WWW 2014

Automatic Detection and Correction of Web Application Vulnerabilities using Data Mining to Predict False Positives

Ibéria Medeiros[1], Nuno F. Neves[1], Miguel Correia[2]

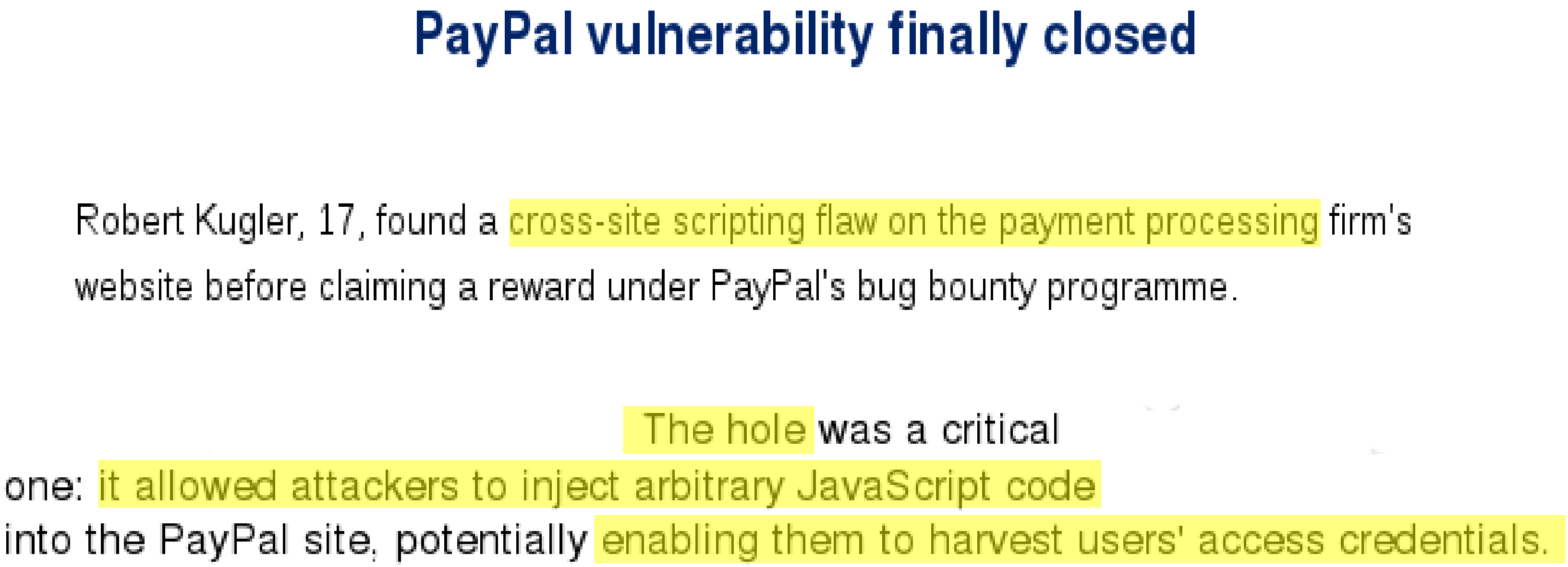
University of Lisboa. [1]Faculty of Sciences, LaSIGE. [2]IST, INESC-ID

LASIGE
Large-Scale Informatics Systems Laboratory

inesc id lisboa

Motivation...

May 30, 2013
TheRegister
TheHSecurity

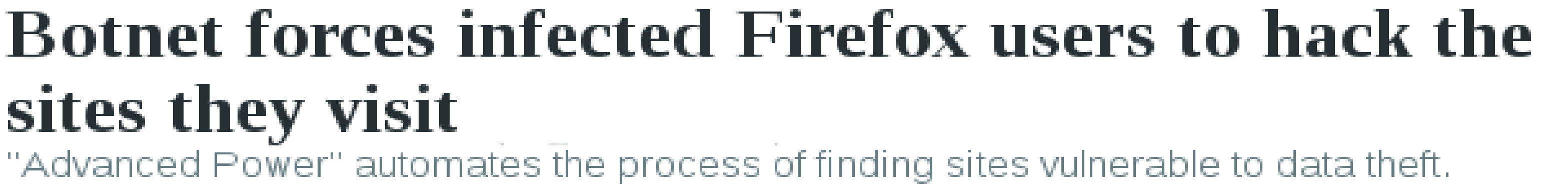


PayPal vulnerability finally closed

Robert Kugler, 17, found a cross-site scripting flaw on the payment processing firm's website before claiming a reward under PayPal's bug bounty programme.

The hole was a critical one: it allowed attackers to inject arbitrary JavaScript code into the PayPal site, potentially enabling them to harvest users' access credentials.

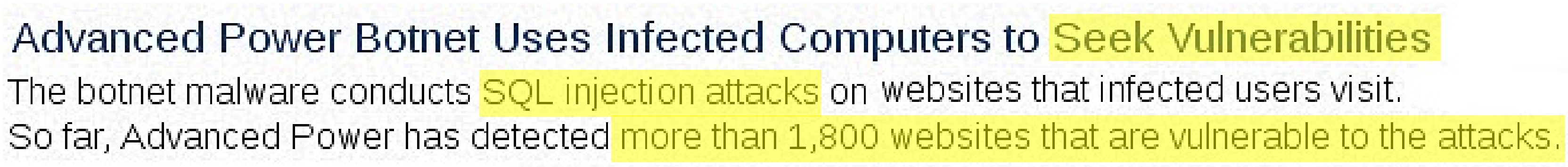
Motivation...



Botnet forces infected Firefox users to hack the sites they visit

"Advanced Power" automates the process of finding sites vulnerable to data theft.

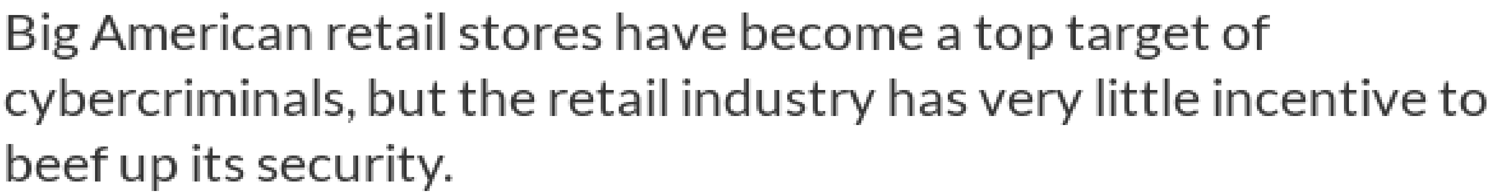
December 16, 2013
ars technica
SANS
KrebsonSecurity



Advanced Power Botnet Uses Infected Computers to Seek Vulnerabilities

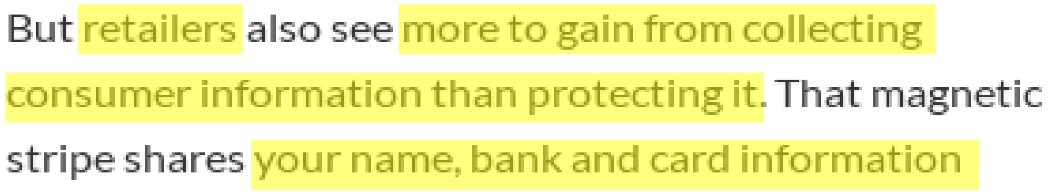
The botnet malware conducts SQL injection attacks on websites that infected users visit.
So far, Advanced Power has detected more than 1,800 websites that are vulnerable to the attacks.

