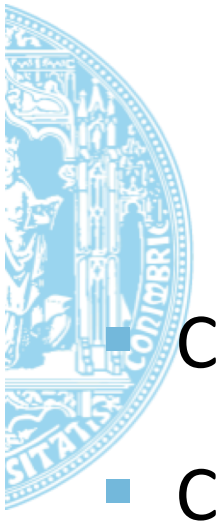
Advances on the Detection of Software Vulnerabilities

LADC 2019
Natal, RN, Brazil
Nov. 19th, 2019

Marco Vieira
mvieira@dei.uc.pt
Department of Informatics Engineering
University of Coimbra - Portugal



OUTLINE

- Concepts on Software Vulnerabilities
- Common Vulnerability Detection Techniques
- Benchmarking Vulnerability Detection Tools
- Recent Advances:
  - Static Analysis: phpSAFE & Exploring Diversity
  - Penetration Testing: Sign-WS
  - Software Metrics to Predict Vulnerabilities
  - Combining Multiple Techniques: Ongoing Work
- Conclusions

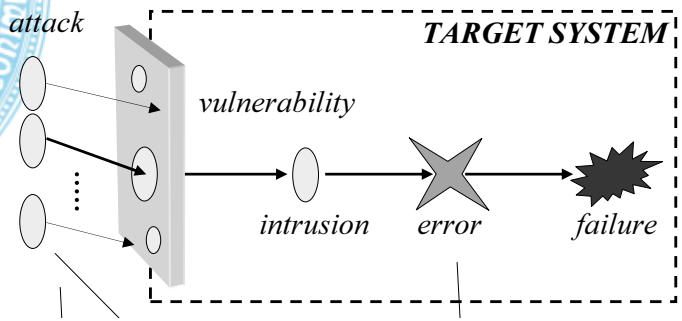
What is a vulnerability?

A ***weakness*** that may allow ***attackers*** to gain access to the system or info

- [Stock07]

- There are many causes:
  - Complexity
  - Design flaws
  - Software bugs
    - Unchecked user input
  - Password and privileges management flaws

VULNERABILITY *VS* FAULT



Fault → Error → Failure

(attack + vulnerability) → intrusion

- Attack
  - Malicious activities that intentionally attempt to violate security properties of the system
- Vulnerability *is a **fault** that leave space for malicious exploitation of a system*
- Intrusion
  - An attack that successfully activates a vulnerability

Neves et al. "Using Attack Injection to Discover New Vulnerabilities", DSN'06

AN IMPORTANT PROBLEM ...

- Create and feed an underground economy

- Companies are aware of that:
  - OWASP Security Spending Benchmarks shows that investment in security is **increasing**

However...

… THAT IS NOT GETTING BETTER!

- NTA Web Application Security Reports show that Web Security is **decreasing**

- According to the WhiteHat Security Website Security Statistics Report, **63%** of assessed websites are vulnerable

- Something is **wrong** in the development of web applications!

SOFTWARE VULNERABILITIES

- Most security problems in computer systems are related to **unknown** attacks and vulnerabilities
  - Very hard to prevent!
  - Some anomaly detection systems can catch one or another attack by detecting abnormal activities
    - Not much more to do other than apply best practices and "pray" ☺
- But, there are vulnerabilities / attacks that **we know!**
  - Some of these vulnerabilities are considered as “solved”
- **Knowing them does not make them less devastating**
  - **SQL Injection** is a good example
    - it surely has the potential for catastrophic consequences…
    - … and it is somewhat easy to avoid!

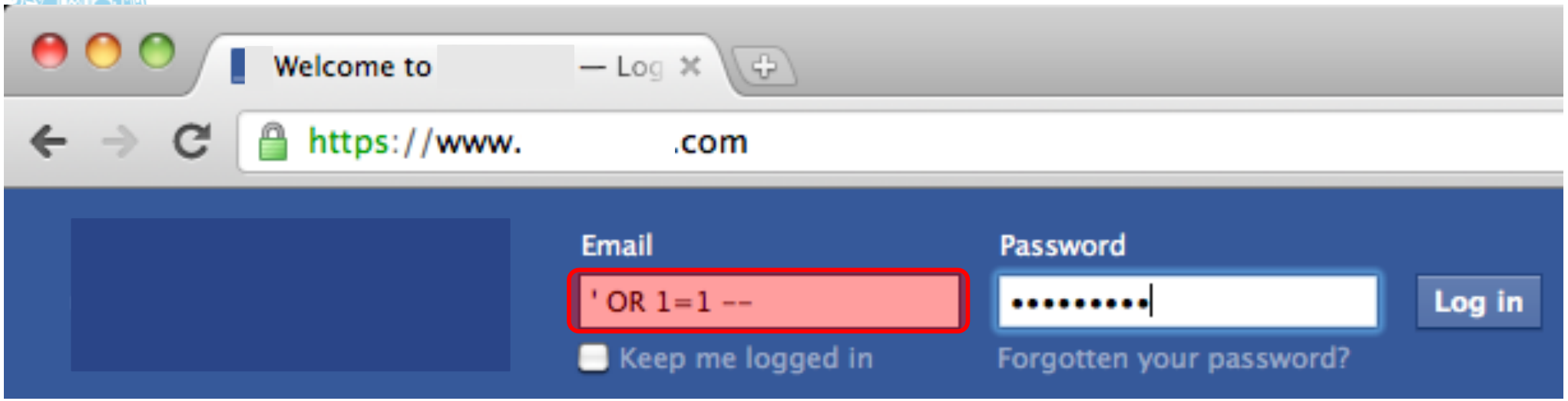
Web Application Vulnerabilities

- Web applications are widely exposed
  - Publicly visible face of an organization
  - Became the preferred targets for hackers
- Hackers moved focus from the network to applications
- Traditional security mechanisms are not effective
  - Firewall, IDS, encryption can't mitigate these vulnerabilities
- Application level attacks
  - Use network ports that are used for regular web traffic
  - Use specially tampered values
  - Exploit inputs of improperly coded applications

