Predicting Vulnerable Components: Software Metrics vs Text Mining

A partial replication and extension of a paper originally published by James Walden, Jeff Stuckman, and Riccardo Scandariato

Rasheed Elsaleh for CSC666
Dept. of Computer Science
Northern Kentucky University
Highland Heights, KY, USA

Abstract— **Identifying security vulnerabilities in software is difficult. Finding vulnerabilities and building secure software can be time consuming and expensive. Vulnerability prediction models aim to identify vulnerability prone software components, which can be used to focus security efforts, thus reducing the time and effort required to build secure software. In addition, researchers attempting to find vulnerabilities can benefit from the predictions to ease the identification of vulnerabilities in software components.**

This paper is a partial replication and extension of the work done by J. Walden, J. Stuckman, and R. Scandariato in [1]. They provided a high-quality, public dataset containing 223 vulnerabilities found in three web applications. This dataset first was used to compare vulnerability prediction models based on text mining with models using software metrics as predictors. It was found that text mining models had higher recall than software metrics based models on average. The dataset was also used in several experiments including using TF-IDF with text mining to enhance prediction, to predict new vulnerabilities in future versions of the same application, and in grouping vulnerabilities to enhance text mining prediction, all showing positive results. Finally, new data was collected for a new application (WordPress), which used the text mining prediction model, results were similar to the results of the other applications.

I. INTRODUCTION

Identifying security vulnerabilities in software is difficult, and building secure software with limited vulnerabilities can be time consuming and expensive. Vulnerability prediction models that identify vulnerability prone software components can be used to focus security efforts, thus reducing the time and effort required to build secure software. In addition, researchers attempting to find vulnerabilities can benefit from the predictions to ease the identification of vulnerabilities in software components.

Vulnerabilites are a specific type of software defect, finding vulnerabilites in software differs significantly from finding defects. The most obvious difference is quantitative: typically there are many more defects than vulnerabilites in software, as one can see by comparing the number of defects reported in a project's defect tracker to the number of vulnerabilities listed on a project's security web page.

However, the qualitative differences between vulnerabilities and defects may be more important than the quantitative differences. Different skills are needed to find vulnerabilities than to find defects. Finding vulnerabilities requires an under standing of both the software and the attacker's mindset [1].

Due in part to these differerences the field of vulnerability prediction is not as mature as the study of software defect prediction [1]. J. Walden, J. Stuckman, and R. Scandariato in [1] released a high quality public dataset of vulnerabilities found in three web applications, and compared two vulnerability prediction models. Their work helped the field of vulnerability prediction take a step towards maturity. This paper is a partial replication and extension of their work and makes three contributions:

- It replicates the comparison of vulnerability prediction models based on text mining using two approaches with models using software metrics as predictors in the context of web application written in PHP.
- It extends the work done in the original paper. The same dataset was used to perform several experiments including using TF-IDF with text mining to enhance prediction, predict new vulnerabilities in future versions of the same application, and in grouping vulnerabilities to enhance text mining prediction. Finally, data was collected for a new PHP web application (WordPress) and text mining prediction techniques were evaluated using it.

This paper is organized into sections as follows. Section II describes the original data set and then discusses the collection of the new WordPress dataset. Section III describes the methodology used in the experiment. Section IV describes the results of the experiments. Section V discusses threats to the validity of this study. Section VI concludes the paper and outlines directions for future work.

II. DATA SETS

One of the main contributions of the original paper was providing a novel, hand-curated dataset of vulnerabilities in three PHP web applications: Drupal, PhpMyAdmin, and Moodle. The dataset contained 223 vulnerabilities from multiple versions of the three open-source web applications written in PHP. The dataset was made publicly available [2] and was used for the work done in this paper. The data contained the following:

- Source code for each web application studied.
- Source code metrics for each PHP file at each application version.
- A mapping from vulnerabilities to files for each version.
- Identifiers for each vulnerability from the vendor or CVE, along with the source code of the commits that introduced and fixed the vulnerability (except for vulnerabilities introduced prior to the first version present in the dataset.)

Following in the steps of the original paper, vulnerabilites for a new PHP web application was collected. The remaining part of this section describes the collection of this dataset.

A. Selecting the application

The same criteria from the original paper was used to select a new application:

- The application must be free, open-source software.
- The application must be a web application written in PHP. The application needed to be in PHP in-order to apply some of the same methods used in the original paper.
- The application needed to contain a large number of vulnerabilities in itself (not including vulnerabilities found in plugins).

