**CHAPTER-1**

**PRELIMINARY**

**INVESTIGATION**

1. **INTRODUCTION**

In the course of doing business, sometimes sensitive data must be handed over to supposedly trusted third parties. For example, a hospital may give patient records to researchers who will devise new treatments. Similarly, a company may have partnerships with other companies that require sharing customer data. Another enterprise may outsource its data processing, so data must be given to various other companies. We call the owner of the data the distributor and the supposedly trusted third parties the agents. Our goal is to detect when the distributor’s sensitive data has been leaked by agents, and if possible, to identify the agent that leaked the data.

Data Leakage can occur through a variety of methods - some are simple, some complex. As such, there is no single ''silver bullet'' to control Data Leakage. Data leakage detection is an increasingly important part of any organization’s ability to manage and protect critical and confidential information. Examples of critical and confidential data that applications can access include: Intellectual Property, Corporate Data, and Customer Data. Watermarks are very useful in a relational database, which involves some modification of data. The goal of our paper is to detect when the distributor’s sensitive data has been leaked by agents and show the probability for identifying the agent that leaked the data using encrypted fake objects.

**UNOBTRUSIVE TECHNIQUES:**

In this section we develop a model for assessing the “guilt” of agents. We also present algorithms for distributing objects to agents, in a way that improves our chances of identifying a leaker. Finally, we also consider the option of adding “fake” objects to the distributed set. Such objects do not correspond to real entities but appear realistic to the agents. In a sense, the fake objects act as a type of watermark for the entire set, without modifying any individual members. If it turns out an agent was given one or more fake objects that were leaked, then the distributor can be more confident that agent was guilty.

**GUILTY AGENTS**

Suppose that after giving objects to agents, the distributor discovers that a set S \_ T has leaked. This means that some third party, called the target, has been caught in possession of S. For example, this target may be displaying S on its website, or perhaps as part of a legal discovery process, the target turned over S to the distributor. Since the agents U1; . . . ; Un have some of the data, it is reasonable to suspect them leaking the data. However, the agents can argue that they are innocent, and that the S data were obtained by the target through other means.

For example, say that one of the objects in S represents a customer X. Perhaps X is also a customer of some other company, and that company provided the data to the target. Or perhaps X can be reconstructed from various publicly available sources on the web. Our goal is to estimate the likelihood that the leaked data came from the agents as opposed to other sources. Intuitively, the more data in S, the harder it is for the agents to argue they did not leak anything. Similarly, the “rarer” the objects, the harder it is to argue that the target obtained them through other means. Not only do we want to estimate the likelihood the agents leaked data, but we would also like to find out if one of them, in particular, was more likely to be the leaker. For instance, if one of the S objects was only given to agent U1, while the other objects were given to all agents, we may suspect U1 more. The model we present next captures this intuition. We say an agent Ui is guilty and if it contributes one or more objects to the target. We denote the event that agent Ui is guilty by Gi and the event that agent Ui is guilty for a given leaked set S by Gi|S. Our next step is to estimate Pr{Gi|S}, i.e., the probability that agent Ui is guilty given evidence S.

**FAKE OBJECTS**

The distributor may be able to add fake objects to the distributed data in order to improve his effectiveness in detecting guilty agents. However, fake objects may impact the correctness of what agents do, so they may not always be allowable. The idea of perturbing data to detect leakage is not new. However, in most cases, individual objects are perturbed, e.g., by adding random noise to sensitive salaries, or adding a watermark to an image. In our case, we are perturbing the set of distributor objects by adding fake elements. In some applications, fake objects may cause fewer problems that perturbing real objects.

For example, say that the distributed data objects are medical records and the agents are hospitals. In this case, even small modifications to the records of actual patients may be undesirable. However, the addition of some fake medical records may be acceptable, since no patient matches these records, and hence, no one will ever be treated based on fake records. Our use of fake objects is inspired by the use of “trace” records in mailing lists. In this case, company A sells to company B a mailing list to be used once (e.g., to send advertisements). Company A adds trace records that contain addresses owned by company A. Thus, each time company B uses the purchased mailing list, A receives copies of the mailing. These records are a type of fake

Objects that help identify improper use of data.

The distributor creates and adds fake objects to the data that he distributes to agents. We let Fi \_ Ri be the subset of fake objects that agent Ui receives. As discussed below, fake objects must be created carefully so that agents cannot distinguish them from real objects.

**Aim of the Project**

The aim of the project is to overcome data allocation problem and to send secured data for third party agent. Our goal is to detect when the distributor’s sensitive data has been leaked by agents, and if possible to identify the agent that leaked the data. We develop unobtrusive techniques for detecting leakage of a set of objects or records.

**Objective**

1. To detect when the distributor’s sensitive data has been leaked by agents, and if possible, to identify the agent that leaked the data.
2. To secure our data from a third party.

**Scope of Project**

Perturbation is a very useful technique where the data is modified and made “less sensitive” before handed of agents. We develop unobtrusive techniques for detecting leakage a set objects or records.

We develop a model for assessing the “guilt” od agent. We also present algorithms for distributing objects to agents, in a way that improves our chances of identifying a leaker. Finally, we also consider the option of adding “fake” objects to the distributed set. Such objects do not correspond to real entities but appear realistic to the agent. In a sense, the fake object acts as a type of watermark for the entire set, without modifying any individual member. If it turns out an agent was given one or more fake object that were leaked, then the distributor can be more confident that agent was guilty.

**CHAPTER-2**

**LITERATURE**

**SURVEY**

**Sandip A. Kale** describes the results of implementation of Data Leakage Detection Model. Currently watermarking technology is being used for the data protection. But this technology doesn’t provide the complete security against date leak age. This paper includes the difference between the watermarking & data leakage detection model’s technology. This paper leads for the new technique of research for secured data transmission & detection, if it gets leaked.

**S Ramkumar et al**, investigate and utilized the characteristic of the group movement of objects to explore the group relationship and tracking them. The goal is to efficiently mine the group movement activity using clustering and sequential pattern mining. Clustering was applied to find both groups of similar teams and similar individual members. Sequential pattern mining was used to extract sequences of frequent events. To enable the continuous monitoring the group object movement, the system introduces a special technique called minor clustering and Cluster Assembling algorithm. Several solutions on route were implemented, but those methods were energy consumed. In order to reduce the energy, the proposed system used data mining methods to effectively handle the group movement of objects.

**Chandni Bhatt et al**,” study a data distributor has given sensitive data to a set of supposedly trusted agents. Sometimes data is leaked and found in unauthorized place e.g., on the web or on somebody's laptop. For example, a hospital may give patient records to researchers who will devise new treatments. Similarly, a company may have partnerships with other companies that require sharing customer data. Another enterprise may outsource its data processing, so data might be given to various other companies. The owner of the data is called as distributors and the trusted third parties are called as agents. Data leakage happens every day when confidential business information such as customer or patient data, company secrets, budget information etc. is leaked out. When this information is leaked out, then the companies are at serious risk. Most probably data are being leaked from agent’s side. So, company has to very careful while distributing such a data to agents. The Goal of Our project is to analyse “how the distributer can allocate the confidential data to the Agents so that the leakage of data would be minimized to a Greater Extent by finding a guilty agent”.

**Amol O. Gharpande et al** gives review idea about data leakage detection techniques. A data distributor has given sensitive data to a set of supposedly trusted agents (third parties). Some of the data is leaked and found in an unauthorized place (e.g., on the web or somebody’s laptop). The distributor must assess the likelihood that the leaked data came from one or more agents, as opposed to having been independently gathered by other means. We propose data allocation strategies (across the agents) that improve the probability of identifying leakages. These methods do not rely on alterations of the released data (e.g., watermarks). In some case, we can also inject “realistic but fake” data records to further improve our chances of detecting leakage and identifying the guilty party.

