An Approach for Cross-Site Scripting Detection and Removal Based on Genetic Algorithms

Isatou Hydara, Abu Bakar Md Sultan, Hazura Zulzalil, Novia Admodisastro
Dept. of Software Engineering and Information System
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 UPM Serdang, Selangor, Malaysia
ishahydara@gmail.com, abakar@upm.edu.my, hazura@upm.edu.my, novia@upm.edu.my

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 eliminate. 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 and remove cross-site scripting vulnerabilities from 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; vulnerability removal.

I. 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. Most of the approaches focused on preventing XSS attacks [1][2][3][4] or detecting XSS vulnerabilities [5][6][7][8] in web applications during software security testing. Few research activities have addressed their removal [9][10].

Software systems are usually deployed to the public with unexpected security holes. This is mainly due to the short time frame in which software are developed. Software project managers do not cater for security issues in their budgeting, scheduling and staffing their software development projects. Despite the fact that attention on software security is increasing, the progress on research for great solutions is slow**.** Notwithstanding that research on software security is very recent, effective solutions are in high demand due to the importance of creating software that is more secure and is less vulnerable to attacks.

In this paper, we propose a genetic algorithm-based approach for the detection and removal of XSS vulnerabilities in web applications. The rest of the paper is organized as follows: Section II gives a background of XSS and Genetic Algorithms. Section III reviews related research conducted on the problems of XSS. In Section IV, we describe our proposed approach and expected experimental results. Section V concludes the paper.

II. BACKGROUND

A. Cross-Site Scripting

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 [11][12]. They are a type of injection problems [12] 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 [11][12].

Cross-site scripting attacks are of three types namely reflected, stored and DOM (Document Object Model)-based [11][12]. 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 the client side. Attackers are able to collect sensitive or important information from the user's computer.

B. Genetic Algorithms

Genetic Algorithms (GAs) are a subset of Evolutionary Algorithms (EAs), which are metaheuristic optimization algorithms based on population and inspired by biology [13]. They employ natural evolution mechanisms, such as mutation, crossover, natural selection, and survival of the fittest [14] to find optimal solutions in a search space. GAs are 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 [14].

Genetic algorithms have many capabilities; they have been used in many areas of computer science, such as software testing [15] and intrusion detection in network security [16][17] 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. Experimentation 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 [8][18][19] and intrusion detection systems [16][17] gives us the hope that they will be useful in detecting and removing XSS security vulnerabilities in Java-based Web applications.

III. RELATED WORK

Avancini and Ceccato [19] 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 [18]. 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. [8] 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 [10] 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 [20] and L-WMxD [21] 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.

IV. PROPOSED APPROACH

The solution being proposed uses a genetic algorithm-based approach in the detection and removal of XSS vulnerabilities in Web applications. The proposed solution is in three 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 [22], is used in this task. The second component focuses on detecting the vulnerabilities in the source codes whiles the third component concentrates on their removal.

The main idea behind our approach is to formulate the security testing for XSS vulnerabilities as a search optimization problem. GAs have proved successful in the generation of minimal number test cases to uncover as many flows as possible in source codes [23]. 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 removal of detected XSS vulnerabilities from the source code of web applications
- The automation of the XSS vulnerabilities detection and removal approach

A. 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 [10][19][24]. 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.

