Arab J Sci Eng (2017) 42:885–895
DOI 10.1007/s13369-016-2362-5

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Analysis of Web Application Code Vulnerabilities using Secure Coding Standards

Divya Rishi Sahu[1] · **Deepak Singh Tomar**[1]

Received: 4 June 2016 / Accepted: 17 November 2016 / Published online: 5 December 2016

Abstract The evolution of modern web application involves a broad range of web technologies such as ActiveX, JavaScript and CGI. It mitigates the demand of bolt-on security services, but remains suffered from code vulnerabilities. Two common issues of web application code vulnerability are the wrong style of writing code and improper server configuration. Enhancing the web application's functionalities and ease of use are the primary concern of developers. Security is their second concern, resulting in code vulnerabilities. The application developers may effectively deal with code vulnerabilities through adhering to Secure Coding Standards. But manually applying all the Secure Coding Standard is susceptible to human errors. Hence, a graph-based interactive system is developed in the context of Secure Coding Standards to handle code vulnerabilities. Evaluation of the developed system is carried out by using standard available datasets such as CVE, NVD, Syhunt Vulnerable PHP Code and OWASP.

Keywords Web application security · Defensive programming principle · Web graph · XML · Code snippet · Vulnerability

1 Introduction

Earlier, web applications were mostly concerned with end-to-end communication and static Web site. In the last decades, there has been a significant enhancement in the existing web applications. Existence of the dynamic Web site, payment system, social media, web portal, etc. is feasible due to the advancement in the coding functionalities. Programming languages are enhancing and introducing new features such as dynamic query execution, remote file inclusion and command execution at run time. On the other hand, the new coding features also introduce the vulnerabilities into the web applications.

Securosis [1] reported that half of the time the security is not the priority for the developers during coding. Developer's inadequate attention on mitigation of the security threats causes their web application vulnerable to potential attack exploitation. Even then, security-conscious developers may mitigate the source code vulnerabilities to an extent through practicing the existing theoretical Secure Coding Standards. Errors are the part of developers, and human nature makes this kind of mistakes that lead to insecure code. It leads to the source code vulnerabilities.

Today's landscapes of web application security are frequently exploited by the attackers due to the vulnerabilities in source code and exploitation of language-specific functionalities. The proper balancing between language-specific features and code vulnerabilities should be maintained by the web developers to reinforce the web application security.

Due to the heterogeneous nature, the ubiquity of web application and proper configuration, it is imperative that developers commit proper attention to pursuing Secure Coding Standards. It is unrealistic to edify every web developer and ensure that they are seeking Secure Coding Standards. However, skilled the developer manually is susceptible to human error and is imperfect due to the human nature of obliteration. Hence, an interactive system is developed in this paper which edifies web developers to practise the theoretical Secure Coding Standards to mitigate the pitfall of code vulnerabilities occurred due to the poor style of writing code.

✉ Divya Rishi Sahu
divyarishi.sahu@gmail.com

Deepak Singh Tomar
deepaktomar@manit.ac.in

[1] Department of CSE, MANIT, Bhopal, M.P, India

This article has the following contribution:

I. Identifies the web application security issues such as logical errors and poorly written codes which lead to the code vulnerabilities.
II. Developed a repository of secure coding rules in the context of text-based Secure Coding Standard to create an interactive system. It helps the security experts actually to train their system according to its roles.
III. Developed an interactive system to scrutinize source code that edifies less experienced developers to improve secure programming skills and also assist in finding out the code vulnerabilities in web application successfully. Salient features of the work are twofold: It is a multi-relational graph-based interactive system in context with the principle of Security Coding Standards. It supports source code comprehension; hence, developer may investigate the code at development time.
IV. The developed system is evaluated and compared by performance metrics against the two open-source projects and standard labeled datasets such as CVE, NVD, Syhunt Vulnerable PHP Code and OWASP. Finally, the results are compared with existing web application tools and techniques such as DevBug, RATs and RIPS. It is a comprehensive approach, which might be acceptable and yield improved performance to mitigate source code vulnerabilities in the web application.

The paper is organized as follows. Section 2 presents the previous research contribution in the area of secure web application development, code vulnerabilities and Secure Coding Standards. Section 3 addresses the two issues of web application security. In Sect. 4, an approach is proposed to edify the web developers during development. After that, an interactive automatic system is developed from the proposed technique. In Sect.5, performance evaluation is carried out in the context of performance metrics. Finally, results are compared with existing web application tools and technologies. Section 6 concludes the work carried out in this paper endows with the possible future scope.

2 Background and Literature Survey

The purpose of literature survey is to gain and understand the web application code vulnerabilities and its analysis techniques, Secure Coding Standards and graph-based generalized association rule mining to establish the context in which source code analysis technique for the web application is developed.

One of the first attempts to detect code injection attacks is based on the Instruction-Set Randomization (ISR) [2]. Hu et al. [3] describe the implementation of ISR using software dynamic translation to detect injected malicious code before execution. It defends application-level binary code injection attacks. The ISR-based approach requires special support for the processor.

Huang et al. [4] proposed a lattice-based static analysis algorithm to ensure web application security. Authors' primary focus is to identify the web application vulnerabilities which occurred due to the data integrity violation.

Anderson et al. [5] detect code injection attack in real time through constructing the Markov chain graph from of system calls executed by the Internet Explorer (IE). The hypothesis of this approach is that exploitation of code injection makes substantial changes in Markov chain for the particular time. The primary cause of false positives is that the size of the chain and time interval may be different for the different environment.