**Garcia-Molina [2010]**

Our scenario is based on the scenario presented by Papadimitriou and Garcia-Molina [2010], with several modifications. In their paper, Papadimitriou and Garcia presented a method for data leakage detection. In the scenario that they address, a distributor distributes sensitive data to almost all agents according to a specific request that is issued for each one of the agents. An example of such a scenario is a proactive CRM system in which the data owner decides which customer or stakeholder to call, and the customer or stakeholder details are forwarded to the third party call agent. If sensitive data is leaked, the data owner would like to be able to identify the source of leakage, or at least be able to estimate the likelihood of each agent to have been involved in the incident. Therefore, A guilt model is proposed for estimating the probability that an agent is involved in a given data leakage. The capability and ability to identify the of the leakage depends on the distribution of data objects among the agents. Therefore, a data allocation method that distributes data records among the agents based on the agents’ requests and optimization models are presented. The proposed allocation method ensures that object sharing among the agents is minimal, and therefore, in the case of a leakage incident, the data owner will be able to use the guilt model to identify the source of leakage with high probability.

**Garcia-Molina [2010]**

Two types of data requests are being considered in Papadimitriou and Garcia-Molina [2010]: explicit request and sample request. An explicit request contains predefined conditions, and all the objects in the dataset that comply with these conditions must be returned. A sample request defines the amount of objects to be randomly selected from the entire dataset. Combined requests (i.e., requests for a sample of objects that comply with a predefined condition) are not handled by the proposed algorithms; However, it is explained how they might be handled using the proposed algorithms. They as well proposed including fake data to the lists of real data objects when distributing them to the agents. Fake data objects may help to better distinguish between the agents and increase the accuracy of the guilt model (e.g., when each untrusted agent receives a unique fake object). Four scenarios can be stated by the two request types (sample or explicit) and the two options of planting fake objects in the result sets (using or not using fake objects). It is assumed that in each scenario, all of the agents issue the same type of requests (i.e., either explicit or sample queries), and if fake objects are up to use, the same amount of objects will be planted for all agents. Several allocation algorithms are proposed to deal with each scenario. Empirical evaluation showed that the proposed algorithms reached a significantly greater ability to identify the source of leakage (compared with simple allocation algorithms), even in cases where there was a large overlap between the objects that the agents received.

**CHAPTER-3**

**DYSTEM**

**ANALYSIS**

**3.1 FEASIBILITY STUDY**

Feasibility study is made to see if the project on completion will serve the purpose of the organization for the amount of work, effort and the time that spend on it. Feasibility study lets the developer foresee the future of the project and the usefulness. A feasibility study of a system proposal is according to its workability, which is the impact on the organization, ability to meet their user needs and effective use of resources. Thus, when a new application is proposed it normally goes through a feasibility study before it is approved for development.

The document provides the feasibility of the project that is being designed and lists various areas that were considered very carefully during the feasibility study of this project such as Technical, Economic and Operational feasibilities.

The following are its features:

**TECHNICAL FEASIBILITY**

The system must be evaluated from the technical point of view first. The assessment of this feasibility must be based on an outline design of the system requirement in the terms of input, output, programs and procedures. Having identified an outline system, the investigation must go on to suggest the type of equipment, required method developing the system, of running the system once it has been designed.

Technical issues raised during the investigation are:

Does the existing technology is sufficient for the suggested one?

Can the system expand if developed?

The project should be developed such that the necessary functions and performance are achieved within the constraints. The project is developed within latest technology. Through the technology may become obsolete after some period of time, due to the fact that never version of same software supports older versions, the system may still be used. So, there are minimal constraints involved with this project. The system has been developed using Java the project is technically feasible for development.

**BEHAVIORAL FEASIBILITY**

This includes the following questions:

* Is there sufficient support for the users?
* Will the proposed system cause harm?

The project would be beneficial because it satisfies the objectives when developed and installed. All behavioral aspects are considered carefully and conclude that the project is behaviorally feasible

**ECONOMIC FEASIBILITY**

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on project, which will give best, return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require.

The following are some of the important financial questions asked during preliminary investigation:

* The costs conduct a full system investigation.
* The cost of the hardware and software.
* The benefits in the form of reduced costs or fewer costly errors.

Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also, all the resources are already available, it gives an indication of the system is economically possible for development.

**3.2 GANTT CHART**

Gantt charts make it easy to visualize project management timelines by transforming task names, start dates, durations and end dates into cascading horizontal bar charts.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **2018-2019** | | | |
| **Dec** | **Jan** | **Feb** | **Mar** |
| **Analysis** |  |  |  |  |
| **Design** |  |  |  |  |
| **Coding** |  |  |  |  |
| **Testing** |  |  |  |  |
| **Documentation** |  |  |  |  |

**CHAPTER-4**

**SYSTEM DESIGN**

**AND**

**DIAGRAM**

**4.1 DATA FLOW DIAGRAM**

A data flow diagram (DFD) is a graphical representation of the flow of data through an information system. A data flow diagram can also be used for the visualization of data processing (structured design). It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. This context-level DFD is then exploded to show more detail of the system being modeled.

Symbols:

The four components of a data flow diagram (DFD) is:

* External Entities/Terminators are outside of the system being modeled. Terminators represent where information comes from and where it goes. In designing a system, we have no idea about what these terminators do or how they do it.
* Processes modify the inputs in the process of generating the outputs.
* Data Stores represent a place in the process where data comes to rest. A DFD does not say anything about the relative timing of the processes, so a data store might be a place to accumulate data over a year for the annual accounting process.

**0-LEVEL DFD**

Request

Request

Response

User/Client

Distributor

Response

**Database**

**1-Level DFD**

Registration details

**Client**

File details

Login details

Encryption keys

**2-Level DFD**

Registration details

**Distributor**

File details

Login details

Object details

Object details

Display result to distributor

**Distributor Sequence Diagram**

Distributor

Authentication

Files

Extract Object

Guilty Agent

Database

Authenticate

Enter Username & password

View leak files

Extract fake object

Find file leaker using fake object hash value

Get Agent information

Display result

Encrypt & Upload file

Uploaded successfully

View request & Accept request

Send file with fake object & save hash value

**Agent Sequence Diagram**

Database

Agent

Enter Username & password

Login successfully

View file

Display file

Send successfully

Send request for file

Download successfully

Get Key, decrypt & Download file

Send successfully

Share with another person

**Class Diagram**

Distributor

Username (String)

Password (String)

Login ()

Registration ()

Upload file ()

Accept request ()

Agent

Username (String)

Password (String)

Email (String)

Login ()

Registration ()

View file ()

Send Request ()

Download file ()

Leak data ()

Update Profile ()

M

M

Login

Work Allocation

Add Fake Object ()

Leakage Analysis

Fake object

Agent-name

Find out leakage ()

Identify leakage ()

Report leakage ()

Leaker

Agent name (String)

Username (String)

Leak data ()

Fake object ();

Hash Value ();

**Use Case Diagram**

Distributor

Agent

Activity diagram for User:

Yes

No

Enter login details

Password Authentication

View files

Try again

Send request for file

Get key on email & enter

Decrypt & Download file

Share file with other

Yes

Leak file

No

Activity diagram for Owner:



Yes

No

Enter login details

Password Authentication

View leak file

Select file

View Request

Extract fake object & match hash value

Encrypt file using Blowfish

Accept Request

Find leaker agent name using fake object

Upload file

Send file with fake object & store hash value using SHA

View leaker agent data

**CHAPTER-5**

**CODING**

**AND**

**OUTPUT**

**CODE**

**Home page**

<!DOCTYPE HTML>

<html>

<head>

<title>Secure Cloud System</title>

<link href=*"css/bootstrap.css"* rel=*"stylesheet"* type=*"text/css"* media=*"all"* />

<link href=*"css/style.css"* rel=*"stylesheet"* type=*"text/css"* media=*"all"* />

<link href=*"css/font-awesome.css"* rel=*"stylesheet"* type=*"text/css"* media=*"all"* />

<link href=*"css/carousel.css"* rel=*"stylesheet"* type=*"text/css"* media=*"all"* />

<link href=*"css/owl.carousel.css"* rel=*"stylesheet"* type=*"text/css"* media=*"all"* />

