Evaluation of Static Analysis tools for Finding Vulnerabilities in C Programs

Helee Patel[1], Prof. Rishikesh Yeolekar[2]

[1]*Information Technology Department & MIT School of Engineering, Pune*
[2]*Information Technology Department & MIT School of Engineering, Pune*

---***---

Abstract - Software security is a critical topic that has been the focus of attention of many researchers and professionals over the years. Software security has evolved into an essential component of the software development process. An organization must maintain software security to assure the integrity, validity, and availability of the software product. One of the most important tasks in ensuring software security is identifying vulnerabilities in the source code before the product is released. Detecting vulnerabilities early in the software development cycle makes the process of resolving those vulnerabilities considerably easier for software engineers. Vulnerability detection can be done either during the production phase, when the software is still being produced, by statically auditing the source code, or dynamically during run time. Static code analysis tools detect vulnerabilities in code by identifying potential security problems and providing examples of how to fix them, some may even change the code to erase the vulnerability. This paper describes the analysis of many static code analysis tools and techniques available for vulnerability detection in C source code, as well as the analysis of some C static code analysis tools.

Key Words: cyber security, software security, vulnerability, C language, static analysis tools

1. INTRODUCTION

The security of software systems has become highly important in the global digitalized era. Confidentiality, Integrity, and Availability, or CIA principles are the fundamentals of computer security [8]. People's life is becoming increasingly dependent on software-intensive systems, and security bugs and vulnerabilities are becoming more frequent. The poor quality of source code is one of the sources of many security problems. A software security problem can provide unauthorized user access to a system, allowing it to behave improperly. Software security vulnerability would be a coding error occurred in the source code of software that may be exploited by an attacker to obtain unauthorized access to the software and force it to behave or operate improperly. As a result, the software development team must focus on identifying and resolving these vulnerabilities in the source code before deploying the software. The following types of security vulnerabilities exist in C programming languages [11, 12].

- SQL injection
- Format String Vulnerabilities
- Input Validation
- Command Execution
- Code Injection attacks
- Dynamic Memory Management
- Buffer overflow

Vulnerability detection is a technique for identifying vulnerabilities in software. Static and dynamic techniques are used in standard vulnerability detection. Static techniques, such as data flow analysis, symbol execution, and formal verification, analyze source code without allowing the software to be run. Static techniques offer a wide range of applications and may be used at any level of software development. However, it has a higher number of false positives. By running the program, dynamic techniques like fuzzy testing and dynamic symbol execution test reveal the nature of the software. Dynamic techniques have a low false-positive rate and are easy to deploy [4]. When static code analysis reveals vulnerabilities in the software development lifecycle, it simplifies the process of correcting such vulnerabilities for the software developer and lowers the cost and time spent by the organization to address those issues. Static code analysis can be performed manually or with the help of automated source code scanning software. Though manual analysis of source code by software developers was formerly widespread, the approach was time-consuming and did not provide very excellent results. Automated scanning tools were used to make the vulnerability detection process more efficient in terms of time and the number of vulnerabilities detected.

2. Static Code Analysis Tools

Different programming languages have their own set of vulnerabilities that must be solved. Different static code analysis tools are available to identify these vulnerabilities. Different open-source static analysis tools for the C programming language are discussed in this study. The static analysis tool used in this study is described in this section.

FLAWFINDER: Flawfinder is a code analysis tool that evaluates C/C++ code and reports potential problems in a risk-leveled manner [14]. Flawfinder has the benefit of being able to handle internationalized programs (special methods like gettext()), as well as reporting column and line numbers of finds. Flawfinder is updated and enhanced regularly, and there are several resources available to assist developers in using the program. In comparison to RATS, Flawfinder lags in terms of speed.

System Requirements: The Flawfinder command-line tool is only ready for installation and usage on Unix-like systems such as Linux, OpenBSD, or MacOS X.

