APPLICATION SECURITY, INC.

Encryption of Data at Rest

- Database Encryption -

White Paper

APPLICATION SECURITY, INC.
WEB: WWW.APPSECINC.COM
E-MAIL: INFO@APPSECINC.COM
TEL: 1-866-9APPSEC • 1-212-420-9270

TABLE OF CONTENTS

INTRODUCTION

The digital age is upon us. No longer is instant access to information a request – it is now a requirement. Serving as a backbone for instant access is the relational database management system. Databases serve as the warehouses of digital information and hold our most critical assets. As such, to properly maintain the integrity and confidentiality of this data, the need for securing databases is growing. One of the requirements for securing databases is to encrypt the information stored within them.

Unfortunately, there are many misconceptions surrounding what database encryption is and how it should be performed. Encryption is a complex subject and properly implementing it requires a grasp of not only the theories behind encryption, but also the practical applications in the real world. All too often, the line between access control and encryption is blurred, and encryption solutions simply supplement the access controls already in place. What we hope to outline in this paper is an appropriate use of encryption as well as its proper implementation.

"DEFENSE IN-DEPTH"

No single security solution can properly protect a system. What is most important is "defense in-depth." This means that more than a single layer of security is required in order to adequately protect a system. A good example of "defense in-depth" is a castle. The castle contains multiple defense systems – a moat, castle walls, archers on the walls, etc… Individually each defense system would not be able to deter an attacker, but combined the castle becomes very difficult to penetrate.

Encryption is one of the layers of security needed to secure your database. Of course, without implementing other security measures first, encryption is an inefficient and ineffective solution. Attempting to encrypt data that is not "locked down" utilizing the proper access controls leads to poor performance and poor security. For example, you may encrypt your PGP private key using a strong password on your laptop. If you did this, would it be appropriate to e-mail the encrypted key or provide public access to the key? Certainly not!

ACCESS CONTROLS

Now before even considering the implementation of encryption, you need to ensure that proper access controls are in place. Setting up access controls require the configuration of users and the actions they should be able to perform within the database. Within a database, access control consists of creating users and granting them the privilege to act on objects or performing certain tasks. The built-in controls and mechanisms within the database are your best means of providing access controls.

ENCRYPTION

Once you have access controls in place, encryption should then be implemented. Encryption provides an additional restriction if access controls are circumvented. In other words, encryption should stop someone who has already broken through the "first line of defense." When the hacker has broken through the first door – ACCESS CONTROL – encryption forms the next barrier of entry.

To demonstrate an appropriate method of using encryption on an operating system flat file, let us take a look at Microsoft's Encrypted File System (EFS) within a Windows environment. In our example, we will walk through securing a document that contains secrets critical to your organization's success stored on a file server.

The first thing you should do is set NTFS permissions on the file to prevent unauthorized users from reading the file. This is access control and should always be the first line of defense. However, there are several weaknesses with access control:

- NTFS permissions do not prevent system administrators from accessing the files
- If an attacker gains control of the operating system, they can use system administrator privileges to read the data
- An attacker can bypass the permission checking by booting the server to a different operating system to get around any access controls implemented by the original operating system

For example, let us say an attacker is aware of a buffer overflow in Microsoft Windows 2000 that allows him/her to run shell commands on the server. The attacker can then reset the Administrator password on the file server. Even though the appropriate permissions are established, they can be evaded by taking control of the operating system using the buffer overflow. Now the attacker has the ability to read the files on your file server using the Administrator account.

Is this type of attack avoidable? First off, your system should be patched well enough to withstand most buffer overflows. However, we live in a world where buffer overflows and other attacks are discovered on a daily basis. There is little chance you can "guarantee" invulnerability to this type of attack.

Encryption offers a reasonable, although far from perfect, solution. One way to protect your data even if an attacker gains full control over the data is by encrypting it. EFS can encrypt the file based on the user's password. By establishing a password as a key to the encryption, we have prevented an attacker with control of the operating system from reading the file. Two points to understand about encryption are the following:

- Encryption does not protect data from being deleted
- Encryption does not protect data from being modified, although it does provide you a way to tell if an unauthorized change has been made

Keep in mind the capabilities of encryption and its purpose. It is important that you maintain the proper backups so that if someone deletes or changes your encrypted data, you can restore the data.

ENCRYPTION OF DATA-IN-MOTION

Encryption can be categorized into two types – encryption of "data-at-rest" and encryption of "data-in-motion". The issues involved with providing each type of encryption are very different. This paper addresses the issues of encrypting "data-at-rest". However, we should touch briefly on encryption of "data-in-motion".

Encryption of "data-in-motion" hides information as it moves across the network from the database to the client or from the client to the database. Data-in-motion includes traffic moving over your local network, the Internet, or even over a wireless network. The various standards for this type of encryption include SSL (Secure Sockets Layer), TLS (Transport Layer Security), and IPSEC (Secure Internet Protocol). Most database vendors have adopted the SSL standard, and include the ability to send traffic between the client and database vendor over an SSL tunnel using some combination of RSA, RC4, DES, and the Diffie-Hellman algorithm.