Riley et al. [6] proposed a structural modification in the memory of processors to address code injection vulnerabilities through virtually separating the memory into code memory and data memory. In this structural-change, injected code is putting aside in data memory and not to be executed by the processor. The software implementation of the proposed architecture as a patch to an operating system incurred moderate overhead.

Papagiannis et al. [7] introduced partial taint tracking approach at source code level to prevent the injection vulnerabilities. Implementing proposed technique as a tool modifies the core of scripting at runtime. It analyzes the input values with taint sources to track their origin to identify injection vulnerabilities.

Kneuss [8] developed a static analysis tool to check bugs, misuse of the scripting language and risk of encountering fatal errors after deployment of the applications. However, scripting features such as an exception, namespace, closure and references remain intact.

Jovanovic et al. [9] proposed a concept of flow-sensitive, interprocedural and context-sensitive data flow analysis to address taint-style vulnerabilities. Author implemented it in Pixy to automatically examine vulnerabilities in PHP code. The author further enhances own work with integrating the alias analysis using shadow variables [10]. It statically analyzes the code to identify taint-style vulnerabilities. The false-positive rate of the developed system is nearly 50% [9,10] which constrains enhancement in accuracy.

Ryan Dewhurst [11] implemented an online Linux-based static code analysis framework called DevBug. It is a taint analysis-based technique written in JavaScript to detect vulnerabilities in PHP-based web applications. It works for ten vulnerabilities from user-supplied input.

Secure Software inc. [12] developed a tool named RATS (Rough Auditing Tool for Security) which supports multiple languages. As its name implies, it performs rough analysis

for security and diminishing the common security issues such as user input and buffer overflows.

Dahse and Holz [13] presented an approach and developed a prototype called RIPS, for the precise static analysis of PHP source code. The method is based on intraprocedural, interprocedural data flow analysis and string analysis. It may detect twenty different taint-style security vulnerabilities in web applications. It supports partially the object-oriented PHP code that leads to false negatives.

Woods and McGuirk [14] stated that written code snippet should handle all the expected and unexpected case scenarios. The aim of this paper is to encourage the programmer and minimize the risk of errors when processing real data. Further, it describes some useful programming concepts to mitigate potential hazards lurking within SAS programs. However, identification of all unexpected case scenarios is impracticable in fact.

Zhu et. al. [15] spotted that skilled developers underestimate secure programming practice which introduces programming vulnerabilities. The contribution of it is the static analysis of source code to mitigate the programming vulnerabilities through reminding and assisting developers at the development phase. The author suggested different ways to provide more meaningful, appropriate descriptions to edify less experienced developers with secure programming, as future work.

Victor [16] highlights the need for security principles to develop secure code and to design security principles which are Principle of least privilege, Validate input, keep it simple and Practice defense in depth. Albeit, there are Secure Coding Standards which are left untouched. Practising all principles manually is also unrealistic for developers.

The number of vulnerabilities may be addressed with relatively straightforward Secure Coding Standards. Secure Coding Standards (SCS) are principles for developers to effectively guard against code vulnerability exploitations. It uses small, simple techniques to save many hours of debugging later in a project. Researchers are working in the direction of developing Secure Coding Standards such as Defensive programming principles (DPP) and Secure Coding principles (SCP).

Defensive programming principles [17] are the rules which are based on the secure code design. The idea behind it is that the mistakes of an associate programmer working on the same project also may be cause of vulnerability.

CERT has developed Secure Coding principles through a broad-based community effort, including by the CERT Secure Coding Initiative and members of the software development and software security communities [18].

OWASP security principles are gathered from the previous edition of the OWASP Development Guide and normalized with the security principles outlined in Howard and LeBlanc's excellent Writing Secure Code [19]. These principles provide assistance to the developer or organization to write vulnerability-free code.

In fact, there exist techniques to detect code vulnerabilities in the web application. However, this does not relieve developers from their responsibility to follow secure programming principles. Background and literature survey deduced that web application security necessitates an efficient approach for invulnerability of web applications through scrutinizing the source code on the basis of existing text-based Security Code Standards. It is imperative to develop rules of the Secure Coding Standards to develop an automatic analysis system.

Yen et al. [20] proposed a graph-based approach for mining association rules on the concept of bit vector model. Sharma [21] extended the work and proposed a graph-based association rule mining approach for correlating the primitives and items which have concept hierarchy. The author compared it with apriori algorithm and evaluated that graph-based approach is better in the context of computational time.

3 Issues in the Web Applications

Technologies used for web application development provide a broad range of functionalities such as command execution, socket development, dynamic and interactive support, and graphics design. It also introduces protection, although sometimes that protection triggers its own problem which becomes the vulnerability. Most web applications are vulnerable due to the following two reasons. First is the improper handling of language-specific functionalities, and second is the poor style of writing code which becomes code vulnerability.

3.1 Language-Specific Functionalities

Language-specific functionalities such as evaluation of code dynamically, file inclusion and use of dynamic variables and functions are potentially vulnerable due to the insecure logic of code snippets. These cases would be because of vulnerability such as command injection, SQL injection, remote file inclusion and local file inclusion. Four potentially vulnerable functionalities of PHP with their potential threats are shown in Table 1.

3.2 Web Application Code Vulnerabilities

The most available web applications are vulnerable owing to the code vulnerabilities. Web application code vulnerabilities are increasing due to the poor style of writing code by the less experienced programmer.

