A

Minor Project Report

on

UN-PRIVILEGE BLACK BOX KEYLOGGER DETECTION

Submitted in Partial Fulfillment of
the Requirements for the Third Year
of

Bachelor of Engineering

in

Computer Engineering

to

North Maharashtra University, Jalgaon

Submitted by

Mayuri D. Patil
Rupali S. Baviskar
Vrushali S. Patil
Minakshi F. More

Under the Guidance of

Mr.Jitendra R. Patil

DEPARTMENT OF COMPUTER ENGINEERING
SSBT's COLLEGE OF ENGINEERING AND TECHNOLOGY,
BAMBHORI, JALGAON - 425 001 (MS)

2015 - 2016

SSBT's COLLEGE OF ENGINEERING AND TECHNOLOGY, BAMBHORI, JALGAON - 425 001 (MS)

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the minor project entitled *Un-Privilege Black Box Keylogger detection*, submitted by

Mayuri D. Patil
Rupali S. Baviskar
Vrushali S. Patil
Minakshi F. More

in partial fulfillment of the Third Year of *Bachelor of Engineering* in *Computer Engineering* has been satisfactorily carried out under my guidance as per the requirement of North Maharashtra University, Jalgaon.

Date: April 12, 2016
Place: Jalgaon

Mr.Jitendra R. Patil
Guide

Prof. Dr. Girish K. Patnaik
Head

Prof. Dr. K. S. Wani
Principal

Acknowledgements

We take this opportunity to express our gratitude to all those who have rendered co- operation and guidance throughout our project Un-Privilege Black Box Keylogger Detection. We express our sincere thanks to Dr. K. S. Wani. Principal of SSBT COET , Bamb- hori and Dr.Girish Kumar Patnaik (Head of the Department,Computer Engineering) SSBT COET, Bambhori. We also extend our sincere thanks to my Project guide Mr.Jitendra R. Patil. guide us and provide us the opportunity to undertake this seminar and their valuable advices and all other members of the faculty of Computer Engineering Department and our parents,friends for their cooperation and encouragement. Last but not the least, We extremely indebted our parents and friends without whose support this effort not reach its successful completion. Thank You...!!!

Mayuri D. Patil
Rupali S. Baviskar
Vrushali S. Patil
Minakshi F. More

Contents

List of Tables

List of Figures

Abstract

To evaluate the ability to detect real-world keyloggers, we experimented with all the keyloggers from the top monitoring free software list, an online repository continuously updated with reviews and latest developments in the area. During the era of 21st century most of the people uses internet to complete their work so to prevent from hacking our accounts we are creating such a software which detects the keylogger present in the system and kill them manually. Software keyloggers are a fast growing class of invasive software often used to harvest confidential information. One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a system. The ability to run in unprivileged mode facilitates their implementation and distribution, but, at the same time, allows one to understand and model their behavior in detail. Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes.Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes.

Chapter 1

Introduction

The keyboard is the primary aim for key loggers to retrieve user input from because it is the most common user interface with a computer. Although both hardware and software key loggers exist, software key loggers are the dominant form and thus are main point in this paper. Software keylogger are most inexpensive easily used program. This keyloggers need to be adapted to each target operating system to ensure I/O is handled appropriately.
In section 1.1 background is describe. In section 1.2 motivation towards the implement system is describe. In section 1.3 the problem definition of the keylogger is describe and the scope and objectives are describe in section 1.4 and 1.5 respectively.

1.1 Background

Software keyloggers are a fast growing class of invasive software often used to harvest confidential information. One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a system. The ability to run in unprivileged mode facilitates their implementation and distribution, but, at the same time, allows one to understand and model their behavior in detail. Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes.We have prototyped our technique as an unprivileged application, hence matching the same ease of deployment of a keylogger executing in unprivileged mode.
We have successfully evaluated the underlying technique against the most common free keyloggers. This confirms the viability of our approach in practical scenarios. We have also devised potential evasion techniques that may be adopted to circumvent our approach and proposed a heuristic to strengthen the effectiveness of our solution against more elaborated attacks. Extensive experimental results confirm that our technique is robust to both false positives and false negatives in realistic settings.Softwarekeyloggers are a fast growing class

of invasive software often used to harvest confidential information.
One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a system. The ability to run in unprivileged mode facilitates their implementation and distribution, but, at the same time, allows one to understand and model their behavior in detail. Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes.

1.2 Motivation

To evaluate the ability to detect real-world keyloggers, we experimented with all the keyloggers from the top monitoring free software list, an online repository continuously updated with reviews and latest developments in the area. During the era of 21st century most of the people uses internet to complete their work so to prevent from hacking our accounts we are creating such a software which detects the keylogger present in the system and kill them manually.

Detailed Problem Definition:

Keyloggers are implanted on a machine to intentionally monitor theuser activity by logging keystrokesand eventually delivering them to a third party. We propose a new approach to detect keyloggers running as unprivileged user-space processes. To match the same deployment model, our technique is entirely implemented in an unprivileged process. As a result, our solution is portable, easy to install, and yet very effective. In addition, the proposed detection technique is completely black-box, i.e., based on behavioral characteristics common to all keyloggers. In other words, our technique does not rely on the internal structure of the keylogger or the particular set of APIs used for this reason, our solution is of general applicability. We have prototyped our approach and evaluated it against the most common free keyloggers. Our approach has proven effective in all the cases. Keylogger detection software can detect keyloggers while anti-virus software does not detect keyloggers. User can see previous status of scanning keylogger processes in our software while it is not possible in other anti-virus software. Malware, viruses, spyware does not detected by our software while other anti-virus software can detect this malware and viruses.

1.3 Problem Definition

One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a

system. The ability to run in unprivileged mode facilitates their implementation and distribution, but, at the same time, allows one to understand and model their behavior in detail. Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes. We have prototyped our technique as an unprivileged application, hence matching the same ease of deployment of a keylogger executing in unprivileged mode. We have successfully evaluated the underlying technique against the most common free keyloggers. This confirms the viability of our approach in practical scenarios. We have also devised potential evasion techniques that may be adopted to circumvent our approach and proposed a heuristic to strengthen the effectiveness of our solution against more elaborated attacks.
The propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes. We have prototyped our technique as an unprivileged application, hence matching the same ease of deployment of a keylogger executing in unprivileged mode. We have successfully evaluated the underlying technique against the most common free keyloggers. This confirms the viability of our approach in practical scenarios. We have also devised potential evasion techniques that may be adopted to circumvent our approach and proposed a heuristic to strengthen the effectiveness of our solution against more elaborated attacks. Extensive experimental results confirm that our technique is robust to both false positives and false negatives in realistic settings.

1.4 Scope

Software that can not only monitor every keystroke and action performed at a PC but also be used as legally binding evidence of wrong-doing has been unveiled. Worries about cyber-crime and sabotage have prompted many employers to consider monitoring employees. They have joined forces to create a system which can monitor computer activity, store it and retrieve disputed files within minutes People need to recognize that you are using a PC as a representative of a company and that employers have a legal requirement to store data. Website monitoring service can check HTTP pages, HTTPS, SNMP, FTP, SMTP, POP3, IMAP, DNS, SSH, TELNET, SSL, TCP, ping and a range of other ports with great variety of check intervals from every 4 hours to every one minute. Typically, most network monitoring services test your server anywhere between once-per hour to once-perminute. Features: Protect intellectual property and business secrets Prevent and stop sabotage and data theft Prevent Internet/email abuse Reduce workplace slackers Improve efficiency and productivity.

1.5 Objectives

The modules which are present in this system is that we detect keyloggers running as unprivileged user-space processes. To match the same deployment model, our technique is entirely implemented in an unprivileged process. As a result, our solution is portable, easy to install, and yet very effective. Following are the components which will be used in the system. Keyloggers are implanted on a machine to intentionally monitor theuser activity by logging keystrokes and eventually delivering them to a third party. We propose a new approach to detect keyloggers running as unprivileged user-space processes. To match the same deployment model, our technique is entirely implemented in an unprivileged process. As a result, our solution is portable, easy to install, and yet very effective. In addition, the proposed detection technique is completely black-box, i.e., based on behavioral characteristics common to all keyloggers. In other words, our technique does not rely on the internal structure of the keylogger or the particular set of APIs used for this reason, our solution is of general applicability. We have prototyped our approach and evaluated it against the most common free keyloggers. Our approach has proven effective in all the cases.

1.6 Summary

In this chapter discussed about the background, motivation, scope, objectives of Keylogger Dectection System. In next chapter discuss about system analysis.

Chapter 2

System Analysis

Keyloggers are implanted on a machine to intentionally monitor the user activity by logging keystrokes and eventually delivering them to a third party. We propose a new approach to detect keyloggers running as unprivileged user-space processes. To match the same deployment model, our technique is entirely implemented in an unprivileged process.
In this chapter discuss about analysis of project.In section 2.1 literature survey,In section 2.2 the proposed system is decribe. In section 2.3 the feasibility study of system describe.Also in Section 2.4, 2.5 and 2.6 the risk analysis, project scheduling, effort allocation are describe respectively.

