Static Detection of Vulnerabilities in Modern PHP Applications

Johannes Dahse

HackPra ’14, November 26th, Ruhr-University Bochum, Germany

1.1 Background

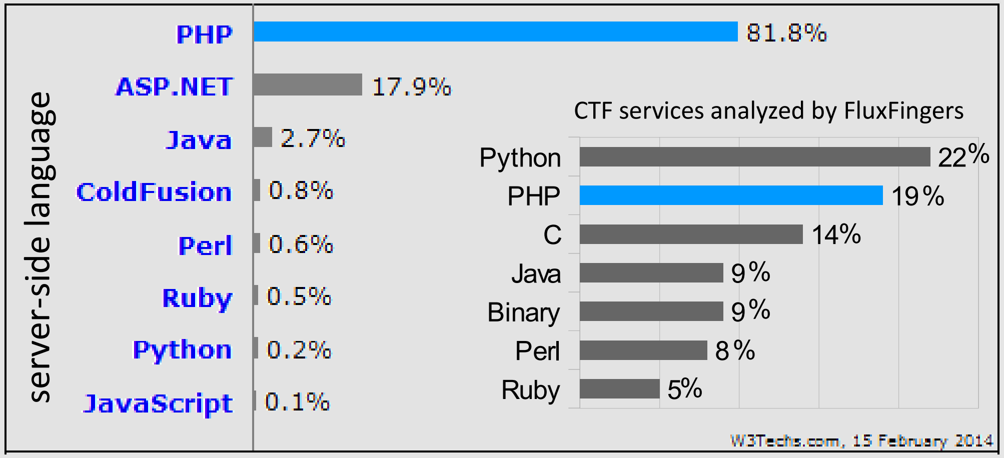
- @FluxReiners / websec.wordpress.com
- IT-Security Student, Ruhr-University Bochum (2006 - 2012)
- Co-founder CTF team *FluxFingers* (2007)
- Co-founder *HackerPraktikum*, *BadBank* Developer (2009)
- Penetration Tester / Code Auditer
- *RIPS 0.5* – Static Code Analyzer (2009 - 2011)
- *RIPS 1.0* – New rewritten prototype (2012 – today, 24/7)
- PhD Student at Chair for Systems Security, RUB (2013 - today)

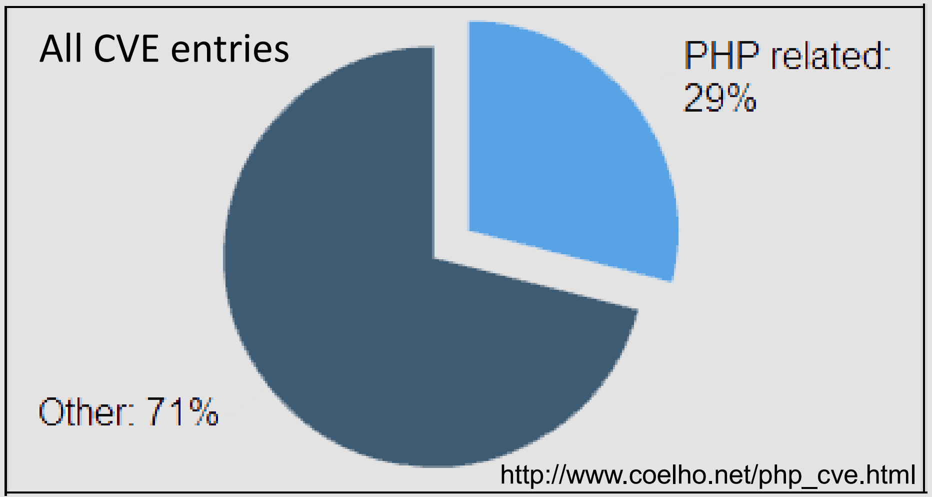
1.2 Why PHP?

WordPress
(23% of all websites)

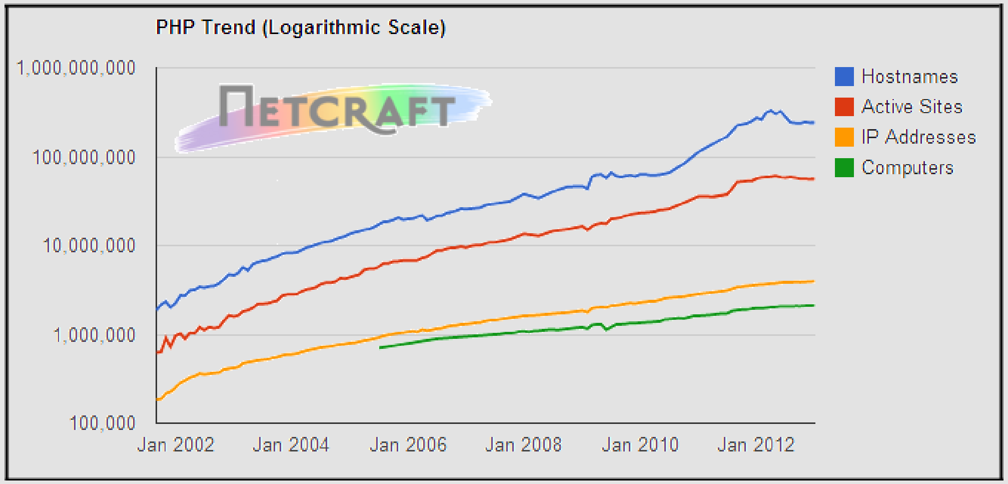
Joomla!
(3% of all websites)

http://w3techs.com





http://www.coelho.net/php_cve.html



1.3 Motivation

- SQL Injection in BadBank

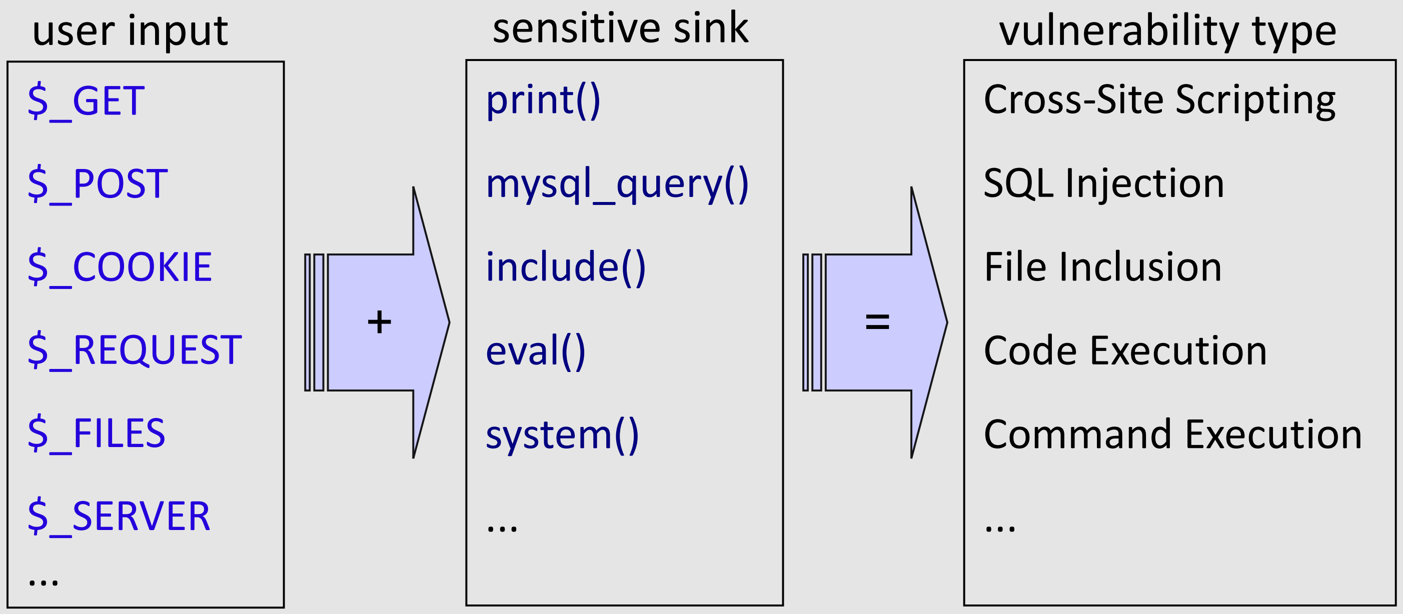
```
$id = $_GET['id'];
$sql = "SELECT id, titel, hinweis
          FROM hinweise WHERE id = ".$id;
$result = mysql_query($sql);
```

