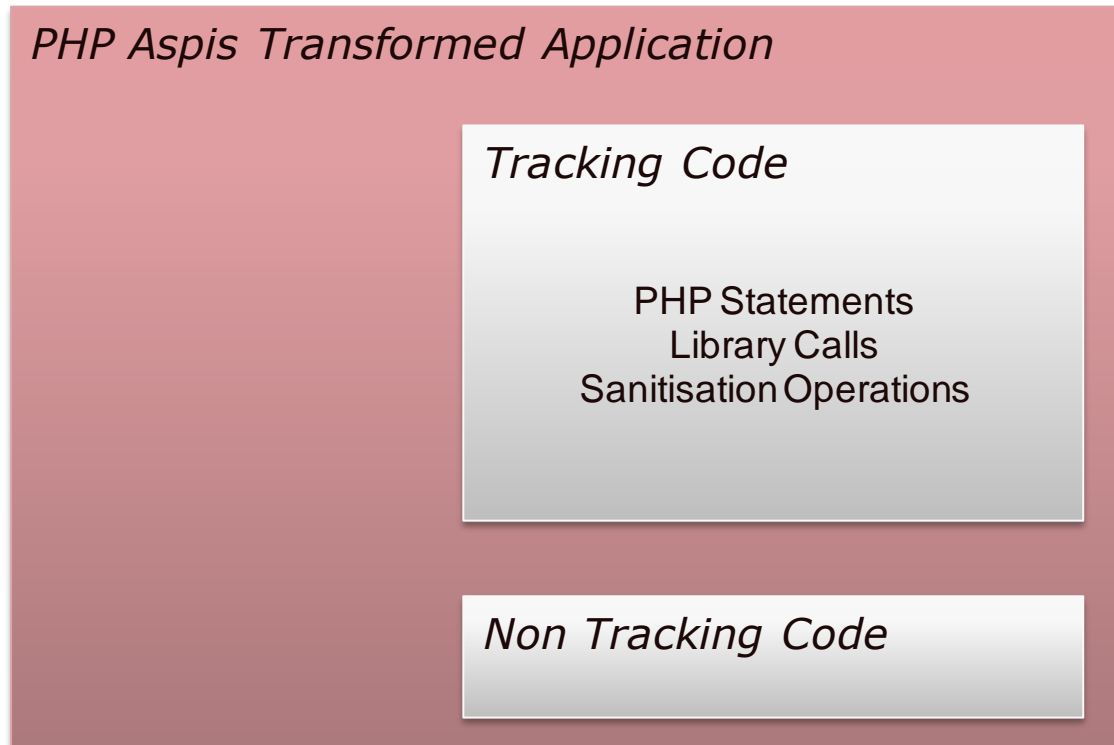
1. Introduction

2. DESIGN

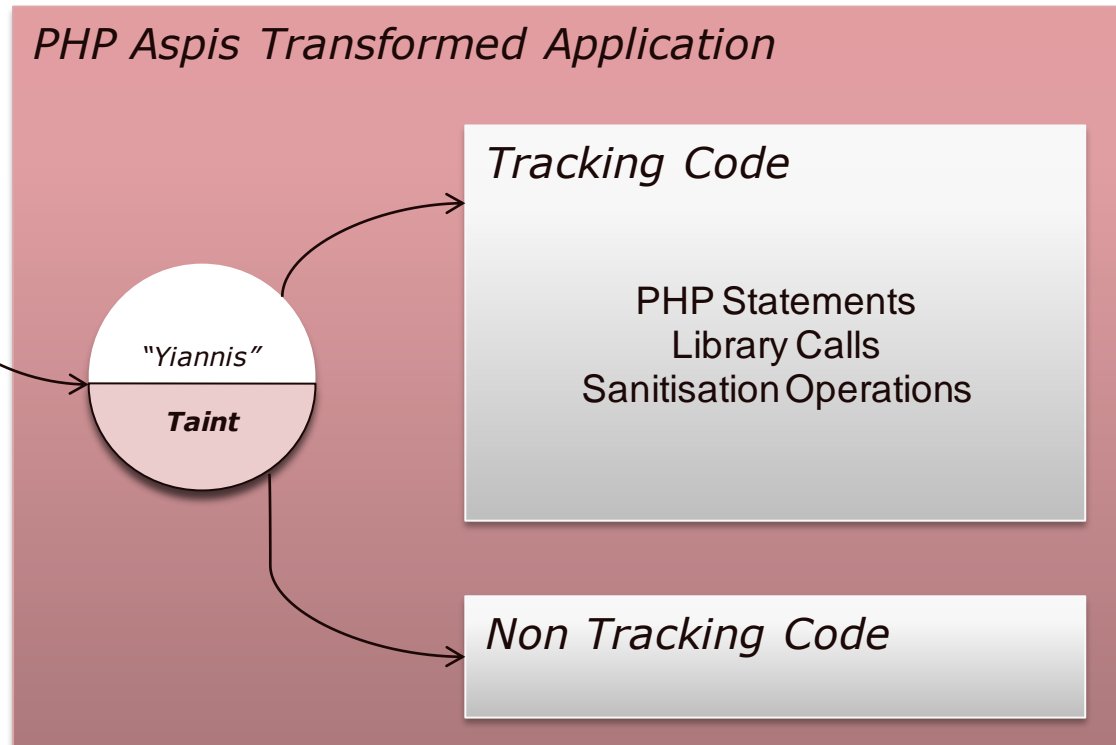
3. Implementation

4. Evaluation

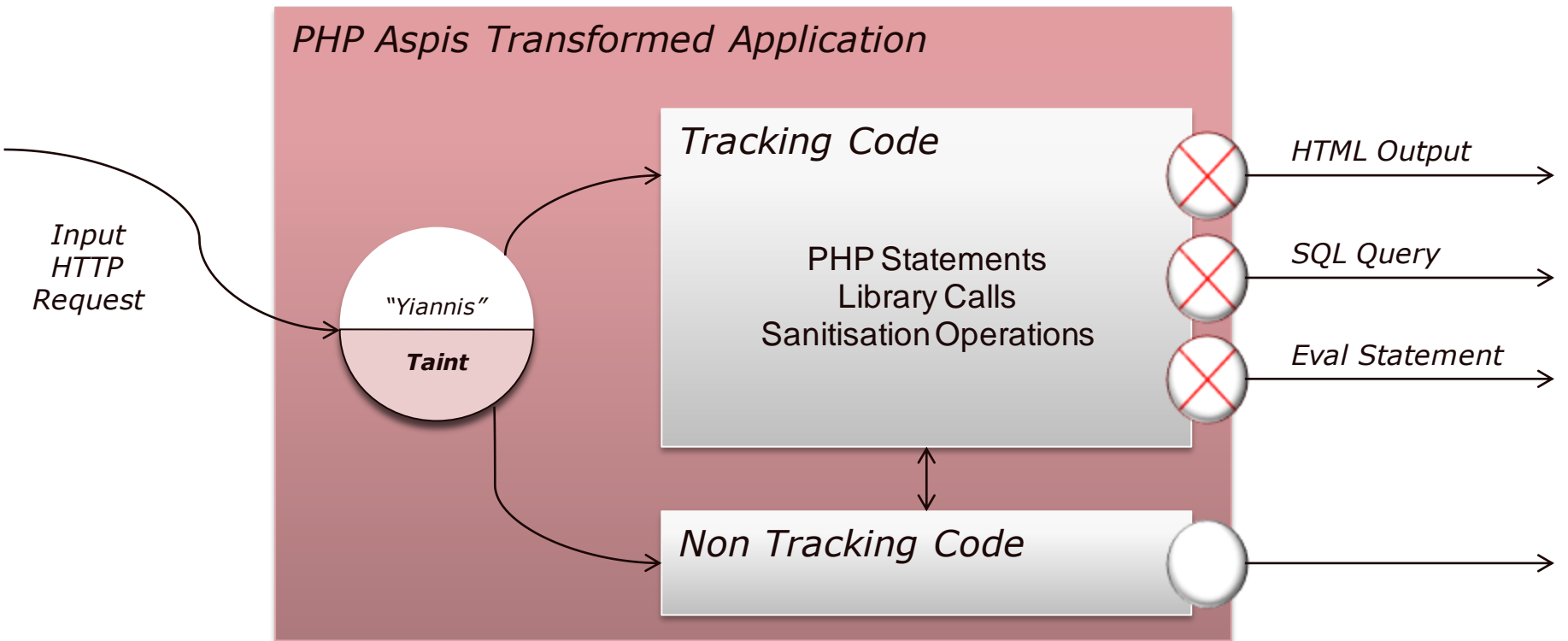
PHP Aspisp Overview



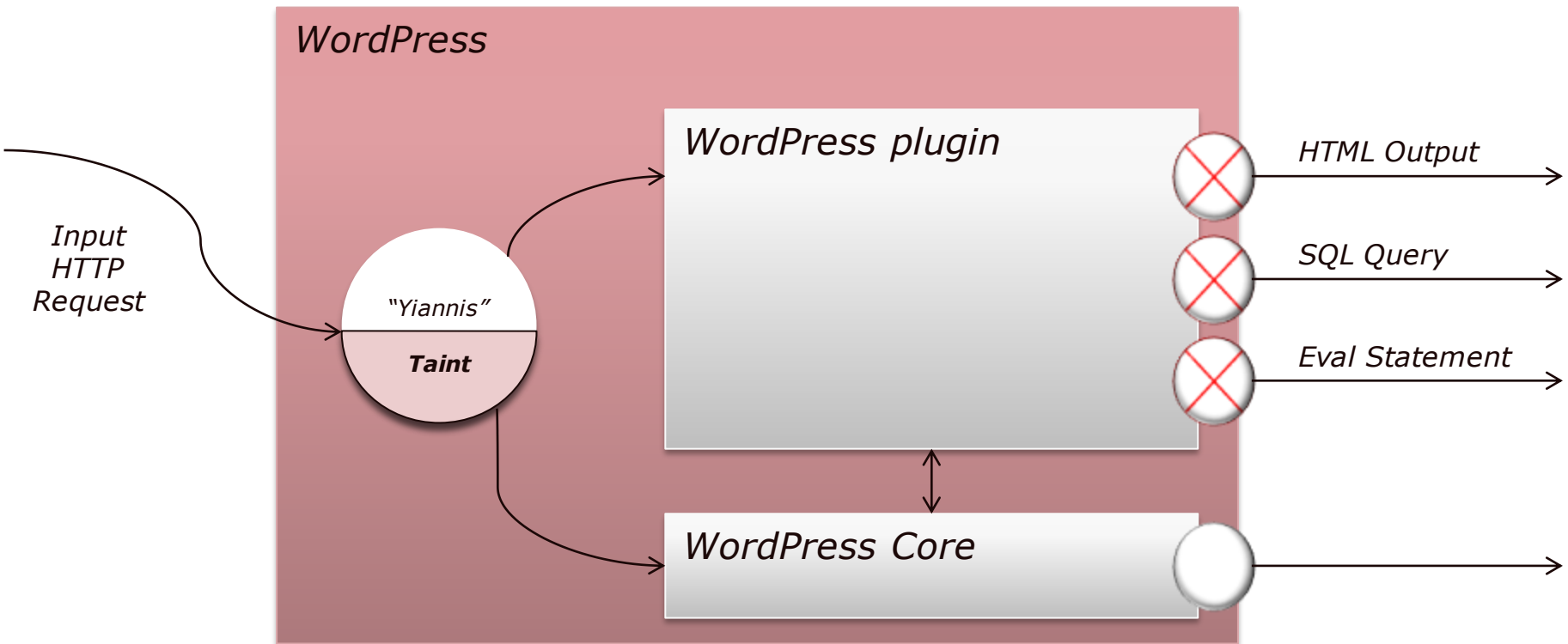
PHP Aspisp Overview



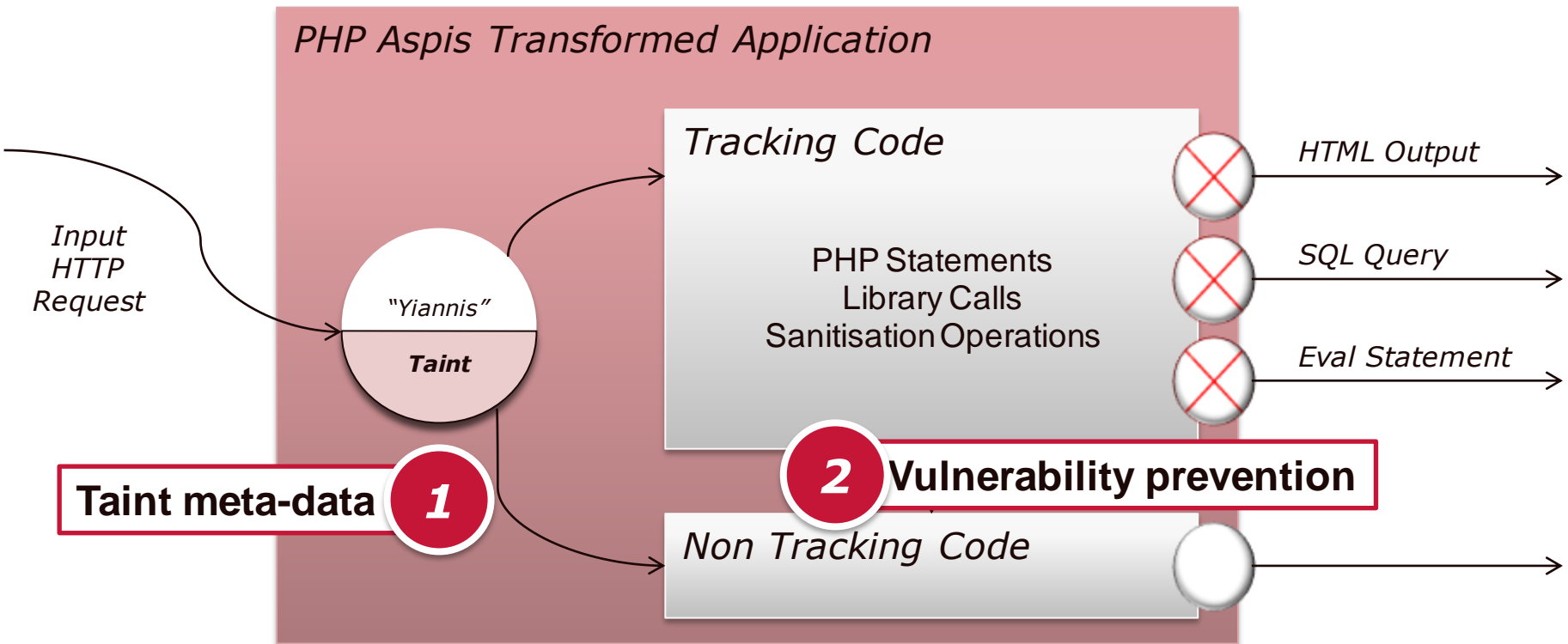
PHP Aspisp Overview



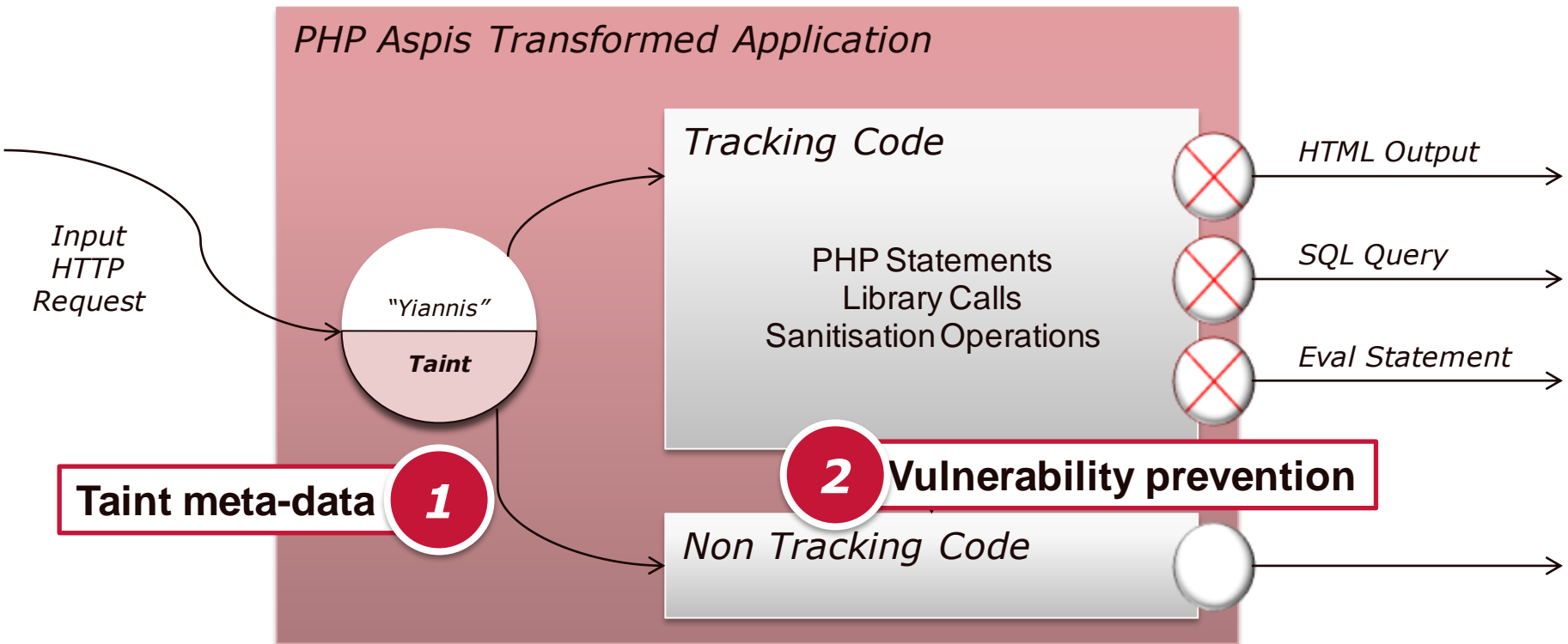
PHP Aspisp Overview



PHP Aspisp Overview



PHP Aspisp Overview



1 Taint Meta-data

```
"SELECT *  
FROM USERS  
WHERE user=yiannis";
```

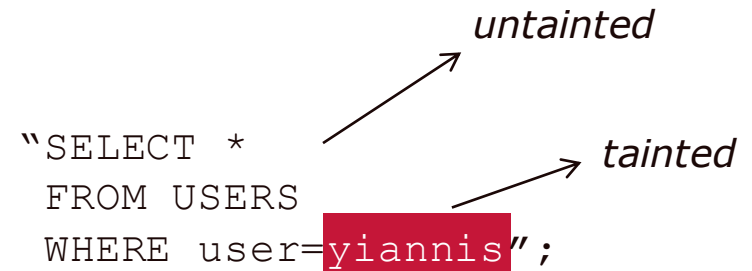


