On the Automation of Vulnerabilities Fixing for Web Application

Kabir Umar, Abu Bakar Sultan, Hazura Zulzalil, Novia Admodisastro, and Mohd Taufik Abdullah
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia, UPM
Serdang, Selangor, Malaysia
emails: kbrumar@yahoo.com, {abakar, hazura, novia, taufik}@upm.edu.my

Abstract -- **Testing Web applications for detection and fixing of vulnerabilities has become an indispensable task in web applications' development process. This task often consumes a lot of time, efforts and other resources. The research community have devoted considerable amount of efforts to address this problem by proposing many techniques for automated vulnerabilities detection and fix generation for web application. Many of these techniques can reliably detect vulnerabilities and generate fix(es), which can be applied to the web application's code, by the developer, for possible fixing of the vulnerabilities. Hence, the actual code modifications that fix the vulnerabilities is not automated and has to be carried out manually. To the best of our knowledge, none of the existing automated techniques is able to do this, and hence the actual fixing of the vulnerabilities is left for the human developer to handle. In this paper, we propose a novel framework for automatic vulnerabilities fixing for web application. We mimic evolutionary idea and employ Evolutionary Programming to evolve web applications whose fitness is evaluated based on their ability to survive test attacks. The reliability of the resulting vulnerabilities-free web application can be further enhanced by co-evolving test sets with generations of web applications in which the fitness of test attack is evaluated based on its ability to break web applications.**

Keywords-Web application; Automated Vulnerabilities Fixing; Evolutionary Programming; SQL Injection.

I. INTRODUCTION

In recent years, web applications and services have gained utmost popularity and acceptance in various fields of human endeavor. Unfortunately, these applications are often deployed with varied degrees of vulnerabilities that are exploitable by hackers through many types of attacks, which can result in unauthorized and, often, harmful transactions with the application, as well as its' underlying database [1][2][3]. The severe consequence of web application attacks is, perhaps, the reason why detection and fixing of these vulnerabilities has been among top priorities of both research communities, governments and industries [4][5].

For more than a decade now, many techniques were proposed in the literature, by different researchers from around the globe, for automated detection and generation of fix for these vulnerabilities. Although many of the proposed techniques can reliably detect vulnerabilities in a subject web application and generate possible fix, unfortunately, the non-trivial task of actual modification of the source code of the web application for fixing the detected vulnerabilities has to be done manually by the human developer. To the best of our knowledge, none of the existing techniques proposed in the literature has achieved complete automation of vulnerabilities fixing, in which actual code modifications to fix vulnerabilities is done automatically. Additional draw back of the manual code modification by applying the auto-generated fix is that, sometimes the resulting application may behave in an unexpected manner [6], thus compelling the developer to undo the changes and revert to the original application. Although in many cases applying the auto-generated fix does fix the vulnerabilities, this can only be ascertained through conducting another testing of the modified application. In order words, there is no guarantee that applying the auto-generated fix to the application will surely fix the detected vulnerabilities, another testing has to be done. This creates an unnecessary additional cost overhead because software testing consumes time, efforts and other vital resources [7][8].

Producing very secure web application is an important goal of software engineering [7] because doing so will greatly reduce or completely prevent attacks on web application and therefore, prevent losses incurred by governments, organizations and individuals. In this paper, we propose a novel framework for complete automation of vulnerabilities fixing for web application. The framework will make the actual source code modifications necessary to fix vulnerabilities. We explore the widely applied ideas of evolutionary computing [9][10][11] and use Evolutionary Programming (EP) to evolve web applications whose fitness is evaluated based on their ability to defend themselves from test attacks and pass legitimate input tests. The actual source code modifications will be achieved through evolutionary operation of mutation.