Encryption of "data-in-motion" is necessary to prevent someone from intercepting traffic as it goes back and forth from the client and the database. Encryption of data-in-motion is also effective at preventing such attacks as session hijacking and replay attacks.

Encryption of data-in-motion is typically implemented at a session level – the network layer above the protocol being encrypted. Network communications are encrypted as they are being transmitted over the wire and decrypted as they are received at the other end. Each command sent by the client is encrypted as it is sent and decrypted as it is received by the database. As each result is returned from the database, it is encrypted as it is sent and decrypted as the client receives it.

ENCRYPTION OF "DATA-AT-REST"

Encryption of "data-at-rest" involves transforming information stored in the database into a form so that it is only readable by authorized individuals. Encrypting "data-in-motion" does nothing to protect data that is attacked at the end points. Consider that most attacks do not occur on data-in-motion. The vast majority of attacks occur against the end points where data sits for long periods of time. With that said, we see that implementation of encryption has been misdirected. Encrypting "data-in-motion" is already widely adopted while even the most "security-conscious" database administrators have not adopted encryption of "data-at-rest."

HOW DO WE ENCRYPT DATA-AT-REST?

There are several possible strategies to encrypt "data-at-rest," and each strategy has certain advantages and disadvantages. The rest of this paper will outline the strategies and propose the best solution.

Encryption of data-at-rest can be performed in several ways. One way is to encrypt the actual database files at the operating system level. An example of using this strategy is to encrypt an entire database file using Microsoft's EFS within a Windows environment.

There are many weaknesses to using this strategy. We have enumerated several of them here:

1) You cannot selectively encrypt individual pieces of data. This approach results in encrypting the entire file, which means all the data is encrypted. This causes serious performance problems for reading from the database. Every time data is read from the database, it is encrypted regardless of the whether or not the data really needs to be secured. This adds significant overhead to any action performed against the database.

2) Encrypting the entire file not only adds the overhead of reading all data, but also leads to other additional overhead when recording pointers, indexes, and other internal data structures that must be encrypted and decrypted for any operation against the database. Ideally, when an insert is made to a database, the only encryption required should be the encryption of the data being inserted. Using file-based encryption, the information to determine where in the file to store the new record, the index, and many other internal file structures must be decrypted in addition to the data being inserted.

3) Another weakness is that different pieces of data cannot be encrypted with different keys. Imagine if you had a database that contained both sales and personnel information. The Human Resources department should have access to the personnel data but not the sales data. The sales department should have access to the sales data but not the personnel data. Using file-level encryption, this cannot be achieved because operating system file encryption encrypts the entire file, not sections of the file.

4) File-based encryption only protects the data from operating system-level attacks. If an operating system user copies the physical database file, the information is secured from that user. It does not protect the data from a user who breaches the database. When someone breaks into the database, it will gladly decrypt the information for the database user. This is because it appears to be a properly authenticated user, thereby circumventing the encryption.

A more efficient and effective way to encrypt information in a database is to perform the encryption on a column and row basis. To further explain this concept, think of a table containing a list of customers. Within this customer table, there is the following information:

- Customer ID
- Customer name
- Customer address
- Customer credit card number

In this table there is little reason to encrypt the customer ID. It is most likely that you would only want the credit card information encrypted. You gain several advantages by only encrypting your most sensitive data, which in this case is credit card data. One advantage is that you can minimize the performance hit incurred by only encrypting sensitive information. For instance, when a user attempts to search the table for a specific user, they incur very minimal overhead because only the row matching the search criteria must be decrypted – a small subset of the data. Even better is the fact that when you select from other tables, which do not require encryption, there is absolutely no additional overhead added.

One of the serious problems encryption solves is protecting data from being read by administrators. This is accomplished by encrypting data utilizing a secret not known by the database administrator. Of course, the most important part of this statement is that the encryption must be dependent on restricting the administrator from discovering this secret, and utilizing it to decrypt the information within the database. For instance, if the administrator can simply reset the password of an account, logon to the account, and access the data, encryption has failed to protect the data from administrators. Encryption should be based on a secret such as a private key encrypted with a pass phrase. Therefore, if encryption were implemented utilizing a private key as a secret, the database administrator could not just simply reset the password of an account to decrypt the data.

Of course, this means that when the user needs to change his or her password, this also involves resetting the decryption keys. The encryption system that we are referring to here would utilize a single key to encrypt and decrypt data in each column. A copy of this key, called the column key, is then stored encrypted with the user's pass phrase. To further illustrate this concept:

> When I decide to change my pass phrase, I must first login with my current pass phrase, decrypt each column key, and then re-encrypt each column key with my new pass phrase replacing the old copies of the key. Next time I login with my new pass phrase, I can decrypt the keys with the new pass phrase. If an administrator simply changes the database pass phrase and logs in, he/she will decrypt the key to the wrong values because they are using the wrong pass phrase, and will be unable to decrypt the values in the table.

