International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 5 | ISSN : 2456-3307

# Inducing Vulnerability Testing to Enhance Performance

**Dr. Selvakumar. S, Ph.D.** [1], **Dereje Regassa**[2], **Dr. S. Subburam, Ph.D.** [3]
[1]Department of Computer Science & Engineering, School of Electrical Engineering & Computing
Adama Science and Technology University, Adama, Ethiopia
[2]Dean, School of Electrical Engineering and Computing, Computer Science and Engineering Program
Adama Science & Technology University, Adama, Ethiopia
[3]Department of Computer Science and Engineering, Prince Shri Venkateshwara Padmavathy Engineering College, Chennai, India

## ABSTRACT

Software engineering principles and practice are primarily focused in a conceptual environment where the entire software as a whole is scrutinized to identify vulnerabilities. There are many inadequacies of such behavior based software engineering as such conceptual software engineering is to integrate the cause of such cases into the complete series of existing and promising software engineering principles and practices. This model is to generate a testing tool that features in identifying vulnerabilities to increase the performance of the software. This is done by trying out the various known sequence, and the results are checked for consistency. Since systems grow in size and complexity, the performance-based vulnerability is becoming a more challenging task. This model is step by step validation processes that are induced to generate the vulnerability prediction tool for the software during its development process. This tool is used to identify vulnerability to provide software quality assurance. This is done by studying most of the known computer attacks and the safeguard measure against such vulnerabilities is applied early in the software development life cycle. This model is to check whether the software exhibits proper behavior when improper usage or improper input is given to the system, and this logic is carried out in every phase of the software development life cycle.

**Keywords :** Test Case Generation, Test suite minimization, concept analysis, lattice, Genetic Algorithm, Empirical analysis, Coverage analysis

## I. INTRODUCTION

Conventional performance based testing the run time performance of the software but chances are that during the process of software development conformed to performance, and the software can develop some vulnerable areas. This vulnerability can happen at any level right from [1] requirement analysis to implementation phase. One of the hectic tasks is to find all possible vulnerabilities that can occur at any time during the development of the software. One way to identify those are by running a [2] vulnerability specific predicates and the other is done by trying out all possible improper inputs at every possible entry points to the system. The conventional or traditional software testing checks to see if the software is stable based on the specification devised during the analysis phase. Developers tend to make mistakes when [3] writing the software, math errors, incomplete logic or incorrect use of functions. Such mistakes can occur even earlier in the development process when such mistakes have a security implication, then that part of the software is termed vulnerable. Vulnerabilities are caused due to increase in malicious software aspects which arises when the system produces inconsistent responses.

The conventional based testing is to uncover performance problems which occur due to the presence of faulty inputs and it helps to look through a way to enhance the performance. Most of such models miss out a way to identify the parts which caused the performance problem instead they focus on the way to increase the performance by other methods like [4] patching up after the development of the whole software which is unreliable and disruptive. During implementation, some of the functionality might be missed out, so the developers tend to include those after the completion of the software which may create some vulnerable areas, but it helps to [5] improve the quality of the software. The advantage of doing so is by inclusion of pre-built components like libraries, which usually does not create any logical change to software but it poses a high risk of leading to new vulnerabilities which pose different level of risk which can be identified with the help of a [6] vulnerability scoring system or by [7] measuring vulnerabilities and their properties.

Generally, such systems are conceptual because when it is developed its outcome is seen only at the closure of the software development process, and the developers assume some conditions by on some functionalities of the software. These conditions may behave in a normal way when it occurs as assumed earlier, only when the input to the functionally complete software satisfies the assumed condition. When this condition is not matched for input, then the software may behave unpredictably. These are mainly due to the model assumed with certain characteristics even before implementing the software component. In some cases, some known abnormal inputs are handled by the use of exceptions which are used when a set of predefined condition fails, but mostly there occurs a new set of abnormal inputs which is not yet defined. Such abnormalities which can occur are hard to track down, so there is a need to make use of a tool to identify the vulnerabilities. Though the identification of all potential vulnerability is hardly possible or in other words is just theoretical, but certain vulnerable areas can be predicted by the way the development process is carried out.