The selected application was WordPress, a popular free and open-source content management system (CMS) based on PHP and MySQL.

B. Obtaining source code

The source code for Wordpress was obtained from the application's public Git repository [4]. The repository contained Major (e.g. v4.0), minor (v4.7), and sub- minor (v4.0.1) versions of the application. A total of 217 versions were included ranging from v1.5 – v4.7.3.

Only files containing PHP code were used for vulnerability prediction excluding the version.php file, which only contains version information. In addition, all client-side code was excluded.

C. Mining vulnerabilities

All WordPress vulnerabilities used in this paper were taken from the WPScan Vulnerability Database [5], which is a catalog of over 6600 WordPress Core, Plugin and Theme vulnerabilities. Only WordPress Core vulnerabilities were used in this paper.

The database contains a list of vulnerabilites for each WordPress version, each vulnerability has a page which contains details regarding the vulnerability including its first apperance, when the vulnerability was fixed, its classification, and several references including in most cases a CVE reference and a link to the Github commit page where the filename containing the vulnerability can be found.

Vulnerabilities for versions 3 and 4 including all their subversions were collected. A custom script was made to find the vulnerabilites for a specific version, then extract the the file name containing the vulnerability from the WordPress commit page. Finally, a set of (ID, filename) were written out to a file for each version.

D. The collected vulnerability dataset

The resulting vulnerability dataset contains a total of 144 vulnerabilites, 91 vulnerabilities in v3 and 53 in v4. Table I depicts the classes of vulnerabilities represented in the dataset for each version. Vulnerabilites were categorized by the creaters of the WPScan Vulnerability Database and are described below:

- **AUTHBYPASS**: Authentication and Session Management vulnerabilities. These are privilege bypass vulnerabilities allowing the attacker to subvert application-enforced user permissions or user logins; Information disclosure vulnerabilities allowing for information from the application's database or the server's disk to be read; Vulnerabilities related to missing or inadequately implemented encryption.
- **BYPASS:** Vulnerabilities where an attacker can bypass application controls and restrictions.
- **CSRF**: Cross-site request forgery vulnerabilities allowing for outside, malicious HTML to induce the user to perform unwanted actions.
- **DOS**: Vulnerabilities allowing availability violations.
- **LFI**: Local File Inclusion vulnerabilities allow an attacker to read and sometimes execute files on a web server.
- **RCE**: Remote Command Execution vulnerabilities allow the execution of arbitrary commands on a web server.
- **REDIRECT:** Untrusted input is accepted and causes the web application to redirect the request to a URL contained within untrusted input.
- **SQLI**: SQL injection vulnerabilities allow attackers to execute malicious SQL queries.
- **UPLOAD**: Arbitrary files can be uploaded to the web server.
- **XSS**: Cross-site scripting vulnerabilities allowing for malicious Javascript to be executed in a user's browser.

TABLE I. CLASSES OF VULNERABILITIES IN EACH VERSION

	Version 3	Version 4
AUTHBYPASS	6	1
BYPASS	6	6
CSRF	7	5
DOS	3	3
LFI	1	1
RCE	1	1
REDIRECT	3	2
SQLI	5	2
UPLOAD	0	1
XSS	32	23
FPD	2	0
MULTI	2	0
SSRF	6	3
XXE	2	0
UNKNOWN	15	5

- **FPD**: Full Path Disclosure vulnerabilities allowing for the installation path of the application to be maliciously obtained.
- **MULTI**: Multimedia related vulnerabilities, allowing attackers to bypass restrictions.
- **SSRF**: Server Side Request Forgery vulnerabilities allow an attacker to make request from the vulnerable server to other servers on the network.
- **UNKNOWN**: Miscellaneous vulnerabilities that were not classified.
- **XXE**: XML External Entity vulnerabilities allowing for XML input containing a reference to an external entity to be injected leading to disclosure of confidential data, denial of service, or SSRF attacks.

Both application versions had similar vulnerability distributions differeing lightly in a few categories. Cross-site-scripting vulnerabilities dominated in both versions while in Version 3 there were many unknown vulnerabilites.

