Cross Site Scripting: Detection Approaches in Web Application

Abdalla Wasef Marashdih and Zarul Fitri Zaaba
School of Computer Sciences,
Universiti Sains Malaysia, 11800 Minden,
Pulau Pinang, Malaysia

***Abstract*—Web applications have become one of the standard platforms for service releases and representing information and data over the World Wide Web. Thus, security vulnerabilities headed to various type of attacks in web applications. Amongst those is Cross Site Scripting also known as XSS. XSS can be considered as one of the most popular type of threat in web security application. XSS occurs by injecting the malicious scripts into web application, and it can lead to significant violations at the site or for the user. This paper highlights the issues (i.e. security and vulnerability) in web application specifically in regards to XSS. In addition, the future direction of research within this domain is highlighted.**

***Keywords*—Web Application Security; Security; Software Security; Security Vulnerability; Cross Site Scripting; XSS; Genetic Algorithm; GA**

I. Introduction

Web applications are becoming more important and growing in number as indicated by web browsers being used by almost everyone. Web applications have entered all areas, either for leisure or work, to manage sensitive personal and financial information [1]. These web applications are always available from anywhere with an Internet connection, and they enable us to communicate and collaborate at a speed that was unthinkable just a few decades ago. However, the presence of security vulnerabilities of web application can steal private information (e.g., cookies and session) and perform other malicious operations, and thus limit the use of applications [2].

XSS vulnerability is among the top web application vulnerability according to OWASP top 10 vulnerabilities [4]. The vulnerabilities can lead to significant violations at the site or for the user by injecting malicious scripts to be accepted later by the user. However, if there is no validation on the input of the application, then the malicious code can steal sessions, cookies, or inject and show private data for the user [5,6].

XSS vulnerability is among the top web application vulnerability according to OWASP top 10 vulnerabilities [4]. The vulnerabilities can lead to significant violations at the site or for the user by injecting malicious

The focal point of the study is to investigate the problems, challenges, and approaches to detect XSS vulnerabilities. This paper summarizes the XSS vulnerability on web application. Section II discusses the concept of web application. Section III further explains web application security. Section IV describes web application vulnerability. Section V and VI narrow the discussion in regards to XSS and the detection approaches. Section VII highlights the related work that have been gathered. Section VIII is a discussion of related work and finally ending with conclusion and future works.

II. Web Application

A web application utilizes web and browser technologies to perform tasks over a network using a web browser [7]. The web applications are stored on the web servers, where all their data are stored. Thus, users do not need to spend extra time on hard drives for installation. Some of the popular technologies that help software developers create dynamically generated web pages are PHP, ASP.NET, and Java server pages (JSP) [8].

PHP is easy to use for learning and for building websites, whereas PERL syntax is difficult for beginners to handle. ASP.NET is a product of Microsoft, is only possible in a Windows machine, and is not free. By contrast, PHP is completely free and is an open source. JSP is slower than PHP because JSP libraries are often written for "correctness" and readability but not for performance. Python hosting is hard to find and expensive, while cheap PHP hosting is everywhere. While PHP can mix with HTML in their source code, Python cannot be mixed with HTML (because it needs a template library). Therefore, PHP is the most popular scripting language and is the most commonly used in web applications.

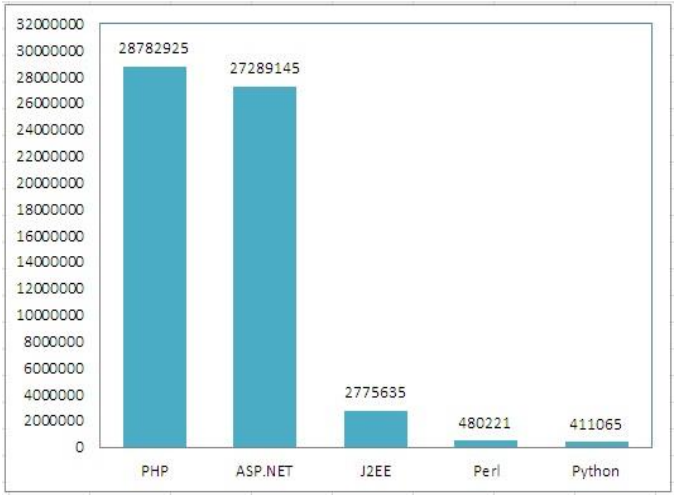

Fig. 1. Usage Statistics of Web Technologies [8]

