ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(44), DOI: 10.17485/ijst/2016/v9i44/104031, November 2016* ISSN (Online) : 0974-5645

Enhancing the Security of C/C++ Programs using Static Analysis

Subburaj Ramasamy, Anuj Singh* and Deepak Singal

Department of Information Technology, SRM University, Kattankulathur-603203, Tamil Nadu, India; anujsingh_raj@srmuniv.edu.in, subburaj.r@ktr.srmuniv.ac.in, deepaksingal_subhash@srmuniv.edu.in

Abstract

Objectives: A vast multitude of application and systems programming is carried out in C or C++ programming languages. Even in programs written in languages such as Java, C libraries find wide use.Therefore, due to their ubiquitous presence, the security of C and C++ code is of paramount importance. **Methods/ Statistical Analysis:** A static analysis tool named "TraC++" was developed to detect security vulnerabilities in C and C++ programs. The tool uses a predefined and dynamically updated list of insecure coding constructs to check their presence in a given C/C++ code. **Findings:** The tool, developed in C#, was found to capture potential security vulnerabilities and insecure coding constructs in a given C/C++ program. A list of vulnerable constructs used in the code along with the line numbers in which they are present are the output provided by the tool. Furthermore, the tool provides suggestions as to how the vulnerable constructs can be replaced with better constructs. **Application/Improvement:** The tool can find use in static analysis for security violations in programs and libraries developed in the C/C++ programming languages.

Keywords: C/C++, Secure Coding, Security Vulnerabilities, Static Analysis

1. Introduction

As C and C++ programming languages facilitate compact and efficient programs that can access hardware easily, many application programs are created with C/C++. Furthermore, libraries of many popular languages such as Java, C# and Python might have been coded in C or C++. Due to their legacy, code written in C/C++ programming languages and their supporting libraries contain many security vulnerabilities. The presence of such security vulnerabilities empowers attackers.

These attackers aim their attacks at governments, corporations, banks, defence establishments, educational institutions, and individuals. Such attacks have resulted in the loss and compromise of sensitive data[1], system damage, lost productivity, and financial loss.

Computer Emergency Response Team (CERT)[2] located in the Software Engineering Institute, Carnegie Mellon University, USA supports the development of coding standards for programming languages including C/C++, Java, and Perl. These are based on community effort by members of software development and software security communities.

In this paper we discuss about Static Analysis in Section 2 and the Proposed Solution in Section 3. Summary and Conclusions are given in Section 4.

2. Need for Static Analysis

Building secure software is a challenge for software developers. A survey on security perspectives reveals that insecure software holds sixty percent of higher business threat than software which is secured. Compilers are not meant to check for security vulnerabilities. This creates the need for analysis tools which are able to capture such security vulnerabilities before run-time. This is where static analysis comes into play as it flags the security vulnerabilities in programs without actually executing

* *Author for correspondence*

them. Studies also reveal that secure software branches out significantly based upon the requirements, design and implementation[3–5].

However, it should be noted that static analysis tools can never outperform a good manual audit of the code. This is because there are too many variables in this type of analysis and it is impossible to include them all into an automated scan. Such tools can only detect vulnerabilities which are pre-programmed[6,7]. Static Analysis is the verification of application programs without actually executing them. Static analysis initially stems from compiler optimization. Currently, compilers focus on completing their function faster thus resulting in quicker development. In order to achieve faster return times, compilers only perform basic verification like type checking along with superficial syntactic checks. Static analysis sets its objective to finding errors which can occur at run-time before the program enters run-time. The advantage of spending more time on analysing the software decreases the amount of time spent in testing[8].

2.1 Security Vulnerabilities in C/C++

Security vulnerabilities are caused by software design and implementation practices which do not protect systems. Software vulnerabilities are being detected at the rate of over 4,000 per year[1].

C and C++ programming languages may have the following types of security vulnerabilities[9]:

- Integer Errors
- Code Injection attacks
- Buffer overflows
- Format String Vulnerabilities
- Pointer Subterfuge
- Dynamic Memory Management
- Race Conditions

Buffer overflows are commonly associated with C/C++ programs since these languages do not provide any built-in protection against accessing and overwriting of data in memory. Many C/C++ functions such as *gets*, *printf*, *strcpy*, *strcat*, etc. are susceptible to buffer overflows and thus compromise the security of the application.

In [10] explains how the pervasive *gets* function is almost always susceptible to buffer overflow attacks.

In[11] describes subtler buffer overflow problems caused by common C functions and even by their safer alternatives.