OWASP Top-10 Vulnerabilities

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

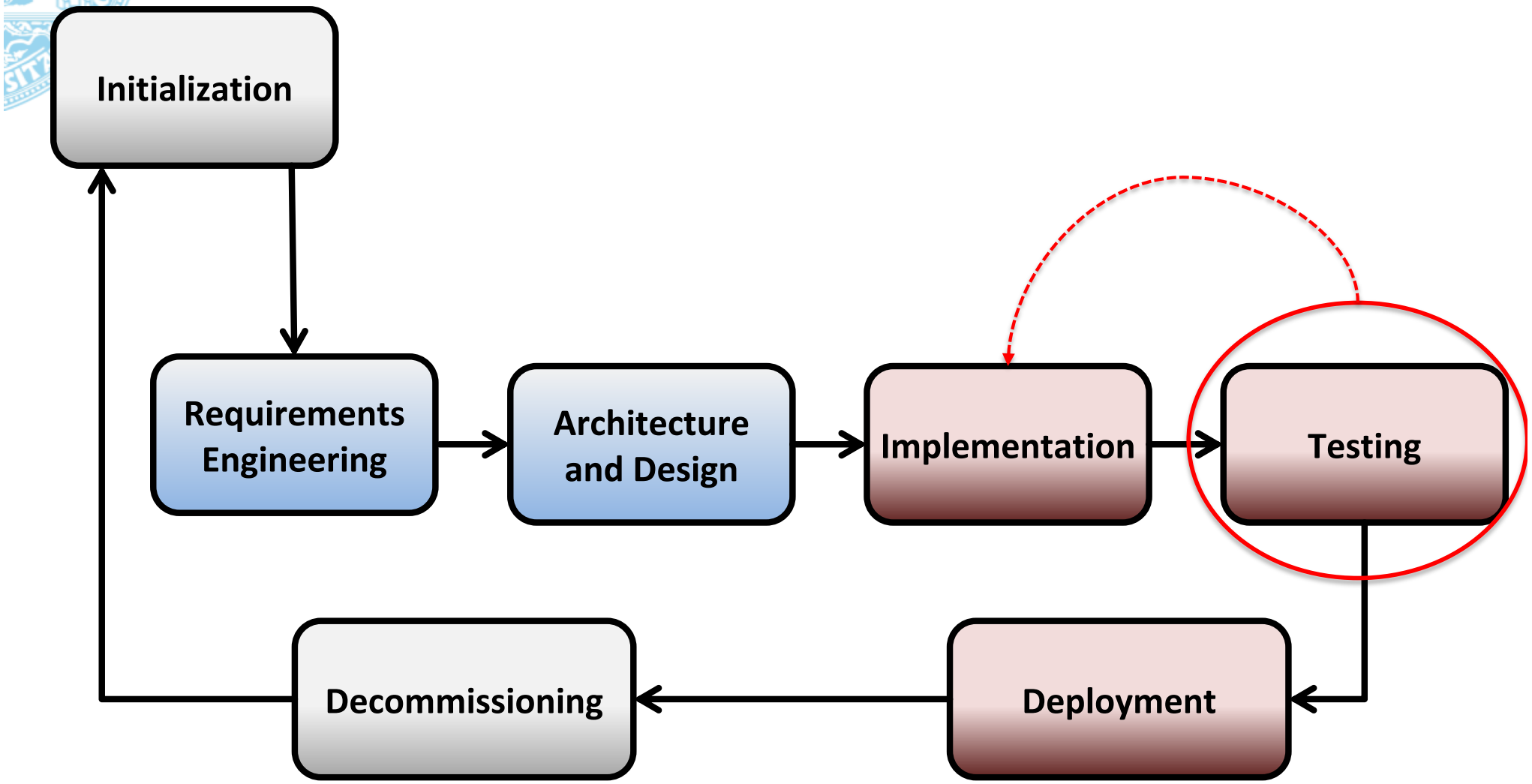
EXAMPLE OF SQL INJECTION

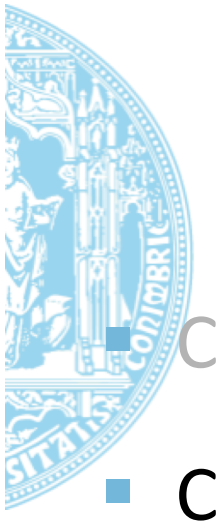



```
(…)
String sql = "SELECT * FROM users WHERE "+
    "username='" + Email + "' AND "+
    "password='" + Password + "'";
(…)
```

```
"SELECT * FROM users WHERE
    username='' OR 1=1 -- ' AND
    password=''";
```

Security in the Development Lifecycle





OUTLINE

- Concepts on Software Vulnerabilities
- Common Vulnerability Detection Techniques
- Benchmarking Vulnerability Detection Tools
- Recent Advances:
  - Static Analysis: phpSAFE & Exploring Diversity
  - Penetration Testing: Sign-WS
  - Software Metrics to Predict Vulnerabilities
  - Combining Multiple Techniques: Ongoing Work
- Conclusions

Vulnerability Detection

- **Goal:** find vulnerabilities, so they can be corrected!
- Several techniques available
  - **White box analysis**, **black box testing**, dynamic analysis, white box testing, anomaly detection, …
  - Can usually be performed both **manually** or using automated **tools**
- **Detection coverage** is important
  - We don't want to leave vulnerabilities undetected
- **False positives** are also important
  - Don't want to waste resources fixing non existing problems!

NOTES…

- All these techniques can be used for more than just detecting vulnerabilities
  - But here, we are **mainly interested** in this perspective
- Although not always the most effective, static analysis and penetration testing are still…
  - … the most widely known
  - … the most used
  - … the ones with more tools implementing it
    - and thus have the better cost/benefit relation in most cases

STATIC CODE ANALYSIS TOOLS

- Provide an automatic way for highlighting possible coding errors
- The analysis varies depending on the tool sophistication
  - Ranging from tools that consider only individual statements and declarations
  - To others that consider the complete code
- Have other usages
  - e.g., model checking and data flow analysis

EXAMPLES OF LIMITATIONS

```
public void operation(String str) {
  int i = Integer.parseInt(str);
  try {
    String sql = "DELETE FROM table" +
                 "WHERE id='" + str + "'";
    statement.executeUpdate(sql);
  } catch (SQLException se) {}
}
```

Analyzers identify the vulnerability because the SQL query is a non-constant string

```
public String dumpDepositInfo(String str) {
  try {
    String path = "//DepositInfo/Deposit"+
                  "[@accNum='" + str + "']";
    return csvFromPath(path);
  } catch (XPathException e) {}
  return null;
}
```

Depending on the complexity of csvFromPath method, a static analysis tool may not be able to find the vulnerability

CAN YOU FIND THE BUG?

```
if (listeners == null)
    listeners.remove(listener);
```

- JDK1.6.0, b105, sun.awt.x11.XMSelection
  - lines 243-244
- Nobody is perfect ☺
- Everyone makes syntax errors, but the compiler catches them…
  - What about bugs one step ahead?

Penetration Testing Tools

- Provide an automatic way to search for vulnerabilities
- Avoid the repetitive and tedious task of doing hundreds or even thousands of tests by hand
- Many tools available
  - Including commercial and open-source
- Different tools target different types of vulnerabilities
- The effectiveness of penetration testing tools is doubtful

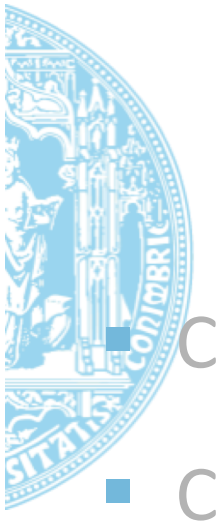
EXAMPLES OF LIMITATIONS

```
public void operation(String str) {
  try {
   String sql = "DELETE FROM table" +
            "WHERE id='" + str + "'";
   statement.executeUpdate(sql);
  } catch (SQLException se) {}
}
```

No return value and exceptions related with SQL malformation do not leak out to the invocator

```
public String dumpDepositInfo(String str) {
 try {
  String path = "//DepositInfo/Deposit"+
                "[@accNum='" + str + "']";
  return csvFromPath(path);
 } catch (XPathException e) {}
 return null;
}
```

Lack of output information



OUTLINE

- Concepts on Software Vulnerabilities
- Common Vulnerability Detection Techniques
- Benchmarking Vulnerability Detection Tools
- Recent Advances:
  - Static Analysis: phpSAFE & Exploring Diversity
  - Penetration Testing: Sign-WS
  - Software Metrics to Predict Vulnerabilities
  - Combining Multiple Techniques: Ongoing Work
- Conclusions

BENCHMARKING VD TOOLS

- Benchmarks are standard approaches to evaluate and compare different systems
  - According to specific characteristics
- Evaluate and compare the existing tools
- Select the most effective tools
- Guide the improvement of methodologies
  - As performance benchmarks have contributed to improve performance of systems

Benchmarking Components

- Workload:
  - Work that a tool must perform during the benchmark execution
  - i.e. services to exercise the Vulnerability Detection Tools
- Measures:
  - Characterize the effectiveness of the tools
  - Must be easy to understand
  - Must allow the comparison among different tools
- Procedure:
  - The procedures and rules that must be followed during the benchmark execution

WORKLOAD

Benchmark	Service Name	Vuln. Inputs	Vuln. Queries	LOC	Avg. C. Comp.
TPC-App	ProductDetail	0	0	121	5
	NewProducts	15	1	103	4.5
	NewCustomer	1	4	205	5.6
	ChangePaymentMethod	2	1	99	5
TPC-C	Delivery	2	7	227	21
	NewOrder	3	5	331	33
	OrderStatus	4	5	209	13
	Payment	6	11	327	25
	StockLevel	2	2	80	4
TPC-W	AdminUpdate	2	1	81	5
	CreateNewCustomer	11	4	163	3
	CreateShoppingCart	0	0	207	2.67
	DoAuthorSearch	1	1	44	3
	DoSubjectSearch	1	1	45	3
	DoTitleSearch	1	1	45	3
	GetBestSellers	1	1	62	3
	GetCustomer	1	1	46	4
	GetMostRecentOrder	1	1	129	6
	GetNewProducts	1	1	50	3
	GetPassword	1	1	40	2
	GetUsername	0	0	40	2
Total		**56**	**49**	**2654**	-