Fig. 1 shows that PHP is used by more than 28,782,925 websites, thus making it the most used language. PHP is followed by ASP.NET. Statistics show that these two languages dominate all other languages. The number of PHP websites is greater than that of other websites using other web technologies. The user chooses the type of technology to build the website depending on his/her knowledge and the requirements of the facilities offered by the technologies. However, the lack of security of PHP web applications is caused by many programmers', because they do not have enough experience in securing their codes, which makes the applications flawed.

III. Web Application Security

Web application security is the practice of safeguarding confidential information stowed online from unlawful access and alteration. It is accomplished by imposing strict policies and practices [12]. In the software domain, security susceptibility is a flaw which could empower an attacker to compromise the veracity, accessibility, or confidentiality of a product. Several web applications set up on the Internet are subjected to security vulnerabilities. According to [13], more than 80% of the websites had experienced at least one grave of vulnerability. Web application security is expected to possess the security properties mentioned below:

- Input Authenticity: The user input should be authenticated before its use by the web application.
- State Integrity: The application state should be maintained unconstrained.
- Logic Exactness: The application logic should be implemented properly, as conceived by the developers.

A web application can be safeguarded through multiple means – for example, administering secure configuration, deploying a secure coding practice, conducting vulnerability evaluation, and employing a web application firewall. However, the total safeguard of the application is not possible. Web applications entail a defence-in-depth tactic to evade and alleviate security vulnerabilities. According to [14], the following is the threat model:

- The application is nonthreatening and hosted on a reliable and hardened infrastructure, i.e. the trusted computing base.
- The attacker hold the potential to regulate or influence the contents or the order of web requests directed towards the web application.

Sometimes, a web application might fail to hold the input validity property. In such a case, the attacker could initiate an XSS attack to thieve the session cookie of the victim, thereby causing an abuse of state integrity property. However, as mentioned earlier, an exhaustive safeguard of the application is impossible. The emphasis of this paper is on vulnerabilities in input validation, considering that input validity has been noted as the top security vulnerability for web applications (for example, XSS and SQL injection) [16]. In the next section, few of the major vulnerabilities of web applications are outlined.

IV. Web Application Vulnerability

Application susceptibility is described as a system imperfection or weakness which could be manipulated to compromise the application's security. Attackers are able to abuse the application susceptibility to trigger a cybercrime once they have noted a weakness or vulnerability which can be overpowered [16]. OWASP is a security community which emphasises on enhancing software security. In 2010, it came up with its annual report that noted the topmost threats and vulnerabilities in web application development; the report was updated in 2013 [4]. Here are the 10 key vulnerabilities identified by OWASP: injection, broken authentication and session management, XSS, insecure direct object references, security misconfiguration, sensitive data exposure, missing function level access control, cross-site request forgery (CSRF), use of components with known vulnerabilities, and invalidated redirects and forwards.

XSS is the most susceptible security threat according to the list [2,4]. The latest report was released in 2013 (Fig. 2), and there has been no new report after that. Veracode, an application security enterprise, has released its state of software security from 2013 until 2015. The report gives information about the number of vulnerabilities for every web technology [17]. A study covering the entire web applications noted that XSS accounts for 25 percent of the vulnerabilities [18].

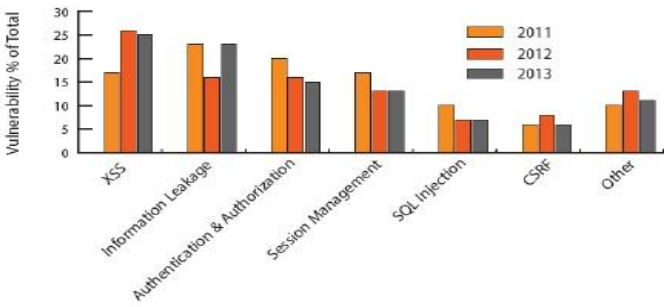

Fig. 2. 2011 vs 2012 vs 2013 Web Application Trends [18]

XSS offers an opening to the invader or hacker to enter the webserver database, mutilate websites, seize the web browser of a user remotely, and compel him/her to take an unfamiliar route [18]. Veracode's state of software security report emphasised on application development and scrutinised over 200,000 individual applications from the period October 2013 to March 2015 [16].

