Predicting Vulnerable Components: Software Metrics vs Text Mining

James Walden[1] Jeff Stuckman[2] Riccardo Scandariato[3]

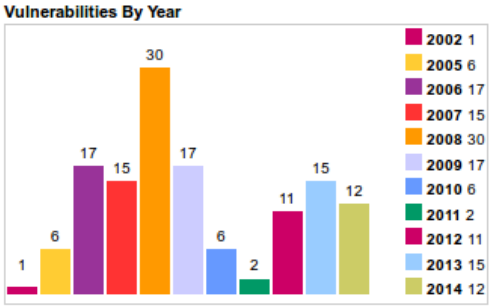
[1]Northern Kentucky University, USA
[2]University of Maryland
[3]KU Leuven

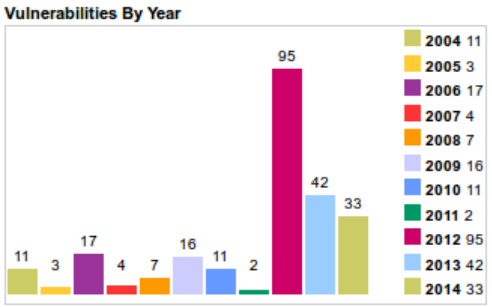
International Symposium on Software Reliability Engineering (ISSRE 2014)

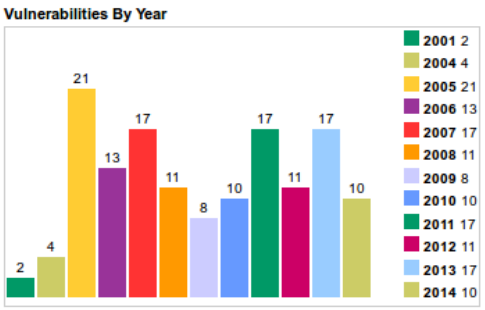
Topics

Web Application Security

Our Applications







Research Questions

Our research questions were to determine whether

1. $IsVulnerableFile \stackrel{?}{=} f(SWmetrics)$
2. $IsVulnerableFile \stackrel{?}{=} f(Textminingfeatures)$

and if such models could be constructed, did one model type perform differently than the other

$$\mathcal{R}_{\mathrm{SM}} \stackrel{?}{=} \mathcal{R}_{\mathrm{TM}}$$

Software Metrics

1. *Lines of code (LOC)*
2. *Lines of code (non-HTML)*
3. *Number of functions*
4. *Cyclomatic complexity*
5. *Maximum nesting complexity*
6. *Halstead's volume*
7. *Total external calls*
8. *Fan-in*
9. *Fan-out*
10. *Internal functions or methods called*
11. *External functions or methods called*
12. *External calls to functions or methods*

Text Mining

```
T_BREAK: 3,
T_CASE: 1,
T_ELSEIF: 1,
T_IF: 2,
T_ISSET: 1,
T_OPEN_TAG: 5,
T_PRINT: 3,
T_REQUIRE_ONCE: 2,
T_STRING: 5,
T_SWITCH: 1,
T_VARIABLE($return): 1
```

Vulnerability vs Defect Prediction

1. Fewer vulnerabilities than defects reported.
2. Fewer projects track vulnerabilities than defects.
3. The "many eyes" effect works differently:
   - Defects cause problems for users, who report them.
   - Vulnerabilities triggered by odd input sequences rarely entered by users.
   - Vulnerabilities offer opportunities to threats, who therefore keep them secret.
4. Different skills required to find vulnerabilities.
5. Developers limit vulnerability information to reduce opportunities for exploitation.
6. No repository of standard vulnerability prediction data sets for model evaluation.

Application Data

Application			
Vulnerabilities	97	51	75
Versions	30 (all 6.x)	71 (all)	95 (since 2.2.0)