Using this technique, encryption is truly dependent on a secret, providing you a secure method of storing data within your database that even an administrator cannot view.

MINIMIZING PERFORMANCE PROBLEMS

One of the most important decisions you will make when utilizing encryption within your database is when you answer the following question: "What data should I encrypt?" Database lookups are designed to be very efficient. Unlike typical file systems, databases are expected to look through millions of rows searching for specific items in seconds. This need for fast access and retrieval places additional hardships on encrypting databases. A database cannot afford to encrypt and decrypt each piece of data it must search. Therefore, it is critical to properly plan encryption based on how an application will use the database. For example, let us imagine that we have a table with five columns and one million rows. The table contains customer information with the following columns: CustomerID, CustomerName, CustomerAddress, SalesRegion, and CreditCardNumber. Here are some alternatives with the respective performance pros and cons for each implementation:

- **ENCRYPTING ALL COLUMNS IN A TABLE**

  What if you encrypt all five columns? If you select from the table for a specific CustomerID, you will be forced to decrypt the CustomerID of all 1 million rows. This will result in a huge amount of overhead. When you insert into the table, the overhead is not substantial, however if you update the column based on the CustomerID, you will again be forced to decrypt the CustomerID for every single row.

- **ENCRYPTING ALL BUT ONE COLUMN IN A TABLE**

  What if instead you encrypt all columns except the CustomerID? When you select from the table for a specific customer ID, you are not required to perform any decryption until you find the actual row that matches the selection criteria. This is because the engine in the database will only look at the column in your where clause for all rows. Only when it finds a row that meets the criteria you indicated would decryption need to occur.

  The additional time required to select from the table in this query has been reduced to almost nothing. Decryption of the rows found will cause some overhead, however this overhead is minimal since the row set is relatively smaller.

  In the same case, if I selected from the table looking for a specific CustomerName, I would again be forced to decrypt all 1 million rows, resulting in a very slow query. This is because the database engine will need to decrypt the customer name for every row in order to find the rows that match the search criteria. The same would apply to Updates and Deletes. Now if I selected from the table based on the CustomerID and the CustomerName, the search would be substantially faster because decryption would only occur for rows that matched the CustomerID. Of course, if the database's query processor decided to search based on CustomerName before CustomerID, the results would be entirely different. When the query processor decides to perform the lookup on CustomerName first, the database is forced to decrypt CustomerName for every row.

- **ENCRYPTING ONLY THE CREDIT CARD NUMBER**

  If I encrypted only the credit card numbers, I would substantially reduce the chance that any query might significantly slow down the database. The only significant performance hit would be if the query processor decided that searching on the CreditCardNumber column would be the most efficient search. This would likely only occur if you were to search for a specific credit card number without providing any other search criteria.

  Before actually deciding which columns to encrypt, you should first gather a list of the most common statements executed against the database. Most large applications are highly dependent on a handful of queries. Analyzing the use and frequency of SQL statements allows you to make an informed decision on how encrypting a column will affect performance.

KEY MANAGEMENT FOR DATABASE ENCRYPTION

One of the main problems with encryption is key management. Very often we see encryption implemented by hard-coding a password into a procedure or script. Even if you use the most robust encryption algorithms and the strongest keys, this implementation of database encryption is inherently flawed.

The problem is reduced to the fact that you are relying on access control to implement security. The idea behind "defense in-depth" should not rely on one layer of security to protect another layer. In this case, once access control is circumvented, encryption is also immediately circumvented. This does little to improve the security posture of your database.

Encryption should be layered on top of access control. Encryption should protect the data when access controls are circumvented. When an operating system user is able to gain full control of the database, they can then circumvent the access controls. Encryption, if implemented properly, uses mathematics to prevent someone with full control over the system from reading data.

SOLUTIONS

Deciding to build your own security system is not a task most people should endeavor to undertake. There are a myriad of details to deal with, such as padding or dealing with NULL values that result in complications, and leaves you open to attack if implemented incorrectly. Your best bet is to find a system that provides all the features you need. While encryption will indubitably require some additional administrative work, you should find a solution that minimizes this impact.

When evaluating the possible solution, you should carefully consider each system. It is most important to understand how the system works and judging for yourself whether the solution provides "true encryption," or the solution breaks down when access controls break down. Talk to the vendor to understand the underlying architecture. If the vendor is not open with the underlying architecture, chances are they do not want you to know that something is wrong. An encryption system should not attempt to hide the implementation details by concealing how the code works. If it does, be wary.

ABOUT APPLICATION SECURITY, INC.

Application Security, Inc. is an organization dedicated to the security, defense, and protection of the application layer. Solutions are provided to proactively secure (penetration testing/vulnerability assessment), actively defend/monitor (intrusion detection), and protect (encryption) your most critical applications including database, groupware, and ERP systems. Free security presentations, white papers, and evaluation versions of products are available for download at - http://www.appsecinc.com.