The categories of vulnerabilites in this dataset are very granular compared to the more general categories used in the original paper (Code Injection, CSRF, XSS, Path Disclosure, Authorization issues, Other). Nevertheless, these categories are all covered by the more general categories and the distribution of this application differs from the other applications.

E. Reasons for excluding vulnerabilities from the dataset

Not all vulnerabilities contained URLs to the Github repository of the commit containing the vulnerability. This made it difficult to find the filename containing the vulnerability. Therefore, these vulnerabilities were excluded from the dataset.

III. METHODOLOGY

The original paper focused on comparing two prediction techniques, one using software metrics and the other using text mining. The first part of this study was replicating this work. The second part included several experiments to extend the original work using the text mining technique. Both parts were achieved by studying the original dataset, and the scripts used to produce the results. For the dataset, one version was considered for each of the three application: Drupal 6.0, PHPMyAdmin 3.3, and Moodle 2.0.

The source files were labeled as either “vulnerable” or “clean”. The vulnerability status of a file was the dependent variable in this study. Independent variables for source code metrics based models included such metrics as as fan-in and fan-out and lines of code, while independent variables for text mining models were the term frequencies.

The scripts contained in the original dataset performed a cross-validation experiment twice on each application version. In the first part of the study two experiments were conducted. In the first, software metrics as predictors were used, while in the second, term frequencies were used. The quality of the predictions in both cases are compared using several performance indicators, which are described below.

A. Dependent variable

Files were labeled vulnerable if at least one vulnerability was present, while files where no vulnerabilities were present were labeled clean.

Because only one particular version of each application of PHPMyAdmin and Moodle was tested for vulnerability prediction, not all vulnerabilities for those two applications were included in this study. Only 31 out of the 75 vulnerabilities in PHPMyAdmin and 25 out of the 51 vulnerabilities in Moodle were used.

B. Independent variables

1) Software metrics: The authors of the original paper developed a custom metrics computation tool, which was used to extract metrics from PHP code. The following metrics were computed and are included in the dataset [1]:

- *Lines of code*: Number of lines in a source file where PHP tokens occur, excluding lines without PHP tokens, such as blank lines and comments. When tokens span multiple lines, such as in the case of strings with embedded newlines, only one line is counted in the total.
- *Lines of code (non-HTML)*: Same as *Lines of code*, except HTML content embedded in PHP files (content outside of php start/end tags) is not considered.
- *Number of functions*: Number of function and method definitions in a file.

- *Cyclomatic complexity*: The size of a control flow graph after linear chains of nodes are collapsed into one. Computed by adding one to the number of loop and decision statements in the file.
- Maximum nesting complexity: The maximum depth to which loops and control structures in the file are nested.
- Halstead's volume: A volume estimate ((N1 + N2) log n1 + n2) using the number of unique oper- ators (n1) and operands (n2) and the number of total operators (N1) and operands (N2) in the file. For the purposes of this metric, operators are method names and PHP language operators, while operands are parameter and variable names.
- Total external calls: The number of instances where a statement in the file being measured invokes a function or method defined in a different file.
- Fan-in: The number of files (excluding the file being measured) which contain statements that invoke a function or method defined in the file being measured.
- Fan-out: The number of files (excluding the file being measured) containing functions or methods invoked by statements in the file being measured.
- Internal functions or methods called: The number of functions or methods defined in the file being measured which are called at least once by a statement in the same file.
- External functions or methods called: The number of functions or methods defined in other files which are called at least once by a statement in the file being measured. (When the target of a function or method call is uncertain, all possible call targets are considered to have been called.)
- External calls to functions or methods: The number of files (excluding the file being measured) calling a particular function or method defined in the file being measured, summed across all functions and methods in the file being measured.

2) Text mining: One of the scripts provided in the dataset tokenizes an application's source files using a "bag of words" approach. In this approach, PHP's built-in `token_get_all` function is called to tokenize each PHP source file and output a list of tokens and their associated frequencies. The set of unique tokens is the vocabulary of the developers.

The tokenizer processes the set of tokens to eliminate unnecessary features. It ignores comments and whitespace. String and numeric literals are converted into fixed tokens, e.g. T_STRING is used to represent a string literal instead of the contents of the string [1].

In this model, each source file is represented as a term vector. These terms are then used as the predictors in the models.

C. Cross-validation and performance indicators

A stratified cross-validation technique to evaluate model performance was used. Files of an application are randomly divided into three folds of equal size. Only three folds are used due to the low absolute number of vulnerabiliites in some applications.