Motivation…

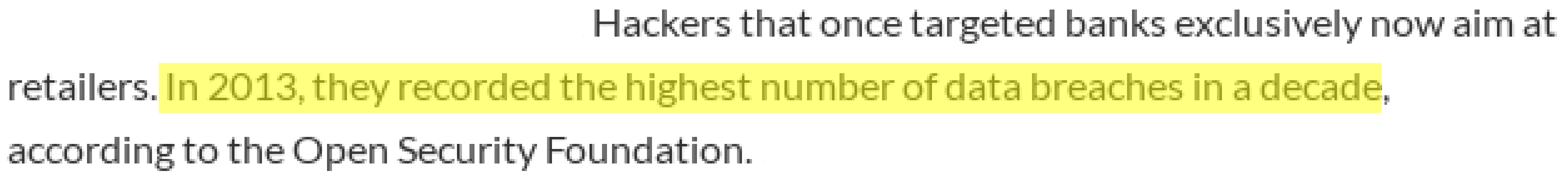


Big American retail stores have become a top target of cybercriminals, but the retail industry has very little incentive to beef up its security.

February 18, 2014
New York, CNN Money

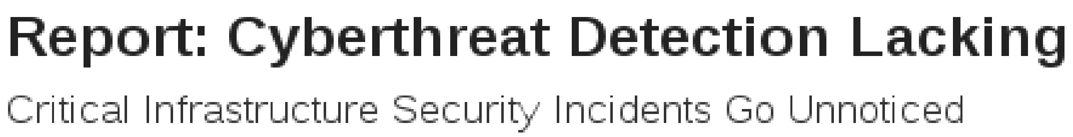


But retailers also see more to gain from collecting consumer information than protecting it. That magnetic stripe shares your name, bank and card information



Hackers that once targeted banks exclusively now aim at retailers. In 2013, they recorded the highest number of data breaches in a decade, according to the Open Security Foundation.

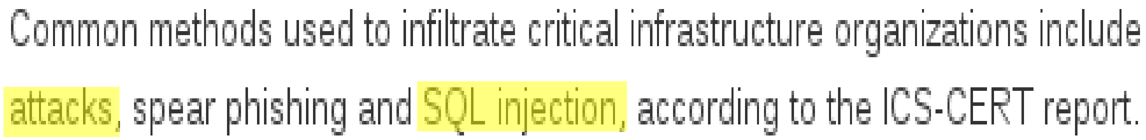
Motivation...



Report: Cyberthreat Detection Lacking

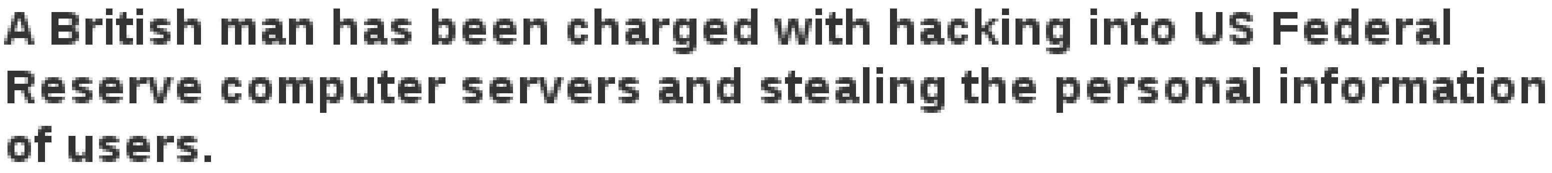
Critical Infrastructure Security Incidents Go Unnoticed

February 19, 2014
GovInfo Security

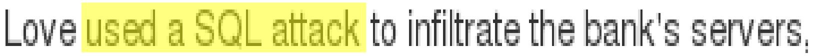


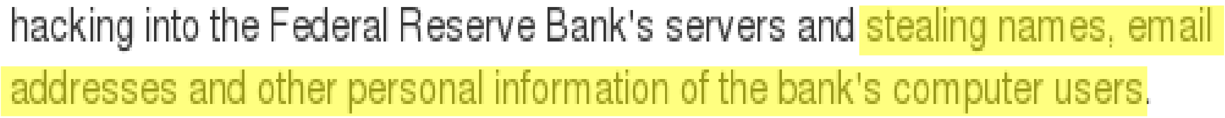
Common methods used to infiltrate critical infrastructure organizations include attacks, spear phishing and SQL injection, according to the ICS-CERT report.

Motivation...


A British man has been charged with hacking into US Federal Reserve computer servers and stealing the personal information of users.

February 27, 2014
BBC, News US & CA
ComputerWorld


Love used a SQL attack to infiltrate the bank's servers,


hacking into the Federal Reserve Bank's servers and stealing names, email addresses and other personal information of the bank's computer users.

Motivation...

Web applications are exposed to malicious user inputs

Web applications with vulnerabilities are insecure

Web applications lack software security !

Web applications are mostly written in PHP... more than 77%

Outline

1. Taint Analysis: detect vulnerabilities
2. Data Mining: predict false positives
3. WAP Tool:
   - Taint analysis
   - Data mining
   - Code correction
4. Evaluation
5. Conclusion

Outline

1. Taint Analysis: detect vulnerabilities

2. Data Mining: predict false positives

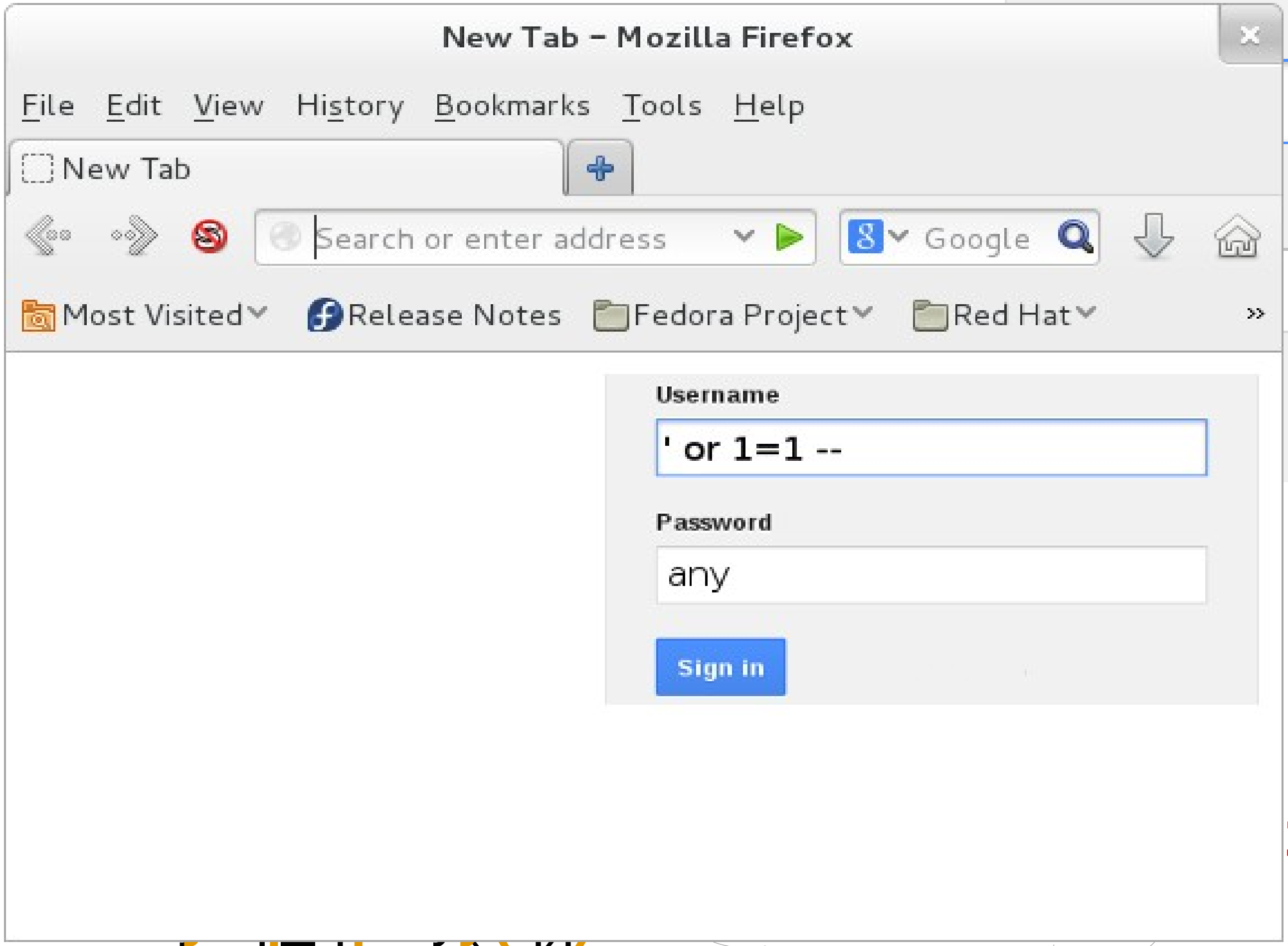
3. WAP Tool:

   - Taint analysis
   - Data mining
   - Code correction

4. Evaluation

5. Conclusion

Vulnerability: SQL Injection example...



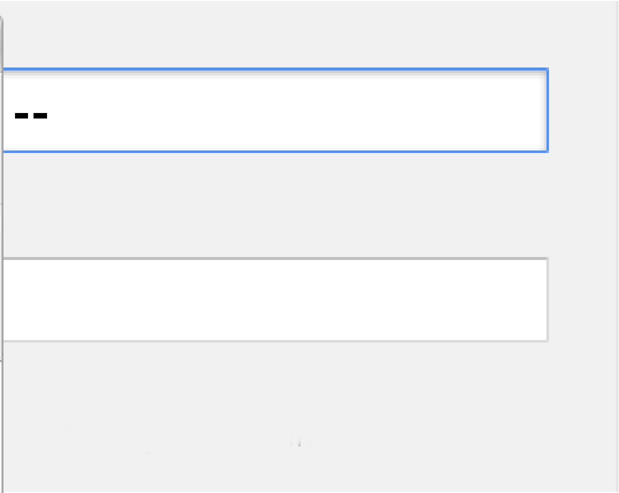
Username

' or 1=1 --

Password

any

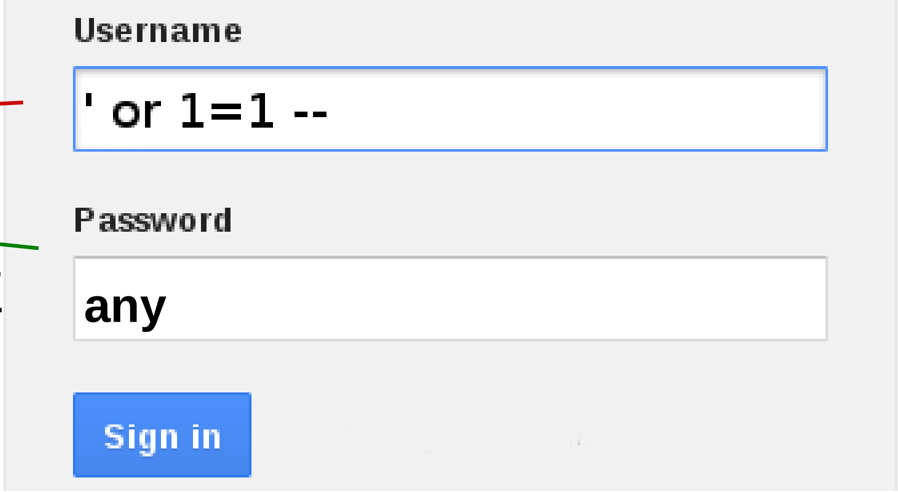
Sign in



1=1--”; ' AND pass='any'”;

Vulnerability: SQL Injection example...

```
$u = $_POST['user'];
$p = $_POST['password'];
$q = "SELECT * FROM users WHERE
$r = mysql_query($q);
```



Username

' or 1=1 --

Password

any

Vulnerability: SQL Injection example...

```
$u = $_POST[’user’];
$p = $_POST[’password’];
$q = “SELECT * FROM users WHERE user='$u' AND pass='$p'”;
$r = mysql_query($q);
```

```
$u = “' or 1=1 -- “;
$p = “any”;
$q = “SELECT * FROM users WHERE user='' or 1=1”;
$r = mysql_query($q);
```

SQL Injection exploited

If we could track the user inputs and verify if they reach some functions, then we could detect the vulnerability...

Taint Analysis: vulnerabilities detected

If we could track the user inputs and verify if they reach some functions, then we could detect vulnerabilities...

↳ ...Taint Analysis

Source Code Static Analysis

Vulnerabilities detected:

Most exploited:

- SQL Injection
- Cross Site Scripting (XSS)

Others:

- Remote file inclusion
- Local file inclusion
- Directory Traversal / Path Traversal
- Source code disclosure
- OS command injection
- PHP code injection

Taint Analysis: taintedness

How?

- taints all entry points (user inputs, e.g., *$_POST*)
- follows the code propagating its taintedness
- until it reaches a sensitive sink
  (some functions, e.g., *mysql_query*)

```
$u = $_POST[’user’];
$p = $_POST[’password’];
$q = “SELECT * FROM users WHERE user='$u' AND pass='$p'”;
$r = mysql_query($q);
```

SQL Injection detected

T	D
1	-
1	-
1	$u, $p
1	$q

T: taintedness
(0: untaint, 1: taint)
D: depends of..

Taint Analysis: untaintedness

PHP has sanitization functions to:

- sanitize the user inputs
- invalidate the exploiting of vulnerabilities

Taint Analysis:

- handles this
- does not propagate the taintedness

```
$u = $_POST['user'];
$p = $_POST['password'];
$uu = mysql_real_escape_string($u);
$pp = mysql_real_escape_string($p);
$q = “SELECT * FROM users WHERE user='$uu' AND pass='$pp'”;
$r = mysql_query($q);
```

Secure code

T	D
1	-
1	-
0	$u
0	$p
0	-
0	-

T: taintedness
(0: untaint, 1: taint)
D: depends of..

Taint Analysis: false positives

Taint analysis tends to generate false positives (detected non-vulnerabilities)

How to avoid them?

- Not propagate taintedness when sanitization functions are present
- Propagate taintedness between: function calls and include files

But... what happens when...

	T	D
`$u = $_POST[’user’];`	1	-
`$p = $_POST[’password’];`	1	-
`$uu = substr($u, 2, 6);`	1	$u
`$uu = trim($uu);`	1	$uu
`$q = “SELECT * FROM users WHERE user='$uu' AND pass='$p'”;`	1	$uu, $p
`$r = mysql_query($q);`	1	$q

False Positive

T: taintedness
(0: untaint, 1: taint)
D: depends of..

We have a problem!!! How to solve it?

Characterization of false positives

What are the symptoms of the possible existence of a false positive?

