I J C T A, 9(16), 2016, pp. 7885-7896
© International Science Press

Predicting web services Vulnerability Using Static and Dynamic Program Analysis

T. Koslandra* and A. Murugan**

ABSTRACT

Web software developers or engineers always in a search of support in finding vulnerable code any practical method for valuable code predicting would enable to give security efforts. While moving in this paper we are going to propose a set of static along with dynamic code attributes that tokenize inputs validating and sanitizing code patterns and is expected to provide the same security features. As all know static and dynamic program analysis always complement each other and both are needed to give accurate and scalable way to predict vulnerability. For many applications in real world may not have past vulnerability data or at least not available fully. hence, to address both situations where labeled past data is fully there or not, we use both supervised and semi supervised learning when building both static and dynamic code patterns. We describe how to use this learning effectively about schema for vulnerability prediction empirical case studies on seven open source projects where we will be creating evaluated supervised and semi-supervised models. The semi-supervised model showed an average improvement of 24% more recall and 3 % less probability of false alarm, thus suggesting semi-supervised learning will be a better solution for many problematic applications where vulnerability data is missing. The supervised are ready to get 77% of recall of false alarm for predicting SQL injection, cross site scripting, remote code execution as vulnerabilities.

Keywords: program analysis, empirical study, input validation and sanitization, Security measures, Vulnerability prediction.

1. INTRODUCTION

Unfortunately, they often have limited time to follow up with new arising security issues and are often not provided with adequate security training to become aware of state of the art web services security techniques. As web software is also highly accessible, web application vulnerabilities arguably have greater impact than vulnerabilities in other types of software. Web developers are directly responsible for the security of web applications. WEB services play an important role in many of our daily activities such as social networking, email, banking, shopping, registrations, and so on.There are also scalable vulnerability prediction approaches such as shin et al. [23]. But the granularity of current prediction approaches is coarse-grained: they identify vulnerabilities at the level of software modules or components. Hence, alternative or complementary vulnerability detection solutions that are scalable, accurate, and fine grained would be beneficial to web developers. To address these security threats, many web vulnerability detection approaches, such as static taint analysis, dynamic taint analysis, modeling checking, symbolic and console testing, have been proposed. Static taint analysis approaches are scalable in general but are ineffective in practice due to high false positive rates [11], [21]. Dynamic taint analysis [11], model checking [22], symbolic [27] and console [21] testing techniques can be highly accurate as they are able to generate real attack values, but have scalability issues for large systems due to path explosion problem [30]. The approach is fine-grained because it identifies vulnerabilities at program statement levels. We use both static and dynamic program analysis techniques to extract IVS attributes. Static analysis can help assess general properties of a program. Yet, dynamic analysis can focus on more specific code characteristics that are complementary to the

* Computer Science and Engineering SRM University Chennai, Tamilnadu, *Email: kosaltoleti@gmail.com*

** Assistant Professor (Sr.G) Computer Science and Engineering SRM University Chennai, Tamilnadu, *Email: murugan.abap@gmail.com*

information obtained with static analysis. We use dynamic analysis only to infer the possible types of input validation and sanitization code, rather than to precisely prove their correctness, and apply machine learning on these inferences for vulnerability prediction. Therefore, we mitigate the scalability issue typically associated with dynamic analysis.

- We had only made use of data dependency graphs to identify input validation and sanitization methods. But some of these methods may be identified from control dependency graphs, e.g., input condition checks, which ensure that valid inputs are often implemented through predicates. Therefore, in this work, to better identify those methods, we leverage control dependency information.
- We propose static slicing and dynamic execution techniques that effectively mine both data dependency and control dependency information and describe the techniques in detail.
- We modified our prototype tool, PhpMiner, to mine the control dependency information and to extract additional attributes.
- First, we evaluated supervised learning models built from IVS attributes. Based on cross validation, the model achieves 77 percent recall and 5 percent probability of false alarm, on average over 15 datasets, across SQLI, XSS, and RCE vulnerabilities. From a practical standpoint, the results show that our approach detects many of the above common vulnerabilities at a very small cost (low false alarm rate), which is very promising considering that the existing approaches either report many false warnings or miss many vulnerabilities.
- Second, we compared supervised and semi-supervised learning models with a low sampling rate of 20 percent (i.e., only 20 percent of the available training data are labeled with vulnerability information). On average, the supervised model achieves 47 percent recall and 8 percent probability of false alarm, whereas the semi-supervised model achieves 71 percent recall and 5 percent probability of false alarm. However, when compared to the supervised model based on complete vulnerability data, on average, the semi-supervised model achieves the same probability of false alarm but a 6 percent lower recall. Therefore, our results suggest that when sufficient vulnerability data is available for training, a supervised model should be favored. On the other hand, when the available vulnerability data is limited, a semi-supervised model is probably a better alternative.

