**Secure Code Analyser**

***A***

***Project Report***

*Submitted in partial fulfillment of the*

*requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE & ENGINEERING**

**With specialization**

**Banking, Finance, Security and Insurance**

**by**

|  |  |
| --- | --- |
| **Name** | **Roll No.** |
| **BINEEK RAJA** | **R133201410** |
| **VIKASH ANAND** | **R133214052** |
| **VANDANA SHARMA** | **R133214051** |
| **MOHIT PANDEY** | **R133214026** |

***Under the guidance of***

**Mr. Jatin Sethi**

**Assistant Professor**

**Computer Science Department, CIT**



**Department of Computer Science & Engineering**

**Centre for Information Technology**

**University of Petroleum & Energy Studies**

**Bidholi, Via Prem Nagar, Dehradun, UK**

**May – 2017**



**CANDIDATE’S DECLARATION**

I/We hereby certify that the project work entitled **“ Secure Code Analyser”** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Banking, Finance, Security and Insurance and submitted to the Department of Computer Science & Engineering at Center for Information Technology, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **January**, **2017** to **May**, **2017** under the supervision of **Mr. Jatin Sethi, Assistant Professor, Computer Science Department.**

The matter presented in this project has not been submitted by me/ us for the award of any other degree of this or any other University.

**(BINEEK RAJA, VIKASH ANAND, VANDANA SHARMA, MOHIT PANDEY)**

**Roll No. R1332104010, R133214052, R133214051, R133214026**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 2nd May 2017 **Jatin Sethi**

Assistant Professor

CIT-COES, UPES

Mr. Christalin Nelson

Program Head – B.Tech CSE-BFSI

Center for Information Technology

University of Petroleum & Energy Studies

Dehradun – 248 001 (Uttarakhand)

**ACKNOWLEDGEMENT**

We wish to express our deep gratitude to our guide **Jatin Sethi**, **Assistant Professor,** for all advice, encouragement and constant support that he has given us through out our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank to our respected **Christalin Nelson**, **Program Head of the Department** for his great support in doing our project.

We are also grateful to Dr. **Manish Prateek, Associate Dean-CIT** and Dr. **Kamal Bansal, Dean CoES**, UPES for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our **friends** for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our **parents** who have shown us this world and for every support they have given us.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **BINEEK RAJA** | **VIKASH ANAND** | **VANDANA SHARMA** | **MOHIT PANDEY** |
| **Roll No.** | **R133214010** | **R133214052** | **R133214051** | **R133214026** |

**ABSTRACT**

Secure code analyser, also known as Static Application security testing tool. We use it to find out security flaws. There are many guidelines that a programmer can implement to aid in the prevention of common security bugs in applications which we will follow. There are many guidelines which we can use in all languages but there are also some guidelines that can only work with a specific language. In this secure code Analyser we only work with C, C++ and Java languages. Analysers allow a programmer to remove much of the common vulnerabilities found in code.

**TABLE OF CONTENTS**

**S.No. Contents Page No**

1. **Introduction 6**
2. **Problem Statement 6**

1. **Objective 6**

1. **Methodology 7**
2. **Pert Chart 8**
3. **Data Flow Diagram 9**
4. **Use Case Diagram 10**
5. **Pseudo Code 11**
6. **Output 13**

**References 15**

**Introduction:**

Secure code analyzer, also referred as Static Application Security Testing (SAST) Tools. It is designed to analyze source code and compiled versions of code to help find security flaws. designed to analyze source code or compiled versions of code. There are many guidelines that a programmer can implement to aid in the prevention of common security bugs in applications. Many of these can be applied to any programming language, but some are specific to only one language. In this project we are mainly working on programming languages like C, C++ and Java. There are numerous guidelines and tips that a programmer can implement to aid in the prevention of common security bugs in applications. Analyzers allow a programmer to remove much of the common vulnerabilities found in code. So with the help of Secure code Analyzer we can find out all the flaws in our programs and remove them.

**Problem Statement:**

* To analyse and find vulnerability of our code.
* Provide solutions for the vulnerability of the code that is being analysed.