Enhancing the Workload

- To create a more realistic workload we created new versions of the services

- For each web service we have:
  - one version without known vulnerabilities
  - one version with N vulnerabilities
  - N versions with one vulnerable SQL query each

- This accounts for:

Services + Versions	Vuln. Inputs	Vuln. lines
80	158	87

MEASURES

- Computed from the information collected during the execution of the vulnerability detection tools
- Relative measures
  - Can be used for comparison or for improvement and tuning
- Different tools report vulnerabilities in different ways
  - Precision
  - Recall
  - F-Measure

$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TV}$$

$$F\text{-}Measure = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

PROCEDURE

- Step 1: Preparation
  - Select the tools to be benchmarked
- Step 2: Execution
  - Use the tools to detect vulnerabilities in the workload
- Step 3: Measures calculation
  - Analyze the vulnerabilities reported by the tools and calculate the measures.
- Step 4: Ranking and selection
  - Rank the tools using the measures
  - Select the most effective tool

Step 1: Preparation

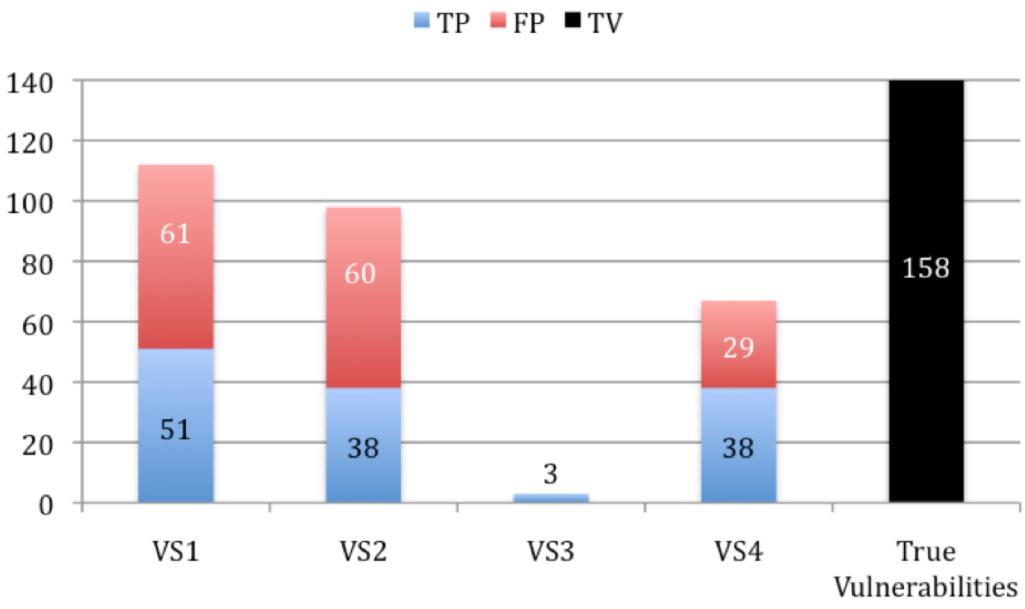
The tools under benchmarking:

Provider	Tool	Technique
HP	WebInspect	Penetration testing
IBM	Rational AppScan	
Acunetix	Web Vulnerability Scanner	
Univ. Coimbra	VS-WS	
Univ. Maryland	FindBugs	Static code analysis
SourceForge	Yasca	
JetBrains	IntelliJ IDEA	
Univ. Coimbra	RAD-WS	Anomaly detection

- Vulnerability Scanners: VS1, VS2, VS3, VS4
- Static code analyzers: SA1, SA2, SA3

STEP 2: EXECUTION

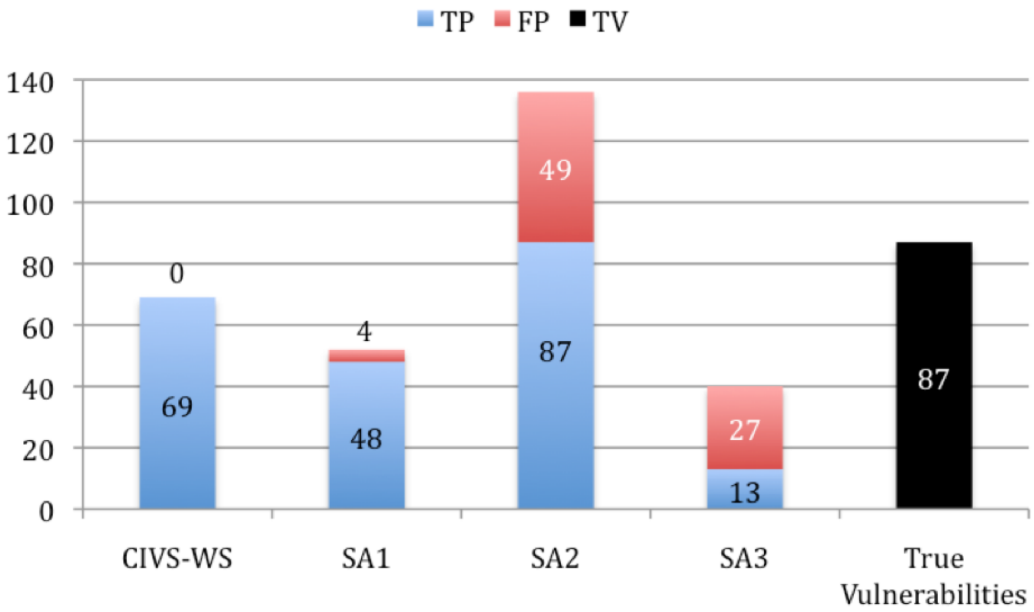
Results for Penetration Testing:



Tool	% TP	% FP
VS1	32.28%	54.46%
VS2	24.05%	61.22%
VS3	1.9%	0%
VS4	24.05%	43.28%

STEP 2: EXECUTION

Results for Static Analysis and Anomaly Detection:



Tool	% TP	% FP
CIVS	79.31%	0%
SA1	55.17%	7.69%
SA2	100%	36.03%
SA3	14.94%	67.50%

STEP 3: MEASURES CALCULATION

Benchmarking results:

Tool	F-Measure	Precision	Recall
CIVS-WS	0.885	1	0.793
SA1	0.691	0.923	0.552
SA2	0.780	0.640	1
SA3	0.204	0.325	0.149
VS1	0.378	0.455	0.323
VS2	0.297	0.388	0.241
VS3	0.037	1	0.019
VS4	0.338	0.567	0.241

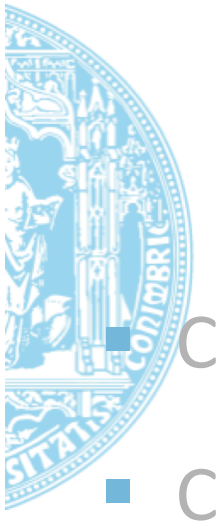
STEP 4: RANKING AND SELECTION

- Rank the tools using the measures
- Select the most effective tool:

	Criteria	1st	2nd	3rd	4th
Inputs	F-Measure	VS1	VS4	VS2	VS3
	Precision	VS3	VS4	VS1	VS2
	Recall	VS1	VS2/VS4		VS3
Queries	F-Measure	CIVS	SA2	SA1	SA3
	Precision	CIVS	SA1	SA2	SA3
	Recall	SA2	CIVS	SA1	SA3

So, About the Tools…

- Not so effective even for a very well-known type of vulnerability…
- More visibility improves the effectiveness
- Static analysis outperforms penetration testing in terms of coverage
  - But the reverse seems to happen regarding false positives
- How to improve the current situation?
  - Further research is need…