<script src=*"js/jquery-1.10.2.min.js"* type=*"text/javascript"*></script>

<script src=*"js/bootstrap.js"* type=*"text/javascript"*></script>

<script src=*"js/bootstrap.min.js"* type=*"text/javascript"*></script>

<script src=*"js/owl.carousel.js"* type=*"text/javascript"*></script>

<script>

$(document).ready(**function**() {

**var** owl = $("#owl-demo");

owl.owlCarousel({

items :4, //10 items above 1000px browser width

itemsDesktop : [1000,4], //5 items between 1000px and 901px

itemsDesktopSmall : [900,3], // 3 items betweem 900px and 601px

itemsTablet: [600,2], //2 items between 600 and 0;

itemsMobile : **false** // itemsMobile disabled - inherit from itemsTablet option

});

// Custom Navigation Events

$(".next").click(**function**(){

owl.trigger('owl.next');

})

$(".prev").click(**function**(){

owl.trigger('owl.prev');

})

});

</script>

<script type=*"text/javascript"*>

// Login Form

$(**function**() {

**var** button = $('#loginButton');

**var** box = $('#loginBox');

**var** form = $('#loginForm');

button.removeAttr('href');

button.mouseup(**function**(login) {

box.toggle();

button.toggleClass('active');

});

form.mouseup(**function**() {

**return** **false**;

});

$(**this**).mouseup(**function**(login) {

**if**(!($(login.target).parent('#loginButton').length > 0)) {

button.removeClass('active');

box.hide();

}

});

});

</script>

</head>

<body>

<!-- Start Header -->

<div class=*"header"*>

<div class=*"header-top"*>

<div class=*"wrap"*>

<div class=*"header-top-left"*>

<p>Support: +0123456789</p>

</div>

<div class=*"header-top-right"*>

<ul>

<li><a href"#"><i class=*"fa fa-comments"*></i>Sales Chat</a></li>

<li class=*"login"*>

<div id=*"loginContainer"*>

<a href=*"#"* id=*"loginButton"*><span><i class=*"fa fa-lock"*></i>Login</span></a>

<div id=*"loginBox"* class=*"login-form"*>

<h3>Login into Your Account</h3>

<form id=*"loginForm"* action=*"login"* method=*"post"*>

<span>

<i><img src=*"images/user.png"* alt=*""* /></i>

<input type=*"text"* placeholder=*"Your username"* name=*"username"* required>

</span>

<span>

<i><img src=*"images/lock.png"* alt=*""* /></i>

<input type=*"password"* placeholder=*"........."* name=*"password"* required>

</span>

<input type=*"submit"* value=*"Login"*>

</form>

</div>

</div>

</li>

<li><a href=*"signup.jsp"* ><i class=*"fa fa-sign-in"*></i>Sign Up</a></li>

</ul>

</div>

<div class=*"clear"*></div>

</div>

</div>

**Checkleaker**

package leak;

import java.io.File;

import java.io.IOException;

import java.io.PrintWriter;

import java.nio.ByteBuffer;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.nio.file.attribute.UserDefinedFileAttributeView;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import org.apache.commons.fileupload.FileItem;

import org.apache.commons.fileupload.disk.DiskFileItemFactory;

import org.apache.commons.fileupload.servlet.ServletFileUpload;

import org.apache.commons.io.FilenameUtils;

import uploader.extnslector;

import uploader.path;

import fakeobject.Signature;

import databaseconnection.DatabaseConnection;

import dbmanager.datamanager;

import endecryption.file\_encryption;

import endecryption.getName;

import endecryption.key;

/\*\*

\* Servlet implementation class checkleak

\*/

@WebServlet("/checkleak")

public class checkleak extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public checkleak() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse

\* response)

\*/

protected void doGet(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse

\* response)

\*/

protected void doPost(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

HttpSession session = request.getSession();

PrintWriter out = response.getWriter();

DatabaseConnection db = new DatabaseConnection();

db.dbconnection();

// process only if its multipart content

if (ServletFileUpload.isMultipartContent(request)) {

try {

List<FileItem> multiparts = new ServletFileUpload(

new DiskFileItemFactory()).parseRequest(request);

for (FileItem item : multiparts) {

if (!item.isFormField()) {

String name = new File(item.getName()).getName();

System.out.println(name); // print with extenssion

path p = new path();

String username = (String) session.getAttribute("user");

File udir = new File(p.leakpath + username);

if (!(udir.exists())) {

udir.mkdir();

}

String filepath = udir + "//" + name;

File a = new File(filepath);

item.write(a);

Attribute atr = new Attribute();

String str = atr.getAttribute(filepath);

if (str.equals("attribute not found")) {

out.println("<script type='text/javascript'>");

out.println("alert('attribute not found')");

out.println("location='leakfile.jsp'");

out.println("</script>");

} else {

String email = "";

String parts[] = str.split(":::");

String owner = parts[0];

String downloader = parts[1];

if (owner.equals(username))

email = insertLog(a.getName(), owner, downloader, db);

downloader = parts[1]+" email:"+email;

out.println("<script type='text/javascript'>");

out.println("alert('guilty user is "+ downloader + "')");

out.println("location='leakfile.jsp'");

out.println("</script>");

} else {

out.println("<script type=\"text/javascript\">");

out.println("alert('this is not your file')");

out.println("location=\"leakfile.jsp\"");

out.println("</script>");

}

}

System.out.println(str);

}

}

} catch (Exception ex) {

System.out.println(ex);

}

} else {

request.setAttribute("message",

"Sorry this Servlet only handles file upload request");

}

}

public static String insertLog(String name,String username,String valueFromAttributes,DatabaseConnection db) {

String sql = "select emailid from user where username='"+valueFromAttributes+"'";

System.out.println(sql);

ResultSet rs = db.getResultSet(sql);

try {

if(rs.next()){

String query1 = "insert into leak values('" + name + "','" + username+ "','" + valueFromAttributes + "')";

db.getUpdate(query1);

return rs.getString("emailid");

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

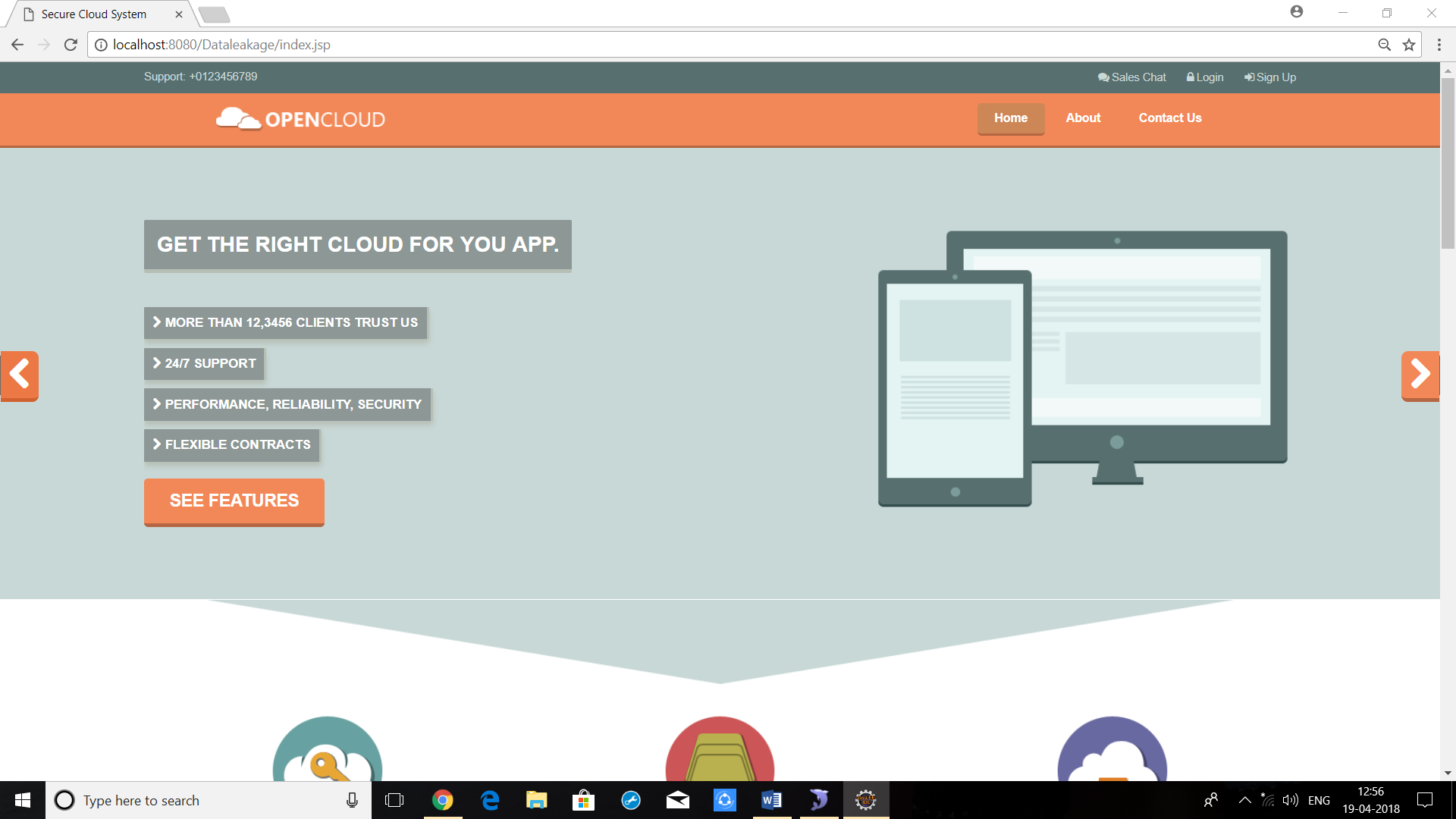
return "Email not found";

