An Implementation of a Tool to Detect Vulnerabilities in Coding C and C++

GRADUATE PROJECT REPORT

Submitted to the Faculty of
The School of Engineering & Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Pradeep Kumar Pothonak
Spring 2012

Committee Members

Dr. Mario Garcia
Committee Chairperson

Dr. Long-zhuang Li
Committee Member

Dr. David Thomas
Committee Member
ABSTRACT

Programming is prone to errors since it is based on some pre-defined syntax and pertaining rules. For a successful execution of the program, it must be free from all the compilation errors. The C programming language is intended to be a lightweight language with a small footprint. This characteristic of C leads to vulnerabilities when programmers fail to implement required logic because programmers assume it is handled by C programming language (but it is not).

This problem is magnified when programmers are familiar with superficially similar languages such as Java, Pascal, or Ada leading them to believe that C protects the programmer better than it actually does. These false assumptions have led the programmers to write the code beyond the boundaries of an array, failing to catch integer overflows and truncations, and calling functions with the wrong number of arguments. With the help of this project it overcomes the vulnerabilities while coding in C and C++.

In this project secure coding software is proposed. It is a desktop application which traces the errors and vulnerable issues within the code, when the C or C++ code is imported in to this application. The brief description of the vulnerable issues will be the output with clear information of error and line number which has error/issue in the code. This application also gives suggestive replaceable code by which code will become secure. Thus this application would be helpful to overcome vulnerable attacks.
# TABLE OF CONTENTS

ABSTRACT .................................................................................................................... ii

TABLE OF CONTENTS ................................................................................................ iii

LIST OF FIGURES ......................................................................................................... vi

1. INTRODUCTION ........................................................................................................ 1

1.1 Strings ...................................................................................................................... 1

1.2 Dynamic Memory Allocation .................................................................................... 3

1.3 Formatted Output .................................................................................................... 5

1.4 Pointers ................................................................................................................... 8

1.5 Integers ..................................................................................................................... 9

2. NARRATIVE ................................................................................................................ 11

2.1 Problem Statement ................................................................................................ 11

2.2 Motivation .............................................................................................................. 11

2.3 Functionality of the project .................................................................................... 11

3. PROPOSED SYSTEM DESIGN ............................................................................... 12

3.1 Secure coding mechanism in C and C++ ................................................................ 12

3.2 Framework ............................................................................................................. 13

3.2.1 Steps in the project development ....................................................................... 13

3.2.2 System requirements .......................................................................................... 14
3.3 Proposed Mechanism ........................................................................................................ 14

3.3.1 Working ......................................................................................................................... 15

3.4 System design..................................................................................................................... 15

3.4.1 Class diagram ................................................................................................................. 16

3.4.2 Use case diagram .......................................................................................................... 17

3.4.3 Sequence diagram ......................................................................................................... 18

3.4.4 State chart diagram ...................................................................................................... 19

4. FUNCTIONALITIES OF THE APPLICATION ................................................................. 21

4.1 Step wise description of functionality............................................................................. 21

4.1.1 Initializing Code analyzer ............................................................................................ 22

4.1.2 Importing C code in to application ............................................................................. 23

4.1.3 Running the application ............................................................................................... 25

4.1.4 Vulnerabilities detection ............................................................................................... 27

4.1.5 Suggesting Invulnerable code ..................................................................................... 28

5. TESTING AND IMPLEMENTATION ................................................................................. 29

5.1 Testing .............................................................................................................................. 29

5.2 Types of Testing ............................................................................................................... 29

5.2.1 White box testing ......................................................................................................... 30

5.2.2 Black box testing .......................................................................................................... 30