## II. Related Work

Software security has been a concern of serious study for at least 40 years, and an important stream of innovations that have improved the ability to protect networks and software applications. But attackers have adapted and changed various methods as the year passes [19]. Where do we the software Industry stand at this stage in the battle between attackers and defenders? Are attackers gaining hold, as it often seems when reading press accounts of the recent data exposure? The analysis [19] seeks to answer these questions using data from the US National Vulnerability Database (NVD) and to identify classes of vulnerabilities where improvements will be the most cost-effective.

The approach can be implemented by the use of different models for tracking vulnerabilities. One such model is to include [8], [9] vulnerability assessment tool to identify vulnerable areas by [10] penetrating known set of inputs that may lead to vulnerability. The other is to make different components of the model to check the relevancy between improper inputs and its corresponding known exceptions for a given input. [15] elucidates the overview and different techniques used in vulnerability assessment and testing for penetration. [16] focuses on the vulnerabilities that occur in any Web application and methods for the removal of these vulnerabilities. A vulnerability assessment is a process of identifying, quantifying, and prioritizing (or ranking) the vulnerabilities in a system [16]. In this work the vulnerability assessment was conducted to find the weaknesses inherent in the Information systems that may be exploited, leading to software system breach. The [16] discusses different types of SQL injection methods. [17] analyses current penetration testing tools and the tests them on a use case web application, build specifically with present security flaws.

The process of penetration testing is described in detail, and the performance of each tool is evaluated. [17] also summarizes the recommended practices to mitigate found flaws. [18] proposes a generic approach for designing vulnerability testing tools for web services, which includes the definition of the testing process and the tool components. Based on the proposed approach, [18] presents the design of three innovative testing tools that implement three complementary techniques (attack signatures, improved penetration testing, interface monitoring, and runtime anomaly detection) for detecting injection vulnerabilities, thus offering extensive support for different scenarios. A case study is designed to demonstrate the tools for the particular case of SQL Injection vulnerabilities [18]. The experimental evaluation of the work presents that the tools can be used in different scenarios with improved effectiveness and that the proposed tool outperform many other commercial tools by achieving improved in detection coverage and the decrease in false-positive rates.

## III. Vulnerability testing in SDLC

This section discusses the computer security model [14] describes the integration of vulnerability testing in the conventional model of testing a system. Increased functionality leads to decreasing the performance of the system. To overcome the above situation, the system is monitored in a controlled environment in which the absence of real-world scenario does not make an effective identification of the vulnerabilities. So the testing of vulnerabilities in [11], [12] hostile environment does not identify all the defects that can occur in a real-time system. Testing of the system in which the actual errors or failures that occur at each stage cannot be directly simulated or predicted, which eventually leads to very minimal testing in the vulnerable area or it is completely ignored. When such a case happens in a real-time system, the chances are that it can lead to data corruption and affect the system stability which is some of the important parameters in defining the performance of the system. Traditionally testing helps in crosschecking specifications of a software system which is mostly verification process. To differentiate from the conventional type of testing we do the vulnerability assessment to [13] identify the unknown vulnerabilities from the exploits by malicious programs at each phase of the Software Development Life Cycle (SDLC).

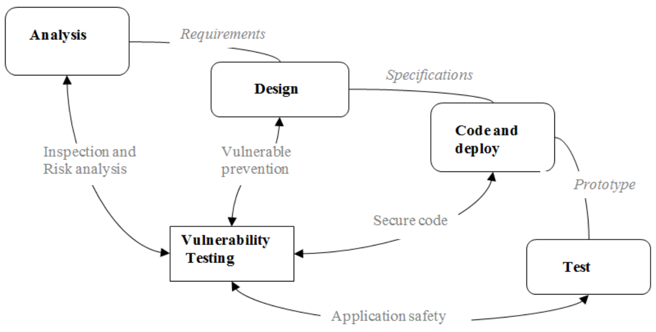

**Figure 1 :** Vulnerability testing in SDLC

The above computer security model [14] describes the integration of vulnerability testing in the conventional model of testing a system. Each of the documented and collected samples or information during the analysis phase is scanned for resources that may trigger some vulnerability, and its corresponding risk is assessed. Since vulnerability testing is planned to be induced in each phase of the software development life cycle, the principles followed for the specification of the system are explored for vulnerable occurring cases, and those set of specifications are redefined.