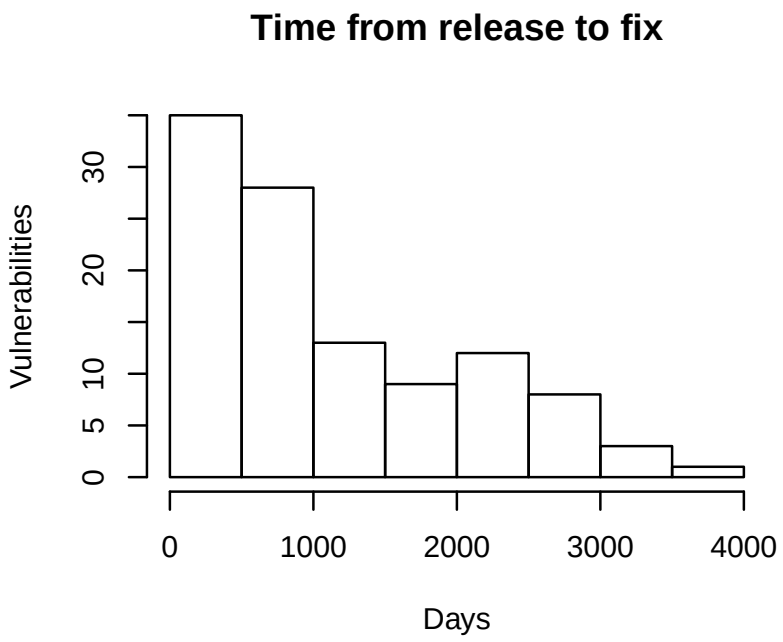
For each application, the data set includes

- Source code for each included version.
- Source code metrics for each PHP file for each version.
- Mapping of vulnerabilities to files in each version.
- Vulnerability identifiers from vendor or CVE.

Vulnerability lifetimes

Vulnerability timeline.

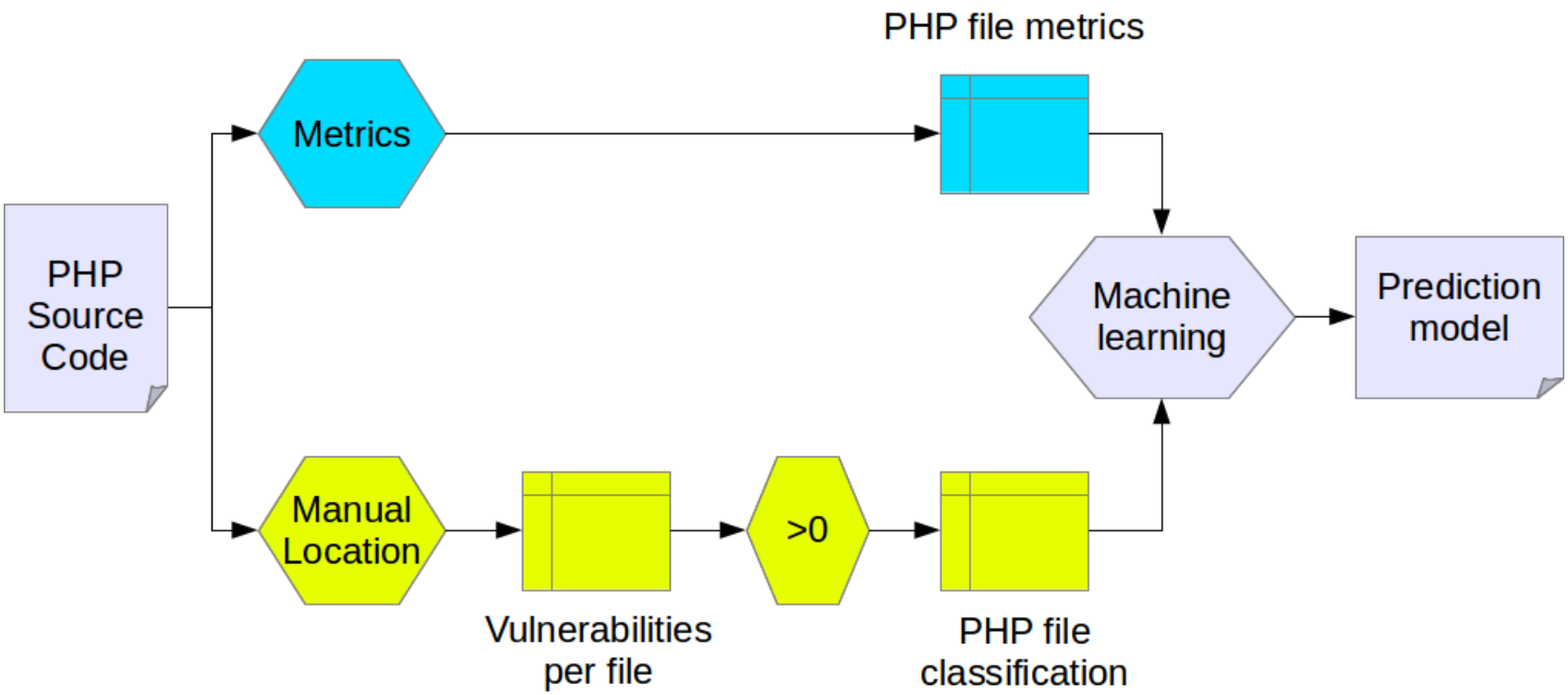
1. Vulnerability introducing commit.
2. Inclusion of vulnerable commit in a released version.
3. Persistence of vulnerability in code, potentially moving to different files through renames or refactorings.
4. Vulnerability fixing commit.
5. Inclusion of fixing commit in a released version.



