Indian Journals of Science and Technology, Vol 9(20), DOI: 10.17485/ijst/2016/v9i20/91312, May 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Static Analysis of Security Vulnerabilities in C/C++ Applications

R. Subburaj, Pooja U. Raikar* and S. P. Shruthi

Department of IT, SRM University, Kattankulathur - 603203, Tamil Nadu, India;
subburaj.r@ktr.srmuniv.ac.in, pooja_raikar@srmuniv.edu.in, shruthi_prakash@srmuniv.edu.in

Abstract

Background/Objectives: With ever increasing number and variety of security threats looming large that continually cause hassles to companies and governments, it is vital to ensure that the software applications are free from such vulnerabilities. The objective is to remove such vulnerabilities in applications coded in C/C++ programming languages conforming to ISO/IEC standards, through static analysis and make the applications secure. **Methods**: Collected insecure coding constructs in C/C++ programming languages from authentic sources and created a repository of the same. Built a static analysis tool named "Vulnerability Reporter" to flag insecure coding constructs in the applications. The insecure coding constructs are identified by referring to the repository of vulnerabilities in C/C++ languages, prewritten to the tool. **Findings:** The tool parses the code and identifies and provides a report containing the vulnerable code in the given application along with their locations. It also provides suggestions for improvement of each potential vulnerability identified by the tool. The tool is scalable. **Implementation/Application**: The tool developed will find immense use in the academia and industry and will thereby enhance the security of application.

Keywords: Static Analysis, Dynamic Analysis, Proposed Solution

1. Introduction

Cyber-attacks have become pervasive causing hardship to the internet users - individuals and organizations, both government owned or private. A study carried out in China[1] reveals that software developed by start-up companies has more serious security issues than other organizations, government and educational institutes have attracted more attention from the attackers. It has been found that most of them are due to vulnerabilities of the code. Avoiding insecure coding practices in the initial stages of Software Development Life Cycle (SDLC) can minimize the time and effort spent on finding and fixing them in later stages, as well as the losses to humanity on this account. According to the 2014 report by the Ponemon Institute[2], the mean annualized cost of a cybercrime for 257 benchmarked organizations was $7.6 million per year, with average of 31 days to contain a cyber-attack. Another survey[3] relies on information in research articles and manuals and includes the types of defects checked for (such as memory management, arithmetic, security vulnerabilities), soundness, value and aliasing analyses and IDE integration. This survey is complemented by practical experiences from evaluations at the Ericsson telecom company. Knowledge about the presence of security vulnerabilities in a programming language does not solve the issue completely, unless the developer remains security conscious during all the phases of software developement[4].

C and C++ programming languages are preferred when it comes to performance and efficiency. Security vulnerabilities in C and C++ programming languages can be classified as follows:

1. Integer vulnerability
2. Buffer overflow
3. String vulnerability
4. Pointer subterfuge

*Author for correspondence

5. Dynamic memory management
6. Race conditions

There are two ways of detecting security vulnerabilities in a program.

1.1 Static Analysis

Static Analysis is carried out statically without executing the code. In this method the tool inspects input program code for possible security vulnerabilities and alerts the user.

1.2 Dynamic Analysis

Dynamic Analysis is carried during the execution of a program. Dynamic analyser monitors system memory, functional behaviour, response time and overall performance of the program to find security vulnerabilities.

[5]Proposes dynamic taint propagation techniques which facilitate finding input validation errors using functional tests. It has been found that static analysis is more suitable[6] in finding vulnerabilities. Static analysis tools are designed to detect security vulnerabilities in software code that, if not corrected, might lead to exploitable security vulnerabilities. The tools built for static analysis can be used to find runtime errors as well as resource leaks and security vulnerabilities statically, i.e. without executing the code.

[7]Presents a game to educate the users in various computer security concepts in which participants are given set of questions to solve.

