

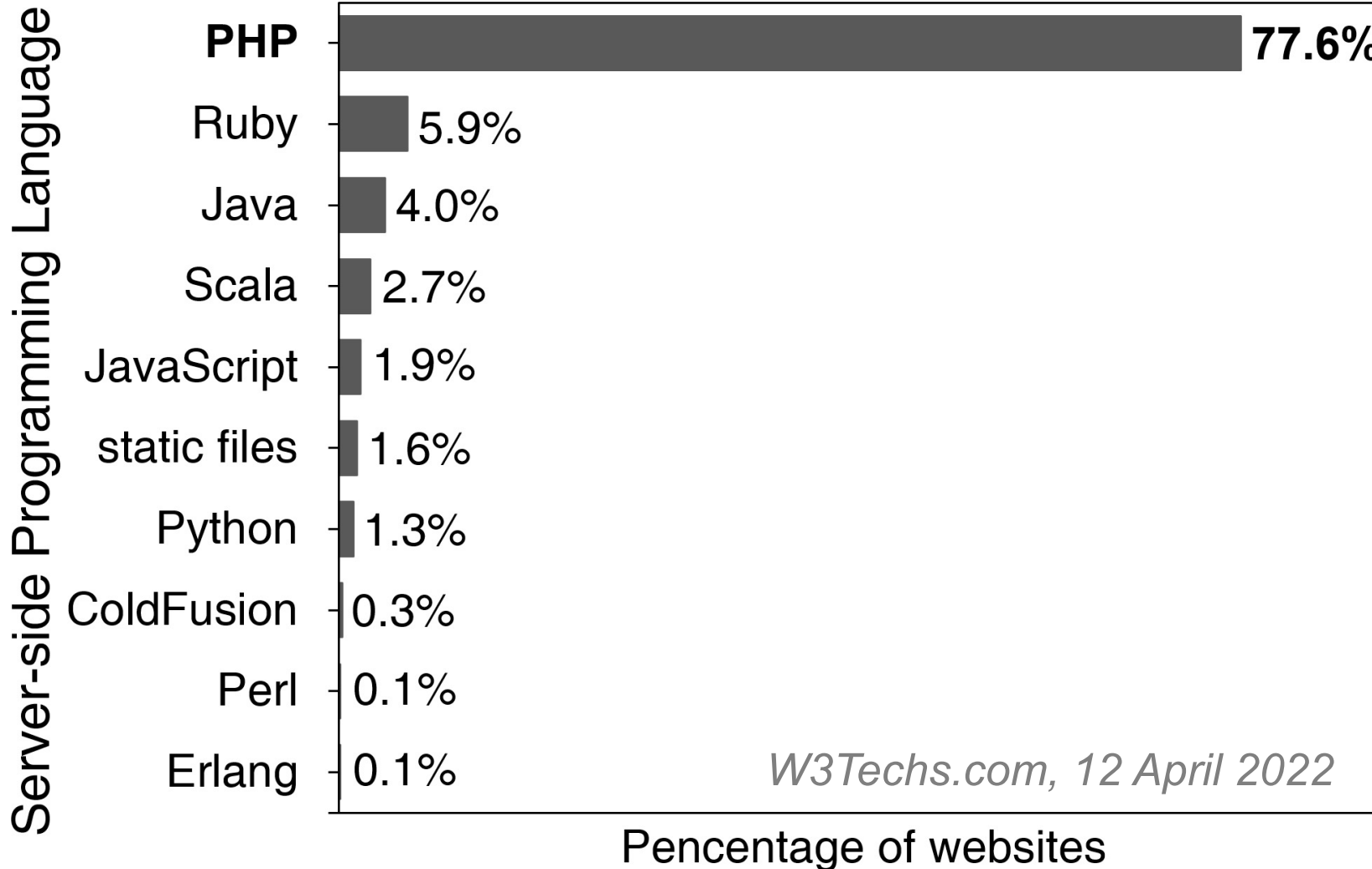
HiddenCPG: Large-Scale Vulnerable Clone Detection using Subgraph Isomorphism of Code Property Graphs

Seongil Wi, Sijae Woo, Joyce Jiyoung Whang, Sooel Son
KAIST

TheWebConf 2022

Popularity of PHP

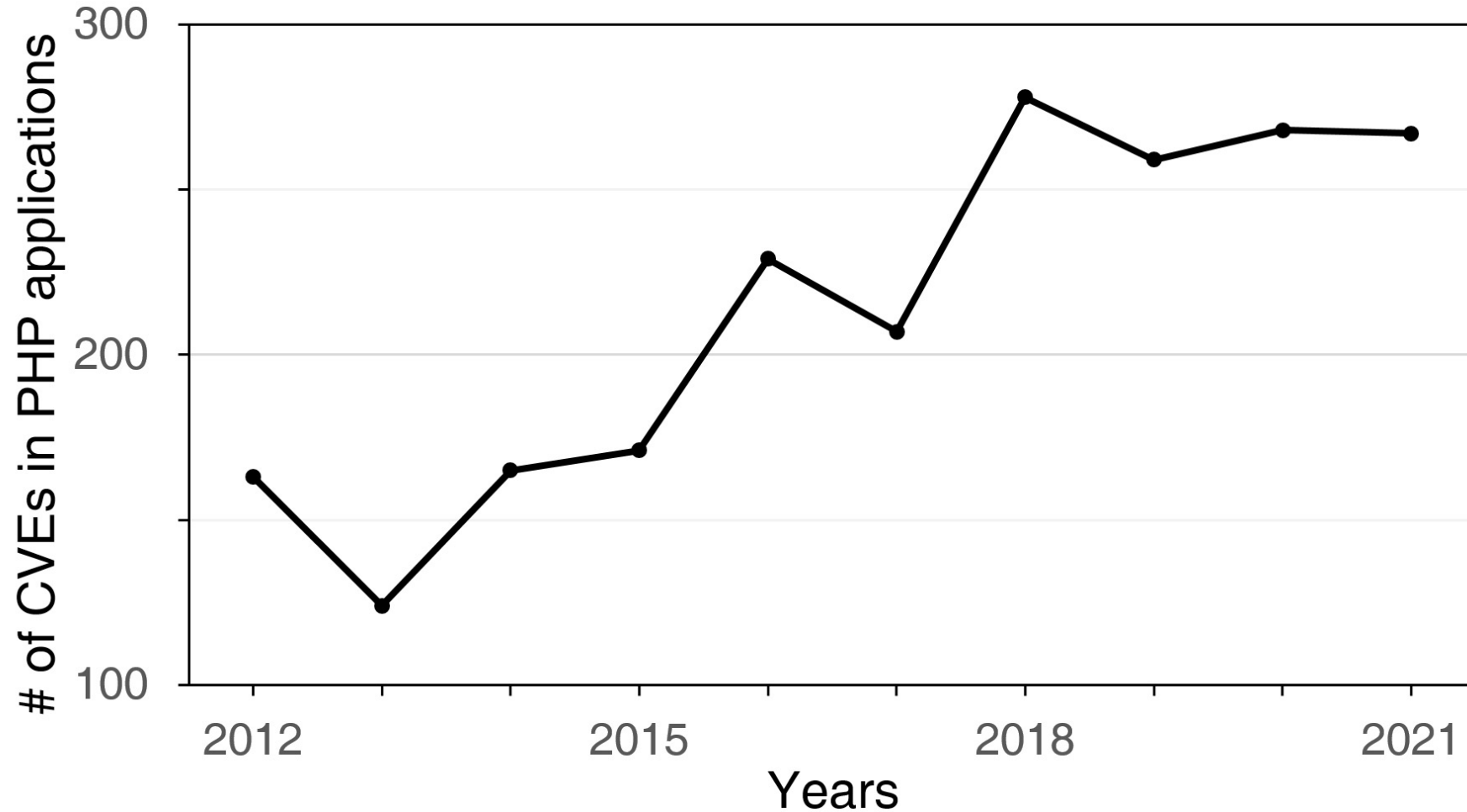
- PHP software has made tremendous progress



140K open-source projects!

Vulnerabilities in PHP Applications

- The number of bugs in PHP applications is increasing



How to Find Web Vulnerabilities?

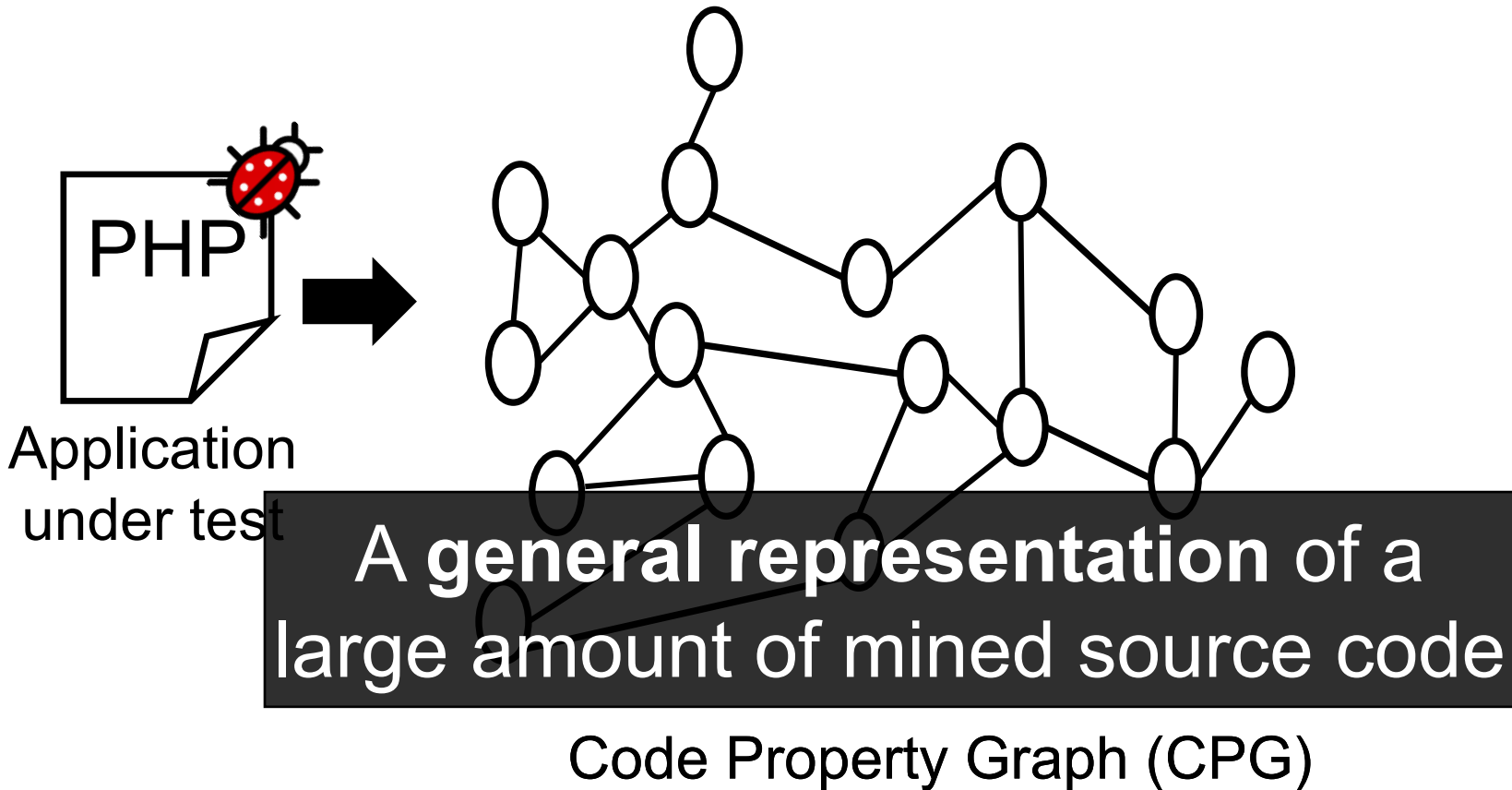
- Previous work – static analysis:
 - DoubleX, **CCS '21**
 - Lchecker, **WWW '21**
 - PHPJoern, **EuroS&P '17**
 - Saner, **Oakland '08**
 - Pixy, **Oakland '06**

How to Find Web Vulnerabilities?