A prediction model is built using two folds as the training set, and the third as the testing set. Iteratively, each fold is used as the testing set. The model is used to predict the class of the files in the testing set. Each cross-validation is repeated 10 times with a different, randomley chosen partitioning of the dataset into folds.

As in the original paper, *Random Forest* was used as the primary machine learning algorithm for the replication part. However, the implementation used in this paper used a more recent implementation provided by Weka 3.8.1 running on openjdk version "1.8.0_121", with the size of the forest set to 100 trees. All other parameters are set to their default values.

At the end of the experiment, each file has been predicted as either vulnerable or clean, and is compared to the known correct vaules in the dataset. Accordingly, each prediction is judged to be either correct (true positive or true negative) or erroneous (false positive or false negative).

Performance indicators defined in Table II are computed for each cross-validation experiment. In security, the most important indicator is recall (R), which can assume values between 0 and 1 [1]. A higher recall means that the model has correctly identified a larger number of vulnerable files. The indicator is defined as follows:

$$R = \frac{TP}{TP + FN}$$

The second key indicator used is the file *inspection* ratio indicator (I), which is the percentage of files that one has to consider (e.g., inspect manually) to make it possible to find the true positives identified by the model [1]. This indicator can assume values between 0 and 1 and is defined as follows:

$$I = \frac{TP + FP}{TP + TN + FP + FN}$$

For completeness, other indicators like precision, false positive rate, and accuracy are reported. However, these indicators are not sued to draw conclusions in the comparison.

D. Undersampling

One of the challenges that is often present in vulnerability datasets is that percentage of vulnerable files is very low. This means that there is an imbalanced dataset, where there are overwhelmingly more clean files (no vulnerabilities) than vulnerable files in the training set during each iteration of the cross-validation. With undersampling, only a subset of clean files is selected during training, while all vulnerable files are retained.

TABLE II. TERMINOLOGY

Measure	Definition
TP	**True positives**: number of files that are predicted as vulnerable and do contain vulnerabilities.
TN	**True negatives**: number of files that are predicted as clean and that do not contain vulnerabilities.
FP	**False positives**: number of files that are predicted as vulnerable but do not contain vulnerabilities.
FN	**False negatives**: number of files that are predicted as clean but do contain vulnerabilities.

The clean file samples are randomly chosen such that they match the number of vulnerable files. The result is a perfectly balanced training set, which is used to build the prediction model.

Undersampling was used in all of the experiments in this paper using the implementation provided by Weka, the SpreadSubsample unsupervised filter.

E. Hypotheses

Two key performance indicators (R and I) are compared for two prediction techniques, one using software metrics (SM) and the other using text mining (TM). As mentioned above, cross validation is repeated 10 times for each application. The goal of the comparison is to determine whether, on average, the SM technique performs better or worse than the TM technique.

Hence, the null hypotheses are as follows:

$$H_0^R : \mu\{R_{SM}\} = \mu\{R_{TM}\}$$

$$H_0^I : \mu\{I_{SM}\} = \mu\{I_{TM}\}$$

F. Additional experiments

After concluding the replication of comparing the two prediction models. Several experiments were performed:

- Experimenting with additional algorithms and word frequencies
- Experimenting with cross-version predictions
- Experimenting with grouping vulnerabilities
- Using a new dataset (WordPress)

These experiments are described in the results section.

IV. RESULTS

Table III provides summary information about the three applications, including number of files, number of vulnerable files, percentage of vulnerable files (P-rate), and the total number of text features. This information was obtained by running the *tokenize.R* script on each application separately. The numbers are identical to those of the original paper except for the text features of PHPMyAdmin, which had 5233, compared to 5232 in the original paper.

This information is helpful in interpreting the results of the experiment. Moodle is the largest of the three applications, with PHPMyAdmin and Drupal being similar in size.

TABLE III. DESCRIPTIVE STATISTICS FOR THE APPLICATIONS. MOODLE IS THE LARGEST APPLICATION AND HAS THE LOWEST POSITIVE RATE. DRUPAL IS AT THE OPPOSITE END OF THE SPECTRUM.