2.1 Literature Survey

Software keyloggers are a fast growing class of invasive software often used to harvest confidential information. One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a system. The ability to run in unprivileged mode facilitates their implementation and distribution, but, at the same time, allows one to understand and model their behavior in detail. Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes. We have prototyped our technique as an unprivileged application, hence matching the same ease of deployment of a keylogger executing in unprivileged mode. We have successfully evaluated the underlying technique against the most common free keyloggers. This confirms the viability of our approach in practical scenarios.
We have also devised potential evasion techniques that may be adopted to circumvent our approach and proposed a heuristic to strengthen the effectiveness of our solution against more elaborated attacks. Extensive experimental results confirm that our technique is robust to both false positives and false negatives in realistic settings.Softwarekeyloggers are a

fast growing class of invasive software often used to harvest confidential information. One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a system. The ability to run in unprivileged mode facilitates their implementation and distribution, but, at the same time, allows one to understand and model their behavior in detail.
Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes. We have prototyped our technique as an unprivileged application, hence matching the same ease of deployment of a keylogger executing in unprivileged mode. We have successfully evaluated the underlying technique against the most common free keyloggers. This confirms the viability of our approach in practical scenarios. We have also devised potential evasion techniques that may be adopted to circumvent our approach and proposed a heuristic to strengthen the effectiveness of our solution against more elaborated attacks. Extensive experimental results confirm that our technique is robust to both false positives and false negatives in realistic settings.

2.2 Proposed System

Software keyloggers are a fast growing class of invasive software often used to harvest confidential information. One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a system. The ability to run in unprivileged mode facilitates their implementation and distribution, but, at the same time, allows one to understand and model their behavior in detail. Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes.We have prototyped our technique as an unprivileged application, hence matching the same ease of deployment of a keylogger executing in unprivileged mode. We have successfully evaluated the underlying technique against the most common free keyloggers. This confirms the viability of our approach in practical scenarios.
We have also devised potential evasion techniques that may be adopted to circumvent our approach and proposed a heuristic to strengthen the effectiveness of our solution against more elaborated attacks. Extensive experimental results confirm that our technique is robust to both false positives and false negatives in realistic settings. Software keyloggers are a fast growing class of invasive software often used to harvest confidential information. One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a system. The

ability to run in unprivileged mode facilitates their implementation and distribution, but, at the same time, allows one to understand and model their behavior in detail.
Leveraging this characteristic, we propose a new detection technique that simulates carefully crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes. We have prototyped our technique as an unprivileged application, hence matching the same ease of deployment of a keylogger executing in unprivileged mode. We have successfully evaluated the underlying technique against the most common free keyloggers. This confirms the viability of our approach in practical scenarios. We have also devised potential evasion techniques that may be adopted to circumvent our approach and proposed a heuristic to strengthen the effectiveness of our solution against more elaborated attacks. Extensive experimental results confirm that our technique is robust to both false positives and false negatives in realistic settings.

2.3 Feasibility Study

The feasibility study is an evaluation and analysis of the potential of a proposed project which is based on extensive investigation and research to support the process of decision making.Feasibility studies aim to objectively and rationally uncover the strengths and weaknesses of an existing business or proposed venture, opportunities and threats present in the environment, the resources required to carry through, and ultimately the prospects for success.

2.3.1 Operational Study

Feasibility studies are crucial during the early development of any project and form a vital component in the business development process. Accounting and Advisory feasibility studies enable organizations to assess the viability, cost and benefits of projects before financial resources are allocated. They also provide independent project assessment and enhance project credibility.
In current research, it is common to use a Markov chain to model how once a country reaches a specific level of economic development, the configuration of structural factors, such as size of the commercial bourgeoisie, the ratio of urban to rural residence, the rate of political mobilization, etc., will generate a higher probability of transitioning from authoritarian to democratic regime.

2.3.2 Economical Study

In its simplest terms, the two criteria to judge feasibility are cost required and value to be attained The objective of the proposed approach is to utilize the strengths of Hidden Markov Models (dealing with temporal data) to complement the weaknesses of other classification techniques. Consequently, instead of finding single isolated patterns, we focus on understanding the relationships between these patterns. The proposed approach was evaluated with a case study. The target of the case study was to classify real drilling data generated by rig sensors. Experimental evaluation proves the feasibility and effectiveness of the approach.

2.3.3 Technical Study

The purpose of the study is to provide the necessary information to enable the company (and Enterprise Ireland) to come to firm conclusions regarding the project's viability.We show that a text retrieval system can be adapted to build a word image retrieval solution. This helps in achieving scalability.We represent the word image as histogram of visual words present in the image. This provides significant improvement in the performance.

2.4 Risk Analysis

Project Risk Analysis and Management is a process which enables the analysis and management of the risks associated with a project. Properly undertaken it will increase the likelihood of successful completion of a project to cost, time and performance objectives. The framework suggests an operational retrieval model that extends recent developments in the language modeling approach to information retrieval. A language model for each document is estimated, as well as a language model for each query, and the retrieval problem is cast in terms of risk minimization. The query language model can be exploited to model user preferences, the context of a query, synonomy and word senses. While recent work has incorporated word translation models for this purpose.

2.5 Project Scheduling

In project management, a schedule is a listing of a project's milestones, activities, and deliverables, usually with intended start and finish dates. Those items are often estimated in terms of resource allocation, budget and duration, linked by dependencies and scheduled events. A schedule is commonly used in project planning and project portfolio management parts of project management. Elements on a schedule may be closely related to the work

breakdown structure (WBS) terminal elements, the Statement of work, or a Contract Data Requirements List.

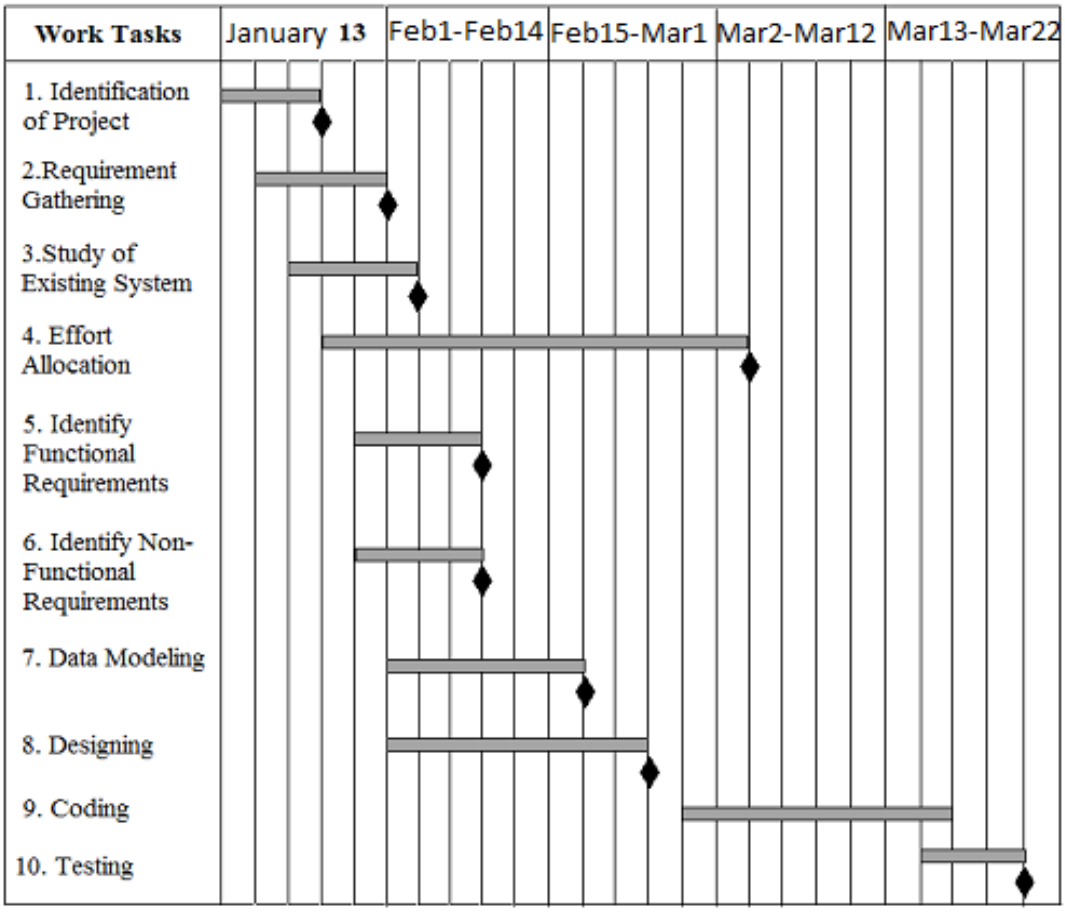

Figure 2.1: Project Scheduling

2.6 Effort Allocation

A criterion was presented to define the most eficient strategy for the exploration and maintenance of plant genetic resources. All of the three factors composing the efficiency. i.e., multiplicity of target populations. the amount of expenses, and goodness for individual populations of the conservation manipulation adopted, were incorporated in the present criterion. Sample size per target population for field collection was investigated on the basis of this criterion, leading to the conclusion that the number of visited populations rather than sample size per population determines the overall efficiency of a collection project as a whole. Without any particular reason, intensive sampling for a limited number of populations is not logical. A sample size as small as ten plants per site or population was estimated reasonable to cover a large target area.

Table 2.1: Effort Allocation

Work Tasks	Mayuri	Rupali	Minakshi	Vrushali	Work in %
Identification of Project	Yes	Yes	Yes	Yes	10
Requirement Gathering	Yes			Yes	10
Study of Existing System		Yes		Yes	10
Identify Requirements		Yes	Yes	Yes	5
Data Modeling	Yes	Yes	Yes		15
Designing	Yes	Yes	Yes	Yes	30
Coding	Yes	Yes	Yes		15
Testing	Yes		Yes		5

2.7 Summary

In this chapter the system analysis discuss in detail. In next chapter discuss about the system requirements specification.

Chapter 3

System Requirement Specification

This chapter describes the system requirements of the project. Section 3.1 describe Hardware Requirements. Software Requirements describe in section 3.2.