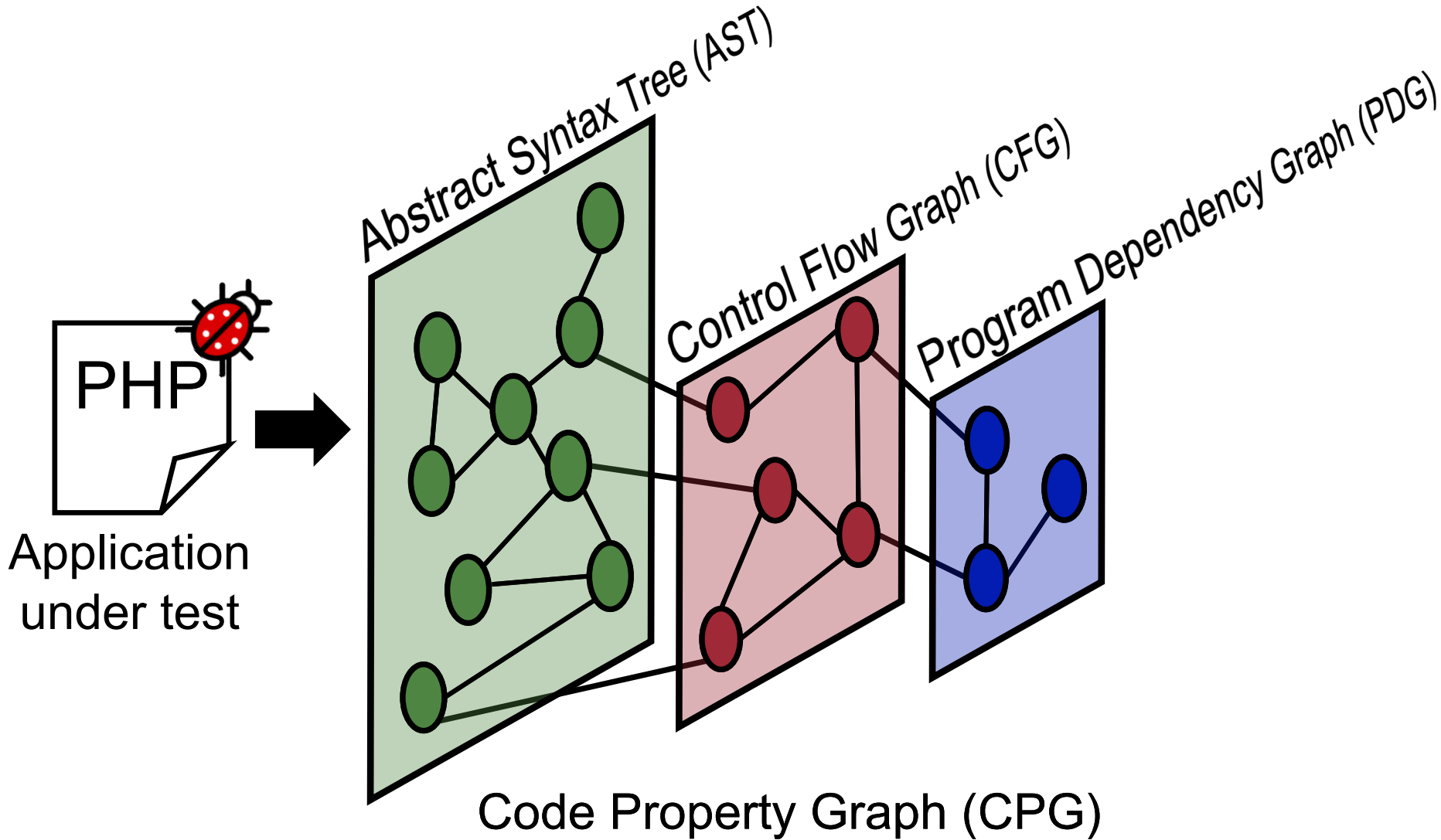
- Previous work – static analysis:
 - DoubleX, *CCS '21*
 - Lchecker, *WWW '21*
 - PHPJoern, ***EuroS&P '17***
 - Saner, *Oakland '08*
 - Pixy, *Oakland '06*

Scalable discovery of
common web vulnerabilities

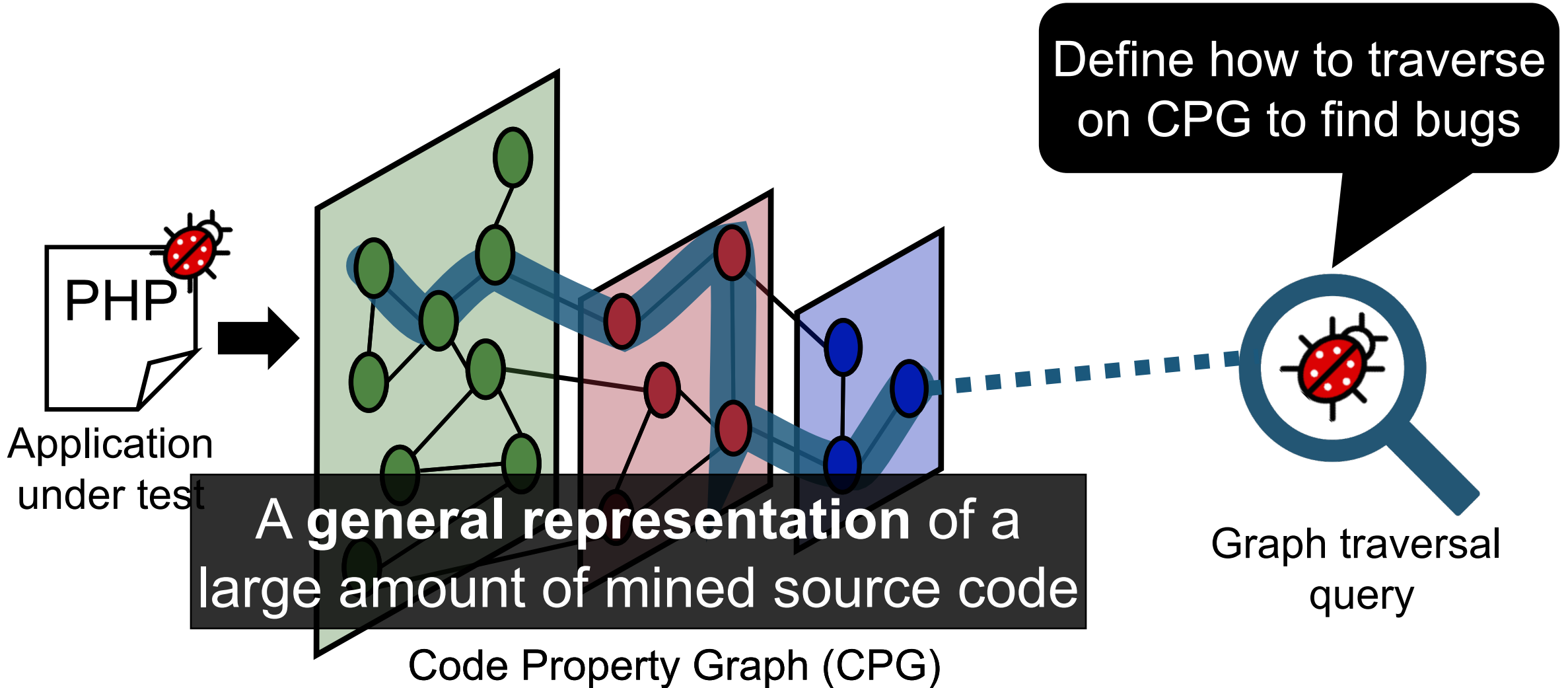
PHPJoern, *EuroS&P '17*



PHPJoern, *EuroS&P '17*



PHPJoern, *EuroS&P '17*



Define how to traverse on CPG to find bugs

Audit a large amount of code in a scalable way

Application under test

A general representation of a large amount of mined source code

Code Property Graph (CPG)

Graph traversal query

PHPJoern, *EuroS&P '17*

Define how to traverse on CPG to find bugs

From 1,854 GitHub projects, PHPJoern identified 196 bugs within 6 days and 13 hours

Application under test

A general representation of a large amount of mined source code

Code Property Graph (CPG)

Graph traversal query

PHPJoern, EuroS&P '17

```
<?php
    $input = $_GET["input"];
    $message = $input;
?>
<a href = “
    <?php echo $message; ?>
”> Content </a>
```



Searching XSS bugs

Define how to traverse
on CPG to find bugs



Graph query

PHPJoern, EuroS&P '17

```
<?php
    $input = $_GET["input"];
    $message = $input;
?>
<a href = "
    <?php echo $message; ?>
"> Content </a>
```



Searching XSS bugs

- (1) Searching sink functions
- (2) Identifying data flows



Graph query

ECHO

AST for
echo \$message;

PHPJoern, EuroS&P '17

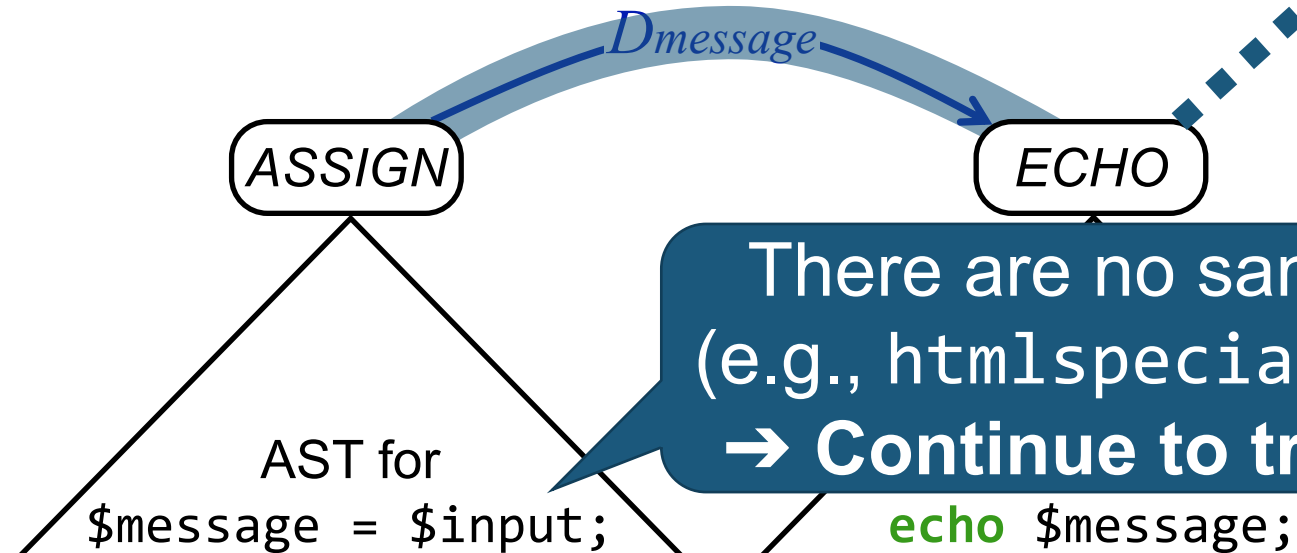
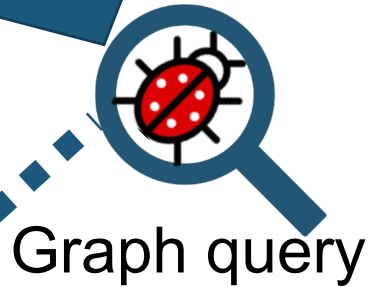
```
<?php
  $input = $_GET["input"];
  $message = $input;
?>
<a href = "
  <?php echo $message; ?>
"> Content </a>
```



Searching XSS bugs

- (1) Searching sink functions
- (2) Identifying data flows

Data flow edge



PHPJoern, EuroS&P '17

```
<?php
$input = $_GET["input"];
$message = $input;
?>
<a href = "
  <?php echo $message; ?>
"> Content </a>
```



Searching XSS bugs

- (1) Searching sink functions
- (2) Identifying data flows



Graph query



There is an input source

There are no sanitizers (e.g., htmlspecialchars) → Continue to traverse

AST for `$input=$_GET["input"];`

AST for `$message = $input;`

AST for `echo $message;`

PHPJoern, EuroS&P '17

```
<?php

$message = $input;
?>
<a href = "
  <?php echo $message; ?>
"> Content </a>
```



Searching XSS bugs

- (1) Searching sink functions
- (2) Identifying data flows



Graph query



There is an input source

There are no sanitizers (e.g., htmlspecialchars) → Continue to traverse

AST for `$input=$_GET["input"];`

AST for `$message = $input;`

AST for `echo $message;`

Limitation of PHPJoern

A query is designed to be coarse-grained



Graph query

Only check the existence of the sanitizations



There are no sanitizers
(e.g., htmlspecialchars)
→ Continue to traverse

Limitation of PHPJoern

A query is designed to be coarse-grained

Coarse-grained query can produce false negatives

There are no sanitizers (e.g., htmlspecialchars) → Continue to traverse

Incorrect Input Sanitizations

```
<?php
    $input = $_GET["input"];
    $message = $input;
?>
<a href = "
    <?php echo $message; ?>
"> Content </a>
```



Vulnerable code

Incorrect Input Sanitizations

```
<?php
$input = $_GET["input"];
$message = htmlspecialchars($input);
?>
<a href = "
    <?php echo $message; ?>
"> Content </a>
```



Incorrectly patched code

A query is designed to be coarse-grained



Graph query

*Only check the existence
of the sanitizations*

There is a sanitization
(i.e., htmlspecialchars)
→ Stop the traversal

Incorrect Input Sanitizations

```
<?php
$input = $_GET["input"];
$message = htmlspecialchars($input);
?>
<a href = "
    <?php echo $message; ?>
"> Content </a>
```