Effectiveness: Many of SANS' Top 25 2011 list of most commonly occurring source code flaws may be detected using

Flawfinder, which is officially compatible with CWE (Common Weakness Enumeration) [13]. Flawfinder successfully discovered CWE-078: OS Command Injection, CWE-119: Failure to Constrain Operations within the Bounds of an Allocated Memory Buffer, and CWE-120: Buffer Copy without Checking Size of Input among the CWE error instances employed in this study. Flawfinder also found stack-based buffer overflows in addition to the flaws it promises to uncover. Although Flawfinder was unable to discover all of the source code flaws utilized in this investigation, it did deliver on its promises and found the flaws. However, Flawfinder produced a large number of false positives; even when a flaw was not there, it occasionally presented non-existent faults.

Ease of use: Flawfinder requires a decent understanding of CLI on Unix-like systems, but it also offers the required command-line inputs. Setup was not as difficult as with the RATS tool, and installation was swift thanks to the Flawfinder instructions, which can be obtained on the utility's official web page. Because loads of extra information on these mistake kinds are readily available online through other sources, Flawfinder's interoperability with CWE makes it particularly easy to grasp the fault types discovered.

Support: The Flawfinder website, instruction manual, and development are all updated regularly. Furthermore, as previously indicated, further information on the CWE error types that it identifies is available online, allowing a developer to quickly determine why an issue occurred and what can be done to prevent future mistakes.

RATS: Rough Auditing Tool for Security (RATS) is a tool that scans source code in C, C++, Perl, PHP, and Python. It also warns the user about common problems like buffer overflows and TOCTOU (Time of Check to Time of Use) race situations.

RATS is a highly valuable tool, although it only does a cursory study of the source code, as the name implies. This program will not identify all mistakes in the source code, and it may find "bugs" in the code that aren't truly faults. RATS is a scanning tool that produces a list of possible trouble places to target as well as a definition of the problems [6].

System Requirements: Only UNIX-based platforms are supported by the RATS command-line program, which is developed and ready to use.

Effectiveness: RATS found certain security vulnerabilities; however, it was unable to detect all known problems. All buffer overflow problems, insufficient control of resource identifiers, OS command injection, and inability to confine operations inside the bounds of an allocated memory buffer were successfully identified by RATS. RATS was unable to find any other flaws.

Ease of use: To use this software, you'll need to know how to utilize the Command Line Interface (CLI). A graphical user interface (GUI) would make this more user-friendly and accessible to non-technical people. After reading the README, the installation went rather well, although there were a few permission concerns when RATS sought to add files to usr/lib/bin, which is set to non-executable by most systems, including OSX. To complete the installation, you must be logged in as root. This is not mentioned in the README.

Support: The installation page should focus on the installation procedure since visitors must crawl through the README file to find out how to do so. Although instructions are given, setting up the system using only the README file's instructions proved difficult.

3. RELATED WORK

The online version of the volume will be available in LNCS Online. Members of institutes subscribing to the Lecture Notes in Computer Science series have access to all the pdfs of all the online publications. Non-subscribers can only read as far as the abstracts. If they try to go beyond this point, they are automatically asked, whether they would like to order the pdf, and are given instructions as to how to do so. Many researchers already had reviewed and compared various static analysis tools for vulnerability detection and found which static analysis tool is best placed for detecting vulnerabilities in a program written in a specific programming language. To examine each software for vulnerabilities discovered by them, the authors developed their C applications and introduced various vulnerabilities into them. Moreover, researchers presented a study comparing commercial static code analysis tools for discovering vulnerabilities in software source code.

To find potentially insecure code patterns, early static analyzers focused on basic syntactic principles (e.g., Flaw Finder, Linux utility grip with special regular expression). These simple tools have a significant false-positive rate and are unable to discover complicated vulnerabilities including code logic. Polyspace, Frama-C, and Astrée are examples of more complex static analyzers that utilize abstract interpretation to show the absence of runtime errors (RTEs). These tools are more suited to detecting safety flaws than security issues. Modern static analyzers, such as Fortify SCA and Coverity, make use of various methodologies that mostly depend on a knowledge base of previously identified security vulnerability patterns. To stay up with newly identified patterns, this knowledge base is updated regularly. These tools are useful, but they need a lot of time and effort to design and maintain, and they might overlook critical vulnerabilities that aren't yet included in their bases. Many new tools are being invented at the time, making it difficult to choose the best one [8].