3.1 Hardware Requirement

1. Processor - Pentium III
2. Speed - 1.1 Ghz
3. RAM - 256 MB(min)
4. Hard Disk - 20 GB
5. Key Board - Standard Windows Keyboard
6. Mouse - Two or Three Button Mouse
7. Monitor - SVGA

3.2 Software Requirement

The software requirement are as follows:

3.2.1 Java-Front End

Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsys- tems Java platform. In case of the many programming languages, we can either compile or interpret the program in order to run that program on computer successfully. But Java is both compiled and the interpreted means any program is both compiled and interpreted. Thus to run any program we have to rst translate that program into an intermediate language called Java byte codes

using the Java Compiler (Javac). Java byte code is nothing but the platform-independent codes which is interpreted by the interpreter on the Java plat- form. Then next step is that java interpreter (JAVA) which parses and runs each Java byte code instruction on the computer. At the last compilation happens just once but the interpretation occurs each time when the program is executed.

Java byte codes are as the machine code and nothing but the instructions for the Java Conversion of Java Code to Machine Code Virtual Machine (JVM). Every Java interpreter is an implementation of the java virtual machine (JVM) even if that java interpreter is a development tool or a Web browser that can run applets. Thus this makes the java as the platform independent as we can compile our program into byte codes on any platform that has with a Java compiler, then in turn byte codes can then be run on any implementation of the JVM. Hence the motto of java is Virtual Machine (JVM). Every Java interpreter is an implementation of the java virtual machine (JVM) even if that java interpreter is a development tool or a Web browser that can run applets. Thus this makes the java as the platform independent as we can compile our program into byte codes on any platform that has with a Java compiler, then in turn byte codes can then be run on any implementation of the JVM. Hence the javas motto comes here that Write once, Run anywhere.

3.2.2 JDK 1.6 and Above

JDK stands for Java Development Kit. This is a collection of software that allows a developer to create and deploy an application written in Java. JDK 7 is a superset of JRE 7, and contains everything that is in JRE 7, plus tools such as the compilers and de-Buggers necessary for developing applets and applications. JRE 7 provides the library the Java Virtual Machine (JVM), and other components to run applets and applications written in the Java programming language.

3.2.3 Netbeans

The Net Beans Platform is a reusable framework for simplifying the development of Java Swing desktop applications. The Net Beans IDE bundle for Java SE contains what is needed to start developing Net Beans plugins and Net Beans Platform based applications; no additional SDK is required. Applications can install modules dynamically. Any application can include the Update Center module to allow users of the application to download digitally-signed upgrades and new features directly into the running application. Reinstalling an upgrade or a new release does not force users to download the entire application again. We have used Net beans for GUI Designing and coding purpose, Net beans helps to develop an application is very fast and efficient manner, it is a IDE with complete Intelligence which

helps to understand many parts of program.

3.3 Summary

This chapter described about hardware software requirement. Next chapter describes designing part of the image retrieval system all the UML diagrams, and architecture of the project.

Chapter 4

System Design

In this chapter we are making the design of the system. Flowchart, how the data flows from the system architecture, and all the UML diagrams including class diagram, use case diagram, sequence diagram, activity data diagram, component diagram, deployment diagram of the project.
In section 4.1 system architecture of keylogger detection system is describe. The UML diagrams are shown in section 4.2 .

4.1 System Architecture

The keyloggers are covert security threat to the privacy and identity of users. The attackers are exploring different techniques of keylogging using hardware keyloggers, software keyloggers and screen capturing software to steal the user sensitive data. The Incognizance of the user is imposing greater risk. Keyloggers are implanted on a machine to intentionally monitor theuser activity by logging keystrokes and eventually delivering them to a third party. Software keyloggers are a fast growing class of invasive software often used to harvest confidential information. One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all the keystrokes typed by the users of a system.
The fig.4.1 shows how to spread keylogger on our system.

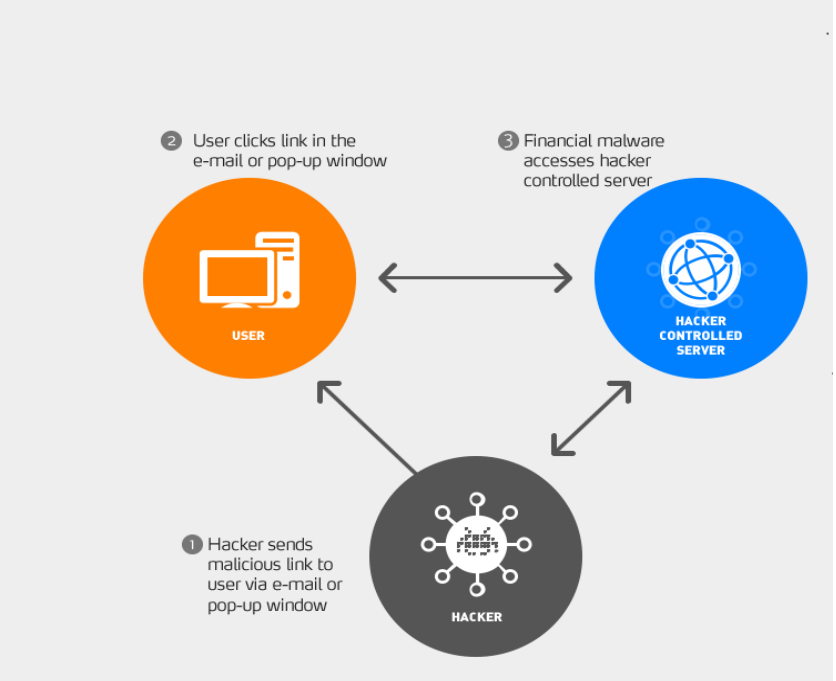

Figure 4.1: How keylogger spread on our system

To overcome this problem, we have proposed a keylogger detection model. In this model solution to Keylogger and Screen Recording Software has been proposed by using the concept of scanning of whole system processes. It searches the all memory, fie, registry, content base file and deskstop icon. After searching, if kelogger is present then it will be detected.
The fig.4.2 shows the scanning/searching process of system.

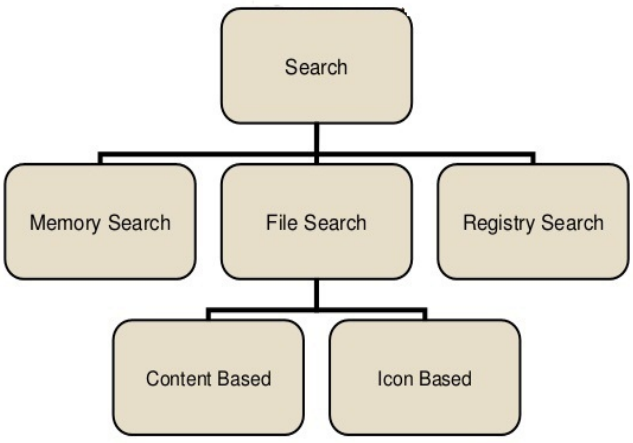

Figure 4.2: Scanning processes of system

In the scanning processes, if the keylogger is detected then it will be remove by our keylogger detection software and our system is free from kelogger. So our system is safe. The modules which are present in this system is that we detect keyloggers running as unprivileged user-space processes. To match the same deployment model, our technique is entirely implemented in an unprivileged process.

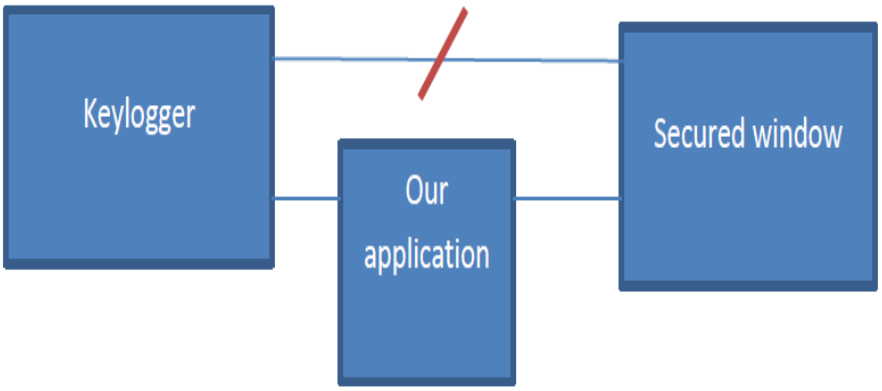

Figure 4.3: Secured system after remove the keylogger

The above fig.4.3 shows secured window after remove the keylogger. As a result, our solution is portable, easy to install, and yet very effective.