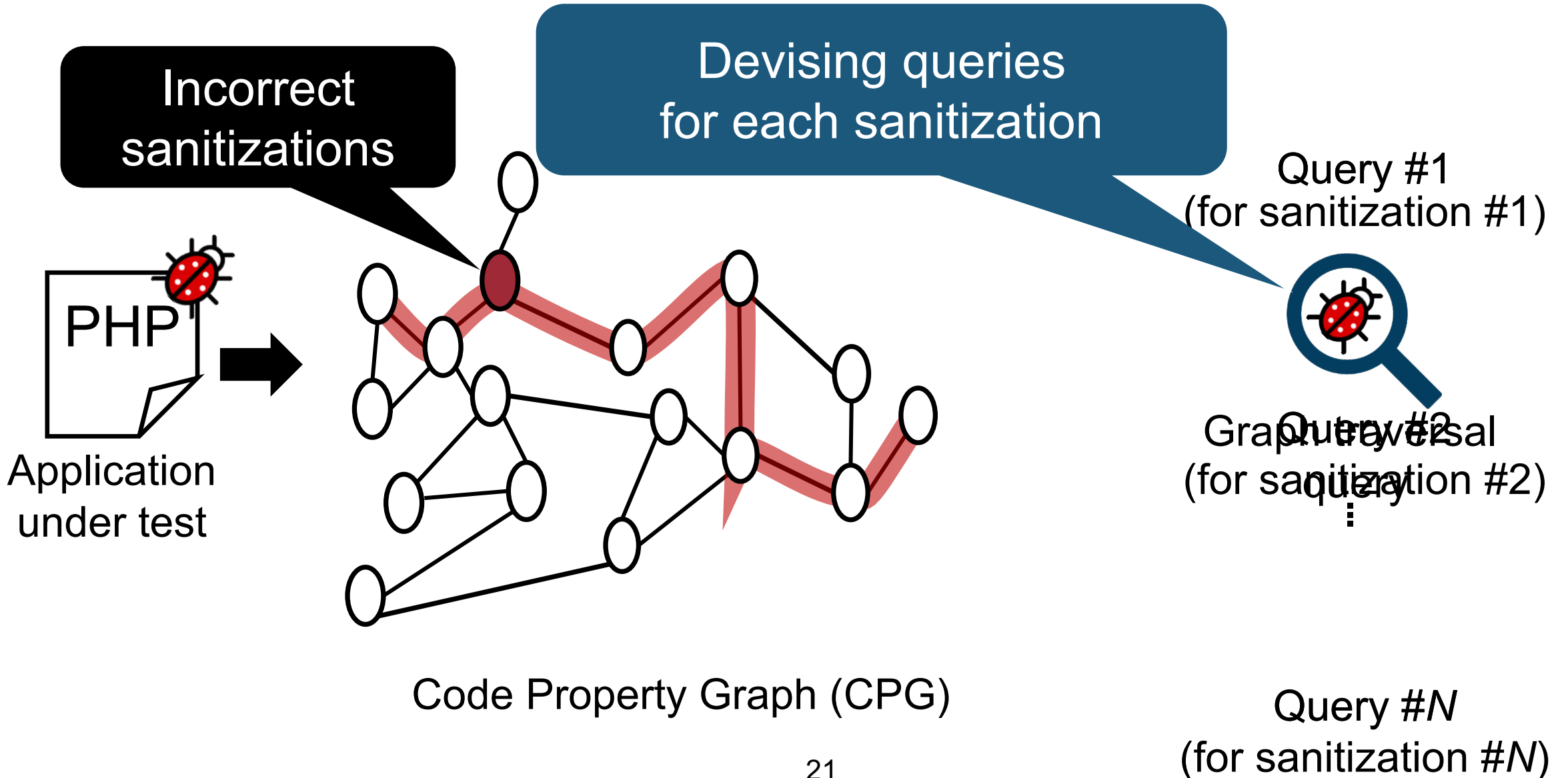
javascript:alert("xss")

Incorrectly patched code

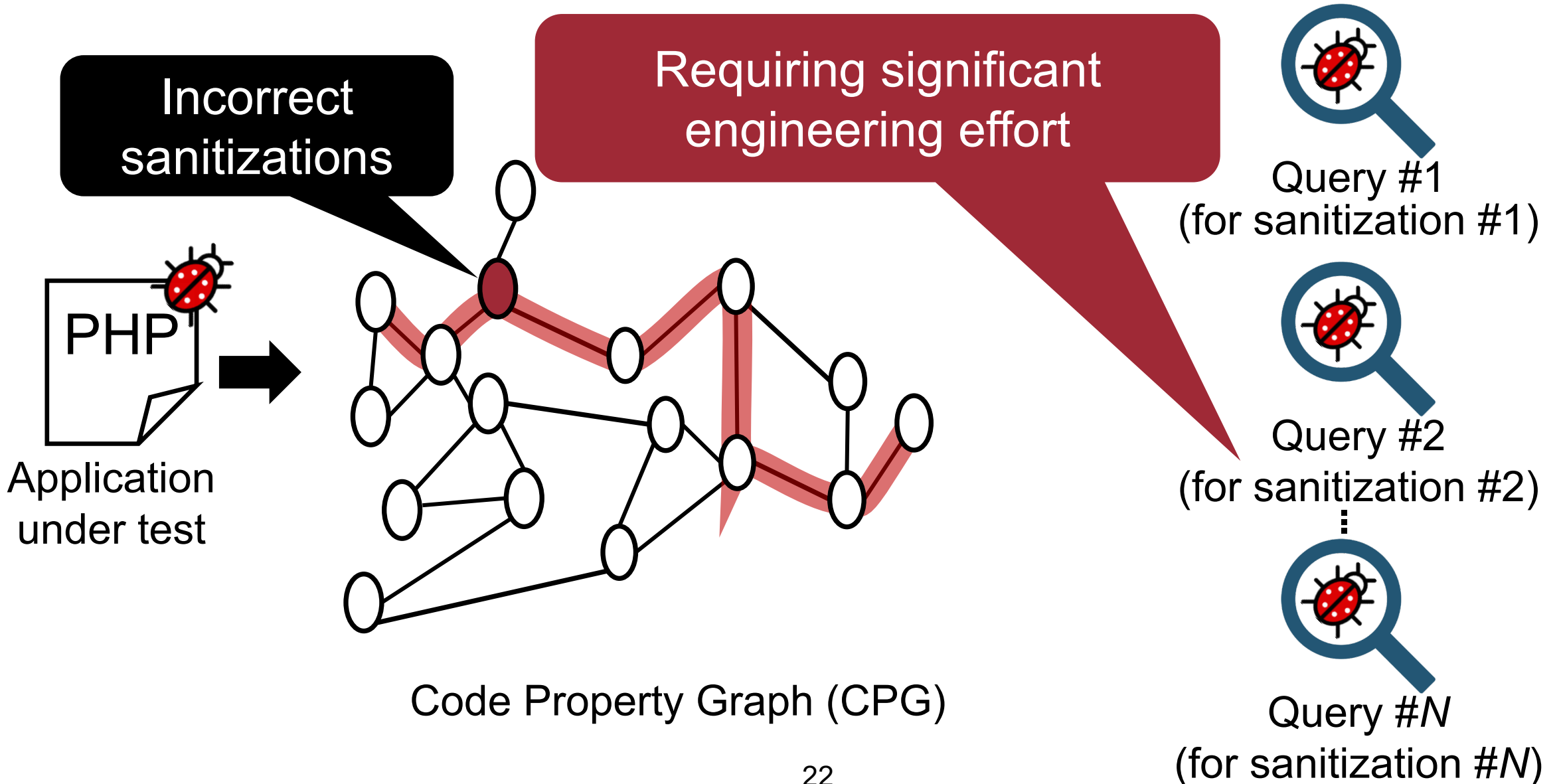
```
<a href = "javascript:alert("xss")"> Content </a>
```

This application is still vulnerable

Challenge for addressing the limitation



Challenge for addressing the limitation

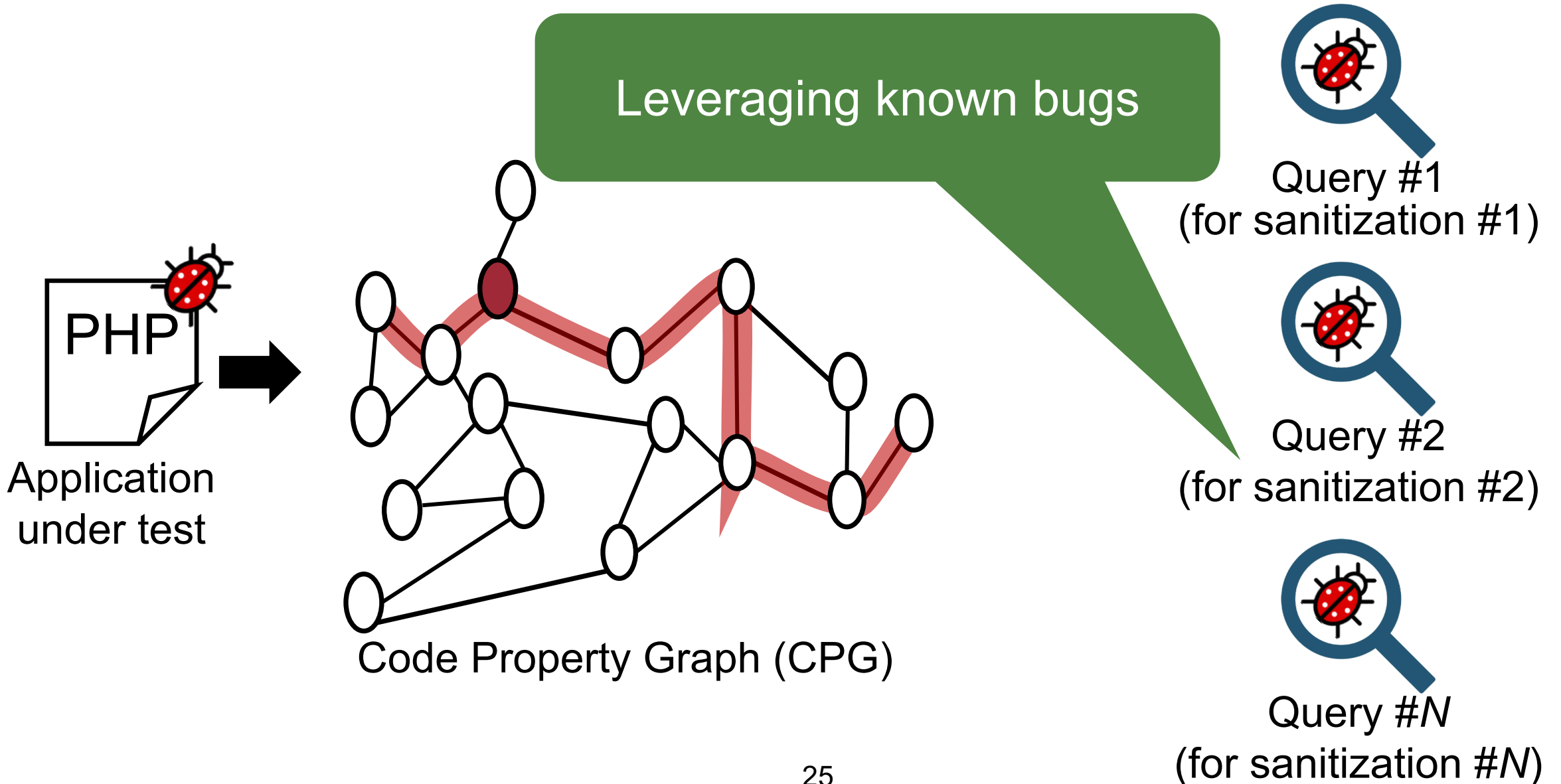


How do we address this challenge?

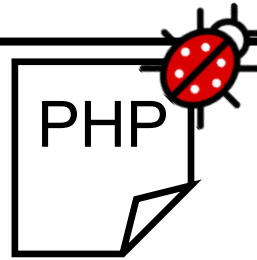
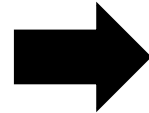
We propose

HiddenCPG

Our Approach



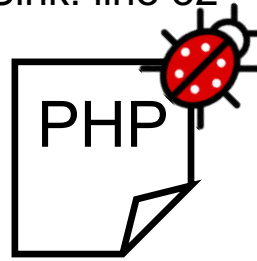
Our Approach – Leveraging Known Bugs



CVE-2019-41432
Source: line 4
Sink: line 32



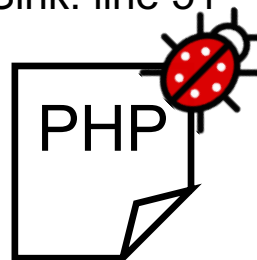
Query #1
(for sanitization #1)



CVE-2021-1482
Source: line 34
Sink: line 51



Query #2
(for sanitization #2)

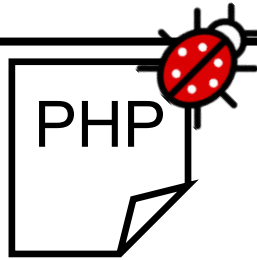
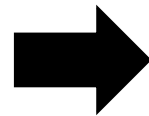


CVE-2018-4251
Source: line 453
Sink: line 552

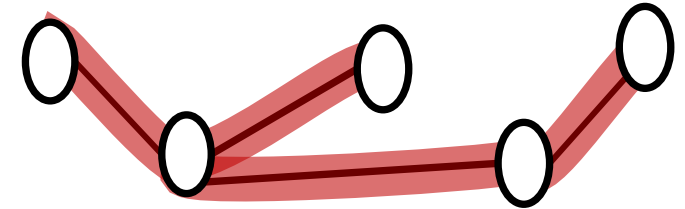
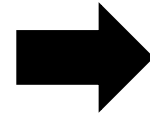