	Vulnerable files	Total files	P-rate(%)	Text features
Drupal	62	202	**30.68**	3886
PHPMyAdmin	27	322	**8.39**	5233
Moodle	24	2942	**0.82**	18306

Furthermore, the three applications are heterogeneous, providing a rich set of testing conditions for the prediction models. Moodle has the smallest P-rate. As the P-rate gets smaller, the task of predicting which files are likely to contain vulnerabilities is more challenging, as vulnerable files are rare. Better prediction performance in case of a smaller positive rate is more valuable due to the larger number of files that a human reviewer would need to examine [1].

A. Replication of original paper:

Two approaches were taken to replicate the prediction results. First, a script from the dataset was used to perform both prediction techniques and return the results for all applications. This approach used a maximum limit of 100 tokens for the text mining technique. Second, each technique was performed separately using standalone scripts for each application. This approach used a maximum of 1000000 most frequent words while performing the text mining predictions. The results from both approaches are obtained from the cross-validation experiment described in Section III-C.

Table IV and Table V show the replication results with Table IV showing the results of the first approach using the single script and Table V showing the result of the second approach using standalone scripts. The tables directly compare the software metrics and text mining techniques over the two key performance indicators recall and inspection. Mean values (μ) were calculated by averaging the performance of the prediction models over ten executions of the cross-validation experiments. Standard deviation (σ) is also reported.

For the results in Table IV, the prediction technique based on software metrics performed slightly better than the text mining technique on average. The recall was higher in the case of Drupal (+1.6 percentage points), PHPMyAdmin (+1.8), and Moodle (+2.1). These results do not show a significant difference between the two approaches. The file inspection ratio was close, with text mining performing slightly better in the case of Drupal (-4.7 percentage points), PHPMyAdmin (-3.3) and Moodle (-4.3). These results are the oposite of the results found in the original paper where on average, the text mining techniques performed better. The reason for this could be the difference in the versions of the tools used to perform the predictions such as Weka.

The results from the second approach shown in Table V are closer to the results from the original paper with the text mining prediction technique performing better on average. The recall was higher in the two cases of Drupal (+5.7 percentage points) and Moodle (+2.1) and metrics performing slightly

TABLE IV. RESULTS OF CROSS-VALIDATION. MEAN AND STANDARD DEVIATION FOR THE KEY PERFORMANCE INDICATORS.

	Indicator		*SW metrics(%)*	*Text mining(%)*
Drupal	Recall	**μ**	78.2	76.6
		σ	5.8	6.5
	Inspection	**μ**	46	41.3
		σ	2.7	3.9
PHPMyAdmin	Recall	**μ**	72.2	70.4
		σ	11.7	9.7
	Inspection	**μ**	42.7	39.4
		σ	5.1	4.8
Moodle	Recall	**μ**	77.1	75
		σ	7.9	7.7
	Inspection	**μ**	32.1	27.8
		σ	3.6	3.4

TABLE V. RESULTS OF CROSS-VALIDATION MEAN FOR THE KEY PERFORMANCE INDICATORS USING STANDALONE SCRIPTS AND 1000000 WORD COUNT FOR THE TEXT MINING TECHNIQUE.

	Indicator		*SW metrics(%)*	*Text mining(%)*
Drupal	Recall	**μ**	78.2	83.9
	Inspection	**μ**	46	44.6
PHPMyAdmin	Recall	**μ**	72.2	68.5
	Inspection	**μ**	42.7	35.7
Moodle	Recall	**μ**	77.1	79.2
	Inspection	**μ**	32.1	26.5

better in the case of PHPMyAdmin (-3.7). The file inspection ratio also performed better with the text mining approach, Drupal (-1.4), PHPMyAdmin (-7), and Moodle (-5.6). In conclusion, text mining performed better given a higher number of words.

Table VI shows additional performance indicators from the first approach of the replication. Surprisingly, the precision for text mining was better for all applications; these results are similar to the results from the original paper. Table VII shows the same additional performance indicators for the second replication approach. Text mining again had better precision for all indicators and the results were similar to those of the first approach.

B. Additional experiments:

1) Experimenting with additional algorithms and word frequencies:

With the text mining technique performing better during the replication study, the first extension of the original paper was to attempt to improve its performance. The experiment included using the frequency of the words (tokens) instead of word occurrence count, and experimenting with additional

TABLE VI. ADDITIONAL PERFORMANCE INDICATORS FOR CROSS-VALIDATION FOR FIRST REPLICATION APPROACH.