4.2 UML Diagrams

4.2.1 Usecase Diagram

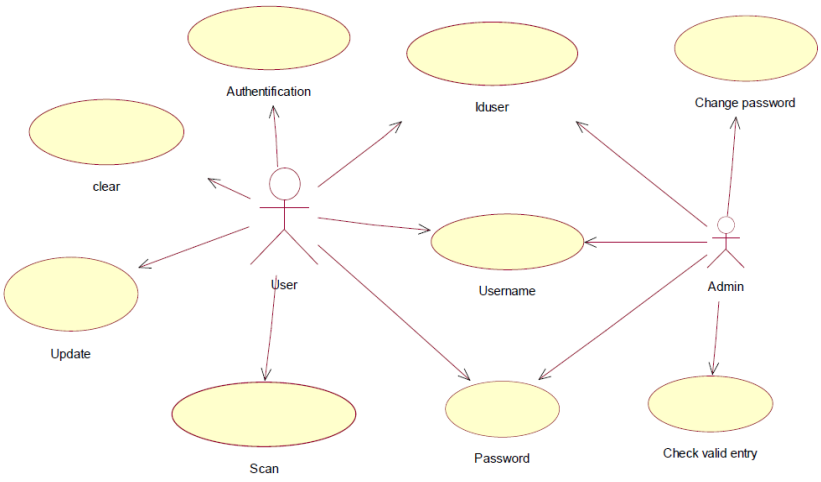

Figure 4.4: Usecase Diagram

4.2.2 Class Diagram

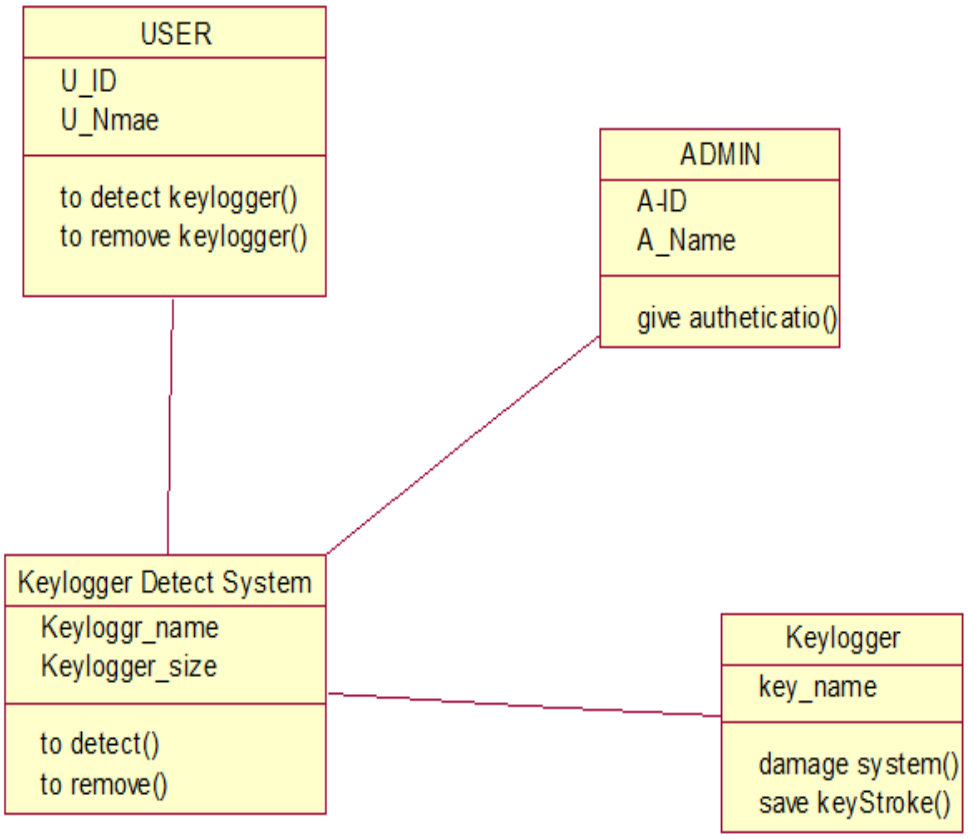

Figure 4.5: Class Diagram

4.2.3 Sequence Diagram

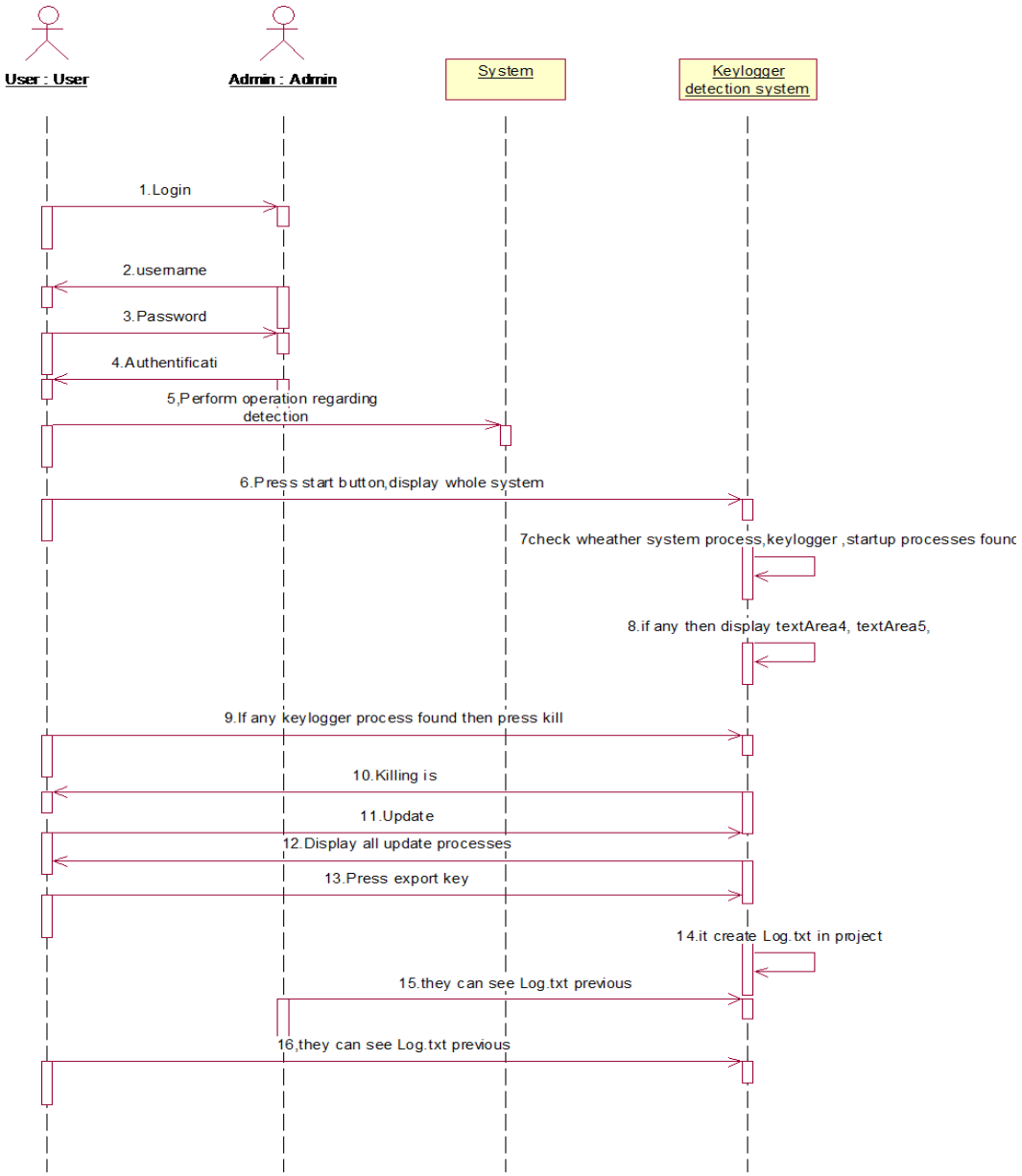

Figure 4.6: Sequence Diagram

4.2.4 Activity Diagram

Figure 4.7: Activity Diagram

4.2.5 Component Diagram

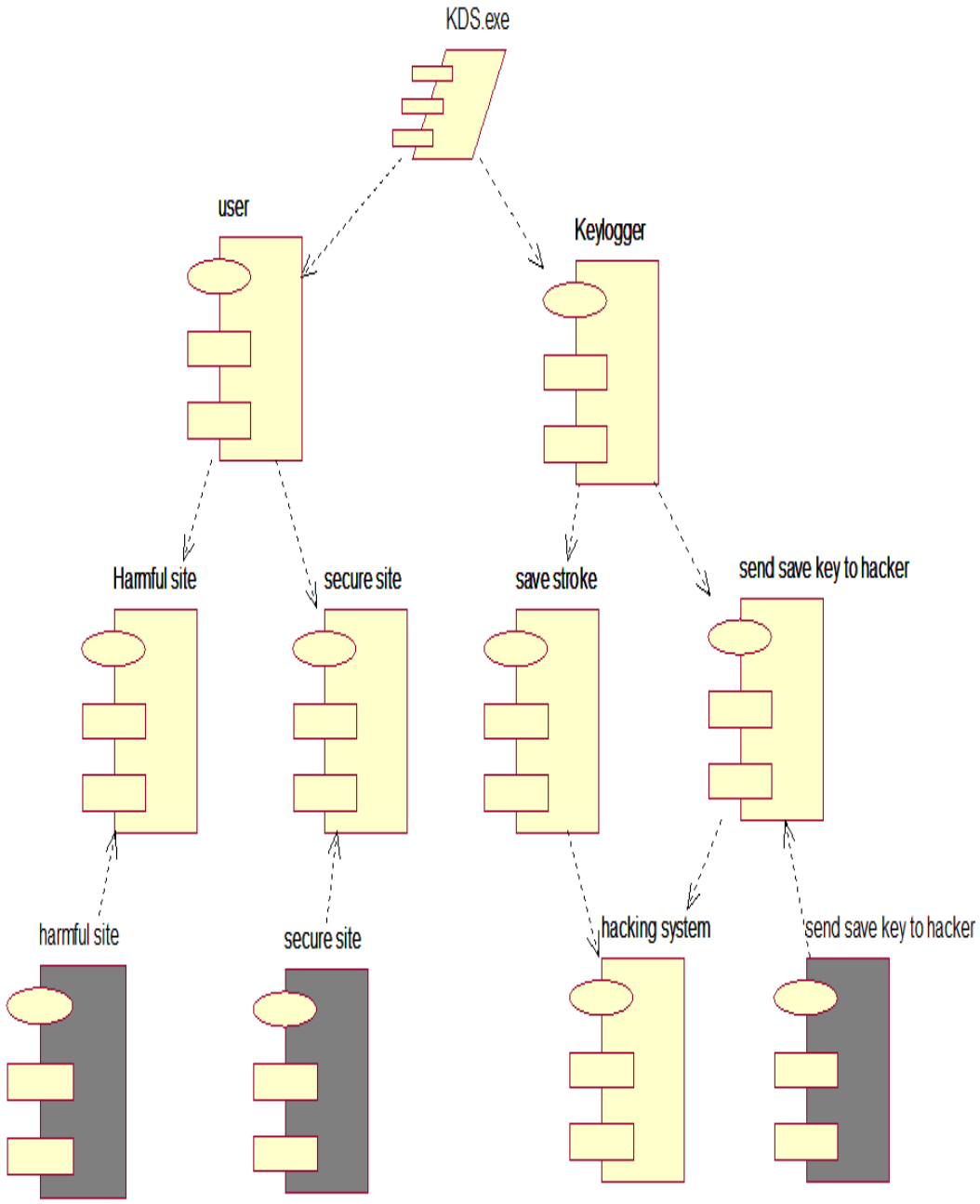

Figure 4.8: Component Diagram

4.2.6 Deployement Diagram

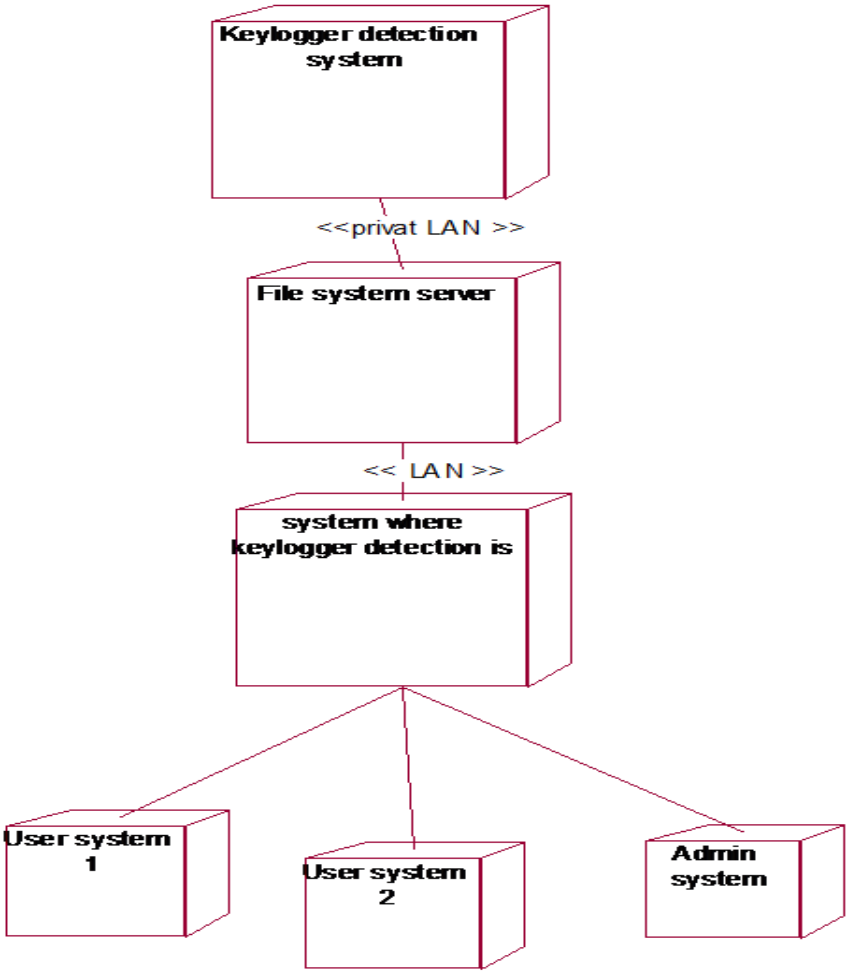

Figure 4.9: Deployement Diagram

4.2.7 State Diagram

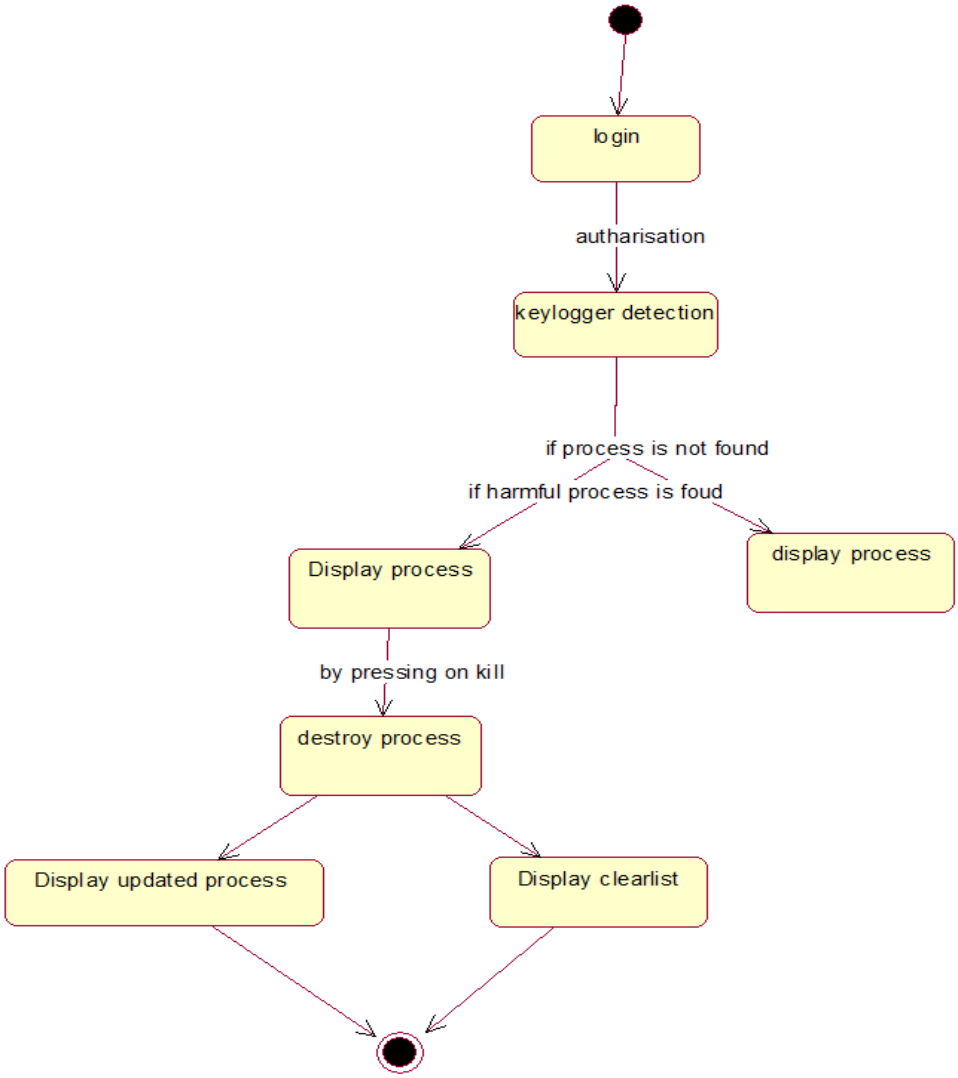

Figure 4.10: State Diagram

4.3 Summary

In this chapter, all the designing part of the project included by using this designing phase user can modeled it practically. All the architecture and UML diagrams are described into it. Next chapter describes the conclusion and future scope of the project.

Chapter 5

Implementation

Implementation is the realization of an application, or execution of a plan, idea, model,design, specialisation, standard, algorithm, or policy. Implementation is the carrying out, exexecution, or practice of a plan, a method, or any design for doing something. As such, implementation is the action that must follow any preliminary thinking in order for something to actually happen. In an information technology context, implementation encompasses all the processes involved in getting new software or hardware operating properly in its environment, including installation, configuration, running, testing, and making necessary changes.