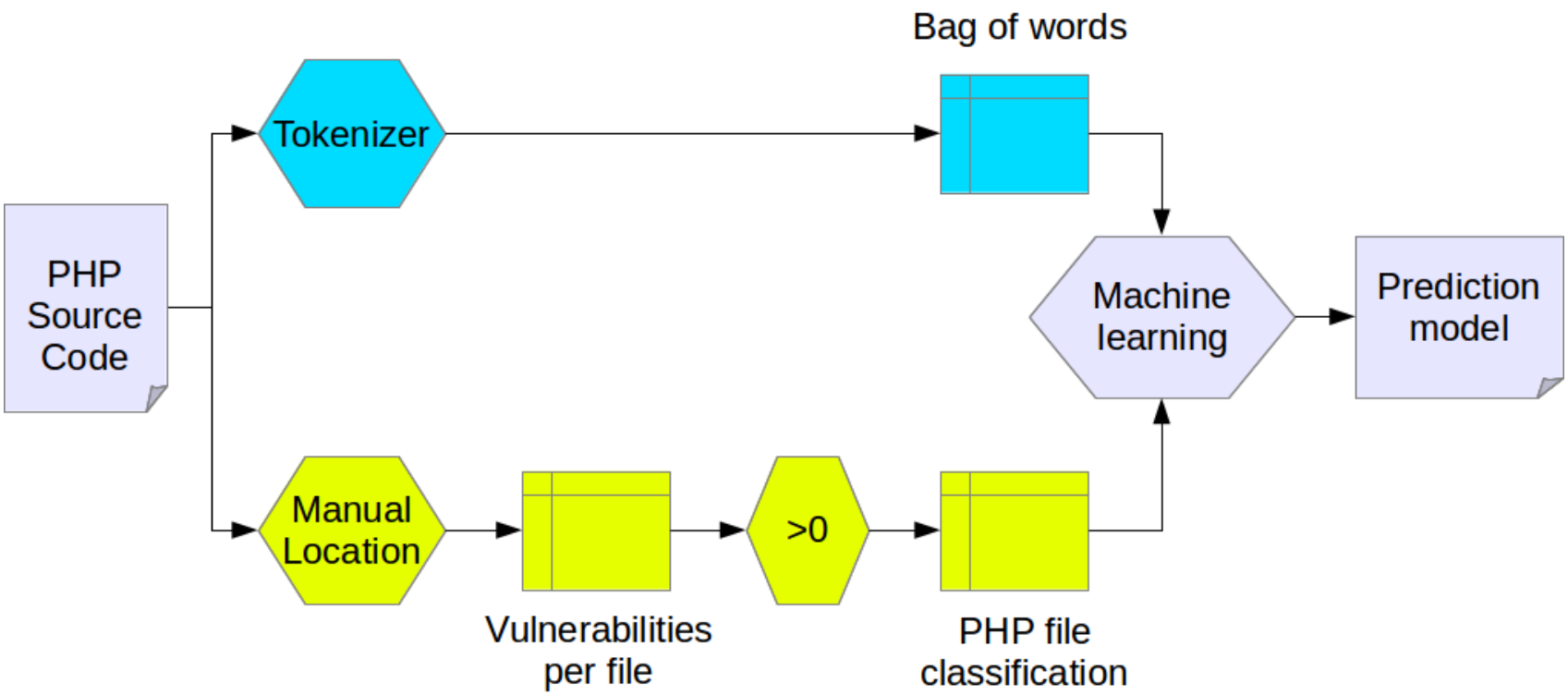
Vulnerability Location Process

1. Determine main release branch of application.
2. Identify commit which fixed the vulnerability on main branch.
3. Construct indicator string by examining code containing vulnerability.
4. Locate first version where indicator string appears.
5. Trace code changes from introduction to fix to validate vulnerability presence.