	Indicator	*SW metrics (%)*	*Text mining (%)*
Drupal	Precision	52.9	56.6
	FP rate	31.4	26.8
	Accuracy	72	75
PHPMyAdmin	Precision	12.3	14.7
	FP rate	40.2	36.8
	Accuracy	60.7	64.3
Moodle	Precision	1.9	2.3
	FP rate	31.7	27.4
	Accuracy	68.4	72.7

TABLE VII. ADDITIONAL PERFORMANCE INDICATORS FOR CROSS-VALIDATION FOR SECOND REPLICATION APPROACH.

	Indicator	*SW metrics (%)*	*Text mining (%)*
Drupal	Precision	52.9	55.8
	FP rate	31.4	28.2
	Accuracy	72	74.5
PHPMyAdmin	Precision	12.3	15.7
	FP rate	40.2	32.9
	Accuracy	60.7	66.8
Moodle	Precision	1.9	2.5
	FP rate	31.7	26
	Accuracy	68.4	74

classification algorithms. The same three applications used in the replication part were used in this experiment in-order to compare the results.

To calculate the frequencies, TF-IDF (Term Frequency-Inverse Document Frequency) was used which is supported by Weka's StringToWordVector filter. TF-IDF first divides the number of occurrences of each word in a document by the total number of words in the document. This is done to avoid the issue of having a higher average count value in longer documents. The new features are called TF for Term Frequencies. A refinement on top of TF is to downscale weights for words that occur in many documents in the corpus. This is done by dividing the number of documents by the number of documents containing a word then multiplying the result by TF. This is known as TF-IDF for "TF * (num of Documents/num of documents containing word i)."

Several machine learning algorithms such as SVM, Logistic Regression, KNN, J-48, Naïve Bayes, and Random Forest were tested using the implementation provided by Weka. Only Naïve Bayes and Random Forest performed well. The results are shown in Table . For comparison reasons, the table also includes the results of performing the text mining technique of the replication part (no TF-IDF) using Naïve Bayes.

TABLE VIII. TEXT MINING (TM) RESULTS USING RANDOM FOREST AND NAÏVE BAYES WITH AND WITHOUT TF-IDF

		Indicator		TM no TF-IDF (%)	TM with TF-IDF (%)
Drupal	Random Forest	Recall	**μ**	83.9	81.5
		Inspection	**μ**	44.6	43.3
	Naïve Bayes	Recall	**μ**	75	84.7
		Inspection	**μ**	45	51.2
PHPMy Admin	Random Forest	Recall	**μ**	68.5	66.7
		Inspection	**μ**	35.7	36.5
	Naïve Bayes	Recall	**μ**	85.2	87
		Inspection	**μ**	66.5	64.1
Moodle	Random Forest	Recall	**μ**	79.2	81.2
		Inspection	**μ**	26.5	26.8
	Naïve Bayes	Recall	**μ**	75	79.2
		Inspection	**μ**	33.4	38.5

On average, Naïve Bayes with TF-IDF performed best, followed by Naïve Bayes without TF-IDF, then Random Forest without TF-IDF and finally Random Forest with TF-IDF had the lowest average.

Looking at the algorithms, Random Forest with TF-IDF outperformed Random Forest without TF-IDF and Naïve Bayes with TF-IDF outperformed Naïve Bayes without TF-IDF.

In conclusion, the results from this experiment show that this approach did not significantly outperform the approach from the replication part. By looking closely at the results for each application, it appears that not a single algorithm performed consistently between applications. This may indicate that different applications require different algorithms for predicting their vulnerabilities.

2) Experimenting with cross-version predictions

This experiment attempted to predict new vulnerabilities on future versions of the same application after training on one version. PHPMyAdmin v3.3.0 was used for the training data and PHPMyAdmin v3.4.0 as the test data. The vulnerabilities in the test data were narrowed down to the vulnerabilities that existed only in v3.4.0, which were a total of 6. Text mining with Random Forest was used for this experiment. The results showed an 83.3% recall rate and 44.7% inspection rate. These results are promising but further testing is needed with more versions and within other application to confirm these results.