In this chapter, section 5.1 describe about implementation details. The flow of system development is describe in section 5.2.

5.1 Implementation Details

```
 /*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package keyloggerdetection;
import java.awt.Color;
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.File;
import java.io.FileReader;
import java.io.InputStreamReader;
import javax.swing.JOptionPane;
```

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {

jLabel9.setText("<html>");


 //jButton5.setEnabled(false);
  jProgressBar3.setVisible(true);
   repaint();
   for(int i=0;i<=100;i+=5)
   {
   jProgressBar3.setValue(i);
   try
    {
     jProgressBar3.paintImmediately(0,0,100,100);
     Thread.sleep(100);
     jProgressBar3.setStringPainted(true);
     }
     catch(Exception e)
     {}
    }
 try
  {
 String process;
  Process p = Runtime.getRuntime().exec(System.getenv("windir")+"\\system32\\"+"task
  BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream()
  while ((process = input.readLine()) != null)
  {
    jTextArea1.setText(jTextArea1.getText() + "\n" + process);
  }
  input.close();
  }
  catch(Exception ex)
  {
  ex.printStackTrace();
  }
try
{
FileInputStream fstream = new FileInputStream("C:\\project\\system.txt");
BufferedReader br = new BufferedReader(new InputStreamReader(fstream));

 String strLine;
 while ((strLine = br.readLine()) != null)
 {
 if(jTextArea1.getText().contains(strLine))
 {
  jTextArea5.setText(jTextArea5.getText()+ "\n" + strLine);
 }
```