tainted

✗ Variable Level

- Leads to false positives

```
"SELECT *  
FROM USERS  
WHERE user=yiannis";
```



untainted

tainted

✓ Character Level

- Precise information

1 Taint Meta-data

```
"SELECT *  
FROM USERS  
WHERE user=yiannis";
```

→ *tainted*

✗ Variable Level

- Leads to false positives

```
"SELECT *  
FROM USERS  
WHERE user=yiannis";
```

→ *untainted*
→ *tainted*

✓ Character Level

- Precise information

➡ Partial sanitisation (e.g. `sanitize_sql($name)`)

```
"SELECT *  
FROM USERS  
WHERE user=yiannis";
```

→ *untainted*
→ *untainted (for SQL Injection)*

➡ More than 1 bit of taint meta-data is required

1 Taint Categories

➔ Example

```
"SELECT *  
FROM USERS  
WHERE user=yiannis";
```

Taint Category		
SQL Injection	XSS	Eval Injection
untainted	untainted	untainted
	tainted	tainted

1 Taint Categories

➔ Example

```
"SELECT *
FROM USERS
WHERE user=yiannis";
```

Taint Category		
SQL Injection	XSS	Eval Injection
untainted	untainted	untainted
	tainted	tainted

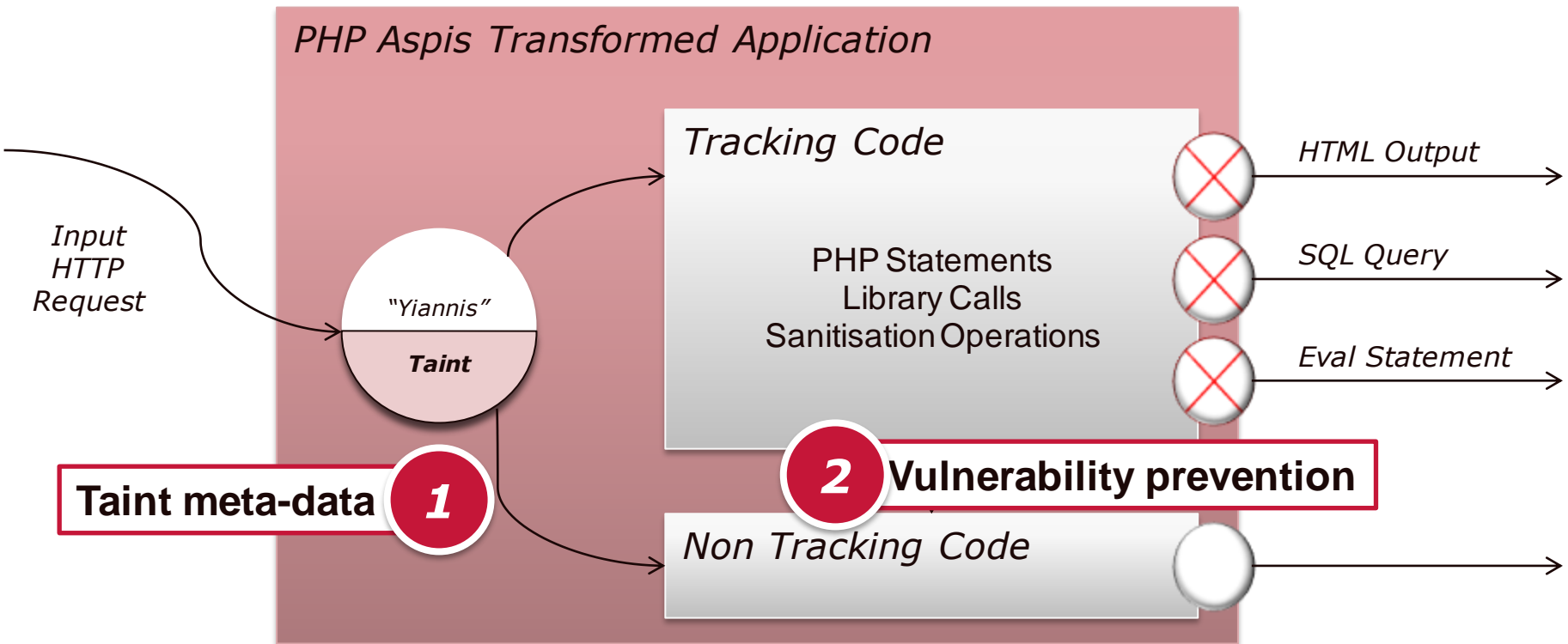
➔ Generic way to define:

- ✓ How an application *is supposed to* sanitise
- ✓ What to do if it doesn't

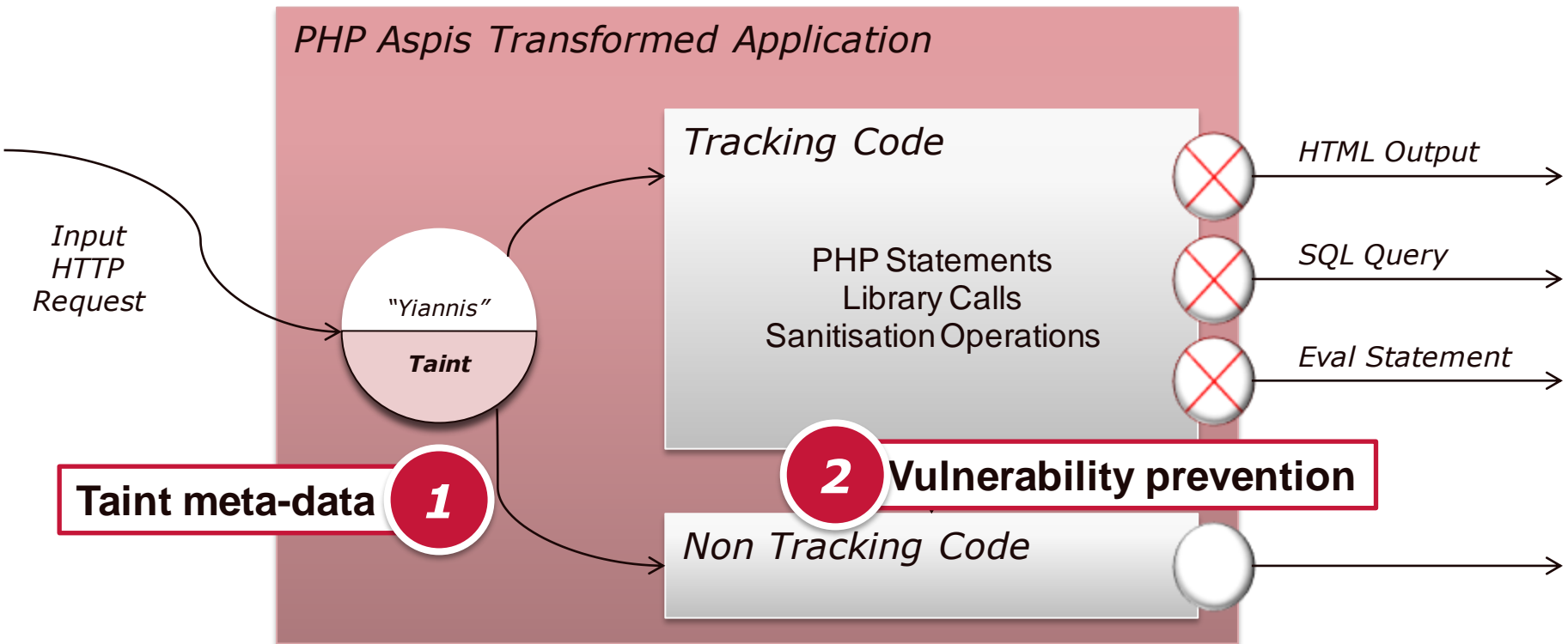
Sanitisation Functions	htmlentities() htmlspecialchars()
Guarded Sinks	echo()→AspisAntiXSS() print()→AspisAntiXSS() ...

XSS taint category excerpt

PHP Aspisp Overview



PHP Aspisp Overview



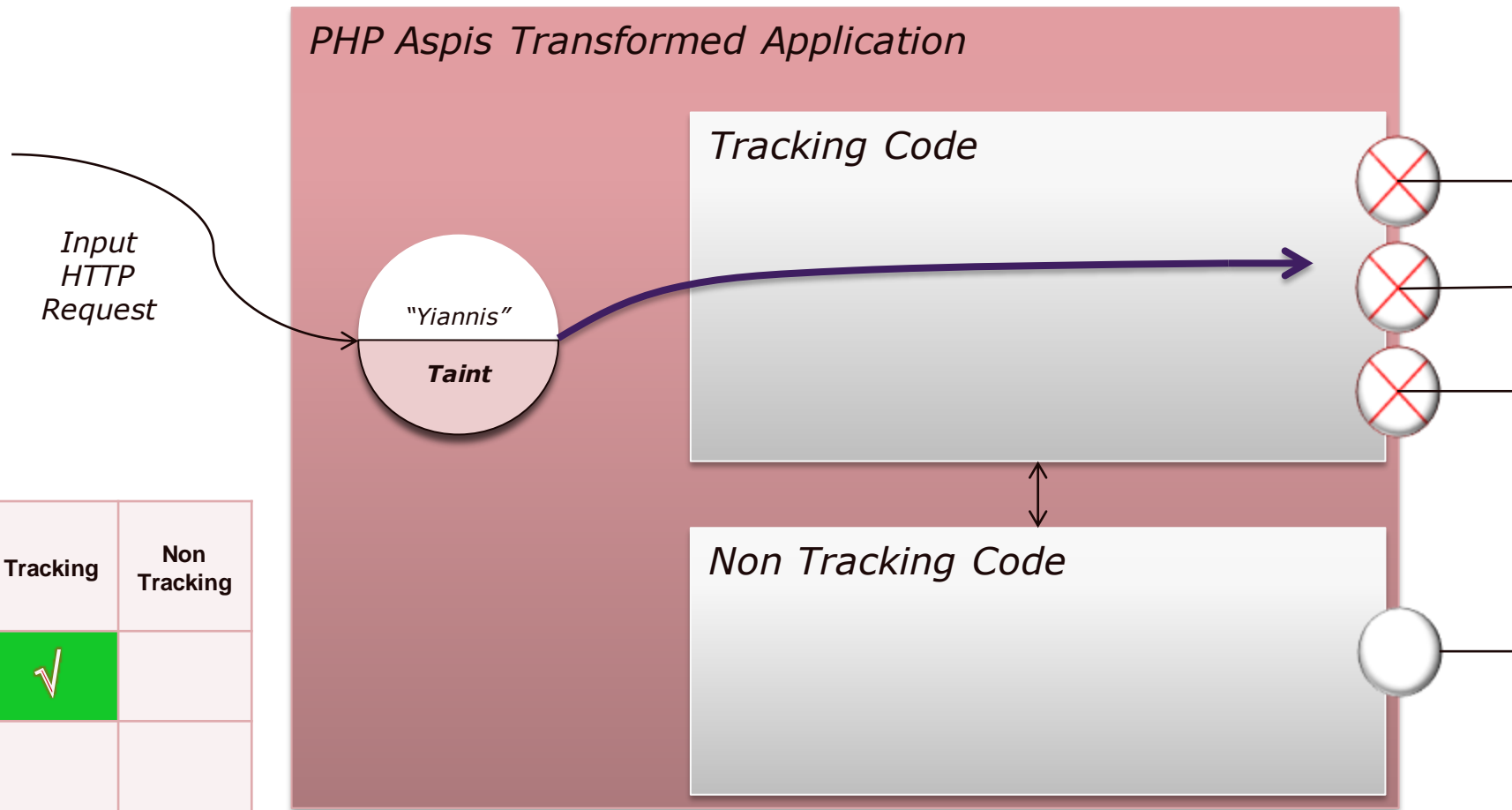
2

Which vulnerabilities can be prevented?