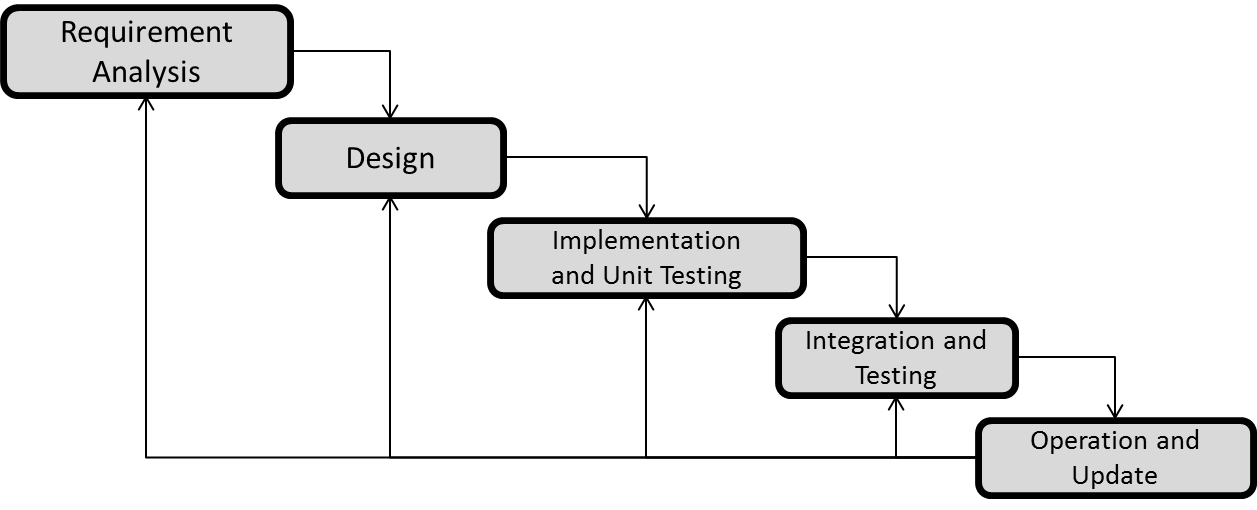
**Objectives:**

* To implement a static application that will Analyse the vulnerability of the code in C, C++ and Java.

**Methodology:**

We are using ITRERATIVE WATERFALL MODEL due to large data size and complexity of our project.

**ITERATIVE WATERFALL MODEL**



1. **Requirement Analysis:**

Data modelling i.e. collection of data which will be used as inputs and outputs.

1. **System Design:**An algorithm will be developed on the basis of formulas and data that are collected and will these data will be sampled as input in further phase. The algorithm will decrease the time complexity and will finally give an automated system for all the calculations under a single dashboard.
2. **Implementation and Unit Testing:**

We will develop the application using the algorithm developed during design phase in C language. The application will be developed in two modules and will be tested separately.

1. **Integration and Testing:**

The modules developed will be integrated to develop an application and tested by using the sample data collected during the analysis phase.

1. **Operation and Update:**

The final application developed will be deployed and updated according to the bugs which we will encounter.

**Project Progress:**



* Study of Secure Code Analyser.
* Database Creation.
* Creation of User Interface.
* Implementation of Rabin Karp Algorithm.

