Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 171 (2020) 2023–2029

Procedia Computer Science

www.elsevier.com/locate/procedia

Third International Conference on Computing and Network Communications (CoCoNet'19)

A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code

Prof. Arvinder Kaur[a*], Ruchikaa Nayyar[b*]

[a]GGSIPU,Sector 16 Dwarka,New Delhi,11018,India

[b]GGSIPU,Sector 16, Dwarka,New Delhi,110018,India

Abstract

Software security has become an essential component of software development process. It is necessary for an organisation to maintain software security in order to ensure integrity, authenticity and availability of the software product. To ensure software security, one of the major task is to identify vulnerabilities present in the source code before the software is being deployed. Detecting vulnerabilities in early phases of software development cycle, makes the process of fixing those vulnerabilities much easier for software developers. The vulnerability detection can be done either at the production phase, this means when the software is still being developed by statically auditing the source code, or dynamically at run time. In this study, vulnerability detection was done through Static code analysis process. Static code analysis can be done either manually or through automated tools. This paper focuses on using automated source code scanning tools for vulnerabilities detection in a software. Automated static Code Analysis tools audits the entire source code for its quality and identify any potential security vulnerability, if present. Unlike dynamic source code analysis that evaluates the source code behaviour during code execution, which is done quite late in the software development life cycle, Static Code Analysis leads to detection of security vulnerabilities in a source code in early stages of software development process, when the software is still in production phase because it does not require code to be in execution state. This paper firstly explains the importance of incorporating static code analysis in software development life cycle process so as to facilitate early detection of vulnerabilities in software product, and then present a comparative study of various static code analysis tools available for vulnerability detection in C/C++ and JAVA source code. The comparative study of three C/C++ static code analysis tools (flawfinder, RATS and CPPCheck) and two JAVA static code analysis tools (spotbugs and PMD) is done using Juliet (version1.3) test suite and APACHE tomcat dataset respectively, on the basis of

* Arvinder Kaur. Tel.: +91-9810434395
E-mail address: arvinder70@gmail.com

10.1016/j.procs.2020.04.217

category of vulnerability detected by each of the selected tool and the likelihood of false positive reported by each tool. Results showed that Flawfinder detected maximum categories of vulnerabilities and RATS and CPPCheck were almost similar in types of vulnerabilities detected. Also, it was observed that CPPCheck reported maximum number of false positives as compared to other two tools. Java static code analysis tools Spot bugs was able to detect more number of vulnerabilities than PMD.

Keywords: software security, vulnerabilities, static code analysis

1. Introduction

One of the most critical and important task for the organizations now-a-days is to maintain security of their software products. For the past years, there has been a tremendous increase in the number of vulnerabilities found and reported in various software products. The term software security refers to various measures implemented to ensure that the software is protected from any kind of malicious attacks that affects the functionality and behavior of the software by giving unauthorized access to the attacker. Various researchers and cyber security experts have given varied explanation of software security vulnerabilities. To sum up these explanations, software security vulnerability can be defined as a coding flaw present in the source code of a software that can be exploited by an attacker to gain unauthorized access of the software and hamper the behavior and functionality of the software product. Therefore, it is important for software development team to focus on finding and fixing these vulnerabilities in the source code before the software is being deployed [1-4].

Vulnerability detection has become an important task to ensure security of a software. There are two methods that are widely used for vulnerability detection in a source code, (1) Static Code Analysis, and (2) Dynamic Code Analysis. Both, Static code analysis and dynamic code analysis, is a process that audits the entire source code to find out security vulnerabilities. In Static code analysis the vulnerability detection is done while the software is still in development stage. On the contrary, dynamic code analysis is done after the developer runs the source code. As observed by many software developers and researchers, static code analysis is proven to be more beneficial and efficient, than dynamic code analysis for detecting software security vulnerabilities in a source code. Since, static code analysis leads to vulnerability detection in early phases of software development lifecycle, it makes the process of fixing those vulnerabilities easier for the software developer, and, also reduces the cost and time put in the by the organization for fixing those vulnerabilities. Static code analysis can be done either manually or through the use of automated source code scanning tools. Though, manual examination of the source code by software developers was a common practice in past, the process was quite tedious and could not yield very impressive results. To make the process of vulnerability detection more efficient in terms of time and the number of vulnerabilities detected, automated scanning tools were put to use.

