Vulnerability Detection in PHP Web Application Using Lexical Analysis Approach with Machine Learning

1st Dhika Rizki Anbiya [1,2]
1) *Laboratory for Information and Communication Technology Services Agency for the Assesment and Application of Technology*
Tangerang Selatan, Indonesia
dhika.rizki@bppt.go.id
2) *School of Electrical Engineering and Informatics*
Bandung Institute of Technology
Bandung, Indonesia

2nd Ayu Purwarianti
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
ayu@stei.itb.ac.id

3rd Yudistira Asnar
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
yudis@stei.itb.ac.id

***Abstract*— Security is an important aspect and continues becoming a challenging topic especially in a web application. Today, 78,9% of websites uses PHP as programming languages. As a popular language, WebApps written in PHP tend to have many vulnerabilities and they are reflected from their source codes. Static analysis is a method that can be used to perform vulnerability detection in source codes. However, it usually requires an additional method that involves an expert knowledge. In this paper, we propose a vulnerability detection technique using lexical analysis with machine learning as a classification method. In this work, we focused on using PHP native token and Abstract Syntax Tree (AST) as features then manipulate them to get the best feature. We pruned the AST to dump some unusable nodes or subtrees and then extracted the node type token with Breadth First Search (BFS) algorithm. Moreover, unusable PHP token are filtered and also combined each other token to enrich the features extracted using TF-IDF. These features are used for classification in machine learning to find the best features between AST token and PHP token. The classification methods that we used were Gaussian Naïve Bayes (GNB), Support Vector Machine (SVM) and Decision Tree. As the result, we were able to get highest recall score at 92% with PHP token as features and Gaussian Naïve Bayes as machine learning classification method.**

Keywords— **Vulnerability detection, classification, machine learning, imbalanced data set.**

I. INTRODUCTION

PHP is one of the most popular language according to a survey in GitHub below javascript, python, java, and ruby [1]. The survey reveals that PHP language is used in many web application [2]. As being used by many WebApps, it tends to contribute more vulnerabilities than other languages. In OWASP top 10 application security risks 2017, injection such as SQL injection (SQLI) still being a number one issue, and cross site scripting (XSS) is also on the list [3]. This vulnerability may cause a huge loss in term of data or cost, as being found on Heartland Payment System [4]. The SQLi vulnerability was responsible for 100 million breaches of card data in more than 650 financial services and reported $300m in losses. This vulnerability is typically caused by malformed inputs that reach some sensitive assets such as databases. This kind of vulnerability can be avoided by passing inputs through sanitization functions for validating and removing malicious/dangerous metacharacters.

Static analysis is the common method that is used for vulnerability detection of an application [5]. This method performs analysis without running the application. There are many static analysis tools for detecting vulnerability automatically in source code. However, it takes significant amount of knowledge to develop these tools due to its complexity for detecting the vulnerability [6, 7]. Also this knowledge may be wrong or misleading so that it leads to false positive or unable to detect vulnerabilities.

This paper will presents an approach using static analysis with lexical analysis technique for reducing an expert knowledge. Lexical analysis is a static analysis technique which is transforming source code to other representation usually token for further use [8]. In this work, both AST and PHP Token will be used as a feature with some pre-processing technique. In pre-processing, pruning method will be applied in AST to improve the learning process. Meanwhile in PHP token we will enrich the PHP token with all variables that appear in source code. Various machine learning methods will be used for classifying the vulnerable or not vulnerable class. The Gaussian Naïve Bayes, Support Vector Machine and Decision Tree will be use as the classification algorithm [9]. Later on we analyse both feature set (i.e, AST vs PHP Token) to get the best feature set for vulnerability detection. In this work, we gather the data manually based on vulnerability database from cvedetails website. However, the data distribution (vulnerable vs not-vulnerable) is imbalanced,

978-1-5386-7259-4/18/$31.00 ©2018 IEEE

therefore it required a special techniques to handle this (e.g., oversampling SMOTE and undersampling Cluster Centroid). Finaly, our model will be tested on cross-project to find out its performance for detecting vulnerability.