5.2.3 Unit testing .................................................................................................................. 30
5.2.4 Integration testing ................................................................. 31
5.2.5 Validation testing ................................................................. 31
5.2.6 Output testing ................................................................. 32
5.2.7 User acceptance testing .................................................. 32
5.3 Test cases .............................................................................. 32
5.3.1 Testing with strings in C and C++ code ......................... 33
5.3.2 Testing with integers in C and C++ code ......................... 34
5.3.3 Testing with Dynamic Memory functions in C and C++ code .............. 38
5.3.4 Testing with Formatted output in C and C++ code ............... 40
5.3.5 Testing with pointers in C and C++ code ......................... 42

6. CONCLUSION AND FUTURE WORK ............................................. 44

BIBLIOGRAPHY AND REFERENCES ............................................. 45
LIST OF FIGURES

Figure 1: Proposed system’s class diagram ............................................................. 16
Figure 2: Proposed system’s use-case diagram ...................................................... 17
Figure 3: Proposed system’s sequence diagram ..................................................... 18
Figure 4: Proposed system’s state chart diagram ................................................... 19
Figure 5: Initialization of code analyzer ................................................................. 22
Figure 6: Importing code in to application ............................................................. 23
Figure 7: Selecting the file for importing ............................................................... 24
Figure 8: Running the application to check vulnerable issues .............................. 25
Figure 9: Running the application in command prompt ......................................... 26
Figure 10: Vulnerabilities detected in C code ......................................................... 27
Figure 11: Suggestive and replacable code ............................................................ 28
Figure 12: Interface for testing string functions ................................................... 33
Figure 13: Test result with vulnerable issues with strings .................................... 34
Figure 14: Interface for login to integer analyzer .................................................. 35
Figure 15: Interface for testing integer operations ................................................ 36
Figure 16: Test result with vulnerable issues with integers ................................... 37
Figure 17: Interface for testing dynamic memory allocation .................................. 38
Figure 18: Test result with vulnerable issues in dynamic memory allocation ....... 39
Figure 19: Interface for testing formatted output .................................................. 40
Figure 20: Test result with vulnerable issues with formatted output .................... 41
Figure 21: Interface for testing pointers ............................................................... 42
Figure 22: Test result with vulnerable issues with pointers ................................... 43
1. INTRODUCTION

Security has become really important to consider while coding. It is no longer acceptable to assume that a program will operate normally with a given range of expected inputs because an attacker will look for input values that produce an abnormal effect. A software vulnerability may result when a program evaluates as an integer, string, pointer, dynamic memory and formatted output to an unexpected value (that is, a value other than the one obtained with pencil and paper) and then uses the value as an array index, size, or loop counter.

Because data type range checking has not been systematically applied in the development of most C and C++ software systems, security flaws involving those data type are certain to exist, and some portion of these are likely to be vulnerabilities.

The software vulnerabilities mainly occur in the integers, strings, dynamic memory, formatted output and pointers. So providing security in the mentioned data types has become really important.

1.1 Strings

Strings form the main part of programming in C and C++. Vulnerabilities in programming might result in catastrophic outcomes. Any vulnerability in the string type variables can lead to many attacks. Strings are not inbuilt in C or C++ usually and hence only the character arrays are considered as contiguous blocks thus forming strings of characters. Attackers usually find the vulnerable points in the code to inject different type of malicious codes that might interrupt the usual flow of execution of the program.
Programmers usually tend to make mistakes while writing the code. Small ignorable errors can be used by the attackers to stop the functioning of the software. Applications or the software can be brought down completely even if a small flaw is exposed. Inexperienced programmers tend to write code without following the conventions defined to write syntaxes while using strings in the function code. [Bourque05].

Several different types of attacks based on strings have come into existence in the recent past. Strings have always been a fascinating field for vulnerable attacks. Strings comprise the main components in the program code. Environment variables, command line arguments in many of the C and C++ programs are strings by default and are converted or type casted in the program as per the functional requirement.

Buffer overflow, null termination errors usually don’t create much of a problem in the normal programming environment [E-Crime 05]. When deployed into client systems and used web based inputs, the scope of being attacked by unwanted intruders is high. Application security needs to be given high importance. Extra care should be taken while implementing strings because there is high probability of being attacked.

