Indian Journal of Science and Technology, Vol 8(30), DOI: 10.17485/ijst/2015/v8i30/86055, November 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

# Cross-Site Scripting Detection Based on an Enhanced Genetic Algorithm

**Isatou Hydara*, Abu Bakar Md Sultan, Hazura Zulzalil and Novia Admodisastro**

Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, UPM, Serdang - 43400, Selangor, Malaysia; ishahydara@gmail.com

## Abstract

Software security vulnerabilities have led to many successful attacks on applications, especially web applications, on a daily basis. These attacks, including cross-site scripting, have caused damages for both web site owners and users. Cross-site scripting vulnerabilities are easy to exploit but difficult to mitigate. Many solutions have been proposed for their detection. However, the problem of cross-site scripting vulnerabilities present in web applications still persists. In this paper, we propose to explore an approach based on genetic algorithms that will be able to detect cross-site scripting vulnerabilities in the source code before an application is deployed. The proposed approach is, so far, only implemented and validated on Java-based web applications, although it can be implemented in other programming languages with slight modifications. Initial evaluations have indicated promising results.

**Keywords:** Cross-Site Scripting, Genetic Algorithm, Software Security, Vulnerability Detection

## 1. Introduction

Security testing is becoming an important part of software development due to the numerous attacks that software applications encounter on a daily basis. Due to their dynamic nature, i.e., the changing of their content in real-time as a result of user input or of being reloaded, web applications are the most exposed to security attacks, such as Cross-Site Scripting (XSS). Many research activities have been conducted to address problems related to XSS vulnerabilities since their discovery. According to this systematic review[1], most of the approaches focused on preventing XSS attacks[2–5] and a fewer number focus on detecting XSS vulnerabilities[6–9] in web applications during software security testing.

Cross-site scripting vulnerabilities are a security problem that occurs in web applications. They are among the most common and most serious security problems affecting web applications[10,11]. They are a type of injection problems[10] that enable malicious scripts to be injected into trusted web sites. This is a result of a failure to validate input from the web site users. What happens is either the web site fails to neutralize the user input or it does it incorrectly[11], thus, opening an avenue for a host of attacks.

Successful XSS can result in serious security violations for both the web site and the user. An attacker can inject a malicious code into where a web application accepts user input, and if the input is not validated, the code can steal cookies, transfer private information, hijack a user's account, manipulate the web content, cause denial of service, and many other malicious activities[10,11].

Cross-site scripting attacks are of three types namely reflected, stored and Document Object Model (DOM)-based[10,11]. Reflected XSS is executed by the victim's browser and occurs when the victim provides input to the web site. Stored XSS attacks store the malicious script in databases, message forums, comments fields, etc. of the attacked server. The malicious script is executed by visiting users thereby passing their privileges to the attacker. Both reflected and stored XSS vulnerabilities can be found on either client side or server side codes. On the other hand, DOM-based XSS vulnerabilities are found on

*Author for correspondence

the client side. Attackers are able to collect sensitive or important information from the user's computer.

In this paper, we propose a genetic algorithm-based approach for the detection of XSS vulnerabilities in web applications. The rest of the paper is organized as follows: Section 2 gives a brief review of related research conducted on the problems of XSS. Section 3 describes the proposed approach and in Section 4 we provide the preliminary results of the approach evaluation. Section 5 concludes the paper.

## 2. Related Work

Avancini and Ceccato[12] investigated the integration of taint analysis with genetic algorithms as an approach in software security testing of web applications. Their method showed some improvement in capturing XSS vulnerabilities and using them as a test case in security testing. They also implemented the integration of static taint analysis, genetic algorithm and constraint solving to automatically generate test cases that detect cross-site scripting vulnerabilities[13]. Their implementation focused only on reflected XSS in PHP code. The results seem promising. However, the fitness function of the genetic algorithm needs to be strengthened and the model tested in a wider range of software systems.