Furthermore, the reliability of the resulting vulnerabilities-free web application (V-freeWA) will be enhanced by co-evolving test sets along with generations of web applications. The test sets comprises test attacks whose fitness is evaluated based on its ability to break web applications and legitimate input test whose fitness is evaluated based on its ability to fail web applications. This creates competitive co-evolution between the population of programs and the population of test sets similar to what happens in nature between preys and predators, such as Antelopes and Tigers [11]. The main goal is to go beyond automated vulnerabilities detection and fix generation, and

 ISBN: 978-1-61208-367-4

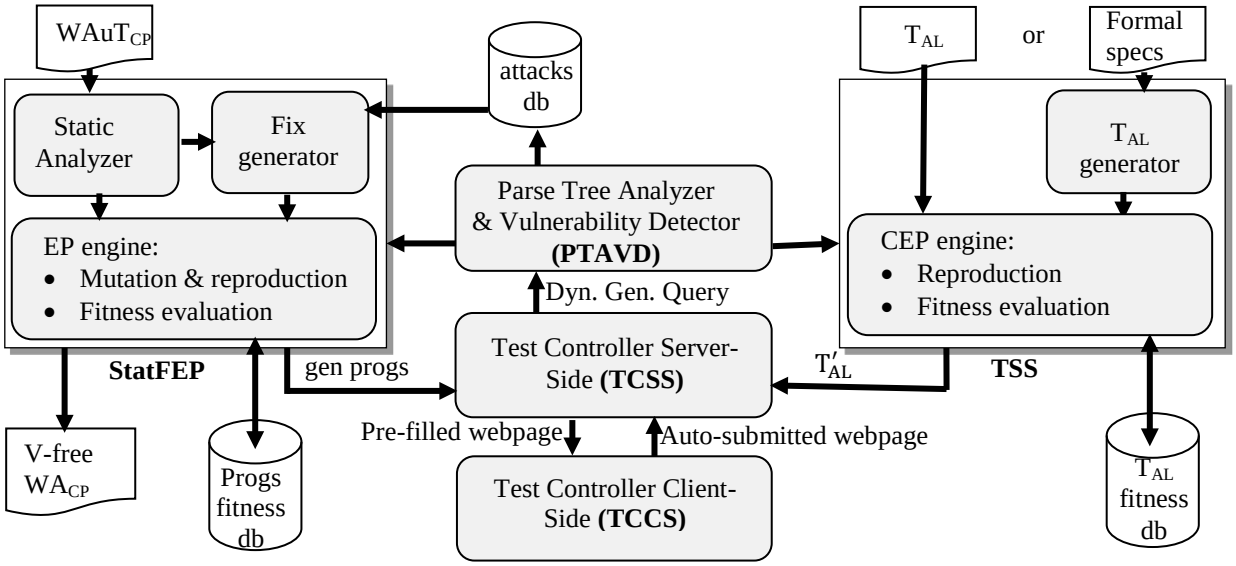

Figure 1. Framework for automated vulnerabilities fixing

have the actual task of code modification for vulnerabilities fixing fully automated. This challenging task is, perhaps, one of the long awaited automations in the field of software engineering. The remaining of this paper is organized as follows. Section II presents an overview of automated vulnerabilities detection and fix generation. Section III presents the proposed framework for automated vulnerabilities fixing. Section IV presents discussion, and Section V presents conclusion, future research work, as well as other possible research areas that can benefit from the framework.

II. OVERVIEW OF AUTOMATED VULNERABILITIES DETECTION AND FIX GENERATION

Many techniques were proposed in the literature for automated vulnerabilities detection and generation of possible fix(es) that can assist the developer to fix the vulnerable web application. Regrettably, the domain of web application vulnerabilities is very broad, diverse and exploitable by hackers through many types of attacks, thus making the task of automating vulnerabilities detection and fixing very challenging. The performance of the proposed techniques, in terms of vulnerabilities detection effectiveness, varies significantly from one category of vulnerabilities to another. While some of the techniques targeted specific category of vulnerabilities, others were proposed to handle considerably wide range of vulnerabilities, for example; Bau et al. [14] presented eight state-of-the-art black box vulnerabilities scanners that, on the average, each targets about six categories of vulnerabilities, namely, Cross Site Scripting (XSS), SQL Injection (SQLI), Cross Channel Scripting, Session Management, Cross Site Request Forgery (XSRF) and Information Leakage. Coincidentally, most of these vulnerabilities happen to be in the 2013 OWASP top 10 most dangerous web application security risk. Details about OWASP Top 10 project can be found in [4].