Today, software development industry has both commercial and open source static code analysis tools. However, in this paper our focus is only on open source static code analysis tools. In this paper, we evaluate and compare different Static Code Analysis tools for the purpose of detection of security vulnerabilities in C/C++ programming language and JAVA programming language. In this paper, we have used Common Weakness Enumeration (CWE) standards for software vulnerability classification[5]. Common Weakness Enumeration (CWE) is used as common taxonomy for software weakness and vulnerabilities. CWE was created by MITRE Corporation with the support from US-CERT and the National Cyber Security Division of the U.S. Department of Homeland Security. In January 2019, the version 3.2 of CWE standards was released. The CWE weakness covers a large class of weaknesses such as flaws, vulnerabilities, bugs, faults and other errors in software implementation, code, design or architecture.

Different static code analysis tools comes with their own set of pros and cons[6]. Each of these tools are able to detect a wide range vulnerabilities, but still misses on some categories of vulnerabilities. In this paper, for the purpose of vulnerability detection in C/C++ source code we compare three open source static code analysis tool namely, FLAWFINDER, CPPCHECK AND SLPINT, and, for vulnerability detection in JAVA source code, the two static code analysis tools compared in this paper are SPOTBUGS and PMD. The comparative analysis is done on the basis of category of vulnerabilities detected and the likelihood of false positives reported by the tool. For the purpose of comparison of C/C++ static code analysis tool, we choose to benchmark these tools against JULIET(version 1.3) test suite for C/C++ programming language[7]. This test suite is a part of Software Assurance Reference Dataset(SARD). SARD is a component of SAMANTE project developed by NIST(National Institute of Standard and Technology)[8,9]. The Juliet (version 1.3) test suite is selected as it has reference to CWE (Common Weakness Enumeration) taxonomy and covers a large class of C/C++ vulnerabilities. The Juliet(Version 1.3) test suite contains various C/C++ classes that are exposed to a particular security vulnerability. The test cases cover a significant number of vulnerabilities from the CWE list. The test suite contains XML manifest file, which documents the weakness of each test cases. For each vulnerability, the manifest file consists of File name, the line number and the CWE id of the weakness. Altogether, the vulnerabilities are spread over 118 test cases. The JAVA source code of APACHE TOMCAT server was selected as dataset for vulnerability detection by various static code analysis tools. Each of the selected tool is then compared based on the category of vulnerabilities detected in the selected dataset by each of the selected tool[10].

Different categories of vulnerabilities have been studied and explored for this research work. Each category of security vulnerability is identifiable by the CWE id assigned to it. These CWE id helps to place each of the identified vulnerability across the globe into different categories. A few of the security vulnerability categories identified and explored during this research work are listed below:

CWE-120 : BUFFER OVERFLOW- A buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. A buffer overflow, or buffer overrun, occurs when more data is put into a fixed-length buffer than the buffer can handle. The extra information, which has to go somewhere, can overflow into adjacent memory space, corrupting or overwriting the data held in that space. This overflow usually results in a system crash, but it also creates the opportunity for an attacker to run arbitrary code or manipulate the coding errors to prompt malicious actions.

CWE-476 NULL POINTER DEREFERNCE-A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.

CWE-401 Memory Leak: A memory leak is an unintentional form of memory consumption whereby the developer fails to free an allocated block of memory when no longer needed. The consequences of such an issue depend on the application itself.

CWE-190 INTEGER OVERFLOW: Integer overflows and other integer manipulation vulnerabilities frequently result in buffer overflows. An integer overflow occurs when an arithmetic operation results in a number that is too large to be stored in the space allocated for it. Integers are stored in 32 bits on the x86 architecture; therefore, if an integer operation results in a number greater than 0xffffffff, an integer overflow occurs.

2. Static code analysis tools

Different programming languages have their own set of defined vulnerabilities. To detect these vulnerabilities , different static code analysis tools are available. In this study, different open source static analysis tools are discussed for C/C++

programming language and JAVA programming language. This section describes the static analysis tool used for this study.

FLAWFINDER: A Static Analyzer used for the purpose of vulnerability detection in a source code written in C/C++ programming language. Flawfinder is compatible with CWE. A C/C++ program is fed as input to the tool. The tool then generates a list of hits(vulnerabilities) sorted by risk level. Each vulnerability reported by Flawfinder is assigned a risk level ranging from 0 to 5, with 0 indicating the vulnerability having the lowest risk of being exploited [11].

RATS: Abbreviated for Rough Auditing Tool for Security, is static code analysis tool for auditing C/C++ source code for security vulnerabilities. Buffer Overflows, TOCTOU (Time of Check, Time of Use) race conditions are few examples of security vulnerabilities detected by this tool [12].

CPPCHECK: A static analyzer for vulnerability detection in C/C++ source code developed by Daniel . CPPCheck fails to detect any syntax errors. The tool focuses on having no false negatives, this means the tool does not report any wrong errors , however, it may not report all the errors. This implies, the tool reports lesser number of errors than actually present in the source code. This, in results lead to many false negatives. CPPCheck strongly focuses on detection of undefined behavior like dead pointers, divide by zero, integer overflows and null pointer dereferences to name a few [13].