For taint-style vulnerabilities, Fabian [13] proposed an unusual technique to detect them. It organizes the initialization of variables that can be passed on to security sensitive functions into groups. The detection system then reports the violation as a potential vulnerability. This method is appropriate for taint-style vulnerabilities, but not for other vulnerabilities. Kim [10] proposed a similarity based vulnerability detection method. This method is restricted to code cloning-related vulnerabilities. Bian [7] proposed a static analysis-based anomaly detection technique. He encodes the AST (Abstract Syntax Tree) with a hash algorithm after converting the program slice to an AST. Buffer overflow and IP fragmentation are two typical sources of system vulnerabilities that Krsul et al. investigated and discussed.

4. FUTURE SCOPE & CONCLUSION

Based on the results of the study, it can be stated that each static code analysis tool has its unique set of advantages and disadvantages. The kinds of vulnerabilities that Flawfinder and RATS discover are similar. When comparing the installation processes of the two programs, FlawFinder, and RATS, it was more difficult to install, but it detected more vulnerabilities than FlawFinder.

We analyzed and assessed generally used open-source static code analysis tools for the C programming languages in this study. According to the data, there are certain vulnerabilities in the source that are not discovered by any of the tools we used. In the future, we'd like to focus on making a tool that can detect vulnerabilities that our current tools ignore. A tool like this might be useful in finding vulnerabilities that aren't covered by the open source static analysis tools we described in our study.

REFERENCES

1. Sonnekalb, T., Heinze, T.S. & Mäder, P. Deep security analysis of program code. Empirical Software Engineering 27, 2 (2021)
2. Midya Alqaradaghi, Gregory Morse and Tamas Kozsik. Detecting security vulnerabilities with static analysis – A case study, An International Journal for Engineering and Information Sciences, September 30, 2021.
3. Code, Kaur A., Nayyar R., A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code, Procedia Computer Science, 171, pp. 2023-2029
4. X. Li, L. Wang, Y. Xin, Y. Yang, and Y. Chen, "Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning," Applied Sciences, vol. 10, no. 5, p. 1692, Mar. 2020.
5. Gadelha, M., Monteiro, F. R., Morse, J., Cordeiro, L., Fischer, B., Nicole, D., ESBMC 5.0: An Industrial Strength C Model Checker [In 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 888-891] 2018.
6. RATS - Rough Auditing Tool for Security. https://github.com/andrew-d/rough-auditing-toolforsecurity. Accessed May 23, 2018.
7. Bian, P.; Liang, B.; Zhang, Y.; Yang, C.; Shi, W.; Cai, Y. Detecting bugs by discovering expectations and their violations. IEEE Trans. Softw. Eng. 2018, 45, 984–1001.
8. Boudjema EH, Faure CL, Sassolas M, Mokdad L. Detection of security vulnerabilities in C language applications, Security and Privacy, December 2017;1:e8
9. Richard Amankwah, Patrick Kwaku Kudjo, Samuel Yeboah Antwi., Evaluation of Software Vulnerability Detection Methods and Tools: A Review. [In International Journal of Computer Applications, Vol.169- No.8], July 2017
10. Kim, S.; Woo, S.; Lee, H.; Oh, H. Vuddy: A scalable approach for vulnerable code clone discovery. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–24 May 2017
11. Subburaj Ramasamy, Anuj Singh and Deepak Singal. Enhancing the Security of C/C++ Programs using Static Analysis. [In Indian Journals of Science and Technology, Vol 9(44)], November 2016.
12. R. Subburaj, Pooja U. Raikar and S. P. Shruthi. Static Analysis of Security Vulnerabilities in C/C++ Applications. [In Indian Journals of Science and Technology, Vol 9(20)], May 2016.
13. Yamaguchi, F.; Maier, A.; Gascon, H.; Rieck, K. Automatic inference of search patterns for taint-style vulnerabilities. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015.
14. D. A. Wheeler, Flawfinder. August 2014. Accessed August 7, 2014. http://www.dwheeler.com/flawfinder/.
15. Cordeiro, L. C., Fischer, B., Marques-Silva, J. P. SMT Based Bounded Model Checking for Embedded ANSI-C Software. In IEEE Transactions on Software Engineering, v. 38, pp. 957-974, 2012.