Interestingly, the techniques that addressed specific category of vulnerabilities also focus attention mostly within the OWASP Top 10 [4]; for example, [6][15][16] target SQL Injection (SQLI), [17][18] target Cross Site Scripting (XSS), [19] targets Buffer Overflow, [20] targets Configurations vulnerabilities, [21] targets Access Control vulnerabilities, [22] targets Session Management and Broken Authentication vulnerabilities, [23] targets Remote Code Execution, and [24] targets Logic vulnerabilities.

Although these techniques employs different software testing methods [7][8], such as static analysis, dynamic analysis, black box testing, penetration testing, mutation testing, search based testing, etc, and demonstrated diversity in their performance and effectiveness in vulnerabilities detection and fix generation, yet they almost have one thing in common, that is: “they were proposed to automate vulnerabilities detection and (in some cases) generate possible fix(es) in order to assist the developer to fix the vulnerable web application (under test)”. To the best of our knowledge, none of these techniques does the actual task of vulnerabilities fixing automatically.

III. AUTOMATED VULNERABILITIES FIXING

The proposed framework is for automatic vulnerabilities fixing for web applications. In this section, we present an overview of the components of the framework, highlight how they interacts, and highlight how fitness is evaluated for programs and test.

A. Components of the Framework

The framework comprises five main components, namely, Static analyzer, fix-generator, and EP engine (StatFEP), Test Set Selector (TSS), Test Controller Server

Side (TCSS), Test Controller Client Side (TCCS) and Parse Tree Analyzer and Vulnerability Detector (PTAVD). See Fig. 1 above.

1) StatFEP: Comprises three sub-components, namely, Static Analyzer, Fix-generator and EP engine. The static analyzer receives current page of web application under test ($WAuT_{CP}$) as input. It statically analyzes the webpage to determine all relevant database accessing points, sql generating statements, un-validated query-input variables, API method calls and parts of the webpage suitable for source code modification. The Fix-generator uses result of static analysis of $WAuT_{CP}$ and information from "attacks db" to generate smart fix(es) that targets likely vulnerabilities in $WAuT_{CP}$. The EP engine evolves population of $WAuT_{CP}$ as genetic programs (gen- progs) through single-parent reproduction and mutation operation. The mutation operation applies the auto-generated fix through source code modification guided by result of static analysis. In addition, the EP engine receives result of PTAVD to evaluate fitness of current individual program being tested and update "progs fitness db" accordingly. Lastly, this component monitors attainment of optimal solution and process stop condition.

2) TSS: The input to this component is test sets' search space T_{AL} or system's specifications of WAuT. T_{AL} is union of set of test attacks T_A and set of legitimate input tests T_L. The CEP-engine of TSS uses customized EP to evolve population of test sets T'_{AL} from the search space T_{AL}. Moreover, TSS uses result of PTAVD to evaluate fitness of individuals in T'_{AL} and update "T_{AL} fitness db" accordingly.

3) TCSS: This component receives current programs' individual (webpage) being tested from StatFEP, pre-fill the webpage with input data from current tests' individual in T'_{AL} (received from TSS), and forward the pre-filled webpage to TCCS. In addition, the component receives form submission (http POST request) from TCCS, intercepts and forward dynamically generated sql queries to PTAVD for analysis

4) TCCS: This component receives pre-filled webpage (current programs' individual being tested) from TCSS and auto-submit the page using http POST.

5) PTAVD: This component receives dynamically generated sql query from TCSS and performs syntax analysis for vulnerability detection by comparing syntax of current dynamic query with syntax of the same query generated using benign (verified legitimate) input from T_{AL}. Moreover, the component feeds information to StatFEP and TSS for appropriate fitness evaluations, and updates "attacks db" accordingly.

B. How it Works

To apply the proposed framework, a human tester might have confirmed the presence of at least one vulnerability in the Web Application under Test (WAuT) by subjecting it to test attacks (T_A) using appropriate testing method, such as applying tool for automated vulnerabilities detection. However, this is a very trivial and optional requirement. Nevertheless, since the framework is to auto-fix vulnerabilities, the presence of the vulnerabilities to fix could be confirmed first. Of course, if no vulnerabilities were detected in the WAuT, then there is no need to apply the framework.