Duchene et al.[9] proposed an approach that combined model inference and evolutionary fuzzing to detect XSS vulnerabilities. Their approach used model inference to obtain a state model of the system under test and then used genetic algorithm to generate test input sequences, which enabled the detection of vulnerabilities. An explanation of their technique indicated it would prove successful when implemented on real world applications.

Lwin and Hee[14] proposed a solution that is able to remove XSS vulnerability from web applications before they can be exploited by hackers. The approach works in two phases. First, it uses static analysis to identify potential XSS vulnerabilities in application source codes. Secondly, it uses pattern matching techniques to come up with appropriate escaping mechanisms to prevent input values from causing script execution.

Researchers have also proposed tools that address the problem of XSS. BIXSAN[15] and L-WMxD[16] are two examples of such tools developed to tackle the XSS problem. BIXSAN filters out harmful HTML content and removes the non-static tags in the HTML page. It has been tested on many web browsers and shown to successfully prevent XSS attacks. L-WMxD, on the other hand, works on webmail services to detect the presence of XSS vulnerabilities. The tool has been tested on real-world webmail applications with some limitations and the results seem promising.

## 3. Proposed Approach

The solution being proposed uses a genetic algorithm-based approach in the detection of XSS vulnerabilities in web applications. The proposed solution is in two components. The first component involves converting the source codes of the applications to be tested to Control Flow Graphs (CFGs) using the White Box Testing techniques, where each node will represent a statement and each edge will represent the flow of data from node to node. A static analysis tool, PMD[17], is used in this task. The second component focuses on detecting the vulnerabilities in the source codes using an enhanced Genetic Algorithm (GA).

The main idea behind our approach is to formulate the security testing for XSS vulnerabilities as a search optimization problem. GAs has proved successful in the generation of minimal number test cases to uncover as many flows as possible in source codes[18]. In the same way, we can use GAs to detect as many XSS vulnerabilities as possible with a minimal number of test cases. The main contributions of this work are:

- The detection of XSS vulnerabilities in the source code of web applications using a GA approach.
- The automation of the XSS vulnerabilities detection approach.

### 3.1 Taint Analysis

Taint analysis is a White Box testing technique that tracks tainted or untainted status of variables throughout the control flow of an application and determines if a sensitive statement is used without validation[12,14,19]. For XSS vulnerabilities, a tainted variable refers to inputs from user or database, and print statements that append a string into a web page.

To perform a complete analysis of an application source code, we need to follow the White Box testing coverage criteria, such as statement coverage, branch coverage, or path coverage. We choose path coverage criterion because it encompasses the previous two. However, it is generally impossible to cover all paths of the source

code in testing. Therefore, we select a subset of the paths that interest us; the vulnerable paths whose execution will reveal XSS vulnerabilities. These are the paths where an input is executed without validation.

## 3.2 Enhanced Genetic Algorithm

Genetic Algorithms (GAs) are a subset of Evolutionary Algorithms (EAs), which are metaheuristic optimization algorithms based on population and inspired by biology[20]. They employ natural evolution mechanisms, such as mutation, crossover, natural selection, and survival of the fittest[21] to find optimal solutions in a search space. GAs is different from other EAs in that they have a crossover (recombination) operation and use binary coding in bits or bit-strings to represent a population[21].

Genetic algorithms have many capabilities; they have been used in many areas of computer science, such as software testing[22] and intrusion detection in network security[23,24] and in many other fields as well. In our proposed research, we believe similar techniques used in intrusion detection can be employed in the detection of cross-site scripting vulnerabilities in web applications. Experimentations need to be carried out to investigate this possibility.

Similarly, GAs can be used to generate source code with proper encoding that will replace parts of a source that is found to contain XSS vulnerabilities. For this part, similar methods used in test case generation with genetic algorithms in software testing can be employed.

Genetic algorithms have proven to be good solutions to many software engineering problems, since their discovery. Their successful use in software security testing[9,12,13] and intrusion detection systems[23,24] gives us the hope that they will be useful in detecting and removing XSS security vulnerabilities in Java-based Web applications.