String based attacks involve truncation of the string messages or modifying incoming patterns of input. These attacks can pave way into the system or redirect the systems, in to the system with privileges, hence being fruitful to the efforts of intruders trying to gain malicious access to the systems resources.
Errors in strings are possible in wide range. This can be observed because of the representation of the strings, declaration of the strings, usage of string based functions. Errors like improper null termination can also cause huge problems when they result in errors like buffer overflows.

OPERATIONS IN STRINGS:

There are several string operations that can be performed on the strings. Direct concatenation of strings or copying the strings can be done directly using string based functions. Problems arise when the representation of the syntax of the string functions is erroneous. Function calls to strings at wrong points of the code can provide gateways to the intruders into the system with privileges [ISO/IEC 05].

BUFFER OVERFLOW:

Buffer overflow is one of the most prevalent attacks when strings are used in the code of a software or application. Buffer overflow is observed when usually data or strings are written beyond the boundaries than the memory allocated. The number of bytes allocated for a string need to be allocated properly to fit the length of the string completely.

1.2 Dynamic Memory Allocation

Dynamic memory management is nothing but the management of memory by a dynamic memory allocator. Normally in C programming language, allocation of memory is done using the functions calloc(), malloc() and realloc() functions and the de-allocation
of memory is done by the usage of free() function. In C++ programming language, allocation of memory is done with the help of the new operator and de-allocation of memory is done with the help of the delete operator [ISO/IEC 09].

**MEMORY MANAGEMENT FUNCTIONS:**

The memory management functions which would be described now are commonly used and will be present on various platforms. The functions of memory management are as follows.

`malloc(size_t size);`

This function would allocate `size` bytes and it would then return a pointer to the memory which is allocated. It should be noted that memory would not be cleared.

`free(void * p);`

This function is used to free the memory which was previously pointed. It should have already be returned by a call to malloc(), realloc() or calloc() previously. Or else, if free(p) was already called initially, undefined behavior would take place.

`realloc(void *p, size_t size);`

This function would change the block of memory which was pointed by `p`. The matter would not be changed to the minimum of the old and also the new sizes. The memory which is newly allocated would also be uninitialized.
Now if \( p \) would be null, the call would be the same as `malloc(size)`. If the size would be equal to `malloc(size)`, the call would be the equal to `free(p)`. Unless \( p \) is null, it will be returned previously with a call to `malloc()`, `calloc()` or `realloc()`.

calloc(size_t nmem, size_t size);

This function would assign memory for an array of elements of bytes equivalent to `size` and would then return a pointer to the memory which is already allocated. The memory here would be set to zero.

### 1.3 Formatted Output

A formatted string together with a multiple number of arguments is defined as a formatted output function. The formatted output function represents (interprets) a set of instructions provided by the format string specifier. The execution control of the formatted output function can be made possible by controlling the content of the format string specifier.

The result obtained by supplying the inputs to the system is called as an output. The outputs depend upon the inputs given to the application.

The inputs can be of the following types:

- Number
- String
- Alphanumeric
- Picture etc.,
The functions consisting of a variable number of arguments with a formatted string is called as a formatted output function. A programmer controls the execution of the formatted output function by controlling the content of the format string. The format string consists of a group of instructions. These instructions are represented by the formatted output functions. Vulnerabilities exist when working on formatted output functions which breach the security of the applications.

OUTPUT FUNCTIONS:

Every output function generally consists of a variable number of arguments and a format string specified. The output functions based on standards are built into two types namely: Variadic and General Functions. Both the functions are defined by a predefined C99 standard.