2. BACKGROUND

This paper targets SQLI, XSS and RCE vulnerabilities. These security risks, if exploited, could lead to serious issues such as disclosure of confidential, sensitive information, integrity violation, and denial of service, loss of commercial confidence and customer trust, and threats to the continuity of business operations.All these types of vulnerabilities are caused by potential weakness in web applications regarding the way they handle user inputs. They are briefly described using PHP code examples in the following.

2.1. SQL Injection

SQLI vulnerabilities occur when user input is used in database queries without proper checks. It allows attackers to trick the query interpreter for execute unintended code parts or access to unauthorized data. Consider the following code:

```
Mysql_query SELECT * FROM user WHERE uid
=( ““‘.$_GET[‘id’].”‘“);
```

As the validity of input parameter $_GET is not checked, an SQLI attack can be conducted by providing the parameter id with the following values:

```
/login.php? id ¼ xxx’þORþ’1’%3D’1
```

The query becomes SELECT * FROM user WHERE

uid ='xxx'
OR '1' ¼ '1'.

Effectively, the attack changes thesemantics of the query to SELECT * FROM user, which providesthe attacker with unauthorized access to the user table. Like mysql_query, any other language built-in functions such as mysql_execute that interact with the database can cause SQLI.

2.2. Cross Site Scripting

XSS flaws arise when the user input is used in HTML output statements without proper validation or escaping. It allows attackers to execute scripts in the victim's browser, which may get access to user data, deface web sites, or redirect the user to malicious sites. Consider the following code:

Echo Welcome. $_GET ['new_user'];

Similar to the above SQLI example, as the input parameter $_GET is not checked; an XSS attack can be Conducted by providing the parameter new_user with the following values:

<Script>alert (document. Cookie) ;< /script>

When the victim's browser executes the script sent by the server, it shows the new user's cookie values instead of the intended user information. Using a more malicious script, a redirection to the attacker's server is also possible and sensitive user information could be redirected. Like echo, any other language constructs or functions such as print that generate HTML output could cause XSS.

2.3. Remote Code Execution

RCE vulnerability refers to an attacker's ability to execute arbitrary program code on a target server. It is caused by user inputs in security sensitive functions such as file system calls (e.g., fwrite), code execution functions (e.g., eval), command execution functions (e.g., system), and directory creating functions (e.g., mkdir). It allows a remote attacker to execute arbitrary code in the system with administrator privileges. It is an extremely risky vulnerability, which can expose a web site to different attacks, ranging from malicious deletion of data to web page defacing. The following code depicts RCE vulnerability.

```
$comments ¼ $_POST ['comments'];
$log ¼ fopen('comments.php','a');
fwrite ($log,'<br />'.'<br />'.'<center>'.
'Comments::'.'<br />'.$comments);
```

The above code retrieves user comments and logs them without sanitization. This means that an attacker can execute malicious requests, ranging from simple information gathering using phpinfo() to complex attacks that obtain a shell on the vulnerable server using shell_exec (). Other sensitive PHP functions and operations associated with this vulnerability type include header, preg_replace() with "/e" modifier on, fopen, $_GET ['func_name'], $_GET['argument'], vassert, create_function, and unserialize.

3. CLASSIFICATION SCHEME

Using these attributes, functions and operations are classified according to their security-related properties (i.e., the type of validation- and sanitization-effects these functions and operations may enforce on the inputs being processed). For example, the PHP function str_replace('<',' ', $input) removes HTML tags from the input. Since the presence of HTML tags in $input could cause XSS, the function has a security property that filtersHTML tags and prevents XSS. Before presenting our proposed approach, in this section, we first describe the IVS attributes (listed in Table 1) on which vulnerability predictors shall be built.

Basically, these attributes characterize various types of program functions and operations that are commonly used (collected from various sources like [1], [17], [18]) as input validation procedures to defend against web services vulnerabilities.

3.1. Static Analysis-Based Classification

That is, developers may implement adequate input validation and sanitization method in fact, they might get failed to find maximum or all the data that could be manipulated by external users, thereby missing all any input used for validating. Therefore, in security analysis, it is important to first identify all the input sources. The first six attributes in Table 1 characterize the classification of user inputs depending on the nature of sources. The reason for including input sources in our classification scheme is that most of the common vulnerabilities arise from the misidentification of inputs. example, Client inputs like HTTP GET parameters should always be sanitized before used in sinks whereas it may not be necessary to sanitize Database inputs if they have been sanitized prior to their storage (double sanitization might cause security problems depending on the context). Which could cause security problems (e.g., an attacker could inject malicious values in HTTP parameters having the same naming as of non-initialized variable by using global parameter in PHP configuration files). The reason for two types of Database inputs—Text-database(string-typedata)andNumeric-database (numeric-type data) is to reflect the fact that string-type data retrieved from data stores can cause second order security attacks such as second order SQLI and stored XSS, while it is difficult to conduct those attacks with numeric-type data.