The paper is organized as follow, Section II describes about related works in vulnerability detection with static analysis. Section III describes about technical background as well as concept feature extraction and machine learning algorithm. Section IV presents our method to detect vulnerability. Finally, Section V explains about experimental results and feature analysis. At the end, we conclude this work.

II. Related Works

There are many researches in vulnerability detection with static analysis using lexical approach technique and token as a feature. Token can be represented as another symbol, parser token or Abstract Syntax Tree (AST). Application Protocol Interface (API) symbol can be use as a token. This API consists of name type, function name, and cast type in C programming language extracted as a feature then transformed to vector with tf.idf algorithm. The vector then calculated with Principal Component Analysis (PCA) to compare the API pattern. This method is able to eliminate form 6,788 to 20 functions that has the same pattern and able to detect 2 flaws, the one that previously known and one zero-day vulnerability in FFMpeg library [10].

Token can also be obtained by extracting AST from source code then embbedding it in vector space. The vector later being computed with Latent Semantic Analysis (LSA) to identify vulnerability and then suggest this vulnerability discovery to analyst. Evaluation being applied to for popular open-source projects LibTIFF, FFmpeg, Pidgin and Asterisk and pick the four known vulnerability as a starting point. This model is able to find real zero-vulnerability [11].

Another representation token is an Intermediate Slice Language (ISL) that will be use in HMM training [12]. There are two phases in this method, the first phase is building the corpus and the second phase is vulnerability detection. The approach later being implemented in DEKANT (the hidDEn marKov model diAgNosing vulnerabiliTies) tools. It first extracts slices from source code, next translates these slices into ISL, retrieves their variable map, and analyses the representation to define whether the code is vulnerable or not. The tool is able to discovered 16 zero-day vulnerabilites in 10 Wordpress plugin and also found 21 vulnerability in 10 open source project written in PHP.

III. Technical Background

A. Vulnerability PHP Web Application

Most of web application today built using scripting language like PHP programming languange. PHP is used by 78.9% of all website with server-side programming language [13]. As a popular programming language, PHP tends to have a high vulnerability level. The vulnerability type that will be gathered as data in this research is SQL injection (SQLi), cross site scripting and directory traversal.

1) SQL injection

This vulnerability happen if attacker send a malicious code through the application. SQL injection is a form of defect in a code that can interrupting a machine, exposing sensitive data or spreading a malicious software [14]. Attacker perform an exploit to user input by injecting some sql command.

```
# Post Variable
username = request.POST['username']
password = request.POST['password']

#vulnerable sql query
sql = "SELECT id FROM users WHERE username="+ username + "AND
password=" +password

#execute sql statement
database.execute(sql)
```

Figure 1 SQLi pseudo code

2) Cross site scripting

Attacker perform cross site scripting (xss) to unvalidated input and display it directly on a web page [14]. The purpose of this attack is to obtain sensitive data, frauding attempt even damaging the website. Xss is usually used for a phishing activity, the attacker may send a bogus url of a website via email or other media that will attract user to visit.

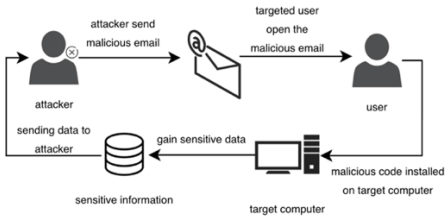


Figure 2 Phishing attack

3) Directory traversal

This vulnerability also known as path traversal, a vulnerability in HTTP protocol that leads attacker for accessing restricted directory and executing a harmful command [15]. For example website with URL:

http://somesite.com/get_page?home=main.html
http://somesite.com/get-files?file=/etc/passwd

Attacker may inject a malicious code through the URL to gain an access to restricted directory and get the sensitive information. In UNIX based operating system, attacker may steal file that contain passwords in /etc/passwd. Attacker also can inject some malicious code that come from outside the website into URL.