}

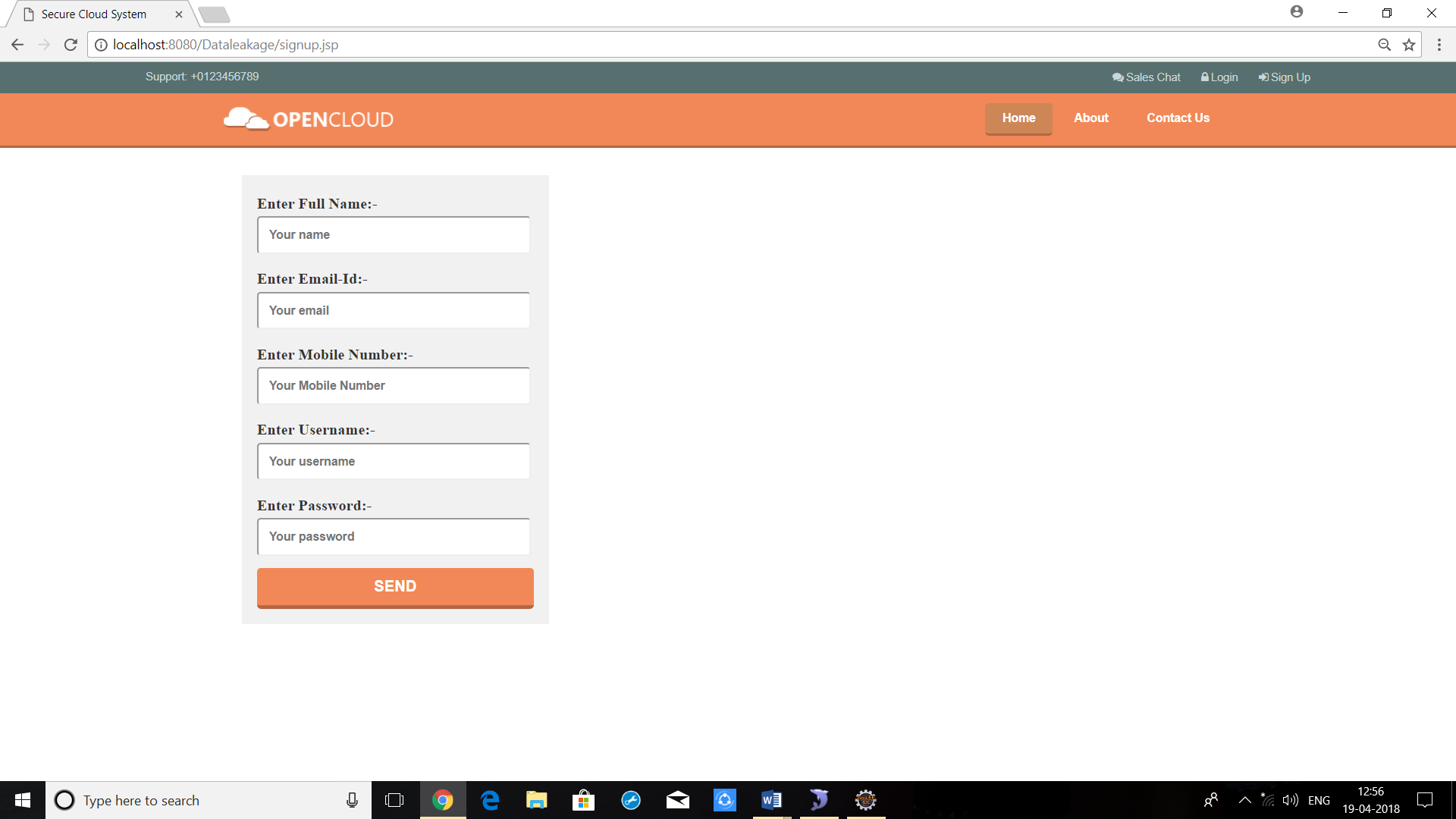
}

**Screen Shot**

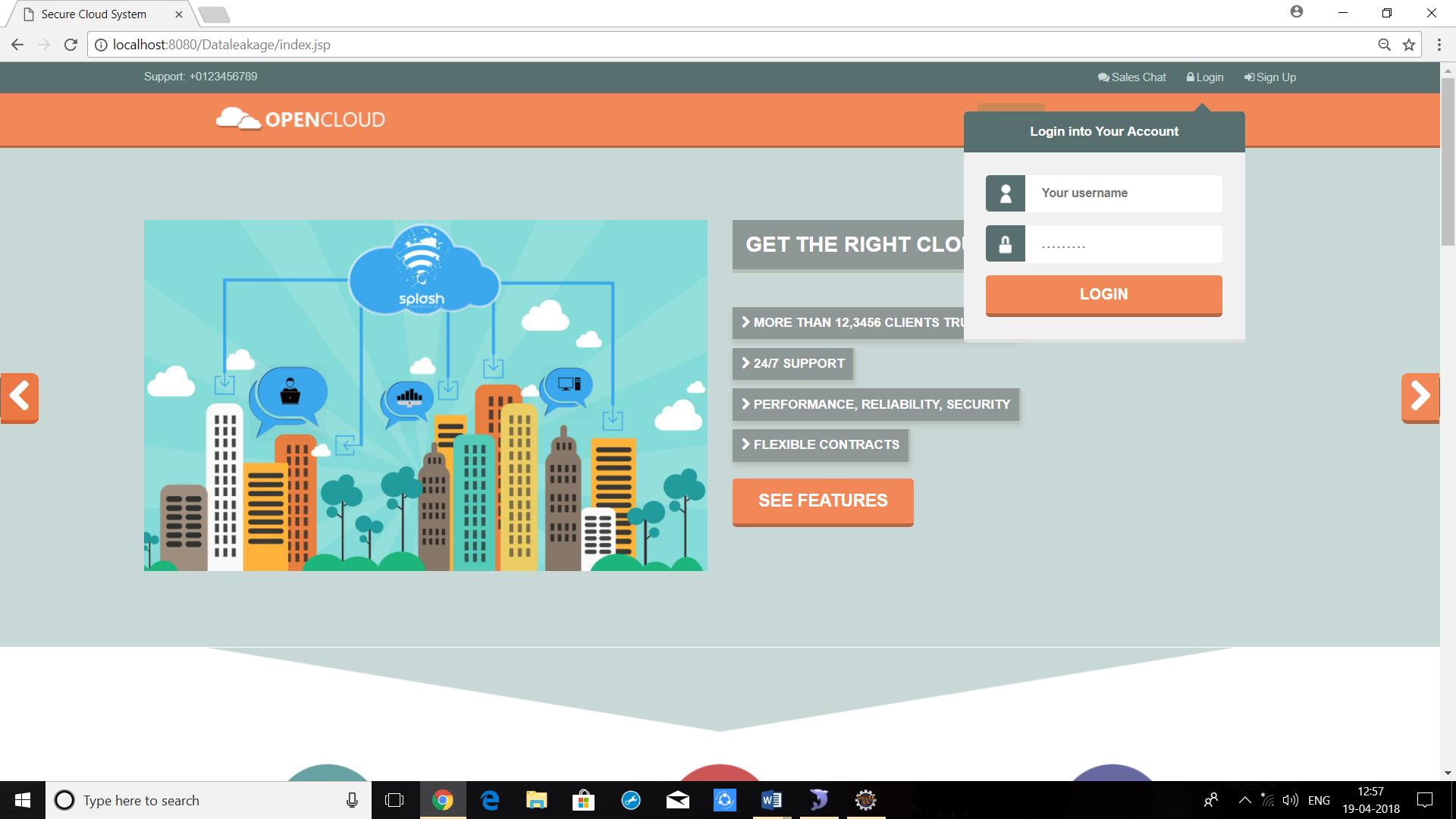
Homepage



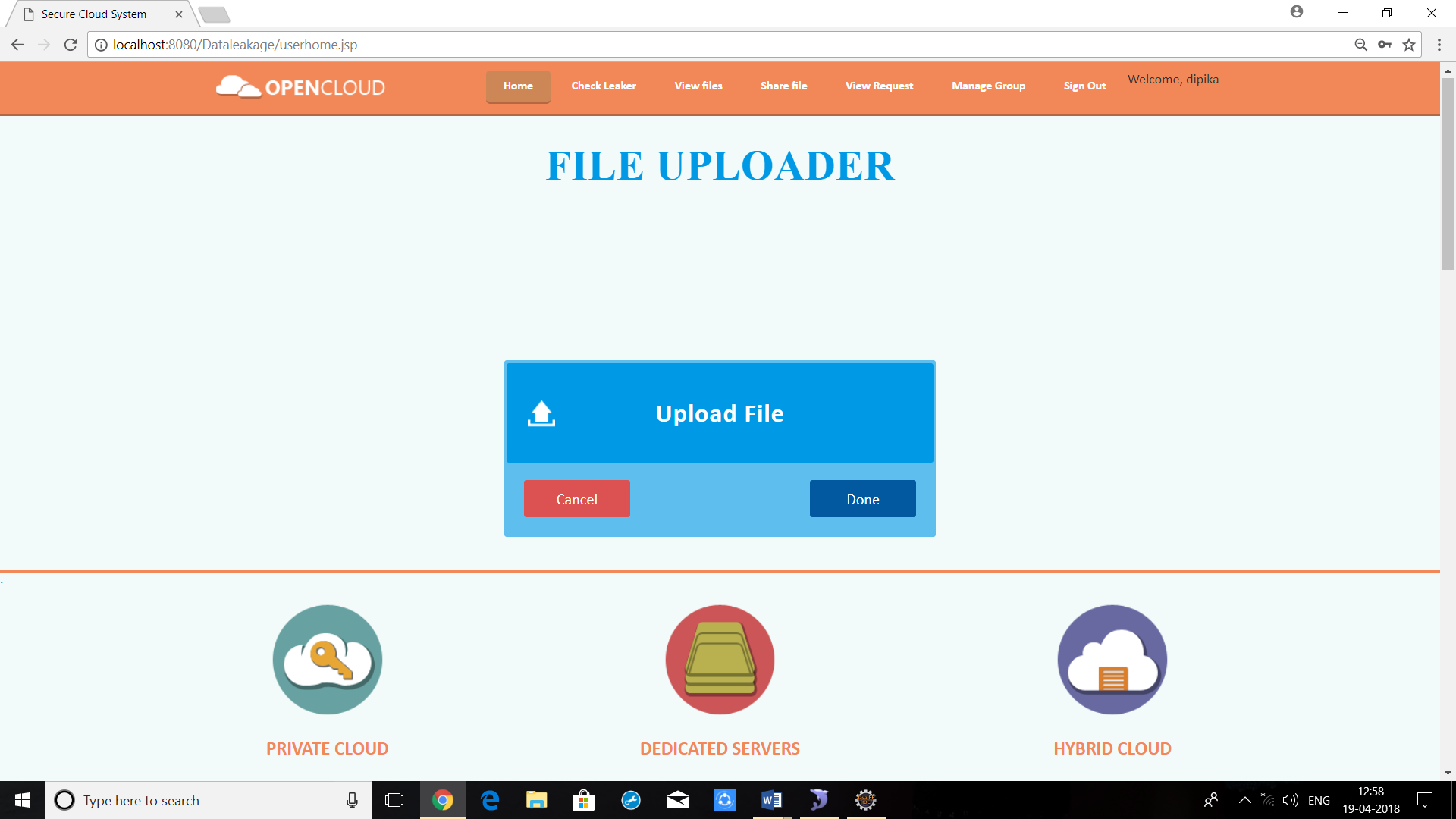
Sign up Page



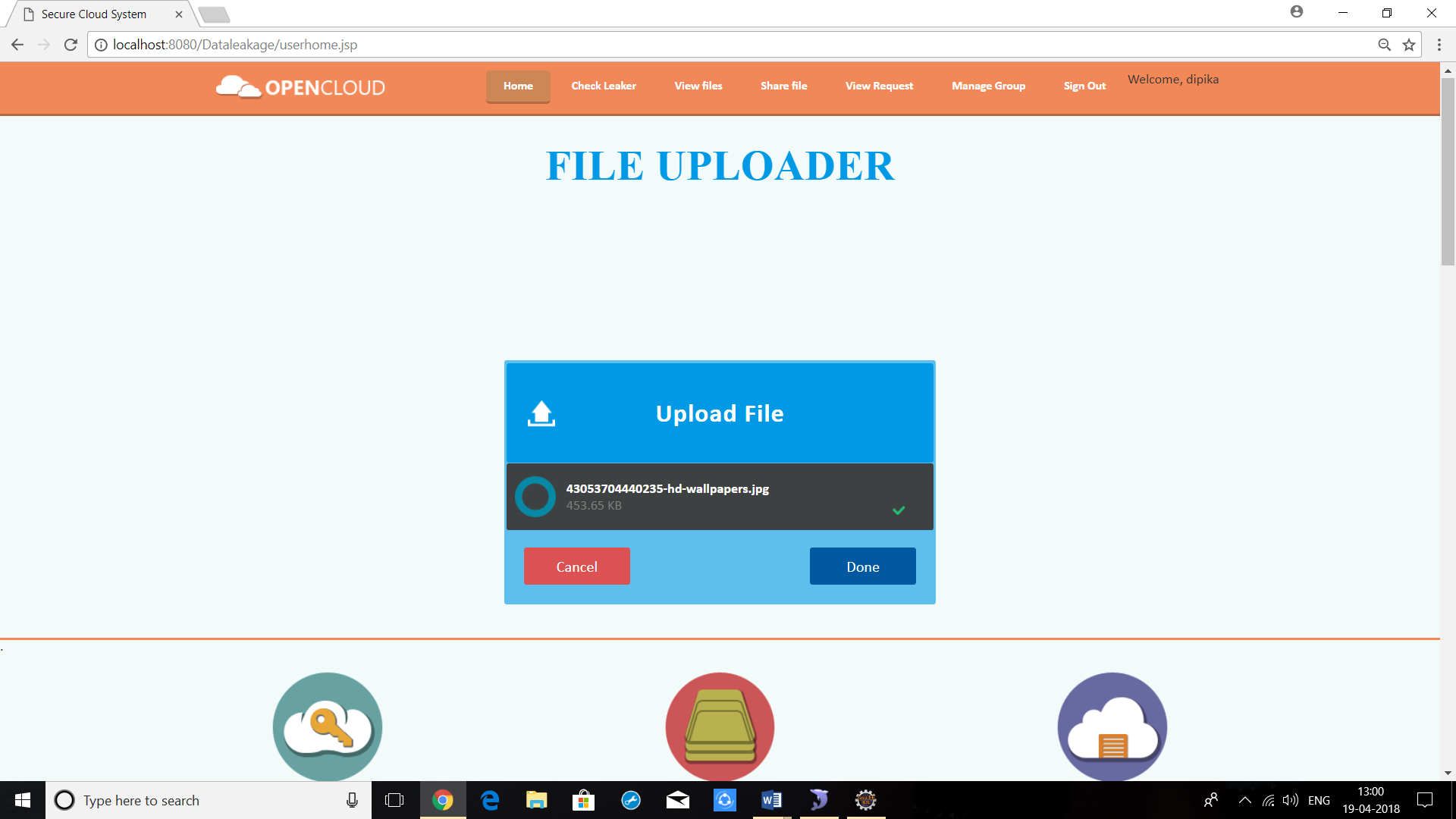
Login Page



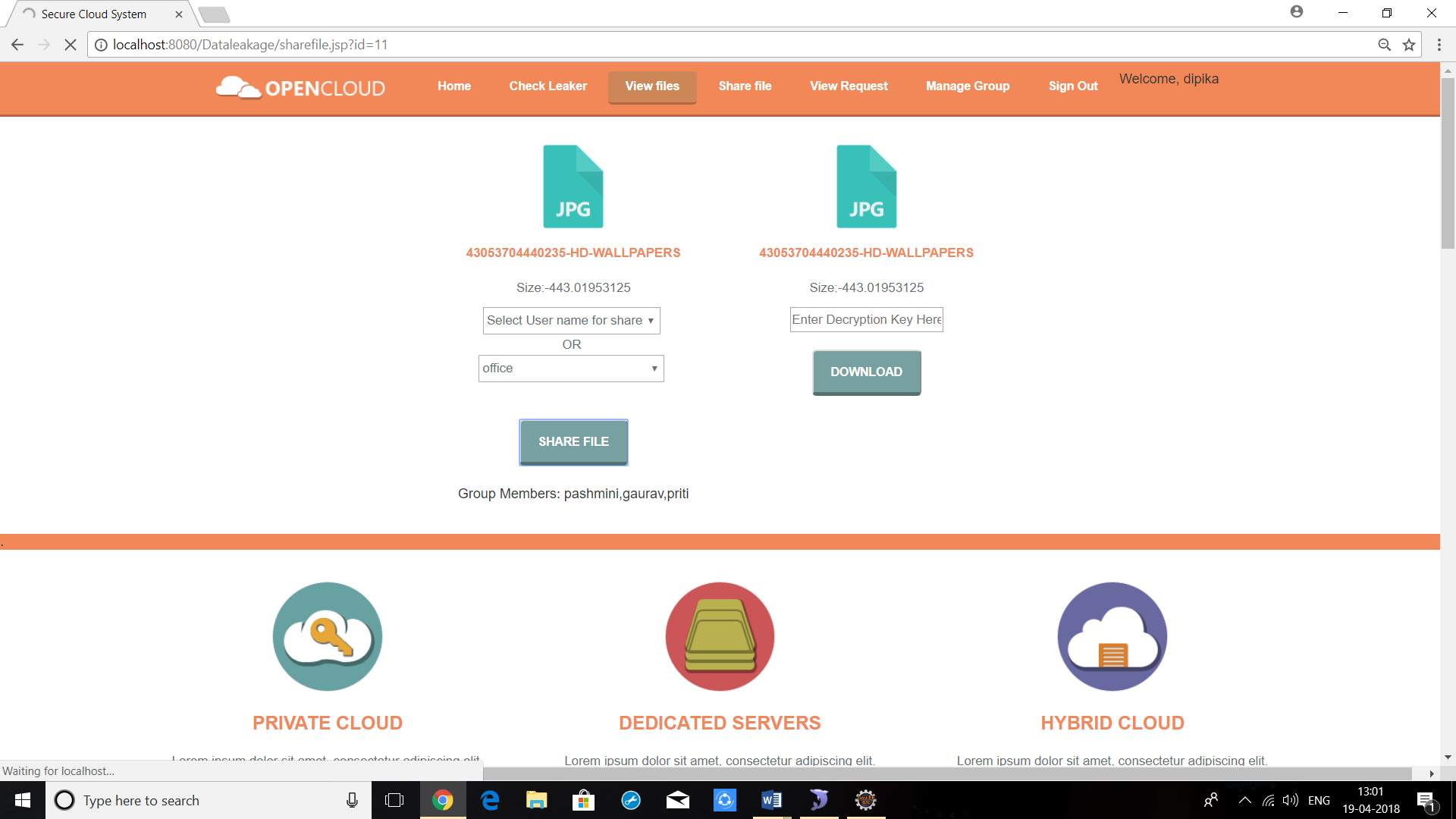