3) Experimenting with grouping vulnerabilities

In this experiment instead of using a single version (e.g. PHPMyAdmin 3.3.0) of an application for training and testing, the vulnerabilities of an entire version of PHPMyAdmin including all the Major (e.g. v3.0), minor (v3.1), and sub-minor (v3.1.0) versions were gathered. The approach used to gather the data was to tokenize each version of the application separately then combine all the tokenized files after removing duplicates. This resulted in a total of 52 vulnerabilities for PHPMyAdmin v3. The goal of this experiment is to evaluate the performance of predicting vulnerabilities with an increased number of vulnerabilities in the data. Text mining with Random Forest was also the technique used for this experiment.

The cross-validation approach described in section III-C was performed to evaluate the model on the gathered data. The results show a significant improvement from the results found for PHPMyAdmin v3.3.0 in the original paper and the results obtained during the replication part of this paper. For v3 recall was 88.3% compared to 73.7% in the original paper and 68.5% in the replication part of this paper.

This result shows that increasing the number of vulnerabilities can help improve the prediction performance. As with the previous experiment, more testing needs to be done to confirm this result, including testing on other versions and using one gathered version as training data and another as test data.

4) Using a new dataset (WordPress)

To further extend the work of the original paper, the cross-validation approach of section III-C was performed with the text mining and Random Forest approach on WordPress vulnerabilities described in section II-A-E.

The majority of vulnerabilities in version 3 were shared among a few versions at 18 vulnerabilities; v3.7.1 was selected for use in this experiment. For version 4, v4.2 was used and it contained 21 vulnerabilities. From the selected versions individual versions contained fewer vulnerabilities than the applications used previously in the replication part. Table IX provides summary information about the application, including number of files, number of vulnerable files, percentage of vulnerable files (P-rate), and the total number of text features.

Table X shows the results for this experiment, version 3.7.1 had a recall rate of 67.9% and an inspection rate of 27.2%. Version 4.2 had a 76.5% recall reate with an inspection rate of 26.5%. Given the low P-rate of the versions used in this experiment, the results are close to those found previously during the replication work for other applications.

In conclusion, the text mining with Random Forest prediction technique worked successfully on a different application performing similarly to how it performed with the original three applications. Other approaches such as grouping vulnerabilites can be applied here as well to imporove prediction performance.

TABLE IX. DESCRIPTIVE STATISTICS FOR THE APPLICATIONS

	Vulnerable files	Total files	P-rate(%)	Text features
WordPress 3.7.1	18	493	**3.6**	9689
WordPress 4.2	21	547	**3.8**	10317

TABLE X. TEXT Mining (TM) results using Random Forest on WordPress data

		Indicator		TM (%)
WordPress 3.7.1	Random Forest	Recall	**µ**	67.9
		Inspection	**µ**	27.2
WordPress 4.2	Random Forest	Recall	**µ**	76.5
		Inspection	**µ**	26.5

TABLE XI. ADDITIONAL PERFORMANCE INDICATORS FOR CROSS-VALIDATION ON WORDPRESS DATA

	Indicator	*Text mining (%)*
WordPress 3.7.1	Precision	6.4
	FP rate	26
	Accuracy	73.8
WordPress 4.2	Precision	8.4
	FP rate	25.2
	Accuracy	74.4

V. THREATS TO VALIDITY

Threats to construct, conclusion, internal, and external validity are discussed below for both parts of the paper.

1) *Replication part*

Construct validity. The authors of the dataset excluded some vulnerabilites from it. This was due to the inablity to determine the location or nature of the vulnerability or because different files were affected during the lifetime of the vulnerability.

Conclusion validity. Threats to conclusion validity relate to issues that affect the validity of statistical inferences. This paper used the same standard techniques as the original paper did for statistics and modeling, well recognized tools for these purposes were used, including Weka and R.

Internal validity. The authors of the dataset did not chose the applications randomly, they chose applications that contained high numbers of vulnerability advisories for their core components.

External validity. The results might be specific to the set of web applications that were used in this paper. While the applications are from different domains, all three applications were open source. This paper extended this a step further by adding a fourth open source application. However, future work is still needed to include a broader set of web applications including commercial applications. This would be needed to generalize the results to the entire class of PHP web application.

2) *Extension part*

Construct validity. During the gathering of WordPress vulnerability data, some vulnerabilities were excluded to do the lack of a reference showing the file containing the vulnerability. In addition, all data was taken as is from the WPScan Vulnerability Database. No validation was done on the vulnerabilities or on the files they were located in. During the analysis it was found that some vulnerable files were not located in the versions they were listed in. Future work should include manually verifying the vulnerabilities and their mappings.