**Data Flow Diagram:**

Level - 0 DFD

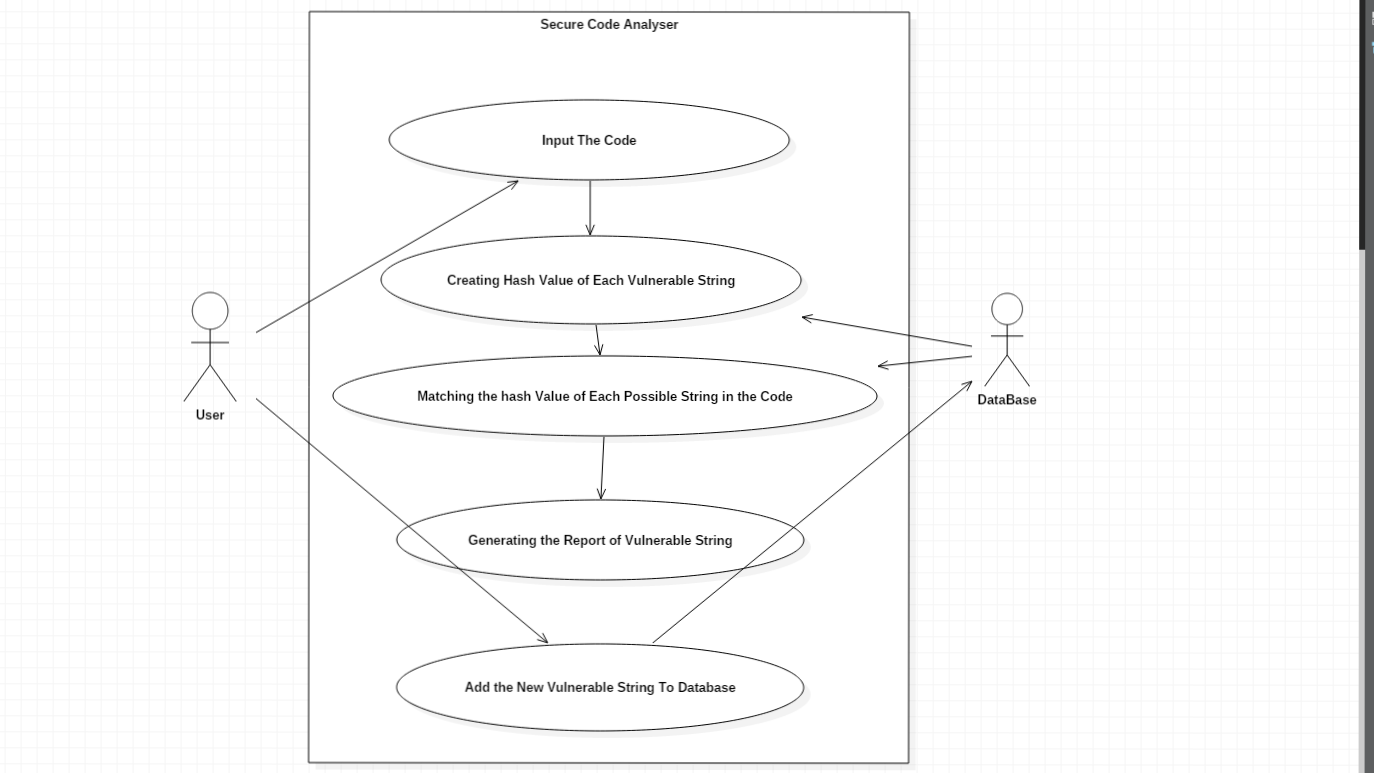
Searching the Vulnerabilities Using Rabin Karp Algorithm and Hashing Techniques

User Interface

Data Set

Generation of Report

**Use Case Diagram:**



**Pseudo Code:**

package securecode;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.\*;

import static javafx.beans.binding.Bindings.select;

import javax.swing.JOptionPane;

/\*\*

\*

\* @author BINEEK RAJA

\*/

public class Reading {

public Reading() throws FileNotFoundException, ClassNotFoundException, SQLException, IOException{

Class.forName("com.mysql.jdbc.Driver");

Connection c=DriverManager.getConnection("jdbc:mysql://localhost:3306/securecode","root","vikashanand");

String s1=("select \* from javadata");

Statement s=c.createStatement();

ResultSet rs=s.executeQuery(s1);

File f=new File("Code.txt");

ArrayList<Long> alindex = new ArrayList<Long>();

ArrayList<String> alstring = new ArrayList<String>();

Scanner scan=new Scanner(f);

while(scan.hasNextLine()){

String a=scan.nextLine();

a=a.replaceAll("[^a-zA-Z0-9\_-]", " ");

//System.out.print(a);

String word[]= a.split(" ");

int wl= word.length;

for(int l=0;l<wl;l++)

{

int prime=101;

long hash=0;

char[] str=word[l].toCharArray();

int end=str.length;

//System.out.println(end);

for(int i=0;i<end;i++)

{

hash+=str[i]\*Math.pow(prime,i);

}

//System.out.println(hash;

if(hash != 0){

alindex.add(hash);

alstring.add(word[l]);

}

}

}

ArrayList<Long> dlindex = new ArrayList<Long>();

ArrayList<String> dlstring = new ArrayList<String>();

while(rs.next())

{

Long p=rs.getLong(1);

String q=rs.getString(3);

dlindex.add(p);

dlstring.add(q);

}

/\*while(itrIndex.hasNext() && itrString.hasNext()){

Long var1=(Long) itrIndex.next();

String var2=(String) itrString.next();

System.out.println(var1+" "+var2);

} \*/

dlindex.retainAll(alindex);

dlstring.retainAll(alstring);

//Iterator itrIndex = dlindex.iterator();

Iterator itrString = dlstring.iterator();

int length= dlstring.size();

String str[]=new String[length];

int i=0;

while(itrString.hasNext()){

str[i]=(String) itrString.next();

//System.out.println(str[i]);

i++;