Possible Data Flows

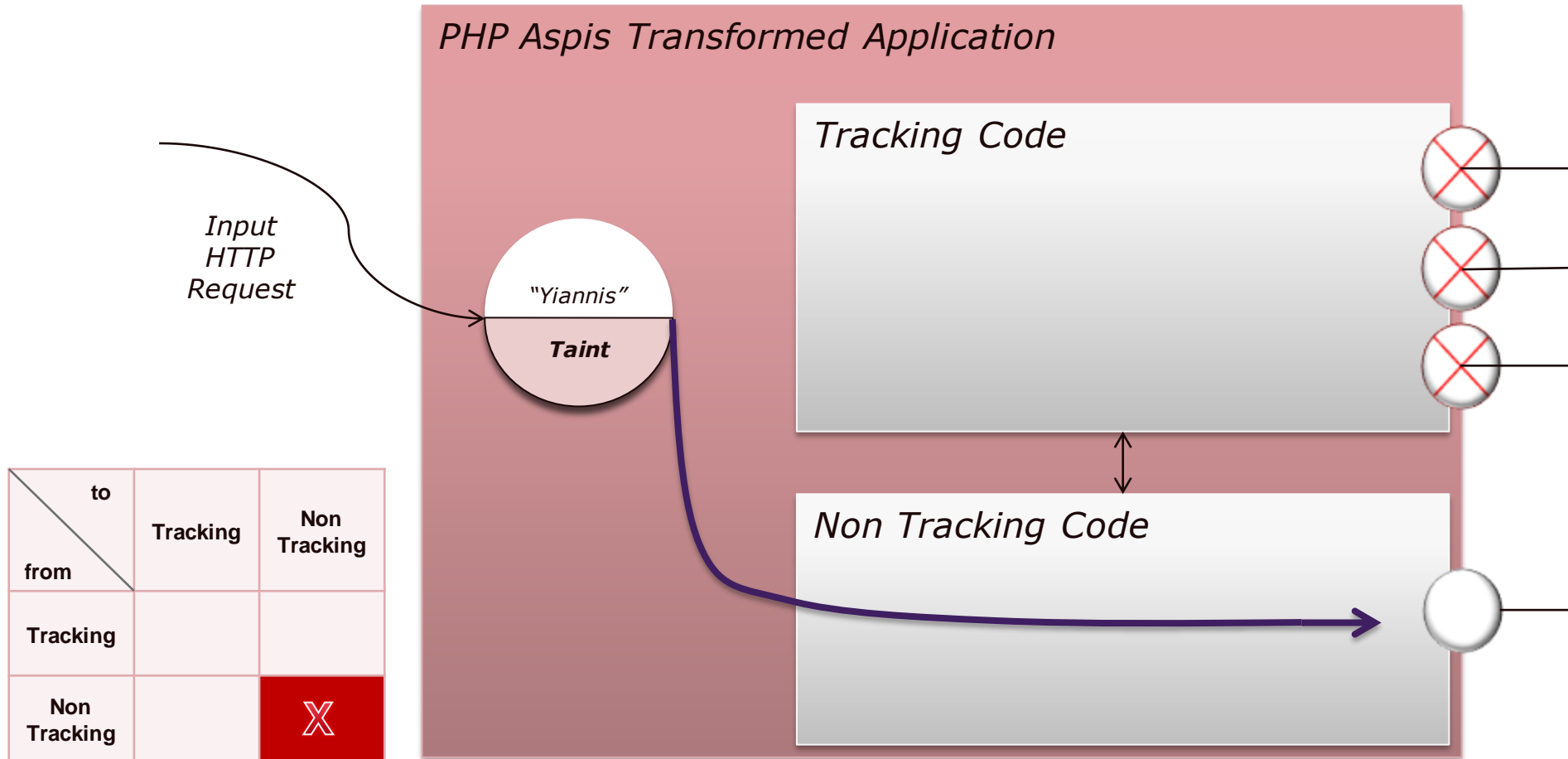
		to	
		Tracking	Non Tracking
from	Tracking	1 4 5	5
	Non Tracking	3	2

