Available online at www.elixirpublishers.com (Elixir International Journal)

Information Technology

Elixir Inform. Tech. 71 (2014) 24965-24969

ISSN: 2229-712X

# Data leakage detection in single sign on system

G.S.Pugalendhi
Information Technology, Maharaja Institute of Technology, Coimbatore, India.

**ARTICLE INFO**

**Article history:**
Received: 13 February 2014;
Received in revised form:
6 June 2014;
Accepted: 13 June 2014;

**Keywords**
Single Sign On;
Random Access Memory.

**ABSTRACT**
In Single Sign On (SSO) Systems clients are allowed to login into multiple websites using a single user id and password. In that case single sign on data distributor has given sensitive data to a set of supposedly trusted agents (third parties). Some of the data are leaked and found in an unauthorized place (e.g. on the web). The distributor must assess the likely hood that the leaked data came from one or more agents, as opposed to having been independently gathered by other means. This paper presents a novel approach for data allocation strategies (across the agents) that improve the probability of identifying leakages of user profile information's. In some cases, we can also inject "realistic but fake" data records to further improve our chances of detecting leakage and identifying the guilty party.

© 2014 Elixir All rights reserved

## 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 have been leaked by agents, and if possible to identify the agent that leaked the data[1]. We consider applications where the original sensitive data cannot be perturbed. Perturbation is a very useful technique where the data are modified and made "less sensitive" before being handed to agents. For example, one can add random noise to certain attributes, or one can replace exact values by ranges. However, in some cases, it is important not to alter the original distributor's data. For example, if an outsourcer is doing our payroll, he must have the exact salary and customer bank account numbers.

If medical researchers will be treating patients (as opposed to simply computing statistics), they may need accurate data for the patients. 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[2]. 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. In this paper, we study unobtrusive techniques for detecting leakage of a set of objects or records. Specifically, we study the following scenario : After giving a set of objects to agents, the distributor discovers some of those same objects in an unauthorized place. (For example, the data may be found on a website, or may be obtained through a legal discovery process.)[1] At this point, the distributor can assess the likelihood that the leaked data came from one or more agents, as opposed to having been independently gathered by other means. Using an analogy with cookies stolen from a cookie jar, if we catch Freddie with a single cookie, he can argue that a friend gave him the cookie. But if we catch Freddie with five cookies, it will be much harder for him to argue that his hands were not in the cookie jar. If the distributor sees "enough evidence" that an agent leaked data, he may stop doing business with him, or may initiate legal proceedings. In this paper, 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[2]. If it turns out that an agent was given one or more fake objects that were leaked, then the distributor can be more confident that agent was guilty.

## System Analysis

### *Existing System*

The guilt detection approach we present is related to the data provenance problem tracing the lineage of S objects implies essentially the detection of the guilty agents. Tutorial provides a good overview on the research conducted in this field[3]. Suggested solutions are domain specific, such as lineage tracing for data warehouses and assume some prior knowledge on the way a data view is created out of data sources. Our problem formulation with objects and sets is more general and simplifies lineage tracing, since we do not consider any data transformation from Ri sets to S.

As far as the data allocation strategies are concerned, our work is mostly relevant to watermarking that is used as a means of establishing original ownership of distributed objects. Watermarks were initially used in imagtes, video, and audio data whose digital representation includes considerable redundancy [4]. Recently other works have also studied marks insertion to relational data. Our approach and watermarking are similar in the sense of providing agents with some kind of receiver identifying information. However, by its very nature, a watermark modifies the item being watermarked. If the object to be watermarked cannot be modified, then a watermark cannot be inserted. In such cases, methods that attach watermarks to the

Tele:
E-mail addresses: pugal_lendhi@yahoo.co.in

© 2014 Elixir All rights reserved

distributed data are not applicable.

Finally, there are also lots of other works on mechanisms that allow only authorized users to access sensitive data through access control policies. Such approaches prevent in some sense data leakage by sharing information only with trusted parties[3]. However, these policies are restrictive and may make it impossible to satisfy agents' requests.

***Proposed System***

A distributor owns a set of valuable data objects. The distributor wants to share some of the objects with a set of agens U1; U2; … ; $U_N$. but does not wish the objects be leaked to other third parties. The objects in T could be of any type and size, e.g., they could be tuples in a relation, or relations in a database. An agent receives a subset of objects Ri T, determined either by a sample request or an explicit request: Sample request Ri ¼ SAMPLEδt;MIρ: Any subset of mi records from T can be given to Ui.Explicit request Ri ¼ EXPLICITδT; condiρ: Agent Ui receives all T objects that satisfy condi. Example. Say that T contains customer records for a given company A. Company A hires a marketing agency U1 to do an online survey of customers. Since any customers will do for the survey, U1 request a sample of 1,000 customer records. At the same time, company A subcontracts with agent U2 to handle billing for all California customers. Thus, U2 receives all T records that satisfy the condition "state is California". Although we do not discuss it here, our model can be easily extended to requests for a sample of objects that satisfy a condition (e.g., an agent wants any 100 California customer records). Also note that we do not concern ourselves with the randomness of a sample. (We assume that if a random sample is required, there are enough T records so that the to-be-presented object selection schemes can pick random records from T.)