Buffer overflows can be prevented by taking measures such as writing secure code, preforming bound checking, static and dynamic code analysis and runtime code instrumentation[12].

If security vulnerabilities are left unchecked, mission critical systems created with the help of C/C++ could be filled with potentially fatal vulnerabilities. For example, one simple software error can cause the loss of an expensive mission (a NASA mission to Mars could cost at least 250 million USD)[8].

CERT Division helps resolve software vulnerabilities by developing tools and methods to analyse vulnerabilities. With the help of research and analysis, CERT develops practical and applicable solutions to relevant problems[13].

3. Proposed Solution

We have developed a static analysis tool, named "TraC++". This tool can capture various security vulnerabilities in a given program.

The developed static analysis tool uses the string and pattern matching approach. It provides immediate feedback even for very large programs. The tool uses a flexible, dynamically editable repository of vulnerable constructs. This repository is saved on disk and can be easily modified thus making the process of adding and removing vulnerable constructs easier.

Furthermore, to make this solution universal, avid programmers can release dictionaries of vulnerable construct names which can act as a basis for the repository. This would be similar to the Common Vulnerabilities and Exposures (CVE) which is a dictionary for cybersecurity vulnerabilities[14].

The tool then parses the given code and verifies it against the vulnerability repository, if any vulnerabilities are found they are reported in the output field along with their line numbers. Also, for well-known vulnerabilities, the tool provides suggestions containing other programming constructs with which the vulnerable construct can be replaced.

The activity Diagram of the developed tool is given in Figure 1.

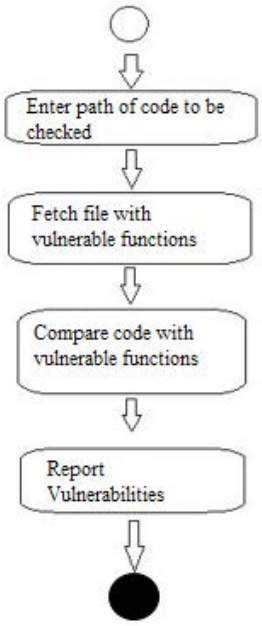

Figure 1. Activity diagram of static analysis Tool, TraC++.

A snap shot of the tool is given in Figure 2.

The tool is user friendly and provides a graphical user interface that facilitates ease of use [15] discusses the effectiveness of various types of static analysis approaches including the string and pattern matching approach.

4. Conclusions

In this paper, we have provided an overview of security vulnerabilities in C/C++ and described the tool "TraC++", a static analysis tool based on the string and pattern matching approach, developed by us. This tool prevents security vulnerabilities from being released in C/C++ application programs. This approach saves the time of the developer and ensures that only secure products are delivered to the end users of the applications. The tool was tested on a number of programs and all the vulnerabilities listed in the repository were captured from the programs successfully, if present. Usage of tool in every project involving development of programs written in C/C++ will enhance security of the Computer Systems.