- Cross-Site Scripting in BadBank

```
$order = $_GET['order'];
$html = " (nach " . $order . ")\n";
echo $html;
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

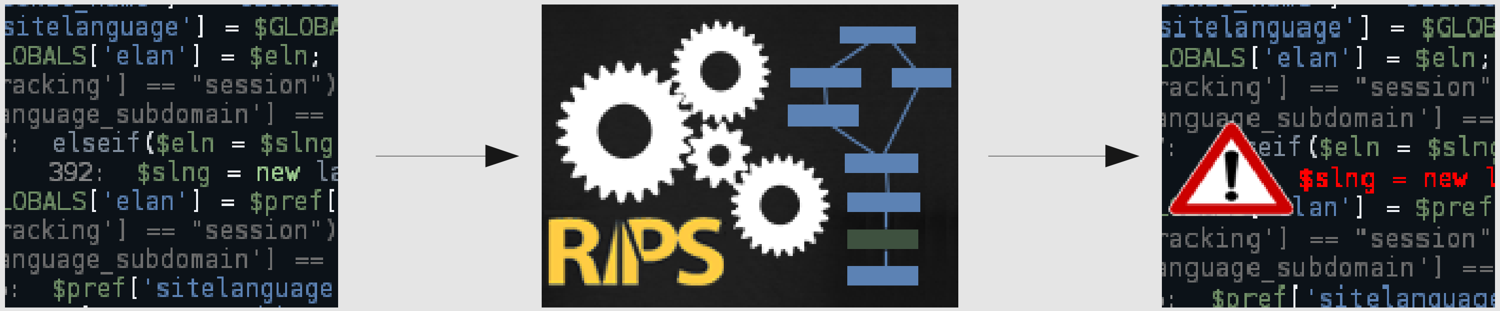
1.4 Taint-style Vulnerabilities



Note: Logical Flaws do not follow such a general concept and are harder to detect

RIPS 0.5

BadBank Demo Scan



Static Detection of Vulnerabilities in Modern PHP Applications

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

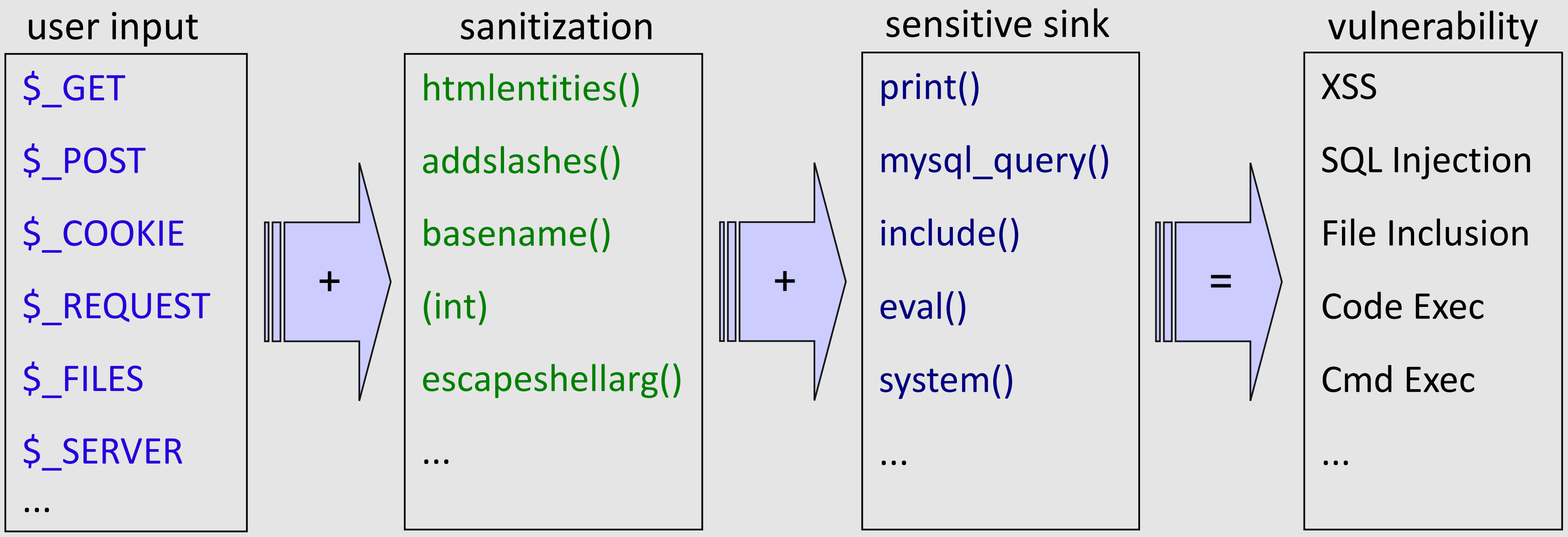
1.4 Taint-style Vulnerabilities

user input		sensitive sink		vulnerability type
$_GET		print()		Cross-Site Scripting
$_POST		mysql_query()		SQL Injection
$_COOKIE	+	include()	=	File Inclusion
$_REQUEST		eval()		Code Execution
$_FILES		system()		Command Execution
$_SERVER		...		...
...				

Note: Logical Flaws do not follow such a general concept and are harder to detect

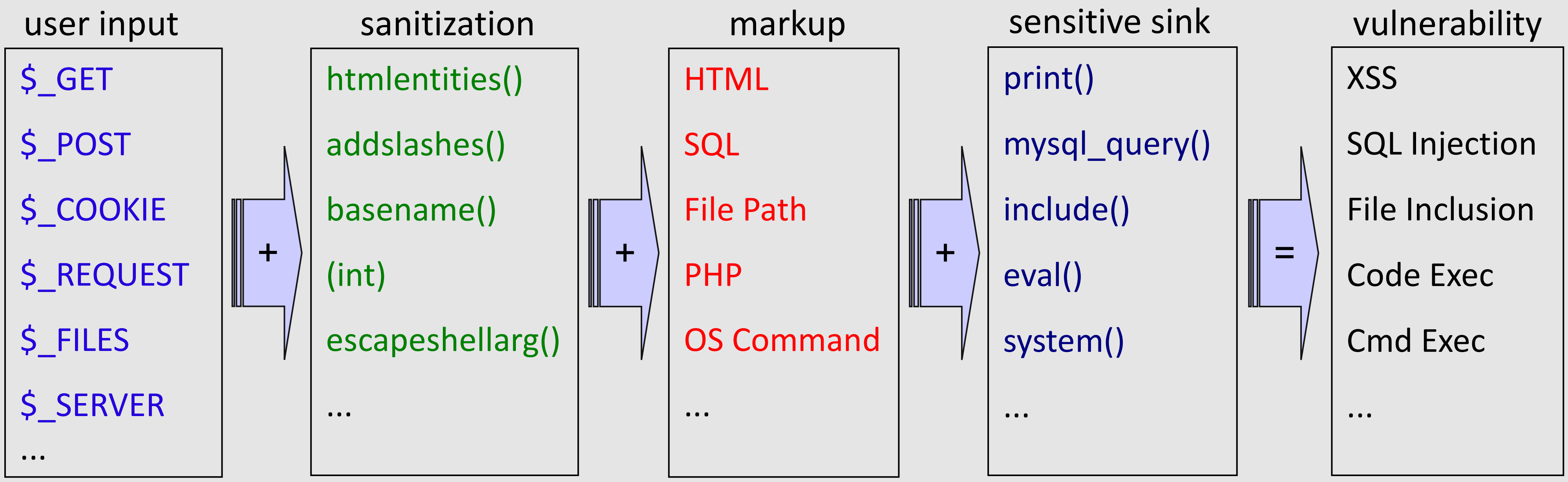
1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

1.4 Taint-style Vulnerabilities



Refined concept of previous prototype and current tools on the market

1.4 Taint-style Vulnerabilities



user input		sanitization		markup		sensitive sink		vulnerability
$_GET		htmlentities()		HTML		print()		XSS
$_POST		addslashes()		SQL		mysql_query()		SQL Injection
$_COOKIE		basename()		File Path		include()		File Inclusion
$_REQUEST	+	(int)	+	PHP	+	eval()	=	Code Exec
$_FILES		escapeshellarg()		OS Command		system()		Cmd Exec
$_SERVER		...		...		...		...
...								

Refined concept of new prototype

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

1.5 Lessons Learned

1. Its *easy* to built a static code analysis tool that detects **simple vulnerabilities**

1.5 Lessons Learned

1. Its *easy* to built a static code analysis tool that detects **simple vulnerabilities**
2. Its *challenging* to built a static code analysis tool that detects **sophisticated vulnerabilities**

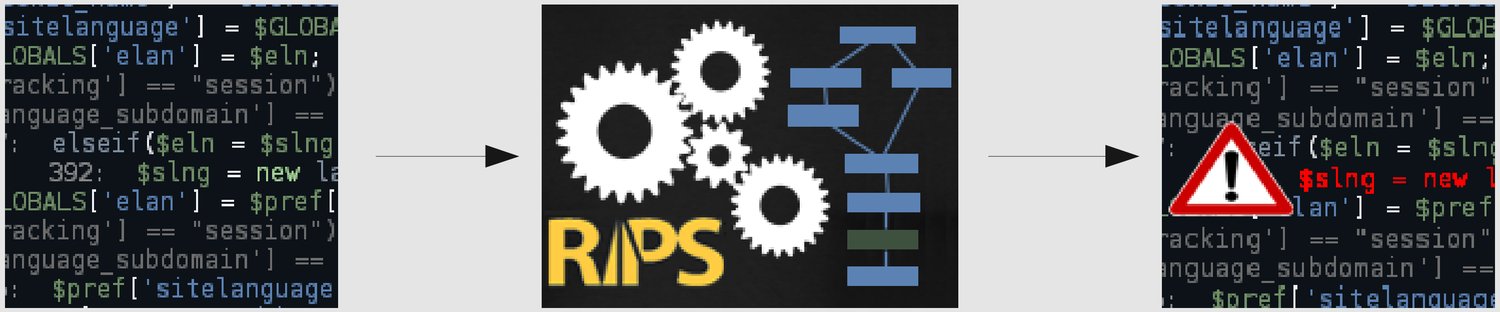
1.5 Lessons Learned

1. Its *easy* to built a static code analysis tool
   that detects **simple vulnerabilities**
2. Its *challenging* to built a static code analysis tool
   that detects **sophisticated vulnerabilities**
3. Its *hard* to built a static code analysis tool
   that detects **sophisticated vulnerabilities**
   in **large applications** with acceptable **performance**
   and a **low false positive** rate

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

RIPS 1.0

BadBank Demo Scan



Static Detection of Vulnerabilities in Modern PHP Applications

2. Static VS Dynamic Code Analysis

	Static analyze code without execution	**Dynamic** analyze code while execution
Code Coverage	full	Single execution path
Data Coverage	Compile-time data	Runtime data (valid for environment)
Decidability	Halting Problem	Real data

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

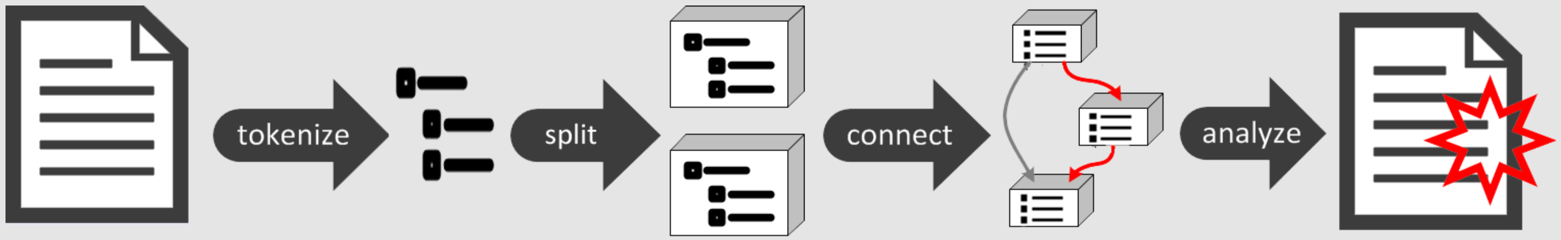
2. Static Code Analysis

	Static analyze code without execution	**Dynamic** analyze code while execution
Code Coverage	full	Single execution path
Data Coverage	Compile-time data	Runtime data (valid for environment)
Decidability	Halting Problem	Real data

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.1 Overview

- Load all PHP files
- Tokenize PHP code and build an Abstract Syntax Tree (AST)
- Split AST into Basic Blocks
- Connect Basic Blocks to a Control Flow Graph (CFG)
- Analyze data flow through CFG
- RIPS uses *block* and *function summaries*

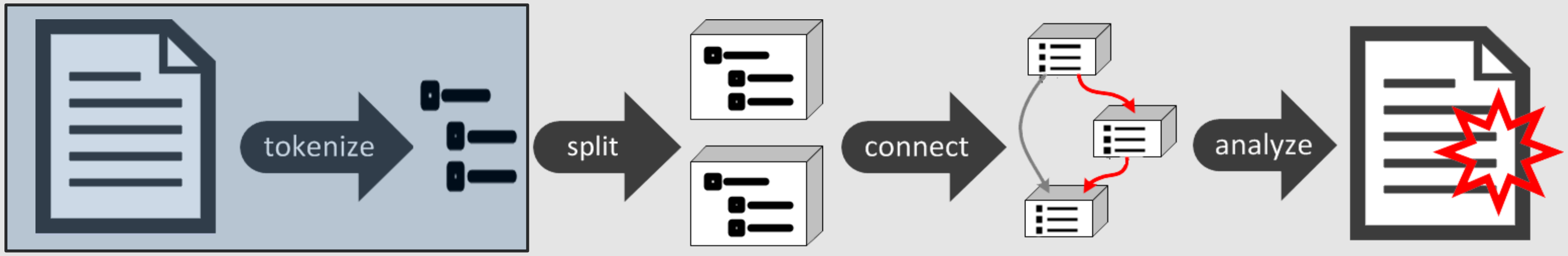


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.2 Abstract Syntax Tree

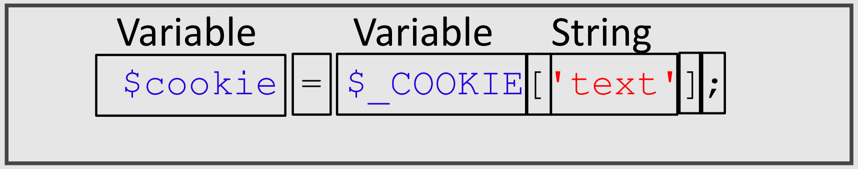
```
$cookie = $_COOKIE['text'];
```

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters

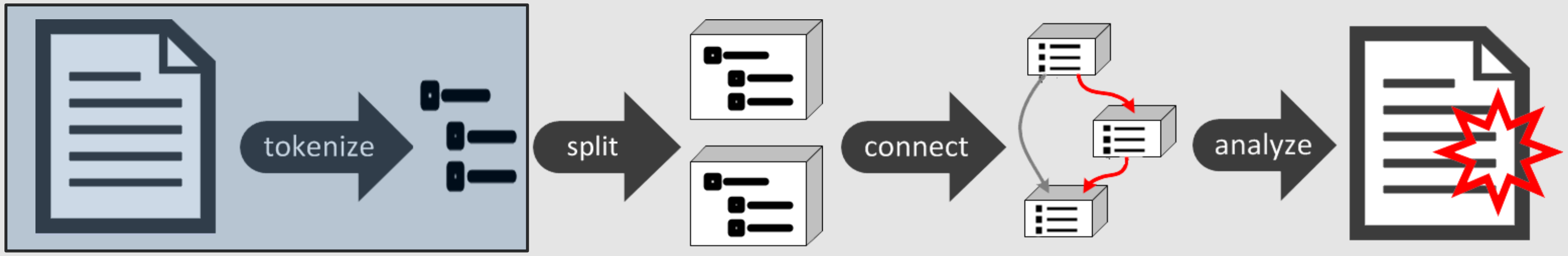


2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters

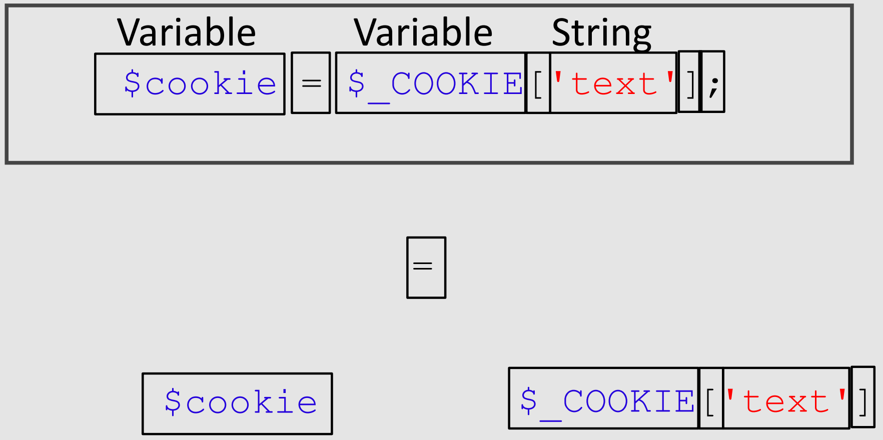


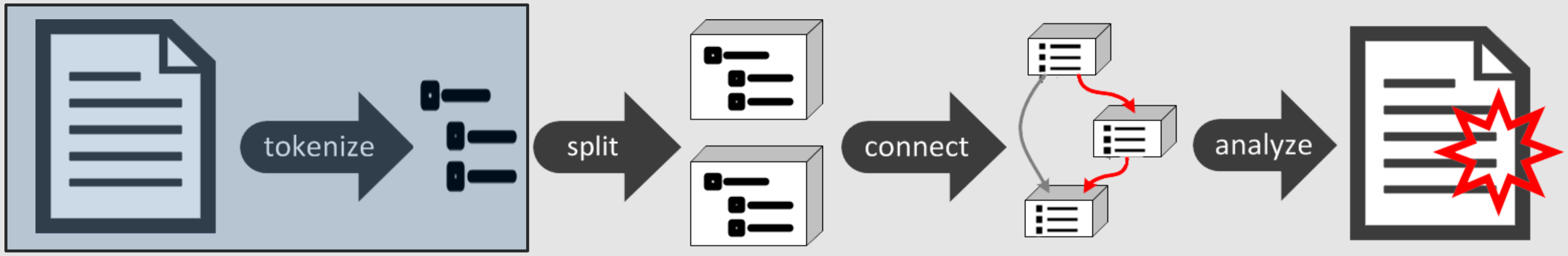
```
$cookie = $_COOKIE['text'];
```



2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters

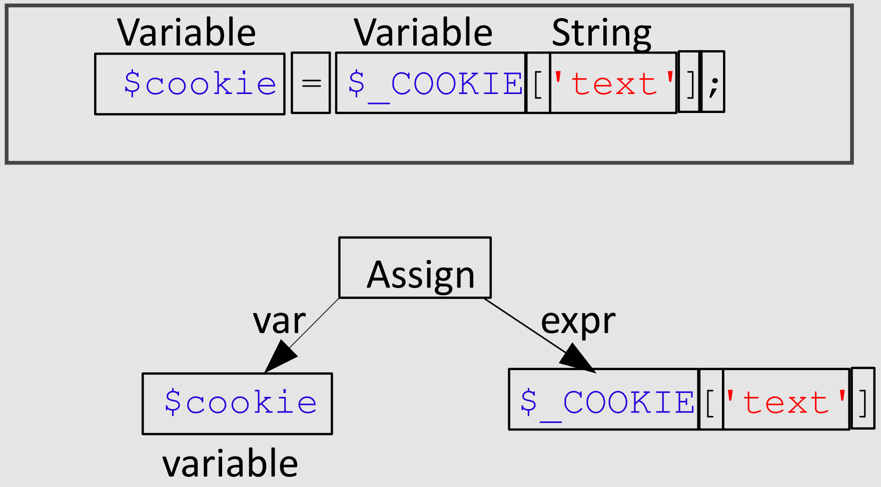


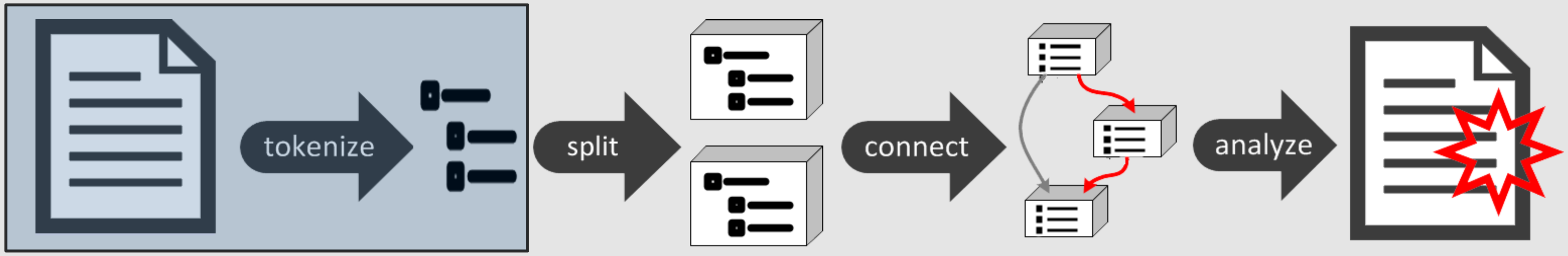


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters

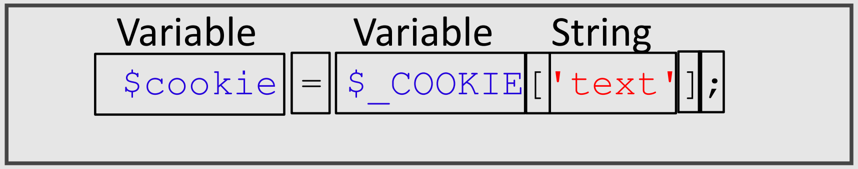


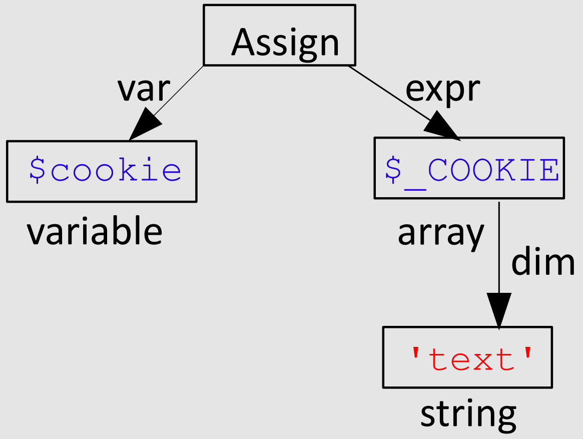


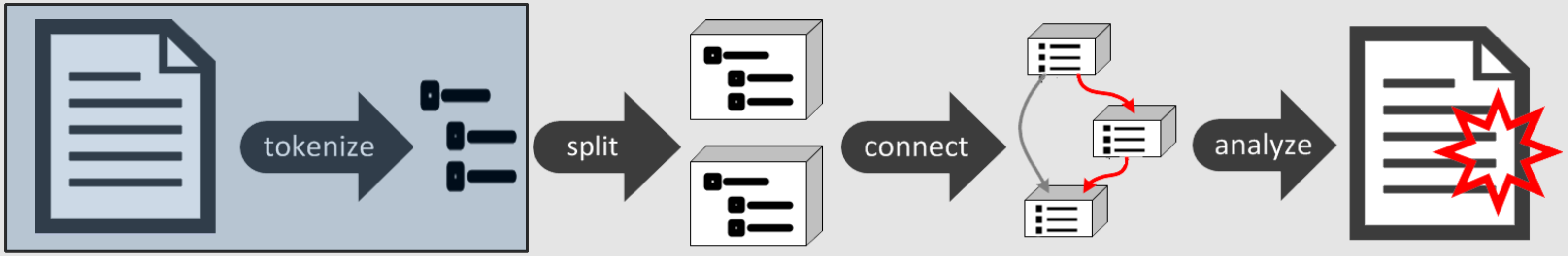
1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters



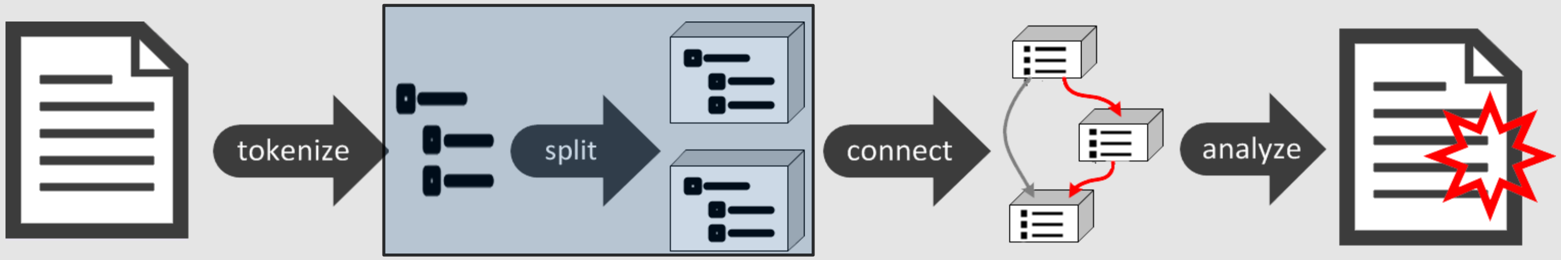




2.3 Basic Block

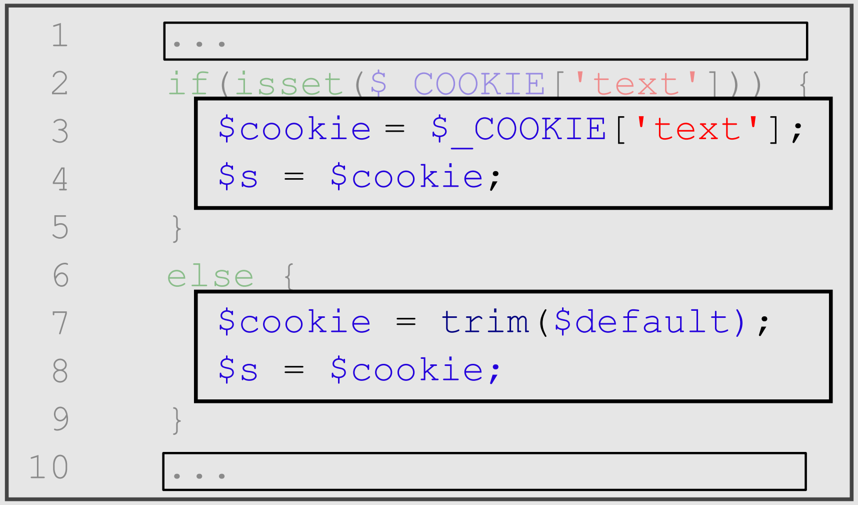
- Split AST at *jump nodes*
- *Single input, single output* code block
- Simulate data flow in basic block
- Precisely model PHP built-in features
- Summarize data flow