Query #N
(for sanitization #N)

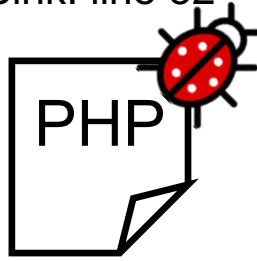
Our Approach – Extracting Buggy CPGs



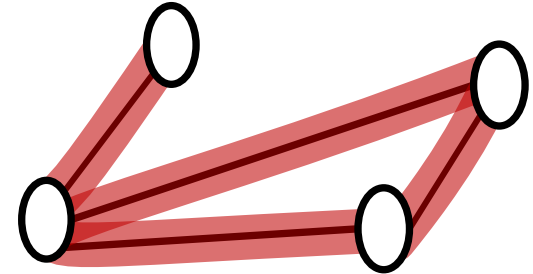
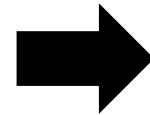
CVE-2019-41432
Source: line 4
Sink: line 32



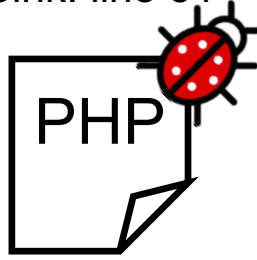
Query CPG #1



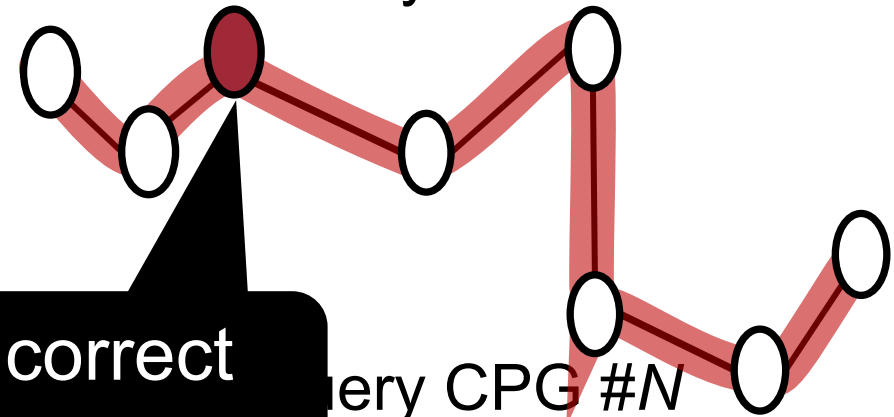
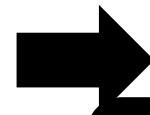
CVE-2021-1482
Source: line 34
Sink: line 51



Query CPG #2



CVE-2018-4251
Source: line 453
Sink: line 552

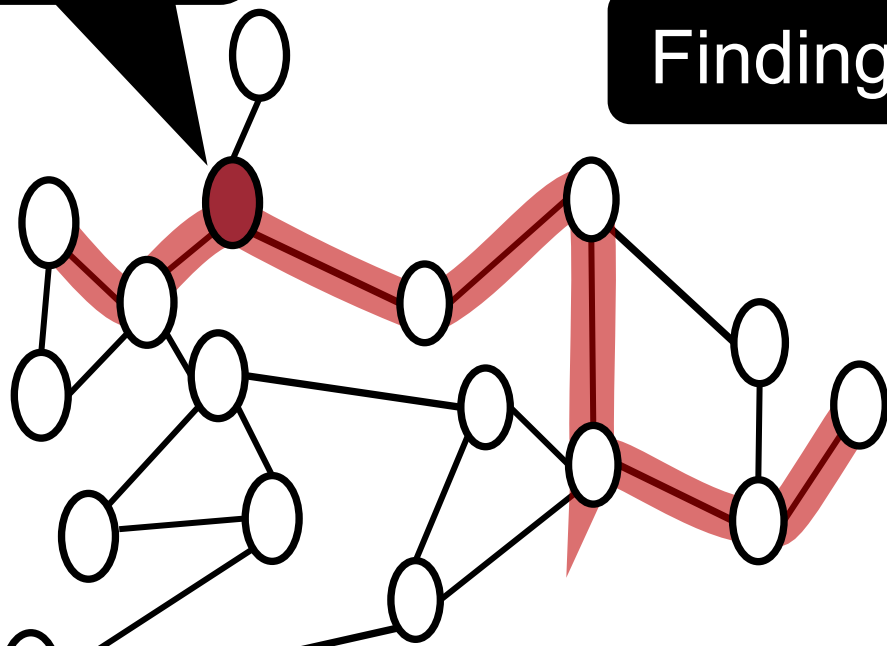


Query CPG #N

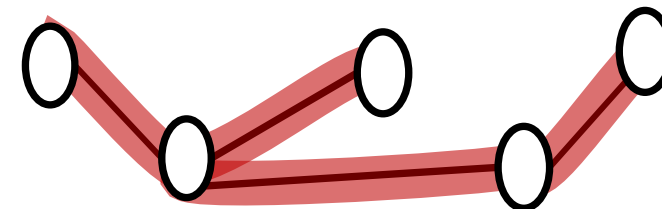
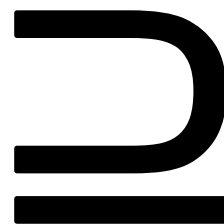
Incorrect sanitization

Our Approach

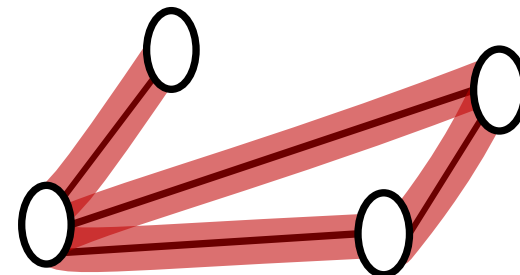
Incorrect sanitizations



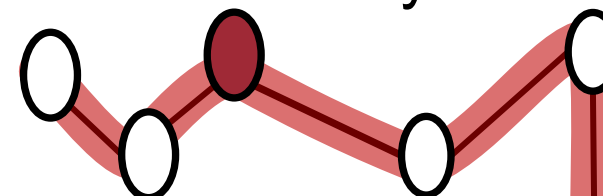
Finding a Subgraph



Query CPG #1



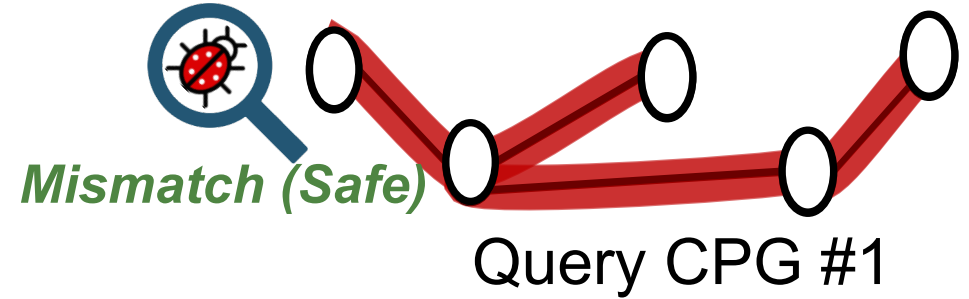
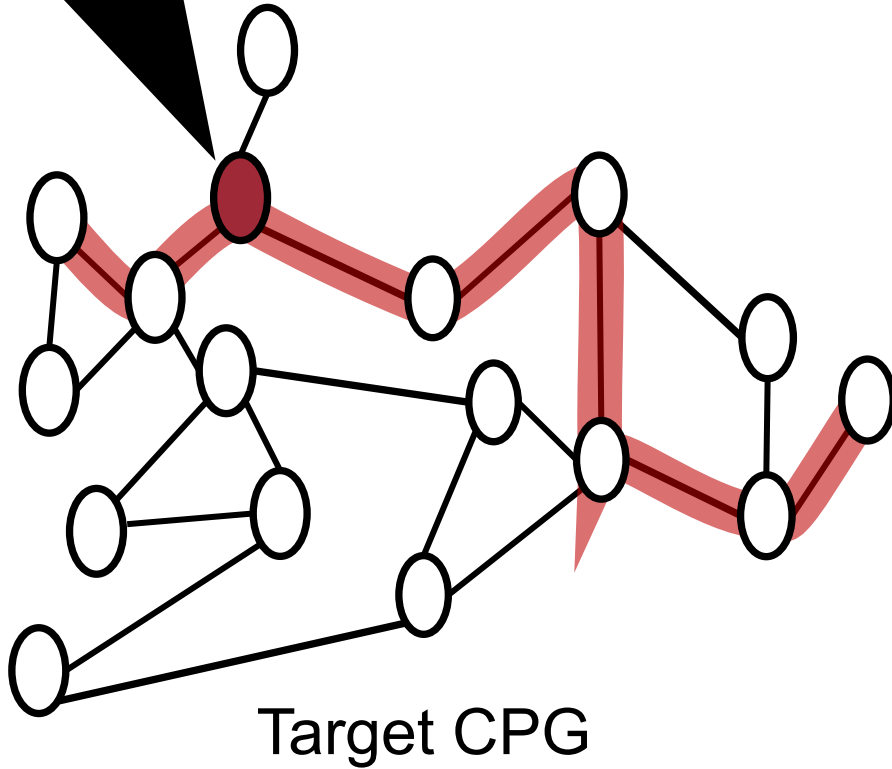
Query CPG #2



Check if the target CPG
contains a vulnerable CPG

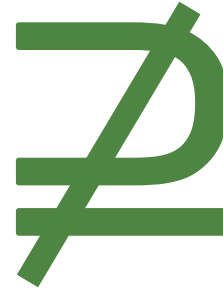
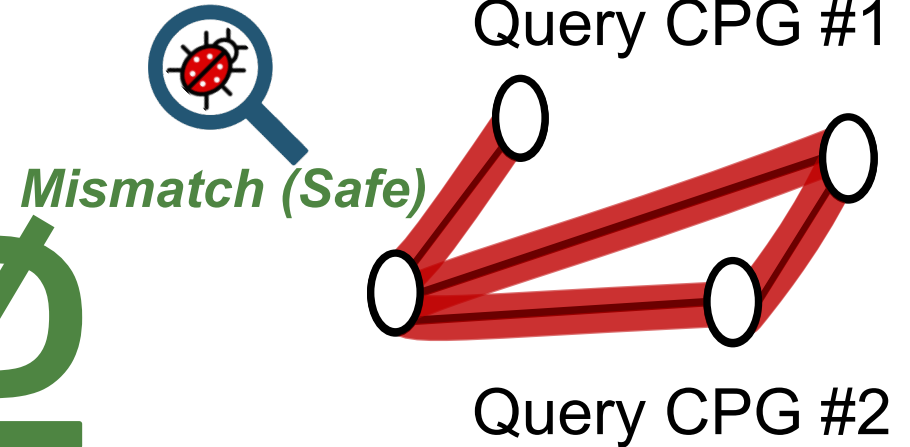
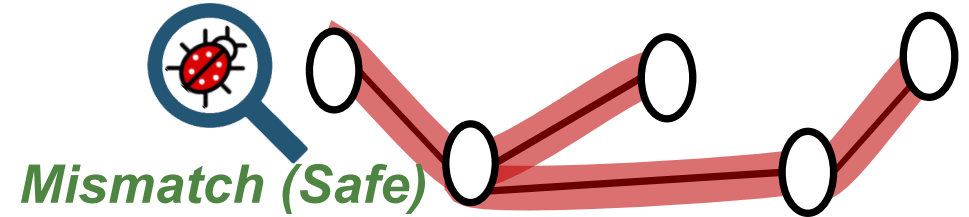
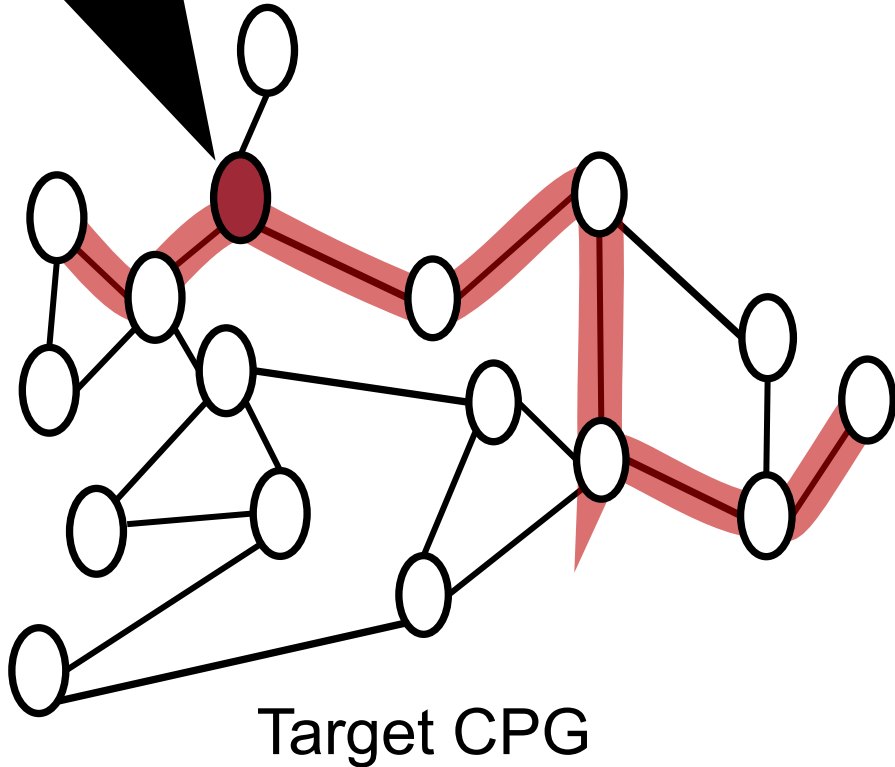
Our Approach – Finding a Subgraph