Known-vulnerable-user corresponds to a class of custom functions that have caused security issues in the past. Known-vulnerable_std characterizes a class of language built-in functions that have caused security issues in the past. For example, according to vulnerability report CVE-2013-3238 [6], security issues. These functions are to be predefined by users based on their experiences or the information obtained from security databases (we referred to CVE [6] and PHP security [47]).

Since the above functions and operations either have clear definitions with respect to security requirements or are associated with known vulnerability issues, they could be predefined in database and classifications can be made statically. This database can be expanded as and when new vulnerability analysis information is available. Un-taint refers to functions of operations that return information not extracted from the input string (e.g., mysql_num_rows). It also corresponds to functions or logic operations that return a Boolean value. The reason for this attribute is that since the outcome values are not obtained from an input, the taint information flow stops at those functions and operations and thus, a sink would not be vulnerable from using those practically.

3.1.1. Hybrid Analysis-Based Classification

In a program, various input validation and sanitization processes may be implemented using language built-in functions and/or custom functions. Since inputs to web applications are naturally strings, string replacement/ matching functions or string manipulation procedures like escaping are generally used to implement custom input validation and sanitization procedures. A good security function generally consists of a set of string functions that accept safe strings or reject unsafe strings this hybrid analysis-based classification is applied for validation and sanitization methods implemented using both standard security functions (i.e., language built-in or custom functions with known and tested security properties) and non-standard security functions. If there are only standard security functions to be classified,

These functions are clearly important indicators of vulnerabilities, but we need to analyze the purpose of each validation and sanitization function since different defense methods are generally required for preventing different vulnerabilities. For example, to protect the code special meaning to SQL parsers is required to escape characters that have meaning to code is used to prevent from XSS vulnerabilities.

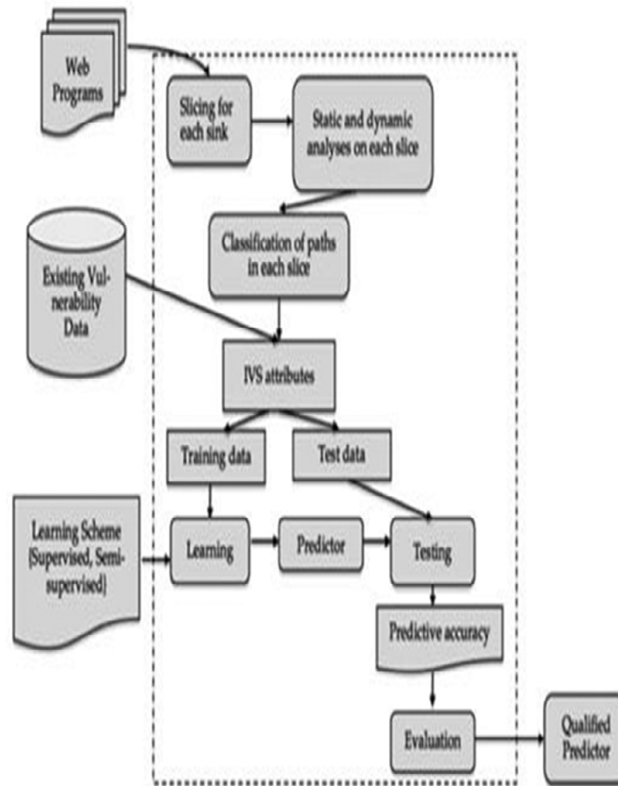

Figure 1: proposed vulnerability prediction model

DB-operator, DB-comment-delimiter, and DB-special basically reflect functions that filter sequence of characters that have special meaning to a database query parser. For example, mysql_real_escape_string is one such built-in function provided by PHP. Clearly, these attributes could predict SQL1vulnerability Null-byte, Dot, DotDotSlash, Backslash, Slash, Newline, Colon, and Other-special reflect functions that filter different types of meta-characters. Filtering Dot (.) character is important to handle unintended file extensions or double file extension cases, which may cause file inclusion attacks (see real world example at CVE-2013-3239). NullByte (%00) characters can be used to bypass sanitization routines and trick underlying systems into interpreting a given value incorrectly

4. FRAMEWORKVULNERABILITY PREDICTION

- Building vulnerability prediction models. We then build vulnerability prediction models from those attributes based on supervised or semi-supervised learning schemes and evaluate them using robust accuracymeasures.
- Hybrid program analysis. For each sink, a backward static program slice is computed with respect to the sink statement and the variables used in the sinks. Each path in the slice is analyzed using hybrid (static and dynamic) analysis to extract its validation and sanitization effects on those variables. The path is then classified according to its input validation and sanitization effects inferred by the hybrid analysis. Classifications are captured with IVS attributes described in Section 3.