Free available templates, open-source web development platforms, minimal hosting cost and their ad hoc nature

Table 1 Features of PHP and related threats

Sr. no.	Features	Functionality	Possible threats
1.	Type safety/ type juggling [8]	It reduces the efforts of the programmer in web development. PHP processor accepts every kind of value in a single variable without any syntax error and executes the code	Due to this functionality, the attacker could execute malicious script in hex or other forms through input fields
2.	'php.ini' File	It manages the configuration settings such as 'allow_url_include' and 'safe_mode' for the security of the PHP Hypertext Processor	It is written in text format which is easily accessible. An attacker could temper php.ini text file through injection attacks
3.	Dynamic code inclusion [8,22]	It allows the developer to execute code or file dynamically in the current domain	An attacker may exploit this feature through remote file inclusion (RFI) or local file inclusion (LFI) attacks
4.	Dynamic command execution [22]	It facilitated developer to execute commands at run time according to the end user requirement	An attacker may use the operating system property to execute two or more commands simultaneously to exploit command injection

Table 2 Taxonomy of web application code vulnerability

Sr. no	Code vulnerability type	Description	Potential attacks
1.	Taint-style vulnerability	A source code vulnerability, which may occur due to the use of un-sanitized data into critical operations of the web application	Injection vulnerabilities are: • PHP injection, • Cross Site Scripting, • Command injection, etc
2.	Path vulnerability	A source code vulnerability, which may occur due to the improper neutralization of special elements within the pathname	Directory traversal: In a web application, it is happening due to the dynamic construction of a file path
3.	Configuration vulnerability	A Source code vulnerability, which may occur due to the unknowingly escaping the programming syntax that modifies the configuration	It includes initialization of variable since uninitialized variable may enable the Global variable in PHP
4.	Error handling vulnerability	A source code vulnerability, which may occur due to the improper error handling	It includes debugging features such as web page status or dumping stack traces on request failure taken place during processing
5.	Zero day vulnerability	A source code vulnerability, which is unknown beforehand, for that developer has not had time to address and patch	It includes the vulnerabilities for which no patch has currently existed

encourage the less experienced programmer to develop more web application. The primary interest of the less experienced developer is to build a user-friendly interface rather than creating the secure system.

Less experienced programmers elide security and write poor code which may become the code vulnerability. The code vulnerability of web application leads to a tempting target for the attacker and creates extremely high risk. Web application code vulnerabilities are classified into following five classes as shown in Table 2.

4 Developed Interactive System

The race to reliability and innovation in smart web technology begins with the secure web application development. Security considerations need to be part of the entire software development process mainly in the development phase.

The developed interactive system is based on Secure Coding Standards, which consists of two phases: training phase and detection phase. Training phase generates the rules according to the developed library of defensive programming

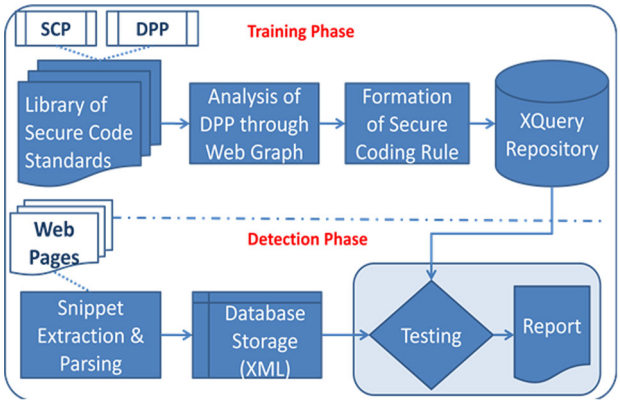

Fig. 1 Architecture of proposed interactive system

principles. Subsequently, it stores the generated rules into an XML database. Detection phase begins with the extraction of code snippets and detects the vulnerabilities, according to the generated rules. The block diagram of proposed model is shown in Fig. 1.

4.1 Building Library of Secure Coding Standards

Secure Coding Standards are explored and gathered from following publicly available sources:

- OWASP Secure Coding Practices Quick Reference Guide [18].
- CWE and CERT Secure Coding Standards [19].
- Principles for Secure Systems [23].
- Fundamental Practices for Secure Software Development [24].

The concepts to write a secure snippet are gathered from the aforementioned text-based security principles and structured to build a library. This library referred to a set of Secure Coding Standards that may be reused to write secure web applications. Among them, three principles of the developed library are represented here as follows.

DPP_1 : Handle unexpected conditions-
It states that developers should keep in mind all the expected conditions which may occur during source code development. In other words, the developer must cover all the conditions that might be possible logically. However, the possibility to forget condition remains to exist. Hence, the developer must write a default message for all unexpected conditions.

DPP_2 : Process external systems data properly-
It states that input values should not pass directly to PHP functions such as exec, system, passthru, require, require-once, include and includeonce. Hence, sanitization of the input data is necessary before passing it to taint functions.

DPP_3 : Monitor Error Messages
It states to avoid error messages, providing valuable info that may be used to break the system and hence to enable log errors for technical support and disable error message display after development. Further, web graph for sample codes is constructed to achieve following objectives.

- To confirm that SCS such as DPP_1, DPP_2 and DPP_3 should be followed correctly.
- To generate rules of the detection engine developed in this paper.