[8]Reports that there is still a need for improvement in the static analysis tools. This paper proposes a method and a tool which statically analyses programs coded in C and C++ to find security vulnerabilities and suggest alternative secure coding constructs for the same.

[9]Identifies different security issues and different type of attacks in cloud based e-learning system and mitigation strategies for same.

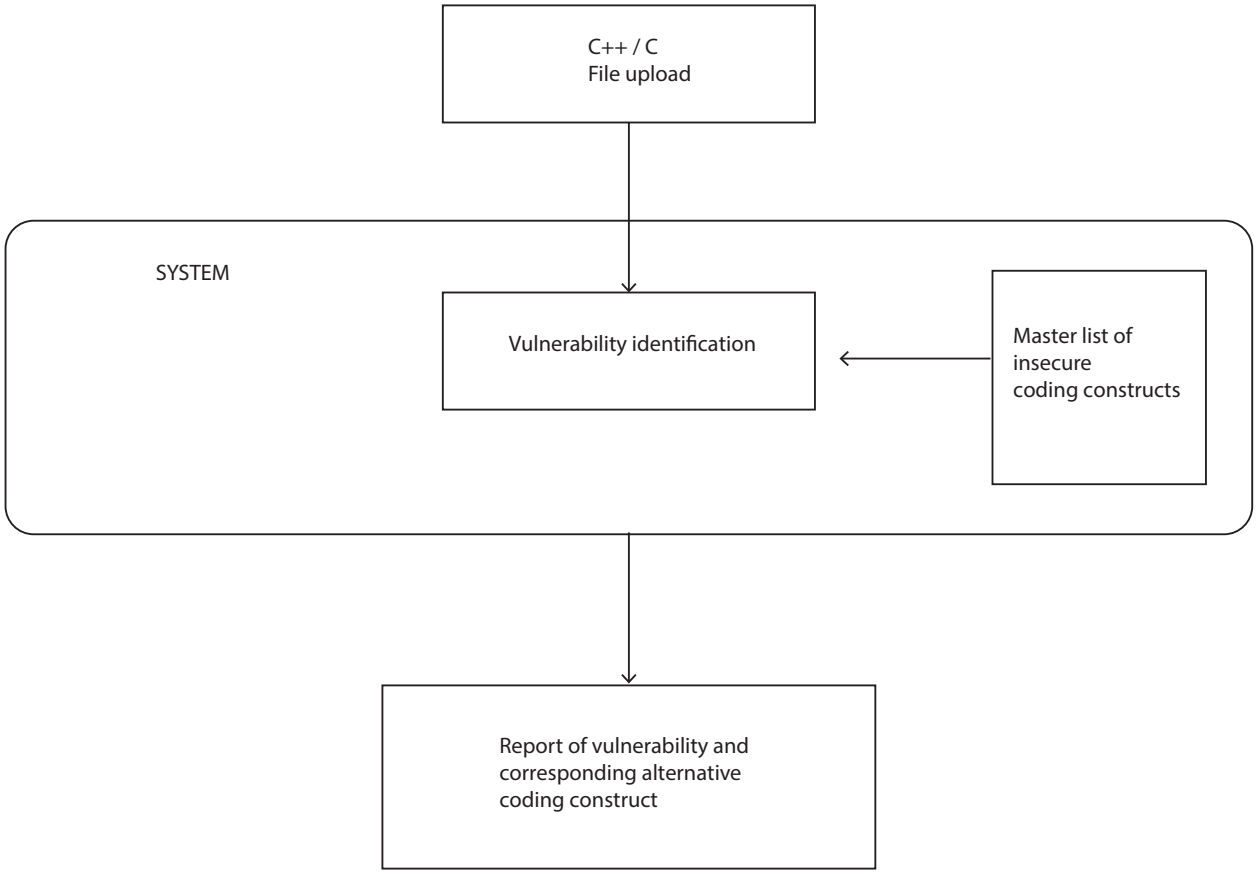

Figure 1. System architecture design.