If in the code slice between the entry point and the sensitive sink the user input is:

- changed
  - string manipulation functions (e.g., *substr*)
  - concatenation operations
- validated
  - type checking functions (e.g., *isset*, *is_string*)
  - white and black lists
- manipulated in SQL queries
  - aggregate functions (e.g., *agv*, *sum*)
  - complex query
  - FROM clause

The presence of each of these symptoms is an attribute

If we could verify the presence of these attributes in the slice, then we could classify the vulnerabilities as being a FP or real

Approach to resolve false positives

If we could verify the presence of these attributes in the slice, then we could classify the vulnerabilities as being a FP or real

Coding this knowledge

- hard and complex task
- a lot of if's statements
- can incur in logic errors

if with these attributes we could retrieve and discover information then we could classify the vulnerabilities

Discover the knowledge

Data Mining

Outline

1. Taint Analysis: detect vulnerabilities

2. Data Mining: predict false positives

3. WAP Tool

   - Taint analysis
   - Data mining
   - Code correction

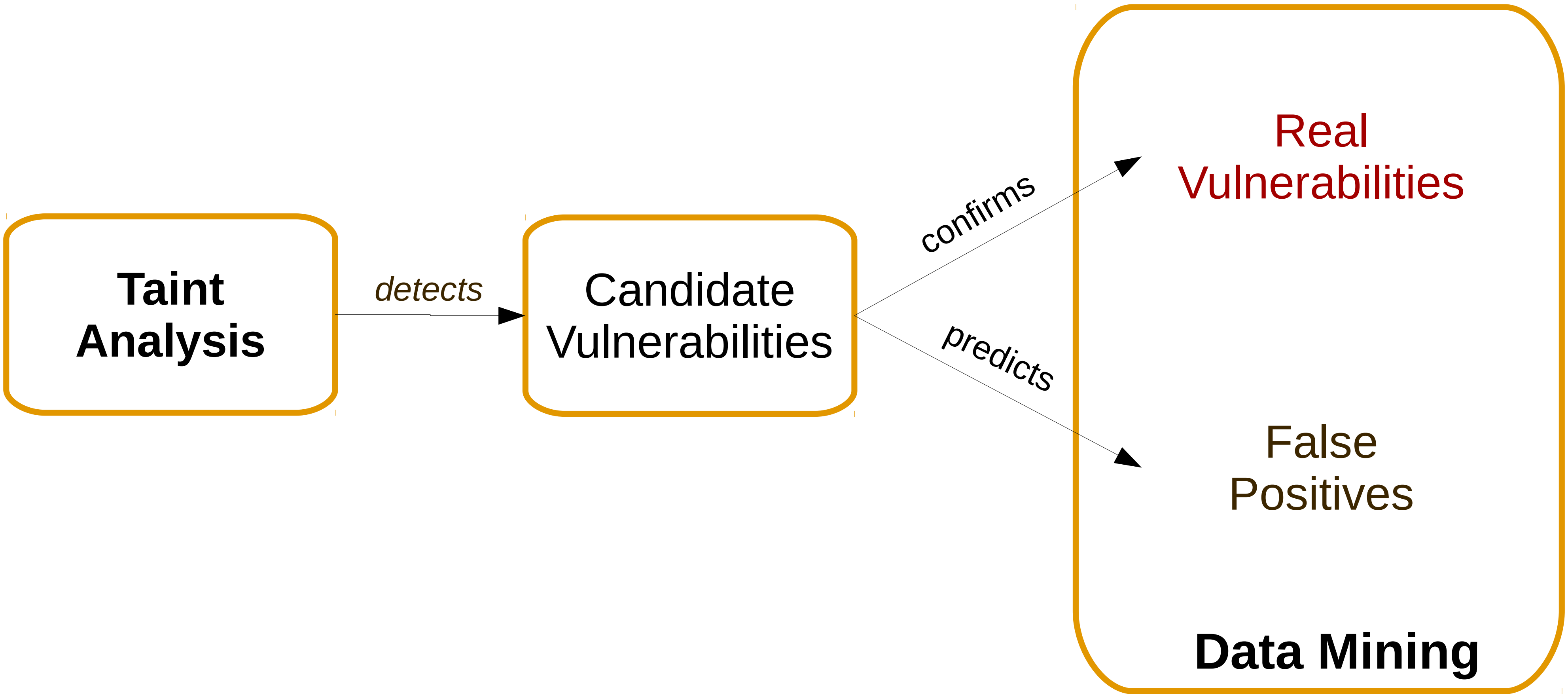
4. Evaluation

5. Conclusion

Have... and... want...

What do we have...

What do we want...



Taint Analysis

detects

Candidate Vulnerabilities

confirms

predicts

Real Vulnerabilities

False Positives

Data Mining

Data Mining: what do we need to apply it...

What do we need to apply data mining?

A set of attributes that characterize a false positive ✓
- string manipulation, validation, manipulation in SQL queries

Class values to classify an instance in the class ✓
- is a FP (Y); is not a FP (N = real Vulnerability)

A data set of instances with FP and real vulnerabilities to train a classifier ✓
- 76 instances: 32 false positives; 44 real vulnerabilities. Data set was obtained manually by a hard and tedious process

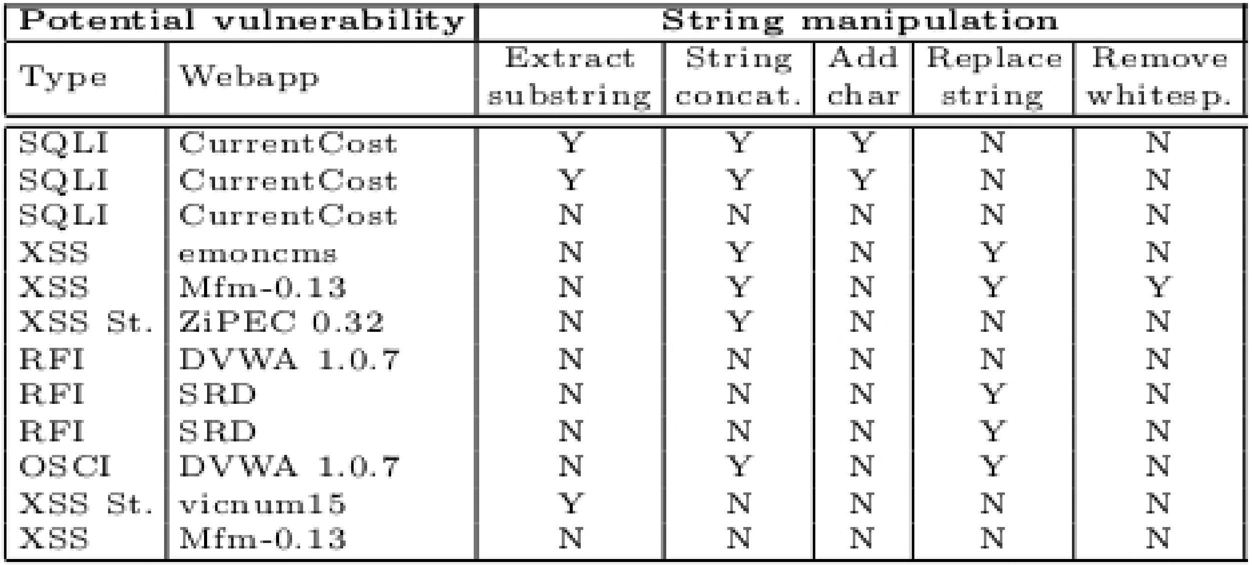
●

A machine learning classifier to classify new instances

We define a <u>process</u> to evaluate machine learning classifiers to choose the best that classify our instances with high accuracy and precision