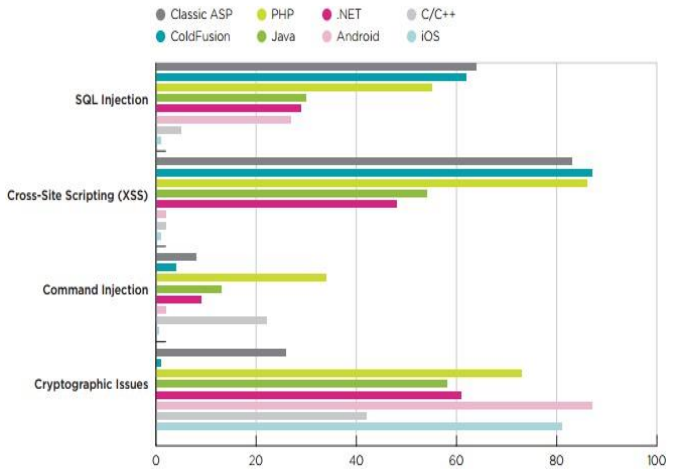

Fig. 3. Comparison of Critical Vulnerability Types [16]

As can be seen in Fig. 3, around 86 percent of PHP and ColdFusion applications comprised at least one XSS susceptibility. SQLi is a precarious and easy-to-abuse web application susceptibility. It comprises 62 percent of ColdFusion and 56 percent of PHP applications. ColdFusion is feeble when it comes to supporting OOP, and hence it might jeopardise input validity. Around 58 percent of PHP applications face issues with credentials management, whereas 73 percent of PHP applications involve cryptographic problems. According to [18], XSS vulnerability is the foremost susceptibility among the existing web applications. It is termed as the foremost vulnerability as it offers the basis for other kinds of attacks, including CSRF and session hijacking [19]. Moreover, XSS can inflict damage on website users as well as owners. It easily manipulates and is tough to alleviate. The next section deliberates and elucidates XSS susceptibility.

V. Cross Site Scripting (Xss)

XSS is termed as a key threat to web application security. Research is in progress to detect an effectual and convenient mode of analysing the source code of web applications and eliminating the threat. XSS is triggered by inserting spiteful scripts into the application, causing substantial abuses for the user or at the site. The malicious scripts are inserted at a place where an application admits user input; in case the input is not authenticated, the malicious code can thieve cookies or user accounts, or transfer private information [5,6]. These contaminated data might comprise portions of HTML code (for example, JavaScript) which may run into the page being attacked.

According to [19], there are four classes of XSS attacks: (i) stored (persistent); (ii) reflected (non-persistent); (iii) induced-XSS; and (iv) Dom-based XSS. The former two are the most commonplace, while the latter two are lesser known XSS attacks.

A. Stored (persistent) XSS

This susceptibility is triggered when the infused malicious code is forever stored on the victim servers. First, the attacker attempts to detect susceptibility in the web application so that he/she can inject the malicious script. Next, the attacker robs the confidential information of the users or inflicts other kinds of damages or risks [19].

The threat is more pronounced when this malicious script is forever stored on the server. The malicious script is affected when a user accesses the information by means of the web application, thus allowing the attacker access to it. According to [20], the persistent XSS is more menacing and devastating compared to other types of XSS vulnerabilities. Pure statistical analysis gives a false positive rate that is on the higher side.

B. Reflected (non-persistent) XSS

There is a difference between reflected XSS attacks and stored XSS attacks. Reflected XSS attacks manipulate website elements which reverberate clients' supplied data, including forms. The injected code is not located on the server. The attacker creates a crafted URL that involves a malicious script code, enticing the victim to believe that the URL is reliable [21]. The malicious links are delivered to victims through an email or by embedding the link into a web page which is located on some other server. The injected code is despatched to the web server of the attacker once the user clicks on the link, and the attack is then launched on the target browser.