4.1. Hybrid Program Analysis

4.1.1. Terms and Definitions Used

A sink is a node in a CFG that uses variables defined from input sources and thus, may be vulnerable to input manipulation attacks. This allows us to predict vulnerabilities at statement levels. Input nodes are the nodes at which data from the external environment are accessed. A variable is tainted if it is defined frominput

nodes.Our analysis is based on the control flow along with (CFG), the program dependence graph used by (PDG), the system dependence graph (SDG) of a web service. Each node in the graphs represents one source code statement. We may therefore use program statement and node inter-changeably depending on the contextslicing algorithm based on the SDG [32]. We first construct the PDG for the main method for the web service andalso used to create the PDGs methods called from the main method according to the algorithm given by Ferranti et al. [8]. We then construct the SDG. A PDG models a pro-gram procedure as a graph in which the main level node use to represent the code statement and the edges represent data or control dependences between statements. SDG extends PDG by modeling interprocedural relations between the main pro-gram and its subprograms.

4.1.2. Hybrid Analysis

Typically, a web application program accesses inputs and propagates them via tainted variables for further process to the service's logic. These processes may often include sensitive program operations such as database updating, generating HTML outputs, and file access rights. In the program variables propagating the input data are not properly checked before being used in those sinks, vulnerabilities arise. Therefore, to prevent web application vulnerabilities, developers typically employ input validation and input sanitization methods along the paths propagating the inputs to the sinks. As default, input generated for web service codes is string. As such, input validation checks and sanitization operations performed in a program are mainly based on string operations. These operations typically include language built-in validation and sanitization functions (e.g., mysql_real_escape_string), string replacement and string matching functions (e.g., str_match), and regular-expression-based string replacement and matching functions (e.g., preg_replace).

Firstly, we maid approached attempts for knowing solution to follow research tasks "Given a slice of sink, from the types and Fig. 3. CFG of a program slice on tainted variables (a) $id and $pwd at sink 7 and (b) $name at sink 10.Numbers of inputs, and the types and numbers of input validation and sanitization functions identified from each path in the slice, so prediction of vulnerability is easy, so our target is to find the potential problems of validation checks and sanitization operations on tainted variables using static and dynamic analyses, and classify those operations based on these inferences. For every path in S_k that propagates the values of tainted variables into k, we carry out this analysis.

Our hybrid (static and dynamic) analysis includes the techniques proposed by Balzarotti et al. [11]. Basically, for each function in a data flow graph, Balzarotti et al. first analyze the function's static program properties in an attempt to determine the potential sanitization effect of the function on the input. If this static analysis is likely to be imprecise, then they simulate the effect of the sanitization functions on the input by executing the code with different test inputs, containing various types of attack strings. The execution results are then passed to a test oracle, which evaluates the functions' sanitization effect by checking the presence of those attack strings. Compulsory static analysis. We classify each path P_i according to our classification scheme (Section 3) using static analysis first. Standard security functions and some of the language built-in functions/operations can be statically and precisely classified based on their known specific security requirements or their functional properties. We classify such functions and operations into different types according to their security-related properties and store that classification in a database. If a node n in P_i processes a tainted variable that is also used by sink k, we analyze its static properties such as the language parameters and operators used by n, and also the functions with known, specific security purposes that are invoked by n. Then, if there is any match to our predefined classifications, n is classified accordingly.

To illustrate, recall the code snippet in Fig. 2:

```
4 $name ¼ $_POST['name'];
8 $name ¼ 'Welcome '. $name;
```

In statement 4, $_POST is a language parameter which we can predefine as a type of input. In statement 8, the language operator that performs string concatenation (.) is used. As this operation only propagates the values of tainted variables to the next operation, we could classify (.) as a taint propagation type.

Likewise, as shown in Fig. 3, standard securities functions can be identified from P_1 of both sink 7 and sink 10. As sink 7 has only one path, it would only require static analysis for the whole classification process. For P_1 of sink 7, we would identify two standard validation and sanitization functions that process $id, which is also used in sink 7.