B. The Genetic Algorithm

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 below.

```
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.

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.

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 IV A. 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) 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

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

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

4) Selection

For each 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 their fitness function values and subjected to crossover and mutation methods in order to produce new chromosomes (offspring) for the next generation.

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.

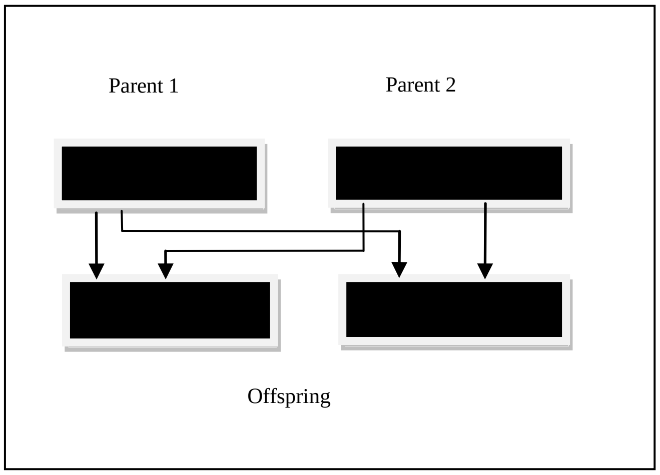


Figure 2. Crossover

6) Mutation

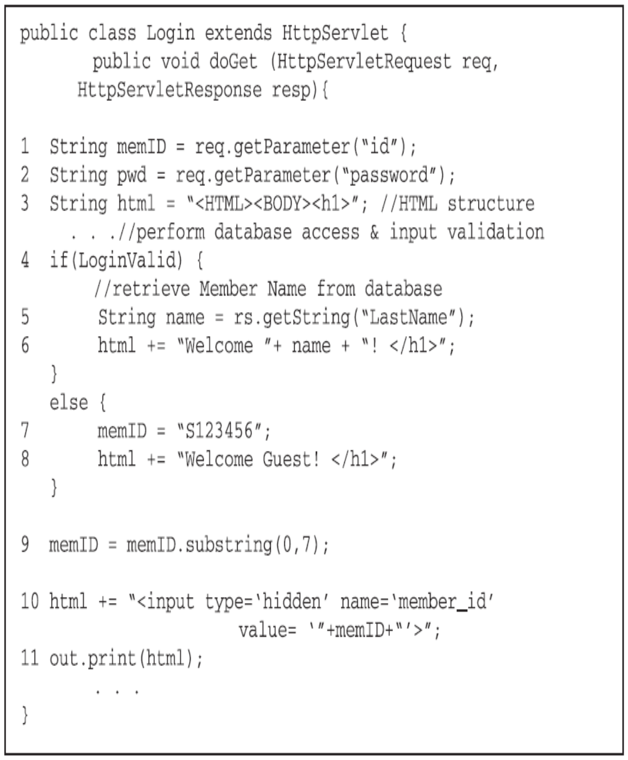
The mutation operator is performed on the offspring after the crossover. It alters the chromosome values according to a specific mutation probability. It helps to guarantee that the entire search space is search, given enough time. It also helps to restore lost information or add more information to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

C. The cross-site scripting removal Stage

Once the XSS vulnerabilities are detected in the source code, the removal stage begins. The OWASP's ESAPI (Enterprise Security API) security mechanisms [25] are followed to remove the detected XSS vulnerabilities. The lines of code where the XSS vulnerabilities are located will be identified. Then, we determine which of the ESAPI escaping rules can be applied to replace those lines of code without compromising their functionality. Finally, we generate the secure codes of the escaping statements and put them in place of the vulnerable statements, using these ESAPI escaping rules:

- **Rule#1:** Use HTML entity escaping for the untrusted data referenced in an HTML element, for example, <body><div>htmlEscape(untrusted_data)</div></body>, where ‘‘htmlEscape()’’ is the HTML entity escaping method
- **Rule#2:** Use HTML attribute escaping for the untrusted data referenced as a value of a typical HTML attribute such as name and value, for example, <input value=‘htmlAttrEscape(untrusted_data)’>, where ‘‘htmlAttrEscape()’’ is the HTML attribute escaping method
- **Rule#3:** Use JavaScript escaping for the untrusted data referenced as a quoted data value in a JavaScript block or an eventhandler, for example, <bodyonload=‘‘x=‘javascriptEscape(untrusted_data)’’’>, where ‘‘javascriptEscape()’’ is the JavaScript escaping method
- **Rule#4:** Use CSS escaping for the untrusted data referenced as a value of a property in a CSS style, for example,<table style= ‘‘width: cssEscape(untrusted_data)’’>, where ‘‘cssEscape()’’ is the CSS escaping method
- **Rule#5:** Use URL escaping for the untrusted data referenced as a HTTP GET parameter value in a URL, for example, <a href=‘http://www.site.com?name=urlEscape(untrusted_data)’>, where ‘‘urlEscape()’’ is the URL escaping method

Figures 3 and 4 [10] present the encoding mechanism of ESAPI.



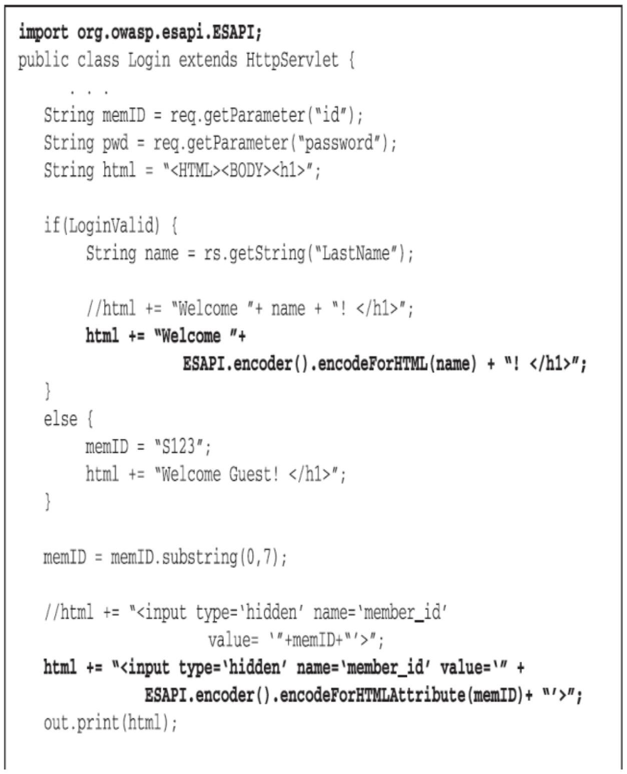
```
public class Login extends HttpServlet {
      public void doGet (HttpServletRequest req,
     HttpServletResponse resp){

