Data Leakage Detection and Security

Mrugesh U. Verekar. Abhijeet Patil. Gaurang Prabhu.
Vidyalankar Institute of Technology, University of Mumbai.

Abstract

In any enterprise, database plays one of the most crucial part since organizations most confidential data (e.g. employee's social security no) is stored in the database. Since the data is of great value, it should not be leaked or sabotaged. In every business field including private, public and individual level, database is widely used. In any organization there is a need to share the data among multiple trusted parties. But during this sharing of the data, it could be possible that any dishonest employees (aka guilty agents) may try to leak the data which results into data vulnerability or alteration. In order to prevent such data leakage, data leakage detection system has been proposed. It comprises of brief idea about data leakage and a methodology to detect the same.

Data leakage is a main obstacle to data distribution. A distributor gives sensitive data to a set of supposedly trusted agents. Sometimes the data gets leaked and is found in an unauthorized place. If the data is found at some places other than the authorized places which imply that the any of the trusted agents has leaked the data, so the distributor needs to identify the guilty agents. we analyse the guilty model that detects the agents by using concept called 'data allocation strategy' which intelligently allocates a set of fake objects (not real object but appears to be realistic to agent) to the trusted agents without making any modification to the original data. The guilty agent is one who leaked a portion of distributed data. In addition to this, we also aim to impart security at various levels using encryption depending upon the hierarchy of the importance of the data.

Main idea is to distribute the data intelligently to agents based on sample and explicit data request in order to improve the chances of detecting the guilty agents. The algorithm implemented using fake object will help to improve the chances of detecting the guilty agents.

Index Terms—fake object, watermark, data leakage detection, data allocation strategy

1. Introduction.

While doing any kind of business, sensitive data is required to be shared with authorized users, employees as well as trusted third parties. The owner of the data is said to be a distributor and the supposedly trusted third parties are said to be agents. Here our aim is to detect whether the distributor's sensitive data has been leaked by agents and if yes, then try to find out the agent who has leaked the data.

Traditionally, data leakage detection is handled by watermarking by embedding a unique code in each distributed copy. If that copy is later discovered in the hands of an unauthorized party, the leaker can be identified. Watermarks can be very useful in some cases. E.g. we may have seen that many of the company's/organizations do make available some documents for the entire world, so they upload it on their respective websites by embedding their watermarks in it. These watermarks do not modify any documents. But these watermarks can be destroyed by using any latest technology if the data recipient is malicious. So our purpose of identifying a guilty agent is not served fully. In order to overcome this problem, in this paper we study a perturbation technique for detecting a leakage of a set of objects or records.

Perturbation is a very useful technique but the problem associated with it is that the data gets modified and becomes less sensitive before it is handed over to the agents. But in some cases, it is important not to alter the distributor's original data. For example, any research stuff done by research organization, patient's medical record. Hence our focus is on use of fake objects. These are not real objects but appears realistic to the user/trusted client i.e., these fake objects are still hidden from the clients and it gives an impression to the clients as if original data is not modified. All details regarding fake object will be stored in database and by making use of it, data distributor can assess the guiltiness by using the concept of probability.[1]

2. Existing System.

Traditionally, leakage detection is handled by watermarking, e.g., a unique code is embedded in each distributed copy. If that copy is later discovered in the hands of an unauthorized party, the leaker can be identified. Watermarks can be very useful in some cases, but again, involve some modification of the original data. Furthermore, watermarks can sometimes be destroyed if the data recipient is malicious.[1]

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. The distributor gives the data to the agents. These data will be watermarked. Watermarking is the process of embedding the name or information regarding the company. The examples include the pictures we have seen in the internet. The authors of the pictures are watermarked within it. If anyone tries to copy the picture or data the watermark will be present. And thus the data may be unusable by the leakers.

Disadvantage

This data is vulnerable to attacks. There are several techniques by which the watermark can be removed. Thus the data will be vulnerable to attacks.

3. Proposed System.

Our aim 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. Using Perturbation technique we can modify the data to make it less sensitive before being handed to agents. We have developed an unobtrusive techniques for detecting leakage of a set of objects or records. In this section we develop a model for assessing the guilt of agents who has the highest probability. We also present algorithms for distributing objects to agents, in a way that improves our chances of identifying a leaker.

We also consider the option of adding fake objects to the distributed set of data. This fake objects do not correspond to real entities but appear realistic to the agents. In a sense, the fake objects acts 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. The Fake object are unique for every set of data and the value of the fake object changes every time the data is accessed and modified by the receiving agent depending on the specific pre specified instructions which can be vary upon type and importance of data and on receiving agent.