Considering the nature of database access in web application, the StatFEP receives input of a page of the web application under test ($WAuT_{CP}$) at a time. The additional required input, received through TSS, is test sets T_{AL} comprising of test attacks and legitimate input tests. In addition, the TSS can receive system's specification of WAuT as input and generate T_{AL} accordingly. The $WAuT_{CP}$ is statically analyzed and represented as genetic program [9][12][13]. EP engine evolves the genetic programs. The CEP engine of TSS evolves the test sets T'_{AL}. The evolved programs in each generation are subjected to tests in T'_{AL}, by TCSS in collaboration with TCCS. During test execution, TCSS intercepts and forwards dynamic query to PTAVD for parse tree analysis, vulnerability detection, and functional correctness verification. Result of analysis is sent to EP engine for program's fitness evaluation, sent to CEP engine for test's fitness evaluation, and used to update "attacks db" accordingly. These fitness evaluations guide the evolution process to an optimal solution, i.e., V-freeWA_{CP}.

At each generation of programs, the operation of mutation, selection and reproduction is performed. Mutation applies the auto-generated fix to evolved programs. The auto-fix is generated by the fix generator module of StatFEP, with reference to information in "attacks db" and results of $WAuT_{CP}$ static analysis. Tournament is used to select programs with lowest fitness as parents. Single parent reproduction is employed to produce offspring [12]. Parents and offspring are combined to produce next generation. The optimal solution is found if an individual program has fitness of zero.

On the other hand, at each generation of test sets, only selection and reproduction is performed. Tournament is used to select test sets with highest fitness as parents. New test sets are randomly selected from test sets' search space to serve as offspring. Parents and offspring are combined to produce next generation of test sets.

C. Fitness Evaluation

1) Fitness of program: For easy reference in expressions, let $WAuT_{CP}$ be denoted by P_{or}, individual genetic program be denoted by P, and population of programs, consisting of n individuals P_1, P_2, …, P_n, be denoted by P_{pop}.

Fitness of P is evaluated based on semantic difference and syntax difference [9][28]. Semantic difference is a measure of how vulnerable, and how functionally incorrect, P is. Thus, we minimize semantic difference to ensure invulnerable and correct solution. On the other hand, syntax difference is a measure of how much P differs from P_{or} syntactically and structurally. Thus, we minimize syntax difference to ensure solution that respects the structure of P_{or}.

Definition 1: Given Q_d as the intercepted dynamic query generated by running P with a test set $t \epsilon T'_{AL}$, Q_b as the same query generated from benign input, Syn(Q) as the syntax tree of query Q, and $T'_{AL}(P)$ as the set of assertions

(consisting of pairs (Q_d, Q_b)) after execution of P on all test sets $t \in \mathrm{T'_{AL}}$, the semantic difference of P is defined as follows:

$$f_{sem}(P) = \sum_{(Q_d,Q_b) \in \mathrm{T'_{AL}}(P)} astPT(Q_d, Q_b) \quad (1)$$

$$\text{Where,} \quad astPT(Q_d, Q_b) = \begin{cases} 0 & Syn(Q_d) = Syn(Q_b) \\ 1 & Syn(Q_d) \neq Syn(Q_b) \end{cases}$$

Definition II: Given $N(P_{or})$ as the number of nodes in syntax tree of $\mathrm{P_{or}}$, $N(P)$ as the number of nodes in syntax tree of P, and δ as the allowable safe nodes difference, the syntax difference of P is defined as follows:

$$f_{syn}(P) = \begin{cases} N(P) - N(P_{or}) - \delta & if\ N(P) > N(P_{or}) + \delta \\ N(P_{or}) - N(P) - \delta & if\ N(P) < N(P_{or}) - \delta \\ 0 & otherwise \end{cases} \quad (2)$$

Equations 1 and 2 are combined to define the fitness function of P. The goal is to minimize the fitness function.

$$f(P) = f_{sem}(P) + f_{syn}(P) \quad (3)$$

2) Test sets fitness function: The test sets T'_{AL} is consist of test attacks and legitimate input tests. The fitness of test attack is evaluated based on its ability to break P, while the fitness of legitimate input test is evaluated based on its ability to fail P. We define an expression that evaluates the fitness of any test set $t \in T'_{AL}$, where t can be a test attack or legitimate input test. The goal is to maximize fitness of t.