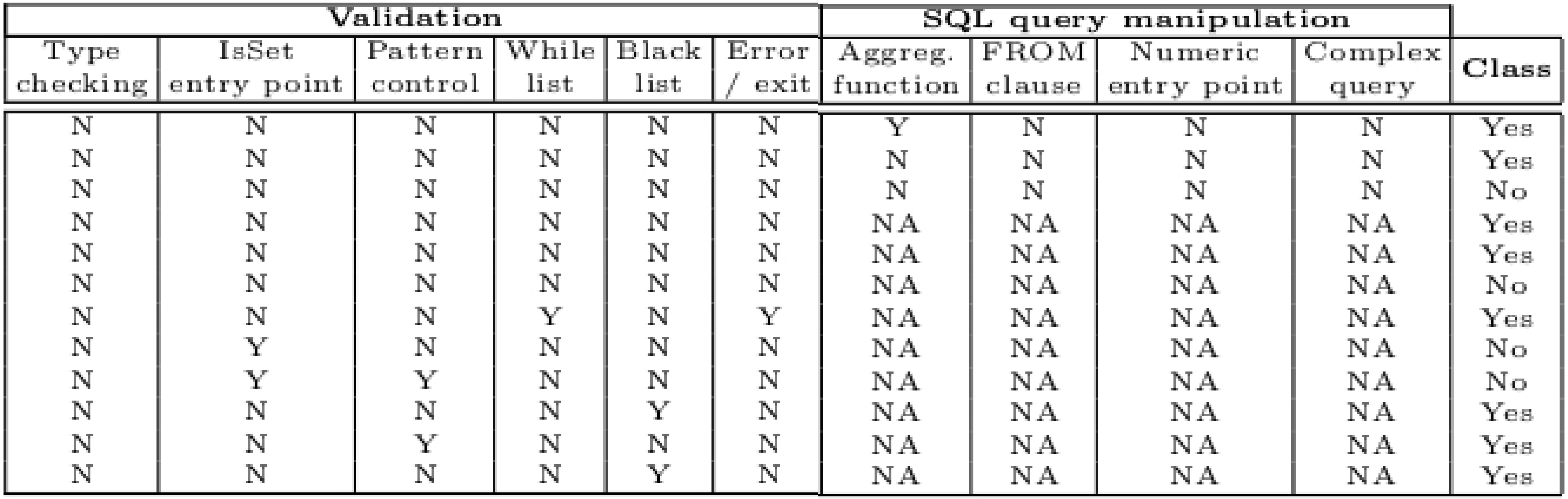
Composition of the data set used

- 76 instances: 32 false positives + 44 real vulnerabilities
- 15 attributes: 14 to characterize a false positive + 1 to classify it



Potential vulnerability		String manipulation				
Type	Webapp	Extract substring	String concat.	Add char	Replace string	Remove whitesp.
SQLI	CurrentCost	Y	Y	Y	N	N
SQLI	CurrentCost	Y	Y	Y	N	N
SQLI	CurrentCost	N	N	N	N	N
XSS	emoncms	N	Y	N	Y	N
XSS	Mfm-0.13	N	Y	N	Y	Y
XSS St.	ZiPEC 0.32	N	Y	N	N	N
RFI	DVWA 1.0.7	N	N	N	N	N
RFI	SRD	N	N	N	Y	N
RFI	SRD	N	N	N	Y	N
OSCI	DVWA 1.0.7	N	Y	N	Y	N
XSS St.	vicnum15	Y	N	N	N	N
XSS	Mfm-0.13	N	N	N	N	N

(...)



Validation						SQL query manipulation				
Type checking	IsSet entry point	Pattern control	While list	Black list	Error / exit	Aggreg. function	FROM clause	Numeric entry point	Complex query	Class
N	N	N	N	N	N	Y	N	N	N	Yes
N	N	N	N	N	N	N	N	N	N	Yes
N	N	N	N	N	N	N	N	N	N	No
N	N	N	N	N	N	NA	NA	NA	NA	Yes
N	N	N	N	N	N	NA	NA	NA	NA	Yes
N	N	N	N	N	N	NA	NA	NA	NA	No
N	N	N	Y	N	Y	NA	NA	NA	NA	Yes
N	Y	N	N	N	N	NA	NA	NA	NA	No
N	Y	Y	N	N	N	NA	NA	NA	NA	No
N	N	N	N	Y	N	NA	NA	NA	NA	Yes
N	N	Y	N	N	N	NA	NA	NA	NA	Yes
N	N	N	N	Y	N	NA	NA	NA	NA	Yes

Evaluation of classifiers

Using WEKA to:

- evaluate 10 machine learning classifiers
  - 5 Graphical/symbolic algorithms (decision trees)
  - 3 Probabilistic algorithms (NB, K-NN, LR)
  - 2 Neural network algorithms (MLP, SVM)

- train and test the classifiers with 10-fold cross validation estimator
  - divides the data into 10 buckets, trains the classifier with 9 of them and tests it with the 10th. This process is repeated 10 times to test every bucket with the classifier trained with the rest

- get 10 metrics to evaluate the classifiers performance
  - 3 for false positives prediction
  - 3 for real vulnerabilities detection
  - 2 global metrics (accuracy and precision)
  - 2 global tests

Evaluation of classifiers

Metrics are based in a confusion matrix

		Observed	
		Yes (FP)	**No (not FP)**
Predicted	**Yes (FP)**	match	not match
	No (not FP)	not match	match

		Observed	
		Yes (FP)	**No (not FP)**
Predicted	**Yes (FP)**	27	**1**
	No (not FP)	5	43

Logistic Regression

Accuracy = 92.1%
Precision = 92.5%

Whenever more data is included in the data set,
we can redo the process to define a better classifier

Outline

1. Taint Analysis: detect vulnerabilities

2. Data Mining: predict false positives

3. WAP Tool
   - Taint analysis
   - Data mining
   - Code correction

4. Evaluation

5. Conclusion

Approach

WAP (*Web Application Protection*)

Analysis

- searches for candidate input validation vulnerabilities in the source code of a PHP web application

Prediction

- predicts if a candidate vulnerability is a false positive or a real vulnerability

Correction

- inserts fixes in the source code to remove the vulnerabilities

Feedback

- reports the real vulnerabilities detected and how they were corrected
- outputs a corrected version of the web application
- reports the false positives predicted

Code Correction... Fixes

Vulnerabilities are removed:

Correction of source code by insertion of fixes in the right places

Fixes:

What do they do?

- Do proper validation or sanitization of user input before it is used in some sensitive sink