Incorrect sanitizations



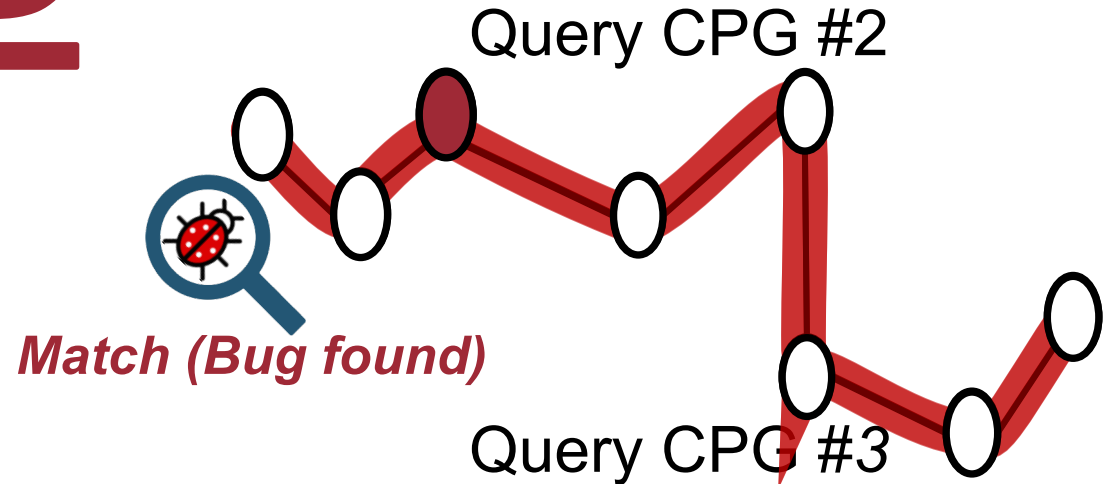
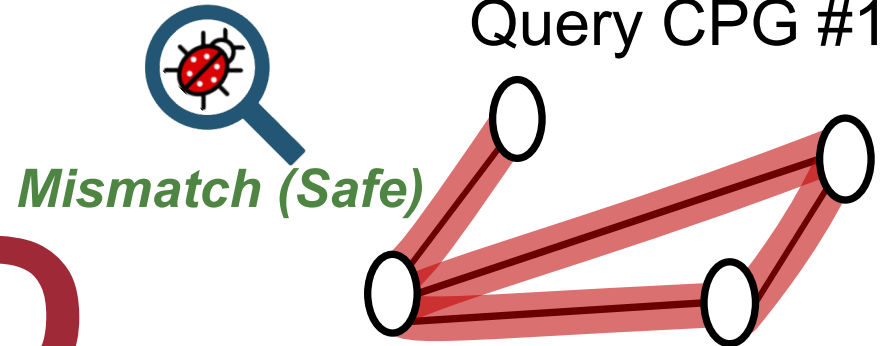
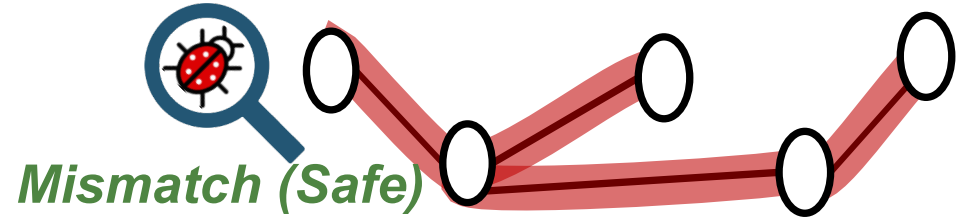
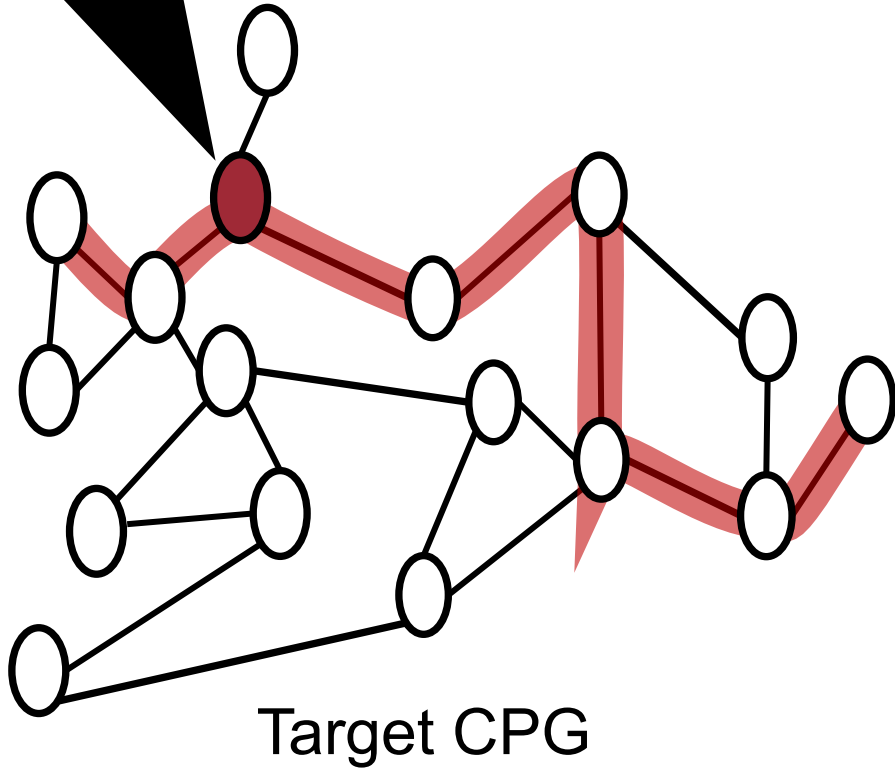
Our Approach – Finding a Subgraph

Incorrect sanitizations

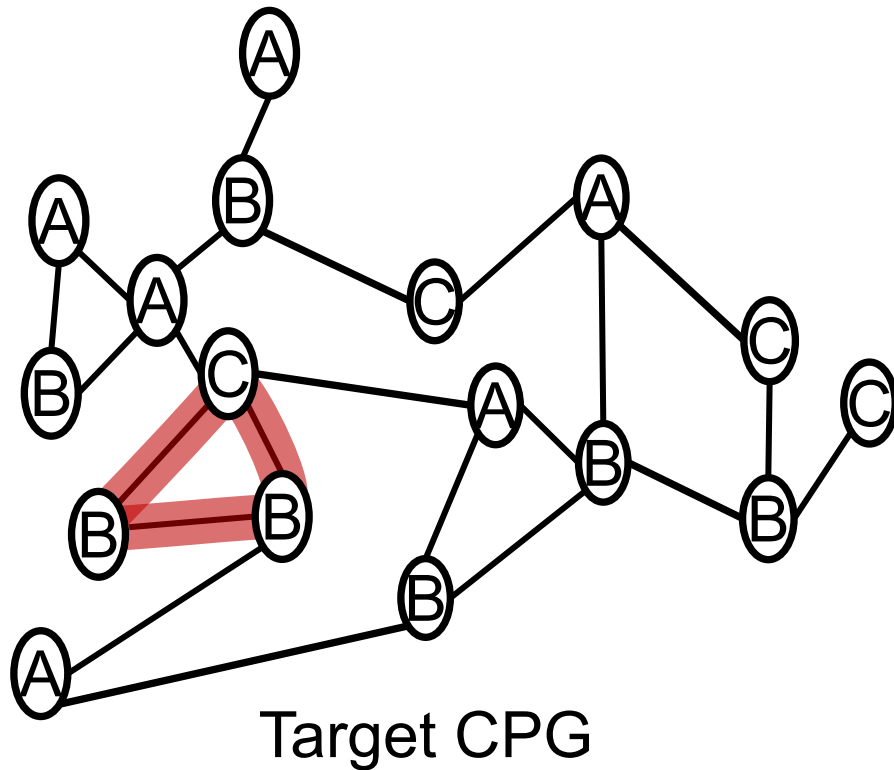


Our Approach – Finding a Subgraph

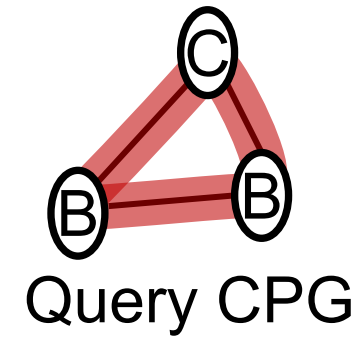
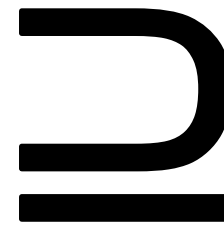
Incorrect sanitizations



Challenge #1: Scalability Problem



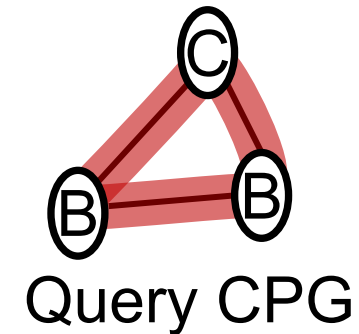
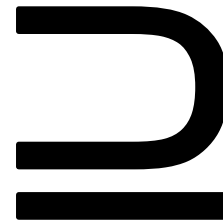
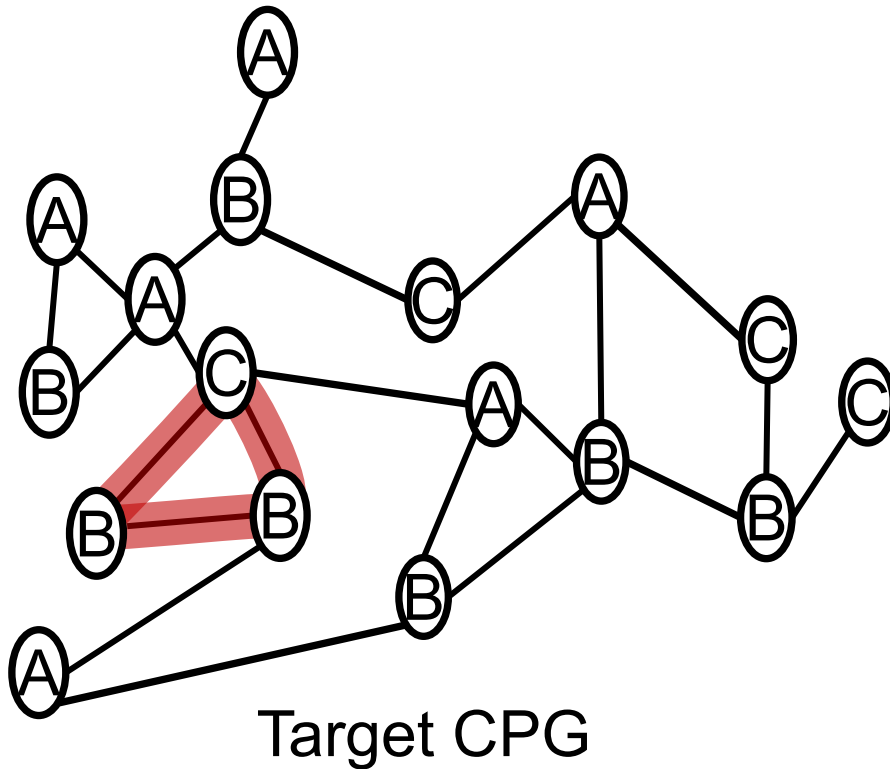
Subgraph Isomorphism problem
→ NP-complete