2. Proposed Solution

Proposed method is intended to let user upload the input program file to be inspected for security vulnerabilities. It inspects the uploaded input source code program file and alerts the user about presence of vulnerabilities if any.

The developed tool maintains a dynamically updatable list (master list) of insecure coding constructs[10,11] which may lead to security vulnerability in any program written in C and C++ languages, along with the corresponding alternative secure coding constructs[12]. As the tool inspects the input program, it checks for the presence of insecure code or functions or keywords from the master list, if found a report is generated listing the insecure coding constructs present in the input program and also providing alternate secure coding constructs that can be used[13]. The tool is developed to overlook vulnerabilities if present in commented section to avoid false positives, thus avoiding false negatives. The architecture of the tool "Vulnerability Reporter" developed by our team is given in Figure 1.

The source code is given as input to the tool as illustrated in Figure 1. The tool may be given the code after clean compilation.

Screenshot in Figure 2 shows the sample output of tool "Vulnerability Reporter" developed by our team:

The tool[14] flags errors when functions which do not carry out input validations such as memcpy() , strcpy() are used in the program.

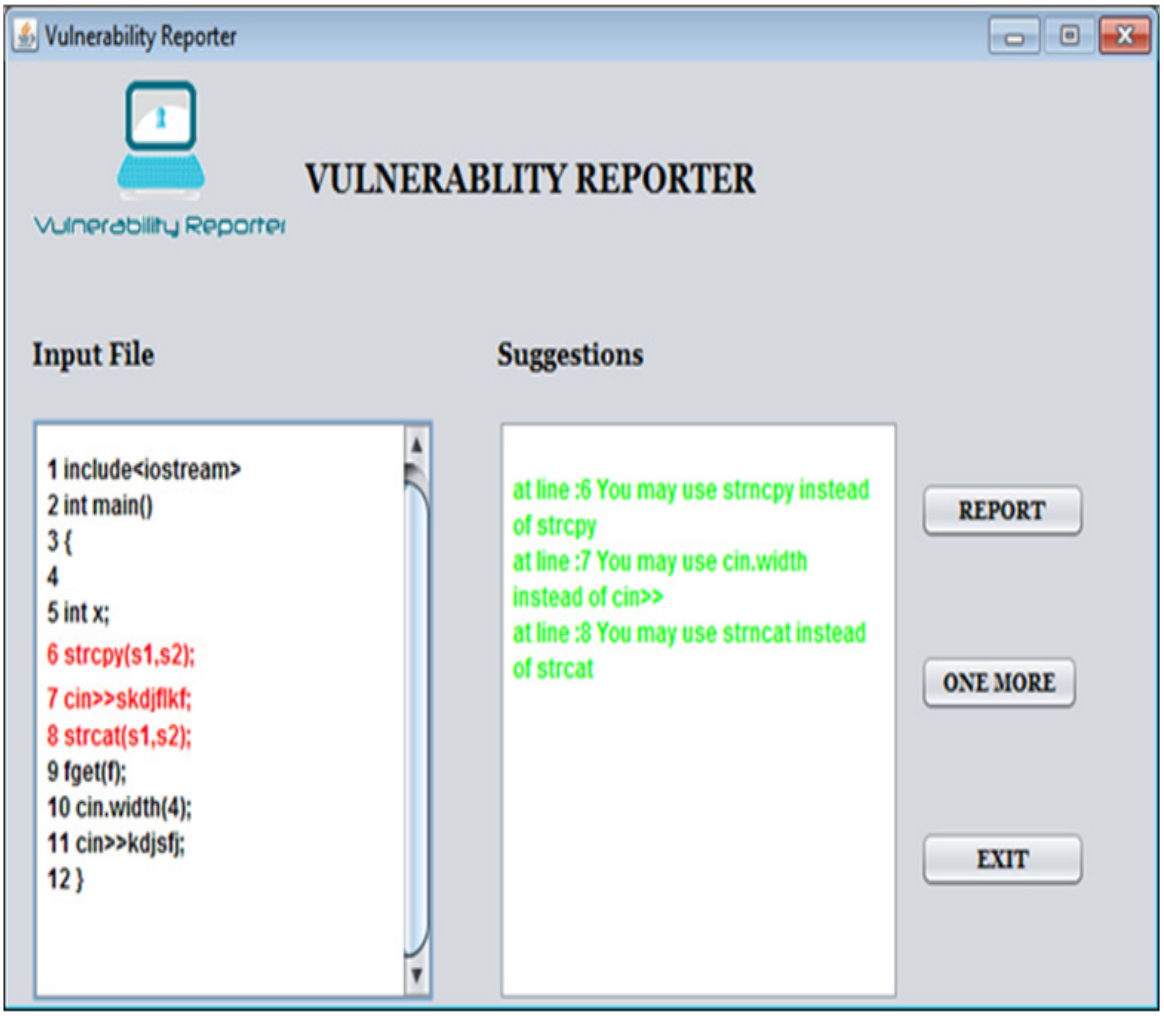

Figure 2. Sample output.

3. Conclusions

In this paper we highlight the problems of insecure coding in software applications and briefly present features of a tool developed by us that analyses the given C and C++ code for security vulnerabilities and generate a report of the vulnerabilities present along with suggestions for improving the same. The tool was put to use extensively and the results confirm that there is a significant improvement over manual code review. Since a large number of security incidents occur of late because of insecure applications, the work presented here has profound implications for the future and may be used in peer reviews of software applications.