Today's technology made the watermarking system a simple technique of data authorization and a simple means of identification of source. There are various software's which can remove the watermark from the data and makes the data as original leaving no way around for its original owner to any legal claim on data.

Also data is encrypted before sharing (transferring) and the encryption is done using a unique key for each client agent. All the information (sensitive data) is stored in a different database server in altogether different Encrypted form to which only main server has access.

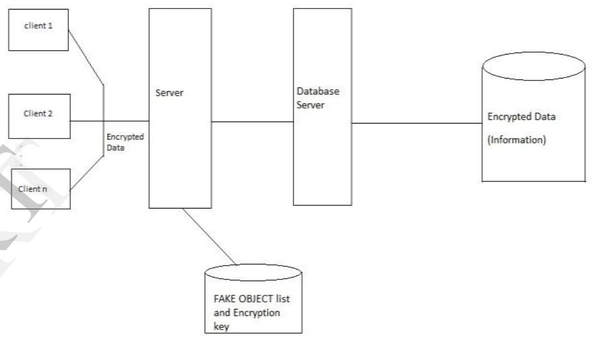


Figure 1: Encrypted Data

Advantage

The system that we have developed includes the facility of data hiding along with the provisional software only by which the data can be accessed. This system gives privileged access to the database administrator (i.e., data distributor) as well as to the agents that are registered by the distributors. Only registered agents can access the system. The user accounts can be activated as well as cancelled. The exported file will be accessed only by the system. The agent has given only the permission to access the software and view the data. The data can be copied by our software. If the data is copied to the agent's system, the path and agent's information will be sent to the distributors (i.e., to the server) thereby the identity of the data leaker can be traced.

4. Implementation.

Our aim is develop an application for an organization or businesses where the organization (Distributor agent) can share information with its clients (receiving agents) we have implemented a three tier architecture in implementation and follows a client-server architecture.

The Distributer agent at server can share some important data object with receiving agents i.e., trusted clients. The server is responsible for adding watermarks, implement Perturbation, add unique fake objects, watermarks if required and encrypt the data before sending and the third tier is a separate server where we can store the data in encrypted form.

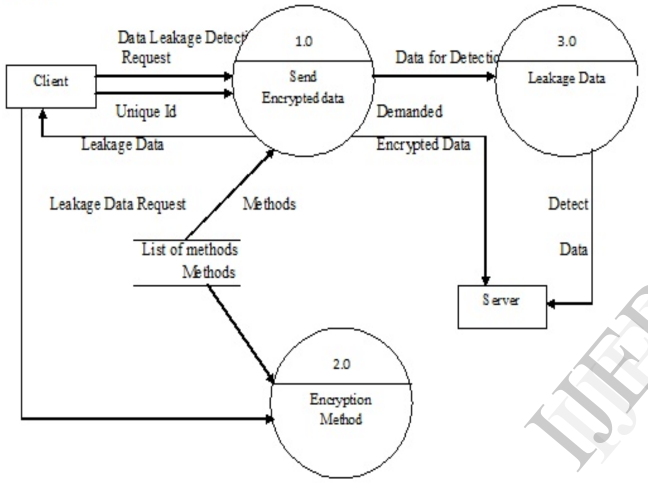


Figure 2: System Architecture

A client agent can access the data information shared and can perform all the other granted operations only through the application. Hence, the client requires the application to decrypt the encrypted shared data and to perform other tasks. A client specific log of every client is kept in server of all operations performed by the client. Every time when client access any information or do any operation with the data or on the application a notification is sent to the server and the client log is updated.

A client agent if having access writes can modify or add new information to the existing shared information and the procedure of client modifying information is carried out. E.g. In a hospital, a doctor can add or modify patient's medical record depending on the current diagnosis and can forward it to another doctor for review which can be modified by him if required.

The data is always distributed in an encrypted form. There will be a client specific encryption with a unique key assigned and possessed by the client. The data is securely stored in a different server using altogether different encryption technique.