OUTLINE

- Concepts on Software Vulnerabilities
- Common Vulnerability Detection Techniques
- Benchmarking Vulnerability Detection Tools
- Recent Advances:
  - Static Analysis: phpSAFE & Exploring Diversity
  - Penetration Testing: Sign-WS
  - Software Metrics to Predict Vulnerabilities
  - Combining Multiple Techniques: Ongoing Work
- Conclusions

PHP Static Analysis Tools

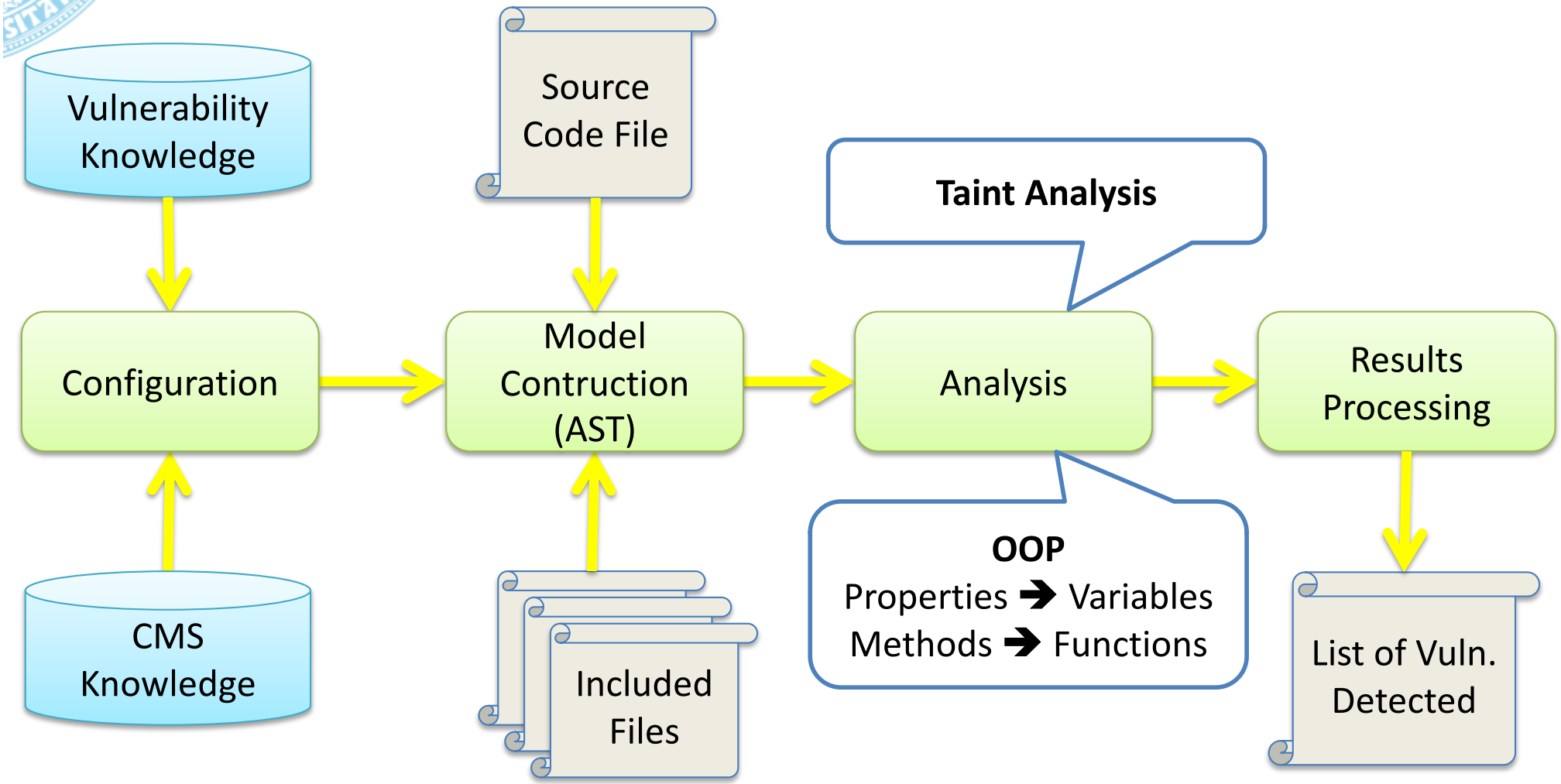
	RIPS rips-scanner.sourceforge.net	Pixy packetstormsecurity.com/files/download/64088/pixy_3_03.zip
Year deployed	**2010**	**2006**
Supported vulnerabilities	XSS, SQLi, others	XSS, SQLi
Plugin support	**Yes**	**No**
OOP support	**No**	**No**

4 out of 5 top CMS frameworks use **OOP**

PHPSAFE

github.com/JoseCarlosFonseca/phpSAFE

Static Analyzer for WordPress plugins



Model Construction - AST

```php
<?php
    include('functions.php');
    $a = $_GET['firstname'];
    echo $a;
?>
```

token_get_all

#	name	value	line
0	T_OPEN_TAG	<?php	1
1	T_INCLUDE	include	2
2		(	
3	T_CONSTANT_ENCAPSED_STRING	'functions.php'	2
4		)	
5		;	
6	T_VARIABLE	$a	3
7		=	
8	T_VARIABLE	$_GET	3
9		[	
10	T_CONSTANT_ENCAPSED_STRING	'firstname'	3
11		]	
12		;	
13	T_ECHO	echo	4
14	T_VARIABLE	$a	4
15		;	
16	T_CLOSE_TAG	?>	5
17	T_INLINE_HTML		6

Abstract Syntax Tree (AST) with token_get_all (php function)

TAINT ANALYSIS

- All input is tainted
- The propagation is also tainted
- Tainted variables are the path to attacks
- Can be untainted by a set of operations

Untaint operations may be specific

`mysql_real_escape_string` ➔ ~~SQLi~~ XSS

`htmlentities` ➔ SQLi ~~XSS~~

`intval` ➔ ~~SQLi~~ ~~XSS~~

OOP READY

Example from *mail-subscribe-list 2.0.9*

– Now fixed thanks to this work

Entry Point

```
$results = $wpdb->get_results("SELECT * FROM ". $wpdb->prefix."sml");
foreach($results as $row) {
  echo '<tr> <td>'.$row->sml_email.'</td></tr>';
}
```

Sensitive sink

XSS vulnerability (Stored)

#	index	name	object	class	scope	input	output	function	file	line	tainted	vulnerability	dependencies_index
4	83	$row->sml_email	$row		local	regular	output	function	index.php	93	tainted	Cross Site Scripting	80