Challenge #1: Scalability Problem

- # of nodes: 200
- # of edges: 300

- # of nodes: 20
- # of edges: 30



Challenge #1: Scalability Problem

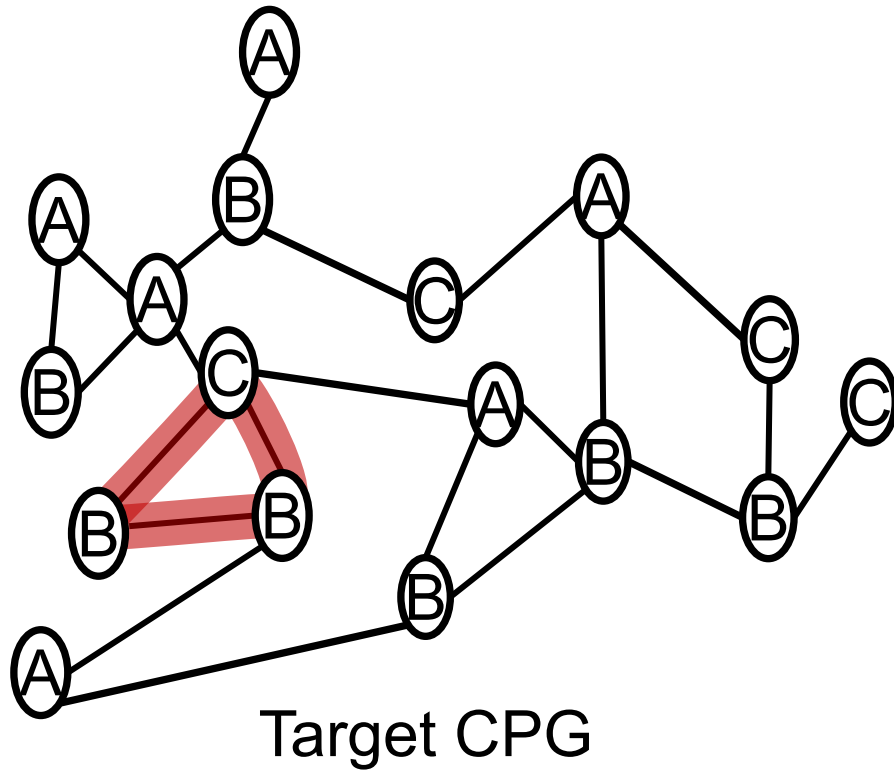
- # of nodes: 200
- # of edges: 300

- # of nodes: 20
- # of edges: 30

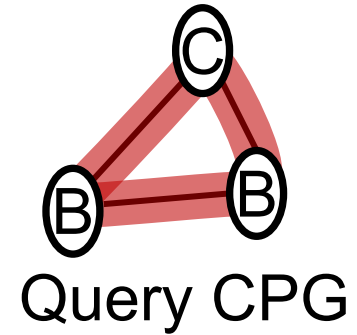
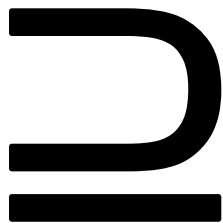
The number of comparisons using VF2: $O(N!N)$
If $N=550$: $O(7,031,875,837,044 \times 10^{1,260})$



Challenge #1: Scalability Problem

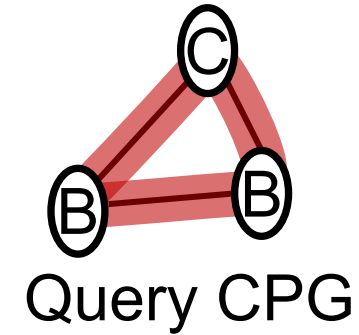
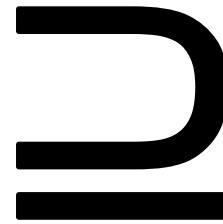
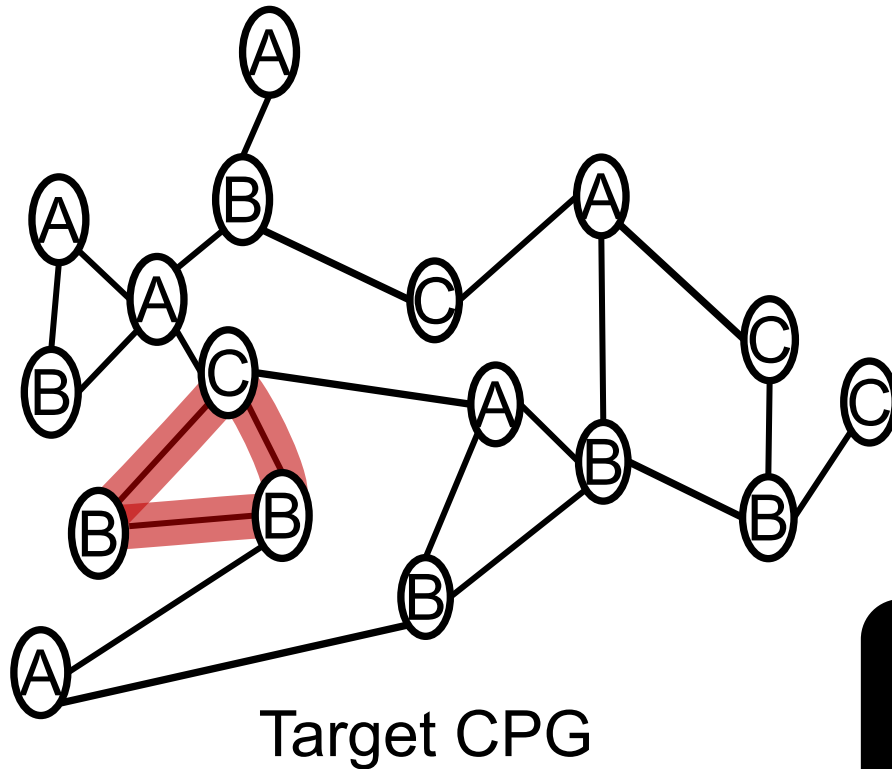


Pruning CPGs
to reduce search space



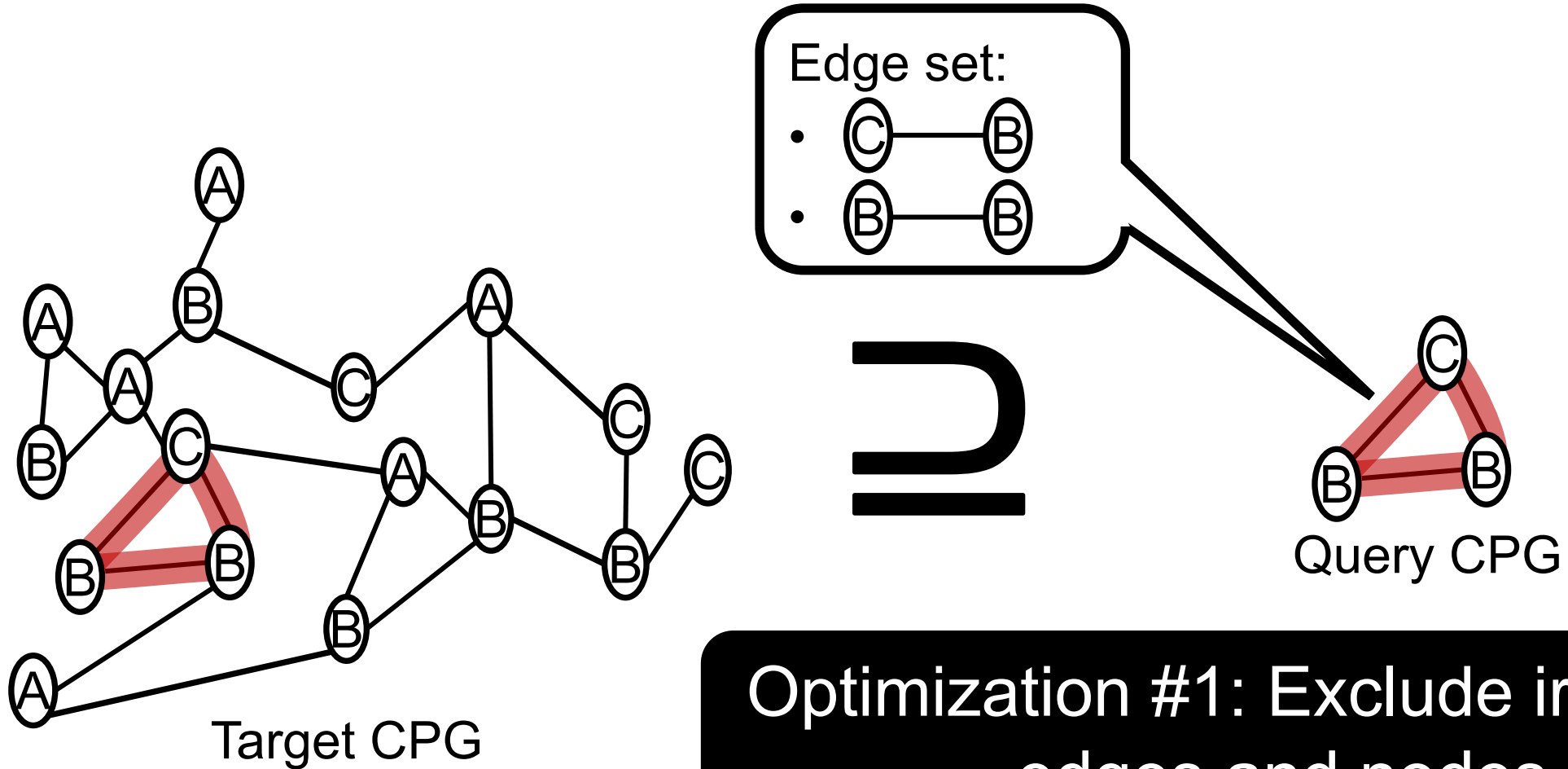
Cloned Buggy Code Detector (CBCD), ICSE '12

- Propose three optimization techniques for pruning nodes and edges



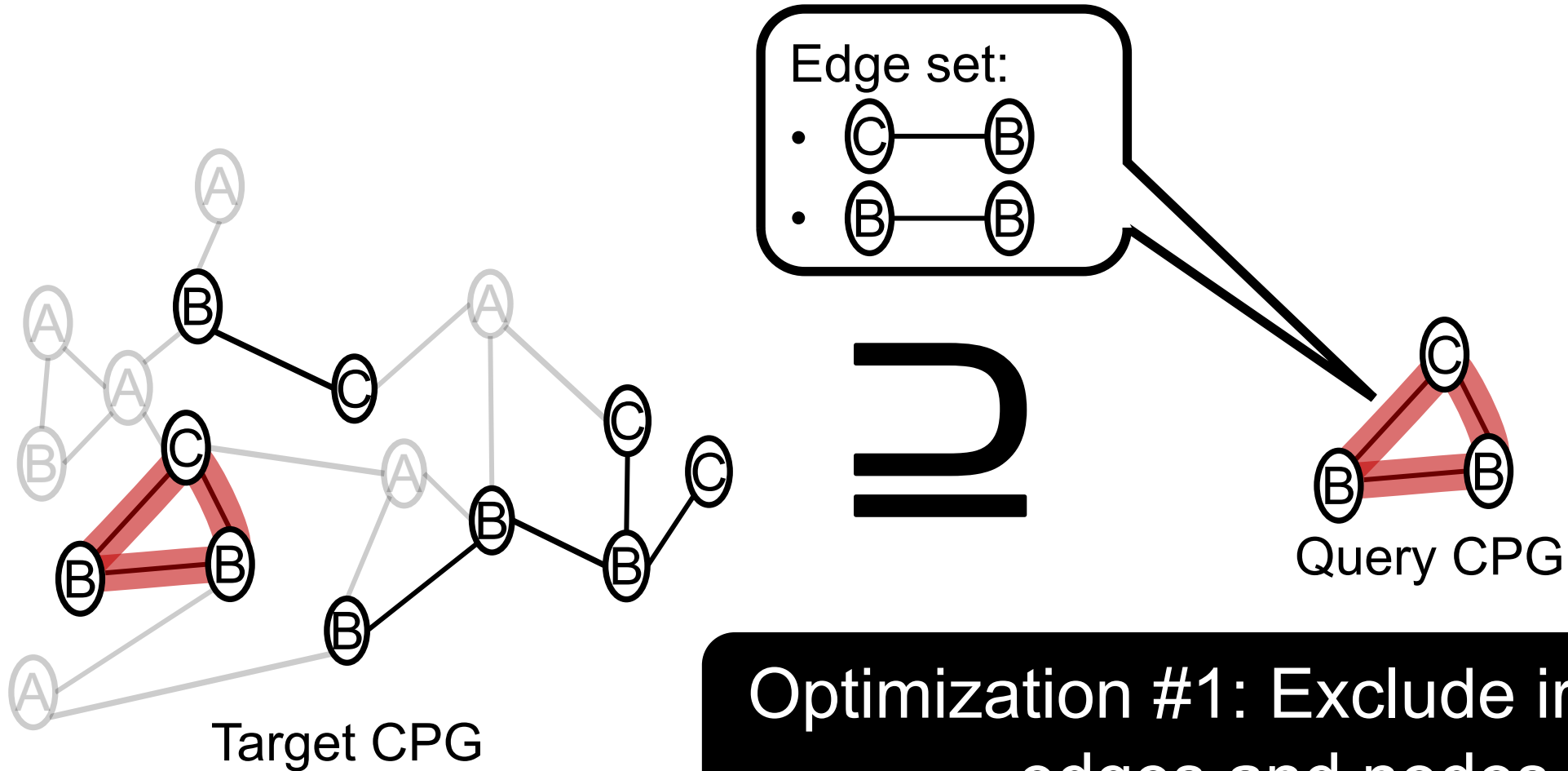
Optimization #1: Exclude irrelevant edges and nodes