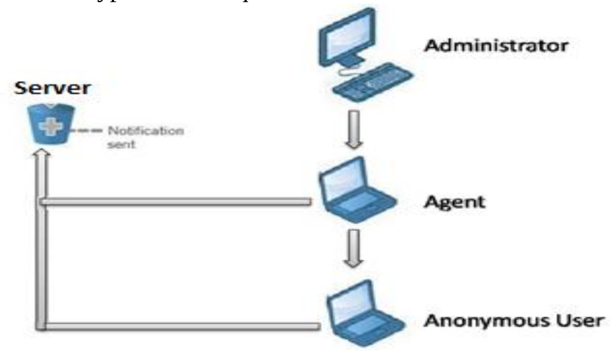


Figure 3: Notification sent to server every time when the data is transferred from one source to another

Major modules of the project are:

Data Allocation Module:

Any client can make a request to the data distributor.

Once the request is received, data distributor first verifies whether the request that is received is from authorized agent/client or not. Once verification is done, data distributor further verifies whether that particular agent/client has that permission to send a request or not. Once verification is done, the data distributor then intelligently gives data to agents by adding fake objects in it in order to improve the chances of detecting a guilty agent.

In our project, we have used an algorithm named as 'explicit data request'. It works as follows.

Let T = {t1, t2, …, tn} be the set of records that are owned by data distributor. First agents send the request for available records to data set T. It can contain sensitive as well as non-sensitive data. It can be represented as Ri=EXPLICIT ({t1, t2, …, tn}, cond1). Such request is said to be an explicit data request which is made to the data distributor. Once such request is received, data distributor add one or more fake object from the set of fake object and then hand over agents requested data. [1]

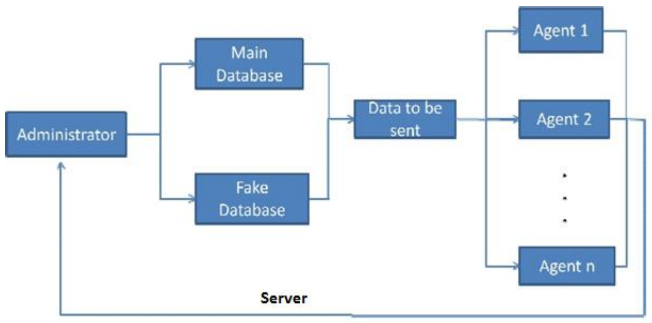


Figure 4: Data sharing mechanism

```
Algorithm: Evaluation of Explicit Data Request
1: Calculate total fake records = sum of fake object allowed.
2: While (total fake objects > 0) do
3: Select agent that have the greatest improvement in the sum objective i.e. i= argmax ((1\|Ri|)-(1\|Ri|+1)) σj Ri∩ Rj
4: Create fake object
5: Add this fake object to the agent's data set and also to the fake object set.
6: Decrement fake object from total fake record set.
```

E.g. Let T = {t_1, t_2} where T be the set that is owned by data distributor and there are two agents u1 and u2 with explicit data requests R_1 = {t_1, t_2} and R_2 = {t_1} respectively. The value of the sum-objective is in this case:

$$\sum_{i=1}^{2} 1/|Ri| \sum_{\substack{j=1 \\ j\neq i}}^{n} | R1 \cap R2|=1/2 +1/1= 1.5$$

Here we can see that object t1 is given to both the agents. This is called as overlap and it is represented as R1 ∩ R2.

But the problem associated with it is that the distributor cannot remove or alter the R_1 or R_2 data in order to decrease the overlap. Hence the distributor can create one fake object let say f and gives it to R_1. Now agent U_1 has now R_1 = {t_1, t_2, f} with F_1 = {f}. Due to this value of the sum-objective decreases to $1/3 + 1/1 = 1.33 < 1.5$. But if the data distributor adds fake object f to R_2 instead of R_1 then in this case the sum-objective would be $1/2 + 1/2 = 1 < 1.33$. This shows that addition of a fake object to R_2 has greater impact on the corresponding summation terms, since

$$1/|R_1|-1/| (R_1| + 1) = 1/6 < 1/|R_2|-1/ (|R_2|+ 1) = 1/2$$

Fake Object Module:

The concept of adding fake object is at the core of our project since it plays a lead role while assessing the guilt of the agent. In our project, perturbation technique is used which makes use of fake objects that are added by data distributor. This technique basically removes some of the data from original data/document and makes it less sensitive i.e., results into the modification of the original data only theoretically but not practically. This means that any client/trusted agent when desire to view their documents can get the documents as it is without any modification but practically data distributor intelligently add fake objects at the server side and modified data is send to the client. Detail information about for which clients which are the fake objects are added is stored in database at the server side. These fake objects are not visible in normal notepad or in any word-processor. These fake objects get visible only when those files are opened at the server side. Only data distributor has access to the server side. This means that there is no way by which client can come to know about which are the fake objects that are added if that client decided to leak that data. This means that fake objects remain hidden from the client.[1]

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 Buses the purchased mailing list, A receives copies of the mailing. These records area type of fake objects that help identify improper use of data.