Definition II1: Given $Q_{P_i(t)}$ as the intercepted dynamic query generated by running $\mathrm{P_i}$ with test set $t \in \mathrm{T'_{AL}}$, Q_b as the same query generated from benign input, Syn(Q) as the syntax tree of query Q, and $P_{pop}(t)$ as the set of assertions (consisting of pairs $(Q_{P_i(t)}, Q_b)$) after execution of all individuals $P_i \in P_{pop}$ with test set $t \in \mathrm{T'_{AL}}$, the fitness of t is defined as follows:

$$f(t) = \sum_{P_i \in P_{pop}} astPT\left(Q_{P_i(t)}, Q_b\right) \quad (4)$$

$$\text{Where,} \ astPT\left(Q_{P_i(t)}, Q_b\right) = \begin{cases} 0 & Syn(Q_{P_i(t)}) = Syn(Q_b) \\ 1 & Syn(Q_{P_i(t)}) \neq Syn(Q_b) \end{cases}$$

IV. DISCUSSION

The fitness of program is evaluated based on its ability to defend attacks and pass legitimate inputs. The evolutionary operation of mutation is applied to make actual source code modifications. For simplicity, we seed the population of first generation of the genetic programs with many duplicate copies of $\mathrm{WAuT_{CP}}$. This is because we assume that $\mathrm{WAuT_{CP}}$ is both structurally, semantically and syntactically near V-free$\mathrm{WA_{CP}}$, considering the famous assumption that, software developers do not write programs at random [25]. The problem that we may encounter with this kind of seeding is lack of diversity in the first generation [12]. However, we can easily resolve this problem and achieve the required diversity by randomly applying the evolutionary operation of mutation to all members of the first generation.

Though our framework seems simple, unfortunately, the task of evolving vulnerabilities-free program is not an easy one. For instance, as a result of mutation operations the EP may evolve a too short program or too large program compared to the $\mathrm{WAuT_{CP}}$. This situation is not always trivial. Moreover, since we assumed that $\mathrm{WAuT_{CP}}$ is structurally and syntactically near the optimal solution, we don't want to have a solution that is too different (structurally and syntactically) from $\mathrm{WAuT_{CP}}$, because it might not be easily understood and maintained by the developer. This problem is handled by adding program size parameter to our fitness function (equation 2). We penalize too short or too large program, thereby minimizing structural and syntactic difference between $\mathrm{WAuT_{CP}}$ and the optimal solution.

Another problem that may be induced by EP mutation is degradation of functional properties of the $\mathrm{WAuT_{CP}}$ due to effects of source code modifications. This could, invariably, impair the correctness of resulting solution. Fortunately, the formulation of our semantic difference (equation 1) and composition of test sets $\mathrm{T_{AL}}$ can effectively take care of the situation. During program's fitness evaluation, the test attacks in $\mathrm{T_{AL}}$ tries to expose residual vulnerabilities, while the legitimate test inputs in $\mathrm{T_{AL}}$ tries to re-affirm functional correctness. This way, the correctness of resulting optimal solution is guaranteed.

An important factor that directly affects the reliability of the optimal solution produced by the framework is the quality of the test sets $\mathrm{T_{AL}}$ in terms of effectiveness and precision in revealing all residual vulnerabilities in, and re-affirming functional correctness of, $\mathrm{WAuT_{CP}}$. The emergence of V-free$\mathrm{WA_{CP}}$ that is able to defend all test attacks and pass all legitimate input tests in $\mathrm{T_{AL}}$ may not guarantee 100% vulnerabilities-free and functionally correct web application. This is true if the quality of the test sets $\mathrm{T_{AL}}$ is poor. One way we can tackle this problem and improve the quality of the test is to have large set of $\mathrm{T_{AL}}$ comprising of many diverse tests that target wide range of possible vulnerabilities and functional correctness. Obviously, this approach is very likely to reveal all vulnerabilities in $\mathrm{WAuT_{CP}}$, while maintaining its functional correctness. Unfortunately, using large set of $\mathrm{T_{AL}}$ will induce very high computational cost of fitness evaluation.