SPOTBUGS: It is a Static Analyzer that audits the Java source code for different categories of bugs. It checks for more than 400 bug patterns. It is a successor of FINDBUGS static analysis tool. Spot bugs report different category of errors, however we are only interested in security vulnerabilities. Spot bugs comes with a plugin for detecting security vulnerabilities in Java source code called FIND_SEC_BUGS (Find Security Bugs) [14].

PMD: A tool for statically analyzing the JAVA source code for various programming flaws like unused variables, empty catch blocks, unnecessary object creation among many others. PMD features many built-in checks. It also supports an exclusive API to write your own rules, that can be done either in JAVA or as a self -contained XPath query. One of its major advantages is it identifies bad practices along with real defects. It's primary weakness is that it is reported to be a slow duplicate code detector. It can be integrated with various platforms like JDeveloper, Eclipse, JEdit, JBuilder, Blue J, Code Guide etc. For this study, we have used Eclipse plug-in [15].

3. Related Work

Many researchers in the past have evaluated and compared various static analysis tool for the purpose of vulnerability detection. , and identified the static analysis tool best suited for detecting vulnerability in a program written in a particular programming language.

Many researchers compared different open source static analysis tool available for C/C++ and Java source code. Different parameters have been used for the purpose of comparative analysis like detection ratio and execution time. In all these papers, authors have developed their own C/C++ and/or JAVA applications, and introduced various vulnerabilities in the application, to check each tool for vulnerabilities detected by them. Also, researchers have presented a study that compares commercial static code analysis tools available for detecting vulnerabilities in a software source code [6,16-20]

One of the shortcomings of all these comparative analysis is that the dataset used in all these paper is not a standard open source applications. Hence, it can be concluded that comparative analysis of the tools do not yield standard results. However, In our study, we have used JULIET (version 1.3) test suite as the dataset to examine tools for vulnerability detection and comparative analysis of the tools. It is a standard open source dataset freely available at NIST[8]. For vulnerability detection in JAVA source code, APACHE tomcat dataset has been used [10]. Also, our study is based on evaluating and comparing widely used open source static code analysis tools for the purpose of vulnerability detection in C/C++ and JAVA source code.

4. Comparative Analysis of JAVA and C/C++ static code analysis tools

This section describes the comparative results of the study performed on various static code analysis tools mentioned in this paper. Table 1 shows the comparative results of the three Static Code analysis tools for C/C++ source code. Juliet test suite (version1.3) was fed as input to all the three static code analysis tools and the output was recorded. The outcome for each category of vulnerability is furnished in table 1, with a tick mark if that category of vulnerability is detected by that particular tool and a cross-mark for non-detection of a vulnerability category. Though, Juliet Test suite (version1.3) had 118 categories of vulnerabilities, however, in table 1, we have displayed only 10 of those categories due to limitation of space. In table 2, we calculated the detection ratio for each of the tools. Table 2, represents the number of categories of vulnerabilities detected out of total number of vulnerabilities present in the dataset.

Table1. Comparative analysis of C/C++ static code analysis tools

VULNERABILITY CATEGORY	FLAWFINDER	RATS	CPPCHECK
CWE-327 Use of broken or risky cryptographic algorithm	✓	✗	✗
CWE-89 SQL Injection	✗	✓	✗
CWE-78 OS Command Injection	✗	✓	✓
CWE-79 Cross-site scripting	✗	✗	✗
CWE-120 Buffer Overflow	✓	✓	✓
CWE-369 Divide-by-zero	✗	✗	✓
CWE-476 Null pointer Dereference	✗	✗	✗
CWE-561 Dead Code	✗	✗	✓
CWE-401 Memory Leak	✗	✗	✓
CWE-190 Integer Overflow	✓	✗	✗

Table 2. Detection Ratio

TOOLS	Number of Vulnerabilities Detected	Detection Ratio
FLAWFINDER	52	52/118
RATS	84	84/118
CPPCHECK	59	59/118

Similarly, Table 3 , displays the comparative analysis results for the JAVA source code Static Analysis Tools. The JAVA source code of Apache TOMCAT server was fed as input to the two static code analysis tools(SpotBugs and PMD) and the output thus obtained, is furnished in table 2, with the help of a tick mark for detection of a particular category of vulnerability and a cross-mark for non-detection of vulnerability category by a particular tool.
The detection ratio of both the tools is thus calculated by counting the number of vulnerabilities detected by each of the tool. The result for detection ratio is furnished in table 4.