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[4]. 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 $U_i$ is guilty and if it contributes one or more objects to the target. We denote the event that agent $U_i$ is guilty by Gi and the vent that agent Ui is guilty for a given leaked set S by $G_{ij}S$. Our next step is to estimate $P_rfG_{ij}S_g$, i.e., the probability that agent Ui is guilty given evidence S.

**Project description**

***Module description***

The template is designed so that author affiliations are not repeated each time for multiple authors of the same affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization). This template was designed for two affiliations.

User Profile

This is used to manage the user profile data onto the database. This assigns a unique account ID to each user. The target audience may be existing users or, in the case of a new site, new users. User profiles (also referred to as user persons) are an excellent way to document and illustrate realistic sample users. User profiles are short bios or narratives about a user and their use of a Web site. These personas typically are concise one page documents.

Creating user personas is often the job of an information architect or designer who understands the target user groups and is experienced in creating these documents. However, in some situations it's beneficial having the combined Web group or project team collaborate to develop the user profiles. Working as a group will the team to focus on and understand the various user groups you are targeting. Typically how profiles are developed will depend on the size of the site, budget, and timeframe[10].

It's important to make sure your personas accurately describe the target (or existing) audiences. They should be based on your current understanding of existing users (if they exist), research, user interviews, and the knowledge of content experts and clients.

**A user profile should include;**

- Name
- Occupation
- Age
- Gender
- Education
- Banking Credentials
- Shopping Details

**Registration**

This contains a registration form which gets user input and stores them into the database.

**Edit / Delete**

User profile modifying is done in this module.

User Data

This module stores the user profile and their website information on the database in addition to their account details. User uploaded data's and their website resource access limits are stored[5].

***Activity Session***

This module creates and manages the user activity details on the website. The User Profile Service class in the Web service includes methods to manage user profiles. For example, to add a link to the My Links page on the My Site for the specified account name, you use the AddLink method of the User Profile Service class. To remove a colleague from the My Colleagues page for the specified account name, you use the Remove Colleague method[9].

You can also use the relevant properties of various classes in the User Profile Service Web service namespace to get or set a particular property. For example, to get or set common memberships that two user profiles share, you use the Memberships property in the In Common Data class. To specify or determine whether a property value was changed for a

particular user profile property, you use the Is Value Changed property of the Property Dataclass[6].

The User Profile Service Web service provides a user profile interface for remote clients to read and create user profiles. To use the User Profile Service Web service library, you must generate a proxy class in either Microsoft Visual C# or Microsoft Visual Basic through which you can call the various Web service methods.

The Web Services Description Language (WSDL) for the User Profile Servcie Web service endpoint is accessed through User Profile Service.asmx?wsdl[5].

The following example shows the format of the URL to the User Profile Service Web Service WSDL file.

http://<server>/<customsite>/_vti_bin/UserProfileService.asmx

***Session Information***

This module stores the details about their session with their login details includes time and type of login onto the website.

A session starts when :

- A new user requests an ASP file, and the Global.asax file includes a Session_OnStart procedure.
- A value is stored in a Session variable
- A user requests an ASP file, and the Global.asax file uses the <object> tag to instantiate an object with session scope.
- A session ends if a user has not requested or refreshed a page in the application for a specified period. By default, this is 20 minutes.

***Page Resource Access Data***

This module monitors and stores information about the user website pages visiting, the type of page, and frequency about the visit into their activity session data.

***Agent Admin***

This module manages the details about the websites that using the access control tools and provides the detail report about the user activity to the admin.

***Agent Registration***

This module is used to register the website, one admin user registration is allowed for the website. Admin registered can ad more websites under their account control.

***Tool Download***

Registered website admin are provided the tools for downloading.

***Tool Implementation***

Tools are automatically configured for the website on which they are going to be installed. Website admin are just want to copy the generated content into their webpages which they want to monitor.

***Tool Management***

This module is used to manage the admin tools; admin can add more tools, edit or delete their tools.

***Tool Sharing***

This module provides the tool sharing functionality to admin in order them to implement the monitoring control tool on their multiple websites.