Conclusion validity. Threats to conclusion validity relate to issues that affect the validity of statistical inferences. This paper used standard techniquesfor statistics and modeling, well recognized tools for these purposes were used, including Weka and R.

Due to the time limit associated with wriitng this paper, limited versions and techniques were used during the extension work. In order to confirm the finding of this paper, future work is needed to be done, this should include using additional versions of the applications and testing the same approches using the metrics data.

Internal validity. The WordPress dataset was selected because it was open source and it contained a high number of vulnerability advisories for its core components.

External validity. The work done in this part extended the original work by applying the prediction techniques on a fourth open source web application. Future work is still needed to include a broader set of web applications including commercial applications. This would be needed to generalize the results to the entire class of PHP web application.

VI. CONCLUSION

This paper is a partial replication and extension of the work done by J. Walden, J. Stuckman, and R. Scandariato in [1]. They provided a dataset which contained open source web applications, including software metrics, source code, and vulnerability locations. This dataset contained 223 vulnerabilities found in three web applications written in PHP: Drupal, Moodle, and PHPMyAdmin. The dataset also included metric data extracted by a custom tool devloped by the original authors.

This dataset was first used to replicate the comparision of vulnerability prediction effectiveness of two modeling

techniques: one based on software metrics and the other based on text mining using a "bag of words" model for the source code. During the replication part of the paper, both models were built using a Random Forest machine learning technique. Undersampling was used to compensate for the small percentage of vulnerable files in most applications.

Secondly, in the extension part of the paper, the dataset was used in three additional experiments:

- The effectiveness of building models using other machine learning algorithms was tested. In-addition, enhancements to the text mining technique were attempted using TF-IDF.
- Cross-version predictions were attempted. A single version was used as training data and predictions were made on the next minor version.
- An attempt to increase the number of vulnerabilities was done by grouping all the vulnerabilities of an application's version. Cross-validation with text mining were used for prediction.
- Finally, a new dataset of an open source web application (WordPress) was collected. The text mining modeling technique with Random Forest was used on this dataset for vulnerability prediction.

The results of this study are summarized below:

- **Predictive value of text features**: Models were evaluated using stratified cross-validation, finding that text mining provided better recall performance when not limiting the number of tokens used, with the cost almost being the same, as measured by the file inspection ratio. Recall and inspection ratio values can be found in Table IV and Table V.
- **Additional algorithms and word frequencies:** TF-IDF with Naïve bayes using a text mining technique on average performed slightly better than Random Forest using the same technique. However, the results varied per application and therefore it is not believed that this approach will generalize for all web applications.
- **Predicting new vulnerabilites on future versions**: A single version of PHPMyAdmin was used as training data to build a model using text mining with Random Forest. Predictions were made on the following minor version with a recall of 83.3% and an inspection rate of 44.7%.
- **Grouping vulnerabilities**: All of the vulnerabilites were grouped for a version of PHPMyAdmin v3, this increased the number of vulnerabiliites uin the data. Again text mining with Random Forest was used and results showed a significant improvement in recall with an 88.3% rate compared to 68.5% for the highest number of vulnerabilities in a single version.
- **Vulnerability prediction in WordPress**: Models were evaluated using stratified cross-validation. Text mining with Random Forest provided recall rates comparable to the other three applications. Recall and inspection ratio values can be found in Table X.

Future work should include expanding the study done on predicting new vulnerabilities in future versions and grouping vulnerabilities to a broader set of versions and applications. In-order to generalize the results of this paper to the entire class of PHP web applications, further open source and commercial application need to be evaluated. Another area to examine is the lifetime of vulnerabilities and whether this can be predicted for certain classes of vulnerabilities. Finally, the examination of additional predictors beyond source code metrics and text mining features is needed.

REFERENCES

[1] J. Walden, J. Stuckman, R. Scandariato, Predicting Vulnerable Components: Software Metrics vs Text Mining, 2014.

[2] PHP Security vulnerability dataset, http://seam.cs.umd.edu/webvuldata, accessed April 2017.

[3] Wordpress Wikipedia page, https://en.wikipedia.org/wiki/WordPress, accessed April 2017.

[4] WPScan Vulnerability Database, https://wpvulndb.com, accessed April 2017.

[5] Open Software Security Community (OWASP), https://www.owasp.org, accessed April 2017.