B. Feature extraction

We use TF-IDF as a feature extraction in our work. TF-IDF is a product of two factor, tf is term frequency, the frequency of the word in document. IDF is the inverse document frequency for assigning higher weigh to words that only occur frequently across the entire collection. IDF is defined using the fraction N/d fi, where N is the total number of documents in the collection, and d fi is the number of documents in which term i occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents. Because of the large number of documents in many collections, this measure is usually squashed with a log function. The resulting definition for inverse document frequency (IDF) is

$$idf = log\left(\frac{N}{df_i}\right) \quad (1)$$

Combining term frequency with IDF results in a scheme known as tf-idf weighting of the value for word i in document j, ew_{ij}:

$$w_{ij} = tf_{ij} \, . \, idf_{ij} \quad (2)$$

C. Features

According to the previous research [10, 11], the features that we used in this work is token from PHP programming language parser (PHP token) and Abstract Syntax Tree (AST). PHP token generation using a native function that PHP programming language provide, token_get_all.

Furthermore, we use a library for generating AST [16]. Then we perform Breadth First Search technique to obtain the node type from it. The PHP token contains 134 features [17] and the AST contains 158 features (see Table I). In PHP token, we modify the feature with adding all variable name to each of generated token T_Variable.

Table I
AST Node Type Name

#	Node type name
1	Arg
2	Const
3	Expr
4	Expr_Array
5	Expr_ArrayDimFetch
6	Expr_ArrayItem
7	Expr_Assign
8	Expr_AssignOp
9	Expr_AssignOp_BitwiseAnd
10	Expr_AssignOp_BitwiseOr
…	…..……

D. Imbalanced Data Set

The real world data tend to have asymetrical class distribution. The distribution of data between classes is not always equal, one class may have more data than others. Learning process using imbalance data will cause problem in training phase on machine learning, especially in classification. The system will perform better classifying in major class and worse for the class with lower data frequency. Accordingly, to handle this problem, the sampling technique is required. There are two kind of sampling techniques, oversampling and undersampling. The common technique for oversampling the data is SMOTE.

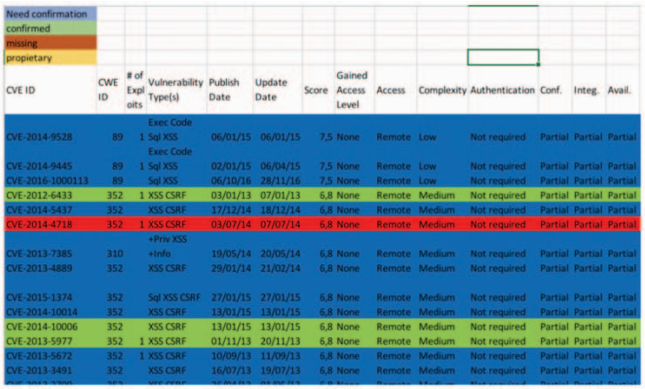
SMOTE (Synthetic Minority Oversampling TEchnique) is an oversampling technique that handle imbalanced data set with adding new syntethic data to the dataset. New minority samples are generated along the lines that connecting minority samples to its nearest minority neighbours. Cluster centroids is one of the undersampling technique. This technique under-samples the majority by replacing cluster of samples by the cluster centroid of a K-Means algorithm.

IV. Proposed Method

This section we describe our method for vulnerability detection. It consists data gathering, pre-processing data and vulnerability detection flows.

A. Data Gathering

Data was manually obtained with CVE (common vulnerability exposure) that provided by cvedetails site. Cvedetails provides an easy to use web interface to CVE vulnerability data [8]. The data are taken from National Vulnerability Database (NVD) feeds provided by National Institute of Standards and Technology. The data contains cve number along with vulnerability description and other information (see Figure 3). Further exploration needed to find the file that affected by vulnerability by visiting each url in the description. Total data that we have already obtained are 461 vulnerable and 136,090 not vulnerable files.This process was take so much time and effort because the cve dump did not include the software copyright. There were many project turned out proprietary or can not be found anywhere (project shutdown).