The creation of a fake object for agent Ui as a black box function CREATEFAKEOBJECT (Ri, Fi, condi) that takes as input the set of all objects Ri, the subset of fake objects Fi that Ui has received so far, and condi, and returns a new fake object. This function needs condi to produce a valid object that satisfies Ui's condition. Set Ri is needed as input so that the created fake object is not only valid but also indistinguishable from other real objects.

The function can either produce a fake object on demand every time it is called or it can return an appropriate object from a pool of objects created in advance. We are using the following strategies to add the fake object to finding guilty agent.

```
Algorithm: Implementing Fake Objects
Input:   R1, .... , Rn, cond1 , ... , condn , b1 ,... ,b, B
Output:  R1, . . . , Rn, F1 ,. . . ,Fn
1:  R ← Ø     .Agents that can receive fake objects
2:  for i = 1, . . . , n do
3:  if b > 0 then
4:  R ← R ∪ {i}
5:  Fi  ← Ø       ..Set of fake objects given to agent U
```

```
6:  while B > 0 do
7:  i ← SELECT_AGENT_RANDOM(R, R)
8:  f ← CR E AT E FA K E OB J E C T(R, F)
9:      Ri ← R∪ {f }
10:     Fi ← Fi ∪ {f }
11:     if b← bi - 1
12:  if bi= 0 then
13:          R ← R\{R}
14:     B ← B – 1
```

Data Distributor Module:

Once data allocation is done by adding fake object Intelligently into the data, data distributor (in reality a server) hands over the requested data to the desired agent. Now it is data distributor's responsibility to keep the track of that data.

e.g. The data distributor gives data to agent u1. Suppose an agent u1 decides to leak that data to one or more other agents or organizations. If the data gets leaked and found in an unauthorized place (e.g., on the web or somebody's laptop) then it is data distributor's responsibility to keep the track of that data and at the end assess the guilt of an agent by using agent's guilt model (i.e., concept of probability).[3]

Probability assumption module:

This module makes use of access patterns and access variations and other client statistics to detect a probability of a particular client/ agent of leaking the data. If a data file or information is leaked, the probability assumption module suggests the client with highest probability as a culprit (i.e., a data leaker).

E.g. Let u1 and u2 be the two authorized agents. Highest value of any probability will be 1. Suppose an agent u1 accesses a file named as (let say) abc.txt for 6 times and an agent u2 accesses the same file for only 3 times. Then in this case, since an agent u1 has accessed a file two times more as compared to an agent u2, the probability of an agent u1 to be guilty should also be twice that that of an agent u2. Hence the probability of agent u1 to be guilty is 0.67 and the probability of an agent u2 to be guilty is 0.33. But suppose it is possible that an agent u1 only accesses a file 6 times but not modified it and an agent u2 not only accesses a file but also modified it one or more times. All this information will be saved in a separate log file which is stored at a server. Then in such case, probability of an agent u2 to be guilty is definitely more than that of an agent u1.

Mathematically, let Gi be an event that represents an agent Ui to be guilty and let S be the set of leaked data objects. Now we need to estimate P (Gi |S) i.e., we need to estimate that the agent Ui to be guilty for a given set S of leaked data object since write access always has more priority over read access.

Let us assume that all T objects have the same the same probability p. We need to take into consideration the following two assumptions regarding the relationship among the various leakage events.

Assumption 1:-

For all t , $t' \in S$ such that $t \neq t'$ the provenance of t is independent of the provenance of t'.

This assumption simply states that an agent's decision to leak an object is not related to other objects. In other words, joint events have a negligible probability. This assumption gives us more conservative estimates for the guilt of agents, which is consistent with our goals.

Assumption 2 :-

Let S be the set that represents a set of leaked data object. Let t be a leaked data object such that $t \in S$. Any agent Ui can obtain an object t by one of the following two ways.

- A single agent U_i leaked t from his own R_i set.

OR

- The target guessed t (or obtained through any other means) without the help of any of the n agents.

Assume that sets *T*, *R*'s and *S* are as follows:-

$T = \{t_1, t_2, t_3\}$, $R_1 = \{t_1, t_2\}$, $R_2 = \{t_1, t_3\}$, $S = \{t_1, t_2, t_3\}$.