```
 }
 if((jTextArea1.getText()).equals("strLine"))
 {
  jTextArea5.setText(jTextArea5.getText()+ "\n" + strLine);
 }
 br.close();
 }
catch(Exception e)
{
 e.printStackTrace();
}
JOptionPane.showMessageDialog(null,"Reliable processes are found");
jLabel9.setText(jLabel9.getText()+ "<br>" + "Reliable processes are found");
try
{
 FileInputStream fstream = new FileInputStream("C:\\project\\malcious.txt");
 BufferedReader br = new BufferedReader(new InputStreamReader(fstream));

 String strLine;
 while ((strLine = br.readLine()) != null)
 {
 if(jTextArea1.getText().contains(strLine))
 {
 jTextArea4.setText(jTextArea4.getText()+ "\n" + strLine);
 }
 }
 if((jTextArea1.getText()).equals("strLine"))
 {
 jTextArea4.setText(jTextArea4.getText()+ "\n" + strLine);
 }
 br.close();
}
catch(Exception e)
{
 e.printStackTrace();
}
JOptionPane.showMessageDialog(null,"Keylogger is found");
jLabel9.setText(jLabel9.getText()+ "<br>" + "Keylogger is found");
jProgressBar2.setVisible(true);
 repaint();
 for(int i=0;i<=100;i+=5)
 {
  jProgressBar2.setValue(i);
  try
  {
  jProgressBar2.paintImmediately(0,0,100,100);
   Thread.sleep(100);
```

```
   jProgressBar2.setStringPainted(true);
  }
catch(Exception e)
{ }
}
try
{
//System.out.println("Hello");
String proc;
Process p = Runtime.getRuntime().exec("wmic startup get caption,command");
BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream()))
 while ((proc = input.readLine()) != null)
 {
  jTextArea2.setText(jTextArea2.getText() + "\n" + proc);
 }
 input.close();
 }
 catch(Exception ex)
 {
 ex.printStackTrace();
 }
try
 {
FileInputStream fstream = new FileInputStream("C:\\project\\reliable.txt")
BufferedReader br = new BufferedReader(new InputStreamReader(fstream));

String strLine;
while ((strLine = br.readLine()) != null)
{
if(jTextArea2.getText().contains(strLine))
 {
 jTextArea3.setText(jTextArea3.getText()+ "\n" + strLine);
 }
}
br.close();
}
catch(Exception e)
{
 e.printStackTrace();
}
JOptionPane.showMessageDialog(null,"System processes are found");
jLabel9.setText(jLabel9.getText()+ "<br>" + "System processes are found");
jLabel9.setText(jLabel9.getText()+ "</html>");
}
private void jButton6ActionPerformed(java.awt.event.ActionEvent evt)
 {
 // TODO add your handling code here:
```

```
  // jButton6.setEnabled(false);
  jProgressBar1.setVisible(true);
  repaint();
  for(int i=0;i<=200;i+=5)
  {
  jProgressBar1.setValue(i);
  try
  {
   jProgressBar1.paintImmediately(0,0,100,100);
   Thread.sleep(100);
   jProgressBar1.setStringPainted(true);
   }
   catch(Exception e)
   {}
  }
 try
 {
  //System.out.println("Hello");
  String proc;
  Process p = Runtime.getRuntime().exec("wmic startup get caption,command");
  BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream()
  while ((proc = input.readLine()) != null)
 {
  jTextArea2.setText(jTextArea2.getText() + "\n" + proc);
 }
 input.close();
}
catch(Exception ex)
{
 ex.printStackTrace();
}
try
{
 FileInputStream fstream = new FileInputStream("C:\\project\\reliable.txt");
 BufferedReader br = new BufferedReader(new InputStreamReader(fstream));
 String strLine;
while ((strLine = br.readLine()) != null)
{
if(jTextArea2.getText().contains(strLine))
{
jTextArea3.setText(jTextArea3.getText()+ "\n" + strLine);
}
}
br.close();
}
catch(Exception e)
{
```

```
e.printStackTrace();
}
JOptionPane.showMessageDialog(null,"System processes are found");
jLabel9.setText(jLabel9.getText()+ "\n" + "System processes are found" +"\n");
}
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
jTextArea1.setText(null);
jTextArea2.setText(null);
jTextArea3.setText(null);
jTextArea4.setText(null);
jTextArea5.setText(null);
jProgressBar3.setVisible(false);
jProgressBar2.setVisible(false);
jProgressBar1.setVisible(false);
//jProgressBar4.setVisible(false);
//jProgressBar5.setVisible(false);
jLabel9.setText("");
}
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
//jLabel9.setText("<html>");
jTextArea1.setText(null);
jTextArea2.setText(null);
jTextArea3.setText(null);
jTextArea4.setText(null);
jTextArea5.setText(null);
jProgressBar3.setVisible(false);
jProgressBar2.setVisible(false);
jProgressBar1.setVisible(false);
//jProgressBar4.setVisible(false);
//jProgressBar5.setVisible(false);
jLabel9.setText("");
//jButton5.setEnabled(false);
jLabel9.setText("<html>");

//jButton5.setEnabled(false);
 jProgressBar3.setVisible(true);
 repaint();
 for(int i=0;i<=100;i+=5)
 {
  jProgressBar3.setValue(i);
  try
  {
  jProgressBar3.paintImmediately(0,0,100,100);
   Thread.sleep(100);
   jProgressBar3.setStringPainted(true);
```

```
   }
   catch(Exception e)
   {

   }
  }
try
{
 String process;
 Process p = Runtime.getRuntime().exec(System.getenv("windir")+"\\system32\\"+"taskl
 BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream())
 while ((process = input.readLine()) != null)
 {
 jTextArea1.setText(jTextArea1.getText() + "\n" + process);
 }
 input.close();
 }
 catch(Exception ex)
 {
 ex.printStackTrace();
 }
try
{
 FileInputStream fstream = new FileInputStream("C:\\project\\system.txt");
 BufferedReader br = new BufferedReader(new InputStreamReader(fstream));
 String strLine;
  while ((strLine = br.readLine()) != null)
   {
    if(jTextArea1.getText().contains(strLine))
    {
     jTextArea5.setText(jTextArea5.getText()+ "\n" + strLine);
    }
   }
 if((jTextArea1.getText()).equals("strLine"))
 {
  jTextArea5.setText(jTextArea5.getText()+ "\n" + strLine);
 }
 br.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
  JOptionPane.showMessageDialog(null,"Reliable processes are found");
  jLabel9.setText(jLabel9.getText()+ "<br>" + "Reliable processes are found");
 try
 {
```

```
FileInputStream fstream = new FileInputStream("C:\\project\\malcious.txt");
BufferedReader br = new BufferedReader(new InputStreamReader(fstream));
 String strLine;
 while ((strLine = br.readLine()) != null)
 {
 if(jTextArea1.getText().contains(strLine))
 {
 jTextArea4.setText(jTextArea4.getText()+ "\n" + strLine);
 }
 }
 if((jTextArea1.getText()).equals("strLine"))
 {
 jTextArea4.setText(jTextArea4.getText()+ "\n" + strLine);
 }
   br.close();
   }
catch(Exception e)
{
e.printStackTrace();
}
JOptionPane.showMessageDialog(null,"Keylogger is not found");
jLabel9.setText(jLabel9.getText()+ "<br>" + "Keylogger is not found");
jProgressBar2.setVisible(true);
repaint();
for(int i=0;i<=100;i+=5)
{
jProgressBar2.setValue(i);
try
{
jProgressBar2.paintImmediately(0,0,100,100);
 Thread.sleep(100);
 jProgressBar2.setStringPainted(true);
 }
 catch(Exception e)
 {}
}
try
{
 //System.out.println("Hello");
 String proc;
Process p = Runtime.getRuntime().exec("wmic startup get caption,command");
BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream())
while ((proc = input.readLine()) != null)
{
 jTextArea2.setText(jTextArea2.getText() + "\n" + proc);
}
input.close();
```