VARIADIC FUNCTIONS:

Variadic functions are implemented the ANSI C approach or the UNIX system. The variadic functions are:

- vprintf( )
- vsprintf( )
- vfprintf( )
- snprintf( )
GENERAL FUNCTIONS:

The functions implemented using any other approaches are the general functions. The general functions are:

fprintf( ): Based on the contents of the format string, this function writes the output to a stream. The format string, a variable list of arguments and the stream are provided as arguments.

printf( ): printf is equivalent to fprintf except that the output stream is to the console i.e., standard output(stdout).

sprintf( ): sprintf is equivalent to fprintf except that the result is stored in an array in-spite of a stream.

snprintf( ): this function is similar to sprintf except that the maximum number of characters that can be stored in the array is ‘n’ where n is a non-zero value. If the array size is ‘n’, then ‘n-1’ values are stored and the last position is filled with a null value.

BUFFER OVERFLOW:

Buffer overflow occurs when the output is stored in arrays by the formatted output functions. Output functions assume buffers with long size.

Example: char overflow [512];

printf(overflow,” array out of bounds: %s”, uname);

The string that exceeds 495 bytes from the above with 512 bytes, 16 character bytes and one null byte finally results in array out of bounds error.
OUTPUT STREAMS:

Vulnerabilities are also caused in formatted output functions as they write output to a stream instead of stdout functions [Seacord 10]. The vulnerabilities like viewing content in memory, stack or obtaining file permissions can occur which finally results in failure of the program.

1.4 Pointers

Variables in a program are used to store the values. Each variable has its own address space. By initializing a variable, an address space is allocated respective to its data type size. Pointers are used to indicate the address space of these variables. Pointers are used a registers for the variables.

Pointers not only store the address of the variables but also contain the addresses of the data structure, or an array or any function that are initialized. The basic program will be run by calling the function pointer (Main( ) function).

There will be two types of pointers. They are function pointers and data pointers. Function pointers are used to store the address space for functions which are initialized. Data pointers are used to store the address of arrays, variables and data structures. An attacker may have a chance to over write the function pointers by using supplied shell code. The attackers supplied shell code will be executed instead of original code when function pointer is executed [Smashing 05].
FUNCTION POINTERS:

Overwriting function pointers have equal importance in any memory segment when stack smashing is not possible in any data segment.

In the every program, it can be observed that both the characters buff and function pointer funcptr are not initialized. They are stored in BSS segment. Strncpy( ) is used as bounded string copy function. If the length of argv exceeds the buffsize, then there will be a buffer flow.

DATA POINTERS:

Data pointers play an important role in C/C++. They are generally dynamically allocated structures. Data pointers are used as different data structures like call-by reference, function arguments, arrays etc., and an attacker can easily modify the data pointer.

For arbitrary memory write, there will be unbounded memory copy in the program. An attacker would try to over write ptr and val after the buffer is over flowed. An arbitrary write will be executed when the condition *ptr=val is satisfied. The presence of common errors will help the attacker in modifying the data pointers.

1.5 Integers

There are different arithmetic operation used with the integers such as addition, subtraction, multiplication, division and modules. These operations are useful for the programmer to perform various mathematical functions for computing. Many students or
developers have very limited knowledge on integer vulnerabilities. Some of the vulnerabilities associated with integers are described below.

**OVERFLOW BITS:**

Overflow bits represent error condition without understanding the context of the integers. Integer overflow only occurs when the data is more in the buffer. It is highly impossible to know whether it is an error condition or not. It can be either an error or may not be an error. In the usage of integers, the overflow occurrence is a very common problem.
2. NARRATIVE

2.1 Problem Statement

As it was discussed, there has been no method developed previously which detects the errors and vulnerable issues in C and C++. There has been no software or tool which can find the vulnerable issues in C and C++ programs.

2.2 Motivation

In today’s scenario, coding is very crucial for developers. Writing the code in a secure way has become a necessity. There was no application developed previously which would help the programmer to find out the vulnerabilities in a given code. This very reason was the main motivation to come up with the proposed project.