```
...
if(isset($_COOKIE['text'])) {
  $cookie = $_COOKIE['text'];
  $s = $cookie;
}
else {
  $cookie = trim($default);
  $s = $cookie;
}
...
```

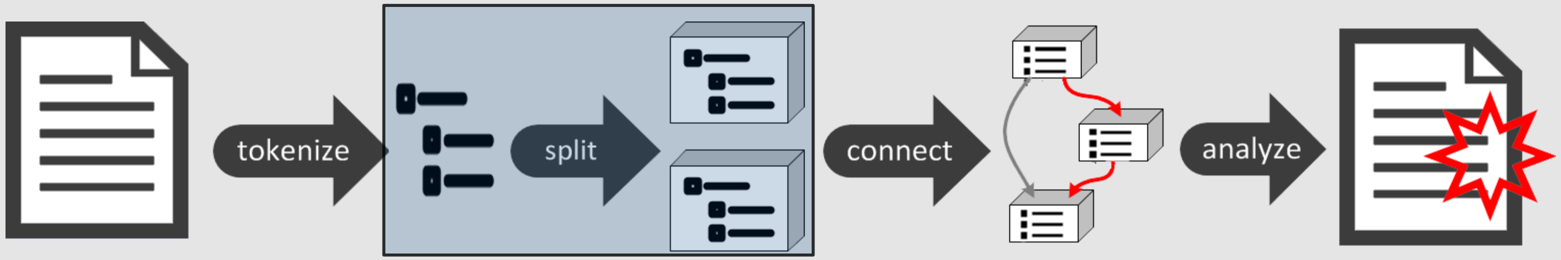


2.3 Basic Block

- Split AST at *jump nodes*
- *Single input, single output* code block
- Simulate data flow in basic block
- Precisely model PHP built-in features
- Summarize data flow



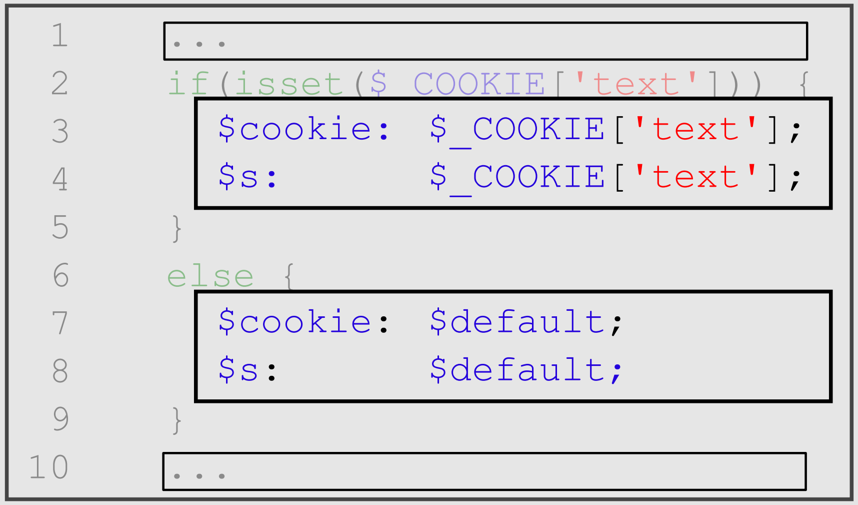
```
...
if(isset($_COOKIE['text'])) {
    $cookie = $_COOKIE['text'];
    $s  = $cookie;
}
else {
    $cookie  = trim($default);
    $s  = $cookie;
}
...
```



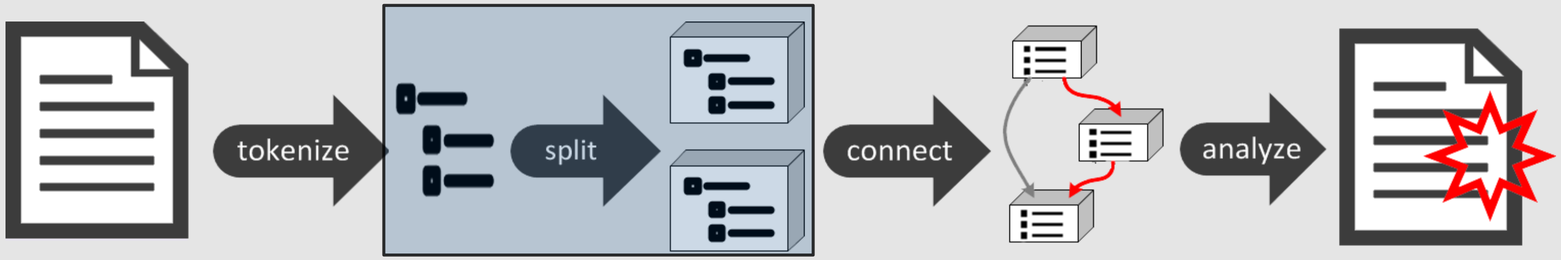
1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.3 Basic Block

- Split AST at *jump nodes*
- *Single input, single output* code block
- Simulate data flow in basic block
- Precisely model PHP built-in features
- Summarize data flow



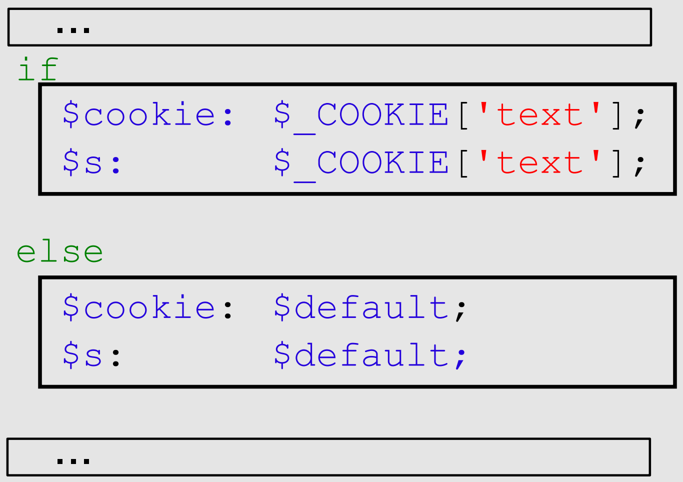
```
...
if(isset($_COOKIE['text'])) {
    $cookie:  $_COOKIE['text'];
    $s:       $_COOKIE['text'];
}
else {
    $cookie:  $default;
    $s:       $default;
}
...
```



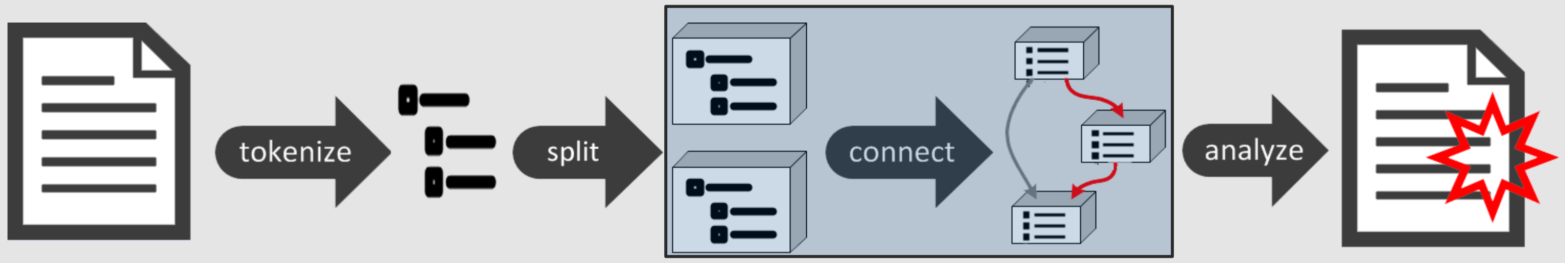
1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.4 Control Flow Graph

- Connect basic blocks to CFG
- Edges are jump conditions
- Represents all code paths
- Efficient data flow analysis on block summaries



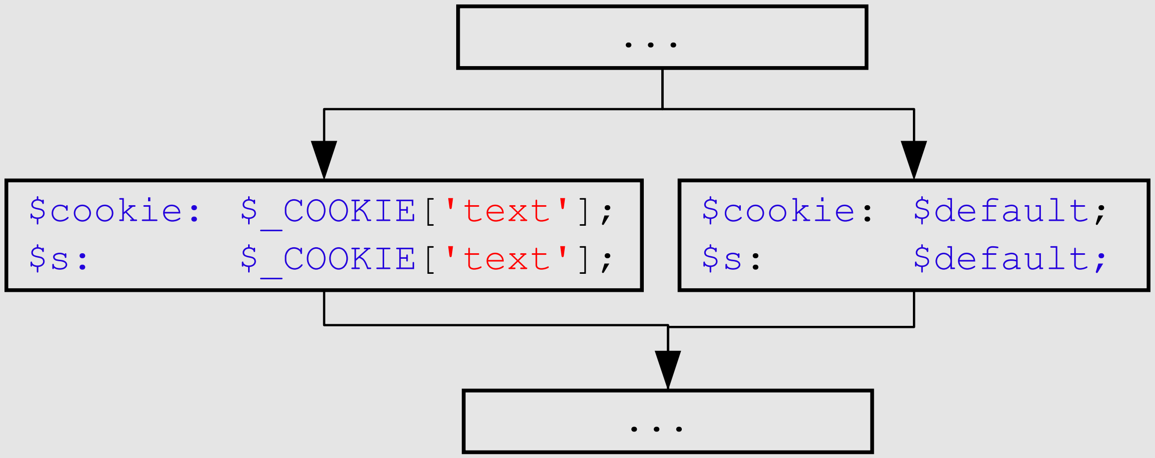
```
...
if
  $cookie:  $_COOKIE['text'];
  $s:       $_COOKIE['text'];

else
  $cookie:  $default;
  $s:       $default;