```
 }
 catch(Exception ex)
 {
 ex.printStackTrace();
 }
try
{
FileInputStream fstream = new FileInputStream("C:\\project\\reliable.txt");
BufferedReader br = new BufferedReader(new InputStreamReader(fstream));
String strLine;
while ((strLine = br.readLine()) != null)
{
if(jTextArea2.getText().contains(strLine))
 {
  jTextArea3.setText(jTextArea3.getText()+ "\n" + strLine);
 }
}
br.close();
}
catch(Exception e)
{
e.printStackTrace();
}
JOptionPane.showMessageDialog(null,"System processes are found");
jLabel9.setText(jLabel9.getText()+ "<br>" + "System processes are found");
jLabel9.setText(jLabel9.getText()+ "</html>");
}
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
try
{
Runtime.getRuntime().exec("taskkill /F /IM spkl.exe");
}
catch(Exception e)
{
 e.printStackTrace();
}
}
}
```

5.2 Flow of System Developement

The flow of system development describes how you performed and develop the proposed methods and procedure. Basic functionality, execution and steps of system execution are:

- User login in to system.

- The Keylogger System will be start.
- Click on start button for detecting keylogger processes.
- If keylogger are detected then click on kill button for remove this keylogger from the system.
- Click on update button if you want to update all running processes.
- The expert log button save previous status of running processes.

5.3 Summary

This chapter describes the information about implementation, implementation detail, how of system development, advantages and disadvantages of the project. Next chapter describes How to implement testing and test case and test result of the project.

Chapter 6

System Testing

Tests are the individual tests specified in a test plan document. Each test is typically described by
1.An initial system state.
2.A set of actions to be performed.
3.The expected results of the test.

All the testing of keylogger detection system is done by tester is explained in this chapter. Section 6.1 describes How to implement testing. Test cases and Test result is described in section 6.2.

6.1 How to implement testing

Test cases are planned in accordance to the test process and documented with detailed test descriptions. These test cases use cases based on projected operational mission scenarios. The testing process also includes stress/ load testing for stability purpose (i.e., at 95 CPU use, system stability is still guaranteed). The test process thoroughly tests the interfaces and modules. Software testing includes a traceable white box testing, black box testing and other test processes verifying implemented software against design documentation and requirements specified.

Types of Testing:

1. White Box Testing: A level of white box test coverage is specified that is appropriate for the software being tested. The white box and other testing uses automated tools to instrument the software to measure test coverage. The program code is examined for defects. It based on design and implementation structures. It deals with internal logic and structure of the code.

2. Black Box Testing: A black box test of integration builds includes functional, interface, error recovery, stress and out-of-bounds input testing. All black box software tests are

traced to control requirements. In addition to static requirements, a black box of a fully integrated system against scenario sequences of events is designed to model field operation. Performance testing for systems is integrated as an integral part of the black box test process.

6.2 Test Cases and Test Results

1. Unit Testing:

   Validations were made for each single module and each module was tested independently. Maximum error detection was made at this stage. Proper now of in bound and out bound information was made. **Java Server Pages:** Java Server Pages include validation for the input from the user. These inputs must satisfy this validation for sending request to the server. For this purpose following test cases are used.
   Input: user name, password.
   Expected Output-: User login is creates successfully.
   Validation:-login page checks whether these fields are filled or not. If the all fields are filled then user login successfully.
   Input -: close.
   Expected Output-: It ends the keylogger detection system.
   Validation-: User session is terminated.

2. Intregated Testing:

   Integration testing was carried out to check whether each module works properly with the integrated module. Different inputs were given to the secondary module from the primary, to check whether it works properly for each input which comes from the higher level. Both Bottom-up and Top-down integration tests were carried out. Tomcat server, Client Browser after selecting create account or login operation, separate windows for both gets displayed on screen. It will show that the connection between client and server gets successfully created. Otherwise connection gets failed. Tomcat server, Client Browser and Authentication User can create account, login and logout after integrating above modules. Information regarding new user stored into the database.