A more feasible approach is to employ a co-evolutionary mechanism in which population (of reasonable number) of test sets $\mathrm{T'_{AL}}$ (consisting of test attacks and legitimate input tests) is co-evolved along with every generation of the programs. To achieve this, the TSS component of the framework adopted the very large set $\mathrm{T_{AL}}$ as a test sets' search space from which generations of $\mathrm{T'_{AL}}$ are evolved using customized EP (CEP engine module). At each generation of test sets $\mathrm{T'_{AL}}$, fitness of individual test attack is evaluated based on its ability to break programs while

 ISBN: 978-1-61208-367-4

fitness of legitimate input test is evaluated based on its ability to fail programs. A test attack that breaks more programs is ranked more fit. A legitimate input test that fails more programs is ranked more fit. Fitness of both test attack and legitimate input test is evaluated using the same fitness function (equation 4). The idea is that, when test attack breaks a program, then the program is vulnerable, and when legitimate input test fails a program, then the program is functionally incorrect. Thus, we seek to maximize the fitness function of T'_{AL}. This approach will create competitive co-evolution between population of programs and population of test sets similar to what happens in nature between preys and predators, such as Antelopes and Tigers, in which an Antelope (prey) is rewarded for its ability to escape Tiger's hunt (predator), whereas a Tiger (predator) is rewarded for its ability to catch an Antelope (prey). In our co-evolutionary scenario, the programs are the preys while the test sets T'_{AL} are the predators. Thus, an evolved program is rewarded for defending against tests' hunting while a test is rewarded for being able to break or fail programs. As the co-evolutionary process go through generations, the population of test sets will go (hunting) after population of programs, while the population of programs try to survive by means of fitness, reproduction and mutation.

Although the co-evolutionary process can lead to emergence of highly reliable V-freeWA_{CP}, along the way, the process may suffer from problem of mediocre stable state and loss of gradient [26], which can occur when both population of preys and predators seem to positively evolve at each generation in an infinite circular pattern without any real improvement. This happens if the fitness evaluation of members of the co-evolving populations (programs and tests) is done without remembering what happened in previous generations. Fortunately, we can adopt Archives technique [26] to handle this problem. At each generation some individuals of programs and tests are stored into "prog fitness db" and "T_{AL} fitness db" respectively. The fitness of current generation is then based on interaction with the old individuals in the Archive, thus enabling the co-evolutionary process to remember history of past generations.

V. CONCLUSION AND FUTURE WORK

In addressing the problem of resolving web application's attacks and exploitations, this paper proposed a novel framework for automating vulnerabilities fixing for web application. The current techniques proposed in the literature are only capable of automating vulnerabilities detection and fix-generation while leaving the task of actual vulnerabilities fixing predominantly manual. We combine ideas of software testing, parse tree analysis, and evolutionary computing in a novel framework to achieve complete automation of the task. We have also shown how reliability of the resulting vulnerabilities-free web application can be further enhanced through co-evolution. The novel framework incorporates functional testing to guarantee correctness of the resulting optimal solution. As we progresses in this on-going research, we are optimistic in revealing and reporting very useful contributions toward automating vulnerabilities fixing, as well as advances in the field of software engineering.

Obviously, research on complete automation of vulnerabilities fixing for web application is at very preliminary stage for two reasons. First, to the best of our knowledge, this paper is the first to address the task. Second, the domain of web application's vulnerabilities is very large, broad and diverse [1][2][4][5]. Hence, there is room for a lots of future research activities. At the moment, we are planning the following:

A. Because the domain of web application vulnerabilities is very large, broad and diverse, we intend to scope our research to case study of SQL Injection vulnerabilities (SQLIVs). Our scope is so chosen for obvious reasons. First, SQLI has been the world's most serious web application security risk since 2004, as shown by OWASP Top 10 project reports of 2004, 2007, 2010, and 2013 [4] and CWE/SANS Top 25 Most Dangerous Software Errors [5]. Second, we are quite optimistic that applying the framework to successfully automate fixing SQLIVs, through well planned and documented experiment, should be sufficient to suggest the applicability of the framework for fixing other web application vulnerabilities related to source code.