2 Tracking code only

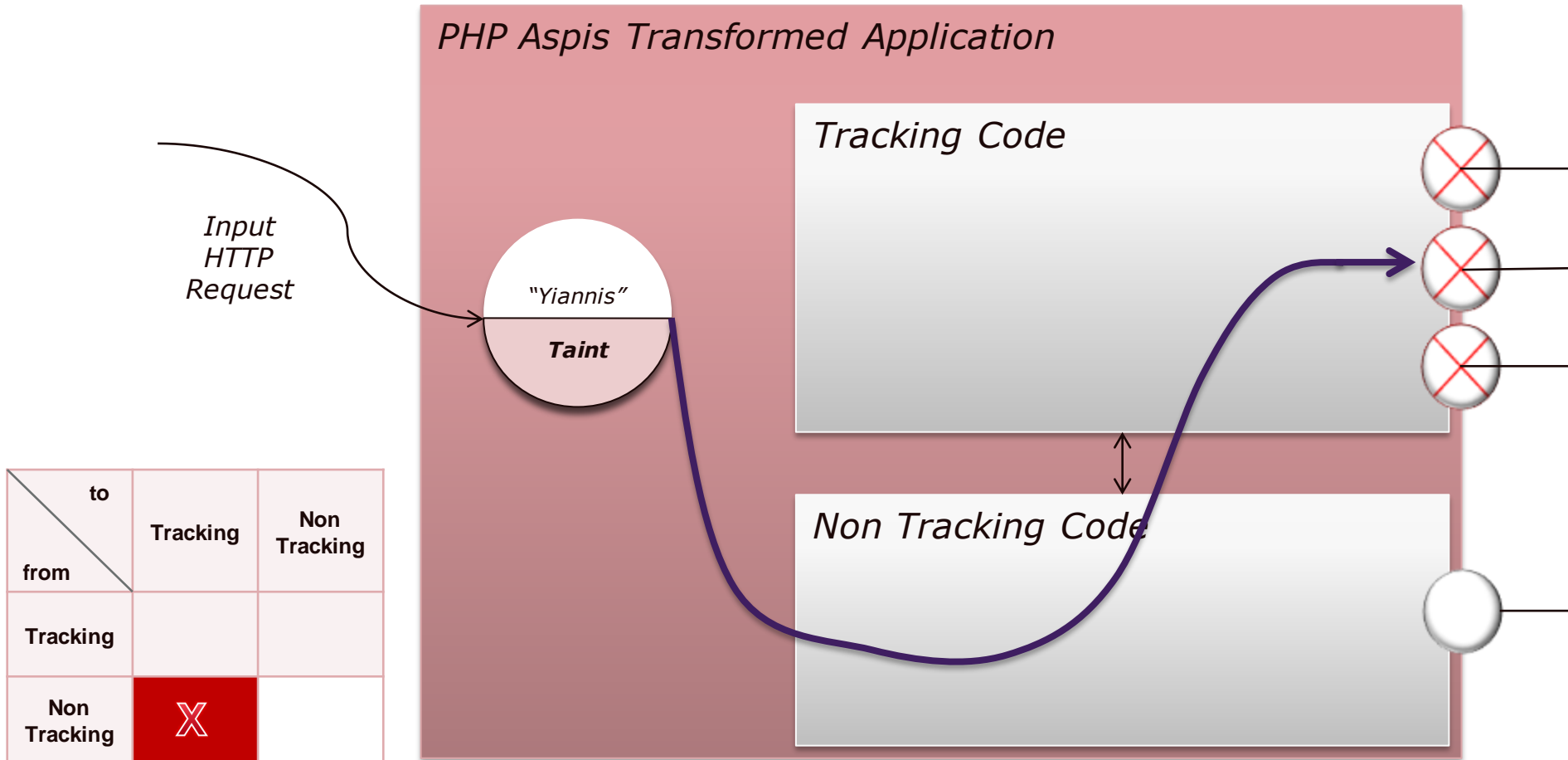


	to	Tracking	Non Tracking
from			
Tracking		✓	
Non Tracking			

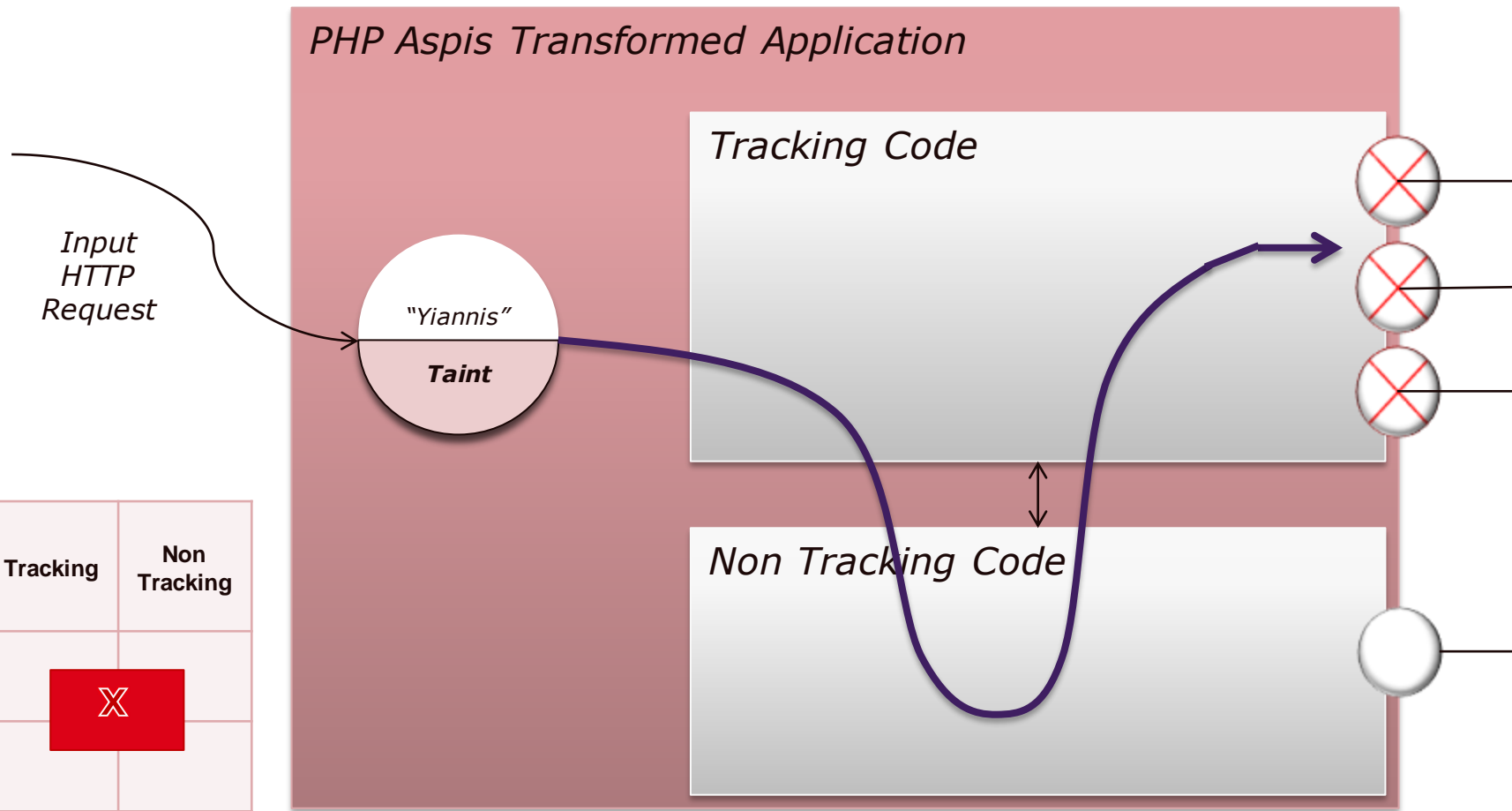
2 Non Tracking code only



2 Non Tracking to Tracking

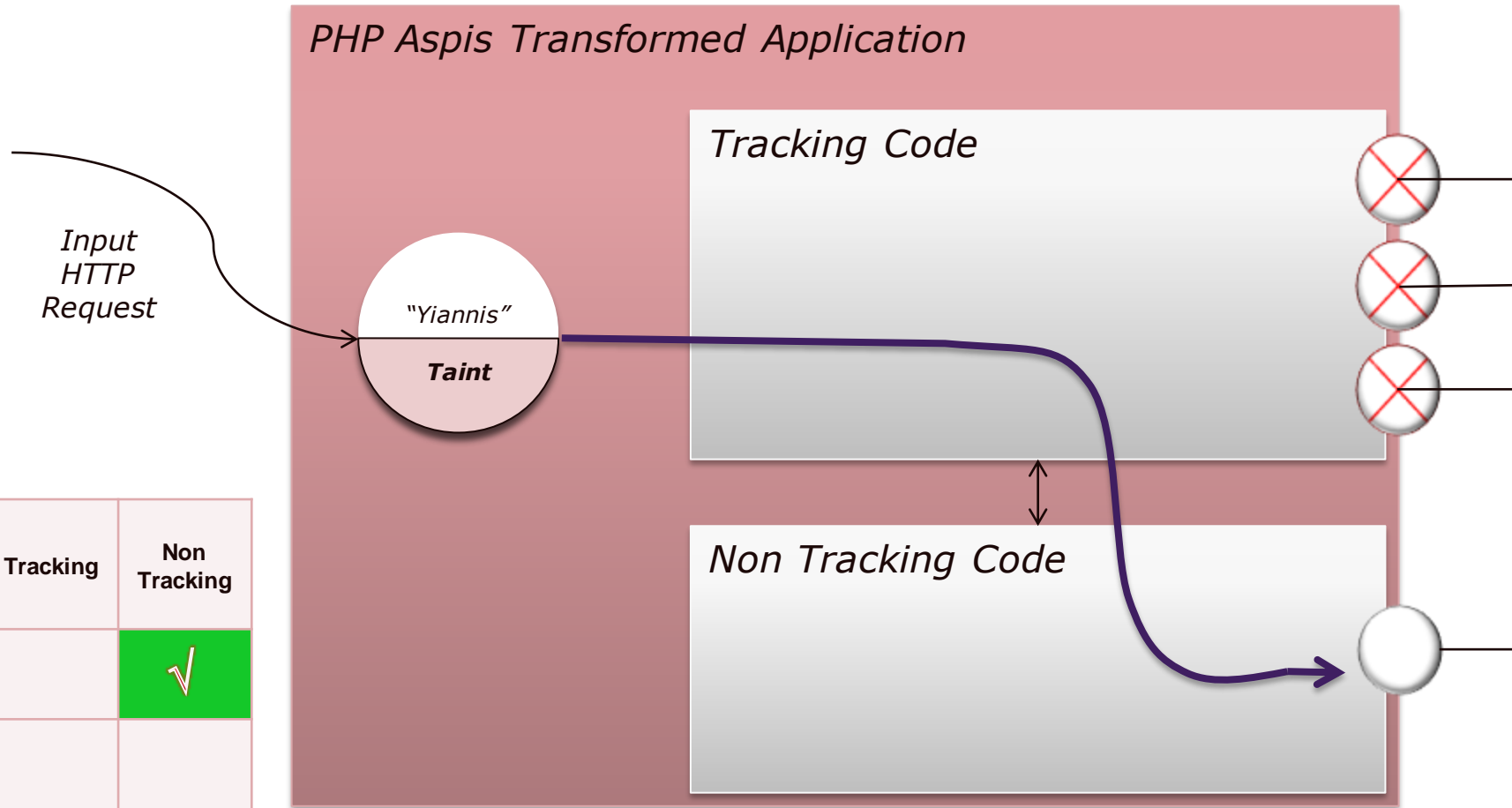


2 Tracking/Non Tracking mixes

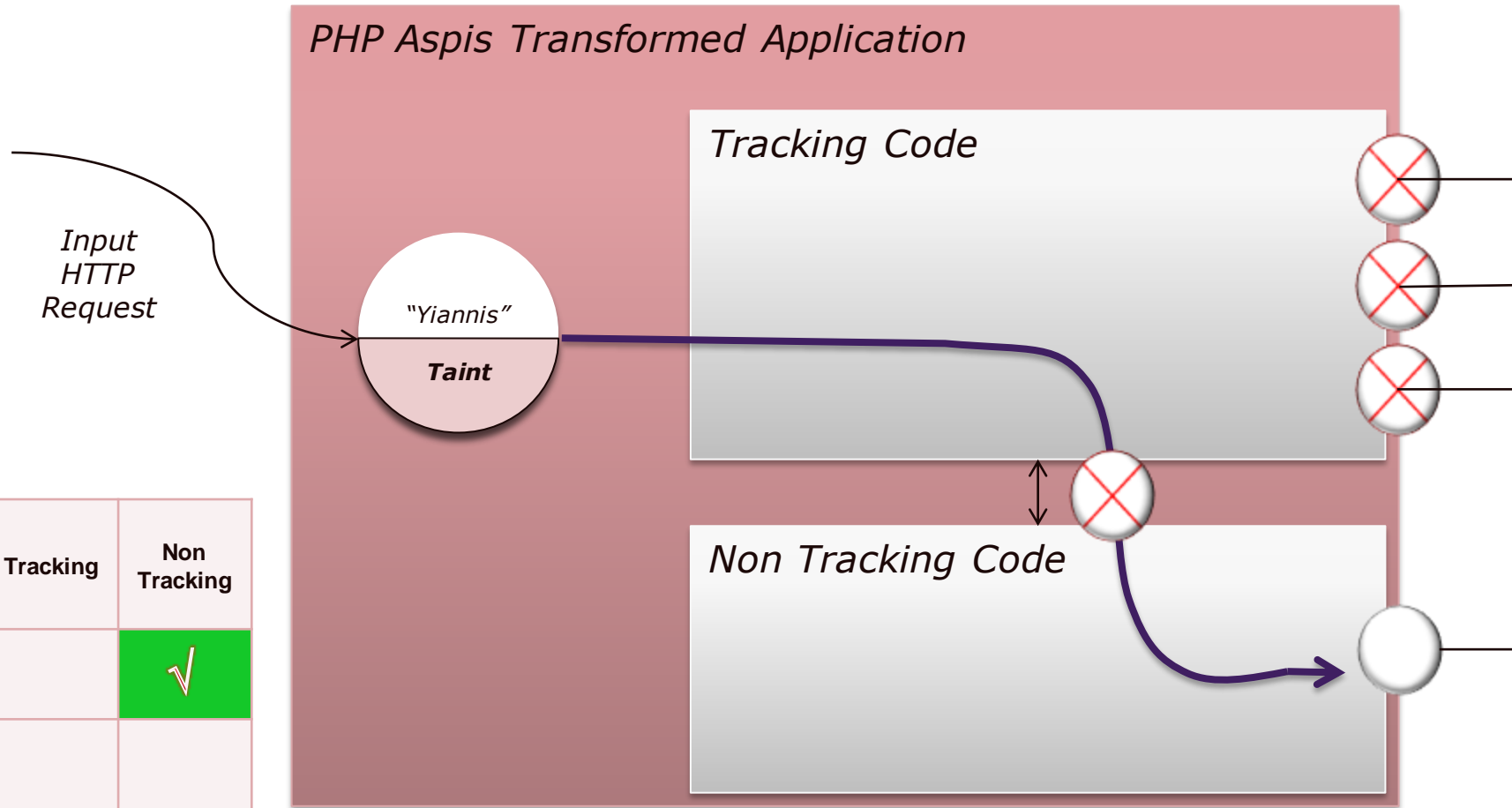


	to	
	Tracking	Non Tracking
from		
Tracking	X	
Non Tracking		

2 Tracking to Non Tracking



2 Tracking to Non Tracking



2 Summary: Prevented Vulnerabilities

➤ Vulnerabilities Prevented

- ✓ Tracking-code only
- ✓ Tracking to non tracking

to \ From	Tracking	Non Tracking
Tracking	1	5
Non Tracking		

➤ Vulnerabilities Not Prevented

- ✗ Non Tracking-code only
- ✗ Non Tracking to Tracking
- ✗ Tracking/Non Tracking mixes

to \ from	Tracking	Non Tracking
Tracking		
Non Tracking	3	2

1. Introduction

2. Design

3. IMPLEMENTATION

4. Evaluation

Storing Taint Meta-Data

Store taint *in place*

- ✓ Interoperation with non-tracking code

Use arrays

- ✓ 2-10x faster than object initialisation
- ✓ Scalar assignment semantics

<i>Original value</i>	<i>Aspis-protected value</i>
<code>"Hello"</code>	<pre>array ("Hello", TaintCats)</pre>
<code>12</code>	<pre>array (12, TaintCats)</pre>