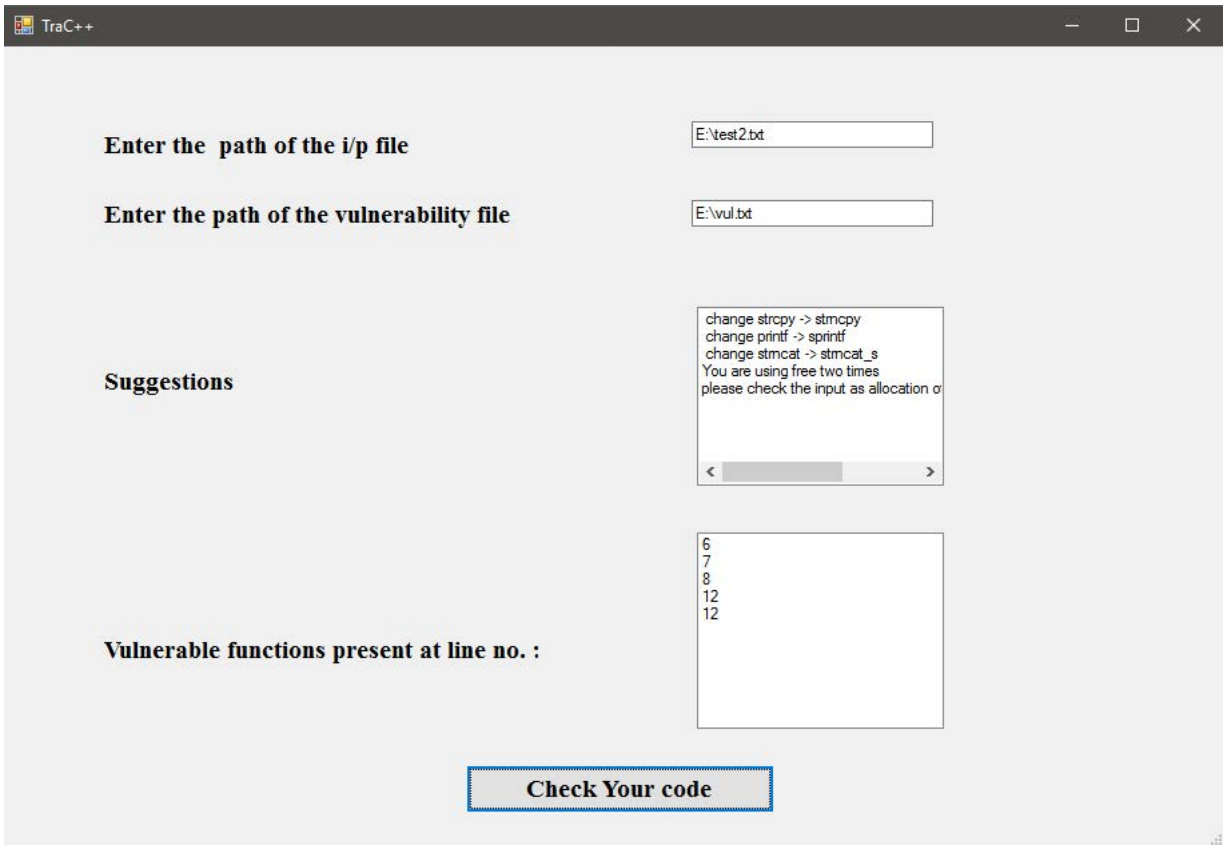

Figure 2. Sample output.

5. References

1. Seacord RC. Secure coding in C and C++. 2nd Ed, USA: Addison-Wesley Professional; 2005.
2. SEI CERT Coding Standards [Internet]. 2016 [cited 2016 Oct 06]. Available from: https:// www.securecoding.cert.org.
3. Nikhat P, Kumar NS, Khan MH. Model to quantify integrity at requirement phase. Indian Journal of Science and Technology. 2016 Aug; 9(29):1–5.
4. Flechais F, Sasse M, Hailes SMV. Bringing security home: A process for developing secure and usable systems. NSPW'03. ACM. USA; 2003 Aug. p. 18–21.
5. Madan BB, Popstojanova KG, Vaidyanathan K, Trivedi KS. A method for modeling and quantifying the security attributes of intrusion tolerant system. An International Journal of Performance Evaluation. 2004; 56(1):167–86.
6. Source code scanners for better code [Internet]. [cited 2016 Oct 06]. Available from: http://www.linuxjournal.com/article/5673?page=0,2.
7. Revnivykh AV, Fedotov AM. Root causes of information systems vulnerabilities. Indian Journal of Science and Technology. 2015 Dec; 8(36):1–6.
8. Brat G, Venet A. Precise and scalable static program analysis of NASA flight software. Proceedings of the IEEE Aerospace Conference India; 2005. p. 1–11.
9. Subburaj R, Raikar PU, Shruthi SP. Static analysis of security vulnerabilities in C/C++ applications. Indian Journal of Science and Technology. 2016 May; 9(20):1–4.
10. Viega J, Bloch JT, Kohno T, Graw MG. ITS4: A static vulnerability scanner for C and C++ code. ACM Transactions on Information and System Security. 2002; 5(2):1–11.
11. Wagner D, Foster J, Brewer E, Aiken A. A first step towards automated detection of buffer overrun vulnerabilities. Proceedings of the Year Network and Distributed System Security Symposium (NDSS), San Diego, CA; 2000. p. 3–17.
12. Fu D, Shi F. Buffer overflow exploit and defensive techniques. IEEE International Conference on Multimedia Information Networking and Security (MINES); Nanjing, China; 2012. p. 87–90.
13. About CERT [Internet]. [cited 2016 Ot 07]. Available from: https://www.cert.org/about/.
14. Common Vulnerabilities and Exposures [Internet]. [cited 2016 Oct 06]. Available from: http://www.cve.mitre.org.
15. Emanuelsson P, Nilsson U. A comparative study of industrial static analysis tools. Electronic Notes in Theoretical Computer Science. 2008 Jul; 217:5–21.