Basically, a genetic algorithm consists of the following steps:

- Step 1: Create an initial population of candidate solutions
- Step 2: Compute the fitness value of each candidate
- Step 3: Select all the candidates that have their fitness values above or on a threshold
- Step 4: Make changes to each of the selected candidates using genetic operators, e.g., crossover and mutation
- Step 5: Repeat from step 2 until solution is reached or exit criteria is met.

The above steps are converted into a pseudocode, as shown in Figure 1.

```
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;
        }
    }
```

**Figure 1.** Genetic algorithm pseudocode.

### 3.2.1 Representation

The most common form of representing or encoding chromosomes in GA is using binary format. However, using binary format in XSS vulnerabilities detection would be very complex since the chromosomes represent patterns of real strings that serve as inputs while testing. Therefore, we decided to use natural numbers as the encoding scheme in our GA.

### 3.2.2 Initial Population

The GA population refers to the set of possible solutions for the problem to be solved. These possible solutions are generally referred to as chromosomes. In this work, the initial population is a set of test data that is generated according to the path coverage criterion, as stated in section 3.1. Since GAs deal with large search space, we will use a large population size of at least 100. After the initial population is selected, each individual chromosome is evaluated for possible inclusion in the next generation based on the fitness function.

### 3.2.3 Fitness Function

The fitness function is a measure of how good a chromosome is at solving the problem under consideration. So, a chromosome has a higher fitness value if it is closer to solving the problem. For our work, the fitness function evaluates the vulnerable paths that a test case needs to follow in order to reveal the presence of XSS vulnerabilities. It calculates the percentage of branches covered by an input traversing a vulnerable path and assigns a value. For example, if an input traverses all the branches of a vulnerable path, it means it has covered 100% of the branches and is assigned the value 1. If it traverses 70%, it is assigned the value 0.7 and so on. Hence, our fitness function is:

$$F(x) = ((Cpaths\% + Diff) * XSSp\%)/100 \quad (1)$$

- F(x): the fitness for an individual chromosome.
- Cpath%: the percentage of branches covered.
- Diff: the difference between the traversed and the targeted paths.
- XSSp%: the percentage of the XSS patterns file that the GA uses to cover a test path.

### 3.2.4 Selection

For iteration of the GA, a sample of chromosomes is selected for evaluation for possible inclusion in the next generation. There are different selection techniques for GA and in this work we choose the roulette wheel selection technique. It is a popular technique whereby the probability of selecting a chromosome for the next generation is proportional to its fitness function value. Two chromosomes (parents) are selected randomly based on their fitness function values and subjected to crossover and mutation methods in order to produce new chromosomes (offspring) for the next generation.

### 3.2.5 Crossover

In the crossover operation, as shown in Figure 2, two chromosomes are combined to form other chromosomes in the hope that the new ones will be better than the parent chromosomes. We use uniform crossover whereby the parent chromosomes contribute to the new offspring according to a specific crossover probability. We use a probability of 0.5 for the crossover operation. This is to give a fifty percent chance for half of the chromosomes to undergo changes while the other half proceeds to the next generation without undergoing any changes. This is because some chromosomes may already contain good genes and need not be changed.

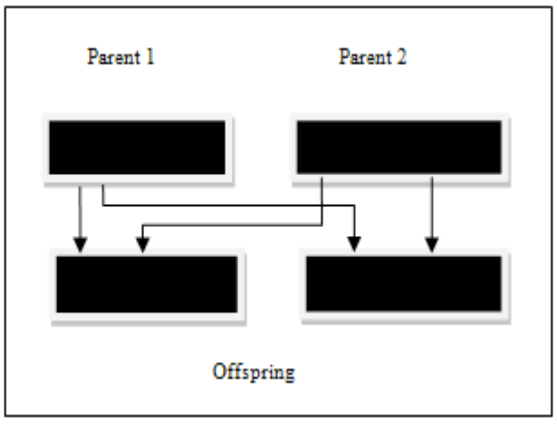


**Figure 2.** Crossover operation.

### 3.2.6 Mutation

The mutation operator is performed on the offspring after the crossover. It alters the chromosome values according to a specific mutation probability. It helps to guarantee that the entire search space is search, given enough time. It also helps to restore lost information or add more information to the population. A low mutation probability of 0.2 is used.

