Open Access Library Journal
2020, Volume 7, e6831
ISSN Online: 2333-9721
ISSN Print: 2333-9705

The Intelligent Software Systems: The Practical Implementation of Software Security Vulnerabilities Detection Modeling

Musoni Wilson[1], Umutesi Liliane[2], Mbanzabugabo Jean Baptiste[3]

[1]Institute of Applied Mathematics and Computer Science, National Research Tomsk State University, Tomsk, Russia
[2]IT and Computer Architecture, University of Kigali, Kigali, Rwanda
[3]Software Engineering, University of Tourism Technology and Business Studies (UTB), Kigali, Rwanda
Email: musoniwilson1@gmail.com

How to cite this paper: Wilson, M., Liliane, U. and Baptiste, M.J. (2020) The Intelligent Software Systems: The Practical Implementation of Software Security Vulnerabilities Detection Modeling. *Open Access Library Journal*, **7**: e6831.
https://doi.org/10.4236/oalib.1106831

Received: September 17, 2020
Accepted: November 13, 2020
Published: November 16, 2020

Abstract

System products are widely used in almost all applications. Most of the human capacity has been converted to software solutions. Measuring and evaluating the quality of software products has become a problem for many companies that may be looking for software solutions. There are a number of skills that are required and used to make good software. As time goes on, new software is advancing and this has caused security problems to persist and thus affect the software's performance. New components have introduced a new security system, which is called software security enhancement. In this study, I found the latest methods and techniques used to detect a few software errors and all other security threats. This method has the potential to identify possible symptoms indicating these causes. The new method has the ability to detect weaknesses before an attack. The VDC, which modifies the structure, compares its resources, and then gives a public account of those attacks. This method is used to remove the best security measures used to convince users and developers of the same model that will be used to crack down on software attacks. This paper presents a description of security objectives and best algorithms to address vulnerability issues to provide better results from planned attacks. The article deals with the implementation of the technical program. Finally, an analysis of the results was conducted to demonstrate the performance of this approach to the development of more secure systems.

Subject Areas

Computer and Network Security

Keywords

Intelligent Software Systems

1. Introduction

Understanding the weaknesses of the system is a matter of concern to the industry of the system. Studies have shown that system designers and researchers are working to use all possible methods to help reduce problems. Although security measures have been put in place, the death toll from the attacks has risen sharply. In this study, we present a model based on the ability to identify anything that could interfere with safety-related programs, called testive testing. An easy way to identify security bugs in software well. Implement a reliable security system that assures developers and software users in a safe and secure environment for collaboration. This approach is unique because developers can search for and change security-related programs. The technology is now used to measure the quality of applications, including insecurity, with a focus on specific security such as bookstores, cross-border error checking and everything else. This has its limitations because users cannot identify a weakness that a particular device can solve, and users do not expect their hardware to be constantly updated since software weaknesses continue to change. Also, modern security users face problems because they do not have the ability to maintain equipment without consulting the dealer. Employers must keep in touch with system owners at all times. The passport technique connects the employer with the developer who needs to use a model to control for visible weaknesses. Software developers have a vested interest in accessing the content of the device, which they know and the tools are easy to develop and have been quickly added to many devices in the past to make them more secure and up-to-date.

The technique employed by this mechanism can detect all the vulnerabilities in both the past and the present software industry. Software security vulnerability through SGM exposes a software to these threats by tying this software to security-related activities such as physical inspection, content analysis, and process upgrading. All vulnerabilities that are known and reported frequently are being detected and tested during software development. Finally, the study introduces a TestInv-Code that uses the passive mechanism to test vulnerability problems in software engineered in C language. The code detects vulnerabilities using passive techniques to measure the standards and efficiency of the method, detecting related susceptibilities in diverse programs. The introduction of a security-oriented model (SGM) has actively contributed to the effective implementation of security models that tests and identifies causes of vulnerability. This helps to avoid developers from learning new security mechanisms hence it is easier to implement this technique, especially with the new developers in the software industry. This research paper was guided by several contributions including: the complete definition of security model language and all the available methods to create security-oriented models. Complete approaches towards the development of a secure and full security model to avoid software ambiguity.

2. Methodology

A security-oriented model copies a given goal to be attained. The model was established to expressively replace the manual and physical way the vulnerabilities are handled including. The SGM employs a mechanism that brings in the excellent relationship in models that are being examined, a critical property that is being used in passive testing. The security goal mechanism handles anything that affects the security of software including software security vulnerabilities. The different types of vulnerabilities being detected include security functionalities; security-related programming activities and other different weaknesses. The graph represents the total number of occurrences and the possible equilibrium where all programs must meet to be accredited as secure. The security goal model intends to add the models and what they can express. The primary origin of SGM was that it is designed to replace an assured attack and the remaining loopholes as unplanned incidents. The ones that are represented with the angled edges are modeled using SGM. All the sub-goals are achieved when either one or all predecessors are completed. The broken edges are a representation of information flow between sub goals. The primary influence is dependent on the programmer either from the admin side or in the programmer side and then identify developer settings depending on which sub-goals need to be achieved.