4.2 Analysis of DPP Through Web Graph

Web graph represents the relationship between web objects such as web pages, forms, frames and statements. A single-relational graph is denoted as:

$$G = (V, E \subseteq (V \times V))$$

where
V = Set of vertex and
E = Set of edges.
Each edge is an ordered pair of nodes (v, v_i), representing a direct connection from v to v_i. In a single-relational graph, all vertices and all edges would be homogeneous, which represents graphs would be independent of each other. Therefore, for this work multi-relational directed graph is more feasible. The multi-relational directed graph represents the relation between multiple types of objects. It is denoted as:

$$G = (V, E)$$

where

$$E = ((E_0, E_1, \ldots, E_m) \subseteq (V \times V))$$

To generate graphs of the code snippet, all keywords of PHP scripting language are assigned to vertices and their relation to the edge. For every two vertices v_i and v_j, v_i would be an ancestor of v_j ($v_i > v_j$) if vertex v_j is within the scope of v_i and a directed link from vertex v_i to vertex v_j is created. Pre-order numbering (PON) method is applied to differentiate these vertices.

This method associates a number with each same label according to the following order. The first node is labeled as '*root,*' and then, all vertices are labels according to their keyword in the source code. Every same type of node is increased by one at the same level and by ten at the next level to differentiate in the levels.

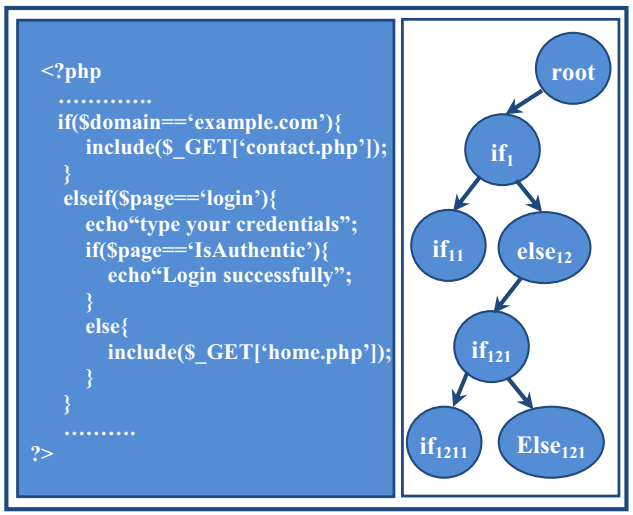

Fig. 2 Constructed graph of ‘*if statement*’ (G_{if}) with code snippet potentially vulnerable to RFI: Case 1

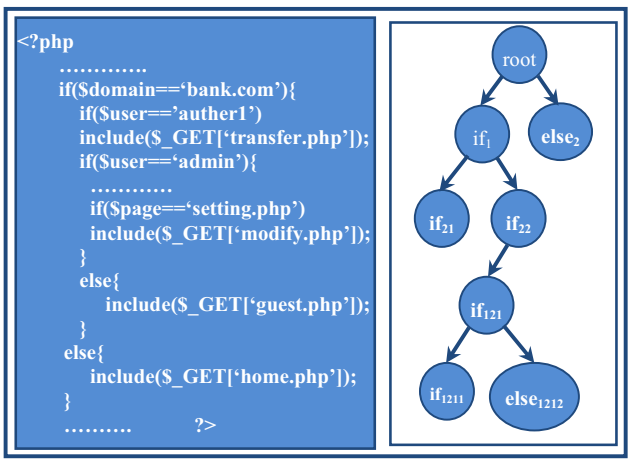

Fig. 3 Constructed graph of ‘*if statement*’ (G_{if}) with code snippet potentially vulnerable to RFI: Case 2

To construct a graph of ‘*if statement’* (G_{if}), each *if* and *else* keyword in the source code is labeled as v_{if} and v_{else}, respectively, and then, a number is associated with each graph node according to the PON method.

The first case of potentially vulnerable code snippets concerning remote file inclusion (RFI) and its constructed graph related to ‘if statement’ (G_{if}) is shown in Fig. 2. There is no symmetry in this potentially vulnerable case among the nodes and child nodes. Each node may contain zero or more than one child nodes.

The second case of potentially vulnerable code snippets concerning remote file inclusion (RFI) and their constructed graph related to ‘if statement’ is shown in Fig. 3. In this potentially vulnerable case, each node has either the zero or the two child nodes.

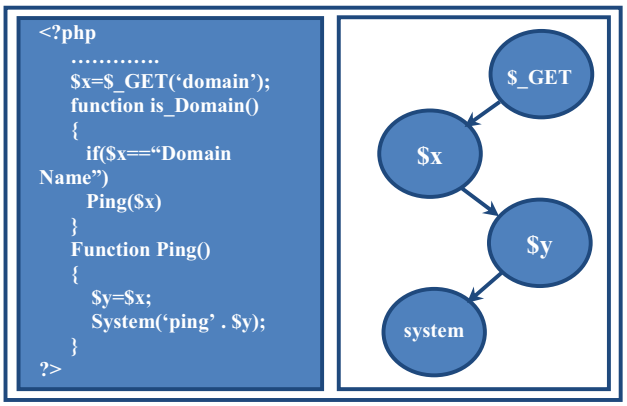

Fig. 4 Constructed graph of *input values* (G_{iv}) with code snippet potentially vulnerable to command injection: Case 3

Another case of potentially vulnerable code snippets concerning command injection attack and its detailed graph of input value (G_{iv}) is shown in Fig. 4.

Analysis of code snippet in the context of the defensive programming principles is useful to build the rules for an interactive system to scrutinize the source code.