***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, e.g. 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[6][7].

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. In many cases, the distributor may be limited in how many fake objects he can create. For example, objects may contain e-mail addresses, and each fake e-mail address may require the creation of an actual inbox (otherwise, the agent may discover that the object is fake). The inboxes can actually be monitored by the distributor: if e-mail is received from someone other than the agent who was given the address, it is evident that the address was leaked. Since creating and monitoring e-mail accounts consumes resources, the distributor may have a limit of fake objects.

If there is a limit, we denote it by B fake objects. Similarly, the distributor may want to limit the number of fake objects received by each agent so as to not arouse suspicions and to not adversely impact the agents’ activities. Thus, we say that the distributor can send up to bi fake objects to agent Ui. Creation. The creation of fake but real-looking objects is a nontrivial problem whose thorough investigation is beyond the scope of this paper. Here, we model the creation of a fake object for agent Ui as a black box function CREATEFAKEOBJECT Ri; Fi;[7] condition that takes as input the set of all objects Ri, the subset of fake objects Fi that Ui has received so far, and condition, and returns a new fake object. This function needs Condition 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. For example, the creation function of a fake payroll record that includes an employee rank and a salary attribute may take into account the distribution of employee ranks, the distributionof salaries, as well as the correlation between the two attributes. Ensuring that key statistics do not change by the introduction of fake objects is important if the agents will be using such statistics in their work. Finally, function CREATEFAKEOBJECT() has to be aware of the fake objects Fi added so far, again to ensure proper statistics. The distributor can also use function CREATEFAKEOBJECT() when it wants to send the same fake object to a set of agents.

In this case, the function arguments are the union of the Ri and Fi tables, respectively, and the intersection of the conditions condis. Although we do not deal with the implementation of CREATEFAKEOBJECT(), WE NOTE THAT THERE ARE TWO MAIN DESIGN OPTIONS. 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.

***Fake Object Allocation Strategies***

In this section, we describe allocation strategies that solve exactly or approximately the scalar versions of for the different

instances presented. We resort to approximate solutions in cases where it is inefficient to solve accurately the optimization problem.

***Explicit Data Requests***

In problems of class EF, the distributor is not allowed to add fake objects to the distributed data. So, the data allocation is fully defined by the agents' data requests. Therefore, there is nothing to optimize. In EF problems, objective values are initialized by agents' data requests. Say, for example, that T ¼ ft1; t2g and there are two agents with explicit data request such that R1 ¼ ft1; t2g and R2 ¼ ft1g.

The distributor cannot remove or alter the R1 or R2 data to decrease the overlap R1\R2. However, say that the distributor can create one fake object (B ¼ 1) and both agents can receive one fake object (b1 ¼ b2 ¼ 1). In this case, the distributor can add one fake object to either R1 or R2 to increase the corresponding denominator of the summation term. Assume that the distributor creates a ake object f and he gives it to agent R1. Agent U1 has now R1 ¼ ft1; t2; $f_g$ and F1 ¼ $f_{fg}$ and the value of the sum-objective decreases to 1 3 ρ 1 1 ¼ 1:33<1:5

If the distributor is able to create more fake objects, he could further improve the objective. We present in Algorithms 1 and 2 a strategy for randomly allocating fake objects. Algorithm 1 is a general "driver" that will be used by other strategies, while Algorithm 2 actually performs the random selection. We denote the combination of Algorithm 1 with 2 as e-random. We use e-random as our baseline in our comparisons with other algorithms for explicit data requests[8]

**System Flow Diagram**

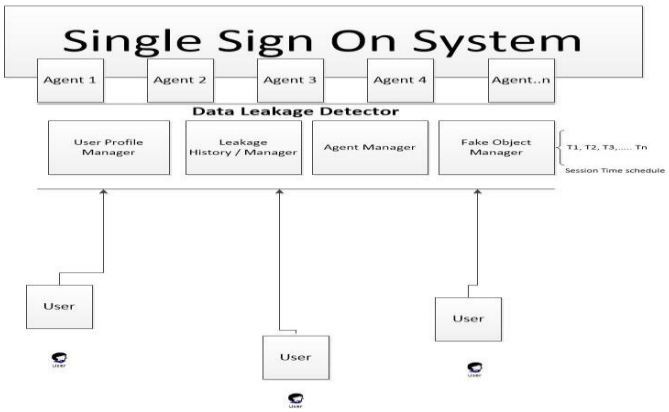


**Figure 1. System Flow Diagram**

**Table design**

**TABLE .1 TB USER TABLE**

| Field name | Data type | Constraint |
|---|---|---|
| Userid | Varchar(50) | Primary Key |
| Name | Varchar(50) | Primary Key |
| Password | Varchar(50) | Primary Key |
| Fullname | Varchar(50) | Primary Key |
| Email | Varchar(50) | Primary Key |
| Age | Varchar(50) | Primary Key |
| Gender | Varchar(50) | Primary Key |
| Country | Varchar(50) | Primary Key |
| City | Varchar(50) | Primary Key |
| State | Varchar(50) | Primary Key |
| Address | Varchar(50) | Primary Key |