2.3 Functionality of the project

This proposed project will have various functionalities. The project can be used in any operating system. This application will help the developers to build a secure code. Also it will help the testers and the end users to find out whether the given code is secure or not. Even if an attack takes place, this application would point out the areas in which the code is susceptible to an attack. Thus, care can be taken to avoid attacks in future. The proposed application will save a lot of time for the programmers or the developers to find out the errors instead of manually checking each line of the code.
3. PROPOSED SYSTEM DESIGN

3.1 Secure coding mechanism in C and C++

The primary aim of the current project is to pick out the vulnerable issues which are probable to happen when coding a program. Program can be checked on secure coding software tool which points out the so-called vulnerabilities in C or C++. Proposed tool also suggests the user, the manner in which the vulnerability can be rectified. In C and C++, vulnerabilities mainly include buffer overflow and stack smashing which will be handled by the proposed software tool.

- **User friendliness**: By noticing the earlier methods, enough care has been taken to see that the secure software coding method proposed in this paper is easy to understand and implement.

- **Vulnerable detection**: The proposed application will have the capability to detect all vulnerable issues and flag the lines of code which have vulnerable issues for reporting.

- **Reporting**: In this proposed project, the application gives out the report of all the vulnerable issues with the best replacing code. By which the vulnerable issues can be fixed manually.
3.2 Framework

Normally, developer undergoes into vulnerable issues when coding in C language. When the developer has a requirement to develop code, it leads to many vulnerable issues which were unknown to the developer. The usage of our proposed method to check or verify the code using secure coding application tool minimizes the vulnerable issues. The security issues were the main reason to implement such a method. Thus taken this proposal as a challenge to make one desktop application, which can catch all the issues at one click and to present by what lines of code it has to be replaced to make coding secure.

3.2.1 Steps in the project development

The steps involved in building this project from the initial point are given below in brief:

1. Build or get a C or C++ code
2. Inject that code in to the secure coding java application
3. Code analyzer undergoes several steps
   a. Identify all the tokens
   b. Write tokens in to regular expressions
   c. Divide conflicting tokens into multiple tokens to avoid errors
   d. Give out the vulnerable lines of code
4. Java application gives out the vulnerable lines of code
5. The best invulnerable code is suggested by application
3.2.2 System requirements

Normally any computer installed with windows operating system and java software will be able to run this project. This application runs best on windows 7 operating system. The basic system requirements that are required in order to run this project’s application will be given below:

**Software requirements: (Minimal requirements)**

- Front End/GUI Tool : Java eclipse IDE
- Operating System : Windows Family
- Language : Java
- Input : C or C++ file

**Hardware requirements: (Minimal requirements)**

- Processor : Pentium dual core
- RAM : 1 GB
- Hard Disk Drive : 80 GB

3.3 Proposed Mechanism

In the proposed approach, secure coding application or software tool is different from all the others since after flagging the errors or vulnerable issues, it also gives out the respective possible right solution to the user.
The entire process will take place in following steps:

1. Build or get a C or C++ code
2. Inject that code in to the secure coding java application
3. Code analyzer undergoes several steps
   a. Identify all the tokens
   b. Write tokens in to regular expressions
   c. Divide conflicting tokens into multiple tokens to avoid errors
   d. Give out the vulnerable lines of code
4. Java application gives out the vulnerable lines of code
5. The best invulnerable code is suggested by application

3.3.1 Working

There will be two things working mainly in this application. One is the user who checks the vulnerable issues in the C or C++ code and another is the secure coding application or analyzer. The main role of the secure coding application is to check the vulnerable issues in the given code. And also gives out the best replaceable invulnerable coding as the output.

3.4 System design

The proposed application design process will be discussed in this section. Various UML diagrams explaining the design of the proposed application will be presented and explained.
3.4.1 Class diagram

The class diagram shows the structure of system by exploring through the system classes, attributes, methods (operations) and their relationships among all the classes. The class diagram concerning the proposed project can be seen in figure 1.