## IV. Instructive Example

In this section, we focus on a typical scenario which can explain the concept of the unknown behavior of the system when abnormal inputs are given.

Consider a reservation system in which all passenger information which is stored in a document format is given as input to the system. The processing of text includes the details of each of the passengers.

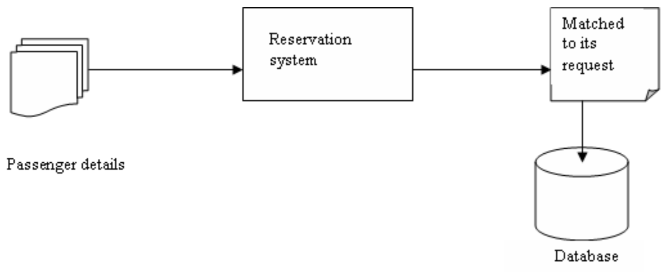

**Figure 2 :** A simple illustration of an example

The processing, when done without the aid of a standard translator, may lead to some vulnerability. The above system process the input documents in say, the English fonts and one user details is in some unknown font, which needs to be stored in the directories of the reservation system. This is not entirely possible because of the security mechanism all the reservation system employs. Thus the system tries to process the data and generates a sequence of abnormal behaviors since the system assumes it to be a normal input.

## V. Problem Analysis

### 5.1 Performance analysis

A performance test on the software tends to find out the performance related problem of the software under the normal environment. The performance-related problems may directly affect the performance of the system in many ways such as lack of appropriate resource, inadequate system capabilities, weak operating system and poorly designed software functionality. We are going to consider only the performance degrade that occurs due to the existence of vulnerable areas and employ suitable safeguard measures to make the software behave consistently. The performance of the system can be explained with the help of some attributes like the number of operations or transactions made, the amount of data it can handle at a time and so on.

Normally in this approach, two different components help to reduce the performance of the software which is load testing and stress testing. The load testing is used to check the performance when the input is given in a variety of combinations, and stress testing determines the amount of input sequence that it can handle at a time. The performance change in normal software can be incorporated by the following graph, in which we assume three factors may play a vital role in determining the stability of the system, but other such factors that may degrade the performance aspects of the software which are at the moment unexplored.

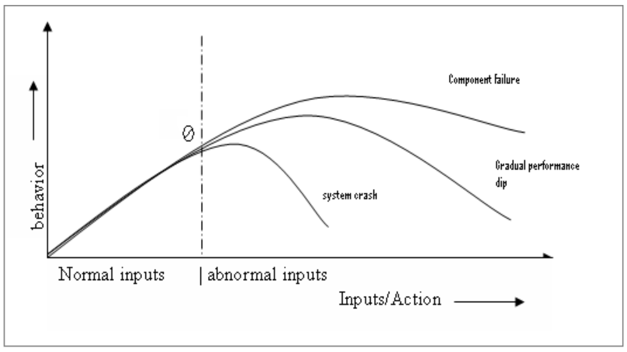

**Figure 3 :** Performance measure based on normal and abnormal inputs

The performance is devised based on the type of input given to a system and how it behaves for each of those inputs. So the performance of the software under a normal environment with normal input sequence will increase steadily consistently, but when the set of abnormal inputs is introduced, then the performance varies based on the different effects caused by the abnormal inputs in the software which can be system crash, component failure or gradual performance decrease. The point at which the software starts to process the abnormal inputs are assumed to be ø point. The behavior of the software typically depends on the intensity of the abnormal input processed by the software. When some components of the software fail, then it will affect the performance in long time execution of the software while in case of recurring failure of functions then the software performance goes down gradually and only when the system cannot handle all those abnormal inputs the software crashes abruptly. As the future work other performance-related factors such as response time for the initiated activities, processing time, memory constraint, security constraint, temporary and permanent resources failure will be considered to determine the software performance.