The model identifies and highlights some ways in which the outbreak by replacing the software can be made. This is possible when the developer gets access to the hosting server, after the successful software upload, the developer completely replaces the software with a modified one, which manipulates data the same way as the original code. The renewed software does all the fore planned activities by the owner of the software, but in the long run, relevant information is tapped through a SQL injection attack (Alwan, & Younis, 2017) [1]. Access to the main servers could be direct or by stealing the password from the administrator. Software vulnerability can be caused in different ways. The main ways through which these attacks are achieved include the unsafe use of data that is read by the system users and by copying data within a loop by controlling the data read from the users' side. The model indicates how vulnerabilities can be caused through the controlled attack to the user data for programs that do not use adaptive buffers and admits information read from users. This increases the risks to attack hence rendering the software to other security attacks including complete denial of service.

2.1. Passive Testing Mode of Detection

This technique detects faults in software or system by examining the subsections of software independently. Passive testing does not use test input values to explore how secure a system is. Passive testing is mostly considered since it can detect faults in the system that is being examined. The system under examination is observed through collecting traces produced, and the deviations in the system outcomes are used to analyze the whole system functionality. The small

hints of deviation can bring a significant difference to the results of a system (Miller & Arisha, 2015) [2]. The formal way of conducting passive testing is highlighted by giving standards through which a system can be pronounced secure or insecure. Passive testing can be employed in different capacities. For example, by using the finite state machine to detect the predictable characteristics of a scheme and in network management to identify the configuration provisioning and security subsystems, the passive testing can be used to analyze the security strengths. The TestInv-Code tool is a reflexive testing mechanism that allows recognized attack models. It assesses by detecting the vulnerability by checking the hints of the software program code while it is implementing. Code testing is done formally under the monitoring of the TestInv code tool. Passive testing takes the following iterative steps:

2.2. Vulnerability Modeling

The security-oriented model whose main goal is to identify the potential vulnerabilities models the vulnerability that is being detected.

2.3. The Formal Definition of Causes

Here, security experts are entitled to identify a cause, and it's predicate. These predicates are used to define precisely what the designing tools will look for in the execution traces.

2.4. Vulnerability Checking

The TestInv-Code is used to check the existence of the vulnerability when a program is executing. The tool can identify the weaknesses found and indicating their correct location. The device analyzes the execution traces. The preceding two steps are done once in a case where a problem occurs. Vulnerability checking is automated to allow free and unconditional simulation of results. When the automatic identification of results fails, user inputs are used so that the condition.

2.5. Vulnerability Detection Conditions

The security-oriented model provides the information about possible problems in the codes. The information is useful since it can be used as a requirement to test software security vulnerabilities. The information will be helpful in future designs of new software that attack resistant. The main aim of vulnerability detection is to detect the occurrence of an attack in software (Antunes & Vieira, 2015) [3]. The causes of security attacks are usually expressed in natural language or coding language. These conditions are easily decoded by the target system yet making them hard to be detected by the users of the arrangements. The SGM detection tool can be used to identify the slight changes in the system behavior by closely monitoring the output. The usual algorithmic patterns are looked into to determine the deviation from the usual pattern. More elaborate

security mode can be designed using different logical operators.

3. Results

The security-oriented mechanism needs to identify the probable cases that may make the testing of scenarios hard to recognize the possible vulnerabilities. The occurrences that may cause susceptibility in building the test cases identified whether in the program that is being tested performs decisive actions under SGM particular circumstances and then recovers it from the attack (Liu, Yang & Zhang, 2015) [4].

For SGM to be implemented fully, it must fulfill the following requirements:

- The subsystems should be replaced by a suitable corresponding model.
- Do away with the qualitative sub-elements of the security mechanism and maintain physical ones. The qualitative goals are done through human intervention hence cannot be checked.
- The SGM must replace the counteracting sub goals with similar contributing subgoals. By making sure, the testing process is successful, SGM checks for the worst situations to control whether the susceptibility is available or not.