Here set T denotes a set of fake object which will be added by data distributor intelligently into the agent's data set. Let U1 and U2 be the two agents having data sets R1 and R2 respectively. Let S be the set that represents a set of leaked data objects. Here we can see that all three of the distributor's objects have been leaked and appear in S.

In above data sets, we can see that an object t1 is given to both the agents. According to assumption 2, the target either guessed t_1 or one of U_1 or U_2 leaked it. We know that the probability of the former event is p. Hence we conclude the following:-

The target guessed t_1 with probability p
Agent U_1 leaked t_1 to S with probability $(1-p)/2$
Agent U_2 leaked t_1 to S with probability $(1-p)/2$

Similarly, we find that an agent U_1 has leaked t_2 to S with probability $(1-p)$ since it is the only agent that

has this data object and agent U_2 has leaked t_2 to S with probability
(1− p) since it is also the only agent that has this data object.

To assess the probability of any guilty agent (u1 and u2 in this case), we can use the basic concept of probability i.e.,

P (at least one) = 1 – P (none).

By using above concept, we can compute the probability that agent u1 is not guilty as follows:-

$$P\{\bar{G}_1|S\} = (1-(1-p)/2)\times(1-(1-p)) \quad \text{----------} \quad (1)$$

Hence, the probability that an agentU_1 is guilty is as follows:-

$$P\{G_1|S\} = 1 - P\{\bar{G}_1|S\} \quad \text{----------} \quad (2)$$

In general, we first consider a set of agents (let say) agents $V_t = \{U_i \mid t \in R_i\}$ where i = 1 to n. Ri be the set of data objects that are allocated to an agent U_i and t be an any data object that belongs to the set Ri.

Now our aim is to find the probability that an agent U_i is guilty given a set S. By using Assumption 2 and known probability p, we have:-

$$P\{\text{some agent leaked } t \text{ to } S\} = 1 - p. \text{ -------} \quad (3)$$

Let us assume that all agents that belong to set V_t can leak a data object t to S with equal probability and by using assumption 2 we obtain:-

$$P\{U_i \text{ leaked} t \text{ to } S\} = \begin{cases} \frac{1-p}{|V_t|}, & \text{if } U_i \in V_t \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Here we assume that any agent U_i is guilty if that agent leaks at least one data object (i.e., object t in our case) to a set S then by using our assumption 1 and eq. 4 given above, we can finally compute the probability $P\{G_i|S\}$ that an agent U_i is guilty as follows:-

$$Pr\{G_i|S\} = 1 - \prod_{t \in S \cap R_i} \left(1 - \frac{1-p}{|V_t|}\right) \text{ --------} \quad (5)$$

5. Graphical User Interface

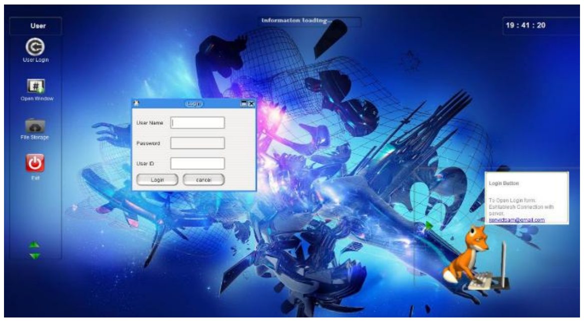


Figure 5.1: Login Screen (Client application)

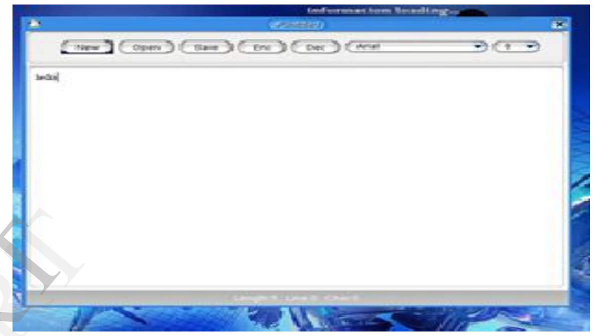
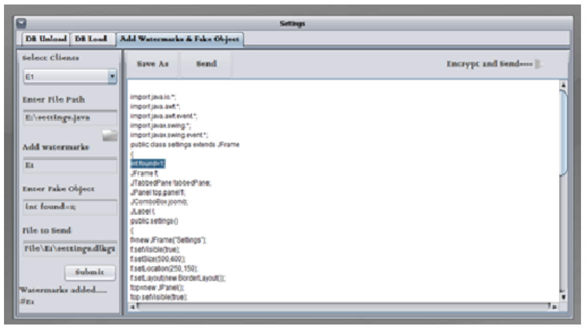