Taint Tracking Transformations

Statements & Expressions must

1. operate with Aspis-protected values
2. propagate taint correctly
3. return Aspis-protected values

<i>Original expression</i>	<i>Transformed Expression</i>
<code>\$s.\$t</code>	<code>concat(\$s, \$t)</code>
<code>if (\$v) {}</code>	<code>if (\$v[0]) {}</code>
<code>\$j = \$i++</code>	<code>\$j = postincr(\$i)</code>

PHP Function Library

- ▶ Library functions do not work with Aspis-protected values
 - ✓ use interceptors!

Default Interceptor

- ✓ *strip input taint*
- ✓ *add empty output taint*
- ✓ *good as the default*
- ➔ *fclose(), fopen()*

Custom Interceptors

- ✓ *guess the taint of the output*
 - ➔ *substr()*
- ✓ *reimplement the function*
 - ➔ *sort()*

- ▶ More custom interceptors, less false negatives
 - Default: drop taint, not abort the call
 - Support existing applications without developer intervention

The rest...

- Taint representation
- Expressions
- Statements
- PHP Library Function Calls
- Taint initialisation
- Tracking/Non Tracking Calls
- Calls to sanitisation functions
- Calls to sinks
- Variable variables
- Variable function calls
- Include Statements
- Dynamic code generation