phpSAFE Output

Example from *mail-subscribe-list 2.0.9*

– Now fixed thanks to this work

Vulnerable Variables

Show Count: 6

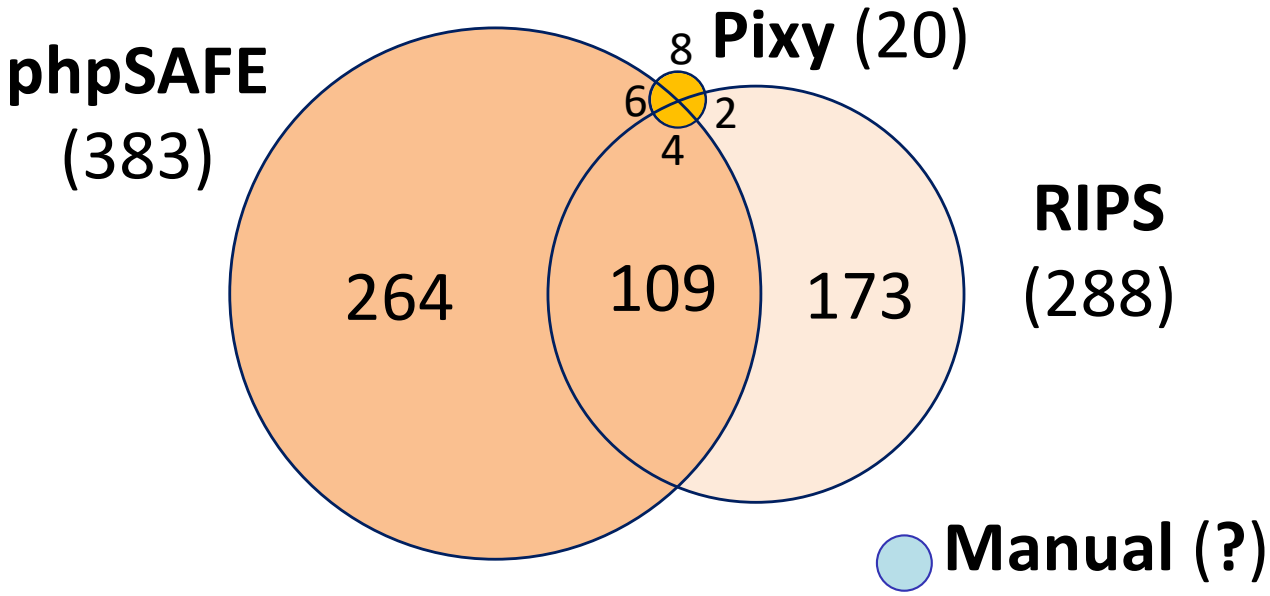
#	index	name	object	class	scope	input	output	function	file	line	tainted	vulnerability	dependencies_index
0	1	$_GET['page']			local	input	output	function	index.php	13	tainted	Cross Site Scripting	
1	13	$id			local	regular	output	function	index.php	21	tainted	SQL Injection	11
2	81	$row->id	$row		local	regular	output	function	index.php	91	tainted	Cross Site Scripting	80
3	82	$row->sml_name	$row		local	regular	output	function	index.php	92	tainted	Cross Site Scripting	80
4	83	$row->sml_email	$row		local	regular	output	function	index.php	93	tainted	Cross Site Scripting	80
5	85	$_GET['page']			local	input	output	function	index.php	121	tainted	Cross Site Scripting	1

Experimental Study

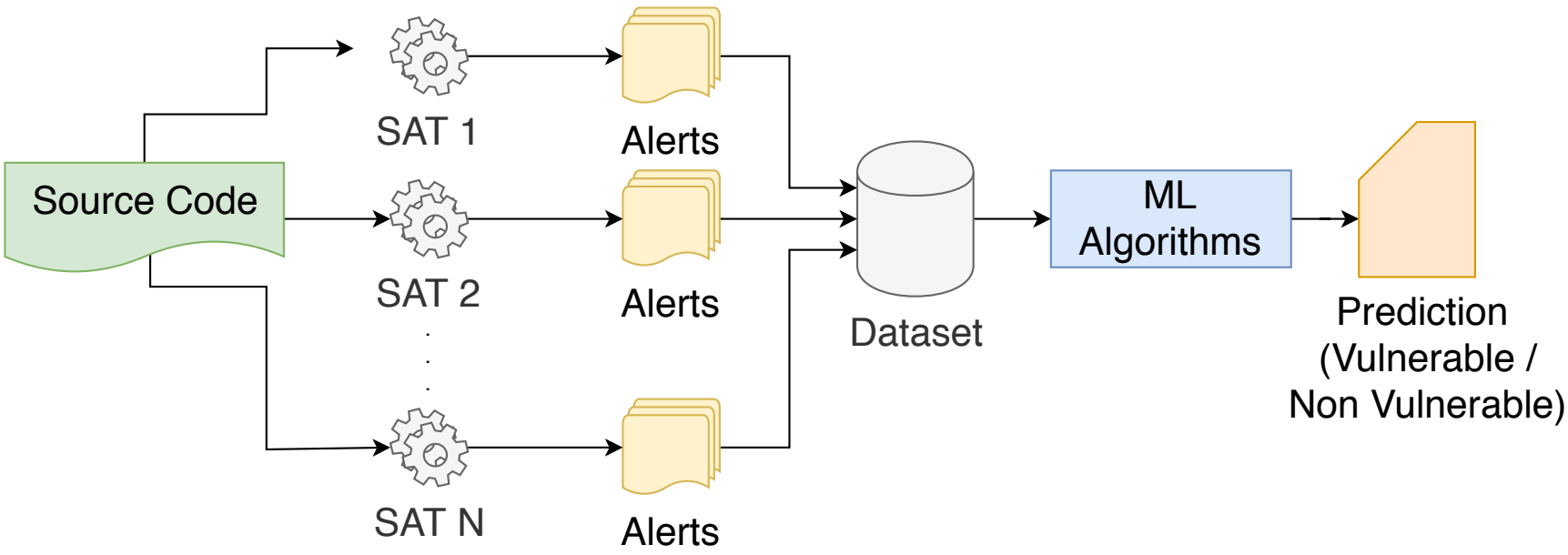
- Select a set of:
  - 35 WordPress plugins
- Execute static analysis tools
  - phpSAFE
  - RIPS
  - Pixy
- Analysis of results
  - Manual verification
  - Data analysis

phpSAFE vs RIPS vs Pixy

	RIPS	Pixy	phpSAFE
True Positives	288	20	383
Precision	86%	9%	86%
Recall	52%	3%	66%
F-score	65%	5%	75%



Exploring Diversity

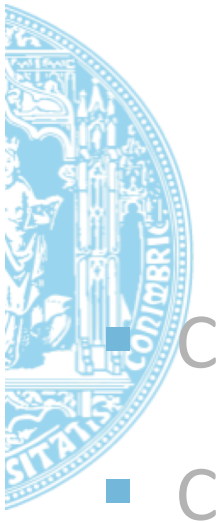


Overall Results

	TN	FP	FN	TP
SQLi	94.66% (4,570)	5.34% (258)	6.94% (50)	93.06% (670)
XSS	95.18% (20,505)	4.82% (1,038)	0.10% (5)	99.90% (4,930)

	Precision	Recall	F-Measure
SQLi	0.722	0.931	0.813
XSS	0.826	0.999	0.904

More tomorrow @ Technical Session 3



OUTLINE

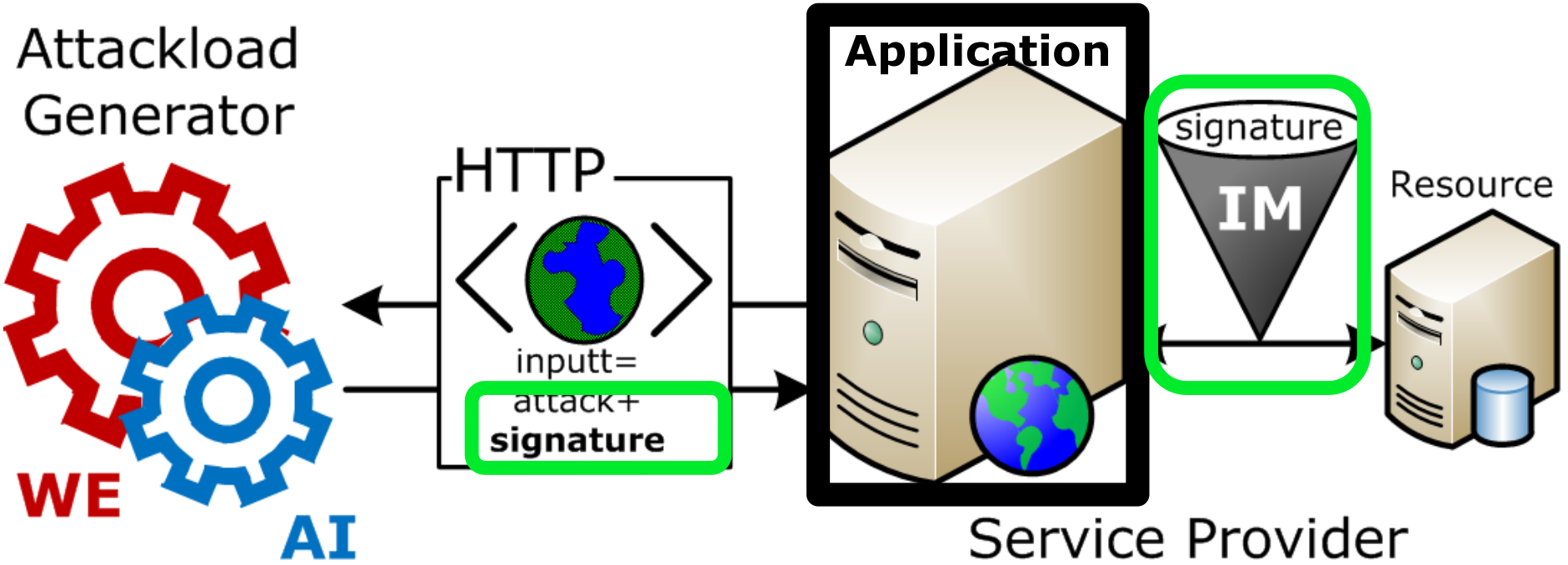
- Concepts on Software Vulnerabilities
- Common Vulnerability Detection Techniques
- Benchmarking Vulnerability Detection Tools
- Recent Advances:
  - Static Analysis: phpSAFE & Exploring Diversity
  - Penetration Testing: Sign-WS
  - Software Metrics to Predict Vulnerabilities
  - Combining Multiple Techniques: Ongoing Work
- Conclusions