Need confirmation
confirmed
missing
propietary

CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
CVE-2014-9528	89	1	Exec Code Sql XSS	06/01/15	06/01/15	7,5	None	Remote	Low	Not required	Partial	Partial	Partial
CVE-2014-9445	89	1	Exec Code Sql XSS	02/01/15	06/04/15	7,5	None	Remote	Low	Not required	Partial	Partial	Partial
CVE-2016-1000113	89		Sql XSS	06/10/16	28/11/16	7,5	None	Remote	Low	Not required	Partial	Partial	Partial
CVE-2012-6433	352	1	XSS CSRF	03/01/13	07/01/13	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2014-5437	352		XSS CSRF	17/12/14	18/12/14	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2014-4718	352	1	XSS CSRF +Priv XSS	03/07/14	07/07/14	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2013-7385	310		+Info	19/05/14	20/05/14	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2013-4889	352		XSS CSRF	29/01/14	21/02/14	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2015-1374	352		Sql XSS CSRF	27/01/15	27/01/15	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2014-10014	352		XSS CSRF	13/01/15	13/01/15	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2014-10006	352		XSS CSRF	13/01/15	13/01/15	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2013-5977	352	1	XSS CSRF	01/11/13	20/11/13	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2013-5672	352	1	XSS CSRF	10/09/13	11/09/13	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial
CVE-2013-3491	352		XSS CSRF	16/07/13	19/07/13	6,8	None	Remote	Medium	Not required	Partial	Partial	Partial

Figure 3 Vulnerability dump list

B. Pre-processing

As we described earlier, we performed pre-processing for both token to clear the noise or enrich the features. We used token representation that extracted from the tokenizer in PHP parser and in AST node type.

```
<?php
$offset = $_GET['var'];
$query  = "SELECT id, name FROM products ORDER BY
name LIMIT 20 OFFSET $offset;";
$result = pg_query($conn, $query);
echo "this is print data";
```

Figure 4 Source code example

The extracted token from php parser will be added with variable name in source code (see Figure 4). The first step is transform all the source code in each file to token with php parser. The generated token is saved in a text file then mapped in the csv file after removing unused token. The unused token are T_WHITESPACE, T_OPEN_TAG and T_CLOSE_TAG. The csv will be used for feature extraction with tfidf.

```
Line 1: T_OPEN_TAG ('<?php')
Line 2: T_VARIABLE ('$offset')
Line 2: T_WHITESPACE (' ')
Line 2: T_WHITESPACE (' ')
Line 2: T_VARIABLE ('$_GET')
……
Line 3: T_ENCAPSED_AND_WHITESPACE ('SELECT id, name
FROM products ORDER BY name LIMIT 20 OFFSET ') Line 3:
T_VARIABLE ('$offset')
Line 3: T_ENCAPSED_AND_WHITESPACE (';')
Line 3: T_WHITESPACE ('')
Line 4: T_VARIABLE ('$result')
Line 4: T_WHITESPACE (' ')
Line 4: T_WHITESPACE (' ')
Line 4: T_STRING ('pg_query')
…
```

Figure 5 Tokenizer source code

We also extracted token from source code using AST tree that was generated by library [16] with BFS (Breadth First-search) algorithm (see Figure 6). Similarly with php parser token, there was node that have an unused information. Therefore we applied pruning with a regex matching against certain pattern that can cause vulnerability. We used global variable as a pattern for regex matching.