## 4. Evaluation

The above approach was implemented in a prototype and evaluated for its effectiveness and performance. The data used for the experiments were full source codes of large Java-based web applications. They were collected mainly from the Source Forge site (sourgeforge.net), as they are freely available.

All development was implemented with the Eclipse IDE using the Java Programming Language. The Java Genetic Algorithm Package (JGAP) engine[25] is integrated into the Eclipse IDE as a library for the easy usage of its Genetic algorithm operators. Java-based static analysis tool, PMD, is used to generate the CFG of the application files to be tested.

### 4.1 Experiment

The experiment to evaluate the proposed approach was conducted using the prototype tool developed for this purpose. The source code analyzer module takes Java files as input and uses PMD[17] to generate their CFGs. The XSS vulnerabilities detector module then uses the enhanced GA in the proposed approach to identify the vulnerable paths in the CFGs of the files under test. This module makes use of customized GA operators from the JGAP library[25]. It also used the XSS attack patterns referenced from OWASP[26].

Five Java-based open source web applications were used as test data to evaluate the effectiveness of the proposed approach. These applications were also used as test subjects in[14] to which some of the results of this research's experiments were compared. Table 1 shows the applications. Events and classifieds have also been used as benchmarks in related works[27,28]. Roomba has been used in[29]. PersonalBlog and JGossip have been used in[30].

**Table 1.** Test subjects

| Java Application | Lines of Code | Description |
|---|---|---|
| Events | 3818 | An event management system |
| Classifieds | 5745 | An online system for advertising and shopping |
| Roomba | 3438 | A room booking system for small to medium-sized hotels |
| PersonalBlog | 17149 | An online blogging system where users can publish their blogs |
| JGossip | 79685 | A large-scale forum application for discussing different topics |

## 4. 2 Results and Discussion

Five different experiments were conducted, one for each of the applications in the dataset. In each experiment the GA was run several times and the best results were collected for each. Taint analysis is first used to identify the possible paths of inputs in the applications source codes. Next the GA is used to generate inputs to follow these paths using the parameters in Table 2.

The results obtained from the XSS detection evaluation are recorded in Table 3. The number of vulnerabilities detected by our tool for each application is recorded under the column named: # Reported XSSV. These values also include the false positives, that is, the vulnerable statements reported by our tool that are not actually vulnerable upon manual inspection. The actual number of vulnerabilities in each application as benchmarked by[14] is recorded in the column named: # Actual XSSV.

Table 4 shows the comparison of our results with those reported in[14]. As shown in the columns named: # Reported XSSV and # Actual XSSV, our approach has reported less number of vulnerabilities but also detected the same number of actual vulnerabilities, respectively. This means that we have less number of false positives in our results. Hence, our approach has made improvement in terms of detection precision of XSS vulnerabilities in the tested applications.

In addition we discovered four DOM-based XSS on two applications in the test data: 3 vulnerabilities in JGossip and 1 vulnerability in PersonalBlog. We could not compare these results with those in Table 4 because that approach did not include DOM-based XSS vulnerabilities in their evaluation.