C. DOM-based XSS

A Dom-based XSS attack is triggered on the client side [19]. DOM allows dynamic scripts, including JavaScript, to reference the document's components – for example, a session cookie or a form field. Such susceptibility could be triggered when an active content (for example, a JavaScript function) is altered by a specially created request, allowing a DOM element to be manipulated by an attacker.

D. Induced XSS

In an induced XSS attack when a web server has an HTTP response splitting susceptibility [7]. The attacker is able to abuse the HTTP header of the server's response in this case. Both Dom-based XSS and induced XSS attacks are uncommon but still mentioned here to ensure the classification is exhaustive.

In contemporary web applications, XSS is a security issue that is exploited the most often [16,17]. Persistent and non-persistent vulnerability can be observed on either server side or client side codes. However, DOM XSS is only noted in the client side [19]. Much research has concentrated on detecting XSS vulnerability [23,24,25,26,27]. However, research is still on to determine an effectual and suitable mode of analysing the source code and identifying the XSS susceptibility in web applications.

VI. Detection of XSS Vulnerability

Detecting susceptibility is a process of locating the weaknesses stated in the application's source code. Several web applications utilise the values furnished by users directly in the HTML exhibited in the browser [22]. This input can be fashioned to alter the contents of the web page that the victim can see, thus vesting the control with the attacker. The most standard approaches to spot vulnerabilities are categorised into dynamic analysis, static analysis, and hybrid analysis [15].

A. Static Analysis

Static analysis establishes the fundamental reason behind the security issue. It can detect errors in the initial stages of development and before the program is executed for the first time. The code coverage in static analysis is better compared to dynamic analysis. However, it is not much accurate as it is unable to access runtime information for the evaluated program [15]. Considering the nature of static analysis, approximations are carried out, which might lead to several false positives, i.e. reported vulnerabilities which are not truly vulnerabilities. [11] made a comparison of the different static analysis methods deployed to find out the various kinds of vulnerabilities from the program source code (lexical analysis, data flow analysis, symbolic execution, type inference, and constraint analysis). The data flow analysis approach is deployed to gather dynamic data from the source code. Static taint analysis is a special case of such type of analysis.

B. Dynamic Analysis

Dynamic analysis takes place when a security tool dynamically strikes the running application on the basis of thousands of identified vulnerabilities and attack designs [15]. In spite of utilising static analysis to locate vulnerabilities in different domains, this method is still ineffective as it has a tendency to come up with false positive and false negative outcomes. Dynamic analysis exposes vulnerabilities by examining the information attained during program implementation.

C. Hybrid Analysis

The hybrid approach combines static and dynamic analysis, wherein the dynamic analysis methods build up on the false alarms of the static analysis methods and offer accurate results. A technique to assist with security auditing and testing offers probabilistic alarms on possibly susceptible code statements.

[10] made a comparison of malware detection approaches on the basis of the dynamic, static, and hybrid analyses. The outcomes of the rates of detection were compared over a considerable number of malware families (Zbot, Security Shield, Smart HDD, Winwebsec, ZeroAccess, Harebot) [10]. The fully static approach is almost effectual in the majority of the circumstances based on API calls. The outcomes of the experiment suggest that a forthright hybrid approach might not be better than a fully dynamic detection or a fully static detection. Conversely, a static/dynamic methodology does not provide a steady improvement.

D. Genetic Algorithm

A genetic algorithm is a search heuristic which simulates the natural selection process. This heuristic (sometimes known as a metaheuristic) is usually used to come up with suitable solutions that can address search and optimisation-related issues. Genetic algorithms are founded on the evolutionary notions of natural selection and genetics. Thus, they signify an intelligent manipulation of a random search deployed to address optimisation issues [9]. The elementary genetic algorithm steps are converted into a pseudocode (Fig. 4).

```
population = generate_random_population();
        for(T in vulnerable paths) {
  while(T not covered AND attempt < max_try) {
        selection = select(population);
      offspring = crossover(selection);
      population = mutate(offspring);
        attempt = atempt + 1;
              }
            }|
```

Fig. 4. Genetic Algorithm Pseudocode [27]

1) Initial population: The most customary kind of encoding or representing chromosomes in genetic algorithms is the binary format. The genetic algorithm population is a suite of likely solutions for a problem.

2) Fitness function: This is the assessment of chromosomes as to how effective they are at addressing the issue. The closer the chromosome is to address the issue, the higher is its fitness value.