Cloned Buggy Code Detector (CBCD), ICSE '12



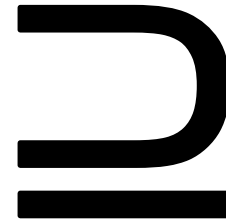
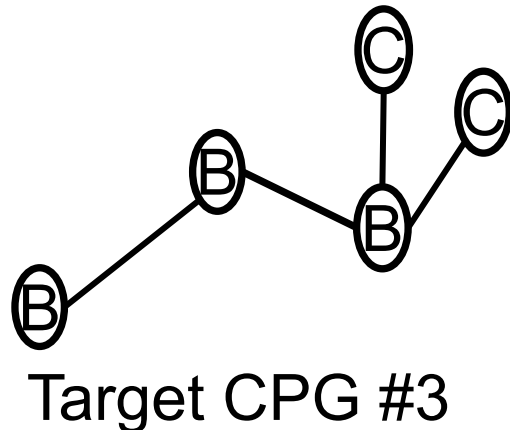
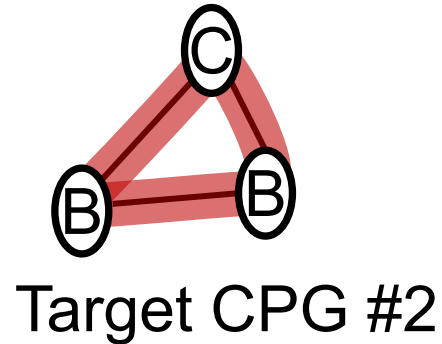
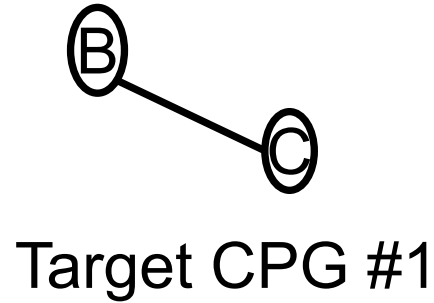
Optimization #1: Exclude irrelevant edges and nodes

Cloned Buggy Code Detector (CBCD), ICSE '12

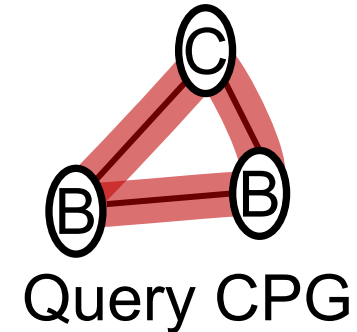
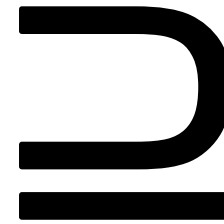


Optimization #1: Exclude irrelevant edges and nodes

Cloned Buggy Code Detector (CBCD), ICSE '12



Each matching focuses on a smaller graph



Optimization #1: Exclude irrelevant edges and nodes

More in the paper

- Optimization #2: Break target CPG into small graphs

- Optimization #3: Exclude irrelevant graphs

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$search = $_GET["search"];
if (hasData($search) {
    $now = time();
}
$content = htmlspecialchars($search);
include("header.html");
print $now;
echo "<body><a title='example' href='";
echo $content;
print "'>" . $now . "</a></body></html>";
?>
```

Application under test

```
<?php
    $input = $_GET["input"];
    $message = htmlspecialchars($input);
?>
<a href="
    <?php echo $message; ?>
">Content</a>
```

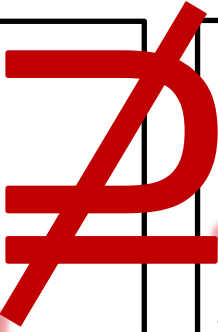
Known bug
(CVE-2018-4251)

**Difficult to match graphs that are
semantically identical but syntactically different**

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$search = $_GET["search"];
if (hasData($search) {
    $now = time();
}
$content = htmlspecialchars($search);
include("header.html");
print $now;
echo "<body><a title='example' href='";
echo $content;
print "'>" . $now . "</a></body></html>";
?>
```

Application under test



```
<?php
$input = $_GET["input"];
$message = htmlspecialchars($input);
?>
<a href="
    <?php echo $message; ?>
">Content</a>
```

CVE-2018-4251

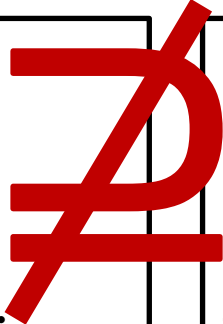
Source: line 2

Sink: line 9

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$search = $_GET["search"];
if (hasData($search) {
    $now = time();
}
$content = htmlspecialchars($search);
include("header.html");
print $now;
echo "<body><a title='example' href='";
echo $content;
print "'>" . $now . "</a></body></html>";
?>
```

Application under test



```
<?php
$input = $_GET["input"];
$message = htmlspecialchars($input);
?>
<a href="
  <?php echo $message; ?>
">Content</a>
```

CVE-2018-4251

Source: line 2

Sink: line 9

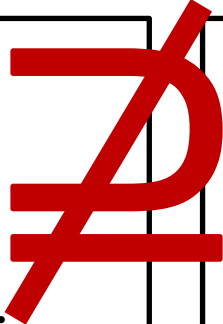
"search" ≠ "input"

Terminal node comparison

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$search = $_GET["search"];
if (hasData($search) {
    $now = time();
}
$content = htmlspecialchars($search);
include("header.html");
print $now;
echo "<body><a title='example' href='";
echo $content;
print "'>" . $now . "</a></body></html>";
?>
```

Application under test



```
<?php
$input = $_GET["input"];
$message = htmlspecialchars($input);
?>
<a href="
  <?php echo $message; ?>
">Content</a>
```

CVE-2018-4251

Source: line 2

Sink: line 9

"search" ≠ "input"

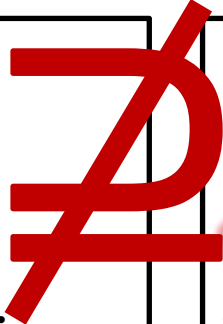
\$content ≠ \$message

Terminal node comparison

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$search = $_GET["search"];
if (hasData($search) {
    $now = time();
}
$content = htmlspecialchars($search);
include("header.html");
print $now;
echo "<body><a title='example' href='";
echo $content;
print "'>" . $now . "</a></body></html>";
?>
```

Application under test



```
<?php
$input = $_GET["input"];
$message = htmlspecialchars($input);
?>
<a href="
<?php echo $message; ?>
">Content</a>
```

CVE-2018-4251

Source: line 2

Sink: line 9

"search" ≠ "input"

\$content ≠ \$message

Terminal node comparison

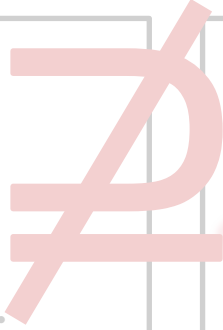
"<body><a title='example' href='"

≠ "<a href='"

Printing context comparison

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$search = $_GET["search"];
if (hasData($search) {
    $now = time();
}
$content = htmlspecialchars($search);
```



```
<?php
$input = $_GET["input"];
$message = htmlspecialchars($input);
?>
<a href="
<?php echo $message; ?>
">Content</a>
```

Determining a proper level of abstraction for CPGs affects the accuracy in matching

"search" ≠ "input"

\$content ≠ \$message

Terminal node comparison

"<body><a title='example' href=''"

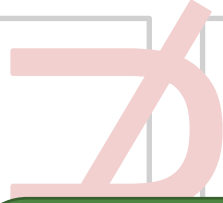
≠ "<a href=''"

Printing context comparison

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$search = $_GET["search"];
if (hasData($search) {
    $now = time();
}
$content = htmlspecialchars($search);
include("header.html");
print $now;
echo "<body><a title='example' href=";
echo $content;
print ">>" . $now . "</a></body></html>";
?>
```

Application under test



```
<?php
$input = $_GET["input"];
$message = htmlspecialchars($input);
?>
```

Optimal level of abstraction:

- Resilient to common modification
- Preserving the vulnerable condition

"search" ≠ "input"
\$content ≠ \$message

Terminal node comparison

"<body><a title='example' href="
≠ "<a href="

Printing context comparison

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$search = $_GET["search"];
if (hasData($search) {
    $now = time();
}
$content = htmlspecialchars($search);
include("header.html");
print $now;
echo "<body><a title='example' href='";
echo $content;
print "'>" . $now . "</a></body></html>";
?>
```

Application under test

```
<?php
    $input = $_GET["input"];
    $message = htmlspecialchars($input);
?>
<a href="
    <?php echo $message; ?>
">Content</a>
```

CVE-2018-4251

Source: line 2

Sink: line 9

"search" ≠ "input"

\$content ≠ \$message

Terminal node comparison

"<body><a title='example' href='"

≠ "<a href='"

Printing context comparison

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$norm = $_GET["norm"];
if (hasData($search) {
    $now = time();
}
$norm = htmlspecialchars($norm);
include("header.html");
print $now;
echo "<body><a title='example' href='";
echo $norm;
print "'>" . $now . "</a></body></html>";
?>
```

Application under test

```
<?php
    $norm = $_GET["norm"];
    $norm = htmlspecialchars($norm);
?>
<a href="
    <?php echo $norm; ?>
">Content</a>
```

CVE-2018-4251

Source: line 2

Sink: line 9

"norm" = "norm"

\$norm = \$norm

Terminal node abstraction

"<body><a title='example' href='"

≠ "<a href='"

Printing context comparison

Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$norm = $_GET["norm"];
if (hasData($search) {
    $now = time();
}
$norm = htmlspecialchars($norm);
include("header.html");
print $now;
echo "norm_a_href";
echo $norm;
print "'>' . $now . "</a></body></html>";
?>
```

Application under test

```
<?php
    $norm = $_GET["norm"];
    $norm = htmlspecialchars($norm);
?>
"norm_a_href";
<?php echo $norm; ?>
">Content</a>
```

CVE-2018-4251

Source: line 2

Sink: line 9

"norm" = "norm"

\$norm = \$norm

Terminal node abstraction

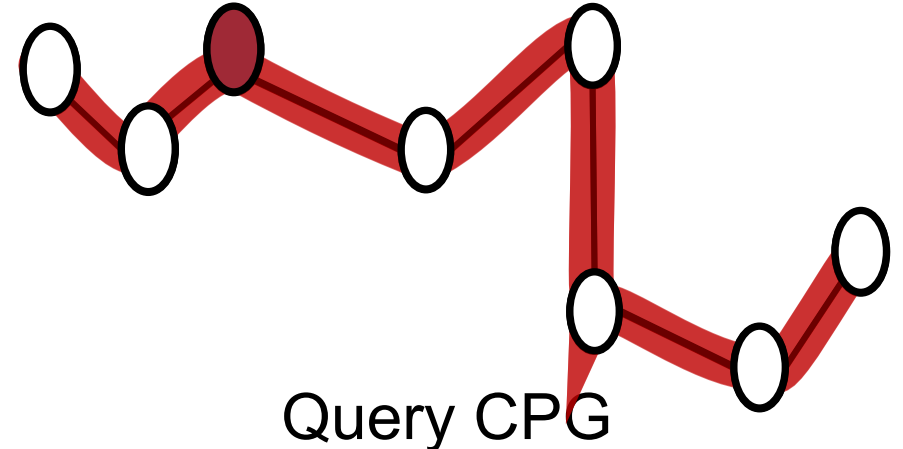
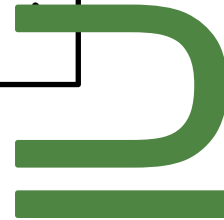
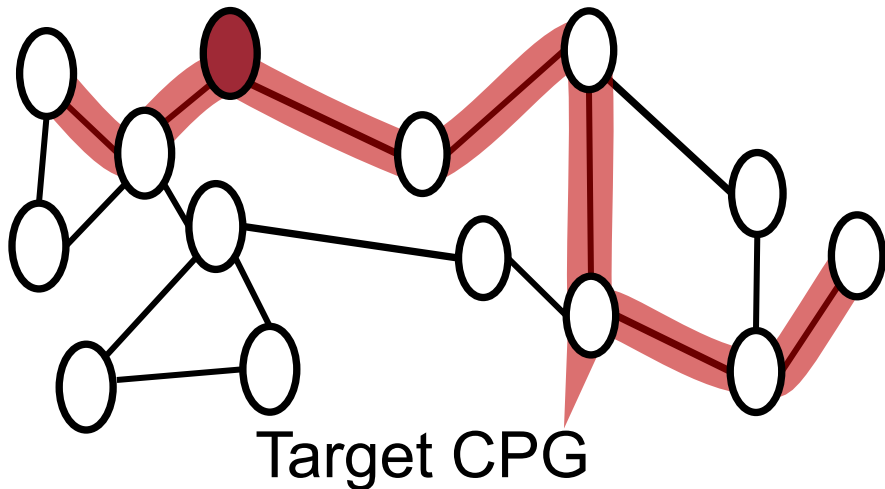
"norm_a_href" = "norm_a_href"
(norm_[tag name]_[attribute name])