Figure 6 Generated AST

There were several global variables that can cause vulnerability, they are _POST, _GET, _Cookie, _Request, _Server and _Files [18]. Based on observation, we found the pattern of thus global variable were:

1.'((?=.*Stmt_Echo)(?=.*Expr_Variable))|Expr_ArrayDim Fetch|_GET|_POST'

2.'((?=.*Stmt_Echo)(?=.*Expr_Variable))|Expr_ArrayDim Fetch|file_get_contents'

After the pruned process, we extracted the node types from AST with first search technique. The Pre-processing also happen in token PHP by added more token to the list. The addition taken place if there is T_Variable token by concatenated the variable name with it, for example if there was variable $time in source code then the new feature will be T_Variable$time.

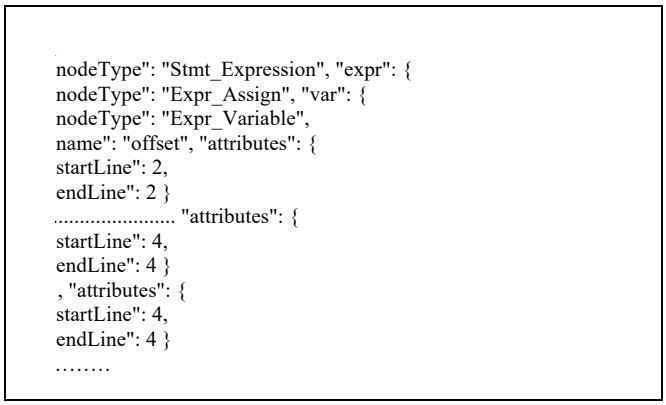


Figure 7 AST after pruning

C. *Feature Extraction*

The generated token then mapped to vector space with tf idf and saved into a csv file. The csv file contains filename, label and features. Each of feature stored into different file for classification with machine learning. The vulnerable source code labeled as 1 and 0 for the not vulnerable.

Table II AST token after feature extraction

_m_4.php.txt	1		0.023		0.162

The total column after label will increased dynamically according to the feature. The AST token has 158 features, PHP token has 134 features while the modify token can not be calculated because it will continued to increase depending on the amount of training data used.

Table III PHP token after feature extraction

	Label		1		134

D. *Vulnerability Detection Flows*

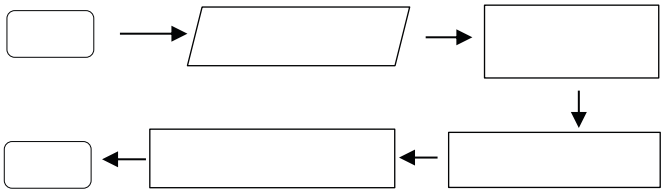


Figure 8 Training data modelling

phases. Firstly, modelling the training data and secondly was the detection phase. The training data that has been collected was preprocessed and then processed using tfidf for feature extraction. Machine learning is performed on the results of

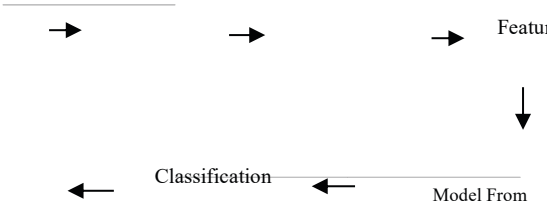


Figure 9 Vulnerability detection

feature extraction using the classification algorithm decision tree, SVM, GNB. In the vulnerability detection process, the test data was pre-processed and feature extraction was performed and then processed by the classification model that has been made. The output of the model is the result of classification.

V. EXPERIMENTAL RESULTS AND ANALYSIS

We did 5 cross validation for every experiment to ensure the result. So far we have PHP token, AST token, AST Token with pruning, and modify PHP token as a feature. We used

GNB, Decision Tree, and SVM as a classifier. Every algorithm is tested with or without sampling technique on each feature as seen on Table IV.