  String memID = req.getParameter("id");
  String pwd = req.getParameter("password");
  String html = "<HTML><BODY><h1>"; //HTML structure
    . . .//perform database access & input validation
  if(LoginValid) {
      //retrieve Member Name from database
      String name = rs.getString("LastName");
      html += "Welcome "+ name + "! </h1>";
  }
  else {
      memID = "S123456";
      html += "Welcome Guest! </h1>";
  }

  memID = memID.substring(0,7);

  html += "<input type='hidden' name='member_id'
                     value= '"+memID+"'>";
  out.print(html);
      . . .
}
```

Figure 3. A potentially vulnerable code.



```
import org.owasp.esapi.ESAPI;
public class Login extends HttpServlet {
     . . .
   String memID = req.getParameter("id");
   String pwd = req.getParameter("password");
   String html = "<HTML><BODY><h1>";

   if(LoginValid) {
       String name = rs.getString("LastName");

       //html += "Welcome "+ name + "! </h1>";
       html += "Welcome "+
               ESAPI.encoder().encodeForHTML(name) + "! </h1>";
   }
   else {
       memID = "S123";
       html += "Welcome Guest! </h1>";
   }

   memID = memID.substring(0,7);

   //html += "<input type='hidden' name='member_id'
                  value= '"+memID+"'>";
   html += "<input type='hidden' name='member_id' value='" +
             ESAPI.encoder().encodeForHTMLAttribute(memID)+ "'>";
   out.print(html);
```

Figure 4. Code secured with ESAPI security API.

D. Evaluation

The above approach is being implemented in a prototype and will be evaluated for its effectiveness and performance. The data needed for the experiments on this research will be full source codes of Web applications. Source codes of complete open-source Java-based large Web applications will be used as experimentation data. These projects will be collected mainly from the Source Forge site [26], as they are freely available.

All development is being implemented with the Eclipse IDE using the Java Programming Language. The JGAP (Java Genetic Algorithm Package) engine [27] 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.

For this research, we used Java as the programming language on which to test our approach. Although most of the existing web sites are built with PHP, JavaScript and other similar scripting languages, there are many web sites built using Java Server Pages. These websites are also exposed to the cross-site scripting problem. Besides, most of the existing research works conducted on cross-site scripting were implemented using languages other than Java; hence, the focus on Java.

E. Expected Results

This research is expected to produce a new approach to detecting and removing XSS vulnerabilities in Java-based web applications. This approach will be an improvement based on the combination of two previously proposed approaches [10][19]. The first approach uses genetic algorithms to detect reflected XSS vulnerabilities only but does not remove them. The second approach is able to detect and remove both reflected and stored XSS vulnerabilities using pattern matching technique, but not DOM-based XSS. By combining them, this research will be able to use an enhanced genetic algorithm to detect and remove not only the same vulnerabilities but also DOM-based XSS vulnerabilities, which are not covered by both approaches. A Java-based tool has been developed to automate this approach. Furthermore, we expect this new approach to benefit web application developers by enabling them to easily test their source codes and get rid of many XSS vulnerabilities before deployment of their systems. This in turn will benefit any user who accesses such web systems by protecting them from malicious attacks.

F. Limitations

The limitations of this work are listed below:

1. Since this work makes use of static analysis, it also suffers its limitations. Therefore, the approach will fail to detect XSS vulnerabilities whose paths cannot be identified by static analysis in the source code.
2. The vulnerabilities removal module of the approach uses the OWASP ESAPI's escaping API only. Therefore, XSS vulnerabilities that are not defined in the context of this API are out of the scope of this work.
3. The approach is so far only implemented on Java Server Pages Web applications. However, the approach can be used with other programming languages.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a genetic algorithm-based approach for XSS detection and removal. 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 two previously proposed approaches. It uses better and improved GA operators to help in the detection and removal of XSS vulnerabilities as well as including all the three types of XSS. Our next step on this progressive work is to fully evaluate and validate the proposed approach. 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 and also improve the prototype tool. We expect our approach to be able to detect and remove the majority of XSS vulnerabilities, if not all, in real world Web applications.

ACKNOWLEDGMENT