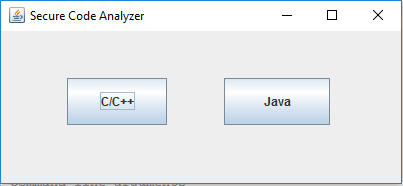
}

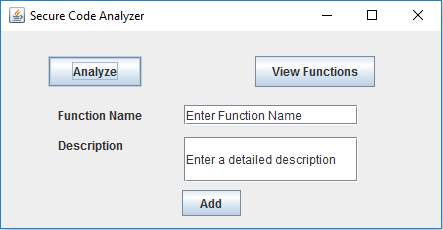
JOptionPane.showMessageDialog(null, str,"The following functions are vulnerable",0);

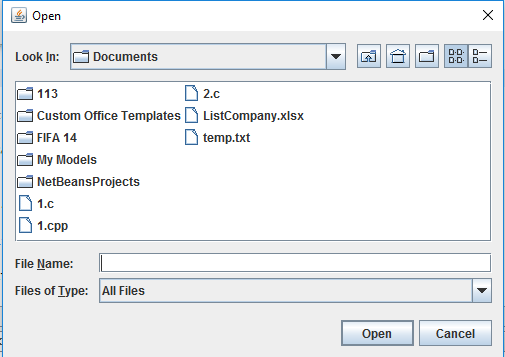
}

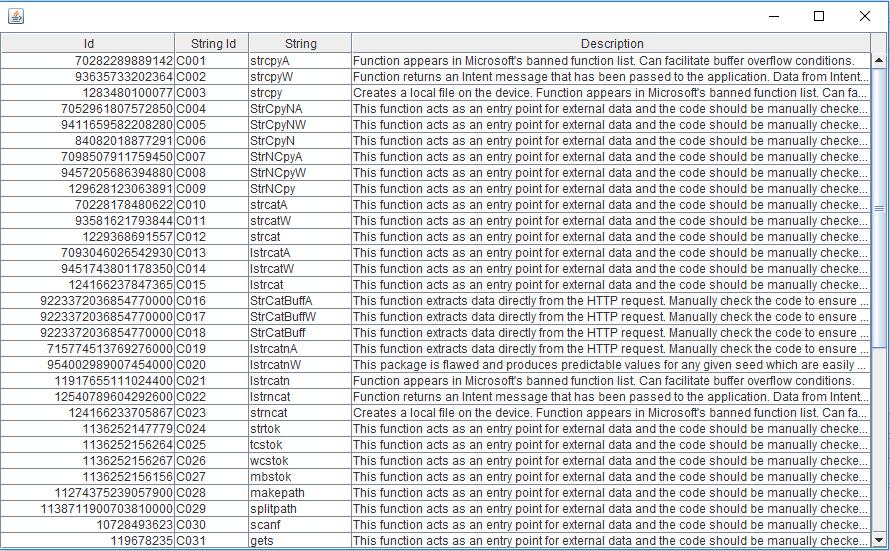
}

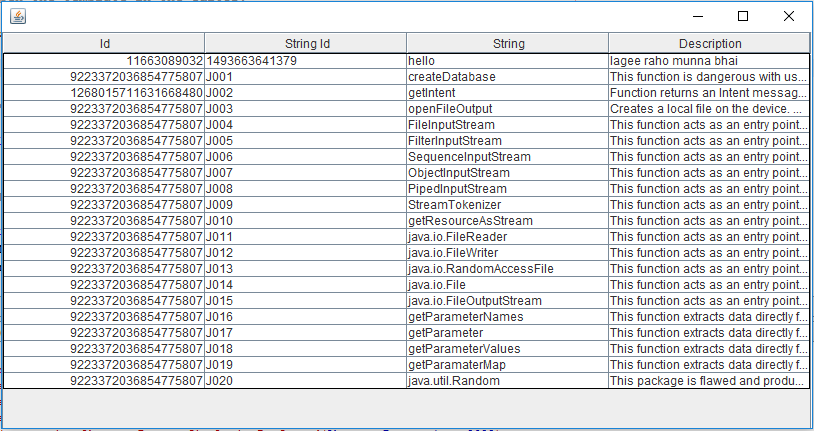
**Output:**











**References:**

1. The Use and Limitations of Static-Analysis Tools to Improve Software Quality." Anderson, Paul, CrossTalk, Vol. 21, No. 6, (June 2008), pp. 18-21
2. Kratkiewicz Kendra, and Lippmann, workshop on the evaluation on software defect detection tools