International Journal of Engineering Research and Development
***e*-ISSN: 2278-067X, *p*-ISSN: 2278-800X, www.ijerd.com**
Volume 4, Issue 1 (October 2012), PP. 11-16

Detection and Avoidance of Data Leakage

Sandesh D Manocharya[1], A Prabhakar[2]
[1]*Department of CSE, CIT, Gubbi, Tumkur, India*
[2]*Department of ISE, CIT, Gubbi, Tumkur, India*

Abstract—A 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 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 cases, We can also inject "realistic but fake" data records to further improve the chances of detecting leakage and identifying the guilty party.

Keywords—Allocation strategies, data leakage, fake objects, hash value.

I. 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 or server and the supposedly trusted third parties the agents or routers. The goal of this project is to detect if the distributor's sensitive data have been leaked by agents, to identify the agent that leaked the data and also to avoid the leakage.

Consider the 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 [1]. 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.

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). 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 Rama is caught with a single cookie, he can argue that a friend gave him the cookie. But if Rama is caught with five cookies, it will be much harder for him to argue that his hands were not in the cookie jar. If distributor sees "enough evidence" that an agent leaked data, he may stop doing business with him, or may initiate legal proceedings.

In this paper, a model is developed for assessing the "guilt" of agents. An algorithm is presented for distributing objects to agents, in a way that improves our chances of identifying a leaker. Finally, to 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 that an agent was given one or more fake objects that were leaked, then the distributor can confidently tell that agent was guilty.

II. ARCHITECTURE

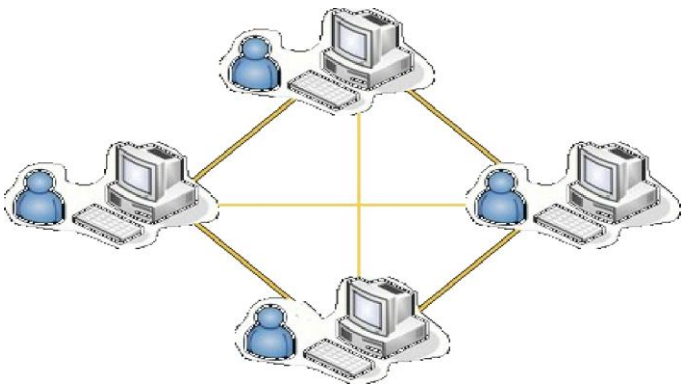

Figure 1. Network Architecture

Computers are connected in a local area network (LAN) as sown in the Figure 1. The system architecture is a way to structure a distributed application so that it consists of many identical software modules, each module running on a different computer. The different software modules communicate with each other to complete the processing required for the completion of the distributed application. One could view the system architecture as placing a server module, router module as well as a client module on each computer. Each computer would need to know the network addresses of the other computers running the distributed application, or at least of that subset of computers with which it may need to communicate. Furthermore, propagating changes to the different software modules on all the different computers would also be much harder. However, the combined processing power of several large computers could easily surpass the processing power available from even the best single computer.

III. OBJECTIVES

1. Aim of this paper is: to detect the leakage of data while transmitting the data from server to client via various routers.
2. Identifying the guilty router.
3. And also avoiding the leakage.

IV. ANALYSIS

As shown in Figure 2, a server, two or more routers and a client are connected in a local area network (LAN).

Server: An already registered user logs into the server with his/her valid user name and password. If the user is not yet registered then he/she has to create a new account by clicking on the link "New User". For security purpose the user name and passwords will be encrypted and stored in database. After a successful login user can change his/her password by clicking on the button "Change Password". Then by clicking the button "Insert Client IP" user must enter the client IP address of intended or authorized client to which the data to be sent. The authorized client IP will be stored in database. If the user clicks the button "Get All IP Address", then system name and IP addresses of all the active systems which are connected in the LAN will be traced. From these lists of IPs user can either select any two IPs to make them as routers or can enter the IP addresses manually in the respective text boxes. Then user has to browse the file to be sent to the client. Also user has to browse the fake object file to pad it with the content of the file. The fake object file is used for uniqueness of the file as server sends same file to both the routers. Server encrypts the file using a secret key. Depending on the number of routers, server selects the characters from the fake object file. In this project I am using two routers. So server selects (Number of routers X 2) = (2 X 2) = first 4 characters from the fake object file. Server will pad the set of first two characters to the encrypted file content and transfers the file to the first router. Similarly server will pad second set of two characters to the encrypted file content and transfers the file to the second router. Before transferring the file, server generates two different hash values for each file. Now after clicking the button "Transfer", a message "File sent successfully" will appear on the screen.

