International Journal of Modern Computer Science (IJMCS)
Volume 5, Issue 5, October, 2017
ISSN: 2320-7868 (Online)

Analysis of Static Code Techniques for Vulnerability Detection: A reviewed Literature

Faith MueniMusyoka
School of Computing and Informatics
Gretsa University
Thika, Kenya
E-mail: faithmueni24@gmail.com

Abstract: Web applications have become an integral part of the daily lives of trillions of users. These systems are usually complex and are developed by different programmers. Regularly programmers make mistakes in the code which could generate critical software vulnerabilities. Despite the knowledge about vulnerabilities nowadays there is still a growing tendency in the number of reported vulnerabilities, reason why software security has become an important field of research. Due to the presence of vulnerabilities it has been necessary to have tools that can help programmers detect them in code development stage. This paper has analysed pattern matching and taints analysis techniques that are currently used in development of static tools to in detection of vulnerabilities.

Keywords: static code analysis techniques, vulnerability, taint analysis, pattern matching

I. INTRODUCTION

The rapid emergence of Information Technology has obligated organizations embrace the automation and integration of their services. This emergence of technology has led many programmers use open source software (OSS) that contain free source code and are. However, security of Web applications has become increasingly important in the last decade since the applications have become the custom for a wide range of software development projects. Most of web based applications are carrying with them sensitive information of customers which, if compromised, can lead to significant downtime and damage to entire system thus, it is crucial to protect such application from hacker attacks which recently, has brought with it data protection and consumer trust security issue. This has led to development of static tools that detect malicious code in the programs during development to ensure its security.

Agile approaches to software development require that the code is refactored, reviewed and tested at eachdevelopment iteration phase. While the testing can be used to check if functional requirements are fulfilled as expected by the customer, checking security and safety of software is more difficult [1]. However, despite safety of Programming Languages, logical programming errors are easily made that lead to security vulnerabilities such as SQL injections and cross-site scripting attacks which malicious hackers are using to create exploits. Security vulnerability is a flaw within a software system that can be exploited to allow an attacker to reduce the system's information assurance[2]. Tactlessly, existing software development environments, such as Eclipse, NetBeans, among others, do not offer the service to make programmers know that they are writing insecure code. So, programmers end up using additional external tools, such as: IBM Appscan, Lapse+ and others to support secure coding. However, these tools regularly are not preferable for development workflow, because they either are not integrated into the development environments or do not detect the vulnerabilities exactly when they are added into the source code. Thus it becomes difficult for a programmer or a company to make a decision on the kind of the tool to get from the market.

In several deep-rooted report, Security vulnerabilities statistics show that 86 percent (%) of all audited websites contained at least one serious security vulnerability in their source code where by an attacker could take control over the whole website [3]. As a bad habit, a concern comes up when someone reports a critical security problem, then the whole company starts to worry about security when it is late. For example in 2001, Microsoft's Internet Information Server (IIS) had a vulnerability in the source code which cost an estimated $2 billion dollars of damage, afterwards Microsoft's chairman Bill Gates demanded employees to focus on building more secure software to avoid this type of problems in the future.

In a study on empirical evaluation of the ability of static code analysis tools to detect security vulnerabilities by [4] they concluded that, in recent times there is advancement in methods for static code analysis and the state-of-the-art tools are not very effective in detecting security vulnerabilities. They carried an experiment and found out that 27% of C/C++ vulnerabilities and 11% of Java vulnerabilities were missed by all three tools that were evaluating. Therefore there is need of automating static code analysis to detect specific vulnerabilities such as input validation-based vulnerabilities. The question remains to be, what are existing techniques that are used to implement these static tools?

II. RELATED WORK

Computer security is primarily a matter of secure design and architecture but still with the best designed architecture, security bugs will still show up due to poor implementation. Researchers have developed several techniques to help in identification of security bugs along the software development lifecycle, from code review to penetration testing. These techniques are two in nature: static code analysis and dynamic code analysis while this study will focus on static code analysis, which is an automated approach to perform code review. On the other hand, dynamic analysis techniques acquire information at runtime and require a running executable code. Static analysis scans all the source code while dynamic analysis verifies certain use cases being executed.

From a report of [5] states that performing of static analysis is a common proactive approach to detect security vulnerabilities in program code which examines input program code, applies specific rules or algorithms, and derives a list of vulnerable code present in a program that might result in vulnerability exploitations.

Static analysis technique analyzes the source code or binary code without executing it and identifies anti-patterns that lead to security bugs. This technique is similar to compilers that builds an Abstract Syntax Tree - a tree representation of the abstract syntactic structure - from the source code and analyzes it [1].The pioneer static analysis techniques such as control flow, data flow, inter-procedural analysis have beendeveloped for compiler generated code optimizations, and they are not intended for detecting securityvulnerabilities in program code [5]

There are three security vulnerabilities caused by missing input validation, or mis-validation of the input which are: SQL Injection, Cross Site Scripting (also called XSS) and Path Traversal[1]in the study, it was found out that many web applications written in ASP suffer from injection vulnerabilities, and static analysis makes it possible to track down these vulnerabilities before they are exposed on the web. Their study proposed a new technique to detect XSS attacks, SQL injection, and Path Traversal vulnerabilities based on taint analysis which tracks various kinds of external input, tags taint types, constructing control flow graph is constructed based on the use of data flow analysis of the relevant information, taint data propagate to various kinds of vulnerability functions, and detect the XSS or SQL Injection vulnerability in web application's source code.

Static code analysis tools are developed by use of several approaches such as pattern matching and string analysis which are in the category of simple techniques, and data flow analysis which is in the category of complex technique [6].