Printing context abstraction

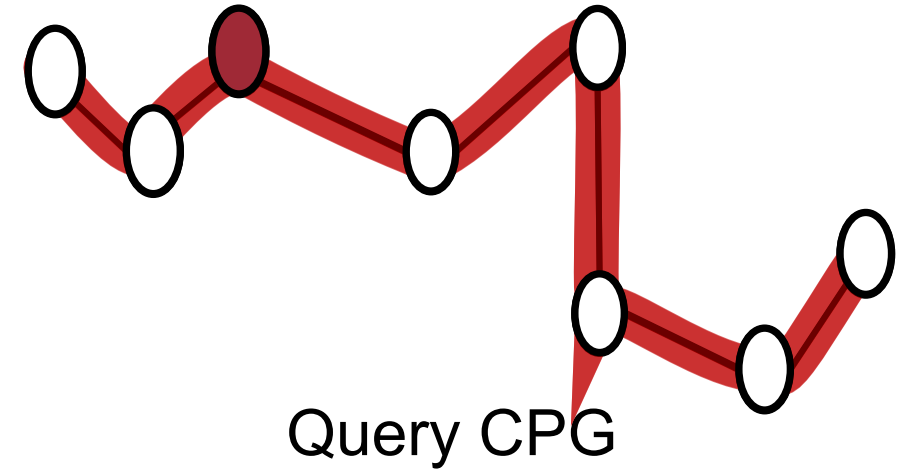
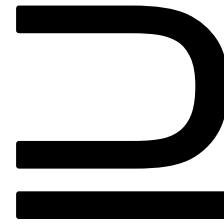
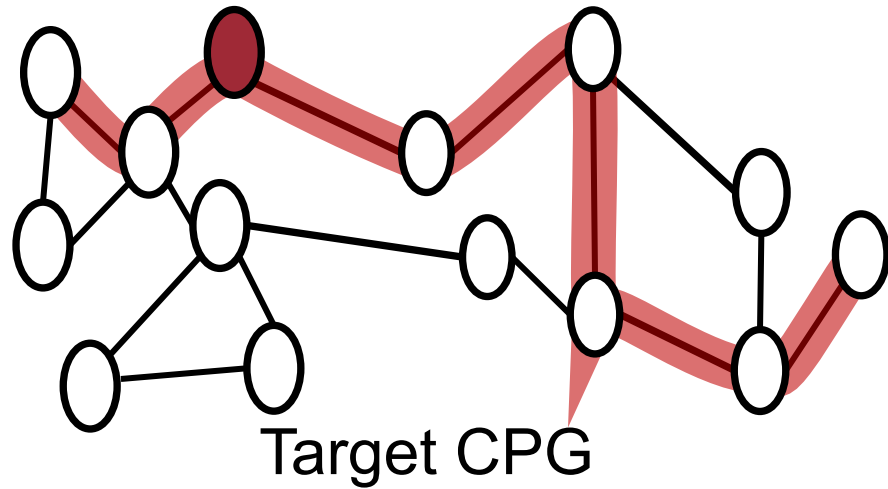
Challenge #2: Graph Abstraction

```
<?php
print "<input>";
$norm = $_GET["norm"];
if (hasData($search) {
    $now = time();
}
$norm = htmlspecialchars($norm);
include("header.html");
print $now;
echo "norm_a_href";
echo $norm;
print "'>" . $now . "</a></body></html>":
?>
```

```
<?php
$norm = $_GET["norm"];
$norm = htmlspecialchars($norm);
?>
echo "norm_a_href"
<?php echo $norm; ?>
">Content</a>
```



Experimental Setup



- **7,174 PHP** applications with more than 100 stars on GitHub
 - **# of nodes:** \simeq 1.1 billion
 - **# of edges:** \simeq 1.3 billion

- **103 queries** from 40 web applications
 - Cross-site Scripting: 66
 - Unrestricted File Upload: 1
 - SQL Injection: 31
 - Local File Inclusion: 5

- Include 10 incorrect sanitizations

The largest collection of PHP applications in a single study

Bugs Found – Matched Subgraphs

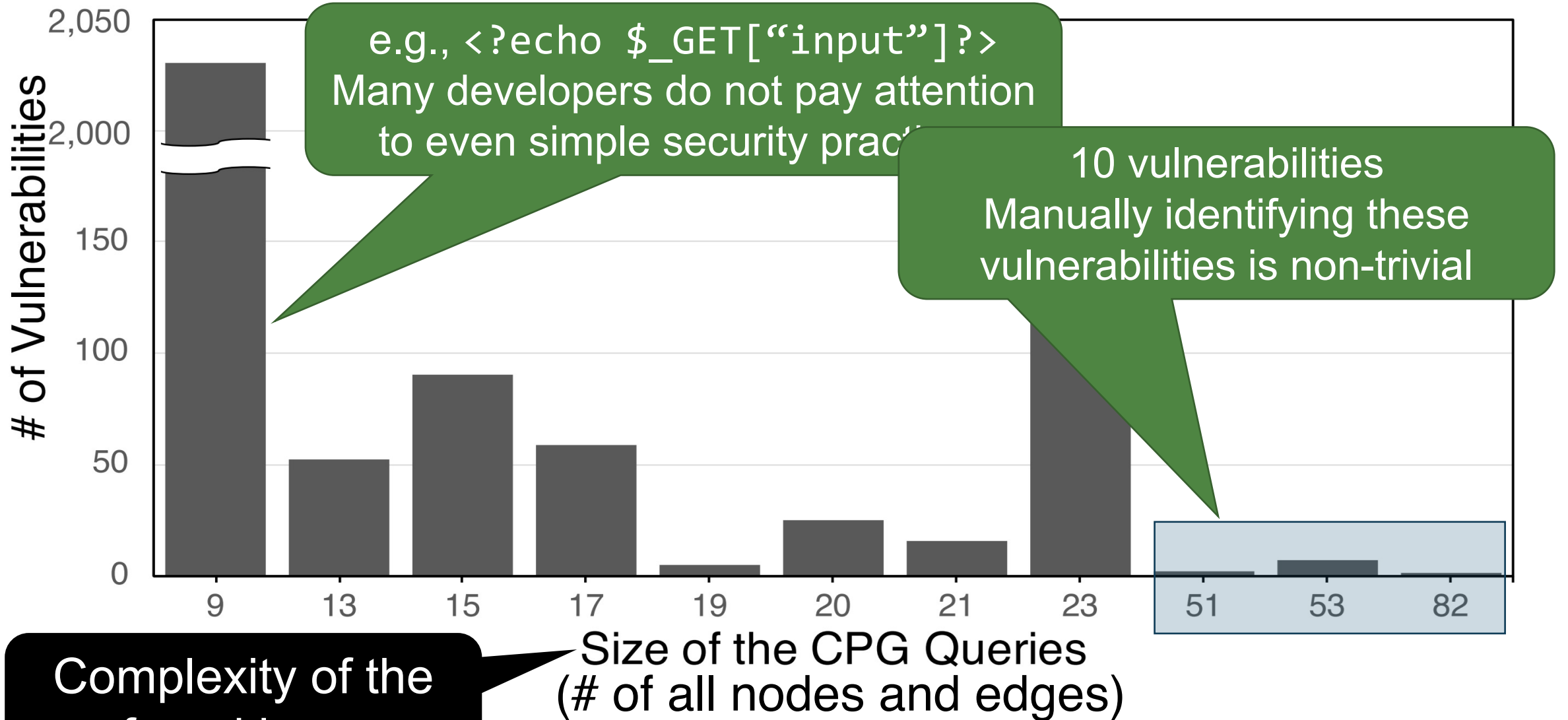
- HiddenCPG found **2,464 distinct potential vulnerabilities** (*i.e.*, matched subgraphs) including **39 incorrect sanitizations**

Vulnerability Type	# of Matched Subgraphs
Cross-Site Scripting	2,416
Unrestricted File Upload	2
SQL Injection	9
Local File Inclusion	37
Total	2,464

Bugs Found – Manual Verification

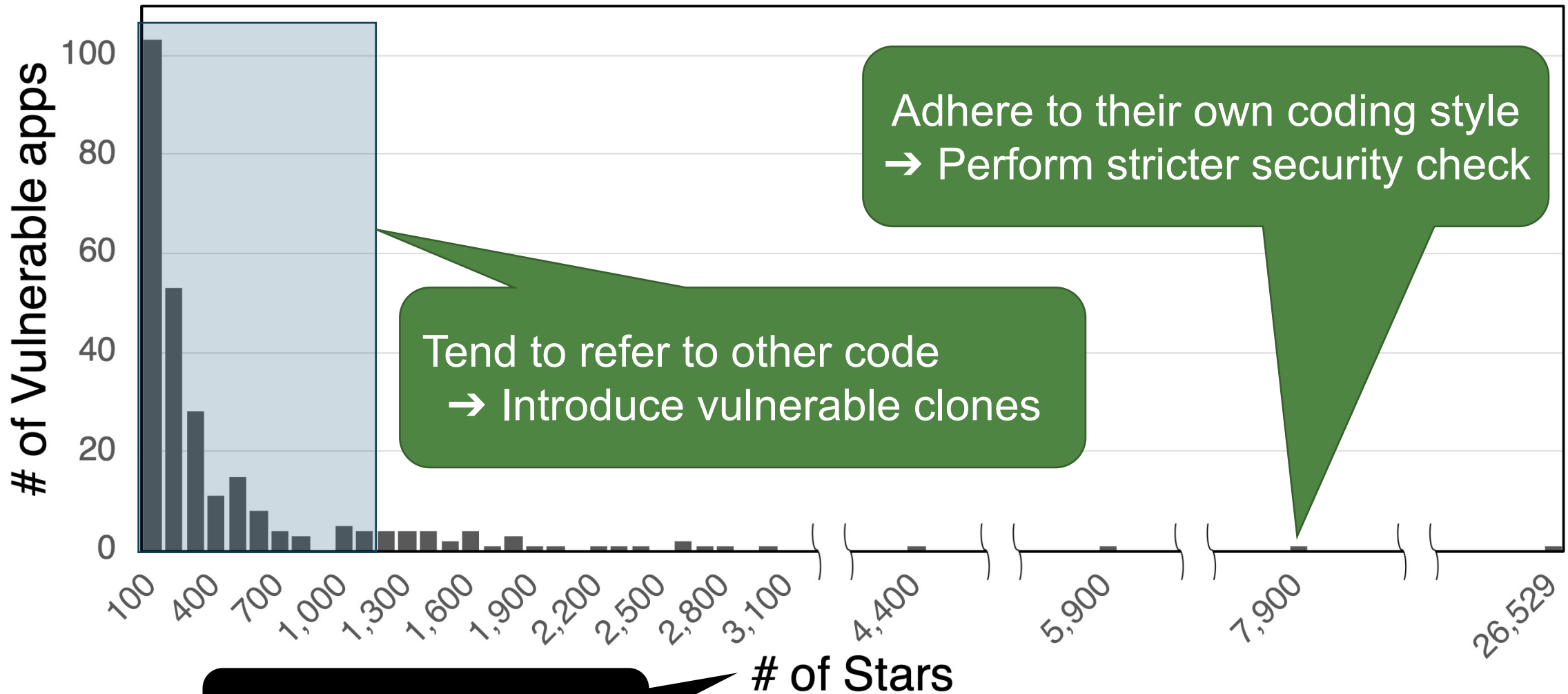
- We analyzed **103 sampled reports**
 - Cross-site Scripting: 94
 - Unrestricted File Upload: 2
 - SQL Injection: 5
 - Local File Inclusion: 2
- **14 reports (13.5%)** were *false positives*
 - 12 reports: separate sanitization logic in dynamic callbacks
 - 2 reports: anti-CSRF protection for POST requests
- We reported **89 vulnerabilities**
 - **42 CVEs** from 17 applications

Query Sizes and Vulnerabilities



Complexity of the found bugs

Project Popularities and Vulnerabilities



Adhere to their own coding style
→ Perform stricter security check

Tend to refer to other code
→ Introduce vulnerable clones

Project popularity

vs. State-of-the-Arts

- PHPJoern: a graph traversal-based vulnerable pattern detection tool
 - RIPS: an open-source taint analysis tool
 - Evaluation benchmark:
 - **Nunes *et al.* [1]**: 16 applications (8 XSS and 16 SQLi vulnerabilities)
 - **Incorrect sanitizations**: 15 applications (20 XSS vulnerabilities)
-

	HiddenCPG	PHPJoern	RIPS
True Positives	39	32	22
False Negatives	5	12	22
False Positives	1	25	24

Why HiddenCPG found more bug?

- Fine-grained queries
 - Detect bugs that stem from incorrect sanitization
- Comprehensive graph abstraction
 - Normalize the WordPress APIs as sinks

	HiddenCPG	PHPJoern	RIPS
True Positives	39	32	22
False Negatives	5	12	22
False Positives	1	25	24

Performance

Demonstrate the effectiveness of
HiddenCPG in
scalable subgraph matching

	HiddenCPG	PHPJoern
Target Projects	7,174 projects	1,854 projects
Execution Time	16 days and 12 hours	6 days and 13 hours
Computing Power	6 core 3.20GHz Intel Core i7-8700 32 GB of RAM	32 core 2.60 GHz Intel Xeon 768 GB of RAM

Limitation


- HiddenCPG requires **manual effort for specifying sources and sinks** of known bugs to extract CPG queries

- HiddenCPG cannot detect **separate sanitization logic** in dynamic callbacks

Conclusion

- We proposed HiddenCPG, a **clone detection system designed to identify various web vulnerabilities**, including bugs that stem from incorrect sanitization
- We applied **three optimization techniques** introduced in CBCD to address the scalability problem
- We proposed several methods of **abstracting CPG**
- HiddenCPG found **2,464 potential web vulnerabilities**, including 89 confirmed bugs in the 7,174 PHP applications

Open Science

 **WSP-LAB / HiddenCPG** Public Watch 4 ★ Star 4 Fork 0

<> Code ! Issues 0 🔗 Pull requests 0 ▶ Actions 📁 Projects 0 📖 V

<https://github.com/WSP-LAB/HiddenCPG>



Question?