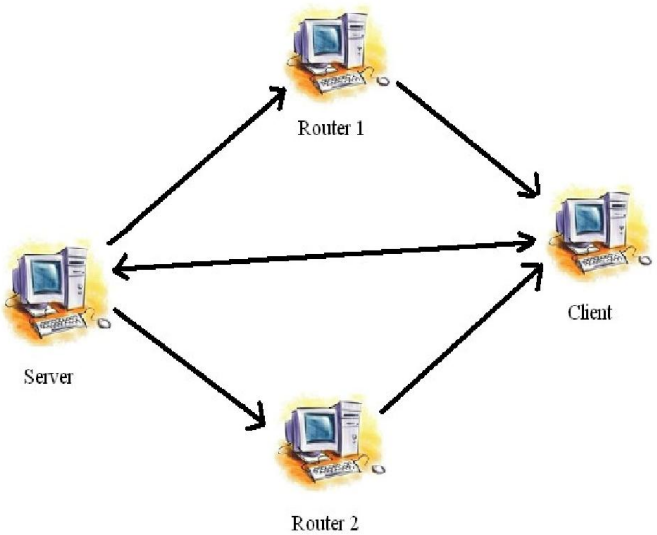

Figure 2. Network Design

Router: After selecting the path where the file has to be stored, the encrypted file will be saved in selected path in the router. Now user has to enter the client IP address to which the file to be sent. If the entered client IP address is same as that of authorized client IP address stored in server, then file will be transferred to the authorized client along with the secret key to decrypt the file. Otherwise the secret key will not be sent to the unauthorized client.

Client : The authorized client selects the path where the file has to be stored, receives the encrypted file from the router, receives the secret key from the server to decrypt the file and stores the decrypted file in the selected path. The unauthorized

client also generates a hash value for received encrypted file content and fake objects. Then transmits the hash value to the server. Server compares the hash value sent by unauthorized client and the computed hash value. Depending on these hash values server comes to know that which router has leaked the data. In a status window at server, a message saying that which router is leaked the data will appear. Also a log file will be generated automatically on desktop of server showing the history of the data leakage with date and time. If the server selects the same guilty router for second time unknowingly or mistakenly, then a text "Untrusted IP" will appear in front of the respective text box of IP address.

V. MD5 ALGORITHM

MD5 is an algorithm that is used to verify data integrity through the creation of a 128-bit message digest from data input (which may be a message of any length) that is claimed to be as unique to that specific data as a fingerprint is to the specific individual.MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit little endian integers); the message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with a 64-bit little endian integer representing the length of the original message, in bits.

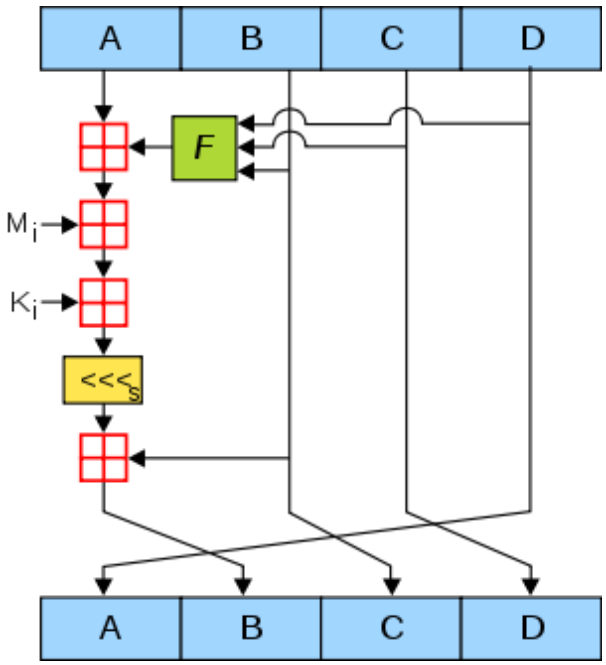

Figure 3. MD5 Algorith

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted *A*, *B*, *C* and *D*. These are initialized to certain fixed constants. The main algorithm then operates on each 512-bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed *rounds*; each round is composed of 16 similar operations based on a non-linear function *F*, modular addition, and left rotation. Figure 3 illustrates one operation within a round. There are four possible functions *F*; a different one is used in each round:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

$\oplus, \wedge, \vee, \neg$
denote the XOR, AND, OR and NOT operations respectively.