4.3 Formation of Secure Coding Rules

Further, a graph-based approach is applied to generate secure coding rules from Secure Coding Standards. It classifies records through a collection of ‘if…then…’ rules.

$$\text{Rule: (Condition)} \rightarrow \text{y}$$

where

Condition = Conjunctions of attributes

Y = Class label

PHP supports two conditional statements ‘*if’ and ‘switch*’ to maintain logical conditions.

Rule-I:

The first rule for if condition states that each vertex (v_i) in the graph of *‘if statement’* (G_{if}) should contain out-degree either equal to 0 or 2. Here, out-degree of a vertex v is represented as the number of distinct edges going out.

$(v, v_1)\ldots\ldots(v, v_k) \quad \in G_{if}$,

Hence, from the DPP_1 first rule is:

Rule I_A:

$$\forall v_i \in G \equiv deg^{+}(v_i) = \begin{cases} 0, & \textit{pendent vertex or null graph} \\ 2, & \textit{inernal vertex} \end{cases}$$

Rule I_A reported false-positive result in the case where the developer uses ‘*if-else’* statement in place of the switch statement. The Case 2 reports one such a case. To cope with identified Case 2, rule I_A is updated and named as I_B.

The rule I_B states that set of child nodes of each vertex (v_i) in the graph of *'if statement'* (G_{if}) must contain at least one *'else node'* (v_{else}). Hence, from the DPP_1 updated version of the first rule is:

RuleI_B:

$$\forall v_i \in G_{if} \exists\ v_{else}$$

where

v_{inter} = all intermediate nodes of a graph

Rule II:

The second rule for switch statement states that a vertex set (V) in the graph of *'switch statement'* (G_{switch}) must contain a *'default node'* ($v_{default}$). This node represents the default condition of the switch statement.

$$\forall v_i \in G_{switch} \exists\ v_{default}$$

Rule III:

According to the defensive programming principle DPP_2, input values should not be passed directly to PHP functions such as exec, system, passthru, require, requireonce, include and includeonce. It defines that in the graph of input value (G_{iv}), there should not exist any direct link from the node of input value (v_{iv}) to the node of system function (v_{sf}).

$$\forall v_{iv}, v_{sf} \in G_{iv} | \nexists\ v_{iv} \rightarrow v_{sys}$$

where V_{iv} and V_{sf} are the set of vertices related to input values (v_{iv}) and system function (v_{sys}), respectively. These sets are defined as:

$$V_{iv} = \{v_{iv} \in G_{iv} | \exists\ v_{iv1} \rightarrow v_{iv2}\}$$
$$V_{sf} = \{v_{sys}, v_{inc}, v_{req}, v_{inc\_o}, v_{req\_o}\}$$

$v_{iv1} \rightarrow v_{iv2}$ denotes a direct link from v_{iv1} to v_{iv2} in the graph of the input value. The vertices v_{exec}, v_{sys}, v_{pass}, v_{inc}, v_{inc_o}, v_{req} and v_{req_o} are related to exec, system, passthru, require, requireonce, include and includeonce, respectively. Generated rules are summarized in Table 3.

To improve the significance of the developed system and make it interactive, generated rule set is transformed into a structured form using XML Queries (XQuery). The developed XQueries are stored in the database to build the repository. It helps the security experts and the team leaders of the particular organization to train the system according to its rules.

4.4 Snippets Extraction and Parsing

The source code of web pages is semi-structured wherein all the keyword follows a nested structure. Hence, the code is structured as labeled ordered rooted trees through XML because it is more structured. The script from the source code is extracted. Extracted code snippets are tokenized to develop the rooted tree. Tokenization is the process of splitting code snippets into predefined language tags and keywords.

Tokenization is carried out through the 'Zend engine' lexical scanner. It returns an array of identified tokens. Each token in the array is either a single character (*i.e., <, >,;,., !, etc…*) or an array having three elements. PHP provides it as predefined function 'token_get_all()'. These tokens are stored in volatile memory for generating the graph.

4.5 Database Storage

Tokens of extracted code snippets are transformed into XML file, and database is created through BaseX. BaseX is a light-weight XML database management system which may be embedded as a library in detection engine. The space complexity to store a graph having n number of nodes and m number of edges is O (n + m) [25]. Snippets' representation of XML for Case 1 is shown in Fig. 5. Similarly, the graph for all cases is generated, saved as an XML file and a database is created.

Table 3 Defensive programming principles and related rules

Principle no.	Defensive programming principles	Rules
DPP_1	Handle Unexpected Conditions	I_A. Each vertex (v_i) the in the graph of 'if statement' (G_{if}) should contain out-degree either equal to 0 or 2
		I_B Set of child nodes of each intermediate vertex (v_i) in the graph of 'if statement' (G_{if}) must contain at least one 'else node'(v_{else})
		II. A vertex set (V) in the graph of 'switch statement' (G_{switch}) must contains a 'default' node
DPP_2	Process external systems data properly	III. In the graph of Input value (G_{iv}), there should not exist any direct link from the node of input value (v_{iv}) to the node of system function (v_{sf})

```
<graphml>
    <graph edgedefault="directed">
      <node id="root" />
      <node id="if1" />
      <node id="if11"/>
      <node id="else12"/>
      <node id="if121"/>
      <node id="if1211"/>
      <node id="else1212"/>
      <edge source="root" target="if1"/>
      <edge source="if1" target="if11"/>
      <edge source="if1" target="else12"/>
      <edge source="else12" target="if121" <edge
      source="if121" target="if1211" <edge
      source="if121" target="else1212"
      directed="false"/>
    </graph>
</graphml>
```