4.2. Building Vulnerability Prediction Model

Many machine learning techniques can be used to build vulnerability predictors. Regardless of the specific technique used, the goal is to learn and generalize patterns in the data associated with sinks, which can then be efficiently used for predicting vulnerability for new sinks. As more sophisticated security attacks are being discovered, it is important for a vulnerability analysis approach to be able to adapt. With machine learning, it is possible to adapt to new vulnerability patterns via re-training.

4.2.1. Data Representation

Our unit of measurement, an instance in machine learning terminology, is a path in the slice of a sink and we characterize each path with IVS attributes. The attribute values may range from zero to an upper bound that depends on the number of classified program operations or functions. Since we propose 33 IVS attributes (Table 1), each path would be represented by a 33-dimensional attribute vector. To illustrate, Fig. 6 shows the attributes for sink 7 and sink 10 extracted from the paths in their respective slices. The last column is the class attribute to be predicted, that is whether a sink is vulnerable or not in a given path. In our case studies, this comes from existing vulnerability data.

Attribute / <$Path_{id}$, $Sink_{id}$>	Client	Text-database	Propagate	Numeric	DB-special	String-delimiter	Script-tag	Vuln? (tagged by user)
<P_1, $Sink_7$>	1	0	0	1	1	1	0	No
<P_1, $Sink_{10}$>	0	1	1	0	0	0	0	Yes
<P_2, $Sink_{10}$>	1	0	1	0	0	1	1	No

4.2.2. Data Preprocessing

Data balancing. As shown in Table 3, in most of our datasets, the proportion of vulnerable sinks to non-vulnerable ones is small. This is an imbalanced data problem and should be expected in many such vulnerability datasets. Prior studies have shown that imbalanced data can significantly affect the performance of machine learning classifiers [19], [49] because some of the data might go unlearned by the classifier due to their lack of representation, thus leading to induction rules which tend to explain the majority class data and favoring its predictive accuracy. Since for our problem, the minority class data capture the ‘vulnerable’ instances, we need a high predictive accuracy for this class as missing vulnerability is far more critical than reporting a false alarm. To address this problem, we use a sampling method called adaptive synthetic oversampling [48]. It balances the (unbalanced) data by generating synthetic, artificial data for the minority class instances, thus reducing the bias introduced by the class imbalance problem. It does not require modification of standard classifiers and thus, can be conveniently added as an additional data preprocessing step [49].

Given an imbalanced data ds with majority class data ds_{maj} and minority class data ds_{min}, the algorithm to generate synthetic data, given by He et al. [48], can be summarized as follows:-

4.2.3. Supervised Learning

Classification is a type of supervised learning methods because the class label of each training instance has to be provided. In this paper, we try to perform logistic regression procedure (LR) and RandomForest method (RF) model from the proposed attributes. There are two reasons for choosing these two types of classifiers: 1) These classifiers were benchmarked as among the top classifiers in the literature [14], 2) LR-based predictor achieved the best result in our initial work [33] and yields results that are easy to interpret in terms of the impact of attributes on vulnerability predictions.

4.2.4. Semi-Supervised Learning

As discussed above, for supervised learning, we use LR and RF, the latter being a type of ensemble learning method that has achieved high accuracy in the literature [14]. However, as ensemble learning works by