### 5.2 Collecting Known sequences

Software vulnerabilities can occur for many reasons, but this paper focuses on those generated by abnormal inputs. To identify those inputs we need to have a procedural approach, which can be done by exploring the problem at hand.

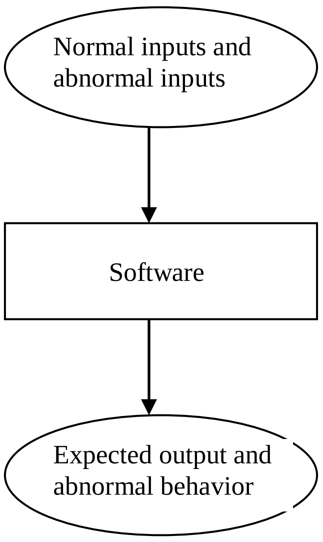
Normal inputs and abnormal inputs
Software
Expected output and abnormal behavior

**Figure 4:** Block diagram showing the behavior of the system

When the abnormal inputs are induced, the system also behaves abnormally in a different way which may lead to vulnerabilities. This case can be minimized by following the VulnerMini algorithm.

1. *Choose $a_i$ from $n_i$ that forces the system to generate error reports*
2. *Design $a_i$ based on the faults*
3. *Repeat for various series of inputs (both $a_i$ and $n_i$)*
4. *Force $O_e$ to be generated*
5. *Interrelate $a_i$, $n_i$ and $O_e$.*
6. *Identify the frequency of errors in each component*
7. *Reframe the error occurring part in an efficient way.*

***Notation***

| | |
|---|---|
| $a_i$ | Abnormal inputs |
| $n_i$ | Normal inputs |
| $O_e$ | Invalid output or output error |

As the algorithm explains the way to reduce the failures in the component of a system the same can be illustrated with the use of the block diagram in which the abnormal inputs are processed in the system with the help of an abnormal input monitor that decides on the processing of such inputs.

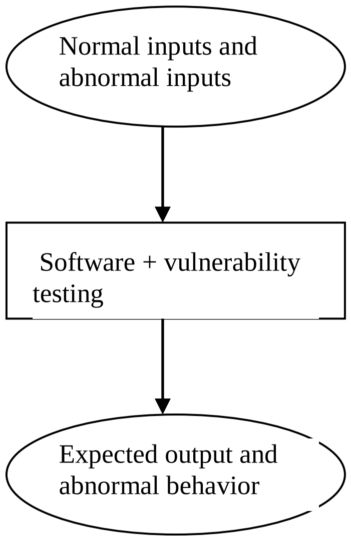
Normal inputs and abnormal inputs
Software + vulnerability testing
Expected output and abnormal behavior

**Figure 5 :** Block diagram to overcome the limitations in a system

### 5.3 Vulnerability analysis

Vulnerability factors are included while identifying the performance measures such conditions are aimed to measure the frequency of occurrence of that results for the given input. A system can handle a large number of simultaneous inputs which are previously defined, but in some case, this may be violated by unknown factors, which are frequently monitored that are affecting the performance of the system. The vulnerability testing is induced in the above case with which the duration for processing the part responsible for performance dip are identified and induced to vulnerability testing.

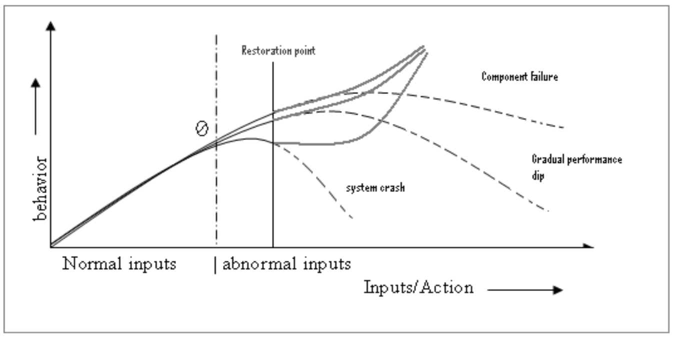

**Figure 6 :** Performance measure for all cases and the decline is checked by a restoration point

When the ø point occurs, the part of the system that is responsible for the failure is identified, and a restoration point is set so that the system is tried to the reinstate the system to normalcy. The restoration point depends upon the complexity of the problem that is induced due to the set of abnormal inputs. Further, the vulnerability of the software can be represented by the following terms.