4. References

1. Huang C, Liu JY, Fang Y, Zuo Z. A study on Web security incidents in China by analyzing vulnerability disclosure platforms. Computers and Security. 2016 May; 58:47–62.
2. Díaz G, Bermejo JR. Static analysis of source code security: Assessment of tools against SAMATE tests. Information and Software Technology. 2013 Aug; 55(8):1462–76.
3. Emanuelsson P, Nilsson U. A comparative study of industrial static analysis tools. Electronic Notes in Theoretical Computer Science. 2008 Jul; 217:5–21.
4. Viega J, Bloch JT, Kohno Y, McGraw G. ITS4: A Static Vulnerability Scanner for C and C++ Code. 16th Annual Conference, IEEE; New Orleans, LA. 2000 Dec. p. 257–67.
5. Brian C, Jacob W. Dynamic taint propagation: Finding vulnerabilities without attacking. Elsevier Information Security Technical Report. 2008; 13(1):33–9.
6. Seacord RC. Secure coding in C and C++. 2nd Ed. USA: Addison-Wesley Professional; 2005.
7. Boopathi K, Sreejith S, Bithin A. Learning cyber security through gamification. Indian Journal of Science and Technology. 2015 Apr; 8(7):642–9.
8. Katerina GP, Andrei P. On the capability of static code analysis to detect security vulnerabilites. Elsevier Information Security Technical Report. 2015 Dec; 68:18–33.
9. Durairaj M, Manimaran A. A study on security issues in cloud based E-learning. Indian Journal of Science and Technology. 2015 Apr; 8(8):757–65.
10. MISRA C++ 2008 Guidelines. 2008. Available from: http://www.misra-pp.com/activities/misrac/tabid/171/default.aspx
11. MISRA C rules. 2002. Available from: http://www.embedded.com/electronics-blogs/beginner-s-corner/4023981/Introduction-to-MISRA-C
12. SEI CERT Coding Standards. 2016. Available from: https://www.securecoding.cert.org
13. Blazy S, Buhler D, Yakobowski B. Improving static analyses of C programs with conditional predicates. Science of Computer Programming. 2016 Mar; 118:77–95.
14. Kamara S, Fahmy S, Schultz E, Kerschbaum F, Frantzen M. Analysis of vulnerabilities in Internet firewalls. Elsevier Computers and Security. 2003; 22(3):214–32.