User Homepage



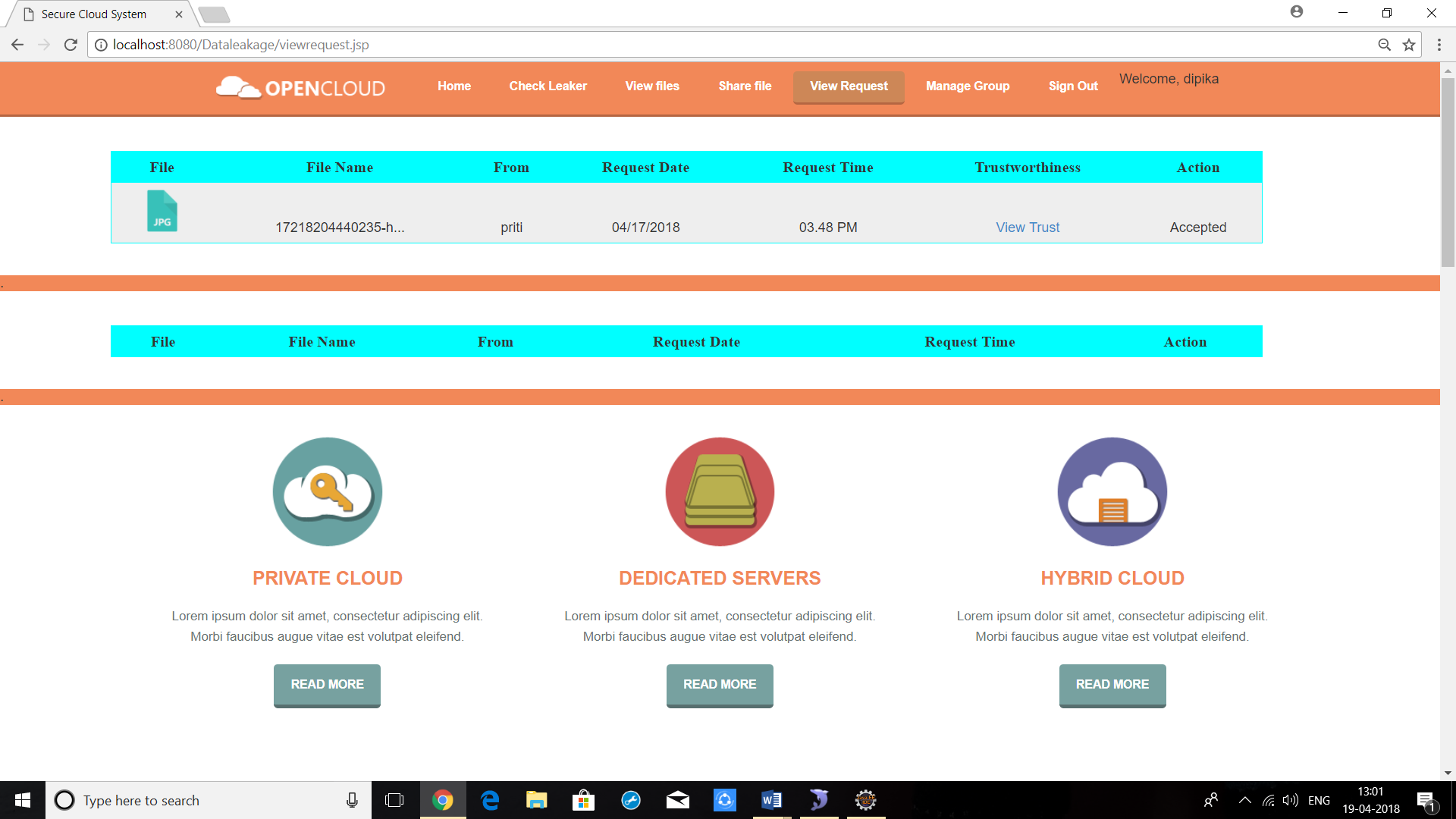
File Upload



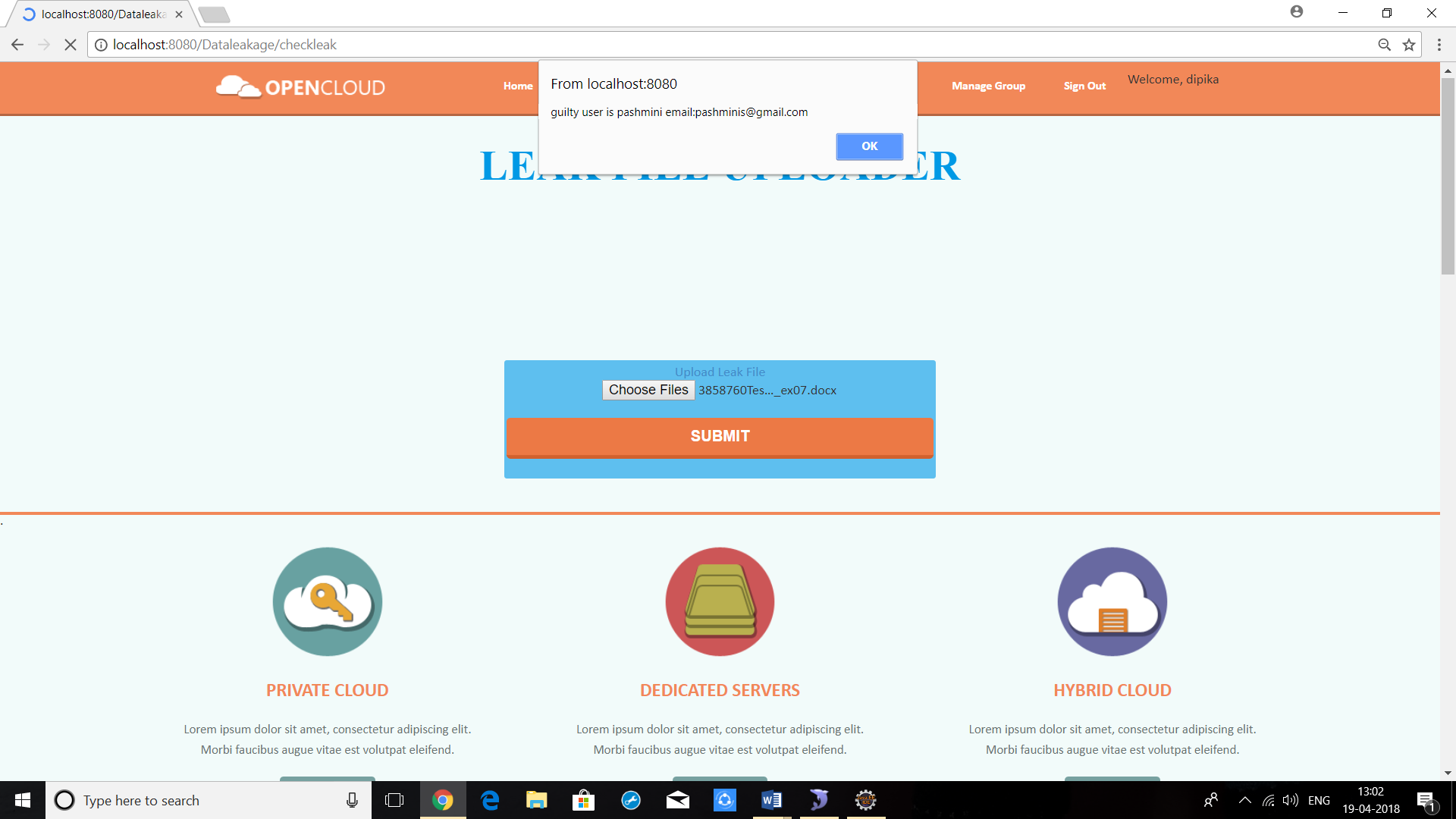
File Sharing



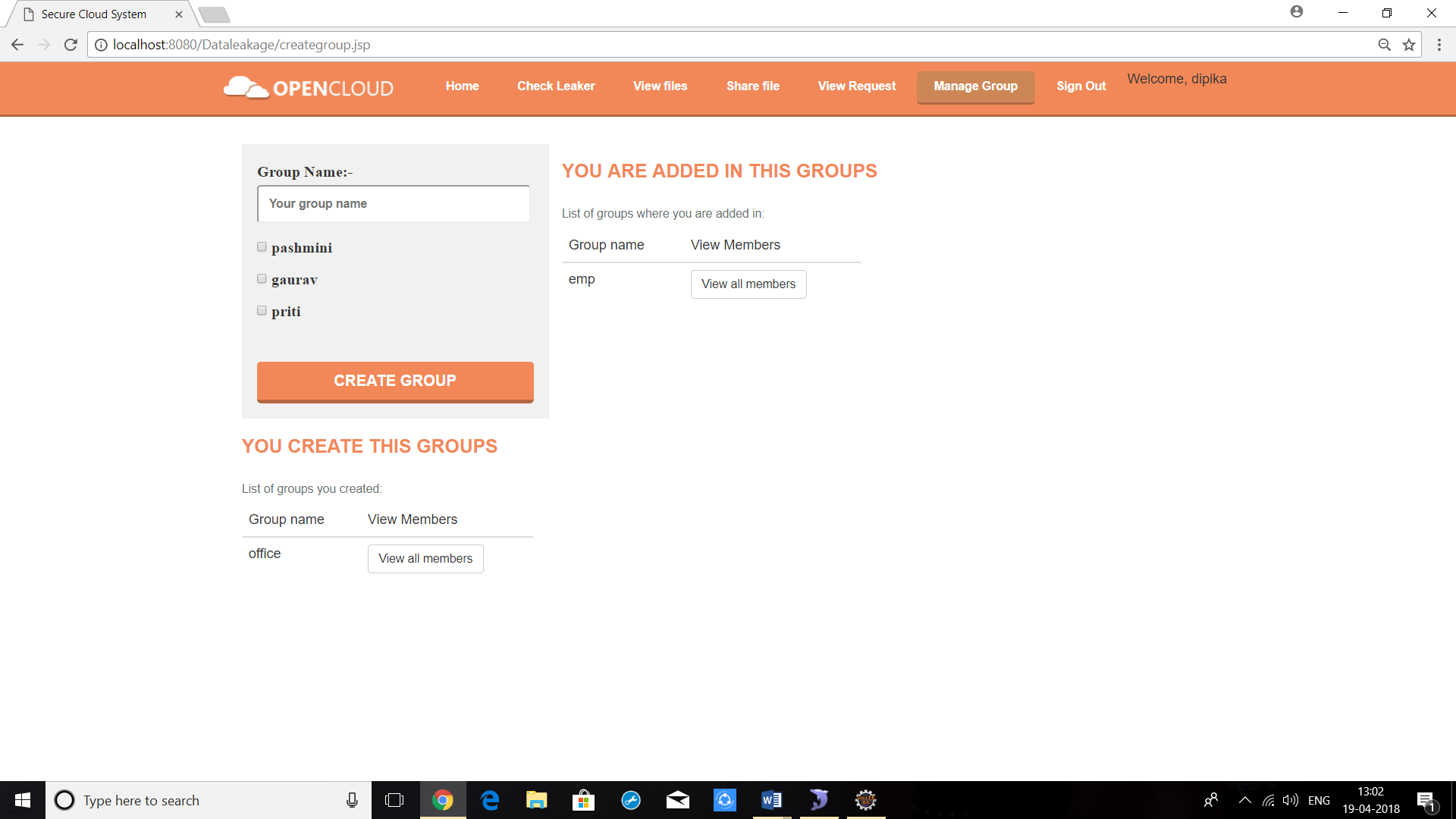
Request page



Checker Page



Manage Group Page



**CHAPTER 6**

**TESTING**

**AND**

**IMPLIMENTATION**

**SOFTWARE TESTING**

**Testing**

Software testing methods are traditionally divided into black box testing and white box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

**Black box testing**

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

**Specification-based testing:** Specification-based testing aims to test the functionality of software according to the applicable requirements. Thus, the tester inputs data into, and only sees the output from, the test object. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case.