SIGN-WS

- Adds visibility to the process testing process
  - Yet, keeping it as black-box as possible
- How? It uses
  Interface Monitoring together with Attack Signatures
  - It is possible to obtain the information necessary to improve the Penetration Testing process…
  - … without accessing or modifying the internals of the application!!!
- **Key assumption**: injection attacks manifest in the interfaces of the attacked web service

[Antunes11]

SIGN-WS


Attackload
Generator
WE
AI
HTTP
inputt=
attack+
signature
Application
signature
IM
Resource
Service Provider

ATTACK SIGNATURE

- Token introduced inside an injection attack
- In a successful attack the token is:
  - Observable in the interfaces of the service
  - Active: outside literal strings
  - Changing the structure of the backed command

```
Active:   select n from t where dsc LIKE '%input' TOKEN%'
Inactive: Select n from t where dsc LIKE '%input TOKEN%'
```

- A successful token must be:
  - Unambiguous
  - Inoffensive
  - Informative
  - Short

Perform Signed Injection Attacks

- Reversed token used to confirm vulnerabilities
  - Reinforce unambiguity and avoid false positives and
- Proposed Model:
  - Delimiters "_"
  - Identifiers
  - Qualifier "o|p"

Regular	Reversed
_12_p	_21_o

- Token must be carefully placed inside the malicious string
  - The location depends on the vulnerabilities tested
  - Must be easily configurable

Interface Monitoring

- Multiple options available:
  - Network packet sniffing
  - Proxy
  - Driver instrumentation
- Process commands to find “active” signatures
  1. Remove escaped slashes, apostrophes and quotes
  2. Remove literal strings
  3. Remaining signatures are active

```
1: select n from t where dsc LIKE '%input' _28_p%';
2: select n from t where dsc LIKE '%input' _28_p%';
3: select n from t where dsc LIKE _28_p%';
```

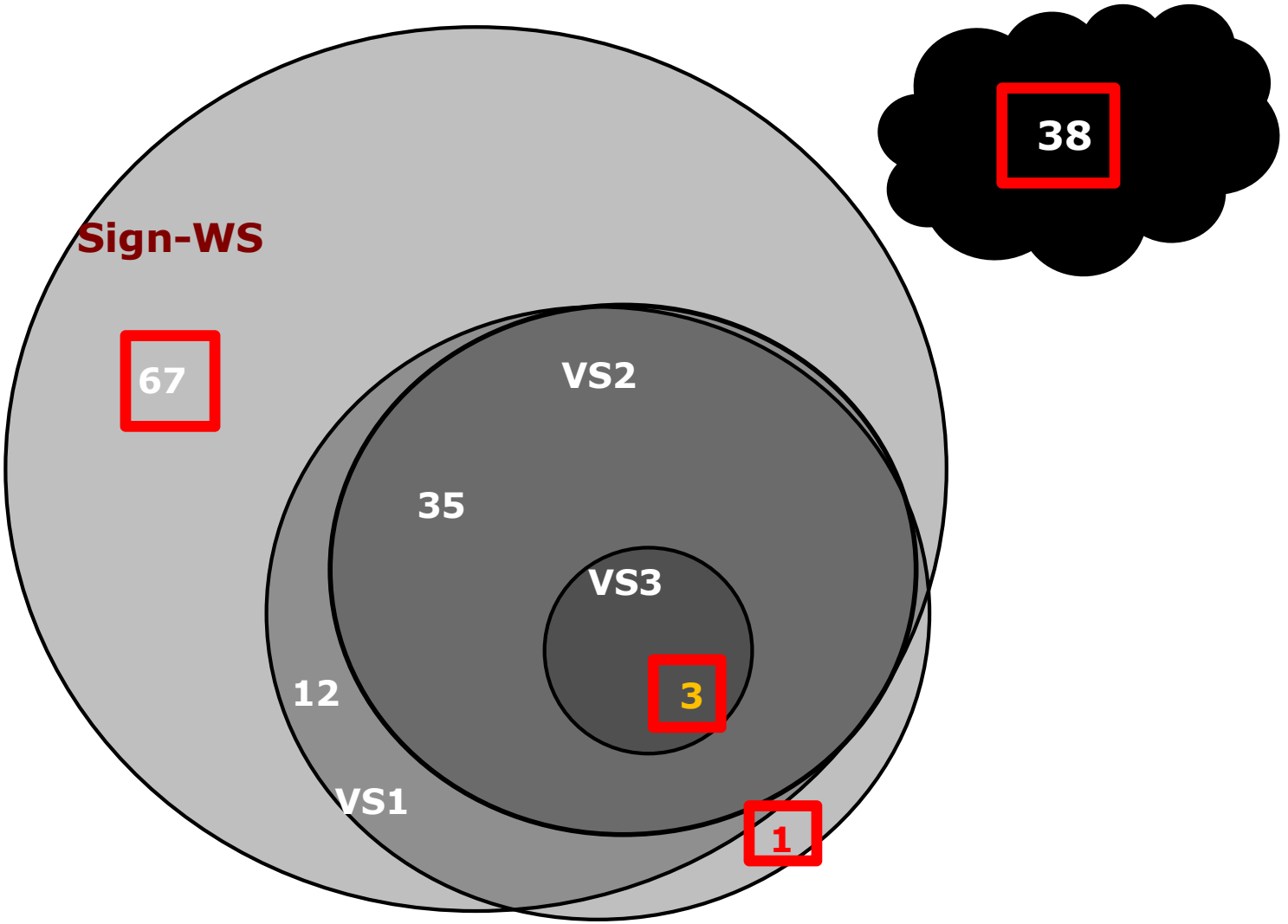
```
1: select n from t where dsc LIKE '%input\' _28_p%';
2: select n from t where dsc LIKE '%input _28_p%';
3: Select n from t where dsc LIKE ;
```

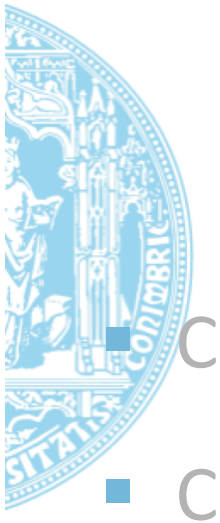
Overall Results

Benchmark for SQLI Vulnerability Detection Tools:

Tool	True Vulnerabilities	True Positives	False Positives	Coverage	F. Detection Rate
Sign-WS	158	117	0	74,05%	0,00%
VS1	158	51	61	32,28%	54,46%
VS2	158	38	60	24,05%	61,22%
VS3	158	3	0	1,90%	0,00%