   Input-: Login of user.
   Output-: Information regarding user is get i.e. username, password. Validation-: User get successfully login.
   Input-: User login name and password

Output-: Profile information regarding user get displayed in the browser.

Validation-: User can successfully interact with system.

Authentication, accessing software Every user interacting with our system should authorized user. If user has already created his account then he can login directly. Information regarding user is maintained by database. User can upload photos and documents, download or delete as per requirement after login. Many users can communicate with system at same time.

6.3 Summary

This chapter describes how to implement testing and test case and test result of the project. Next chapter describes the Result and Analysis of Un Privilage.

Chapter 7

Result and Analysis

All the result reguarding project is explained in Result and Analysis. All the analysis is done by tester is explaine further. Section 7.1 sample snapshot of important processing and its explanation. Summary of the chapter is explained in section 7.2. Sample snapshot of important processing and its explanation is given below.

7.1 Sample snapshot of important processing and its explanation

This login window which provided for user to login to system. After filling the username and password field then click on to the login button for next process.

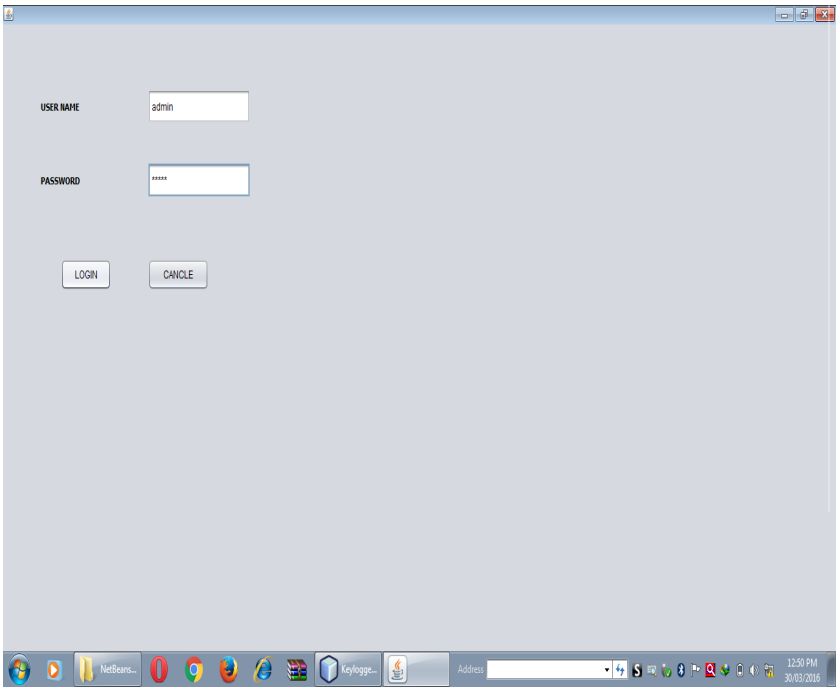

Figure 7.1: login window

After click on login button then keylogger system should be open shown in below figure.

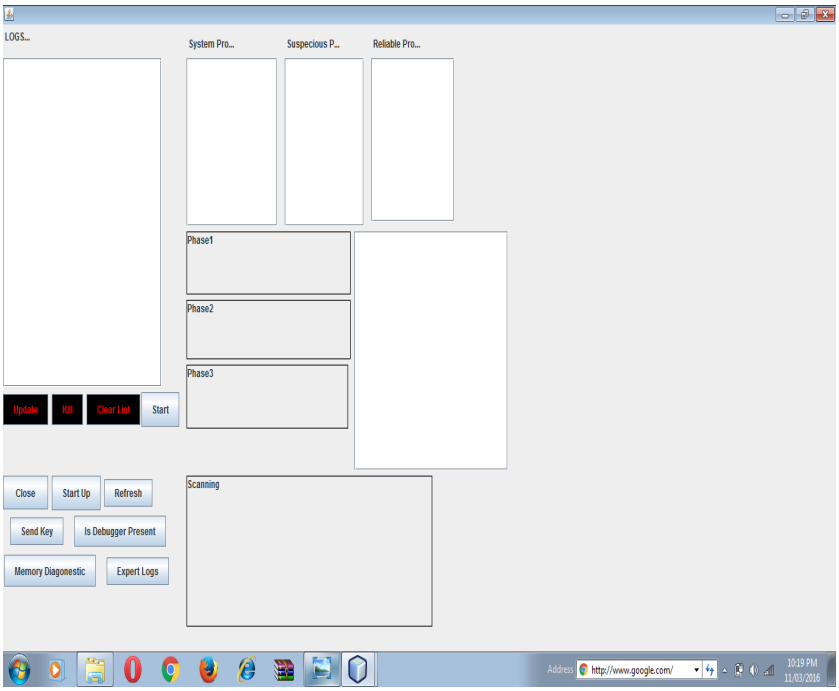

Figure 7.2: Startup initial window

After click on start button progress bar is completed and load the all running processes of system.

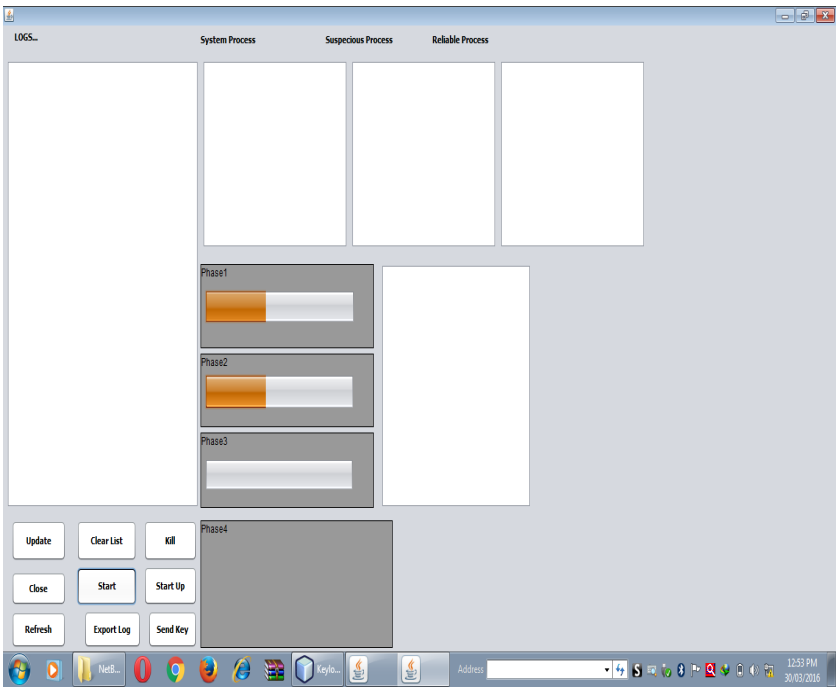

Figure 7.3: running processes window

After click on start button progress bar is completed and load the all running processes

of system as shown below.

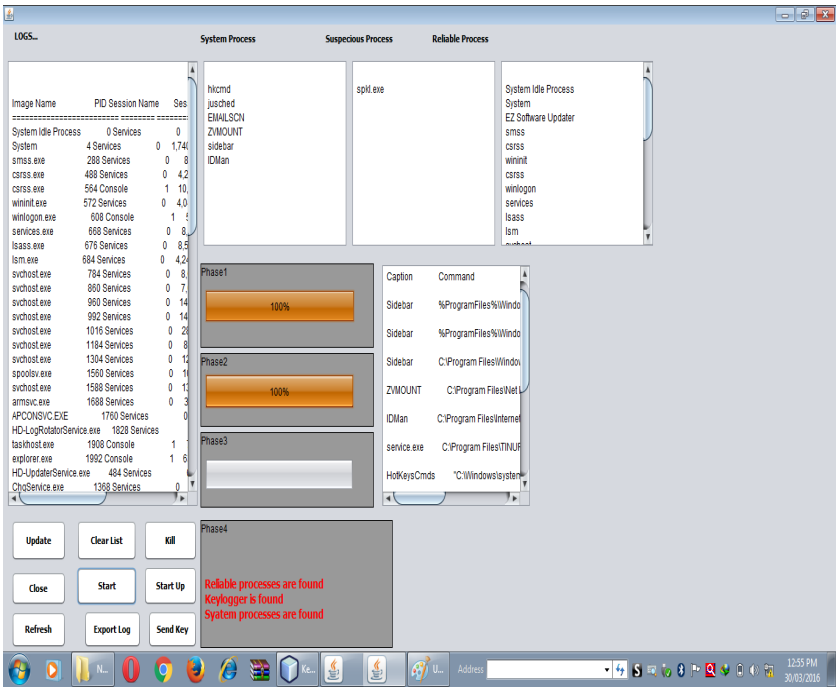

Figure 7.4: running processes window

After click on kill button the detection system remove the keyloggers. After click on update button get the updated current running processes as shown below.

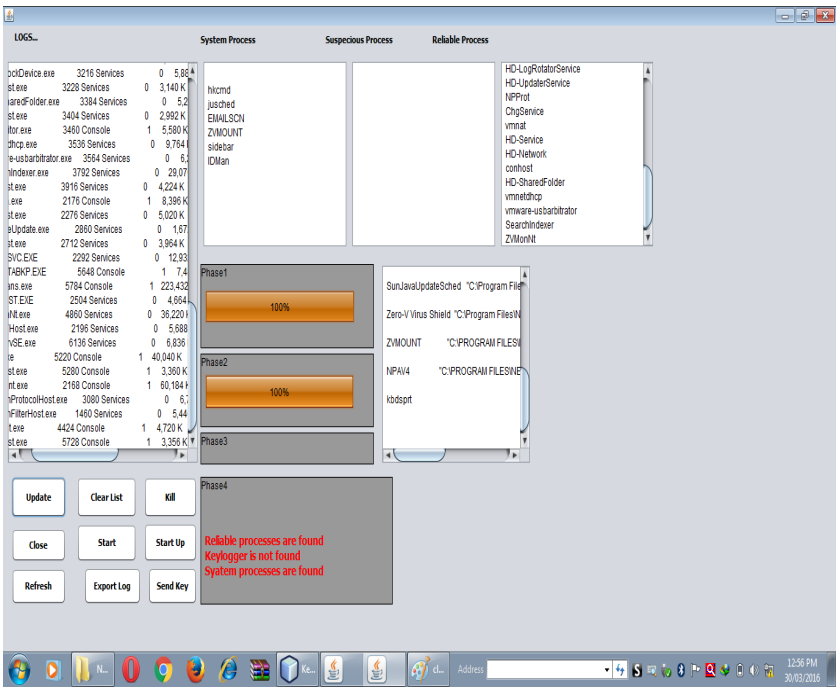

Figure 7.5: updated running processes window

After click on expert log button it stored the previous record of running proccess as

shown below.

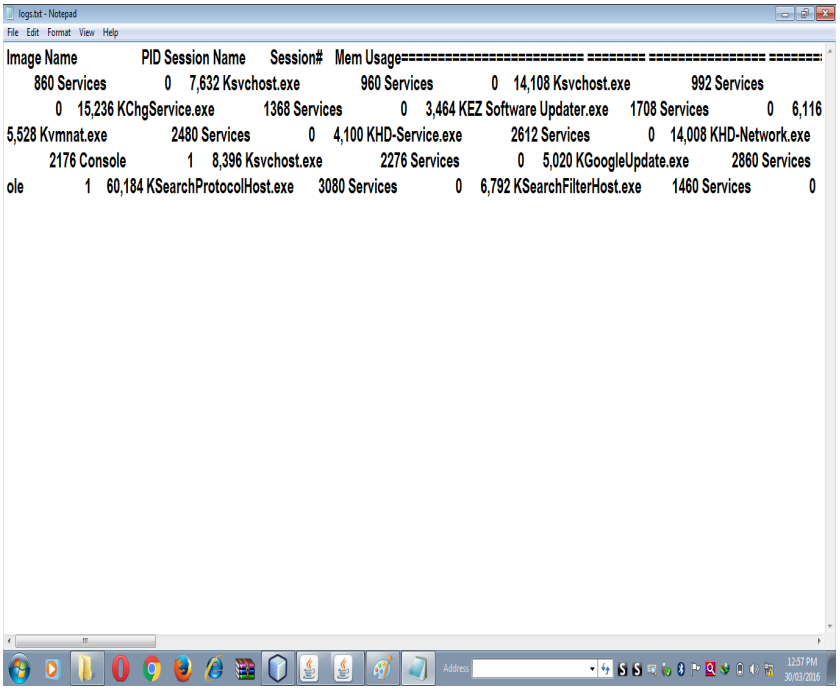

Figure 7.6: expert log window

After click on clearlist button it clears the all running proccesses detected by the system as shown below.

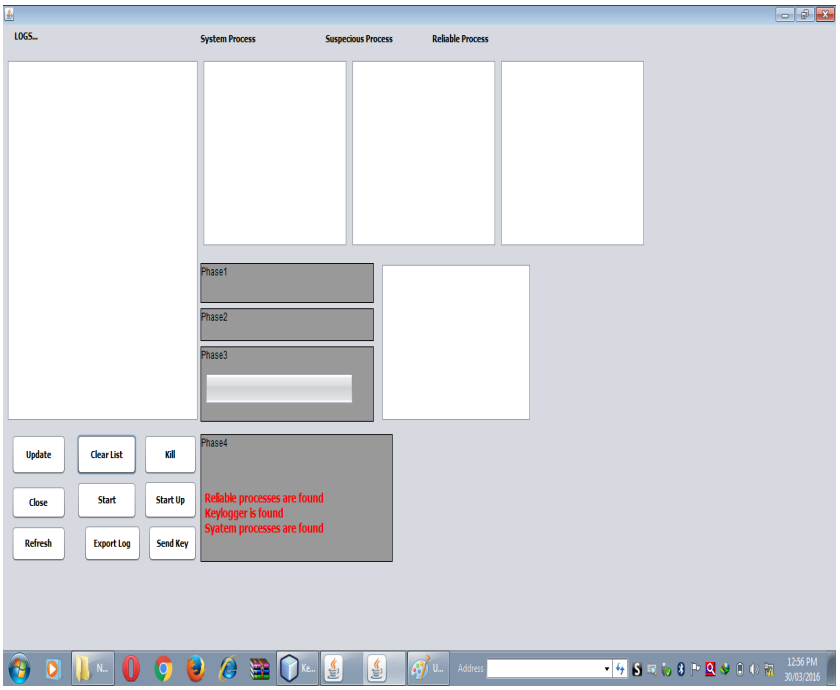

Figure 7.7: clear processes window

then after click on exit button software is closed.

7.2 Summary

This chapter describes the Result and Analysis of Keylogger detection system. Next Chapter Contains Conclusion and future scope of the project.

Chapter 8

Conclusion

This application allow maximum flexibility to the user to work on internet. There is software which install on the system, continuously monitoring to find the existing key-logger which is present in the systems and give alert to prevent them. It is most efficient way to get secured from hacking. As a result, our solution is portable, easy to install, and yet very effective. In addition, the proposed detection technique is completely black-box, i.e., based on behavioral characteristics common to all keyloggers. In other words, our technique does not rely on the internal structure of the keylogger or the particular set of APIs used for this reason, our solution is of general applicability. We have prototyped our approach and evaluated it against the most common free keyloggers. Our approach has proven effective in all the cases.

Bibliography

[1] S. Sagiroglu and G. Canbek, Keyloggers, IEEE Technology and Society Magazine, vol. 28, no. 3, pp. 10 17, fall 2009.

[2] ThinkGeek.com,Spykeylogger,2010(accessed May 8, 2010), http://www.thinkgeek.com/gadgets/security/c49f/.

[3] G. Hoglund and J. Butler, Rootkits: Subverting the Windows Kernel. Addison-Wesley Professional, 2005.

[4] C.Wood and R. K. Raj, Sample keylogging programming projects, 2010 , http:// www.cs.rit.edu/ rkr/ keylogger2010.

[5] Bauer, Michael D., Chapter 10 (System Log Management and Monitoring) of Building Secure Servers with LINUX, OReilly, 2002.

[6] Babbin, Jacob et al, Security Log Management: Identifying Patterns in the Chaos, Syngress, 2006.

http://www.sans.org/rr/papers/52/540.pdf

[7] Stout, Kent,Central Logging with a Twist of COTS in a Solaris Environment.,SANS Institute, March 2002, URL: http://www.sans.org/rr/papers/52/540.pdf

[8] G. Canbek, "Analysis, design and implementation of keyloggers and anti-keyloggers" Gazi University, Institute Of Science And Technology, M.Sc. thesis (in Turkish), Sept. 2005, pp. 103

[9] Williams, "I know what you did last logon: Monitoring software, spyware, and privacy," microsoft Security News., vol. 4, no. 6, June 2007.

[10] S. Seref and C. Gurol "Keyloggers increasing threats to computer security and privacy" IEEE Technology and society magazine,2009,pp.10-17.