***Notation***

| | |
|---|---|
| Ø | The point the performance decreases |
| $S_i$ | System impact that influences performance degrade |
| $P_i$ | Duration for the system to restore |
| $C_{f0}$ | Gradual decrease in performance |
| $C_{f1}$ | Crashing down of the system |
| $D_{f0}$ | Data not secure |
| $D_{f1}$ | Data is secure |

The $P_i$ factor is known as the probation period during which the faulty components of the system are identified, while the factors $C_{f0}$ and $C_{f1}$ are Boolean variables that determine the crash factor. $D_{f0}$ and $D_{f1}$ are data factors with the Boolean logic that determines whether the data is still safe or not.

## 6. Conclusion and Future Work

The main contributions of this paper are the concept of including vulnerability testing in the conventional methods to enhance the performance of the system. This approach is focused mainly on the implementation part which is where some vulnerability factors can be identified easily, and these are predefined for the next phase of processing. This idea of this paper has proposed a software testing process in the name of vulnerability testing that deviates from conventional software testing. When software models are completed in an organization, the input data may not be available immediately which needs collecting the data for all input logic. So we put forward simple procedure to identify the defects in the software based on the past summary of defects in the organization. From the procedural point of view, the explicit contribution of this idea is making the vulnerability testing with the behavioral patterns of the system. Also, this process can carry out different testing strategies according to the behavior of the system. It can generate new test cases by supervising the input actions, and it generates probable outputs from the implementation of the various components of a system. As a part of future work, we intend to identify some known techniques to identify vulnerabilities in an easier way, through which some wide variations of abnormal sequences can be found.

## VI. REFERENCES

1. Premkumar T. Devanbu, Stuart Stubblebine, Software engineering for security: a roadmap, International Conference on The Future of Software Engineering, ICSE '00, Limerick, Ireland - June 04 - 11, Pages 227-239, 2000
2. Ashlesha Joshi, Samuel T. King, George W. Dunlap, and Peter M. Chen, "Detecting past and present intrusions through vulnerability-specific predicates", Twentieth ACM symposium on Operating systems principles (SOSP '05). ACM, New York, NY, USA, 91-