VULNERABILITIES INTERSECTION





OUTLINE

- Concepts on Software Vulnerabilities
- Common Vulnerability Detection Techniques
- Benchmarking Vulnerability Detection Tools
- Recent Advances:
  - Static Analysis: phpSAFE & Exploring Diversity
  - Penetration Testing: Sign-WS
  - Software Metrics and ML to Predict Vulnerabilities
  - Combining Multiple Techniques: Ongoing Work
- Conclusions

EXPERIMENTAL STUDY

Vulnerability dataset containing software metrics of functions, classes, and files of Mozilla

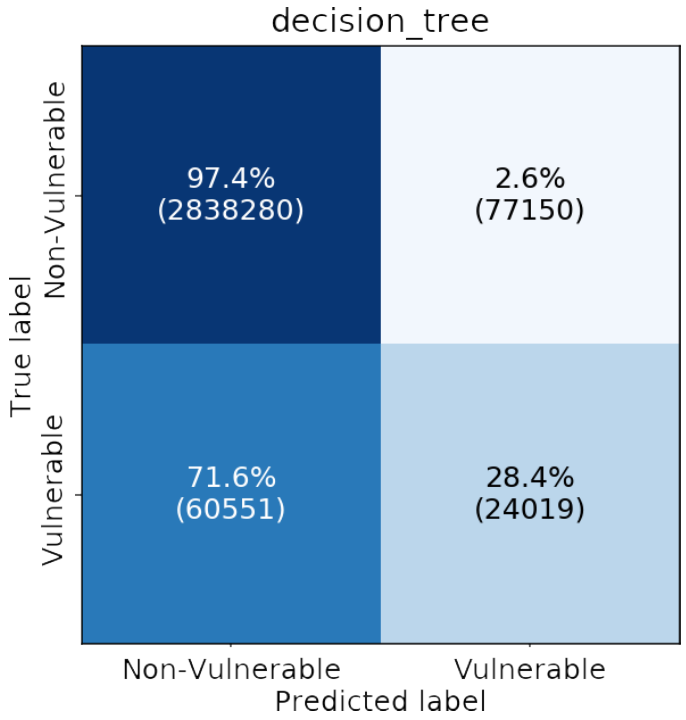
- 604304 files (2819 vulnerable), 56 metrics; limited to 100000 samples

TABLE I
EXPERIMENTS CONFIGURATIONS

Parameter	Values
Dim. Reduction	Variance, Correlation, PCA, RFE
Sampling	Random under-/oversampling, SMOTE, Near Miss, ADASYN, Instance Hardness Threshold
Sampling Ratios	Oversampling: [1, 3] , Undersampling: [1, 4]
Algorithms	SVM, Gradient Boosting, Bagging, DT, Adaboost, Extra Trees, NN, RF, k Nearest Neighbors (k-NN)

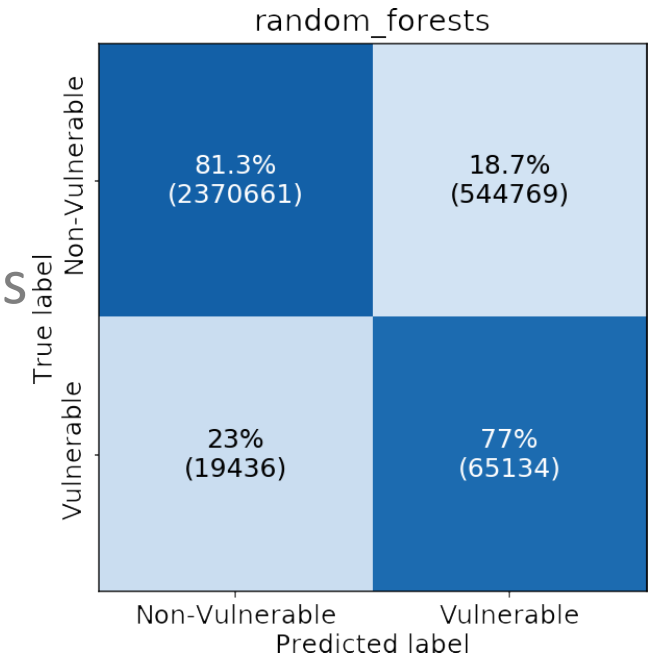
Some Results...

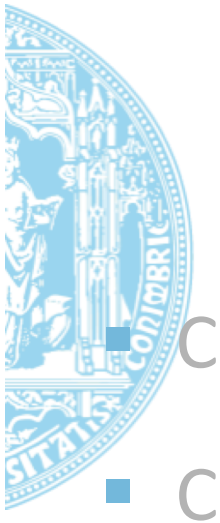
- Without sampling/dimensionality reduction no algorithm was able perform acceptably
  - "Useless" models, low FPs but low TP:
- Undersampling improved vulnerable samples prediction
  - 75% of non-vulnerable and 82% of vulnerable samples
  - many FPs, but still realistic models (acceptable ratio TNs and TPs)



Some Results...

- Dimensionality reduction lead to some more interesting results:
  - RF with feature selection by correlation predicted 88% and 69% of non-vulnerable and vulnerable
  - Some loss of TPs, significantly fewer FPs
- A fine-tuning of the best models improved results
  - RF with sampling by Instance Hardness Threshold, feature selection by variance and correlation
    - predicting 81% of non-vulnerable and 77% of vulnerable