Building the Software Metrics Model

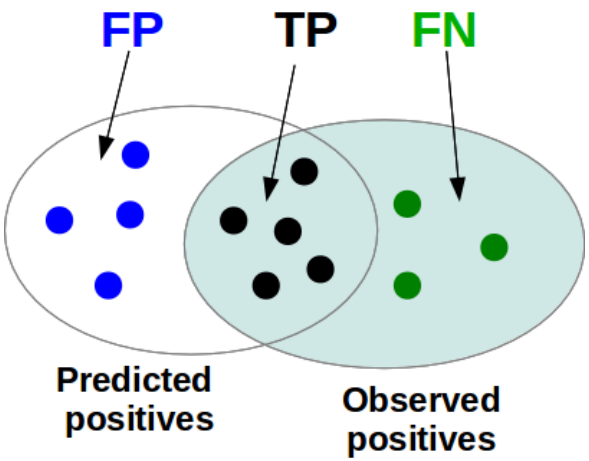


Building the Text Mining Model



Performance Indicators

Recall	Inspection Ratio
$\mathcal{R} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$	$\mathcal{I} = \frac{\mathrm{TP}+\mathrm{FP}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$
(% of vulnerable files found)	(% of files to inspect to be possible to find TP)



Undersampling

	Vulnerable files	*Total files*	*P-rate (%)*	*Text features*
Drupal	62	202	**30.68**	3886
PHPMyAdmin	27	322	**8.39**	5232
Moodle	24	2942	**0.82**	18306

Positive rates (percentage of vulnerable files) are low above, so we undersample to create a balanced training set by

- Keeping all positives.
- Randomly selecting a number of negatives equal to the number of positives.

We used Weka's `SpreadSubSample` filter to undersample.