Table 3. Comparative analysis of JAVA static analysis tool

Categories of Vulnerability	SPOTBUGS	PMD
CWE-327 Use of Broken or risky cryptographic algorithm	✓	✗
CWE-79 Cross-Site scripting	✓	✗
CWE-259 Use of hard coded password	✗	✗
CWE-89 SQL Injection	✓	✗
CWE-476 Null-pointer Dereference	✓	✓

Table 4. Detection Ratio

TOOLS	Number of vulnerabilities detected	Detection ratio
SPOTBUGS	92	92/171
PMD	46	46/171

5. Conclusion and Future Direction

From the comparative analysis thus performed, it is concluded that, each static code analysis tool has its own set of pros and cons. Flawfinder and RATS are somewhat similar in type of vulnerabilities that they detect. Comparing the installation process of the two tools i.e. FlawFinder and RATS, the latter one was more difficult to install but it also managed to detect more number of vulnerabilities than FlawFinder. However, it was observed that CPPCheck detected maximum number of vulnerabilities out of the three tools. For Java static analysis tools, the comparative analysis shows that Find Bugs tool was able to find certain number of vulnerabilities that were not reported by PMD tool. A few of such vulnerabilities are equals() method fails on subtypes, clone method may return null, reference comparison of Boolean values, impossible cast, 32bit int shifted by an amount not in the range of 0-31, a collection which contains itself, equals method always returns true, an infinite loop.

In this study we have evaluated and compared widely used open source static code analysis tools for C/C++ programming language and JAVA programming language. From the study, it was observant that there are certain vulnerabilities present in the source code (that has been selected for the purpose of evaluation and comparison), that are not detected by either of the tools that we have selected. In future, we wish to work on designing a tool that can detect vulnerabilities missed by our selected tools. Such a tool would be helpful in detecting those categories of vulnerabilities that are not covered by open source static analysis tools that we have discussed in our study.

References

[1] C.Kunag, Q.Miao, and H.Chen, “Analysis of software vulneability,” WSEAS Transactions on Computers Research, vol 1, p.45, 2006.

[2] I.V.Krsul,”Software vulnerability analysis,” Purdue University, 1998

[3] W.Jimenez, A.Mammar, and A. Cavalli, “Software Vulnerabilities, Prevention and Detection Mthods: A Review1,” Security in Model-Driven Architecture, p. 6, 2009

[4] E.E.Schultz Jr, D.S. Brown, and T.A. Longstaff, “Respondingto cmputer security incidents: Guidelinesfor incident handling,” Lawrence Livermore National Lab., CA (USA) 1990.

[5] Common weakness Enumeration (CWE), website, URL: https://cwe.mitre.org/

[6] On analyzing static analysis tools”, National security Agency Centre for Assured Software, July 26, 2011, pp. 1-13

[7] https://samante.nit.gov/SARD/testsuite.php, 2019 on Emerging Security Information, Sytems and technologies, pp. 15-22, IEEE, 2009.

[8] Juliet Test suite for C/C++ (version 1.3)

[9] NIST-SARD, “Software assurance references dataset (SARD) testsuites, “ https://samante.nist.gov/SARD/testsuite.php ”, 2019

[10] APACHE TOMCAT

[11] Flawfinder [online] available:” https://www.dwheeler.com/flawfinder/”, 2019

[12] RATS [online] available: “https://security.web.cern.ch/security/recommendations/en/codetools/rats.shtml,”, 2019

[13] CPPCHECK [online] available: “: http://cppcheck.sourceforge.net/, 2019

[14] SPOTBUGS [online] available: “https://spotbugs.readthedocs.io/en/stable/”, 2019

[15] PMD [online] available: “https://pmd.github.io/latest/index.html”, 2019

[16] M.K. Brar, and P.J.Kaur, “Comparing detectio ratio of three static analysis tools” , vol. 124- no. 13,pp. 35-40, August 2015

[17] A.Arusoaie, S.Ciobaca, V.Craciun, D.Gavrilut, and D.Lucanu, “AComparison of static analysis tools for vulnerability detection in C/C++ code”, Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI UEFISCDI, project number PN-III-P2-2.1-BG-2016-0394, within PNCDI III.

[18] R.Mahmood, and Q.H. Mahmoud, “Evaluation of static analysis tools for finding vulnerabilities in JAVA nad C/C++ source code”, arXiv e-prints, abs/1805.09040,August 2018

[19] M.K. Brar, and P.J.Kaur, “Static Analysis tools for security: A Comparative Evaluation”, vol 5, issue 7, July 2015

[20] R.Amankwah, and P.K.Kudjo, “Evaluation of software vulnerability detection methods and tools: A Review”, vol 169, issue 8, pp. 22-27, July 2017