THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

Code Hound – Security Analyzer in C/C++ Programs

Akash Govind
Student, Department of Information Technology, SRM University, Kattankulathur, Chennai, Tamil Nadu India
Avinash Kumar Verma
Student, Department of Information Technology, SRM University, Kattankulathur, Chennai, Tamil Nadu India
Radha Singhal
Student, Department of Information Technology, SRM University, Kattankulathur, Chennai, Tamil Nadu India
Akshay Singh Kanwar
Student, Department of Information Technology, SRM University, Kattankulathur, Chennai, Tamil Nadu India

Abstract:
The exponential development in the field of Information Technology has led to the enormous increase in financial, personal and statistical data which, if compromised, can cause irreparable loss. Hackers exploit the vulnerabilities in the application code to use this data to their advantage. One of the best ways to identify the exploits in any software application is to use static analysis. This paper addresses some well- known vulnerabilities found in the C and C++ programs that can be used by an attacker to conduct malicious actions and, the paper also details about statically analyzing the vulnerabilities in the C/C++ code.

Keywords: *C/C++, vulnerability, attack, program.*

1. Introduction

C/C++ is one of the foundations for modern information technology (IT) and computer science (CS).
Many working principles of IT and CS, such as programming languages, computer architectures, operating systems, network communication, database, graphical user interface (GUI), graphics, image processing, parallel processing, multi-threads, real-time systems, device drivers, data acquisition, algorithms, numerical analysis, and computer game, are based on or reflected in the functionalities and features of C.
The programming languages such as C/C++ suffer from memory management and security of code especially when their codes are used in critical systems. Therefore, we need an efficient mechanism to detect memory and type errors (Mcheick, Dhiab, Dbouk & Mcheik, 2010).
Even though C/C++ is an excellent programming language, there are many security considerations on coding. C is a powerful, robust language build with strong security features. The paper is structured as follows: Section II presents literature survey discussing the vulnerabilities when programming. In section III, a scheme is proposed for statically analyzing a C/C+++ program and Summary and Conclusions are given in Section IV.

2. Literature Survey

There are numerous ways through which a hacker can compromise the security of a system and access private and personal data. With years of research, computer scientists were successful to figure out several vulnerabilities in programming languages like C and C++. Of all vulnerabilities identified in computer applications, problems caused by unchecked input are recognized as being the most common (Livshits & Lam, 2005). Many solutions were suggested to avoid security breaches. These solutions intended to protect the user by hampering access to some platform constructs and APIs which could be exploited by malware, such as accessing the local filesystem, running arbitrary commands, or accessing communication networks. But not all solutions were effective enough to stop the hackers from running malicious codes, exposing the user to several attacks like D-DOS, DOS etc. Soon security related alternative methods for known exploits started to develop, but even these alternatives were not attained with success because the user didn't use them promptly (Viega, Mutdosch, & McGraw, Felten, 2000). Surveys and consensus showed that lack of awareness by many users about the vulnerability itself also, how to remove them.

2.1. Reasons for Vulnerabilities in C/C++

According to CERN Computer Security, most vulnerabilities in C are related to buffer overflows and string manipulation. In most cases, this would result in a segmentation fault, but specially crafted malicious input values, adapted to the architecture and environment could yield to arbitrary code execution (CERN Computer Security Team, 2016).

2.2. Programming faults in C/C++ (Seacord, 2005)

1. Buffer overflow vulnerability

Unbounded string copies occur when data is copied from an unbounded source to a fixed length character array (for example, when reading from standard input into a fixed length buffer). Example,

```
#include <iostream>
void main(void)
{
   char Password[8];
         puts("Enter 8 character password:");
         gets(Password);
}
```

The above code may lead to out of bound array indexing and also buffer overflow. Due to the buffer overflow, the C/C++ programs provide illegal privileges to the hacker. Attackers can run arbitrary codes to conduct malicious actions.

2. Blocks without braces vulnerability

This vulnerability occurs when the condition blocks like if, else, while, try, catch are not within the braces. Example, *if (condition == true) // execute code*

The attacker may create a situation such that instead of running the content inside if block, the content in the else block gets executed, which is called resource tampering attack.

3. Unbounded string copy- vulnerability

This vulnerability arises when a programmer tries to copy data from once location to another location where the source is larger than the destination. Example,

```
#include <iostream>
int main(int argc, char *argv[])
{
   char name [20];
   char name1[21];
         gets(name);
         gets(name1);
         strcpy(name, name1);
         return 0;
}
```

Here, a character array name of size 20 bytes can be overflown using character array of size 21 bytes.

4. Unexpected behavior of for loop

Unexpected execution condition of for loop leads to this vulnerability. Example, *for (int i = 0 ; ; i++) { // code }*

This leads to infinite execution of for loop.

5. Zero-length arrays vulnerability

The zero length array vulnerability arises when an empty array is declared in the program, i.e., an array with zero length size is executed. Example,

```
int arr1 = new int[0];
int arr1 = new Int[]
{ };
```

The above code snippet results in unreleased resources attacks.

6. Negative length arrays vulnerability

Negative length array vulnerability might occur when the array is initialized with a value smaller than zero, or the array length goes to negative value because of negligent computations in the program. Example, int [ ] temp = new int [-1];

This can lead to buffer overflow or integer overflow leading to a denial of service attacks

7. Rethrowing exception vulnerability

Rethrowing the exception multiple times causes a denial of service attacks. Example, *catch(whatever exception e) {throw e;}*

8. Abrupt program termination vulnerability

This vulnerability might occur when the function exit() is used to abruptly exit the program. This can lead to DOS attacks.

9. Return null vulnerability
This vulnerability occurs when null is returned by a function. Example, *return null;*
The above may result in abnormal program termination when the calling function performs operations on null and results in denial of service attack.