3) Selection: This stage intends to choose the fittest chromosome to reproduce as per certain selection techniques. A chromosome is chosen as per the fitness value to carry on in the next generation.

4) Crossover and mutation: This is an offspring produced by perturbing the chosen candidates using genetic operators – for example, mutation rate and crossover rate. The crossover operation combined two chromosomes to reproduce a new solution with better traits. On the other hand and according to specific mutation probability, the mutation operation occurs by altering the chromosome values.

The primary individual population is generated by the high-quality GA of the individuals. A solution is represented by each individual for the problem [3]. Table (1) presents a review of the approaches, areas of focus, and limitations of detecting XSS vulnerability.

TABLE I. REVIEW APPROACHES AND THEIR FACILITIES IN DETECTION XSS VULNERABILITIES

Article	Approach	Area on focus	Limitation
Shar and Tan [23]	Static Analysis (JAVA)	Detection of SQL Injection and XSS vulnerabilities.	High false positive rate in their detection results.
Toma and Islam [24]	Dynamic Analysis (Javascript)	Detection XSS vulnerability and applied it during the run.	They focused on some types of XSS vulnerability, and their results still not accurate.
Shar, et al. [25]	Hybrid Analysis (PHP)	Detection SQL injection and XSS vulnerabilities.	It is not accurate as full dynamic or static approach.
Avancini and Ceccato [26]	Genetic Algorithm + Static Analysis (PHP)	Detection XSS vulnerability in PHP Web applications.	They detect reflected XSS only.
Hydara et	Genetic Algorithm +	Detection XSS vulnerability in	The other language is still out side of

al. [27]	Static Analysis (JAVA)	JAVA.	their area.

As shown in Table 1, [23] use of static analysis still generates false negative and false positive results; and this finding is the main limitation of static analysis while it is executed before the run.

Hybrid analysis combines static and dynamic analyses as a better approach, but the combined approach was focused on to benefit from the two types of analyses (static and dynamic) [25]; nevertheless, it still has some problems in terms of the accuracy of its result, such as the training data.

In PHP, [26] detected one type of reflected XSS vulnerability. On the other hand, [27] proposed an approach based on static analysis with GA on Java web applications. Their approach combines the detection approach from [26] and the removal approach from [25]. [27] approach detects XSS vulnerabilities with significant results as compared with the approach of [25]. However, their approach is only available for Java web application.

VII. Related Work

While there are many approaches used to detect XSS vulnerability in the source code [23,24,25,26,27]. However, research is still on to determine an effectual and suitable mode of analysing the source code and identifying the XSS susceptibility in web applications. [23] proposed an approach to detect XSS vulnerability by using static analysis in Java web application. However, their approach still generates false negative and false positive results; and this finding is the main limitation of static analysis while it is executed before the run. On the other hand, [24] construct the JavaScript's call graph by using dynamic analysis, in a way to secure the client side of web application. Dynamic analysis used to find the limitations of the graphs art. Then, they evaluated their approach in regards of accuracy, and the results shown that their approach is acceptable.

[25] proposed attributes to check the input validation from SQL injection and XSS vulnerabilities based on hybrid analysis. They adopted static analysis to classify the nodes and dynamic analysis to find the vulnerable nodes. However, the static analysis still imprecise in classification of such nodes. The authors performed the experiments in six PHP web applications, and their results seems to be promised in the future. [26] presented an approach based on taint analysis with GAs as a method to improve taint analysis. They used taint analysis to find the vulnerable paths from the control flow of the program execution. Then, genetic algorithm defines security test cases by re-sorting the paths that enable the execution flow to traverse target paths. They employed the Pixy tool to report the control flow paths from the source code; these paths represent the target paths for genetic search. As their approach is only for detecting the reflected XSS vulnerability in the PHP web application.

[27] used static analysis with genetic algorithm, in a way to minimize the false positives rate in static analysis results. Their results minimize the false positive after embed genetic algorithm with static analysis, and they detect all vulnerabilities in JAVA web applications. Furthermore, the detection of vulnerabilities before run the program for the first time will minimize the threats on applications, rather than dynamic analysis which requires the actual run of the program, and that may leads to security vulnerabilities if they do not detect it quickly.