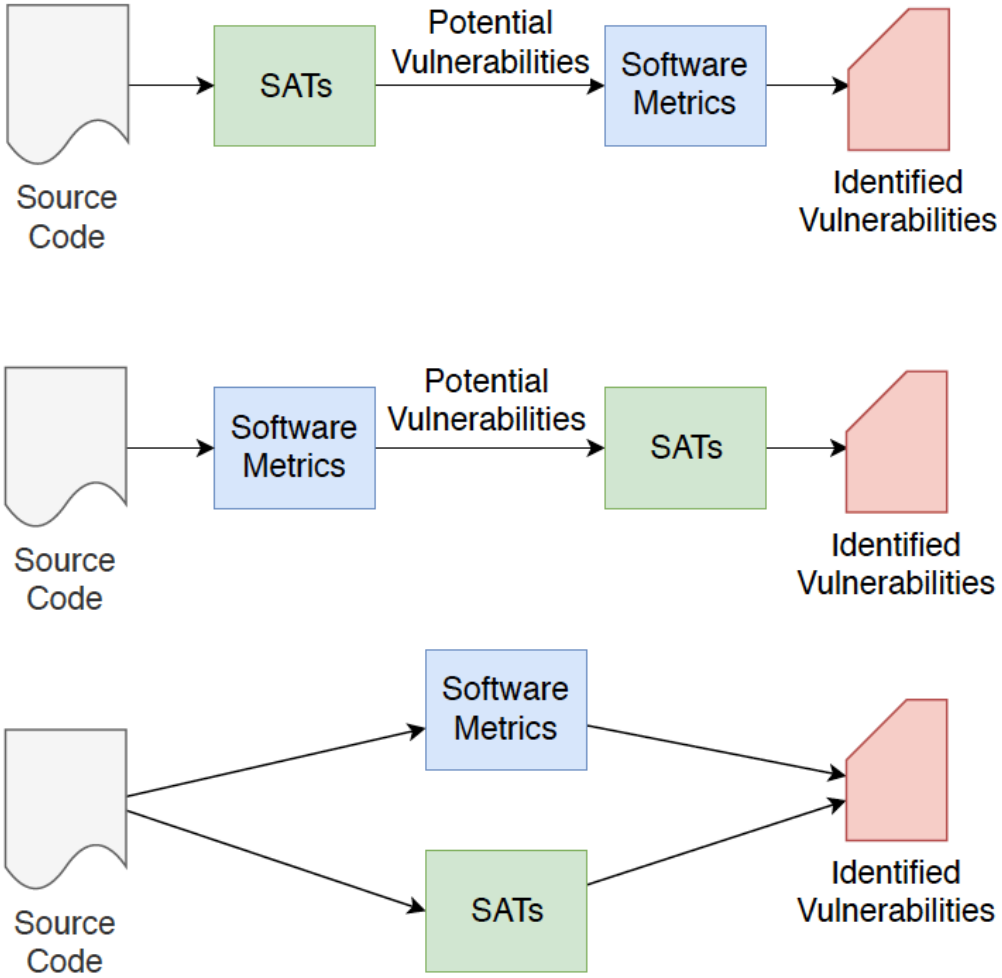




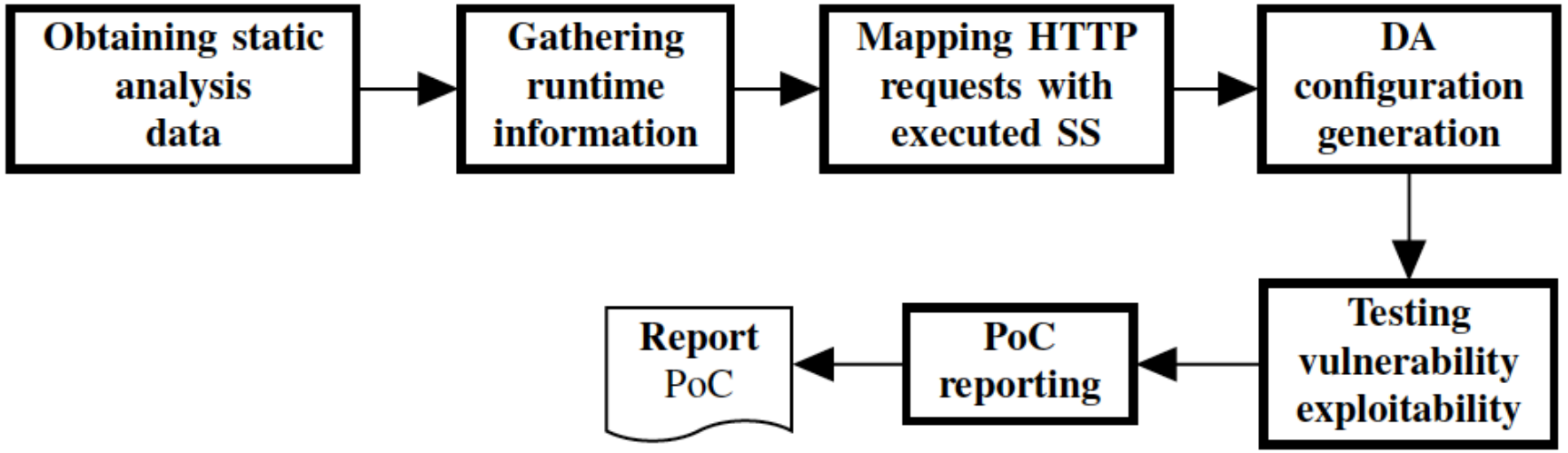
OUTLINE

- Concepts on Software Vulnerabilities
- Common Vulnerability Detection Techniques
- Benchmarking Vulnerability Detection Tools
- Recent Advances:
  - Static Analysis: phpSAFE & Exploring Diversity
  - Penetration Testing: Sign-WS
  - Software Metrics and ML to Predict Vulnerabilities
  - Combining Multiple Techniques: Ongoing Work
- Conclusions

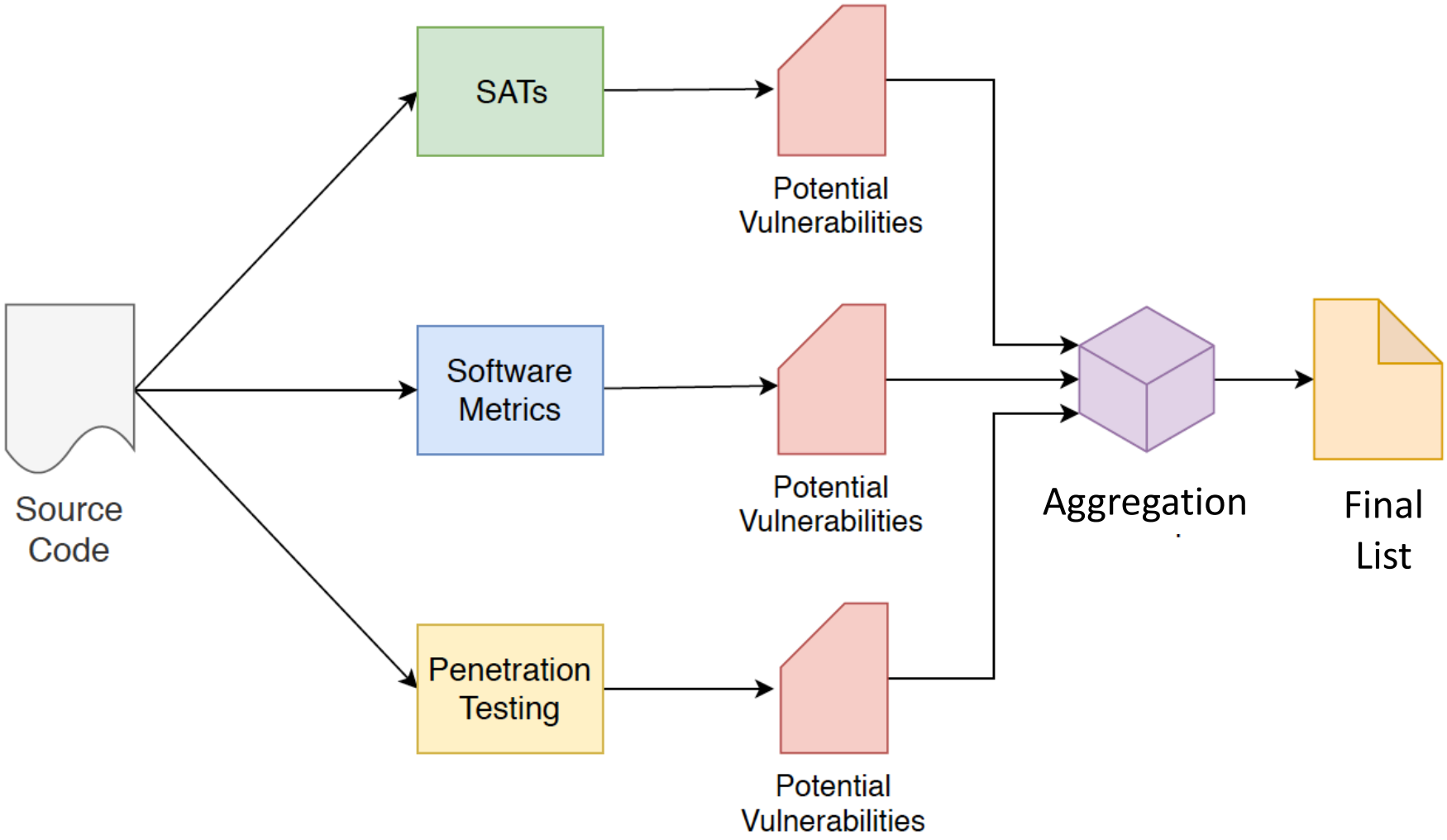
SOFTWARE METRICS TO GUIDE SCA



SCA to Guide Penetration Testing



Integration of Multiple Approaches



CONCLUSIONS

- Vulnerability detection is still an open problem
  - Different techniques exist, but performance is low
- Alternative approaches can be studied
  - e.g. using advanced Machine Learning capabilities
- Characterizing the performance of vulnerability detection tools needs to be studied
  - Practical solutions for developers
- Software security, in general, needs to be studied and better integrated with the development lifecycle

More Questions than Answers

QUESTIONS?

Marco Vieira
Department of Informatics Engineering
University of Coimbra
mvieira@dei.uc.pt
http://eden.dei.uc.pt/~mvieira