2.3. Attacks due to flaws in the code (Younan,2012) (Constatine, 2013) (Abdullah, Aklima, Håkan & Robert, 2010) (Rutar, Almazan & Foster, 2004)
10. Denial of service attack
The gets( ) method has no upper bound on the number of characters to be read from standard input. As a result, the attacker can easily overflow the buffer and creates a possibility of overflowing the memory causing a DOS attack.

11. Path manipulation Attack
The path manipulation attack arises when the user input is directly embedded into the program and executed, without validating the correctness and appropriateness input. The attack becomes more critical if the input values are directly embedded into path statements that read a critical system resource (example, a file) and the program executes them with elevated privileges.

3. Proposed Approach
Need for a static analyzer seems crucial and is highly recommended. The root cause behind the exploits in any language is the lack of awareness and negligent programming. Hence, a static analyzer analyzes a piece of code supplied to it and hence detects security vulnerabilities which can be avoided statically. (Chess &West, 2008)
This paper proposes a static analyzer called CodeHound, version 1.0, developed in C# Windows Forms for finding vulnerabilities caused by unchecked inputs and detects all vulnerable functions. CodeHound reads a C/C++ program as an input as a file. It then checks the existence of the vulnerable functions in the program. In this world of continuous evolution, one extra feature which makes the software unique is the ability to dynamically add vulnerability rules and their alternatives, so as to compensate for the C/C++ functions which may be proved to be vulnerable in future. These vulnerabilities are stored in a database with their corresponding alternatives. Finally, if any vulnerability exists in the program among the pre-listed vulnerabilities or dynamically added vulnerabilities, then the system prompts the user about the vulnerability and suggests an alternative.

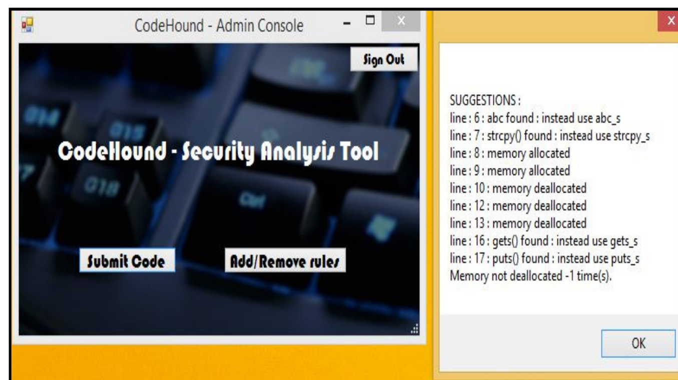

Figure 1: Prototype for the static analyzer.

This tool, as shown in Figure 01, finds all potential matches statically.
CodeHound reads the input, i.e., the C/C++ code line by line and comparing every word with that in the database and pre-listed vulnerabilities. Upon comparison, all the vulnerable methods used in the program are listed with the alternatives. If all the vulnerabilities are replaced by their alternatives, the program's security is enhanced.

4. Summary & Conclusions
The key conclusion of the entire study can be packed down to one sentence: exploits cannot be completely brought down to zero by releasing patches or writing new alternative secure functions. Users usually ignore updates and cyber criminals, knowing that C/C++ is used by an enormous number of people and enterprises, never miss a chance of catching a vulnerability in the software so they can exploit them.
The major benefit of static code analysis is that it can help the developer to figure out vulnerabilities in a program early in development cycle which helps to minimize the cost. Furthermore, static analysis is one of the safest options to have control over the attackers.

5. Acknowledgment
The first author takes this opportunity to express his profound gratitude and deep regards to all the professors for their exemplary guidance, monitoring, and constant encouragement and also obliged to all staff members of SRM University, Chennai, for their valuable guidance.

6. References

i. Mcheick, H., Dhiab, H., Dbouk, M., & Mcheik, R. (2010). Detecting Type Errors and Secure Coding in C/C++ Applications. Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications, 1-9.

ii. Benjamin Livshits, V., Lam, M S. (2005). Finding Security Vulnerabilities in Java Applications with Static Analysis. Proceedings ISSYM'05 proceedings of the 14th conference on USENIX Security Symposium. 1-18.

iii. Viega, J., Mutdosch, T., & McGraw, G., Felten, E. W. (2000). Statically Scanning Java Code for Security Vulnerabilities. 14th USENIX Security Symposium.

iv. CERN Computer Security Team. (2016). Retrieved from https://security.web.cern.ch/security/recommendations/en/codetools/c.shtml

v. Seacord R. (2005). Secure coding in C and C++: Strings. Retrieved from http://www.informit.com/articles/article.aspx?p=430402&seqNum=2

vi. Younan, Y., Security research program, Research in motion. (2012). C and C++: vulnerabilities, exploits and countermeasures. Retrieved from https://handouts.secappdev.org/handouts/2012/Yves%20Younan/C%20and%20C++%20vulnerabilities.pdf

vii. Constatine., L. (2013). Most Business network riddled by vulnerable JAVA installation. Retrieved from http://www.pcworld.com/article/2044623/most-enterprise-networks-riddled-with-vulnerable-java-installations-report-says.html

viii. Abdullah, Al M., Aklima, K., Håkan, K., & F. Robert. (2010). Comparing Four Static Analysis Tools for Java Concurrency Bugs. Blekinge Institute of Technology, Karlskrona, Sweden.

ix. Rutar, N., Almazan, C.B., & Foster, J.S. (2004). A Comparison of Bug Finding Tools for Java. Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium.

x. Chess, B., & West, J. (2008). Secure Programming with Static Analysis. (1st ed.). Boston, MA, USA: Addison-Wesley.