**TABLE 2. TB USER IN SESSION**

| Field name | Data type | Constraint |
|---|---|---|
| Sessionname | Varchar(50) | Primary Key |
| Status | Varchar(50) | Primary Key |

**TABLE 3. TB SESSION**

| Field name | Data type | Constraint |
|---|---|---|
| Sessionid | Varchar(50) | Primary Key |
| Userid | Varchar(50) | Primary Key |
| Websiteid | Varchar(50) | Primary Key |
| Lastlogin | Varchar(50) | Primary Key |
| Visitedarea | Varchar(50) | Primary Key |
| Country | Varchar(50) | Primary Key |
| State | Varchar(50) | Primary Key |
| Webtype | Varchar(50) | Primary Key |
| Gender | Varchar(50) | Primary Key |
| Age | Varchar(50) | Primary Key |

**TABLE 4. TB AGENT TABLE**

| Field name | Data type | Constraint |
|---|---|---|
| Website Id | Varchar(50) | Primary Key |
| Password | Varchar(50) | Secondary Key |
| Name | Varchar(50) | Secondary Key |
| Web type | Varchar(50) | Primary Key |
| Country | Varchar(50) | Secondary Key |
| State | Varchar(50) | Primary Key |

**TABLE 5. TB LEAKAGE DETECTION**

| Field name | Data type | Constraint |
|---|---|---|
| User id | Varchar(50) | Primary Key |
| Name | Varchar(50) | Primary Key |
| Password | Varchar(50) | Primary Key |
| Full name | Varchar(50) | Primary Key |
| Email | Varchar(50) | Primary Key |
| Age | Varchar(50) | Primary Key |
| Gender | Varchar(50) | Primary Key |
| Country | Varchar(50) | Primary Key |
| City | Varchar(50) | Primary Key |
| State | Varchar(50) | Primary Key |
| Address | Varchar(50) | Primary Key |

**Conclusion**

In a perfect world, there would be no need to hand over sensitive data to agents that may unknowingly or mailiciously leak it. And even if we had to hand over sensitive data, in a perfect world, we would watermark each object so that we could trace it origins with absolute certainity. However, in many cases, we must indeed work with agents that may not be 100 percent 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 that 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 that 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.

**References**

[1] R.Agrawal and J.Kiernan, "Watermarking Relational Databases," Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02), VLDB Endowment, pp. 155-166, 2002.

[2] P.Bonatti, S.D.C. di Vimercati, and P.Samarati, "An Algebra for Composing Access Control Policies," ACM Trans. Information and System Security, Vol.5.no.1, pp.1-35,2002.

[3] P.Buneman, S.Khanna, and W.C. Tan, "Why and Where: A Characterization of Data Provenance," Proc. Eighth Int'l Conf. Database Theory (ICDT '01), J.V. den Bussche and V.Vianu, eds., pp.316-330, Jan. 2001

[4] P.Buneman and W.-C.Tan, "Provenance in Databases," Proc.ACM SIGMOD, pp. 1171-1173, 2007.

[5] Y.Cui and J.Widom, "Lineage Tracing for General Data Warehouse Transformations," The VLDB J., vol. 12, pp. 41-58, 2003.

[6] S.Czerwinski, R. Fromm, and T. Hodes, "Digital Music Distribution and Audio Watermarking," http://www.scientificcommons.org/43025658, 2007.

[7] F.Guo, J.Want, Z. Zhang, X.Ye, and D.Li, "An Improved Alorithm to Watermark Numeric Relational Data," Information Security Applications, pp. 138-149, Springer, 2006.

[8] F. Hartung and B. Girod, "Watermarking of Uncompressed and Compressed Video," Signal Processing, vol. 66, no.3, pp. 283-301, 1998.

[9] S. Jajodia, P.Samarati, M.L. Sapino, and V.S. Subrahmanian, "Flexible Support for Multiple Access Control Policies," ACM Trans. Database Systems, Vol. 26, no. 2, pp. 214-260, 2001.

[10] Y. Li, V. Swarup, and S. Jajodia, "Fingerprinting Relational Databases: Schemes and Specialities," IEEE Trans. Dependable and Secure Computing, vol. 26, no.2, pp. 34-45, Jan – Mar 2005.