1. Introduction

2. Design

3. Implementation

4. EVALUATION

Goals

1 Is PHP Aspisp **effective** in preventing injection attacks?

2 What is the **performance overhead**?

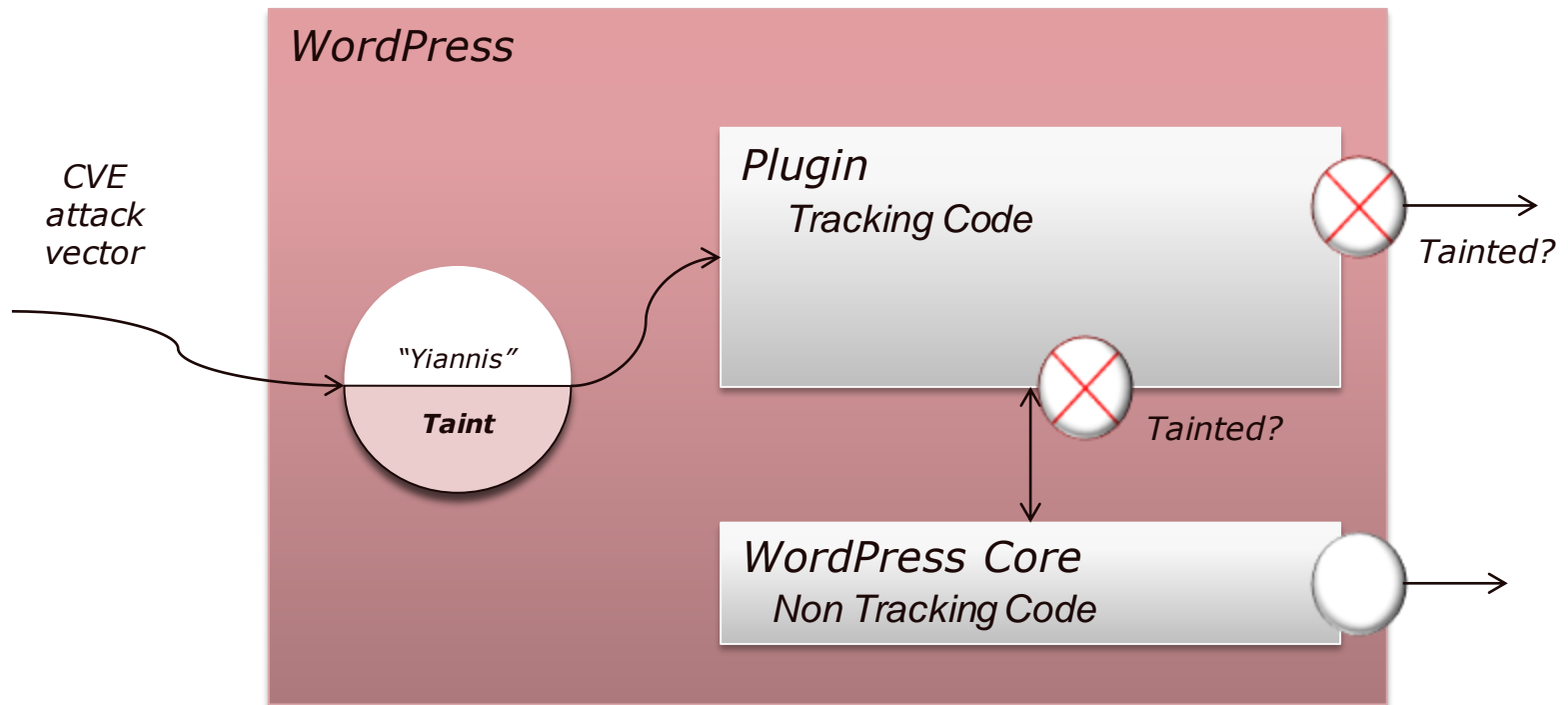
Goals

1 Is PHP Aspis **effective** in preventing injection attacks?

2 What is the **performance overhead**?

1 Evaluation methodology

- Use all reported WordPress plugins to the CVE since 1/1/2010
- Replay the provided attack vectors where possible

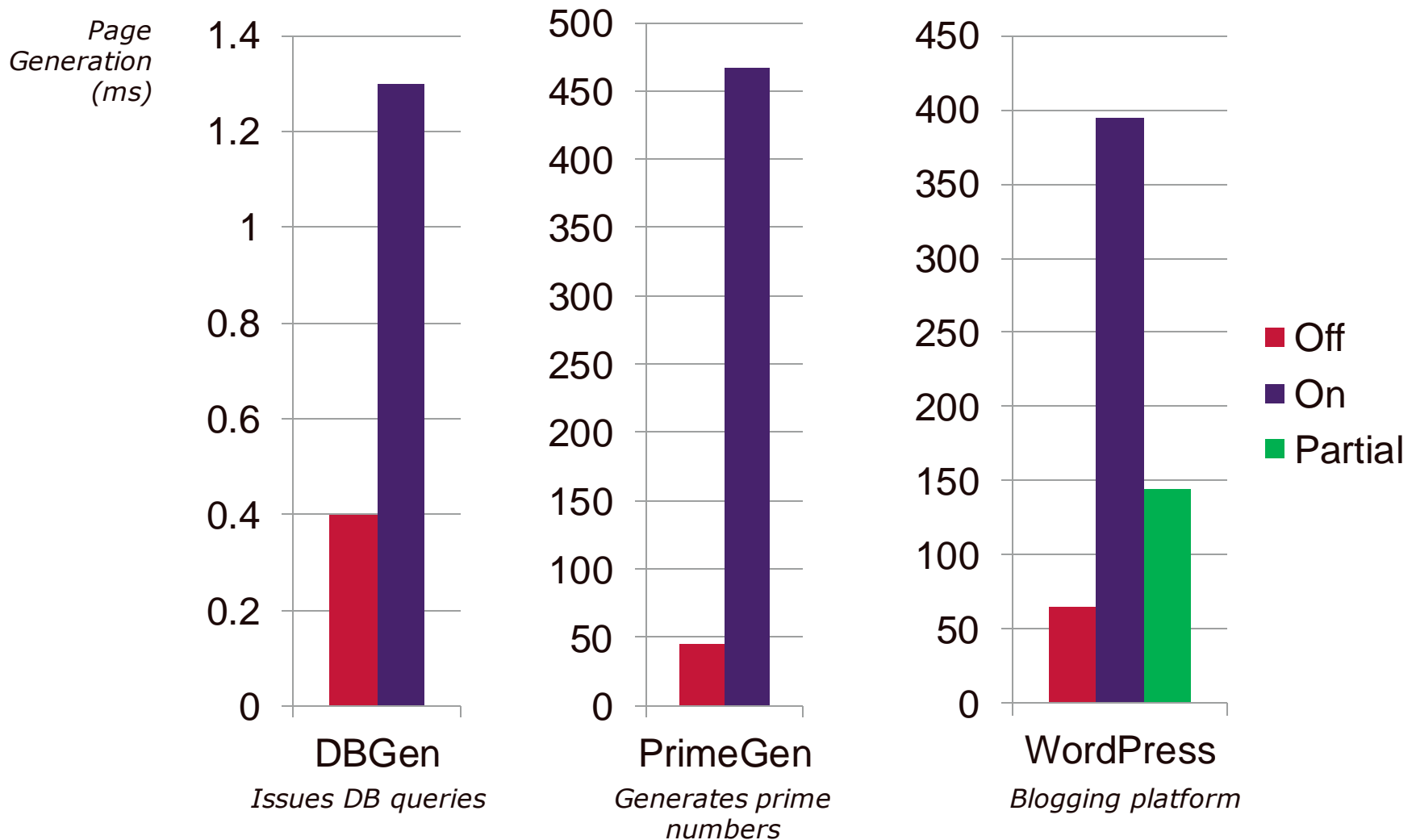


1

WordPress Plugins Results

- ➔ Total plugins with injection vulnerabilities: **15**
 - ➔ Tested plugins: **14**
 - 1 plugin was not publicly available
 - ➔ Vulnerabilities prevented: **12**
 - 10 XSS cases
 - 2 SQL Injection cases
 - ➔ Vulnerabilities not prevented: **2 (false negatives)**
 - Stored XSS attacks
 - PHP Aspisp does not track taint in the database
- ✓ No observed false positives

2 PHP Aspisp Overhead



Conclusion

- ➔ Partial taint tracking
 - Can be applied by **source code transformations**
 - Is suitable for applications that support **plugins**
 - Is **effective** for real world plugin vulnerabilities
- ➔ PHP Aspis design favours **false negatives**
 - False negatives are a consequence of partial taint tracking
 - False positives break existing applications
- ➔ “Reasonable” performance overhead
 - Suitable for deployments where **security is more important than performance**
 - Partial taint tracking for the WordPress case: 2.2x slowdown

The End



Ioannis Papagiannis

DoC, Imperial College London

```
$git clone git://github.com/jpapayan/aspis.git
```

ip108@doc.ic.ac.uk

References

- [1] Venema, W. Runtime taint support proposal. In PHP Internals mailing list (2006)
- [2] Yip, A. Wang X., et all. Improving application security with data flow assertions. In SOSP 2009
- [3] Pietraszek, T and Berghe, C. Defending against injection attacks through context sensitive string evaluation. In Recent Advances in Intrusion Detection (2006)