B. We are planning to build our first prototype implementation of the framework in Java programming language to auto-fix vulnerabilities in web application designed in Java Server Pages (JSP) with MySQL as the backend database. The choice of JSP with MySQL backend is due to the fact that, most web applications are built in JSP, and MySQL database is, perhaps, the most widely used database server for the web.

Our novel framework can be applied (with slight modifications where necessary) to other research areas.

A. The framework can be used to automate fixing vulnerabilities in other non-web based database systems, such as Java database applications, which are equally liable to SQL injection attacks [27].

B. The framework can be adapted to automate Networks Vulnerabilities fixing through simulation. The idea is to have real network under test (NuT) simulated and evolved through generations. Fitness is then evaluated by subjecting the simulated networks to test attacks T_A.

Finally, we hope that our novel framework can benefit research community and lead to further research activities.

AKNOWLEDGEMENT

We acknowledge efforts and contributions of various authors whose work has benefited our research. We also acknowledge that this research received support from the Fundamental Research Grant Scheme (FRGS/2/2013/ICT01/UPM/02/8) awarded by Malaysian Ministry of Education to the Faculty of Computer Science and Information Technology at Universiti Putra Malaysia.

REFERENCES

[1] D. Watson and U. K. H. Project, “Web application attacks,” Journal of Network Security, vol. 2007, iss. 10, Oct. 2007, pp. 10–14, doi:10.1016/S1353-4858(07)70094-6.

[2] D. Gollmann, “Securing web applications,” Information Security Technical Report, ELSEVIER, vol. 13, no. 1, Jan. 2008, pp. 1–9, doi:10.1016/j.istr.2008.02.002.

[3] A. Garg and S. Singh, “A review on web application security vulnerabilities,” International Journal of Advanced Research in Computer Science and Software Engineering, (IJARCSSE), vol. 3, no. 1, 2003, pp. 222–226.

[4] OWASP, "OWASP top 10 project," https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project , [retrieved: May 2014].

[5] CWE, “CWE-SANS top 25 most dangerous software errors,” Common Weakness Enumeration, Http://cwe.mitre.org/ top25/. (http://cwe.mitre. Org/top25/), [retrieved: May 2014].

[6] J. M. Chen and C. L. Wu, "An automated vulnerability scanner for injection attack based on injection point," Proc. IEEE International Computer Symposium, IEEE Press, Tainan, Dec. 2010, pp. 113-118, doi:10.1109/COMPSYM.2010.5685537.

[7] L. Luo, "Software testing techniques," Class Report for 17-939A, Institute for Software Research International, Carnegie Mellon University, USA. (http://mcahelpline.com/tutorials/testing/testing.pdf).

[8] J. Irena, “Software testing methods and techniques,” The IPSI BgD Transactions on Internet Research, 2008, internetjournals.net.

[9] A. Arcuri, "Evolutionary repair of faulty software," Journal of Applied Soft Computing, ELSEVIER, vol. 11, iss. 4, June. 2011, pp. 3494–3514, doi:10.1016/j.asoc.2011.01.023.

[10] A. Arcuri, "On search based software evolution," Proc. IEEE 1st International Symposium on Search Based Software Engineering, IEEE Press, Windsor, May. 2009, pp. 39–42, doi:10.1109/SSBSE.2009.12.

[11] A. Arcuri, "On the automation of fixing software bugs," Proc. ACM 30th International Conference on Software Engineering, ACM, Leipzig, Germany, May. 2008, pp. 1003-1006, doi:10.1145/1370175.1370223.

[12] A. Abraham, "Evolutionary computation: from Genetic Algorithms to Genetic Programming," in Genetic Systems Programming: Theory and Experiences, Ecological Studies 185, N. Nedjah and A. Abraham, Springer, 2005, pp. 1-20.

[13] R. G. S. ASTHANA, "Evolutionary Algorithms and Neural Networks," in Soft Computing and Intelligent Systems: Theory and Applications, A volume in Academic Press Series in Engineering, N. K. Sinha, M. M. Gupta and L. A. Zadeh, ELSEVIER Inc, 2000, pp. 111-136.

[14] J. Bau, E. Bursztein, D. Gupta and J. Mitchell, "State of the art: automated black-box web application vulnerability testing," Proc. IEEE Symp. Security and Privacy (SP), IEEE Press, Oakland, CA, May. 2010, pp. 332-345, doi:10.1109/SP.2010.27.