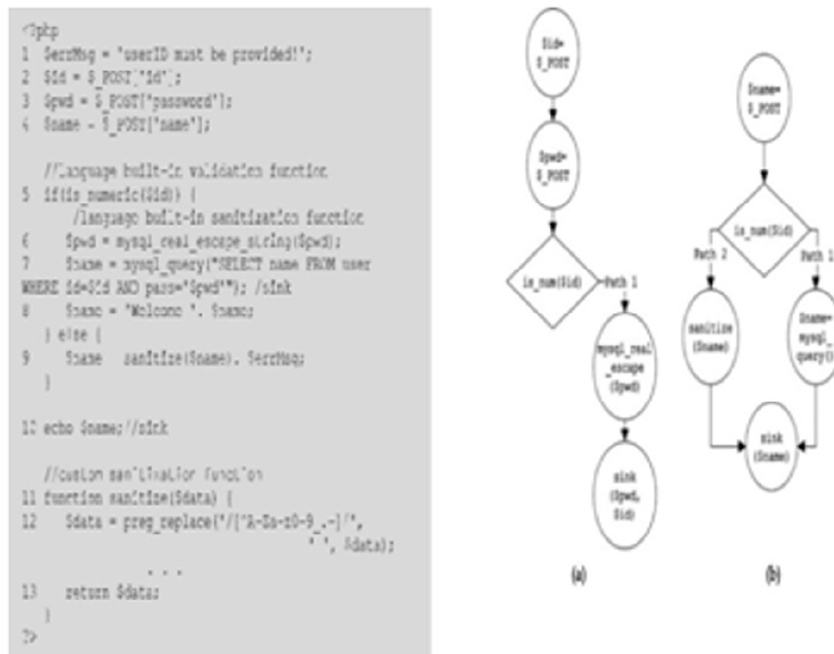

Figure 2: Prediction model evaluation procedure.

```
Procedure cross_validate (Dataset ds, Learner C) {
    Prediction model M
    Accuracy acc ← 0
    repeat 10 times {
        B ← Randomly divide ds into 5 equal bins
        for each bin b_i in B {
            ds_test        ← b_i
            ds_train       ← B - {b_i}
            ds_train_bal   ← adaptive synthetic
                                 oversampling of ds_train
            bestAttr       ← correlation-based feature
                               subset selection from ds_train_bal
            M              ← Train C on ds_train_bal
                                           and bestAttr
            Test M on ds_test
            acc            ← acc + predictive accuracy of M
        }
    }
    return acc ← acc / 50
}
```

Dataset	#Instances	#Vuln. instances
(a) Datasets with SQL injection vulnerabilities		
schmate-sqli	189	152
faqforge-sqli	42	17
phorum-sqli	122	5
cutesite-sqli	63	35
(b) Datasets with cross-site scripting vulnerabilities		
schmate-xss	172	138
faqforge-xss	115	53
utopia-xss	86	17
phorum-xss	237	9
cutesite-xss	239	40
myadmin1-xss	305	20
myadmin2-xss	425	14
(c) Datasets with remote code execution vulnerabilities		
myadmin1-rce	221	3
myadmin2-rce	297	5
(d) Datasets with file inclusion vulnerabilities		
myadmin1-fi	139	5
myadmin2-fi	121	2

Dataset	Learner	*pd*	*pf*	*pr*
schmate-sqli	LR	91	4	98
	RF	93	4	98
faqforge-sqli	LR	82	21	51
	RF	87	3	89
phorum-sqli	LR	100	2	42
	RF	100	1	46
cutesite-sqli	LR	85	6	95
	RF	89	8	94
Mean results on	**LR**	**90**	**8**	**72**
SQLI prediction	**RF**	**92**	**4**	**82**

Dataset	Learner	*pd*	*pf*	*pr*
schmate-xss	LR	72	19	94
	RF	84	18	95
faqforge-xss	LR	76	21	79
	RF	89	10	90
utopia-xss	LR	67	14	61
	RF	69	21	53
phorum-xss	LR	79	3	53
	RF	69	2	56
cutesite-xss	LR	83	4	78
	RF	76	5	75
myadmin1-xss	LR	77	11	35
	RF	68	4	56
myadmin2-xss	LR	56	5	29
	RF	48	1	62
Mean results on	**LR**	**73**	**11**	**61**
XSS prediction	**RF**	**72**	**9**	**70**

Dataset	Learner	*pd*	*pf*	*pr*
myadmin1-rce	LR	67	1	43
	RF	67	1	53
myadmin2-rce	LR	60	1	47
	RF	60	1	56
Mean results on	**LR**	**64**	**1**	**45**
RCE prediction	**RF**	**64**	**1**	**55**

Dataset	Learner	*pd*	*pf*	*pr*
myadmin1-fi	LR	62	1	79
	RF	52	2	60
myadmin2-fi	LR	100	1	65
	RF	100	0	94
Mean results on	**LR**	**81**	**1**	**72**
FI prediction	**RF**	**76**	**1**	**77**

combining individual classifiers, it typically requires significant amounts of labeled data for training. In certain industrial contexts, relevant and labeled data available for learning may be limited.

Semi-supervised methods [39] use, for training, a small amount of labeled data together with a much larger amount of unlabeled data. This method that exploits non-labeled dataset that can start logical learning tasks with very few labeled data. As explained by Zhou [43], combining semi-supervised learning with ensembles has many advantages. Unlabeled data is exploited to help enrich labeled training samples allowing ensemble learning:

5. CONCLUSION WORK

In proposed model we are trying to separate different vulnerabilities and for each would be using the token parser approach and conforming the risk and performing necessary actions to overcome and in future also if same error will come, the system has to take its own independent decision.