VIII. Discussion

We discussed the general approaches used to detect XSS vulnerabilities and differentiate their methods in detecting XSS vulnerabilities in Table 1. [26,27] used genetic algorithm with static analysis in a way to decrease the false positive rate in their results. [26] detected one type of reflected XSS vulnerability in PHP web applications using static analysis and GA. However, their approach will be argued because some paths in the source code cannot be executed. To detect XSS vulnerabilities without any false positive results, they need to remove the infeasible paths from the control flow graph. Once they remove the infeasible paths, they will detect the actual XSS vulnerability from the source code without any false positive in their results. [27] detected the three types of XSS vulnerability. While they detected all XSS vulnerabilities in Java source code, their approach still reveals false positive results. Therefore, the removal of the infeasible paths help to minimize the false positive results, because when the GA generator runs only on the feasible paths, it will be more fast and accurate to find the results. Therefore, to complete the approach of using GA with static analysis, the researchers should remove the infeasible paths from the control flow graph, in a way to minimize the false positive rate in their results.

IX. Conclusion

Web applications have been deployed to the public with unexpected security holes. The reason for these security holes is mainly the short time frame of this program's development. Although research on security programs is modern, effective solutions are highly demanded because of the importance of creating programs that are secure and less vulnerable to attacks. Cross-Site Scripting (XSS) vulnerability is one of the most common security problems in web applications. It can lead to the stealing of cookies and user accounts and to the transferring of private data if the input is not validated. While there are many studies have been conducted to address problems related to XSS vulnerability, but their results seems to be not efficient to address the problem as well. Static analysis still contains many false positive and the dynamic analysis still need to improve the accurateness of the results. However, the hybrid approach is not efficient as the fully static or dynamic approaches. On the other hand, genetic algorithm used to detect XSS vulnerability. Genetic algorithm successes to detect all XSS vulnerability in JAVA web application without any false positive results. However, when the researchers implement it in PHP, their results still contain many false positive results, because they did not remove the infeasible paths from the Control Flow Graph.

The future work should involve the removal stage of the infeasible paths from the control flow graph that will lead to minimize the false positive rate in their results and to detect all XSS vulnerability from the source code as well. Since GA has

proven to be effective in detection of XSS vulnerabilities, it can used for other web security vulnerabilities, such as (SQL Injection, insecure direct object references and cross-site request forgery).

REFERENCES

[1] Zhao, J. and Gong, R. "A New Framework of Security Vulnerabilities Detection in PHP Web Application," *Proc. Int. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing (Munich)*, pp 271-276, 2015.

[2] Gupta, M. Govil, M. and Singh, G. "Predicting Cross-Site Scripting (XSS) Security Vulnerabilities in Web Applications," *Proc. Int. Joint Conf. on Computer Science and Software Engineering (JCSSE)*, pp 162-167, 2015.

[3] Gupta, P. and Shinde, S. K. "Genetic algorithm technique used to detect intrusion detection," *Advances in Computing and Information Technology Anonymous Springer*, vol. 198, pp 122-131, 2011.

[4] OWASP 2013 Top-10 threats for web application security –2013 [Online] Available: https://www.owaspp.org/index.php/Top_10_2013-T10. [Accessed : 23/2/2016]

[5] Guo, X. Jin, S. and Zhang, Y. "XSS Vulnerability Detection Using Optimized Attack Vector Repertory," *Proc. Int. Conf. on Cyber-Enabled Distributed Computing and Knowledge*, pp 29-36, 2015.

[6] Dong, G. Zhang, Y. Wang, X. Wang, P. and Liu, L. "Detecting Cross Site Scripting Vulnerabilities Introduced by HTML5," *Proc. Int. Joint Conf. on Computer Science and Software Engineering (JCSSE)*, pp 319-323, 2014.

[7] Sadana, S. J. and Selam, N. "Analysis of Cross Site Scripting Attack," *Proc. International Journal of Engineering Research and Applications (IJERA)*, vol. 1, no 4, pp 1764-1773, 2011.

[8] Mishra, A. "Critical Comparison Of PHP And ASP.NET For Web Development - ASP.NET & PHP," *Proc. International Journal of Scientific & Technology Research*, pp 331-333, 2014.