[15] F. Dysart and M. Sherriff, "Automated fix generator for SQL injection attacks," Proc. IEEE 19th International Symposium on Software Reliability Engineering, (ISSRE), IEEE Press, Seattle, WA, Nov. 2008, pp. 311-312, doi:10.1109/ISSRE.2008.44.

[16] Z. Djuric, “A black-box testing tool for detecting SQL injection vulnerabilities,” Proc. IEEE Second International Conference on Informatics and Applications, (ICIA), IEEE Press, Lodz, Sept. 2013, pp. 216-221, doi:10.1109/ICoIA.2013.6650259.

[17] B. Qu, B. Liang, S. Jiang and C. Ye, "Design of automatic vulnerability detection system for web application program," Proc. IEEE 4th International Conference on Software Engineering and Service Science, (ICSESS), IEEE Press, Beijing, May. 2013, pp. 89-92, doi:10.1109/ICSESS.2013.6615262.

[18] G. Wassermann and Z. Su, "Static detection of Cross-Site Scripting vulnerabilities," Proc. ACM/IEEE 30th International Conference on Software Engineering, (ICSE), ACM/IEEE, Leipzig, May. 2008, pp. 171-180, doi:10.1145/1368088.1368112.

[19] A. Smirnov and T. Chiueh, "Automatic patch generation for Buffer Overflow attacks," Proc. IEEE Third International Symposium on Information Assurance and Security, (IAS), IEEE Press, Manchester, Aug. 2007, pp. 165-170, doi:10.1109/IAS.2007.87.

[20] B. Eshete, A. Villafiorita, K. Weldemariam and M. Zulkernine, "Confeagle: Automated analysis of Configuration vulnerabilities in web applications," Proc. IEEE 7th International Conference on Software Security and Reliability, (SERE), IEEE Press, Gaithersburg, MD, June. 2013, pp. 188-197, doi:10.1109/SERE.2013.30.

[21] F. Gauthier and E. Merlo, "Fast detection of Access Control vulnerabilities in PHP applications," Proc. IEEE 19th Working Conference on Reverse Engineering, (WCRE), IEEE Press, Kingston, ON, Oct. 2012, pp. 247-256, doi:10.1109/WCRE.2012.34.

[22] D. Huluka and O. Popov, "Root cause analysis of Session Management and Broken Authentication vulnerabilities," IEEE World Congress on Internet Security, IEEE Press, Guelph, ON, June. 2012, pp. 82-86.

[23] Y. Zheng and X. Zhang, "Path sensitive static analysis of web applications for remote code execution vulnerability detection," Proc. IEEE 35th International Conference on Software Engineering, (ICSE), IEEE Press, San Francisco, CA, May. 2013, pp. 652-661, doi:10.1109/ICSE.2013.6606611.

[24] V. Felmetsger, L. Cavedon, C. Kruegel and G. Vigna, "Toward automated detection of logic vulnerabilities in web applications," Proc. ACM 19th USENIX conference on Security, ACM, Berkeley, CA, USA, 2010, pp. 10-10.

[25] R. A. DeMillo, R. J. Lipton and F. Sayward, "Hints on test data selection: help for the practicing programmer," Computer, vol. 11, iss. 4, IEEE Press, 2006, pp. 34–41, doi:10.1109/C-M.1978.218136.

[26] C. D. Rosin and R. K. Belew, "New methods for competitive coevolution," in Evolutionary Computation, vol. 5, iss. 1, MIT Press Cambridge, MA, USA, 1997, pp. 1-29.

[27] C. Zhou and P. Frankl, "JDAMA Java database application mutation analyser," ACM Journal of Software Testing, Verification & Reliability, vol. 21, iss. 3, Sept. 2011, pp. 241-263, doi:10.1002/stvr.462.

[28] W. B. Langdon, M. Harman and Y. Jia, “Efficient multi-objective higher order mutation testing with genetic programming,” ELSEVIER Journal of Systems and Software, vol. 83, iss. 12, Dec. 2010, pp. 2416–2430, doi:10.1016/j.jss.2010.07.027.

 ISBN: 978-1-61208-367-4