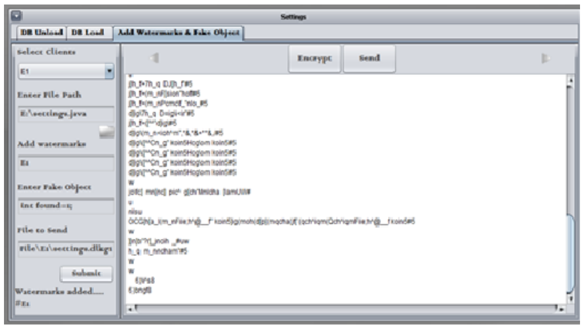
Figure 5.2: Client Accessing and modifying information



Select Clients: E1
Enter File Path: E:\settings.java
Add watermarks: E1
Enter Fake Object: int found=1;
File to Send: File\E1\settings.dllags
Watermarks added.....
#E1

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class settings extends JFrame
{
int found=1;
JFrame f;
JTabbedPane tabbedPane;
JPanel top,panel1;
JComboBox jcomb;
JLabel l;
public settings()
{
f=new JFrame("Settings");
f.setVisible(true);
f.setSize(500,400);
f.setLocation(250,150);
f.setLayout(new BorderLayout());
top=new JPanel();
top.setVisible(true);
```

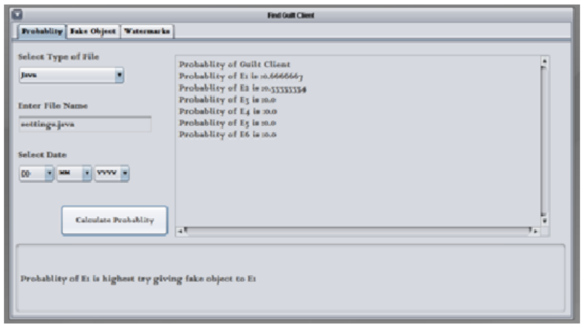
Figure 5.3: Add Watermark & Fake Object



Select Clients: E1
Enter File Path: E:\settings.java
Add watermarks: E1
Enter Fake Object: int found=1;
File to Send: File\E1\settings.dlkg1
Watermarks added.....
#E1

```
[illegible]
```

Figure 5.4: Encrypted Data

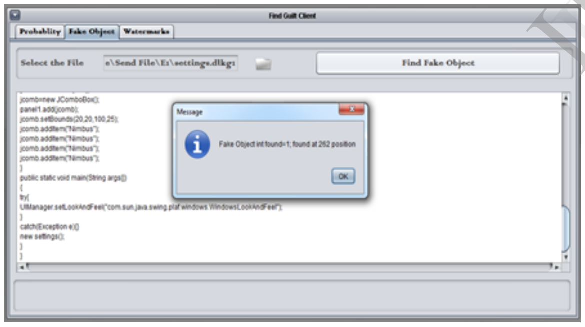


Select Type of File: Java
Enter File Name: settings.java
Select Date: DD MM YYYY

```
Probablity of Guilt Client
Probablity of E1 is :0.6666667
Probablity of E2 is :0.33333334
Probablity of E3 is :0.0
Probablity of E4 is :0.0
Probablity of E5 is :0.0
Probablity of E6 is :0.0
```

Probablity of E1 is highest try giving fake object to E1

Figure 5.5: Probability of Clients



Select the File: e\Send File\E1\settings.dlkg1

```
jcomb=new JComboBox();
panel1.add(jcomb);
jcomb.setBounds(20,20,100,25);
jcomb.addItem("Nimbus");
jcomb.addItem("Nimbus");
jcomb.addItem("Nimbus");
jcomb.addItem("Nimbus");
}
public static void main(String args[])
{
try{
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
}
catch(Exception e){}
new settings();
}
}
```

Fake Object int found=1; found at 262 position

Figure 5.6: Find Fake object

6. References

[1] Panagiotis Papadimitriou, Hector Garcia-Molina, “Data Leakage Detection” , IEEE Transactions on Knowledge and Data Engineering”, January 2011
[2] Mungamuru and H. Garcia-Molina, “Privacy, Preservation and Performance: The 3 P’s of Distributed Data Management,” technical report, Stanford Univ., 2008.
[3] Buneman and W.C. Tan, “Provenance in Databases,” Proc. ACM SIGMOD, pp. 1171- 1173, 2007