**Table 2.** GA parameters

| Parameter | Value |
|---|---|
| Population Size | 100 |
| Maximum no. of generations | 80 |
| Number of inputs per individual | 2 |
| Types of input | String |
| Crossover rate | 0.5 |
| Mutation rate | 0.2 |

**Table 3.** Experiment results

| Java Application | Lines of Code | # Reported XSSV | # Actual XSSV |
|---|---|---|---|
| Events | 3818 | **37** | **20** |
| Classifieds | 5745 | **64** | **19** |
| Roomba | 3438 | **146** | **129** |
| Personal Blog | 17149 | **3** | **1** |
| JGossip | 79685 | **41** | **28** |

**Table 4.** Comparison of results

| Java Application | Lines of Code | # Reported XSSV | | # Actual XSSV | |
|---|---|---|---|---|---|
| Events | 3818 | **37** | 81 | **20** | 20 |
| Classifieds | 5745 | **64** | 123 | **19** | 19 |
| Roomba | 3438 | **146** | 153 | **129** | 129 |
| PersonalBlog | 17149 | **3** | 84 | **1** | 1 |
| JGossip | 79685 | **41** | 312 | **28** | 28 |

# 5. Conclusion

In this paper, we presented a genetic algorithm-based approach for XSS detection in web applications. Cross-site scripting is a major security problem for web applications. It can lead to account or web site hijacking, loss of private information, and denial of service, all of which victimize the site users. Our proposed approach is an improvement based on previously proposed approaches. It uses better and improved GA operators to help in the detection of XSS vulnerabilities as well as including all the three types of XSS. A prototype tool has been developed to automate this process. Preliminary evaluation show promising results. We will continue to test the approach on real world Web applications with lager dataset and also improve the prototype tool.

## 6. Acknowledgement

This work was supported by the Malaysian Ministry of Education under the Fundamental Research Grant Scheme (FRGS 08-02-13-1368).

## 7. References

1. Hydara I, Sultan ABM, Zulzalil H, Admodisastro N. Current state of research on cross-site scripting (XSS) - A systematic literature review. Information and Software Technology. Elsevier B. V; 2015; 58:170–86. Available from: http://dx.doi.org/10.1016/j.infsof.2014.07.010
2. Sharma P, Johari R, Sarma SS. Integrated approach to prevent SQL injection attack and reflected cross site scripting attack. International Journal of System Assurance Engineering and Management. 2012 Sep; 26;3(4):343–51. Available from: http://link.springer.com/10.1007/s13198-012-0125-6
3. Sun Y, He D. Model checking for the defense against cross-site scripting attacks. International Conference on Computer Science and Service System IEEE; 2012. p. 2161–4. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6394855
4. Van Gundy M, Chen H. Noncespaces: Using randomization to defeat cross-site scripting attacks. Computer Law and Security Review. Elseiver Ltd; 2013 May; 31(4)612–28. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0167404811001477
5. Scholte T, Robertson W, Balzarotti D, Kirda E. Preventing input validation vulnerabilities in web applications through automated type analysis. IEEE 36th Annual Computer Software and Applications Conference. IEEE. 2012. p. 233–43. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6340148
6. Agosta G, Barenghi A, Parata A, Pelosi G. Automated security analysis of dynamic web applications through symbolic code execution. 9th International Conference on Information Technology - New Generations. IEEE. 2012. p. 189–94. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6209165
7. Al-amro H, El-qawasmeh E. Discovering security vulnerabilities and leaks in ASP. NET Websites. International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec). Kuala Lumpur: IEEE; 2012. p. 329–33.
8. Van-Acker S, Nikiforakis N, Desmet L, Joosen W, Piessens F. FlashOver: Automated discovery of cross-site scripting vulnerabilities in rich internet applications. ASIACCS '12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security; New York: ACM; 2012. p. 12–3.
9. Duchene F, Groz R, Rawat S, Richier J-L. XSS vulnerability detection using model inference assisted evolutionary fuzzing. IEEE 5th International Conference on Software Testing, Verification and Validation. IEEE; 2012. p. 815–7. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6200193
10. OWASP. Cross-site Scripting (XSS) - OWASP. OWASP. 2014. Available from: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
11. CWE. CWE-79: Improper neutralization of input during web page generation ('Cross-site Scripting') (2.5). The MITRE Corporation. 2014. Available from: http://cwe.mitre.org/data/definitions/79.html
12. Avancini A, Ceccato M. Towards security testing with taint analysis and genetic algorithms. Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems; Cape Town: ACM; 2010. p. 65–71.
13. Avancini A, Ceccato M. Security testing of web applications: A search-based approach for cross-site scripting vulnerabilities. IEEE 11th International Working Conference on Source Code Analysis and Manipulation. IEEE; 2011. p. 85–94. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6065200
14. Shar LK, Tan HBK. Automated removal of cross site scripting vulnerabilities in web applications. Information and Software Technology. Elsevier B. V; 2012 May; 54(5):467–78. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0950584911002503
15. Chandra SV, Selvakumar S. Bixsan: Browser independent XSS sanitizer for prevention of XSS attacks. ACM SIGSOFT Software Engineering Notes. 2011 Sep; 36(5):1–7. Available from: http://dl.acm.org/citation.cfm?doid=2020976.2020996
16. Tang Z, Zhu H, Cao Z, Zhao S. L-WMxD: Lexical based webmail XSS discoverer. IEEE Conference on Computer Communications Workshops INFOCOM WKSHPS. IEEE; 2011. p. 976–81.
17. PMD. PMD: Source Code Analyzer. Available from: http://pmd.sourceforge.net/
18. Berndt D, Fisher J, Johnson L, Pinglikar J, Watkins A, Management IP. Breeding software test cases with genetic algorithms. 36th Hawaii International Conference on System Sciences. Hawaii: IEEE; 2002. p. 1–10.
19. Shuai B, Li M, Li H, Zhang Q, Tang C. Software vulnerability detection using genetic algorithm and dynamic taint analysis. 3rd International Conference on Consumer Electronics, Communications and Networks (CECNet). IEEE; 2013 Nov. p. 589–93. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6703400
20. Weise T. Global Optimization Algorithms – Theory and Application. 2nd ed. 2009. p. 820.
21. Aljahdali SH, Ghiduk AS, El-Telbany M. The limitations of genetic algorithms in software testing. IEEE/ACS International Conference on Computer Systems and Applications (AICCSA). 2010 May. p. 1–7. Available