[9] Duchene, F. Groz, R. Rawat, S. and Richier, J. "XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing," *IEEE Fifth Int. Conf. on Software Testing, Verification and Validation*, pp 815-817, 2012.

[10] Damodaran, A. Troia, F. D. Corrado, V. A. Austin, T. H. and Stamp, M. "A Comparison of Static, Dynamic, and Hybrid Analysis for Malware Detection," *Journal of Computer Virology and Hacking Techniques*, pp 1-12, 2015.

[11] Bingchang, L. Shi, L. and Cai, Z. "Software Vulnerability Discovery Techniques: A Survey," *Fourth Int. Conf. on Multimedia Information Networking and Security*, pp 152-156, 2012.

[12] Kumar, R. "Mitigating the authentication vulnerabilities in Web applications through security requirements," *Information and Communication Technologies (WICT)*, vol. 60, pp 651–663, 2011.

[13] Thankachan, A. Ramakrishnan, R. and Kalaiarasi, M. "Web application security vulnerabilities detection approaches: A systematic mapping study," *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp 1-6, 2015.

[14] Thankachan, A. Ramakrishnan, R. and Kalaiarasi, M. "A Survey and Vital Analysis of Various State of the Art Solutions for Web Application," Security *Information Communication and Embedded Systems*, pp 1-9, 2014.

[15] Gupta, M. K. Govil, M. C. and Singh, G. "Static Analysis Approaches to Detect SQL Injection and Cross Site Scripting Vulnerabilities in Web Applications: A Survey," *IEEE Int. Conf. on Recent Advances and Innovations in Engineering (ICRAIE-2014)*, pp 1-5, 2014.

[16] Veracode 2015b Application Security Vulnerability: Code Flaws, Insecure Code [Online] Available: http://www.veracode.com/security/application-vulnerability. [Accessed : 13/4/2016]

[17] Shanmugasundaram, G. "A study on removal techniques of Cross-Site Scripting from web applications," *Proc. Int. Conf. on Computation of Power, Energy, Information and Communication*, pp 0436-0442, 2015.

[18] Gupta, B. "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art," *National Institute of Technology Kurukshetra* (Kurukshetra, India), pp 1-19, 2015.

[19] Malviya, V. K. Saurav, S. and Gupta, A. "On Security Issues in Web Applications through Cross Site Scripting (XSS)," *Asia-Pacific Software Engineering Conf.*, pp 583-588, 2013.

[20] Li, Y. Wang, Z. and Guo, T. "Program Slicing Stored XSS Bugs in Web Application," *Fifth IEEE Int. Conf. on Theoretical Aspects of Software Engineering*, pp 191-194, 2011.

[21] Li, Y. Wang, Z. and Guo, T. "Reflected XSS Vulnerability Analysis," *International Research Journal of Computer Science and Information Systems (IRJCSIS),*vol. 2, pp 25-33, 2013.

[22] Fonseca, J. and Vieira, M. "A Practical Experience on the Impact of Plugins in Web Security," *IEEE 33rd Int. Symposium on Reliable Distributed Systems*, pp 21-30, 2014.

[23] Shar, L. K. and Tan, H. B. K. "Automated removal of cross site scripting vulnerabilities in web applications," *Inf. Softw. Technol.*, vol. 54, pp 467–478, 2012.

[24] Toma, T. R. and Islam, Md. S. "An Efficient Mechanism of Generating Call Graph for JavaScript using Dynamic Analysis in Web Application," *3rd Int. Conf. on Informatics, Electronics & Vison*, pp 1-6, 2014.

[25] Shar, L. S. Tan, H. B. K. and Briand, L. C. "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis," *Proc. of Int. Conf. on Software Engineering (ICSE '13)* IEEE Press, pp 642-651, 2013.

[26] Avancini, A. and Ceccato, M. "Towards Security Testing with Taint Analysis and Genetic Algorithms," *ICSE Workshop on Software Engineering for Secure Systems*, vol. 5, pp. 65–71, 2010.

[27] Hydara, A. I. Sultan, Md. Zulzalil, H. and Admodisastro, N. "An Approach for Cross-Site Scripting Detection and Removal Based on Genetic Algorithms," *Ninth Int. Conf. on Software Engineering Advances*, pp 227–232, 2014.