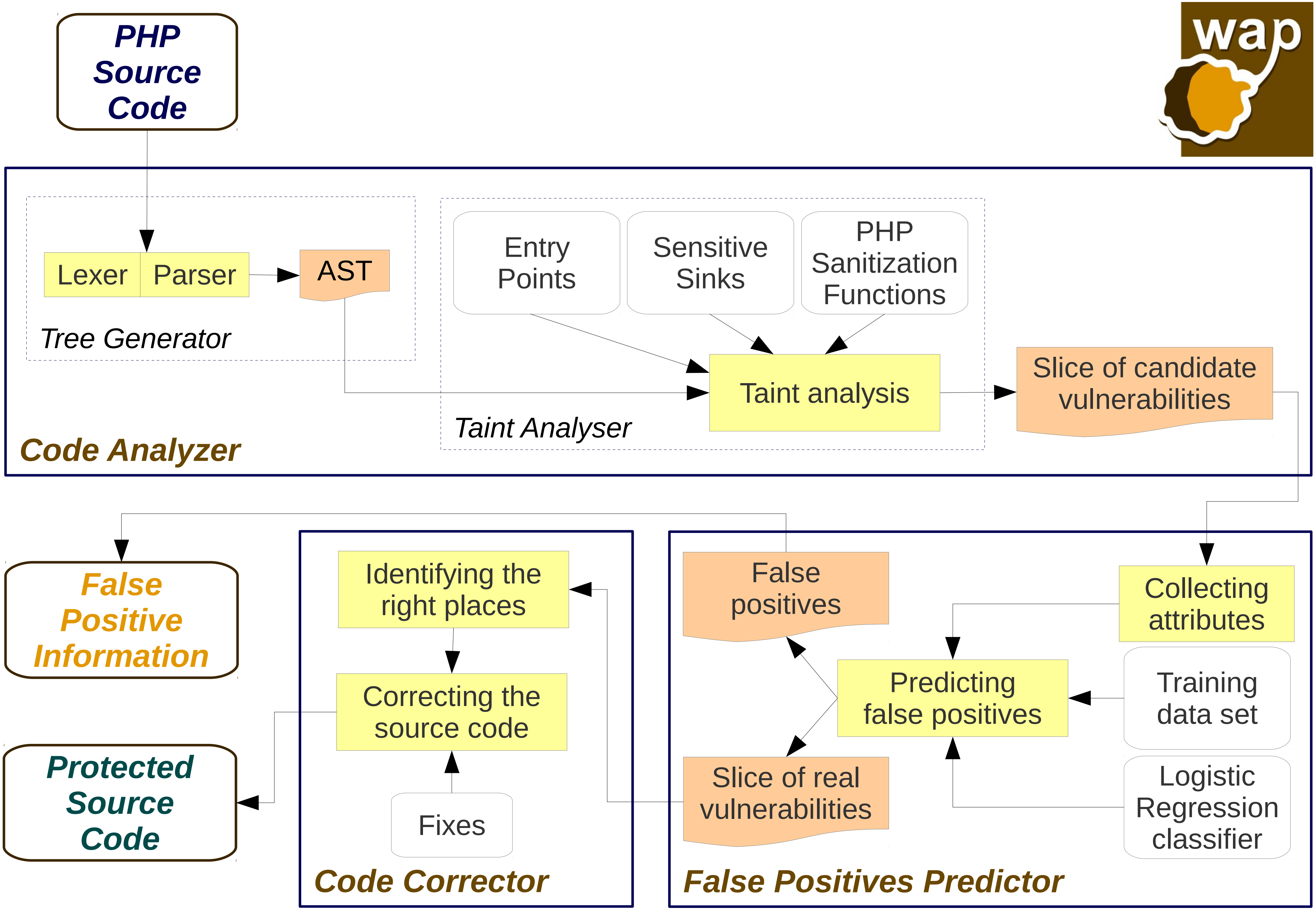
Where are they inserted?

- In sensitive sinks or close to them
- Do not compromise the behavior of the web application

What are they?

- Small PHP functions developed by us
- Some do sanitization and others do validation

Architecture


PHP Source Code
Lexer
Parser
AST
Tree Generator
Entry Points
Sensitive Sinks
PHP Sanitization Functions
Taint analysis
Taint Analyser
Slice of candidate vulnerabilities
Code Analyzer
False Positive Information
Protected Source Code
Identifying the right places
Correcting the source code
Fixes
Code Corrector
False positives
Collecting attributes
Predicting false positives
Training data set
Slice of real vulnerabilities
Logistic Regression classifier
False Positives Predictor

Challenges of implementing WAP

Global, interprocedural and context-sensitive analysis

- To avoid false positives and false negatives the taintedness propagation has to be:
  - interprocedural: between functions or methods calls
  - global: even if the functions/methods belong to different modules
  - context-sensitive: to the point of the program where the function call was made

Environment variables

- Resolve the name of the include files to perform the corrected global analysis

Class analysis

- Handles taintedness propagation between objects and methods calls

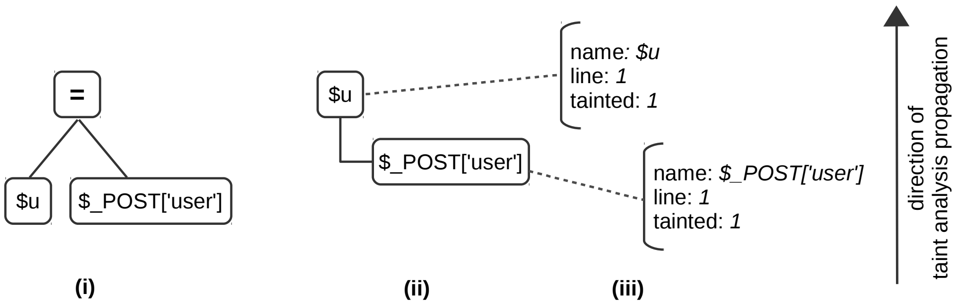
Uncertainty about PHP's syntax

- PHP is weakly typed and not formally specified. Frequent use of poorly documented features that can break the parser

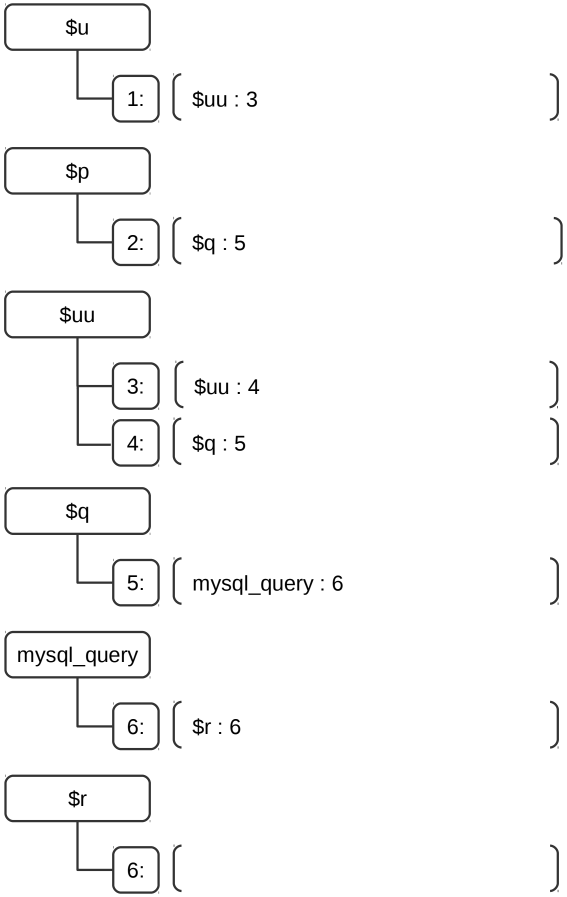
Challenges of implementing WAP

Need of both top-down and bottom-up approaches

- navigates in the AST using the top-down approach to taint the entry points, then following the bottom-up approach to propagate its taintedness to its parent
- identifies the vulnerable path and the right places to insert the fixes using the bottom-up approach
- collects the attributes and performs the correction of the source code using the top-down approach



AST **(i)**, TST **(ii)** and taint analysis of the *$u = $_POST['user'];* statement **(iii)**