Static code analysis tools are developed for different programming languages. In the experiment carried out by [4] on evaluation of three static code analysis tool capabilities, the experiment showed that 27% of C/C++ vulnerabilities and 11% of Java vulnerabilities were missed by all three tools while, 41% of C/C++ and 21% of Java vulnerabilities were detected by all three tools. It was concluded that, the state-of-the-art tools are not very effective in detecting security vulnerabilities.

III. TAINT ANALYSIS TECHNIQUE

Taint analysis technique for developing static code analysis tool is majorly used to detect Tainted flow attacks which originate from program inputs maliciously crafted to exploit software vulnerabilities where these attacks are common in server-side scripting languages, such as PHP [7]. The technique solves the problem by use of $O(v^3)$ algorithm to solve it, where V is the number of program variables while the algorithm was, ten years later, implemented on the Pixy tool. [7]in their study they applied the algorithms using $O(v^2)$ and extended Static Single Assignment (e-SSA) program representation in conjunction with sparse dataflow analysis. The approaches used detected 36 vulnerabilities in well-known PHP programs. The study found out that tainted analysis approach outperformed the dataflow algorithm for larger inputs.

Input variables are marked as tainted and their proliferations are traced. According to a study carried out on mitigating and monitoring program security vulnerabilities, tainted data flow-based works was divided into four categories: static data type, implicit, grammar-based, and query-based tainting [5]. In static data type tainting, tainted information is marked by extending variable type information. In implicit tainting, program variables are not explicitly labeled as tainted and the approach is suitable for languages where there are no static type declarations in the code such as PHP. In grammar-based approach, tainted data flow can be tracked by grammar production rules where non terminals can be marked as tainted. In query-based tainting technique, a query specifies source objects and rules to transform sources to sink objects. In addition, a taint propagation technique which is a well-known and useful analysis tool in the fields of static and dynamic analysis.

IV. PATTERN MATCHING TECHNIQUE

In the implementation of static analysis tools, several structures have been applied. In a study carried out by [8], they applied the structure to develop a Static Code Analysis of IEC 61131-3 Programs. The structure had five parameters: PLC source program, analysis structures, rule framework, and Results. It had three stages to give analysis of results of the static code analysis tool which include: First, the input is the source

code of the PLC program, which is translated to an extended AST (Abstract Syntax Tree) representation. Secondly, AST representation is composed of advanced analysis structures (CFG, DFG, CG, and PS) which are computed. Thirdly, The AST and the computed analysis structures are input to a rule framework which is configurable with a set of rules where each rule is programmed to search for particular, possibly problematic program structures. The analysis results are the elements detected to cause rule violations. In AST representation stage, which is the central data structure, is created by a parser from source code, it is a tree of nodes where each program element is represented by a node of a specific type.

In a study of comparative analysis of static analysis, [9], string pattern-matching technique was pointed out as the pioneer static analysis approach. This approach relies on a known set of library function calls that might cause vulnerabilities. A set of rules are developed which represent signature of vulnerable code patterns. The program code is then tokenized to identify vulnerable pattern of strings that represent vulnerable function calls and arguments. A recent variation of string pattern matching-based technique is to scan executable program code to detect vulnerable function calls.

V. CONCLUSION

Pattern matching is one of the most attractive features of functional programming languages while taint analysis is greatly adopted in server-side scripting languages. To ensure positive vulnerabilities are detected, then there is need to combine static analysis techniques with data mining technique.

REFERENCES

[1] M. Guarnieri, P. El Khoury, and G. Serme, "Security vulnerabilities Detection and protection using Eclipse." Università di Bergamo, Italy, 2013.

[2] R. de Janeiro, "Early Vulnerability Detection for Supporting Secure Programming," Pontifícia Universidade Católica, 2015.

[3] J. Grossman, "Whitehat website security statistics report.," 2013.

[4] K. Goseva-Popstojanova and A. Perhinsch, "On the capability of static code analysis to detect security vulnerabilities," Orig. Res. Artic. Inf. Softw. Technol., vol. 68, no. 1, pp. 18–33, Dec. 2015.

[5] H. Shahriar and M. Zulkernine, "Mitigating and Monitoring Program Security Vulnerabilities," Queen's University Kingston, Ontario, Canada, 2010–572, Jun. 2010.

[6] G. Serme, A. De Oliveira, M. Guarnieri, and P. El Khoury, "Towards Assisted Remediation of Security Vulnerabilities," presented at the The Sixth International Conference on Emerging Security Information, Systems and Technologies, 2012, pp. 49–56.

[7] A. Rimsa, M. d'Amorim, F. Pereira, and R. Bigonha, "Efficient static checker for tainted variable attacks," J. Sci. Comput. Program., vol. 80, no. 1, pp. 91–105, Feb. 2014.

[8] H. Prahofer, F. Angerer, R. Ramler, and F. Grillenberger, "Static Code Analysis of IEC 61131-3 Programs: Comprehensive Tool Support and Experiences from Large-Scale Industrial Application," IEEE Trans. Ind. Inform., vol. 13, no. 1, pp. 37–47, Feb. 2017.

[9] H. Shahriar and M. Zulkernine, "Mitigating program security vulnerabilities: approaches and challenges," ACM Comput. Surv. vol. 44, no. 3, 2012.

AUTHOR'S BIOGRAPHY

Faith MueniMusyokareceived Bsc in Computer Science and Msc. in Information Technology both from MasindeMuliro University of Science and Technology, Kenya. She is currently a PhD in Information Technology student in Kabarak University, Kenya. She is Assistant Lecturer and Head, School of Computing and Informatics at Gretsa University, Kenya. The author has research interests in eHealth, software quality, Information system security, and data management.