Fig. 5 GraphML representation of graph of Case 1

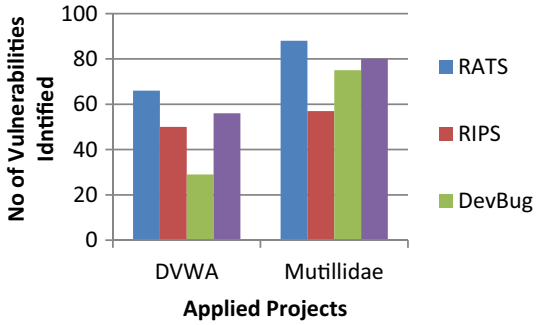

Fig. 6 Performance of evaluated techniques against two projects and one applied dataset

4.6 Testing and Alert Reporting Module

To visualize and tag the code vulnerabilities, a report is generated through applying the XQueries onto the created XML databases of extracted source code. XQuery yields the two results: True value indicates that code is secure and free from vulnerability, and the false value indicates that coding rules are violated, and the code suffers from code vulnerability.

5 Result Analysis and Discussion

Evaluation is the process of assessment to identify the performance and significance of the developed systems. As a result, this paper includes the result analysis against two datasets to compare the developed interactive system with three existing code analysis techniques such as RATS, RIPS and DevBug.

The first dataset is the real-time one which is suitable to identify the detection capabilities of developed systems. Second is labeled dataset which is suitable to evaluate the performance statics.

5.1 Analysis Through Real-Time Dataset

The real-time dataset such as online Web sites and web application projects is applied to test the system against real events. Initially developed interactive system and three existing scanning techniques are separately evaluated on two open-source projects as real-time dataset:

- *DVWA-project*
- *Mutillidae-project.*

It is appropriate to evaluate the developed system as it includes real web application and real web-based attacks. The total number of vulnerabilities identified (both correctly identified and incorrectly identified) by developed interactive system and three evaluated scanning techniques against the DVWA-project and Mutillidae-project is shown in Fig. 6.

The real-time dataset by definition is unlabeled which is inappropriate to identify the performance statics such as precision, recall, accuracy and false-positive rate (FPR). Hence, further exhaustive performance evaluation is carried out from labeled dataset to evaluate the remaining parameters.

5.2 Analysis Through Labeled Dataset

In labeled dataset, each available string is pointed as vulnerable or invulnerable labels. Also, it includes the non-attack events which are used to identify the false-positive cases and to help to differentiate among attacks and legitimate events. The labeled dataset is assembled through integrating the existing vulnerable code snippets available at the publicly available standard source, shown in Table 4.

These sources consisted of distinct code snippets of different types of code attacks, wherein labeled dataset consisted of 142 distinct vulnerable code snippets of 7 attacks and 130 distinct types of invulnerable code snippets. One of the benefits of creating labeled dataset is to evaluate the scanning techniques concerning false-negative and false-positive cases.

For cognitive performance analysis, remaining performance metric of developed interactive system and three existing scanning techniques are computed from assembled labeled dataset, and results are compared as shown in Fig. 7.

One effective way to compare the result of developed interactive system with three scanning techniques is to arrange the results concerning computed performance metrics as shown in Fig. 8. Its insight that developed interactive system detects code vulnerabilities more accurately rather than others with better precision and recall. False-positive rate of the developed system is least in comparison with DevBug and RIPS techniques.

However, developed interactive system causes little bit more false-positive rate rather than RATs. However, RATs

Table 4 Sources of labeled datasets

Sr. no.	Source	Description
1.	CVE [26]	It is a vulnerability naming scheme used as a dictionary of unique, common names for publicly known software flaws
2.	NIST National Vulnerability Database (NVD) [27]	It is a repository of standard-based vulnerability management data representation developed by the US government
3.	Syhunt Vulnerable PHP Code [28]	It is a hybrid multi-language web application security assessment suite that scans web application flaws from a hacker's perspective
4.	OWASP Filter Evasion Cheat Sheet [29]	It provides a guide to assist in Cross Site Scripting testing with the example

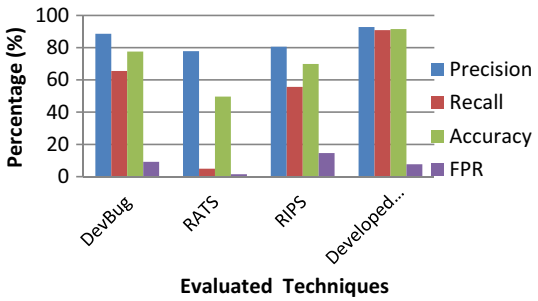

Fig. 7 Comparison of evaluated techniques through performance metric

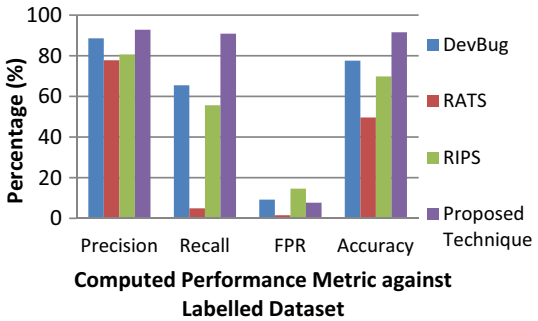

Fig. 8 Contrast in the performance metric of evaluated techniques

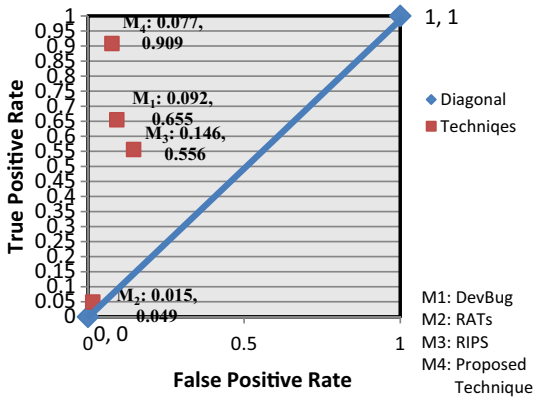

Fig. 9 ROC curve for measuring and comparing the developed system with existing evaluated scanning techniques