VI. DES (DATA ENCRYPTION STANDARD) ALGORITHM

DES is the archetypal block cipher — an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bitstring of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits, and it is never quoted as such. Every 8th bit of the selected key is discarded, that is, positions 8, 16, 24, 32, 40, 48, 56, 64 are removed from the 64 bit key leaving behind only the 56 bit key.

Like other block ciphers, DES by itself is not a secure means of encryption but must instead be used in a mode of operation. FIPS-81 specifies several modes for use with DES. Further comments on the usage of DES are contained in FIPS-74.

Feistel Structure of DES Algorithm

The algorithm's overall structure is shown in Figure 4: there are 16 identical stages of processing, termed *rounds*. There is also an initial and final permutation, termed *IP* and *FP*, which are inverses (IP "undoes" the action of FP, and vice versa). IP and FP have almost no cryptographic significance, but were apparently included in order to facilitate loading blocks in and out of mid-1970s hardware. Before the main rounds, the block is divided into two 32-bit halves and processed alternately; this crisscrossing is known as the Feistel scheme. The Feistel structure ensures that decryption and encryption are very similar processes — the only difference is that the subkeys are applied in the reverse order when decrypting. The rest of the algorithm is identical. This greatly simplifies implementation, particularly in hardware, as there is no need for separate encryption and decryption algorithms. The ⊕ symbol denotes the exclusive-OR (XOR) operation. The *F-function* scrambles half a block together with some of the key. The output from the F-function is then combined with the other half of the block, and the halves are swapped before the next round. After the final round, the halves are not swapped; this is a feature of the Feistel structure which makes encryption and decryption similar processes.

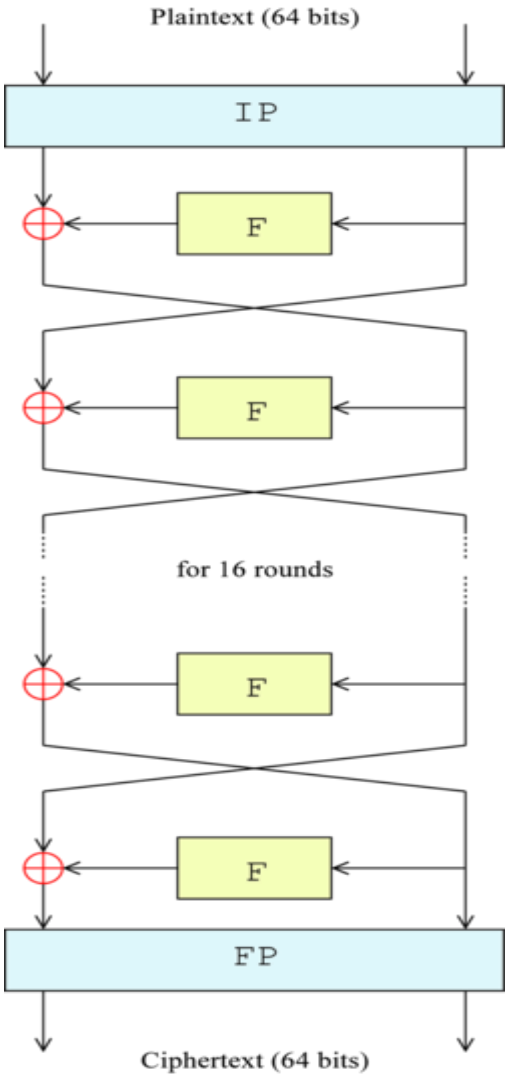

Figure 4. Feistel Structure of DES Algorithm

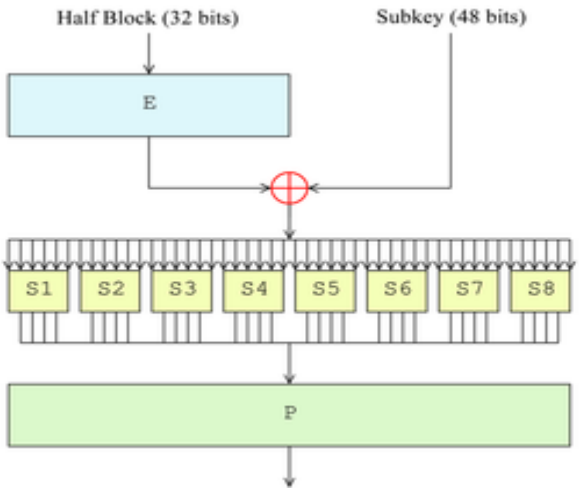

Figure 5. The Feistel Function (F-function) of DES

Feistel(F) Function

The F-function, depicted in Figure 5, operates on half a block (32 bits) at a time and consists of four stages:

1. Expansion–the 32-bit half-block is expanded to 48 bits using the expansion permutation, denoted E in the diagram, by duplicating half of the bits. The output consists of eight 6-bit (8 * 6 = 48 bits) pieces, each containing a copy of 4 corresponding input bits, plus a copy of the immediately adjacent bit from each of the input pieces to either side.

2. Key mixing– the result is combined with a subkey using an XOR operation. 16 48-bit subkeys — one for each round — are derived from the main key using the key schedule.

3. Substitution– after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by the S-boxes or substitution boxes. Each of the eight S-boxes replaces its six input bits with four output bits according to a non-linear transformation, provided in the form of a lookup table. The S-boxes provide the core of the security of DES — without them, the cipher would be linear, and trivially breakable.

4. Permutation– finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation, the P-box. This is designed so that, after expansion, each S-box's output bits are spread across 6 different S boxes in the next round.

The alternation of substitution from the S-boxes, and permutation of bits from the P-box and E-expansion provides so-called "confusion and diffusion" respectively, a concept identified by Claude Shannon in the 1940s as a necessary condition for a secure yet practical.

VII. RESULTS

Figure 6. Status of Leakage

When a particular router leaks the data i.e. forwards the encrypted file to the unauthorized client then a status window will appear on the screen of the server system saying that the particular router has leaked the data as shown in the Figure 6. Now the server does not send the secret key to the unauthorized client to decrypt the received encrypted file. If the router forwards the data to the authorized client IP address then the status will be 'Normal'.

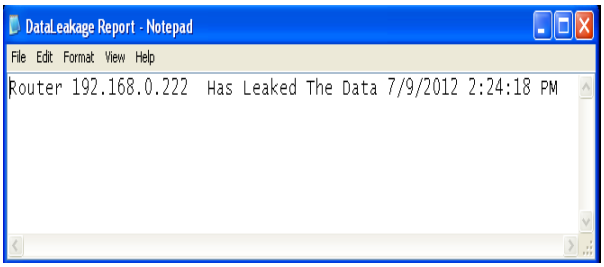

Figure 7. History of Leakage

After a data leak by a guilty router a status window will appear on screen of the server system. The status window contains a button 'View Log' button. When the user clicks the button 'View Log' a log file will open that contains the history of the data leak with date and time as shown in Figure 7.

VIII. CONCLUSION

In a perfect world, there would be no need to hand over sensitive data to agents that may unknowingly or maliciously leak it. And even if the sensitive data needs to be transmitted, watermarking each object can help in tracing its origins with absolute certainty. However, in many cases, there is a need to work with the agents that may not be trusted and it 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, this project has 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. This model is relatively simple, but it can capture the essential trade-offs. The algorithms presented can improve the distributor's chances of identifying a leaker. It is also 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.

BIBLIOGRAPHY

[1]. L. Sweeney, "Achieving K-Anonymity Privacy Protection Using Generalization and Suppression," http://en.scientificcommons.org/43196131, 2002.
[2]. 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 Busscheand V. Vianu, eds.,pp. 316-330, Jan. 2001.
[3]. P. Buneman and W.-C. Tan, "Provenance in Databases," Proc.ACM SIGMOD, pp. 171-1173, 2007.
[4]. Y. Cui and J. Widom, "Lineage Tracing for General Data Warehouse Transformations," The VLDB J., vol. 12, pp. 41-58,2003.
[5]. 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.
[6]. R. Sion, M. Atallah, and S. Prabhakar, "Rights Protection forRelational Data," Proc. ACM SIGMOD, pp. 98-109, 2003.
[7]. S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian, "Flexible Support for Multiple Access Control Policies," ACMTrans. Database Systems, vol. 26, no. 2, pp. 214-260, 2001.
[8]. 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.