6. DISCUSSIONSAND CONCLUDING REMARKS

The main goal of this paper is to achieve both high accuracy and good scalability in detecting web application vulnerabilities. In principle, our proposed approach leverages all the advantages provided by existing static and dynamic taint analysis approaches and further enhances accuracy by using prediction models developed with machine learning techniques and based on available vulnerability information. Static analysis is generally sound but tends to generate many false alarms. Dynamic analysis is precise but could miss vulnerabilities as it is difficult or impossible to exercise every test case scenario. Our strategy consisted in building predictors using machine learners trained with the information provided by both static and dynamic analyses and available vulnerability information, in order to achieve good accuracy while meeting scalability requirements. To generalize our current results, our experiment can be easily replicated and extended as we made our tool and data available online [7]. We also intend to conduct more experiments with industrial applications. While we believe that the proposed approach can be a useful and complementary solution to existing approaches, studies need to be carried out to determine the feasibility and usefulness of integrating multiple approaches.

REFERENCES

[1] OWASP. (2012, Jan.). The open web application security project [Online]. Avaialble: http://www.owasp.org

[2] LK. Shar and HBK. Tan, "Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns," Inf. Soft. Technol., vol. 55, no. 10, pp. 1767– 1780, 2013.

[3] Y. Xie and A. Aiken, "Static detection of security vulnerabilities in scripting languages," in Proc. USENIX Security Symp., 2006, pp. 179–192.

[4] (2012, Mar.). SourceForge. [Online]. Available: http://www.sour-ceforge.net

[5] (2013, May). CVE: Distributions of vulnerabilities by types [Online]. Available: http://www.cvedetails.com/vulnerabilities-by-types.php

[6] PhpMiner Availble-http://sharlwinkhin.com/ phpminer.html, 2013.

[7] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The service dependence table flow and optimization," ACM Trans. Program. Languages Syst., vol. 9, pp. 319–349, 1987.

[8] I. H. Witten, E. Frank, and M. A. Hall, Data Mining, 3rd ed. San Mateo, CA, USA: Morgan Kaufmann, 2011.

[10] 2012, Mar.). RSnake [Online]. Available: http://ha.ckers.org

[11] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C.Kruegel, and G. Vigna, "Composing static along with dynamic analysis to validate sanitization in web applications," in Proc. IEEE Symp. Private Security , 2008, pp. 387–401.

[12] L. C. Briand, J. Wust,€ J. W. Daly, and D. V.Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," J. Syst. Softw., vol. 51, no. 3, pp. 245–273, 2000.

[13] E. Arisholm, L. C. Briand, and E. B. Johannessen, “A systematic and comprehensive investigation of methods to build and evalu-ate fault prediction models,” J. Syst. Softw., vol. 83, no. 1, pp. 2–17, 2010.

[14] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction: a proposed framework and novel findings,” IEEE Trans. Softw. Eng., vol. 34, no. 4, pp. 485–496, Jul./Aug. 2008.

[15] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, “Defect prediction from static code features: current results, limi-tations, new approaches,” Automated Softw. Eng., vol. 17, no. 4, pp. 375–407, 2010.

[16] L. K. Shar and H. B. K. Tan, “Predicting common web application vulnerabilities from input validating code patterns,” in Proc. Int. Conf. Automated Softw. Eng., 2012, pp. 310– 313.

[17] C. Anley, Advanced SQL Injection in SQL Server Applications, Next version of Security Software Ltd., White Paper, 2002.

[18] S. Palmer, Web services applicable vulnerabilities: Detect, exploit, pre-vent, Syngress, 2007.

[19] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. Matsu-moto, “The effects of over and under sampling on fault-prone module detection,” in Proc. Int. Symp. Empirical Softw. Eng. Meas., 2007, pp. 196–204.

[20] J. Demsar, “Statistical comparisons of classifiers over numerous data parts,” J. Mach. Learning Res., vol. 7, pp. 1–30, 2006.

[21] A. Kiezun, P. J. Guo, K. Jayaraman, and M. D. Ernst, “Automated created of SQLI and cross-site scripting attacks,” in Proc. Int. Conf. Softw. Eng., 2009, pp. 199–209.

[22] M. Martin and M. S. Lam, “Automatic generation of XSS and SQL injection attacks with goal directive modeled check,” in Proc. USENIX Security Symp., 2008, pp. 31–43.

[23] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, “Evaluattion of complex code churn, and developed metrics as indicators of software vulnerabilities,” IEEE Trans. Softw. Eng., vol. 37, no. 6, pp. 772–787, Nov./Dec. 2011.

[24] J. Walden, M. Doyle, G. A. Welch, and M. Whelan, “Security in open web services ,” in Proc. Int. Symp. Empirical Softw. Eng. Meas., 2009, pp. 545–553.

[25] N. Nagappan, T. Ball, and B. Murphy, “Using historical in-process and product metrics for early estimation of software failures,” in Proc. Int. Symp. Softw. Rel. Eng., 2006, pp. 62–74.