Specification-based testing is necessary, but it is insufficient to guard against certain risks.

**Advantages and disadvantages:** The black box tester have no "bonds" with the code, and a tester's perception is very simple: a code must have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers do not. But, on the other hand, black box testing has been said to be "like a walk in a dark labyrinth without a flashlight," because the tester doesn't know how the software being tested was actually constructed. As a result, there are situations when (1) a tester writes many test cases to check something that could have been tested by only one test case, and/or (2) some parts of the back-end are not tested at all. Therefore, black box testing has the advantage of "an unaffiliated opinion," on the one hand, and the disadvantage of "blind exploring," on the other.

#### White box testing

#### White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

Types of white box testing

The following types of white box testing exist:

* API testing (application programming interface) - Testing of the application using Public and Private APIs.
* Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once).
* Fault injection methods - improving the coverage of a test by introducing faults to test code paths.
* Mutation testing methods.
* Static testing - White box testing includes all static testing.

**Code completeness evaluation**

White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested.

Two common forms of code coverage are:

* ***Function coverage***, which reports on functions executed.
* ***Statement coverage***, which reports on the number of lines executed to complete the test.

They both return code coverage metric, measured as a percentage.

**Integration Testing**

**Integration testing** is any type of software testing, that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localized more quickly and fixed.

### Acceptance testing

### Acceptance testing can mean one of two things

1. A smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e. before integration or regression.
2. Acceptance testing performed by the customer, often in their lab environment on their own HW, is known as user acceptance testing (UAT).

### Non-Functional Software Testing

Special methods exist to test non-functional aspects of software.

* Performance testing checks to see if the software can handle large quantities of data or users. This is generally referred to as software scalability. This activity of Non-Functional Software Testing is often referred to as Endurance Testing.
* Stability testing checks to see if the software can continuously function well in or above an acceptable period. This activity of Non-Functional Software Testing is oftentimes referred to as load (or endurance) testing.
* Usability testing is needed to check if the user interface is easy to use and understand.
* Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.
* Internationalization and localization is a needed to test these aspects of software, for which a pseudo localization method can be used.

In contrast to functional testing, which establishes the correct operation of the software (correct in that it matches the expected behavior defined in the design requirements), non-functional testing verifies that the software functions properly even when it receives invalid or unexpected inputs. Software fault injection, in the form of fuzzing, is an example of non-functional testing. Non-functional testing, especially for software, is designed to establish whether the device under test can tolerate invalid or unexpected inputs, thereby establishing the robustness of input validation routines as well as error-handling routines. Various commercial non-functional testing tools are linked from the Software fault injection page; there are also numerous open-source and free software tools available that perform non-functional testing.

**Destructive testing**

Destructive testing attempts to cause the software or a sub-system to fail, in order to test its robustness. We will be using the black box testing initially to test out the modules and once we are confident of each module working in a cohesive manner, it will be integrated, and integration testing will be performed.

**CHAPTER-7**

**MAINTANCE**

**AND EVALUATION**

**SYSTEM MAINTENANCE**

Maintenance Process finds out essential changes of the market or business or to correct some errors and tries to implement it in the existing system:

**CORRECTIONS:**

1. Minor changes in the processing logic of system
2. Improving Response time
3. Revisions on formats of data input
4. Improving User interface.

**ADAPTATIONS:**

1. The Proposed system should work on a any device with equal response time as for any greater version device

Achieving greater Scalability and Performance.

**FUTURE ENHANCEMENT**

Our future work includes the inquiring of agent guilt models that capture leakage scenarios that are not studied in this paper. For instance, what is the appropriate model for cases where agents can collude and identify fake tuples? Another open problem is the extension of our allocation strategies so that they can handle agent requests in an online fashion (thepresented strategies assume that there is a fixed set of agents with requests known in advance).

**CHAPTER-8**

**CONCLUSION**

**CONCLUSION**

In a perfect world there would be no need to handover sensitive data to agents that may unknowingly or maliciously leak it. And even if we had to hand over sensitive data, in a perfect world we could watermark each object so that we could trace its origins with absolute certainty. However, in many cases we must indeed work with agents that may not be 100% trusted, and we may not be certain if a leaked object came from an agent or from some other source, since certain data cannot admit watermarks. In spite of these difficulties, we have shown it is possible to assess the likelihood that an agent is responsible for a leak, based on the overlap of his data with the leaked data and the data of other agents, and based on the probability that objects can be “guessed” by other means. Our model is relatively simple, but we believe it captures the essential trade-offs. The algorithms we have presented implement a variety of data distribution strategies that can improve the distributor’s chances of identifying a leaker. We have shown that distributing objects judiciously can make a significant difference in identifying guilty agents, especially in cases where there is large overlap in the data that agents must receive. Our future work includes the investigation of agent guilt models that capture leakage scenarios that are not studied in this paper. For example, what is the appropriate model for cases where agents can collude and identify fake tuples? A preliminary discussion of such model is available in. Another open problem is the extension of our allocation strategies so that they can handle agent requests in an online fashion (the presented strategies assume that there is a fixed set of agents with requests known in advance). Another and most problem to be solved is protecting data before getting leaked.

**CONCLUSION**

The project titled as “E-GOVERNANCE” is a web site. The **E-GOVERNANCE** is a web based online application to automate the process for utilization of various schemes provided by government for various panchayats.

Website provides facility for reporting online crimes, complaints, missing persons, show criminal details. This website is developed with scalability in mind. Additional modules can be easily added when necessary. The website is developed with modular approach. All modules in the system have been tested with valid data and invalid data and everything work successfully. Thus, the system has fulfilled all the objectives identified and is able to replace the existing system. The project has been completed successfully with the maximum satisfaction of the organization. The constraints are met and overcome successfully. The system is designed as like it was decided in the design phase. The project gives good idea on developing a full-fledged application satisfying the user requirements. The system is very flexible and versatile.

This system clearly keeps track of the status of user’s application by tracking the actual activities that are under process within the panchayat or at higher level at any point of time, by the administrators. This system provides effective way to manage the important information in a very secure manner by authenticating users at various levels.

This website is a user-friendly screen that enables the user to use without any inconvenience. Validation checks induced have greatly reduced errors. The application has been tested with live data and has provided a successful result.  Hence the website has proved to work efficiently.

**CHAPTER-9**

**REFERENCES**

**AND**

**BIBLOGRAPHY**

**REFERENCES AND BIBLIOGRAPHY**

[1] Wu, Jiangjiang, “An Active Data Leakage Prevention Model for Insider Threat Intelligence Information Processing and Trusted Computing” (IPTC), 2011 2nd international Symposium on, 22-23 Oct. 2011.

[2] Papadimitriou, Panagiotis, “A Model for Data Leakage Detection”, Data Engineering, 2009. ICDE '09. IEEE 25th International Conference, March 29

2009-April 2 2009.

[3] Xiaosong Zhang , “Research and Application of the Transparent Data Encryption in Intranet Data Leakage Prevention,” Computational Intelligence and

Security, 2009. CIS '09. International Conference.11-14 Dec. 2009.

[4] Guo, Hongyu, “Identifying and Preventing Data Leakage in Multi-relational Classification,” Data Mining Workshops (ICDMW), 2010 IEEE International Conference, 13 Dec,2010.

[5] Bhaduri, Kanishka , “Privacy-Preserving Outlier Detection Through Random Nonlinear Data Distortion,” Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions, Volume: 41, Issue: 1 Feb. 2011.

[6] Li, Xiao-Bai , “A Tree-Based Data Perturbation Approach for Privacy-Preserving Data Mining”, Knowledge and Data Engineering, IEEE Transactions, Volume: 18, Issue: 9, 2006.

[7] Lizhong Zhang, Wei Gao; Nan Jiang; “Relational databases watermarking for textual and numerical data”, Mechatronic Science, Electric Engineering and Computer (MnEC), 2011 International Conference, 9-22 Aug. 2011.