from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5586984
22. Srivastava PR, Kim T. Application of genetic algorithm in software testing. International Journal of Software Engineering and Its Applications. 2009; 3(4):87–96.
23. Bankovic Z, Stepanovic D, Bojanic S, Nieto-Taladriz O. Improving network security using genetic algorithm approach. Computer Electrical Engineering. 2007 Sep; 33(5-6):438–51. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0045790607000584
24. Islam AB, Al A, Azad MA, Alam MK, Alam MS. Security attack detection using Genetic Algorithm (GA) in policy based network. IEEE International Conference on Information and Communication Technology. 2007 Mar. p. 341–7. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4261430
25. JGAP. JGAP: Java Genetic Algorithms Package. 2014. Available from: http://jgap.sourceforge.net/
26. OWASP. XSS Filter Evasion Cheat Sheet - OWASP. 2015. Available from: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
27. Halfond WGJ, Orso A. AMNESIA: Analysis and monitoring for neutralizing SQL-injection attacks. Proceedings of 20th IEEE/ACM International Conference on Automobile Software Engineering; 2005. p. 174–83. Available from: http://doi.acm.org/10.1145/1101908.1101935
28. Halfond WGJ, Orso A, Manolios P. WASP: Protecting web applications using positive tainting and syntax-aware evaluation. IEEE Transaction on Software Engineering. 2008; 34(1):65–81.
29. Liu H, Kuan Tan HB. Testing input validation in web applications through automated model recovery. Journal of System Software. 2008; 81(2):222–33.
30. Martin M, Lam MS. Automatic generation of XSS and SQL injection attacks with goal-directed model checking. Proceedings of 17th Conference on Security Symposium; USENIX Association Berkeley, CA, USA. 2008. p. 31–43. Available from: https://www.usenix.org/legacy/event/sec08/tech/full_papers/martin/martin.pdf