...
```

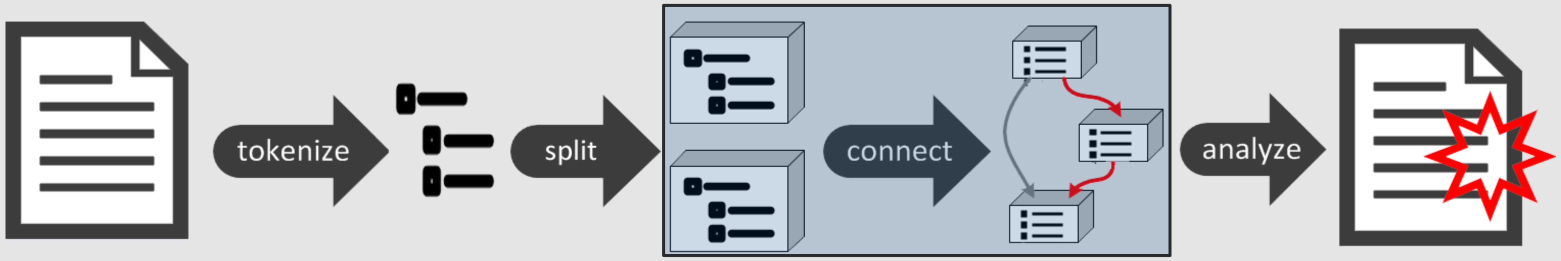


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.4 Control Flow Graph

- Connect basic blocks to CFG
- Edges are jump conditions
- Represents all code paths
- Efficient data flow analysis on block summaries

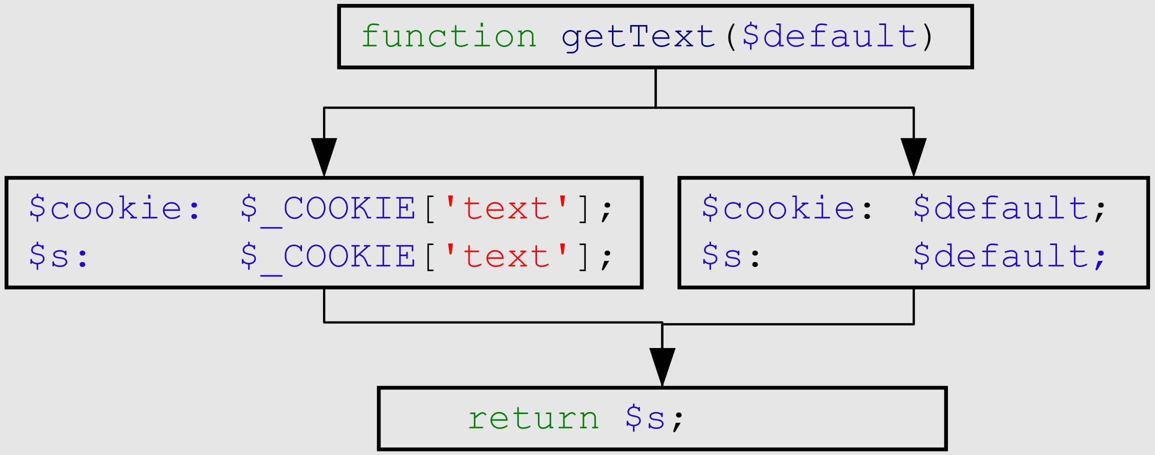




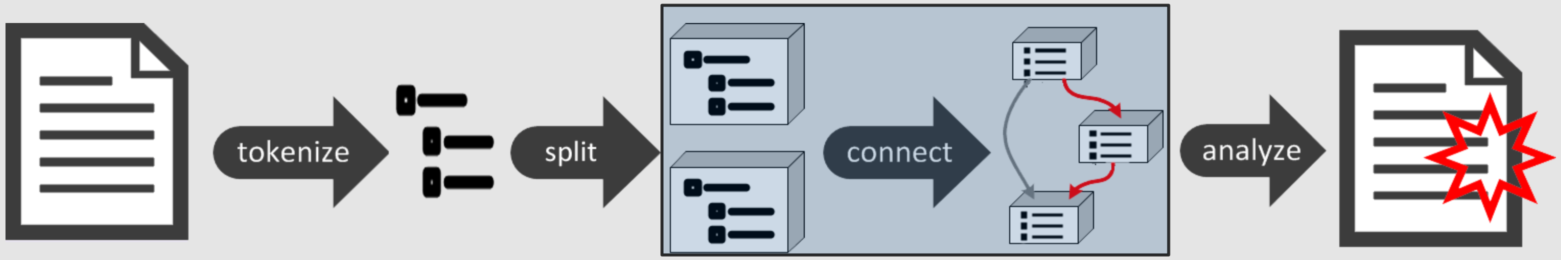
1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.5 Inter-procedural Analysis

- Call to user-defined function invokes intra-procedural analysis of this function
- Effects of function are stored in *function summary*

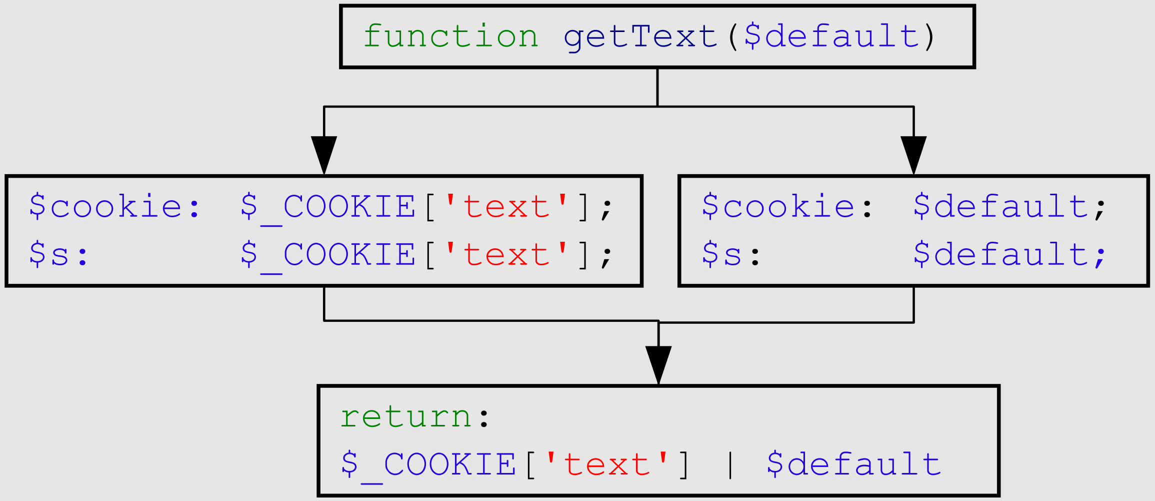


```
function getText($default)

$cookie:  $_COOKIE['text'];
$s:       $_COOKIE['text'];

$cookie:  $default;
$s:       $default;

return $s;
```

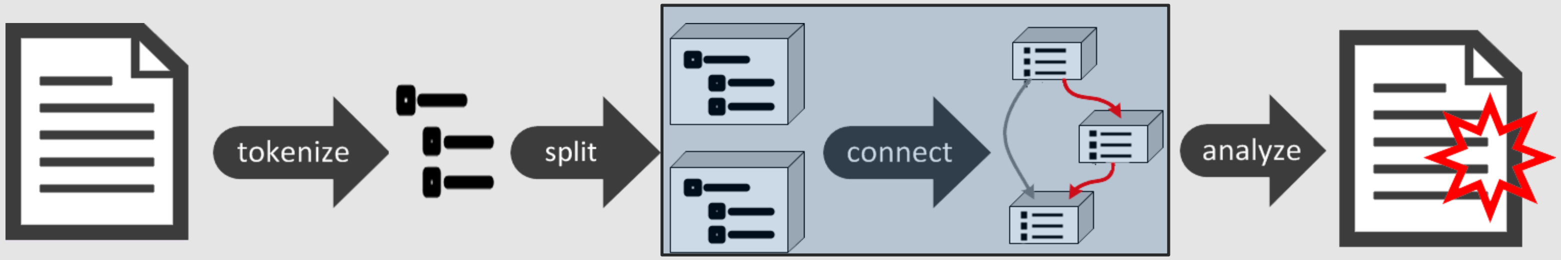


2.5 Inter-procedural Analysis

- Call to user-defined function invokes intra-procedural analysis of this function
- Effects of function are stored in *function summary*



```
function getText($default)

$cookie:  $_COOKIE['text'];
$s:       $_COOKIE['text'];

$cookie:  $default;
$s:       $default;

return:
$_COOKIE['text'] | $default
```

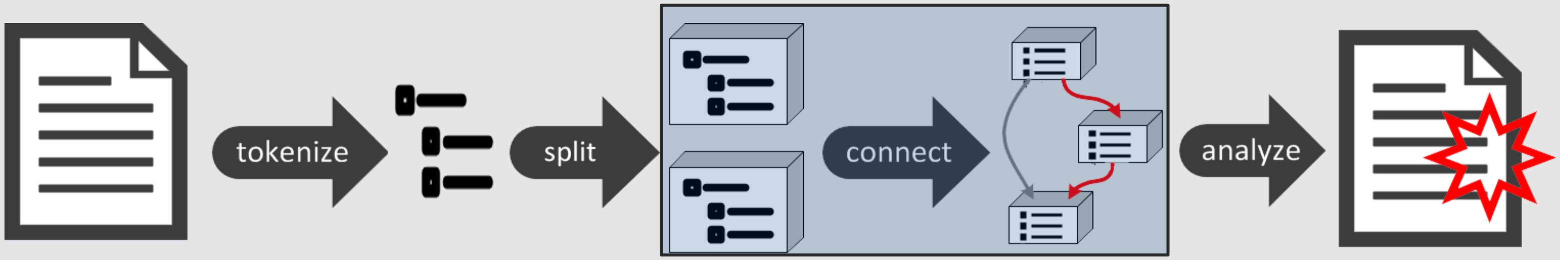


2.5 Inter-procedural Analysis

- Call to user-defined function invokes intra-procedural analysis of this function
- Effects of function are stored in *function summary*

```
function getText($arg1)
return: $_COOKIE['text'] | $arg1
```

```
...
$text = getText('foo');
...
```

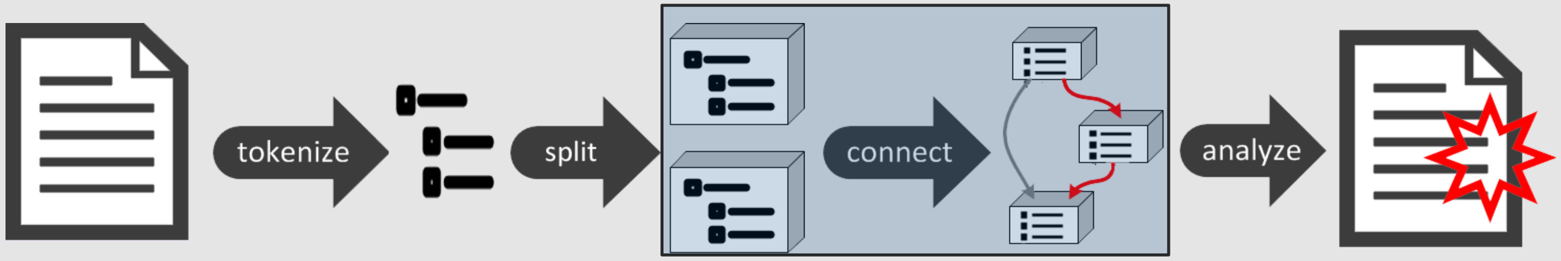


2.5 Inter-procedural Analysis

- Call to user-defined function invokes intra-procedural analysis of this function
- Effects of function are stored in *function summary*

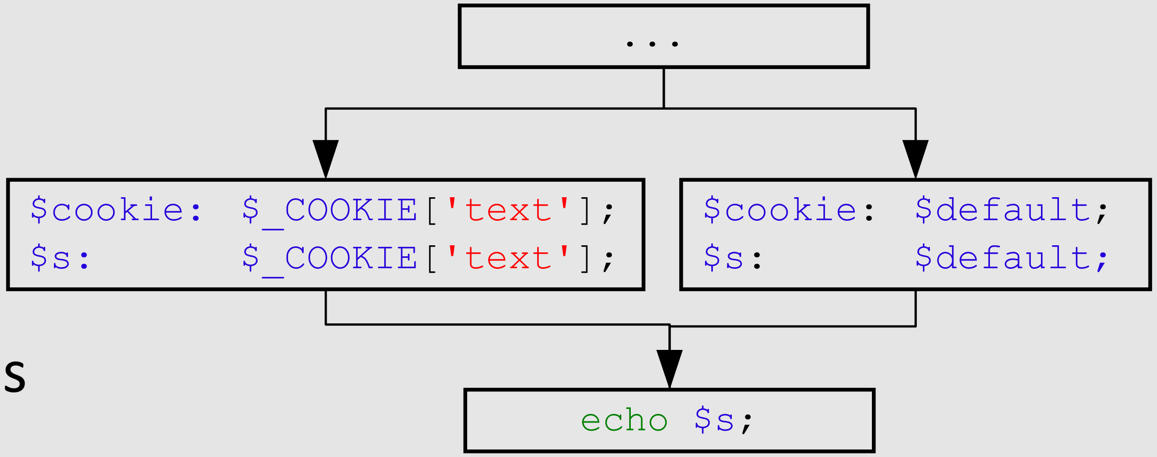
```
function getText($arg1)
return: $_COOKIE['text'] | $arg1
```

```
$text: $_COOKIE['text']|'foo'
```



2.6 Taint Analysis

- Identify a configured set of sensitive sinks
- Resolve sensitive arguments from previous block summaries
- Issue vulnerabilty report if argument is resolved to a source

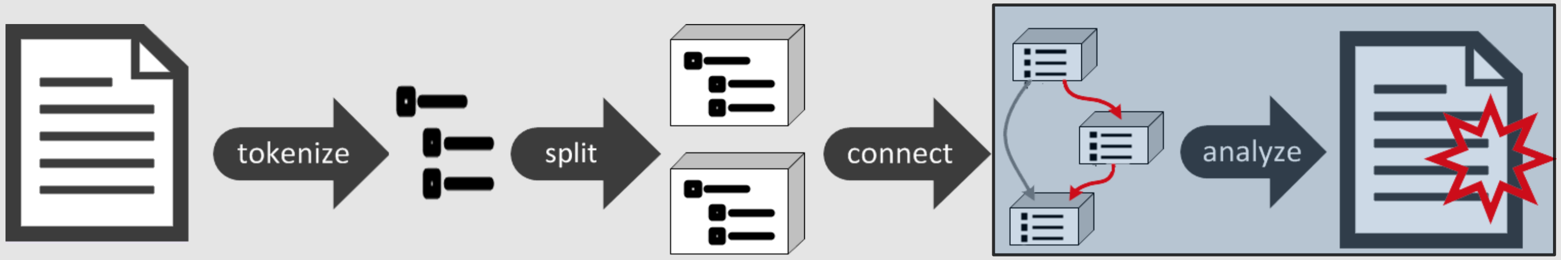


```
...
```

```
$cookie:  $_COOKIE['text'];
$s:       $_COOKIE['text'];
```

```
$cookie:  $default;
$s:       $default;
```

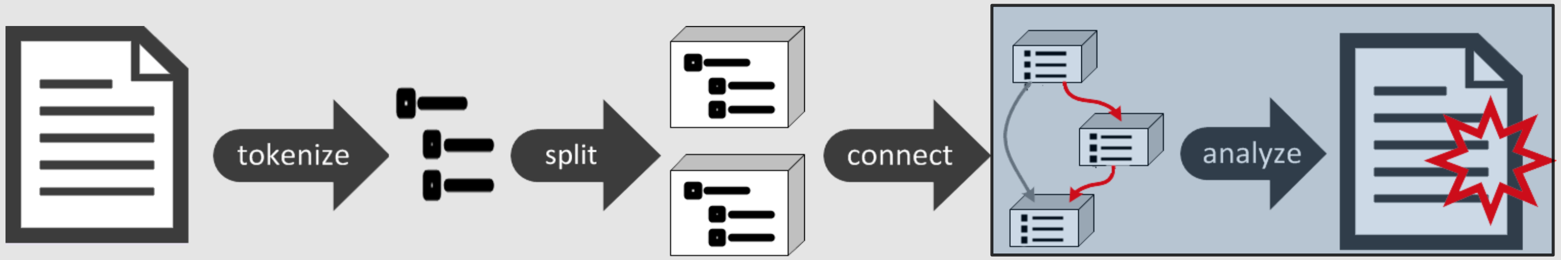
```
echo $s;
```



2.7 Context-sensitive String Analysis

- Resolve all strings (markup) of a sensitive argument
- Replace sources with a placeholder
- Invoke markup parser
- Evaluate sanitization of each source regarding its markup context

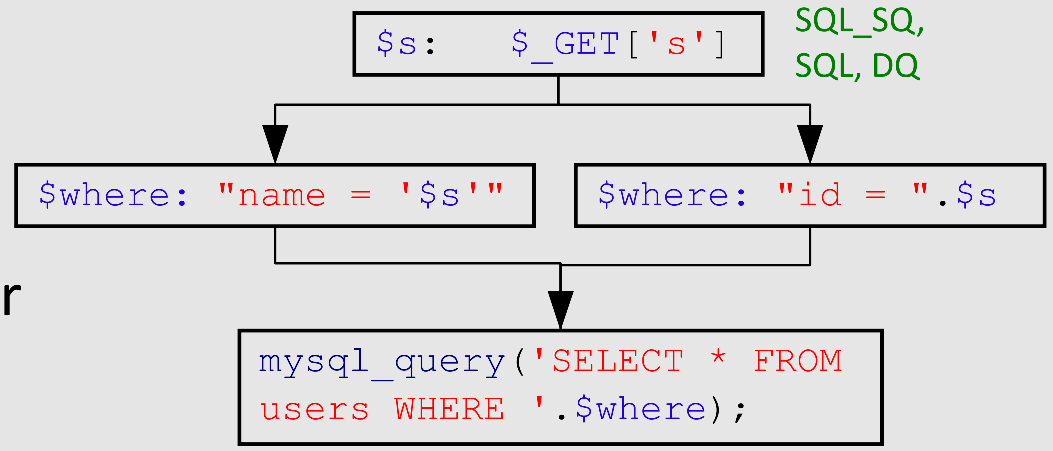
```php
$s = addslashes($_GET['s']);
if($mode == 1) {
   $where = "name = '$s'";
}
else {
   $where = "id = $s";
}
mysql_query('SELECT *
   FROM users ' . $where);
```



1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.7 Context-sensitive String Analysis

- Resolve all strings (markup) of a sensitive argument
- Replace sources with a placeholder
- Invoke markup parser
- Evaluate sanitization of each source regarding its markup context



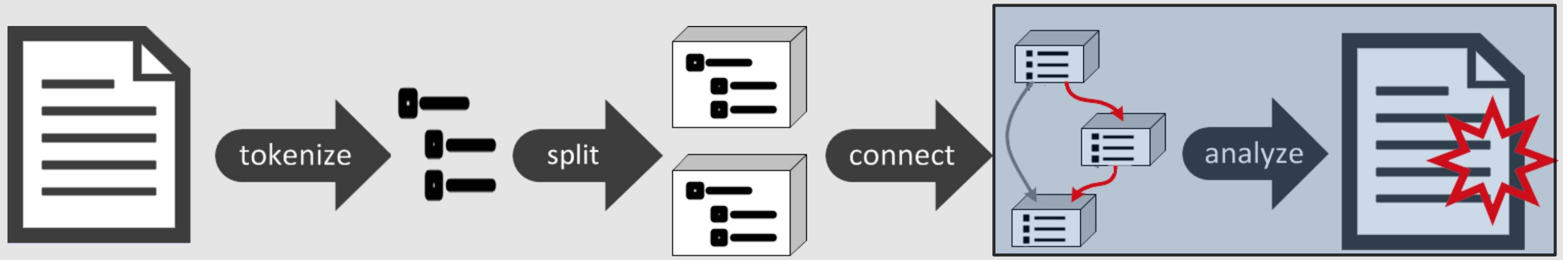
```
$s:    $_GET['s']
```

SQL_SQ, SQL, DQ