![Class Diagram](image)

**Figure 1: Proposed system’s class diagram**

As seen in figure 1, the details in this proposed application is shown graphically. The user imports C code in to the secure coding application then the application identifies the vulnerabilities. The identified vulnerabilities are then sent to the token identifier to know which line of the code has vulnerability and that is reported to user.
3.4.2 Use case diagram

The purpose of the use case diagram is to graphically represent the functionality of the system with regard to actors, their aims (shown as use cases) and any kind of dependencies arising between the particular use cases. The proposed system’s use case diagram can be seen in figure 2.

![Use case diagram](image)

**Figure 2: Proposed system’s use-case diagram**

As seen in figure 2, the functionality of the proposed system with respect to the actors, their aims and dependencies are represented graphically. The user is actor here and he performs the actions of importing code and running the application.
3.4.3 Sequence diagram

The sequence diagram is a type of interaction figure which shows how various processes operate with each other and in what order they operate. The proposed system’s sequence diagram can be seen in figure 3.

![Sequence diagram](image)

**Figure 3: Proposed system’s sequence diagram**

As seen in figure 3, the proposed system’s interactions are seen in an order with the help of the sequence diagram. The user imports code on to the secure coding application. The application then checks with all the vulnerability issues and they were displayed on the user screen.
3.4.4 State chart diagram

State chart diagram is composed of the finite number of states by which it describes the system. There are many forms of state chart diagrams, which differ by semantics slightly.

![State chart diagram](image)

*Figure 4: Proposed system’s state chart diagram*
As seen in figure 4, the proposed system is represented in the form of the state chart diagram which makes it easier to understand what are grouped together. Firstly user sends code snippet on to the application. The application collects all the tokens which have issues. Then on the user screen the vulnerable issues were replaced with the possible replaceable lines of code.
4. FUNCTIONALITIES OF THE APPLICATION

The proposed application’s steps would be discussed in this section. The steps would be explained with the respective screenshots.

4.1 Step wise description of functionality

The proposed project consists of user and application involving in it. Both of these work simultaneously.

- Initializing the code analyzer or secure code application
- Importing the C or C++ code
- Running the application
- Vulnerabilities detection
- Suggesting with Invulnerable code

The mentioned functionalities would be provided with the screenshots and would be explained in detail in accordance with the respective screenshot.
4.1.1 Initializing Code analyzer

As seen in figure 5, the user started using the application to get the vulnerabilities in the C or C++ code.

Figure 5: Initialization of code analyzer
4.1.2 Importing C code in to application

Figure 6: Importing Code in to the application

As seen in figure 6, the User imports the C or C++ code into the application to find out the issues in the code.
As seen in figure 7, the user selects the C code from the specified folder in his desktop or machine. And the application only accepts the .C or .CPP files.
4.1.3 Running the application

After successful importing of C and C++ code, run the secure coding application by clicking a green run button on the top of the application. By which application starts checking vulnerable issues.

Figure 8: Running the application to check vulnerable issues

As seen in figure 8, the user selected or imported code and started the code analyzer to check the vulnerable issues within it.
As seen in figure 9, the command prompts running backside of the application, breaks the code into tokens and each token of code separately checked to see if there are any conflicting errors. And after completing all the operations, vulnerable issues are traced.
4.1.4 Vulnerabilities detection

The application traces out all the issues within the C code which is given as input. The screenshot would be given in figure 10.

![Image of C code and vulnerabilities]

**Figure 10: Vulnerabilities detected in C code**

As seen in figure 10, the application traced out all the vulnerable issues and those issues are embedded into one place and shown on the command prompt. It clearly tells which line of code has what sort of issue.
4.1.5 Suggesting Invulnerable code

The command prompt issues were shown on the application interface. The application also suggests the invulnerable code and also tells what the problem with the previous lines. Also specifies which lines have those issues with the clear information.