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

REFERENCES

[1] P. Sharma, R. Johari, and S. S. Sarma, “Integrated approach to prevent SQL injection attack and reflected cross site scripting attack,” Int. J. Syst. Assur. Eng. Manag., vol. 3, no. 4, Sep. 2012, pp. 343–351.

[2] Y. Sun and D. He, “Model Checking for the Defense against Cross-Site Scripting Attacks,” in 2012 International Conference on Computer Science and Service System, 2012, pp. 2161–2164.

[3] M. Van Gundy and H. Chen, “Noncespaces: Using randomization to defeat cross-site scripting attacks,” Comput. Secur., vol. 31, no. 4, Jun. 2012, pp. 612–628.

[4] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, “Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis,” in 2012 IEEE 36th Annual Computer Software and Applications Conference, 2012, pp. 233–243.

[5] G. Agosta, A. Barenghi, A. Parata, and G. Pelosi, “Automated Security Analysis of Dynamic Web Applications through Symbolic Code Execution,” in 2012 Ninth International COnference on Information Technology - New Generations, 2012, pp. 189–194.

[6] H. Al-amro and E. El-qawasmeh, “Discovering Security Vulnerabilities And Leaks In ASP . NET Websites,” in Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on, 2012, pp. 329–333.

[7] S. Van-Acker, N. Nikiforakis, L. Desmet, W. Joosen, and F. Piessens, “FlashOver : Automated Discovery of Cross-site Scripting Vulnerabilities in Rich Internet Applications,” in ASIACCS ’12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, 2012, pp. 12–13.

[8] F. Duchene, R. Groz, S. Rawat, and J.-L. Richier, “XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing,” in 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, no. Itea 2, pp. 815–817.

[9] P. Bathia, B. R. Beerelli, and M. Laverdière, “Assisting Programmers Resolving Vulnerabilities in Java Web Applications,” in CCIST 2011: Communications in Computer and Information Science, vol. 133, no. 1, 2011, pp. 268–279.

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

[11] CWE, “CWE - CWE-79: Improper Neutralization of Input During Web Page Generation (’Cross-site Scripting') (2.5),” The MITRE Corporation. [Online]. Available: http://cwe.mitre.org/data/definitions/79.html. [retrieved: April, 2014]

[12] OWASP, “Cross-site Scripting (XSS) - OWASP,” OWASP. [Online]. Available: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS). [retrieved: March, 2014]

[13] T. Weise, Global Optimization Algorithms – Theory and Application –, 2nd Editio. 2009, p. 820.

[14] S. H. Aljahdali, A. S. Ghiduk, and M. El-Telbany, “The limitations of genetic algorithms in software testing,” ACS/IEEE Int. Conf. Comput. Syst. Appl. - AICCSA 2010, May 2010, pp. 1–7.

[15] P. R. Srivastava and T. Kim, “Application of Genetic Algorithm in Software Testing,” Intenational J. Softw. Eng. Its Appl., vol. 3, no. 4, Oct. 2009, pp. 87–96.

[16] Z. Banković, D. Stepanović, S. Bojanić, and O. Nieto-Taladriz, “Improving network security using genetic algorithm approach,” Comput. Electr. Eng., vol. 33, no. 5–6, Sep. 2007, pp. 438–451.

[17] A. B. . A. Al Islam, M. A. Azad, M. K. Alam, and M. S. Alam, “Security Attack Detection using Genetic Algorithm (GA) in Policy Based Network,” 2007 Int. Conf. Inf. Commun. Technol., Mar. 2007, pp. 341–347.

[18] A. Avancini and M. Ceccato, “Security Testing of Web Applications: A Search-Based Approach for Cross-Site Scripting Vulnerabilities,” in 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation, 2011, pp. 85–94.

[19] A. Avancini, F. Bruno, and K. Irst, “Towards Security Testing with Taint Analysis and Genetic Algorithms,” in ICSE Workshop on Software Engineering for Secure Systems, 2010, no. Section 5, May 2010, pp. 65–71.

[20] S. V. Chandra and S. Selvakumar, “Bixsan: Browser Independent XSS Sanitizer for prevention of XSS attacks,” ACM SIGSOFT Softw. Eng. Notes, vol. 36, no. 5, Sep. 2011, pp. 1–7.

[21] Z. Tang, H. Zhu, Z. Cao, and S. Zhao, “L-WMxD: Lexical based Webmail XSS Discoverer,” in 2011 IEEE Conference on Computer Communications Workshops INFOCOM WKSHPS, 2011, pp. 976–981.

[22] PMD, “PMD: Source Code Analyzer.” [Online]. Available: http://pmd.sourceforge.net/. [retrieved: August, 2014]

[23] D. Berndt, J. Fisher, L. Johnson, J. Pinglikar, A. Watkins, and I. P. Management, “Breeding Software Test Cases with Genetic Algorithms,” in 36th Hawaii International Conference on System Sciences, Jan. 2002, pp. 1-10.

[24] B. Shuai, M. Li, H. Li, Q. Zhang, and C. Tang, “Software vulnerability detection using genetic algorithm and dynamic taint analysis,” 2013 3rd Int. Conf. Consum. Electron. Commun. Networks, Nov. 2013, pp. 589–593.

[25] OWASP, “XSS (Cross Site Scripting) Prevention Cheat Sheet,” OWASP. [Online]. Available: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet. [retrieved: April, 2014]

[26] Sourceforge Community. [Online]. Available: sourceforge.net. [retrieved: September 2014]

[27] JGAP, “JGAP: Java Genetic Algorithms Package,” 2014. [Online]. Available: http://jgap.sourceforge.net/. [retrieved: August, 2014]