```
$where: "name = '$s'"
```

```
$where: "id = ".$s
```

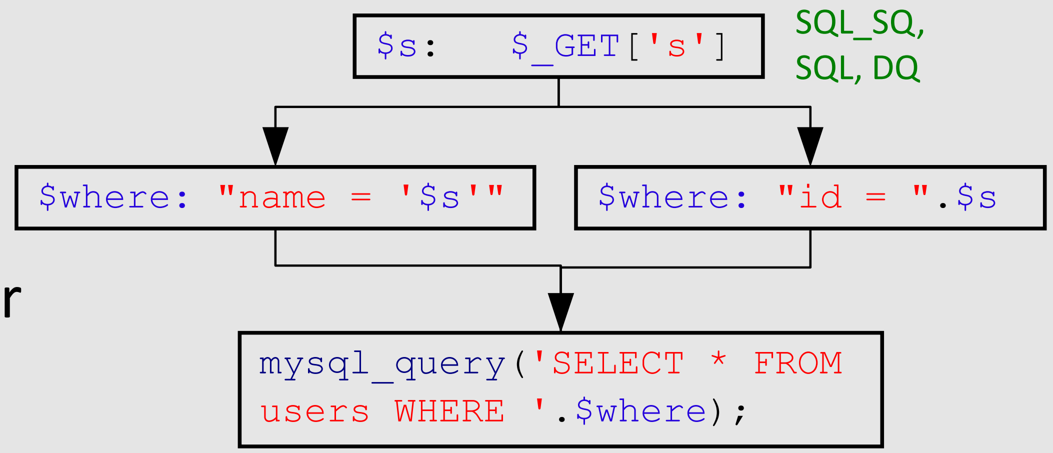
```
mysql_query('SELECT * FROM
users WHERE '.$where);
```



1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.7 Context-sensitive String Analysis

- Resolve all strings (markup) of a sensitive argument
- Replace sources with a placeholder
- Invoke markup parser
- Evaluate sanitization of each source regarding its markup context

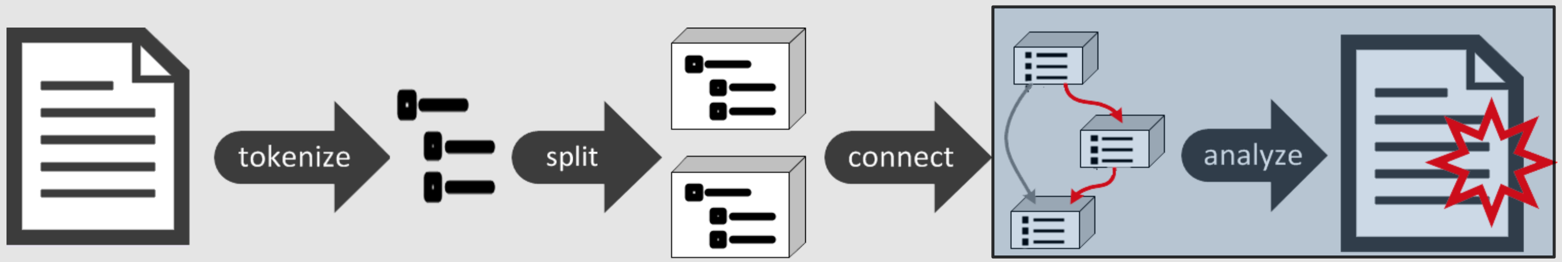


```
$s:    $_GET['s']
```

SQL_SQ,
SQL, DQ

```
$where: "name = '$s'"
$where: "id = ".$s
mysql_query('SELECT * FROM
users WHERE '.$where);
```

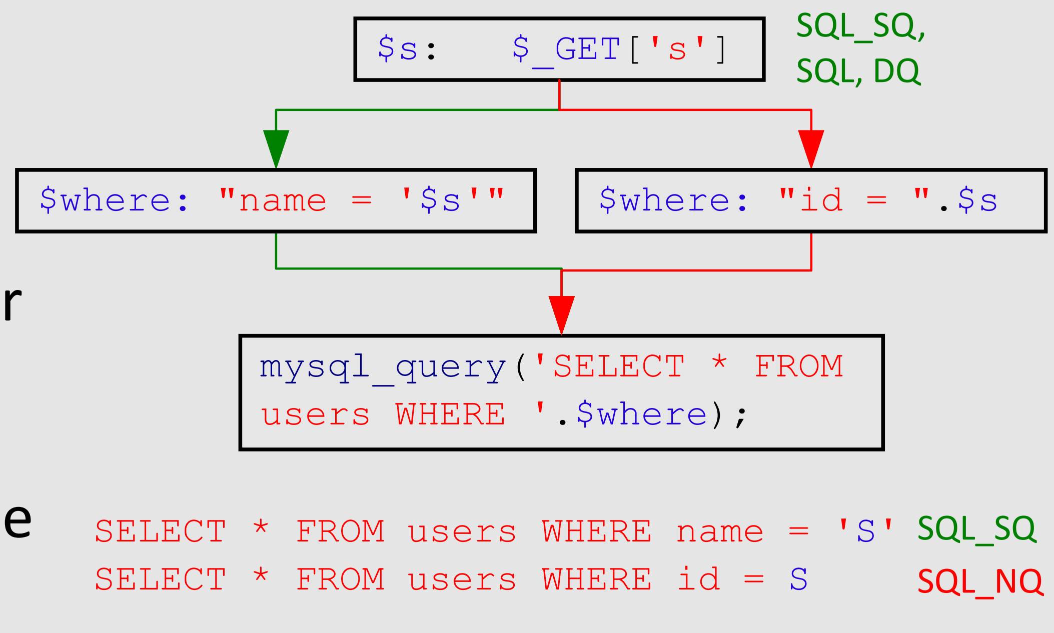
```
SELECT * FROM users WHERE name = 'S'  SQL_SQ
SELECT * FROM users WHERE id = S      SQL_NQ
```



1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

2.7 Context-sensitive String Analysis

- Resolve all strings (markup) of a sensitive argument
- Replace sources with a placeholder
- Invoke markup parser
- Evaluate sanitization of each source regarding its markup context



```
$s:    $_GET['s']
```

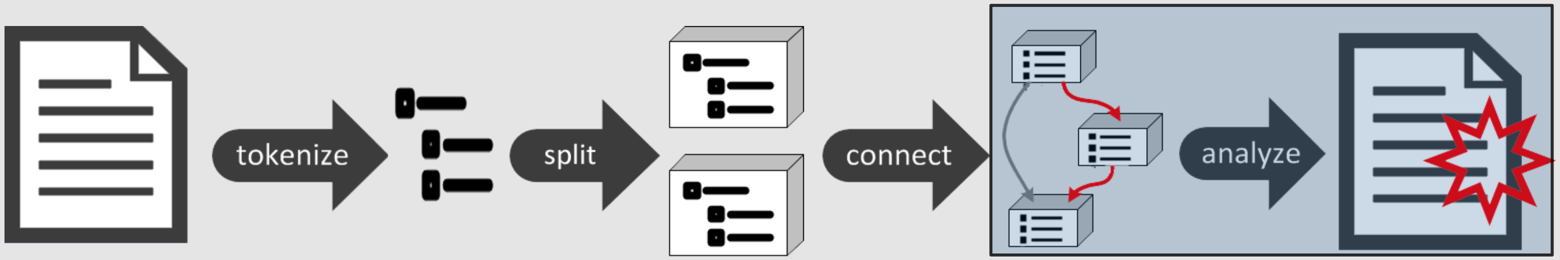
SQL_SQ, SQL, DQ

```
$where: "name = '$s'"
$where: "id = ".$s
```

```
mysql_query('SELECT * FROM
users WHERE '.$where);
```

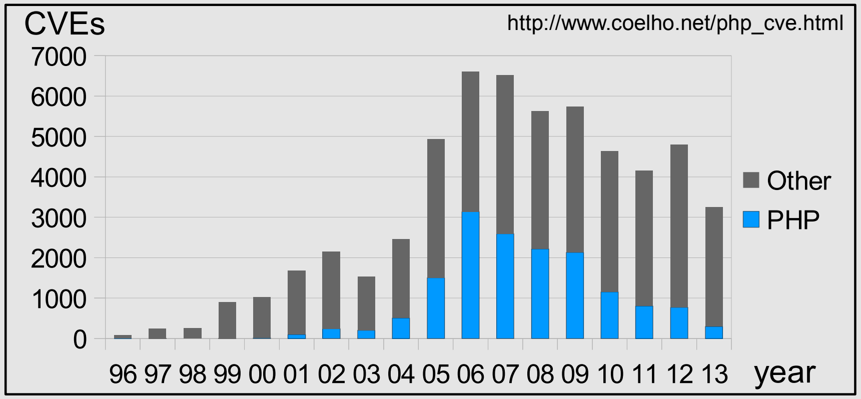
```
SELECT * FROM users WHERE name = 'S'
SELECT * FROM users WHERE id = S
```

SQL_SQ
SQL_NQ



3. Detecting *Modern* Vulnerabilities

- Taint-style concept remains: source → sink
- Data flow is more complex
  - Dynamic language features
  - Array Handling
  - PHP built-in features
  - **Input sanitization + validation**
  - Object-Oriented Programming
- More LOC (100k-200k)



1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.1 Analyze Exceptional Sources

- Often sanitized: $_GET, $_POST, and $_REQUEST
- Often overlooked: $_SERVER, $_FILES, and $_COOKIE
- Examples:

PHP Source	Developer Assumption	Exploit Example
$_SERVER['PHP_SELF']	*/index.php*	*/index.php/"><svg+onload=alert(1)>*
$_SERVER['REQUEST_URI']	*/index.php?payload=%22%27*	GET /?a='or(1)=1-- HTTP/1.0 or IE
$_SERVER['HTTP_HOST']	localhost	Host: ' or (1)=1-- -
$_FILES['picture']['name']	alphanumeric.jpg	'or (1)=1-- -.jpg

- RIPS analyzes sources context-sensitively

3.2 Precise Array Handling

- PHP superglobals ($_GET, $_POST, etc.) are arrays
- RIPS analyzes built-ins precisely
- Array keys are tainted too (Drupal)

```
SugarCRM 6.5.18                          /include/export_utils.php
function export($type, $records) {
   $records = explode(',', $records);
   $records = "'" . implode("','", $records) . "'";
   $where = "{$focus->table_name}.id in ($records)";
}
export(clean($_REQUEST['module']), $_REQUEST['uid']);
```

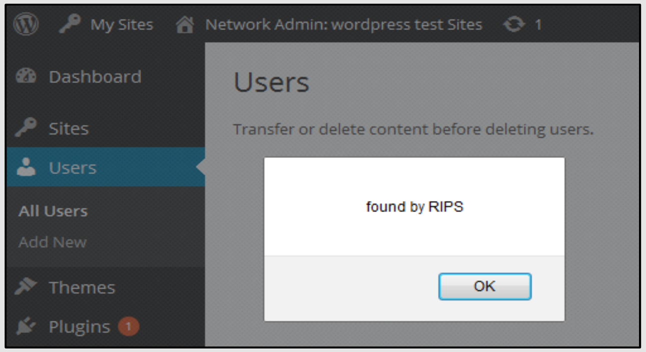
1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.2 Precise Array Handling

- PHP superglobals ($_GET, $_POST, etc.) are arrays
- RIPS analyzes built-ins precisely
- Array keys are tainted too (Drupal)

```
SugarCRM 6.5.18                    /include/export_utils.php
function export($type, $records) {
   $records = explode(',', $records);
   $records = "'" . implode("','", $records) . "'";
   $where = "{$focus->table_name}.id in ($records)";
}
export(clean($_REQUEST['module']), $_REQUEST['uid']);
```

```
Wordpress 4.0.1              /wp-admin/network/users.php
function confirm_delete_users($users) {
   foreach($_POST['allusers'] as $key => $val) {
      echo "<input name='user[]' value='{$val}'/>\n";
   }
}
allusers[0]='><script>alert('found by RIPS')</script>
```



Users

Transfer or delete content before deleting users.

found by RIPS

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.3 Object-Oriented Programming

Assist *backwards-directed* data flow analysis with *forwards-directed* data propagation

osCommerce 2.3.4 /admin/backup.php

```php
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
tep_db_query("insert into " . TABLE_CONFIG . " values
          ('DB_RESTORE', '" . $read_from . "', '6', '')");
```

3.3 Object-Oriented Programming

Assist *backwards-directed* data flow analysis with *forwards-directed* data propagation

osCommerce 2.3.4 /admin/backup.php

```
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
tep_db_query("insert into " . TABLE_CONFIG . " values
         ('DB_RESTORE', '" . $read_from . "', '6', '')");
```

3.3 Object-Oriented Programming

Assist *backwards-directed* data flow analysis with *forwards-directed* data propagation

```
osCommerce 2.3.4                                            /admin/backup.php
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
tep_db_query("insert into " . TABLE_CONFIG . " values
           ('DB_RESTORE', '" . $read_from . "', '6', '')");
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.3 Object-Oriented Programming

Assist *backwards-directed* data flow analysis with *forwards-directed* data propagation

osCommerce 2.3.4 /admin/backup.php

```
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
tep_db_query("insert into " . TABLE_CONFIG . " values
          ('DB_RESTORE', '" . $read_from . "', '6', '')");
```