Outline

1. Taint Analysis: detect vulnerabilities

2. Data Mining: predict false positives

3. WAP Tool

   - Taint analysis
   - Data mining
   - Code correction

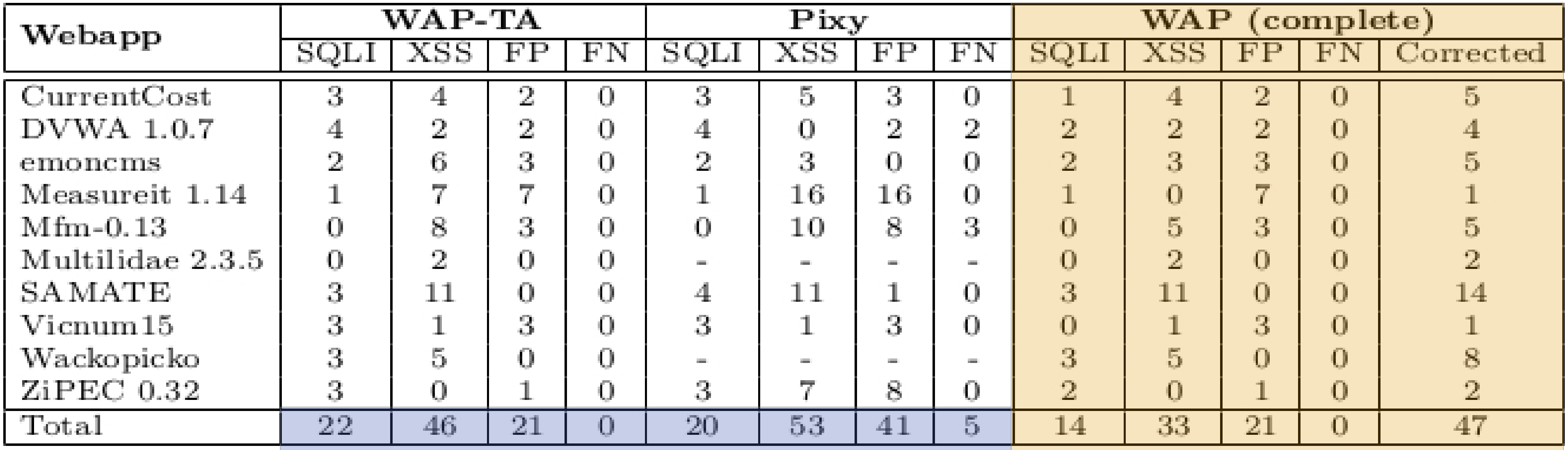
4. Evaluation

5. Conclusion

Comparison of WAP vs Pixy

- *Pixy* is a static analysis tool that performs taint analysis to detect SQL injection and XSS vulnerabilities

- Taint analysis comparative evaluation between WAP and *Pixy* in analysis of 10 open source code packages



Webapp	WAP-TA				Pixy				WAP (complete)				
	SQLI	XSS	FP	FN	SQLI	XSS	FP	FN	SQLI	XSS	FP	FN	Corrected
CurrentCost	3	4	2	0	3	5	3	0	1	4	2	0	5
DVWA 1.0.7	4	2	2	0	4	0	2	2	2	2	2	0	4
emoncms	2	6	3	0	2	3	0	0	2	3	3	0	5
Measureit 1.14	1	7	7	0	1	16	16	0	1	0	7	0	1
Mfm-0.13	0	8	3	0	0	10	8	3	0	5	3	0	5
Multilidae 2.3.5	0	2	0	0	-	-	-	-	0	2	0	0	2
SAMATE	3	11	0	0	4	11	1	0	3	11	0	0	14
Vicnum15	3	1	3	0	3	1	3	0	0	1	3	0	1
Wackopicko	3	5	0	0	-	-	-	-	3	5	0	0	8
ZiPEC 0.32	3	0	1	0	3	7	8	0	2	0	1	0	2
Total	22	46	21	0	20	53	41	5	14	33	21	0	47

68 vuln.: 21 are FP

0 false negatives

Same 11 FP than Pixy

73 vuln.: 41 are FP

5 false negatives

Same 11 FP than WAP

+ 30 FP than WAP

47 real vulnerabilities

21 predicted false positives

0 false negatives

47 vulnerabilities corrected

Without data mining

With data mining

Comparison of WAP vs PhpMinerII

- PhpMinerII is a tool that predicts the presence of SQLI/XSS vulnerabilities in PHP applications. On the contrary to WAP, it does not identify where the vulnerabilities are, only predicts their existence

- PhpMinerII does data mining of slices that end at a sensitive sink, independently of data being propagated through them starting at an entry point or not

		Observed	
		Yes (Vuln.)	**No (not Vuln.)**
Predicted	**Yes (Vuln.)**	48	**5**
	No (not Vuln.)	5	20

Logistic Regression

Accuracy = 87.2%
Precision = 85.2%

The 48 vulnerabilities can contain false positives

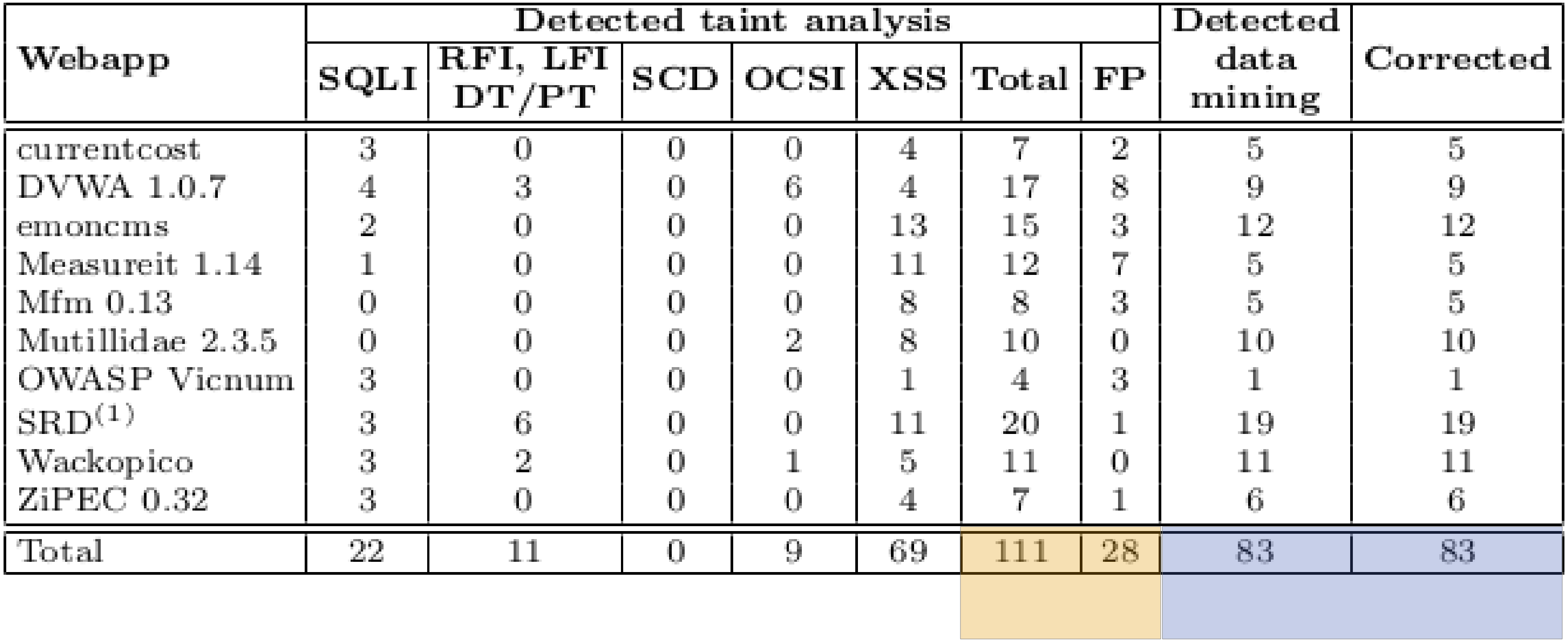
47 real vulnerabilities
21 predicted false positives
0 false negatives
47 vulnerabilities corrected