Table IV
EXPERIMENT SCENARIO

Feature	Experiment
PHP Token	Classification with SVM, Decision Tree, GNB
	Classification with (SVM, Decision Tree, GNB) + SMOTE
	Classification with (SVM, Decision Tree, GNB) + Cluster Centroid
AST Token	Classification with SVM, Decision Tree, GNB
	Classification with (SVM, Decision Tree, GNB) + SMOTE
	Classification with (SVM, Decision Tree, GNB) + Cluster Centroid
AST Token + Pruning	Classification with SVM, Decision Tree, GNB
	Classification with (SVM, Decision Tree, GNB) + SMOTE
	Classification with (SVM, Decision Tree, GNB) + Cluster Centroid
Modify PHP Token	Classification with SVM, Decision Tree, GNB
	Classification with (SVM, Decision Tree, GNB) + SMOTE
	Classification with (SVM, Decision Tree, GNB) + Cluster Centroid

Based on our experiment, the best algorithm for each feature can be seen in Figure 11. The difference in recall values between the four features tested in two classeuls was not significant but the token modification feature can be concluded to be the best feature in detecting vulnerabilities.

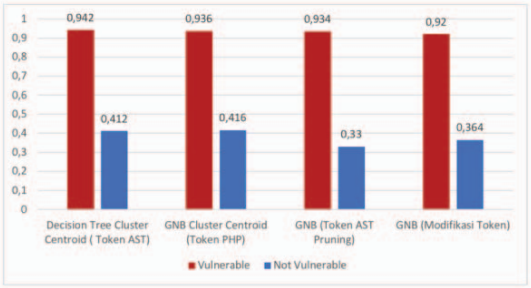


Figure 10 Result of all feature

This was happened because the modification token features do not required pruning and no techniques are needed to handle imbalanced data sets. We also analysed the detection result between AST pruned token and modify PHP token to find out the difference in detection results. The result can later be used to find out the characteristics of data (see Table VI). The vulnerable file is used for the testing data for both feature. The modify PHP token is able to detect vulnerability better than AST pruned token with 6 file correctly.

Table V Test result with pruned AST and modify PHP

No	Filename	Pruned AST	Modify PHP	True Label
1	237__deviceadd.txt	0	0	1
2	106__central.class.txt	0	0	1
3	30__Location.txt	0	1	1
4	115__sendcard.txt	0	0	1
5	25__scan.txt	0	0	1
6	23__security.inc.txt	0	1	1
7	9__wikka.txt	0	1	1
8	24__admin.txt	0	1	1
9	14__doku.txt	0	0	1
10	88__maincore.txt	0	1	1

The AST pruning feature failed to detect vulnerability, because of the pruning technique Some file was pruned all the nodes in a source code others failed because the source code was too complex. For example the AST feature found detection errors pruning on file 24__admin.txt even though the pruning process was correct and the file contains vulnerability which can be read clearly by the code reviewer (see Figure 12) the AST pruning feature, the modify PHP token feature, detection failure also occured because the file source code is very complex and requires other files to carry out further vulnerability.

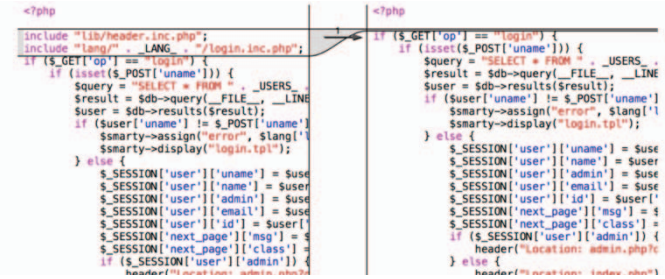


Figure 11 Pruned AST for 24__admin.txt

VI. Conclusions

The result of experiment showed Gaussian Naïve Bayes is the best algorithm for vulnerability detection regardless the features. Moreover, the sampling method has not to do with the recall value in GNB algorithm. The feature itself either PHP token or AST can be used for vulnerability detection. Several, error in detection using the AST feature occurs because the pruning process is still not effective and the studied source code is too complex and require analysis with multiple files. Meanwhile, the error in using PHP token occurs because of the project uses a PHP framework. The use of the framework requires analyzing not only one vulnerable file but other related files.We choose the higher recall amount in because in security domain, the recall value in vulnerable class more likely chosen. Overall, our method shown that modify PHP token able to detect vulnerability and the result will become a guidance to code reviewer for further analysis. Moreover, it needs further studies to get a better result.