[26] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, “Predicting vulnerable software components,” in Proc. ACM Conf. Comput. Commun. Security, 2007, pp. 529–540.

[27] X. Fu and C.-C. Li, “A string constraint solver for detecting web application vulnerability,” in Proc. Int. Conf. Softw. Eng. Knowl. Eng., 2010, pp. 535–542.

[28] G. Wassermann and Z. Su, “Sound and precise analysis of web applications for injection vulnerabilities,” in Proc. ACM SIGPLAN Conf. Program. Language Des. Implementation, 2007, pp. 32–41.

[29] L. K. Shar and H. B. K. Tan, “Automated removal of cross site scripting vulnerabilities in web applications,” Inf. Softw. Technol., vol. 54, no. 5, pp. 467–478, 2012.

[30] K.-K. Ma, K. Y. Phang, J. S. Foster, and M. Hicks, “Directed sym-bolic execution,” in Proc. Int. Conf. Static Anal., 2011, pp. 95–111.

[31] M. Weiser, “Program slicing,” in Proc. Int. Conf. Softw. Eng., 1981, pp. 439–449.

[32] S. Horwitz, T. Reps, and D. Binkley,” slice procedure using dependence graphs,” ACM Trans. Program. Languages Syst., vol. 12, no. 1, pp. 26–61, 1990.

[33] L. K. Shar, H. B. K. Tan, and L. C. Briand, “Mining SQL injection and CSS vulnerable using program analysis,” in Proc. Int. Conf. Softw. Eng., 2013, pp. 642–651.

[34] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors.13, Jan. 2007.

[35] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, “A general softwaredefect-proneness prediction framework,” IEEE Trans. Softw.Eng., vol. 37, no. 3, pp. 356–370, May/Jun. 2011.

[36] D. Fisher, L. Xu, and N. Zard, “Ordering effects in clustering,” in Proc. Int. Workshop Mach. Learning, 1992, pp. 163–168.

[37] L. Breiman, “Random forests,” Mach. Learning, vol. 45, no. 1,pp. 5–32, 2001.

[38] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, Applied Logistic Regression, 3rd ed. New York, NY, USA: Wiley, 2013.

[39] O. Chapelle, B. Sch€olkopf, and A. Zien, Eds, Semi-Supervised Learning. Cambridge, MA, USA: MIT Press, 2006.

[40] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, “Sample-based software defect prediction with active and semi-supervised learning,” Automated Softw.Eng., vol. 19, pp. 201–230, 2012.

[41] H. Lu, B. Cukic, and M. Culp, “Software defect prediction usingsemi-supervised learning with dimension reduction,” in Proc. Int. Conf. Automated Softw. Eng., 2012, pp. 314–317.

[42] M. Li and Z.-H. Zhou, “Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples,” IEEE Trans. Syst., Man Cyberne., Part A: Syst. Humans, vol. 37, no. 6, pp. 1088–1098, Nov. 2007.

[43] Z.-H. Zhou, “When semi-supervised learning meets ensemble learning,” in Proc. Int. Workshop Multiple Classifier Syst., 2009, pp. 529–538.

[44] Chord: A versatile platform for program analysis. (2011). Proc. Tutorial ACM Conf. Program.Language Des.Implementation [Online]. Available: http://pag.gatech.edu/chord

[45] F. Yamaguchi, M. Lottmann, and K. Rieck, “Generalized vulnerability extrapolation using abstract syntax trees,” in Proc. Annu.Comput. Security Appl. Conf., 2012, pp. 359–368.

[46] F. Yamaguchi, C. Wressnegger, H. Gascon, and K. Rieck, “Chucky: Exposing missing checks in source code for vulnerability discovery,” in Proc. ACM SIGSAC Conf. Comput. Commun. Security, 2013, pp. 499–510.

[47] PHP Security [Online]. Available: http://www.php.net/manual/en/security.php, 2013.

[48] H. He, Y. Bai, E. A. Garcia, and S. Li, “ADASYN: Adaptive synthetic sampling approach for imbalanced learning,” in Proc. Int. Joint Conf. Neural Netw., 2008, pp. 1322–1328.

[49] H. He and E. A. Garcia, “Learning from imbalanced data,” IEEE Trans. Knowl. Data Eng., vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[50] M. A. Hall, “Correlation-based feature selection for machine learning,” Ph.D. thesis, Dept. Comput. Sci., Univ. Waikato, Hamilton, New Zealand, 1998.[51] PHPTop 5 [Online]. Available: https://www.owasp.org/index. php/PHP_Top_5, 2014.