3.1. Extract the Testing Information Using Templates

This is done to pedal the extraction of the affected part of the code. The process is made possible by creating two models, one to represent master action and another one behind which every sub-master is processed. The subgroups are treated logically to produce vulnerability detection control. Every case should contain an independent master plan that is the Act_master, which produce related vulnerability. SGM does not proceed if there is a missing master action. (K. M. Khan and J. Han. 2013) [5] formulates the algorithm for the axes of these paths as conditions $\{X_1, \cdots, \cdots, Y_n\}$. A specific situation may occur, called missing condition C_k that need to satisfy the following Acting Master. Let $\{Y_1, \cdots, Y_K, \cdots, Y_n\}$ be the predicate condition $\{Z, \cdots, C, \cdots, C_n\}$. The formal VDC expressing this risky scenario information is represented by: $Action\ Master/(Y_1 \wedge \ldots \wedge Y_{k-1} \wedge Y_{K+1} \ldots \wedge Y_n)$; Y_K. When the analysis of master actions is complete, the similar sub goal is extended to give more information about the pre-existing conditions. Information edges must be taken into account since the subgroups of the sub goals provide sensitive information about the roots of the attacks. The design helps in the production of more secure software. The main template is designed to record the relevant parameter that is related to the main SGM and the expected possible feeds and the normal parameters. The table below explains briefly.

3.2. Produce the Templates Automatically to Obtain Vulnerability Detection Control

The collected information from both the master action whose conditional templates are automatically executed to produce the expressions of vulnerabilities detection is used to predict the future of the rest of the systems. Outcomes must

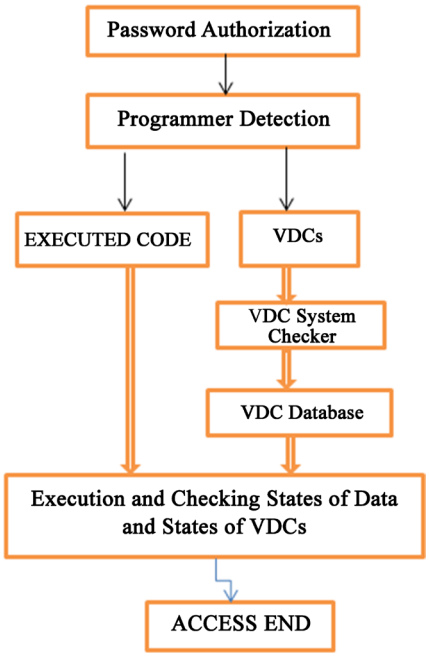

Figure 1. Flowchart of accessing the software system.

be corresponding to the responsible testing scenario. This means all the tests in the survey systems are done to detect the considered vulnerability.

In Figure 1, the code finds the evidence of vulnerabilities during the decoding of a software system. The VDCs analyses the execution traces of attacks from malware by identifying the weaknesses that are evident in the program. The TestInv-code tool can detect calls by malicious software to a software system and interpret its intended attack target to the system. It does this by individually scanning through variable bounds and in cases where there will be variables; the program will check numerical values. SGMe other conditions may not be easily detected since SQL injection makes it simple for trespassers to get access to SGMe specific services using the customer page.

4. Conclusion

Application control is an important part of all application development activities. Continuous growth in the use of automated tools has contributed to the development of reliable and reliable software. The scientific guidelines provided require the use of intelligent equipment. Complete equipment is only being renovated so that newcomers can learn the techniques used to prevent weakness. In the future, I look forward to using the method of demonstrating how to make more or less secure software for all vulnerabilities in the application industry, and also to continue to look at the impact of computer attacks.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Alwan, Z.S. and Younis, M.F. (2017) Detection and Prevention of SQL Injection Attack: A Survey. *International Journal of Computer Science and Mobile Computing*, **6**, 5-17.

[2] Miller, R.E. and Arisha, K.A. (2015) Fault Identification in Networks by Passive Testing. 34*th Annual Simulation Symposium*, Seattle, 26-26 April 2001, 277-284.

[3] Antunes, N. and Vieira, M. (2015) Assessing and Comparing Vulnerability Detection Tools for Web Services. *IEEE Transactions on Services Computing*, **8**, 269-283. https://doi.org/10.1109/TSC.2014.2310221

[4] Liu, W., Yang, L. and Zhang, W. (2015) Modelling Binary Oriented Software Buffer-Overflow Vulnerability in Process Algebra. 2015 *Seventh International Symposium on Parallel Architectures, Algorithms and Programming* (*PAAP*), Nanjing, 12-14 December 2015, 20-25. https://doi.org/10.1109/PAAP.2015.15

[5] Khan, K.M. and Han, J. (2013) A Security Characterisation Framework for Trustworthy Component Based Software Systems. 27*th Annual International Computer Software and Applications Conference* (*COMPAC* 2003), Dallas, TX, 3-6 November 2003, 164-169.