3.3 Object-Oriented Programming

Assist *backwards-directed* data flow analysis with *forwards-directed* data propagation

```
osCommerce 2.3.4                               /admin/backup.php
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
tep_db_query("insert into " . TABLE_CONFIG . " values
          ('DB_RESTORE', '" . $read_from . "', '6', '')");
```

3.3 Object-Oriented Programming

Assist *backwards-directed* data flow analysis with *forwards-directed* data propagation

osCommerce 2.3.4 /admin/backup.php

```
  1 class Upload {
  2     function parse($name) {
  3         $this->set_filename($_FILES[$name]['name']);
  4     }
  5     function set_filename($filename) {
  6         $this->filename = $filename;
  7     }
  8 }

199 $sql_file = new Upload();
201 if ($sql_file->parse('sql_file') == true) {
202     $read_from = $sql_file->filename;
203 }
273 tep_db_query("insert into " . TABLE_CONFIG . " values
             ('DB_RESTORE', '" . $read_from . "', '6', '')");
```

3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

Pligg CMS 2.0.2 /admin/admin_categories.php

```
$parent = substr(addslashes($_REQUEST['parent']),9,100);
$sql = "update ".table_categories." set category_parent
         = " . $parent . " where category_id=" . $id . ";";
$db->query($sql);
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

```
Pligg CMS 2.0.2                  /admin/admin_categories.php
$parent = substr(addslashes($_REQUEST['parent']),9,100);
$sql = "update ".table_categories." set category_parent
         = " . $parent . " where category_id=" . $id . ";";
$db->query($sql);
```

12345678901234567890'

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

```
Pligg CMS 2.0.2                      /admin/admin_categories.php
$parent = substr(addslashes($_REQUEST['parent']),9,100);
$sql = "update ".table_categories." set category_parent
         = " . $parent . " where category_id=" . $id . ";";
$db->query($sql);
```

1234567890123456789O\'

3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

```
Pligg CMS 2.0.2                    /admin/admin_categories.php
$parent = substr(addslashes($_REQUEST['parent']),9,100);
$sql = "update ".table_categories." set category_parent
         = " . $parent . " where category_id=" . $id . ";";
$db->query($sql);
```

12345678901234567890\

3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

```
Pligg CMS 2.0.2                              /admin/admin_categories.php
$parent = substr(addslashes($_REQUEST['parent']),9,100);
$sql = "update ".table_categories." set category_parent
         = " . $parent . " where category_id=" . $id . ";";
$db->query($sql);
```

1, category_title=(SELECT password ...)

3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

Pligg CMS 2.0.2 /admin/admin_categories.php

```
$parent = substr(addslashes($_REQUEST['parent']),9,100);
$sql = "update ".table_categories." set category_parent
      = " . $parent . " where category_id=" . $id . ";";
$db->query($sql);
```

Mambo CMS 4.6.2 /includes/cmtclasses.php

```
$mosmsg = mosGetParam($_REQUEST, 'mosmsg', '');
$mosmsg = addslashes($mosmsg);
echo "\n<div class=\"message\">$mosmsg</div>";
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

```
Pligg CMS 2.0.2                         /admin/admin_categories.php
233 $parent = substr(addslashes($_REQUEST['parent']),9,100);
245 $sql = "update ".table_categories." set category_parent
         = " . $parent . " where category_id=" . $id . ";";
246 $db->query($sql);
```

```
Mambo CMS 4.6.2                             /includes/cmtclasses.php
88  $mosmsg = mosGetParam($_REQUEST, 'mosmsg', '');
90  $mosmsg = addslashes($mosmsg);
91  echo "\n<div class=\"message\">$mosmsg</div>";
```

```
Couch CMS 1.4                       /couch/includes/fileuploader/io.php
603 echo '<script type="text/javascript">';
621 $rpl = array( '\\' => '\\\\', '"' => '\\"' );
623 echo 'OnUploadCompleted("' . strtr($msg, $rpl) . '");';
623 echo '</script>';
```

3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

Pligg CMS 2.0.2 /admin/admin_categories.php

```
$parent = substr(addslashes($_REQUEST['parent']),9,100);
$sql = "update ".table_categories." set category_parent
      = " . $parent . " where category_id=" . $id . ";";
$db->query($sql);
```

Mambo CMS 4.6.2 /includes/cmtclasses.php

```
$mosmsg = mosGetParam($_REQUEST, 'mosmsg', '');
$mosmsg = addslashes($mosmsg);
echo "\n<div class=\"message\">$mosmsg</div>";
```

Couch CMS 1.4 /couch/includes/fileuploader/io.php

```
echo '<script type="text/javascript">';
$rpl = array( '\\' => '\\\\', '"' => '\\"' );
echo 'OnUploadCompleted("' . strtr($msg, $rpl) . '");';
echo '</script>';
```

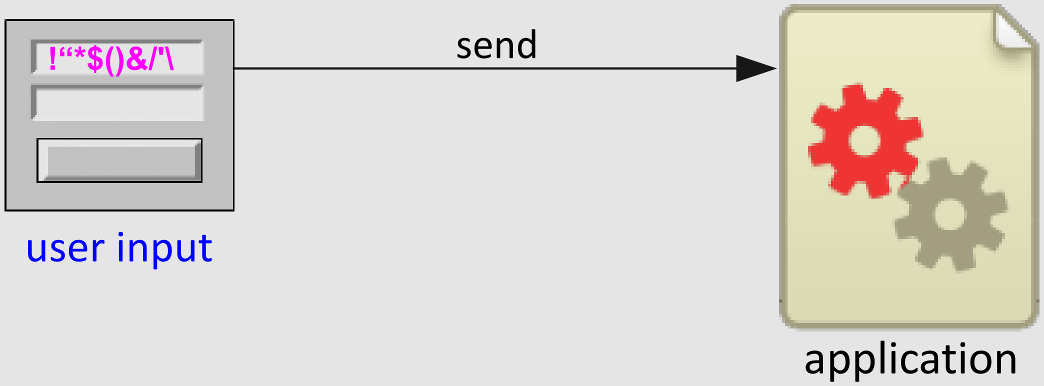
```
</script><script>alert(1)</script>
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.5 Second-Order Vulnerabilities

„First-Order“ SQL Injection:

```
$name  = $_POST['name'];
$pwd   = md5($_POST['pwd']);
$query = "INSERT INTO users VALUES('$name','$pwd')";
$result = mysql_query($query);
```

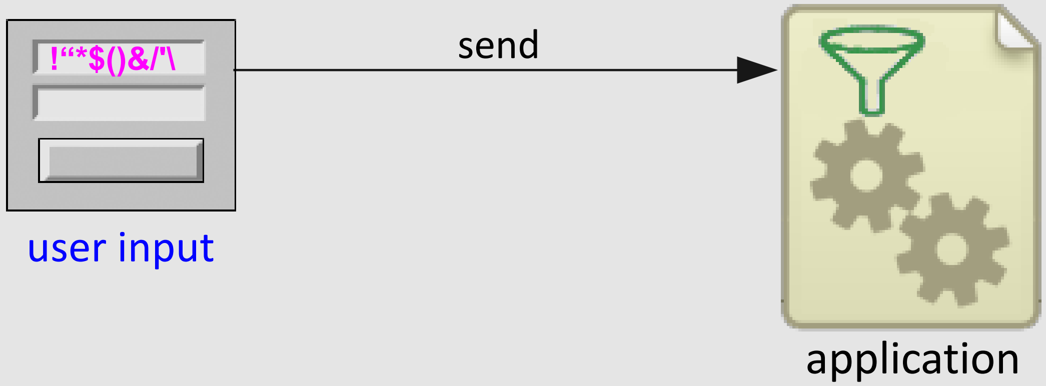


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.5 Second-Order Vulnerabilities

„First-Order“ SQL Injection (sanitized):

```
$name = mysql_real_escape_string($_POST['name']);
$pwd  = md5($_POST['pwd']);
$query = "INSERT INTO users VALUES('$name','$pwd')";
$result = mysql_query($query);
```

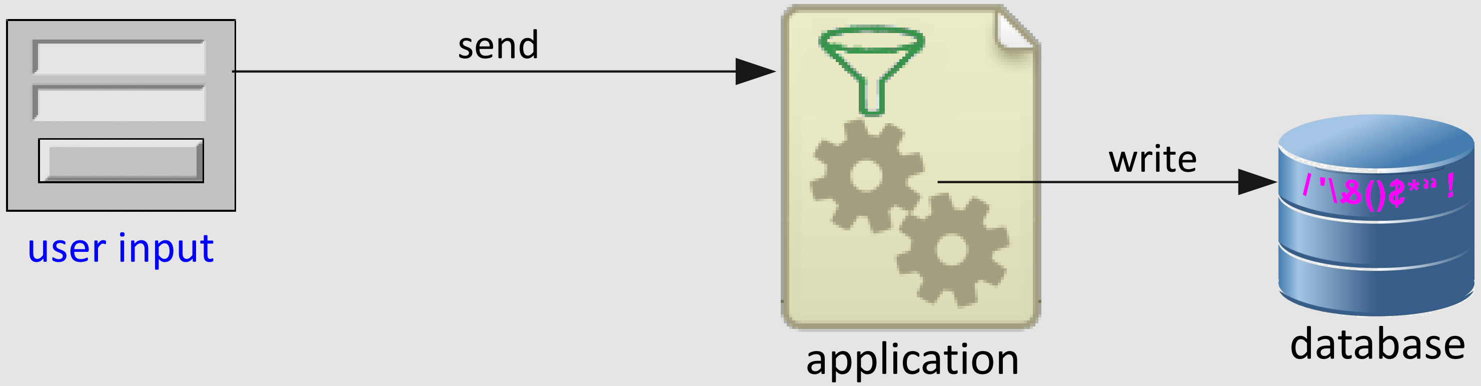


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.5 Second-Order Vulnerabilities

Database Write:

```
$name  = mysql_real_escape_string($_POST['name']);
$pwd   = md5($_POST['pwd']);
$query = "INSERT INTO users VALUES('$name','$pwd')";
$result = mysql_query($query);
```

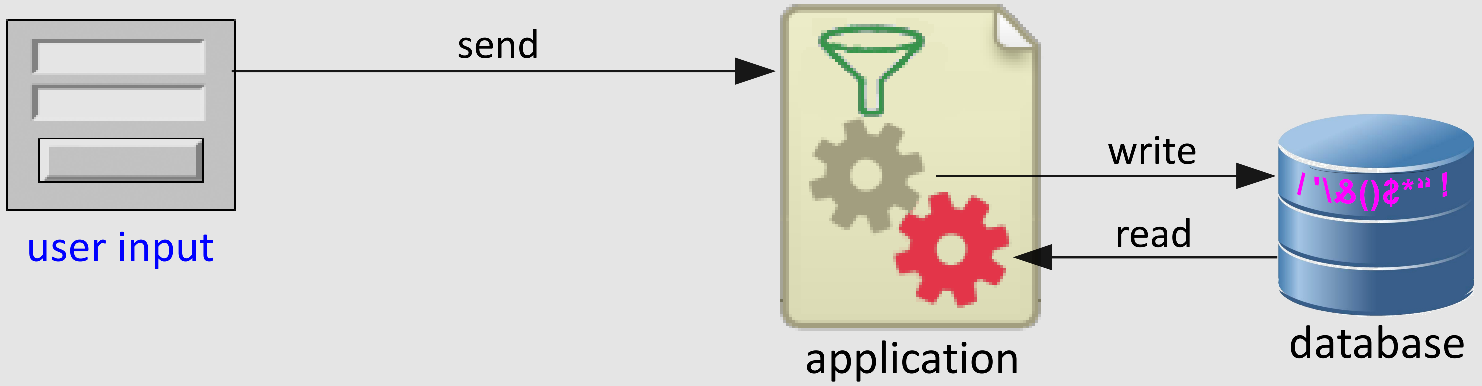


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.5 Second-Order Vulnerabilities

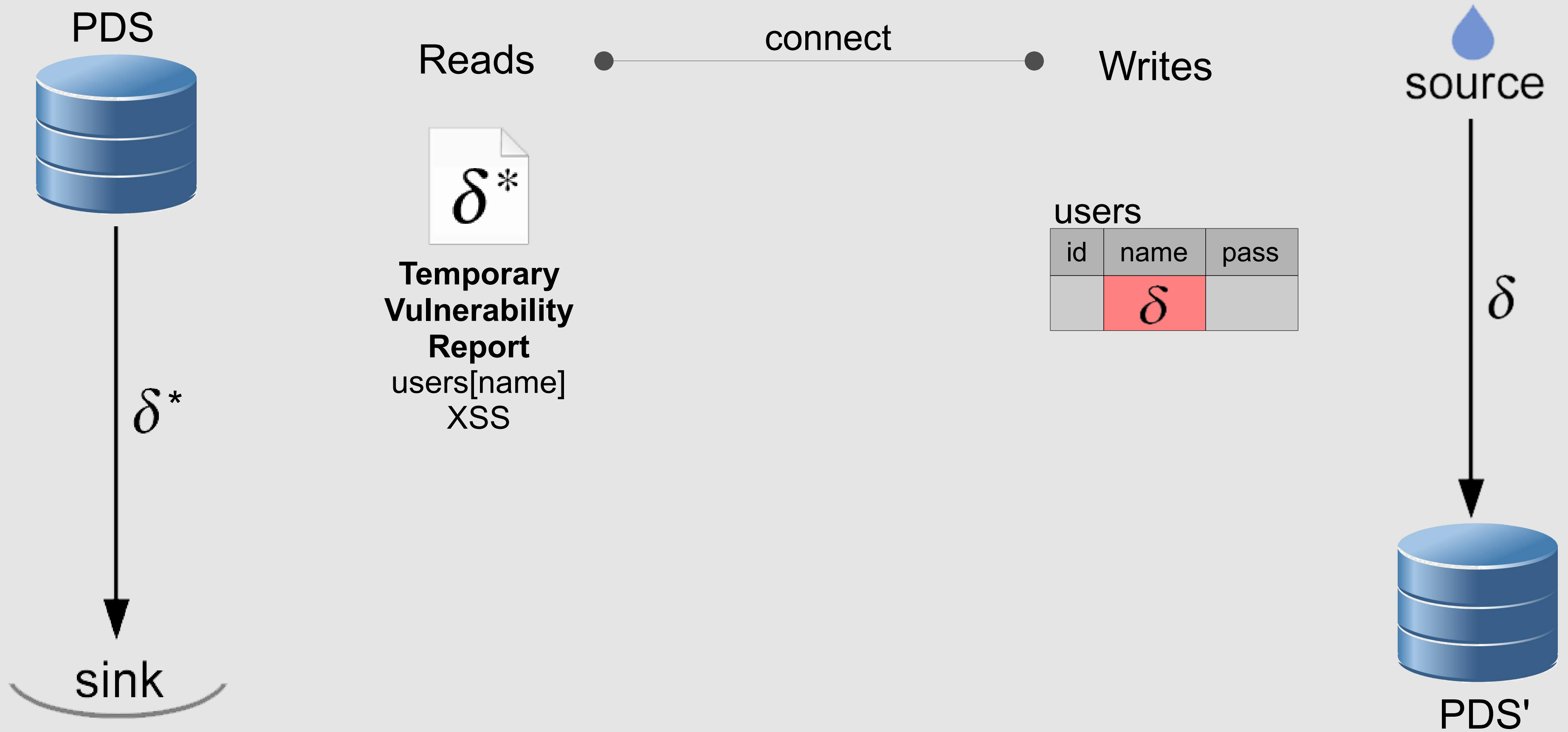
Database Read:

```
$query = "SELECT * FROM users WHERE id = 1";
$result = mysql_query($query);
$user = mysql_fetch_assoc($result);
echo $user['name'];
```



1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.5 Second-Order Vulnerabilities



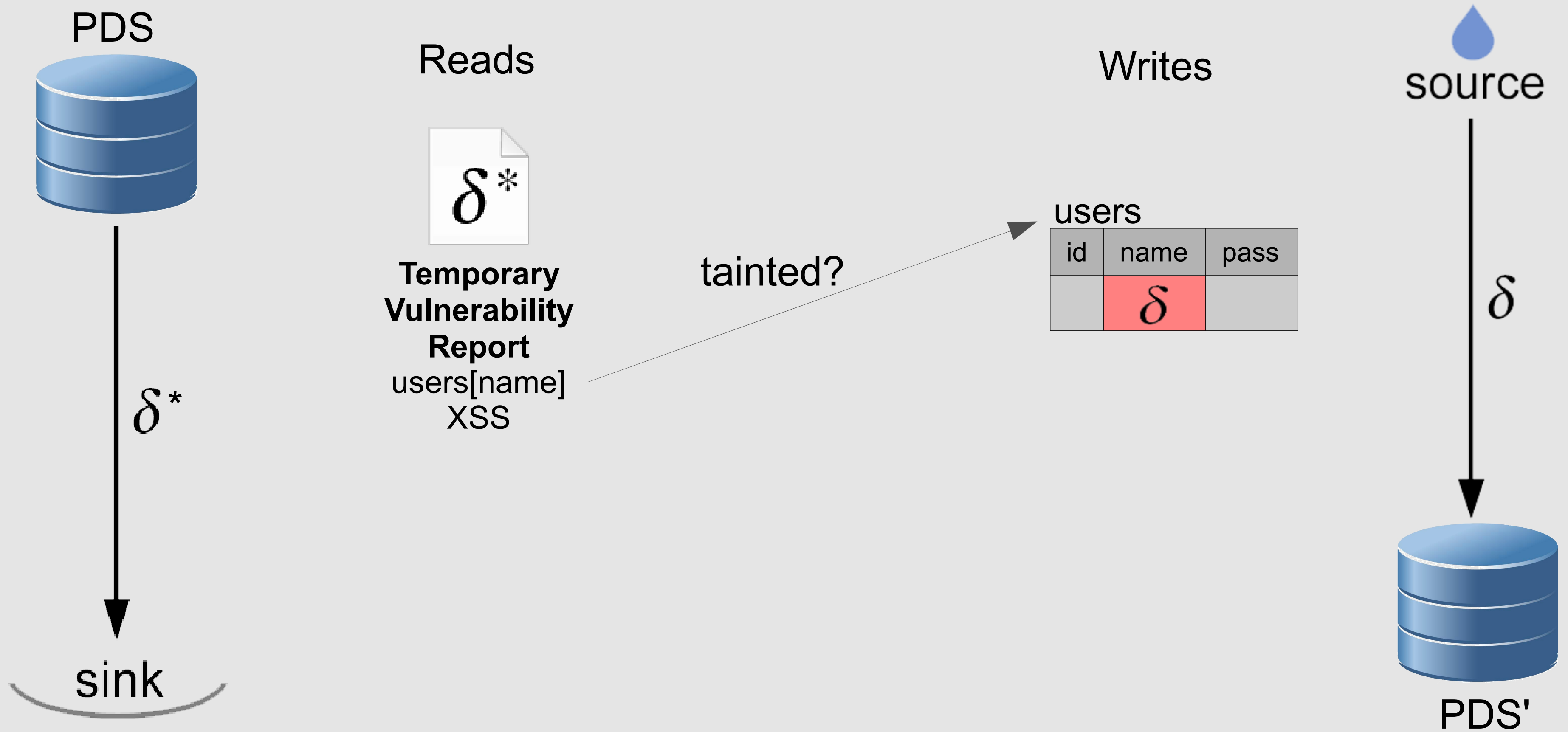
Temporary Vulnerability Report
users[name]
XSS

users

id	name	pass
	δ	

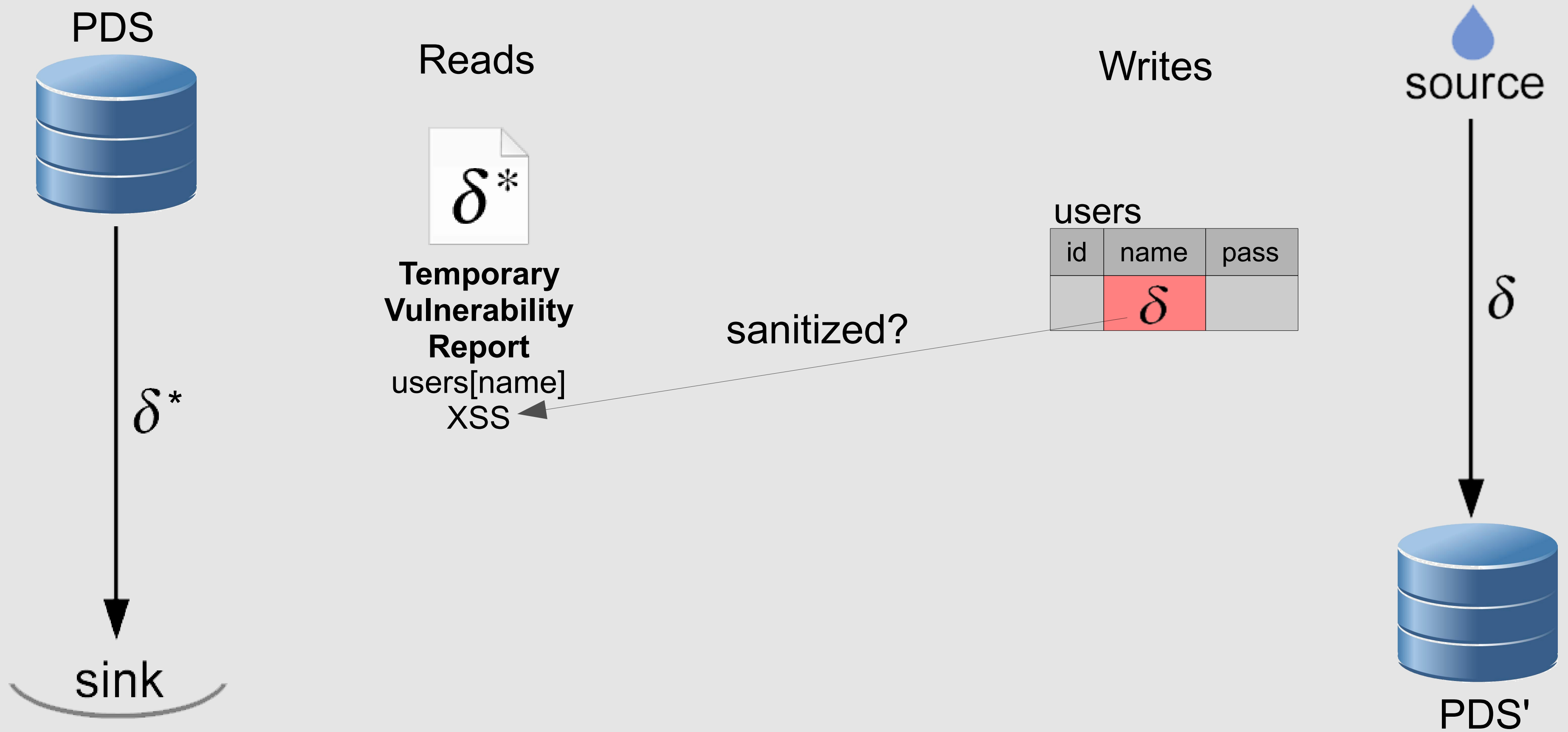
1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.5 Second-Order Vulnerabilities

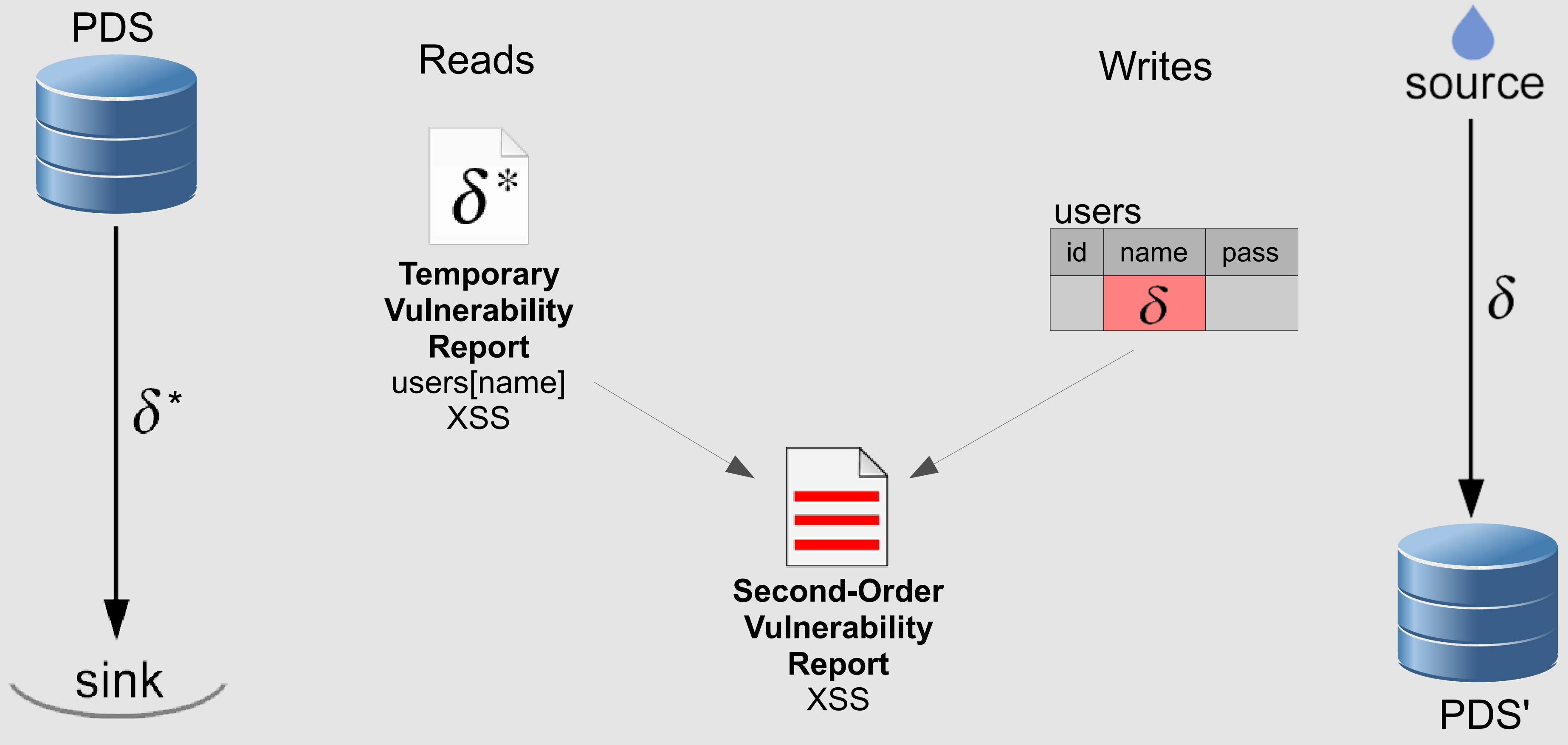


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.5 Second-Order Vulnerabilities



3.5 Second-Order Vulnerabilities

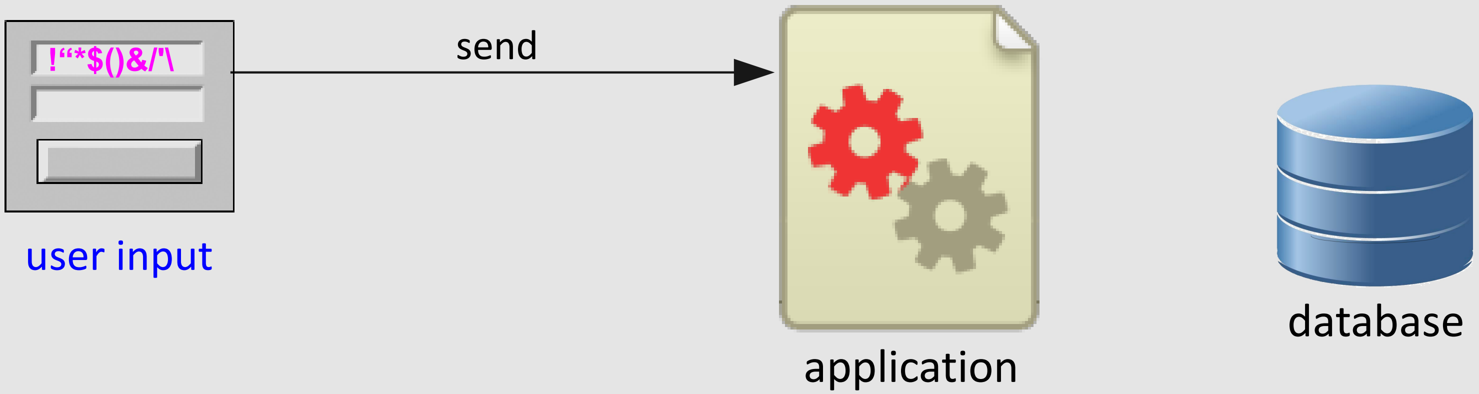


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.6 Multi-Step Exploits

„First-Order“ SQL Injection:

```
$name  = $_POST['name'];
$pwd   = md5($_POST['pwd']);
$query = "INSERT INTO users VALUES('$name','$pwd')";
$result = mysql_query($query);
```

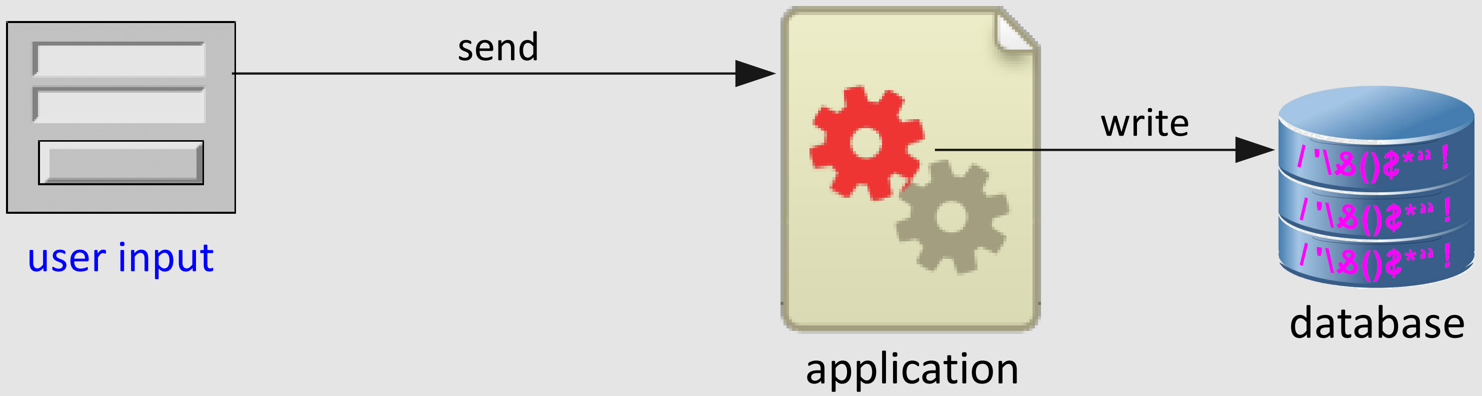


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.6 Multi-Step Exploits

Exploit „First-Order“ SQL Injection to taint database:

```
$name = $_POST['name'];          // ', 'payload')-- -
$pwd  = md5($_POST['pwd']);
$query = "INSERT INTO users VALUES('$name','$pwd')";
$result = mysql_query($query);
```

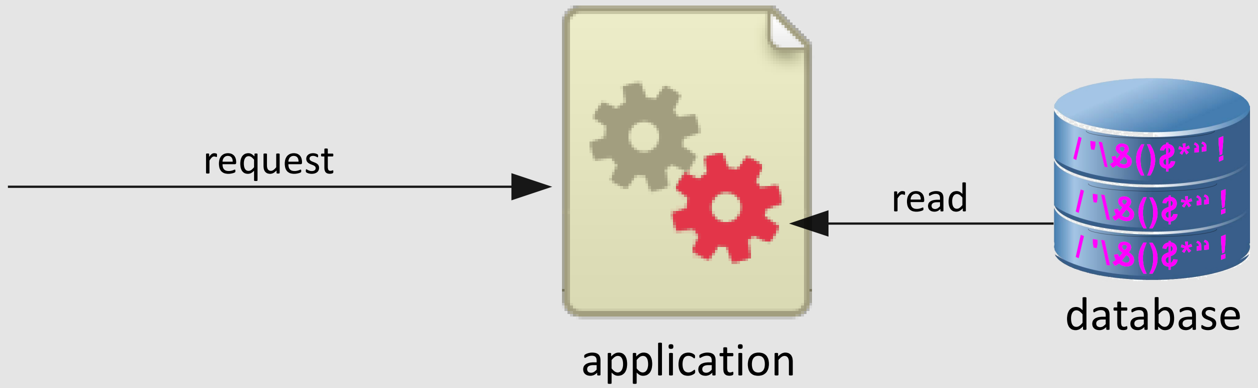


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.6 Multi-Step Exploits

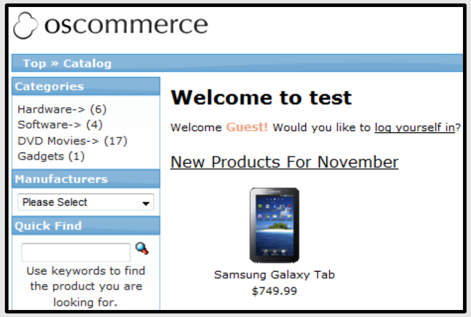
Data from tainted database used in sensitive sink:

```
$query = "SELECT * FROM users WHERE id = 1";
$result = mysql_query($query);
$user = mysql_fetch_assoc($result);
file_put_contents($user['pwd'] . '.txt', $data);
```

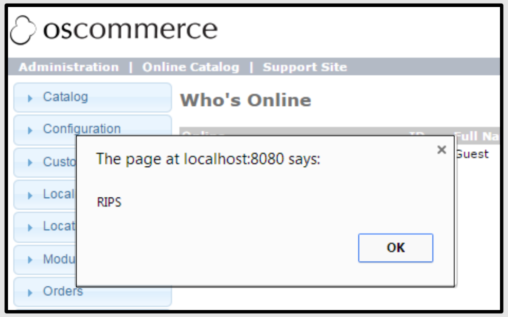


1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.6.1 Multi-Step Exploit – osCommerce 2.3.4



Inject JavaScript (pXSS)

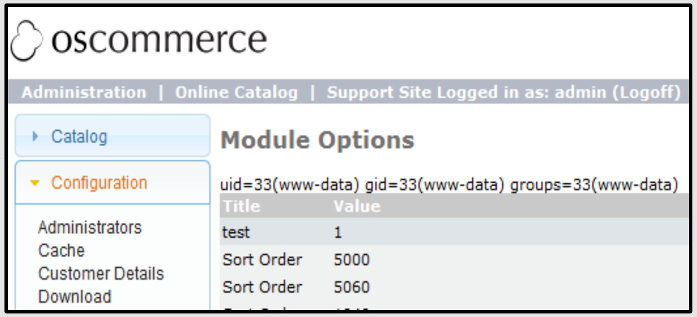


The page at localhost:8080 says:

RIPS

Admin triggers payload

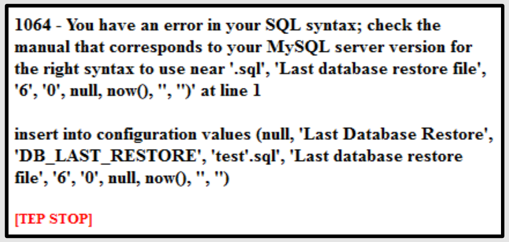
3. load



```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
Title         Value
test          1
Sort Order    5000
Sort Order    5060
```

Remote Command Execution

1. trigger



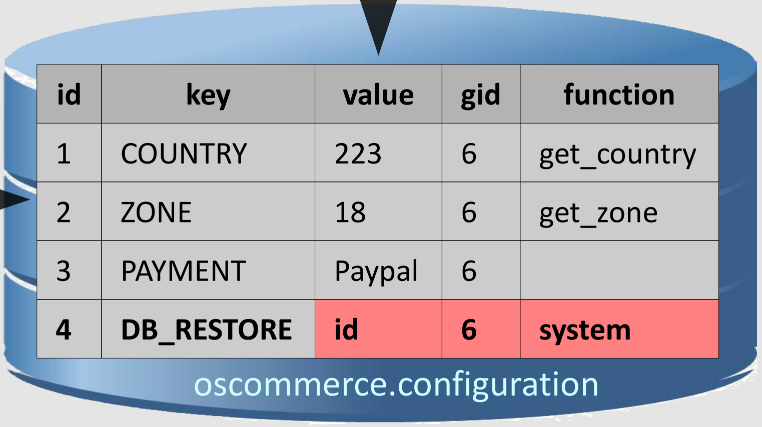
1064 - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '.sql', 'Last database restore file', '6', '0', null, now(), '', '')' at line 1

insert into configuration values (null, 'Last Database Restore', 'DB_LAST_RESTORE', 'test'.sql', 'Last database restore file', '6', '0', null, now(), '', '')

[TEP STOP]

SQL Injection (INSERT)

2. modify

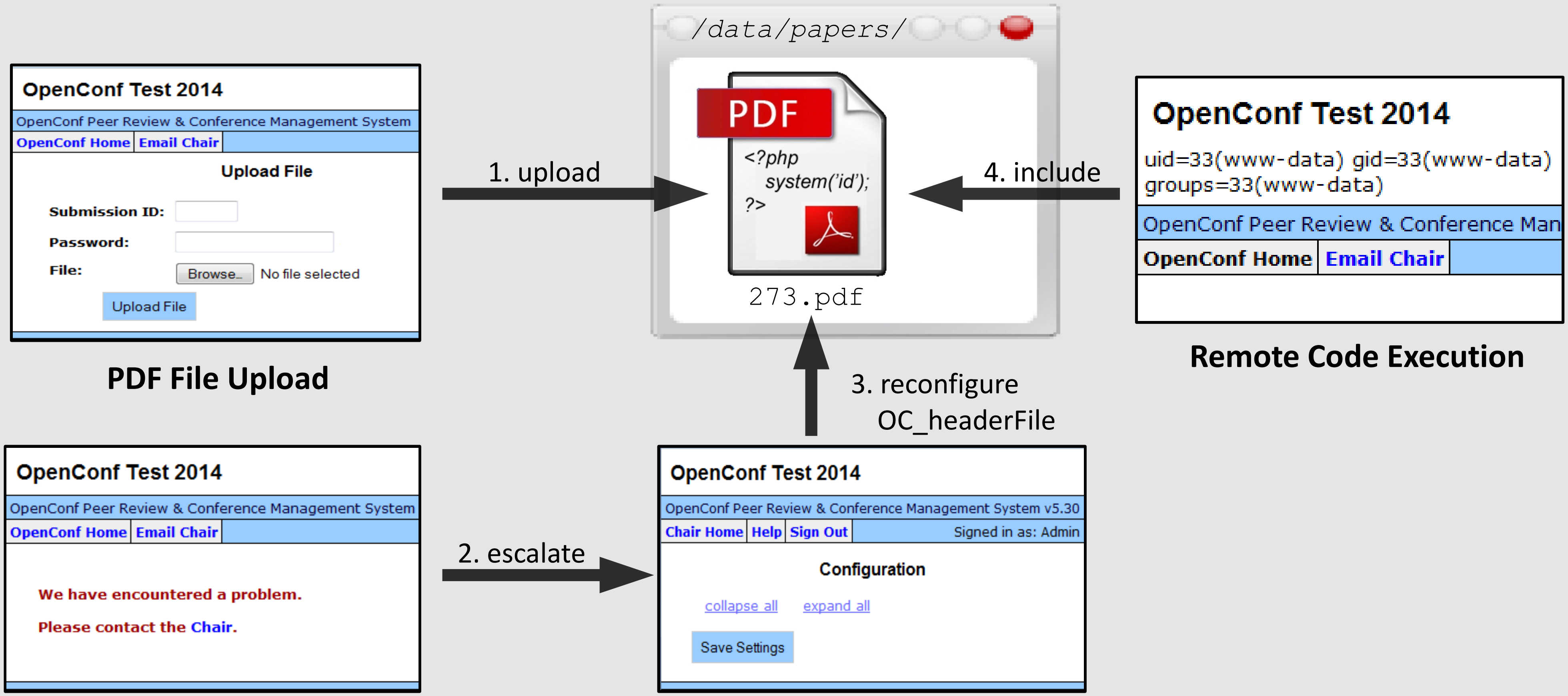


id	key	value	gid	function
1	COUNTRY	223	6	get_country
2	ZONE	18	6	get_zone
3	PAYMENT	Paypal	6	
4	DB_RESTORE	id	6	system

oscommerce.configuration

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

3.6.2 Multi-Step Exploit – OpenConf 5.30



```
<?php
  system('id');
?>
```

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

We have encountered a problem.

Please contact the Chair.

PDF File Upload

Remote Code Execution

Pre-auth SQLi

Second-Order LFI

3.7 Exceptional Vulnerability Types

- ✗ **Authorization Bypass**
- ✗ Cross-Site Request Forgery
- ✓ Cross-Site Scripting
- ✓ Code Execution
- ✓ Command Execution
- ✓ Connection String Injection
- ✓ Denial of Service
- ✓ Directory Listing
- ✓ **Execution After Redirect**
- ✓ File Delete
- ✓ File Disclosure
- ✓ File Inclusion
- ✓ File Overwrite
- ✓ File System Manipulation
- ✓ File Upload
- ✓ HTTP Response Splitting
- ✓ Information Leakage
- ✓ LDAP Injection
- ✓ Log Forgery
- ✓ **Mass Assignment**
- ✓ Memcached Injection
- ✓ Open Redirect
- ✓ **PHP Object Injection**
- ✓ **Reflection/Autoload Injection**
- ✗ Resource Contention
- ✓ Server-Side JavaScript Injection
- ✓ Server-Side Request Forgery
- ✓ Session Fixation
- ✓ SQL Injection
- ✓ Variable Manipulation
- ✗ **Weak Cryptography**
- ✓ **XML/XXE Injection**
- ✓ XPath Injection

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

4. Open Challenges

- ✗ **Authorization Bypass**
- ✗ Cross-Site Request Forgery
- ✓ Cross-Site Scripting
- ✓ Code Execution
- ✓ Command Execution
- ✓ Connection String Injection
- ✓ Denial of Service
- ✓ Directory Listing
- ✓ **Execution After Redirect**
- ✓ File Delete
- ✓ File Disclosure
- ✓ File Inclusion
- ✓ File Overwrite
- ✓ File System Manipulation
- ✓ File Upload
- ✓ HTTP Response Splitting
- ✓ Information Leakage
- ✓ LDAP Injection
- ✓ Log Forgery
- ✓ **Mass Assignment**
- ✓ Memcached Injection
- ✓ Open Redirect
- ✓ **PHP Object Injection**
- ✓ **Reflection/Autoload Injection**
- ✗ Resource Contention
- ✓ Server-Side JavaScript Injection
- ✓ Server-Side Request Forgery
- ✓ Session Fixation
- ✓ SQL Injection
- ✓ Variable Manipulation
- ✗ **Weak Cryptography**
- ✓ **XML/XXE Injection**
- ✓ XPath Injection

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

4.1 Loops

```php
function insert_query($table, $array) {
    foreach($array as $key => value) {
        $fields .= $key . ",";
        $values .= "'" . $value . "',";
    }
    ...
    $this->write_query("INSERT INTO {$table} (" . $fields . ")
        VALUES ('" . $values . "')");
    return $this->insert_id();
}

$new_profile_field = array(
    "name"        => $db->escape_string($mybb->input['name']),
    "description" => $db->escape_string($mybb->input['description'])
);
$fid = $db->insert_query("profile", $new_profile_field);
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

4.1 Loops

```php
function insert_query($table, $array) {
    foreach($array as $key => value) {
        $fields .= $key . ",";
        $values .= "'" . $value . "',";
    }
    ...
    $this->write_query("INSERT INTO {$table} (" . $fields . ")
        VALUES ('" . $values . "')");
    return $this->insert_id();
}

$new_profile_field = array(
    "name"        => $db->escape_string($mybb->input['name']),
    "description" => $db->escape_string($mybb->input['description'])
);
$fid = $db->insert_query("profile", $new_profile_field);
```

4.1 Loops

```php
function insert_query($table, $array) {
    foreach($array as $key => value) {
        $fields .= $key . ",";
        $values .= "'" . $value . "',";
    }
    ...
    $this->write_query("INSERT INTO {$table} (" . $fields . ")
        VALUES ('" . $values . "')");
    return $this->insert_id();
}

$new_profile_field = array(
    "name"        => $db->escape_string($mybb->input['name']),
    "description" => $db->escape_string($mybb->input['description'])
);
$fid = $db->insert_query("profile", $new_profile_field);
```

4.1 Loops

```php
function insert_query($table, $array) {
    foreach($array as $key => value) {
        $fields .= $key . ",";
        $values .= "'" . $value . "',";
    }
    ...
    $this->write_query("INSERT INTO {$table} (" . $fields . ")
        VALUES ('" . $values . "')");
    return $this->insert_id();
}

$new_profile_field = array(
    "name"        => $db->escape_string($mybb->input['name']),
    "description" => $db->escape_string($mybb->input['description'])
);
$fid = $db->insert_query("profile", $new_profile_field);
```

```sql
INSERT INTO profile (name, description) VALUES ('$1', '$2')
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

4.2 Frameworks

- Hard to analyze components lead to false negatives
- Query builders `$db->select('users')->where(array('id', $var));`
- Template engines `$template->assign('B', $var)`
- Reflection logic in configuration file `$controller();`

- Partial solution: Framework-specific configuration
- Problem: high maintainance overhead
- Drupageddon

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

4.3 Path-sensitivity

- Infeasable paths lead to false positives
- Partial solution: *satisfiability solvers (Z3-str, S3)*
- Problem: Performance

```
...
if(!is_numeric($_GET['id'])) {
  $error = true;
}

if(!$error) {
  echo $_GET['id'];
}
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

4.3 Path-sensitivity

- Infeasable paths lead to false positives
- Partial solution: *satisfiability solvers (Z3-str, S3)*
- Problem: Performance

```php
$numeric = is_numeric($_GET['id']);
...
if(!$numeric) {
  $error = true;
}

if(!$error) {
  echo $_GET['id'];
}
```

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

Questions?

johannes.dahse@rub.de

If you think you have a stupid question,
just remember NASA engineers once asked Sally Ride
if 100 tampons were enough for a 7 day mission.

References

- Simulation of Built-in PHP features for Precise Static Code Analysis

  Johannes Dahse, Thorsten Holz

  Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, February 2014

- Static Detection of Second-Order Vulnerabilities in Web Applications

  Johannes Dahse, Thorsten Holz

  23rd USENIX Security Symposium, San Diego, CA, USA, August 2014

- Code Reuse Attacks in PHP: Automated POP Chain Generation

  Johannes Dahse, Nikolai Krein, Thorsten Holz

  21st ACM Conference on Computer and Communications Security (CCS), Scottsdale, AZ, USA, November 2014

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

Join the RIPS project!

(€ / CP)

johannes.dahse@rub.de