VII. Acknowledgement

This paper is supported by STEI IF ITB and Laboratory for Information dan Communication Techology of Agency for The Assesment and Application Technology as a part of thesis research. We would like to thank the Agency for The Assessment and Application of Technology for providing the server infrastructure.

REFERENCES

[1] Github, "The fifteen most popular languages on github," October 2017. [Online]. Available: https://octoverse.github.com. [Accessed December 2017].

[2] w3tech, "Usage statistics and market share of PHP for websites," 15 October 2018. [Online]. Available: https://w3techs.com/technologies/details/pl-php/all/all. [Accessed 16 October 2018].

[3] The Open Web Application Security Project (OWASP), "OWASP Top 10 - 2013," OWASP, 2013.

[4] L. Vaas, "Hackers sentenced for SQL injections that cost $300 million," Naked Securities by SOPHOS, 19 February 2018. [Online]. Available: https://nakedsecurity.sophos.com/2018/02/19/hackers-sentenced-for-sql-injections-that-cost-300-million/. [Accessed 15 10 2018].

[5] J. Zhao and R. Gong, "A New Framework of Security Vulnerabilities Detection in PHP Web Application," *9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing,* 2015.

[6] N. Jovanovic, E. Kirda and C. Kruegel, "Precise alias analysis for static detection of web application vulnerabilities," in *Proceedings of the 2006 Workshop on Programming Languages and Analysis for Security*, Ontario, 2006.

[7] D. Johannes and T. Holz, "Simulation of Built-in PHP Features for Precise Static Code Analysis," in *Proceedings of the 21st Network and Distributed System Security Symposium*, 2014.

[8] A. I. Sotirov, Automatic Vulnerability Detection Using Static Source Code Analysis, University of Alabama. Departement of Computer Science, 2005.

[9] S. Brindha, K. Prabha and S. Sukumaran, "A survey on classification techniques for text mining," in *Advanced Computing and Communication Systems (ICACCS), 2016 3rd International Conference on*, Coimbatore,India, 2016.

[10] F. Yamaguchi, F. Lindner and K. Rieck, "Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning," in *WOOT'11 Proceedings of the 5th USENIX conference on Offensive technologies*, San Fransisco,CA, 2011.

[11] F. Yamaguchi, M. Lottman and K. Rieck, "Generalized Vulnerability Extrapolation using Abstract Syntax Trees," in *Proceedings of the 28th Annual Computer Security Applications Conference*, Orlando, Florida, USA , 2012.

[12] I. Medeiros, N. Neves and M. Correia, "DEKANT: a static analysis tool that learns to detect web application vulnerabilities," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, Saarbrücken, Germany, 2016.

[13] w3tech, "Usage statistics and market share of PHP for websites," W3Tech - Web Technology Surveys, 15 10 2018. [Online]. Available: https://w3techs.com/technologies/details/pl-php/all/all. [Accessed 16 10 2018].

[14] M. Howard, D. LeBlanc and J. Viega, 19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them (Security One-off), California: McGraw Hill, 2005.

[15] Acunetix, "What is directory trasversal," Acunetix, [Online]. Available: https://www.acunetix.com/websitesecurity/directory-traversal/. [Accessed 10 2017].

[16] N. Popov, "Nikic PHP Parser," [Online]. Available: https://github.com/nikic/PHP-Parser.

[17] "List of PHP token," [Online]. Available: http://php.net/manual/en/tokens.php.

[18] M. Backes, K. Rieck, M. Skoruppa, B. Stock and F. Yamaguchi, "Efficient and Flexible Discovery of PHP Application Vulnerabilities," *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on,* 2017.