WAP with data mining

Full comparative evaluation

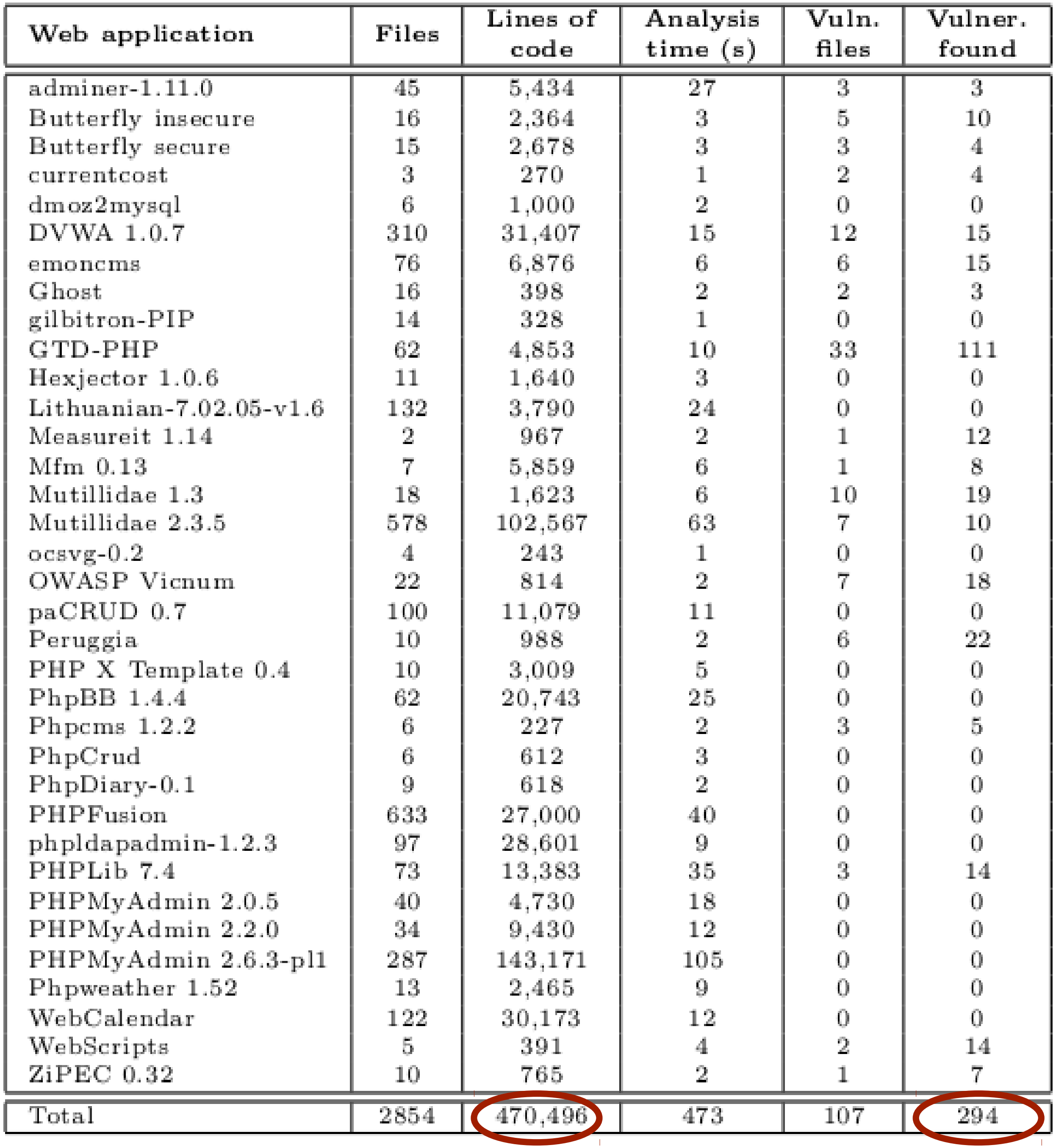
Metric	**WAP**	***Pixy***	***PhpMinerII***
accuracy	92.1%	44.0%	87.2%
precision	92.5%	50.0%	85.2%

WAP analysis to all vulnerabilities



Webapp	Detected taint analysis							Detected data mining	Corrected
	SQLI	RFI, LFI DT/PT	SCD	OCSI	XSS	Total	FP		
currentcost	3	0	0	0	4	7	2	5	5
DVWA 1.0.7	4	3	0	6	4	17	8	9	9
emoncms	2	0	0	0	13	15	3	12	12
Measureit 1.14	1	0	0	0	11	12	7	5	5
Mfm 0.13	0	0	0	0	8	8	3	5	5
Mutillidae 2.3.5	0	0	0	2	8	10	0	10	10
OWASP Vicnum	3	0	0	0	1	4	3	1	1
SRD(1)	3	6	0	0	11	20	1	19	19
Wackopico	3	2	0	1	5	11	0	11	11
ZiPEC 0.32	3	0	0	0	4	7	1	6	6
Total	22	11	0	9	69	111	28	83	83

Summary of the WAP analysis



Web application	Files	Lines of code	Analysis time (s)	Vuln. files	Vulner. found
adminer-1.11.0	45	5,434	27	3	3
Butterfly insecure	16	2,364	3	5	10
Butterfly secure	15	2,678	3	3	4
currentcost	3	270	1	2	4
dmoz2mysql	6	1,000	2	0	0
DVWA 1.0.7	310	31,407	15	12	15
emoncms	76	6,876	6	6	15
Ghost	16	398	2	2	3
gilbitron-PIP	14	328	1	0	0
GTD-PHP	62	4,853	10	33	111
Hexjector 1.0.6	11	1,640	3	0	0
Lithuanian-7.02.05-v1.6	132	3,790	24	0	0
Measureit 1.14	2	967	2	1	12
Mfm 0.13	7	5,859	6	1	8
Mutillidae 1.3	18	1,623	6	10	19
Mutillidae 2.3.5	578	102,567	63	7	10
ocsvg-0.2	4	243	1	0	0
OWASP Vicnum	22	814	2	7	18
paCRUD 0.7	100	11,079	11	0	0
Peruggia	10	988	2	6	22
PHP X Template 0.4	10	3,009	5	0	0
PhpBB 1.4.4	62	20,743	25	0	0
Phpcms 1.2.2	6	227	2	3	5
PhpCrud	6	612	3	0	0
PhpDiary-0.1	9	618	2	0	0
PHPFusion	633	27,000	40	0	0
phpldapadmin-1.2.3	97	28,601	9	0	0
PHPLib 7.4	73	13,383	35	3	14
PHPMyAdmin 2.0.5	40	4,730	18	0	0
PHPMyAdmin 2.2.0	34	9,430	12	0	0
PHPMyAdmin 2.6.3-pl1	287	143,171	105	0	0
Phpweather 1.52	13	2,465	9	0	0
WebCalendar	122	30,173	12	0	0
WebScripts	5	391	4	2	14
ZiPEC 0.32	10	765	2	1	7
Total	2854	470,496	473	107	294

Conclusion

- Web applications can have input validation vulnerabilities

- We present an approach and a tool called WAP to automatically identify and correct these vulnerabilities and to predict false positives using data mining

- The WAP tool was compared with Pixy and PhpMinerII, analyzing 10 open source code packages, and:
  - had better performance in detection of SQLI and XSS vulnerabilities with and without data mining

- WAP analyzed 35 applications (~470K LOC in ~2900) and identified ~300 vulnerabilities *http://awap.sourceforge.net*

Thank you!