Figure 11: Suggestive and replaceable code

As seen in figure 11, the application detects the issues and shows the possible changes to be done so that the code will be so secure.
5. TESTING AND IMPLEMENTATION

5.1 Testing

- Testing is the procedure to find out whether any errors arise while testing the application.
- Testing is really important in terms of software engineering.
- Testing is done in order to build confidence of the developer.
- Testing consists of a number of activities which can be planned initially and conducted in a systematic manner.
- Software testing can also be synonymous with validation and verification.

5.2 Types of Testing

The following are the types of the testing:

- White box testing
- Black box testing
- Unit testing
- Integration testing
- Validation testing
- Output testing
- User acceptance testing
5.2.1 White box testing

- The white box testing can also be referred to as glass box testing. The white box testing makes use of control structure in the procedural design to get the test cases.
- With the help of white box testing, a tester can make test cases that does the following things:
  - It makes sure that all the parts present in the module have been used at least once.
  - The logical decisions will be analyzed with the true criteria and false criteria.

5.2.2 Black box testing

- The black box testing can also be referred to as the behavioral testing. The black box testing stresses mostly on the software’s functional requirements.
- The black box testing is an approach of complementary type to present different type of errors from the white box testing errors.
- Black box testing makes use of some input criteria to fully exercise the program’s functional requirements.

5.2.3 Unit testing

In unit testing, every module would be tested and then would be grouped with the system. Unit testing stresses to test on the tiniest unit of the software to make sure it is working. It can also be referred to as the module testing.
Each module presented in the system would be tested separately. This testing would be done at the programming time itself. In this testing process, it is made sure that every module would work as per the requirements. The errors are easy to find in the initial stage itself rather than the final stage.

### 5.2.4 Integration testing

Integration testing stresses on the fact that data may be lost over an interface. A particular module can have a drastic effect on remaining sub functions and when combined will not give the required functionality. The integration type is like the systematic testing to find out the uncovered errors inside the interface. The testing will be done with a small sample of data. The system which is developed will be tested with this sample data to check whether it will run successfully or not. With the help of integration testing, the system’s overall performance can be evaluated.

### 5.2.5 Validation testing

When the black box testing is at its peak, software is grouped as one package, interface related errors are corrected, the validation testing begins. There can be many definitions for validation testing. In simple words, validation will be successful only when the software produced will be as per the satisfaction of the customer. So the validation tests will be confirming whether the final user will be satisfied with the product or not.
5.2.6 Output testing

After the completion of the validation testing, output testing would be performed. The output testing would be performed because every system would need the output to be as per the specified requirements. The output testing will prove whether the output is satisfactory or not.

5.2.7 User acceptance testing

This testing is used to find out whether the application developed is user friendly or not. The user acceptance testing will be successful if the users can access the application easily.

And all the tests which were discussed above are done successfully when developing the code. All the seven tests were very useful for developer when implementing or proposing the new application or software.

5.3 Test cases

The proposed project will now be tested for all the following test cases sequentially:

- Testing with strings in C and C++ code
- Testing with integers in C and C++ code
- Testing with dynamic memory functions in C and C++ code
- Testing with formatted output in C and C++ code
- Testing with pointers in C and C++ code
5.3.1 Testing with strings in C and C++ code

Now the C program with string functions would be tested. The user imports C code on the application. Then java application will result in all the vulnerable issues in provided C code.

![String Vulnerability Analyzer](image)

**Figure 12: Interface for testing string functions**

As seen in figure 12, the user imported the C code with “char strings”, like “strcmp” and “strcat”. The imported code is checked for the vulnerabilities associated with strings in C and C++ code.
As seen in figure 13, the application detects the vulnerable issues related with string functions and gives out all the issues, for C or C++ code given by the user as input. And the application gave out the vulnerable issues in the line 6 and 7 of code with replaceable lines of code.