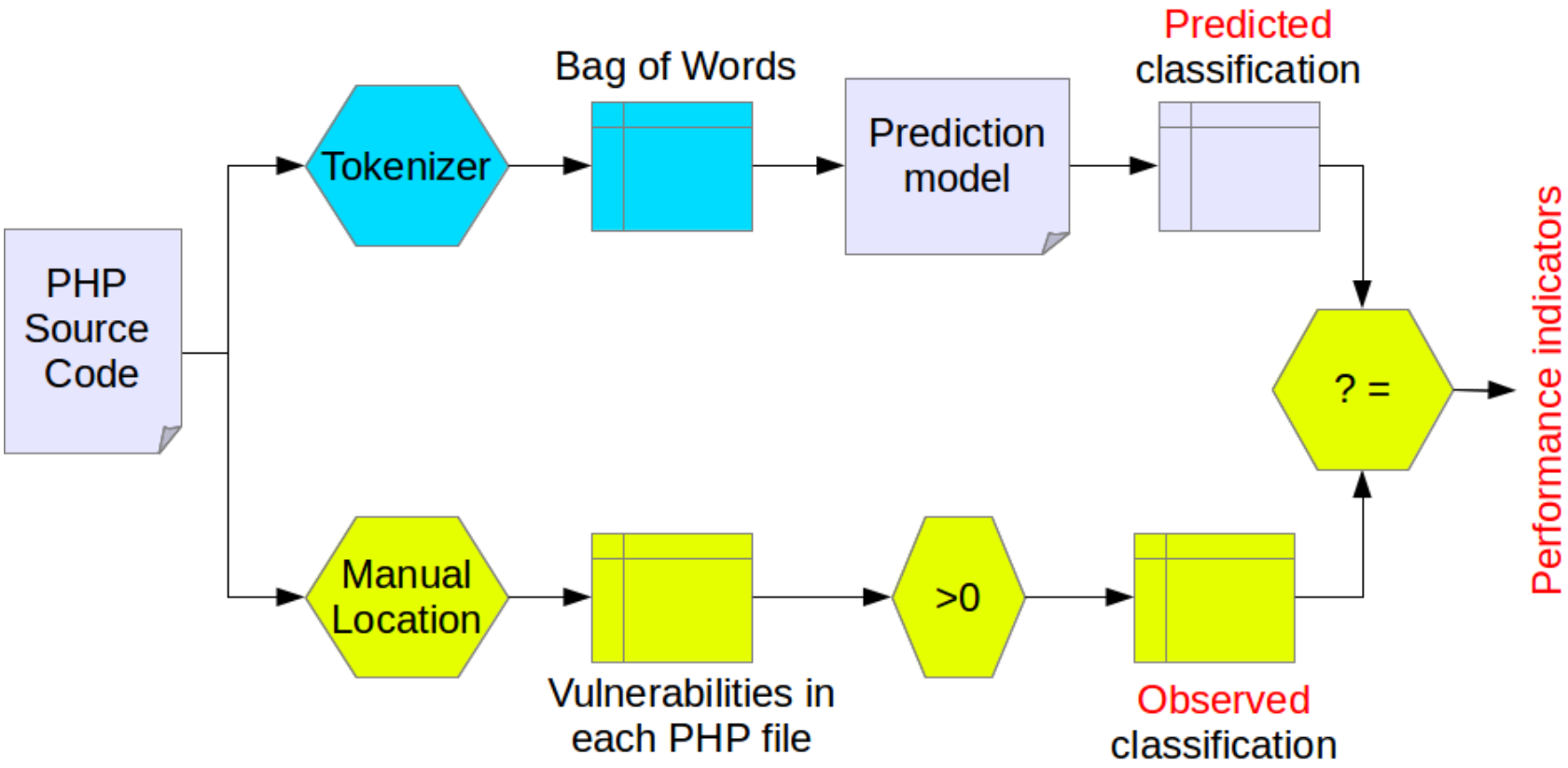
Hypotheses

Our null hypotheses are that the recall and inspection metrics will not differ between software metrics (SM) and text mining (TM).

$$H_0^{\mathcal{R}} : \mu\{\mathcal{R}_{\mathrm{SM}}\} = \mu\{\mathcal{R}_{\mathrm{TM}}\}$$
$$H_0^{\mathcal{I}} : \mu\{\mathcal{I}_{\mathrm{SM}}\} = \mu\{\mathcal{I}_{\mathrm{TM}}\}$$

We use a Wilcoxon rank-sum non-parameter test for independent samples with a significance level of 0.05 for determining if a difference exists.

Testing the Models



Cross-validation results

	Indicator		*SW metrics (%)*	*Text mining (%)*
Drupal	Recall	μ	**76.9**	**80.5**
		σ	2.8	3.3
	Inspection	μ	**45.5**	**43.3**
		σ	2.3	2.3
PHPMyAdmin	Recall	μ	**66.3**	**73.7**
		σ	12.0	9.8
	Inspection	μ	**42.0**	**43.4**
		σ	2.7	4.6
Moodle	Recall	μ	**70.4**	**80.0**
		σ	10.8	6.1
	Inspection	μ	**32.1**	**28.7**
		σ	4.1	3.7

Conclusions

1. Data set with 223 vulnerabilities in versions of 3 PHP web applications at `http://seam.cs.umd.edu/webvuldata/`.
2. Effective vulnerability prediction models can be developed for web applications.
3. Text mining outperformed software metrics with
   - Significantly better recall.
   - Almost same inspection rate (cost).
4. Models built on one application could not effectively predict vulnerable files in other applications.

Future Work

1. Build prediction models that predict vulnerable components in future versions.
2. Build prediction models based on vulnerability categories.
3. Include data for Drupal 5.x and possibly Drupal 7.x.

Extra Slides

Vulnerabilities by Class

	Drupal	Moodle	PHPMyAdmin
Code Injection	2	7	10
CSRF	8	3	1
XSS	32	9	45
Path Disclosure	0	2	12
Authorization issues	39	28	6
Other	16	2	1

Additional performance metrics

	Indicator	*SW metrics (%)*	*Text mining (%)*
Drupal	Precision	52.0	57.1
	FP rate	31.6	26.9
	Accuracy	71.0	75.4
PHPMyAdmin	Precision	13.2	14.3
	FP rate	39.8	40.6
	Accuracy	60.7	60.6
Moodle	Precision	1.8	2.3
	FP rate	31.7	28.3
	Accuracy	68.3	71.8