generate minimum true-positive rate which indicates that developed system is better and accurate than RATs. To precisely analyze the accuracy, ROC curve is generated.

ROC curve is plotted to analyze the performance through making the trade-off between hit rates/TPR and false alarm rates/FPR. It identifies the consequence of positives in the system through comparing the TPR and FPR. The point lies toward the top leftmost corner (1, 0) which indicates the best result in ROC curve.

Once four discrete techniques are applied to a labeled dataset, it yields a separate confusion matrix for each technique. Thus, single technique produced only a single point in the ROC space, represented in Fig. 9.

ROC curve prefers linear interpolation that yields an overly optimistic estimate of performance. Linear interpolation of points in ROC curve as shown in Fig. 9 yields that the point (0.077, 0.909) presents the optimistic performance because it lies in the upper left corner of the graph.

The result of the developed interactive system is coming out optimum among evaluated scanning techniques which may increase through adding up new rules according to the defensive programming concepts.

The performance of the developed system is increased along with the upgradations in rule sets which is shown in Fig. 10. Rule-set1 contains a rule for the conditional statement that is 'Out-degree of each vertex (v_i) in web graph of "if statement" (G_{if}) should be an even number.' Later it is modified due to the more FP and FN cases. TPR, FPR, accuracy and precision of developed interactive system at different rule set are also calculated and shown in Fig. 10.

The increase in result with adding up of defensive programming concepts depicts that result may be fine through accumulating more rules. ROC curve is plotted to obtain the finest rule set among three and comparing the overall performance of all evaluated techniques. Point in the upper leftmost corner of ROC curve provides the paramount result coming out for the final rule set, as shown in Fig. 11.

Point (0.077, 0.909) in ROC curve for the rule set lies toward the leftmost corner indicates the better rule set among

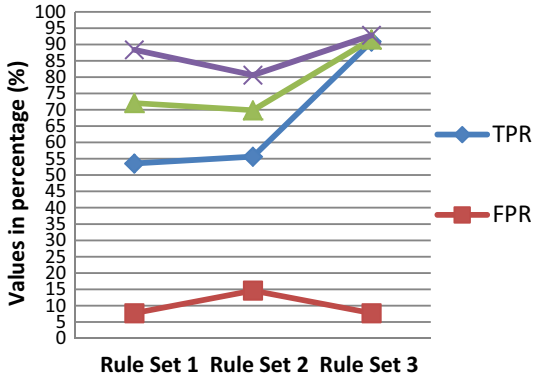

Fig. 10 Performance metric of developed system at different rule sets

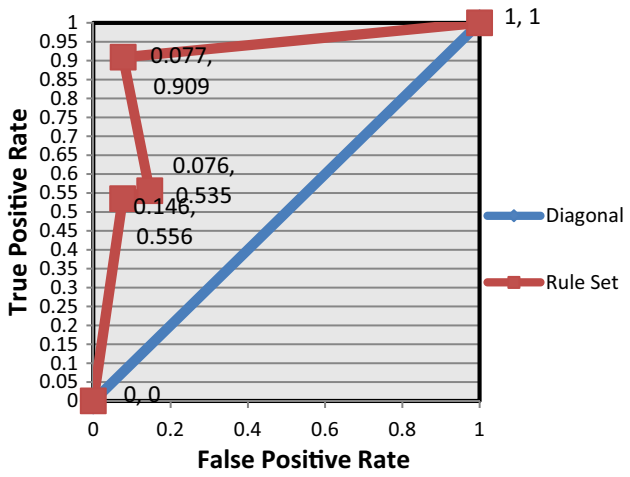

Fig. 11 ROC plot for developed system with three rule set

others. An interactive system is developed through the rule set that belongs to point (0.077, 0.909). It has the 91.54% accuracy which is best among evaluated scanning techniques.

An approach is developed for web programmers to scrutinize code vulnerabilities in the context of Secure Coding Standards. The developed technique is integrated into the interactive system, and then, the evaluation is done producing the 91.54% accurate results. Results are evaluated successfully on open-source standard real-time datasets as well as labeled dataset is built through merging the strings available on the basis of standard sources.

6 Conclusion and Future Work

Secure Coding Standards have evolved into one of the most dynamic and powerful web application security principles to develop reliable web applications. The contribution of this work is to formalize a graph-based comprehensive approach to applying textual Secure Coding Standards for developing the bulletproof web applications.

An interactive static code analysis system is developed which helps out web developers to write secure code and also assists web programmers in mitigating code injection vulnerabilities. The percentage and contribution of false positives and false negatives are efficiently evaluated, which helps to improve the performance of the detection system. Secure code successfully prevents against injection vulnerabilities with high accuracy and low false alarm rate which is 91.54 and 7.69%, respectively.