5.3.2 Testing with integers in C and C++ code

Now the C program with integer operations would be tested. The user provides C code on the application. Then java application will result in all the vulnerable issues in provided C code.
As seen in figure 14, the user login in to the Integer analyzer interface. The user enters the user name and secret code in order to enter in to the application.
Figure 15: Interface for testing integer operations

As seen in figure 15, the user imported the C code with integer functions like “unsigned int” and “int”. The imported code is checked for the vulnerabilities associated with integers in C and C++ code.
As seen in figure 16, the application detects the vulnerable issues related with integer operations and gives out all the issues, for C or C++ code given by the user as input. And the application gave out the vulnerable issues in the line 4, 5, 6 and 7 of code with solution to overcome the issues.
5.3.3 Testing with Dynamic Memory functions in C and C++ code

Now the C program with Dynamic memory functions would be tested. The user provides C code on the application. Then java application will result in all the vulnerable issues in provided C code.

![Dynamic Allocation Interface]

**Figure 17: Interface for testing Dynamic memory allocation**

As seen in figure 17, the user imported the C code with Dynamic memory functions like “malloc” and “alloc”. The imported code is checked for the vulnerabilities associated with dynamic memory functions in C and C++ code.
Figure 18: Test result with vulnerable issues in dynamic memory allocation

As seen in figure 18, the application detects the vulnerable issues related with Dynamic memory functions and gives out all the issues, for C or C++ code given by the user as input. And the application gave out the vulnerable issues in the line 9, 10 and 11 due to the free memory, in the line 12 due to the heap allocation and in the line 6 due to malloc operations in the code with solution to overcome the issues.
5.3.4 Testing with Formatted output in C and C++ code

Now the C program with formatted output would be tested. The user provides C code on the application. Then java application will result in all the vulnerable issues in provided C code.

![Figure 19: Interface for testing formatted output](image)

As seen in figure 19, the user imported the C code with formatted output functions like “printf” and standard library function “stdio.h”. The imported code is checked for the vulnerabilities associated with dynamic memory functions in C and C++ code.
Figure 20: Test result with vulnerable issues in formatted output

As seen in figure 20, the application detects the vulnerable issues related with formatted output and gives out all the issues, for C or C++ code given by the user as input. And the application gave out the vulnerable issues in the line 7 and 1 due to the susceptible to buffer overflow and security issue with library function respectively in the code with solution to overcome the issues.
5.3.5 Testing with pointers in C and C++ code

Now the C program with pointers would be tested. The user provides C code on the application. Then java application will result in all the vulnerable issues in provided C code.

![Image: Pointer Vulnerability Analyzer Interface](image)

**Figure 21: Interface for testing pointers**

As seen in figure 21, the user imported the C code with pointers. The imported code is checked for the vulnerabilities associated with pointers in C and C++ code.
As seen in figure 22, the application detects the vulnerable issues related with pointers and gives out all the issues for C or C++ code given by the user as input. And the application gave out the vulnerable issues in the line 9, 10 and 11, due to the vulnerability in function pointer, vulnerability in long jump and vulnerability in exit functions respectively in the code with solution to overcome the issues.
6. CONCLUSION AND FUTURE WORK

The application implemented would work as a powerful tool to make sure that the code is not vulnerable to attacks. For a wider range of code inputs, the tool helps in finding the reasons for vulnerabilities found during the execution. Coding practices can be improved to a greater extent with the help of this application.

Any C or C++ functions, when not used properly with precise memory allocation, the results obtained would be disastrous. Knowledge and training about functions and their usability is very much required to ensure that while coding, developers do not leave path ways for intruders to step into. The developers can use this tool to check the vulnerable issues in their developed code.

The developed application is useful for searching and reporting the vulnerable issues but those issues would have to be fixed by the human manually. As part of the future work, a compiler can be developed which fixes the vulnerable issues within the code automatically.
BIBLIOGRAPHY AND REFERENCES


http://www.cuj.com/documents/s=8188/cuj0510seacord/