104, 2005. DOI: https://doi.org/10.1145/1095810.1095820

3. Filippo Ricca, Paolo Tonella, "Detecting Anomaly and Failure in Web Applications" IEEE Multimedia, pg. no:44, April-June 2006
4. Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. "Shield: vulnerability-driven network filters for preventing known vulnerability exploits", International Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '04). ACM, New York, NY, USA, 193-204, 2004, DOI: https://doi.org/10.1145/1015467.1015489
5. Ashish Arora, Rahul Telang, "Economics of Software Vulnerability Disclosure", IEEE SECURITY & PRIVACY JANUARY/FEBRUARY 2005.
6. Peter Mell, Sasha Romanosky, Karen Scarfone "Common Vulnerability Scoring System", IEEE SECURITY & PRIVACY NOVEMBER/DECEMBER 2006.
7. Yehuda Vardi and Cun-Hui Zhang, "Measures of Network Vulnerability", IEEE SIGNAL PROCESSING LETTERS, VOL. 14, NO. 5, May 2007 Pg.no 313
8. Elspeth Wales, "Vulnerability assessment tools", Network Security, Vol 2003, Issue 7, Pages 15-17, July 2003, https://doi.org/10.1016/S1353-4858(03)00712-8
9. Andrew Blyth, "An XML-based architecture to perform data integration and data unification in vulnerability assessments", Information Security Technical Report 8 (4), 14-25, 2003
10. Sankalp Singh, James Lyons, and David M. Nicol. Fast model-based penetration testing. 36th International conference on Winter simulation (WSC '04). Winter Simulation Conference 309-317, 2004.
11. W. Du and A. P. Mathur, "Testing for software vulnerability using environment perturbation," in Proceedings of the International Conference on Dependable Systems and Networks (DSN 2000), Workshop On Dependability Versus Malicious Faults, New York City, NY, June 2000, pp. 603-612
12. Herbert H. Thompson, James A. Whittaker, and Florence E. Mottay, "Software security vulnerability testing in hostile environments", ACM Symposium on Applied Computing (SAC '02). ACM, New York, NY, USA, 260-264, 2002. DOI=http://dx.doi.org/10.1145/508791.508844
13. Jedidiah R. Crandall, Zhendong Su, S. Felix Wu, "On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic and Metamorphic Worm Exploits" in CCS-05, November 7-11, 2005, Alexandria, Virginia, USA. Pg. no 235
14. George Whitson. 2003. Computer security: theory, process and management. J. Comput. Sci. Coll. 18, 6 (June 2003), 57-66.
15. P. S. Shinde and S. B. Ardhapurkar, "Cyber security analysis using vulnerability assessment and penetration testing," 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare, Coimbatore, 2016, pp. 1-5. DOI: 10.1109/STARTUP.2016.7583912
16. I. Altaf, F. U. Rashid, J. A. Dar and M. Rafiq, "Vulnerability assessment and patching management," 2015 International Conference on Soft Computing Techniques and Implementations (ICSCTI), Faridabad, 2015, pp. 16-21.
17. F. Holik and S. Neradova, "Vulnerabilities of modern web applications," 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, 2017, pp. 1256-1261.
18. Nuno Antunes, Marco Vieira, "Designing vulnerability testing tools for web services: approach, components, and tools", International Journal of Information Security archive, Vol. 16, Issue 4, Pp 435-457, August 2017.
19. Richard Kuhn, Mohammad Raunak, Raghu Kacker, An Analysis of Vulnerability Trends, 2008-2016, IEEE International Conference on Software Quality Reliability and Security (QRS-C 2017), Prague, Czech Republic, July 25-29, pp. 587-588, 2017
20. Roger S. Pressman, "Software Engineering: A Practitioner's Approach" (7th ed.). McGraw-Hill Higher Education, 2010.
21. Tao Xie, "Improving Effectiveness of Automated Software Testing in the Absence of Specifications" , 22nd IEEE International Conference on Software Maintenance (ICSM'06), 2006

22. J. Derrick and E. Boiten, "Testing refinements of state-based formal specifications," Software Testing, Verification and Reliability, no. 9, pp. 27-50, July 1999.
23. M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing," Department of Computer Science, The University of Waikato, New Zealand, Tech., Rep. 04/2006, April 2006.
24. C. Ramakrishnan and R. Sekar, "Model-based vulnerability analysis of computer systems," International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI-98), Pisa, September, 1998.
25. G. Wimmel, H. Loetzbeyer, A. Pretschner, and O. Slotosch, "Specification based test sequence generation with propositional logic", Special Issue on Specification Based Testing, Software Testing, Verification and Reliability, vol. 10, no. 4, pp. 229-248, 2000.
26. T.Downs, P. Garrone, "Some New Models of Software Testing with Performance Comparisons", IEEE Transactions on Reliability, VOL. 40, NO. 3, 1991
27. Sigrid Eldh, Hans Hansson, Sasikumar Punnekkat, Anders Pettersson, Daniel Sundmark, "A Framework for Comparing Efficiency, Effectiveness and Applicability of Software Testing Techniques", Proceedings of the Testing: Academic & Industrial Conference - Practice And Research Techniques (TAIC PART'06), 2006.
28. Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin, and Chung-Hung Tsai, "Web application security assessment by fault injection and behavior monitoring", Proceedings of the 12th international conference on World Wide Web (WWW '03). ACM, New York, NY, USA, 148-159., DOI=http://dx.doi.org/10.1145/775152.775174, . 2003.