The future work focuses on the study of wider experiments involving the large set of vulnerable scripts and integrating the developed code analysis approaches to make the effective solutions to detect these vulnerabilities on social networking environments. Further, the wide range of experiments involving a large number of rules in the context of Secure Coding Standards are transformed into XQuery to provide the effective solution for emerging code vulnerabilities.

References

1. Securosis: 2014 Open source development and application security survey analysis, (2014). [Online]. https://securosis.com/assets/library/reports/Securosis_OpenSourceSurvey_Analysis.pdf
2. Gaurav, S.K.; Angelos, D.K.; Vassilis, P.: Countering code-injection attacks with instruction-set. In: Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS' 03), New York, USA, pp. 272–280 (2003)
3. Hu, W. et al.: Secure and practical defense against code injection attacks using software dynamic translation. In: Proceedings of the 2nd International Conference on Virtual Execution Environments, New York, USA, pp. 2–12 (2006)
4. Huang, Y. W. et al.: Securing web application code by static analysis and runtime protection. In: Proceedings of the 13th International Conference on World Wide Web. ACM, New York, 2004, pp. 40–52 (2004)
5. Anderson, B.; Quist, D.; Lane, T.: Detecting code injection attacks in internet explorer. In: IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW), Munich, pp. 90–95 (2011)
6. Riley, R.; Jiang, X.; Xu, D.: An architectural approach to preventing code injection attacks. IEEE Trans. Depend. Secure Comput. **7**(4), 351–365 (2010)
7. Papagiannis, I.; Migliavacca, M.; Pietzuch, P.: PHP Aspis: Using partial taint tracking to protect against injection attacks. In: Proceedings of 2nd USENIX Conference on Web Application Development, USA, pp. 13–24 (2011)
8. Kneuss, E.: Static analysis for the PHP language. EPFL-cosmopolitan technical university, Europe, No. LARA-STUDENT-2010-001, (2010)
9. Jovanovic, N.; Kruegel, C.; Kirda, E.: Precise alias analysis for static detection of web application vulnerabilities. In: Proceedings of the Workshop on Programming Languages and Analysis for Security. ACM, New York, USA, pp. 27–36 (2006)
10. Jovanovic, N.; Kruegel, C.; Kirda, E.; Pixy: a static analysis tool for detecting web application vulnerabilities. In: IEEE Symposium on Security and Privacy (S&P'06). Berkeley/Oakland, CA , pp. 258–263 (2006)
11. Ryan: DevBug—PHP static code analysis. [Online]. http://www.devbug.co.uk/#

12. Secure Software Inc. Rough auditing tool for security (RATS)" [Online]. https://code.google.com/p/rough-auditing-tool-for-security/
13. Dahse, J.; Holz, T.: Simulation of built-in PHP features for precise static code analysis. In: Symposium on Network and Distributed System Security (NDSS'14), USA, pp. 23–26 (2014)
14. Woods, J.; McGuirk, J.: Defensive programming techniques. In: Pharmaceutical users software exchange (PhUSE) 2006, Dublin, Ireland, 1–5 October 2006
15. Zhu, J.: Supporting secure programming in web applications through interactive static analysis. J. Adv. Res. **5**(4), 449–462 (2014)
16. Victor, S.: Secure development of code, ACC 626 Term Paper, ISA/BIT Learning Centre, University of Waterloo, Canada, June (2013)
17. Lemos, M.: 8 defensive programming best practices to prevent breaking your sites - PHP classes blog [Online]. http://www.phpclasses.org/blog/post/65-8-defensive-programming-best-practices-to-prevent-breaking-your-sites.html
18. Seacord, R.C.; Martin, R.: MITRE CWE and CERT Secure Coding Standards. Software Engineering Institute. Carnegie Mellon University, Pittsburgh (2010)
19. OWASP Foundation: OWASP Secure Coding Practices Quick Reference Guide V2. [Online]. https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf (2010)
20. Yen, S.J.; Chen, A.L.P.: A graph-based approach for discovering various types of association rules. IEEE Trans. Knowl. Data Eng. **13**(5), 839–845 (2001)
21. Sharma, H.K.: Graph based approaches used in association rule mining. Dissertation, Thapar University Patiala, Punjab, India (2010)
22. Gerkis, A.: Obvious and not obvious code injection and evaluation. [Online]. http://php-security.org/2010/05/20/mops-submission-07-our-dynamic-php/ (2010)
23. Paxson, V.: Principles for building secure systems, EECS instructional and electronics support, University of berkeley [Online]. http://www.icir.org/vern/cs161-sp13/notes/1.31.principles.pdf
24. Belk, M., et al.: Fundamental practices for secure software development. A guide to the most effective secure development practices in use today [Online]. http://www.safecode.org/publication/SAFECode_Dev_Practices0211.pdf (2011)
25. Brandes, U.; et al.: Graph Markup Language (GraphML), pp. 517–541. Taylor & Francis Group, London (2013)
26. MITRE Corporation: Common Vulnerabilities and Exposures [Online]. https://cve.mitre.org/cve/data_sources.html
27. U.S. Department of Commerce: National Vulnerability Database. [Online]. https://nvd.nist.gov/home.cfm (2015)
28. Syhunt, Vulnerable PHP Code: [Online]. Available: http://www.syhunt.com/docwiki/index.php?n=Vulnerable.PHP (2013)
29. OWASP, XSS Filter Evasion Cheat Sheet. [Online]. https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet (2015)