Tony Morelli

IS 675

12/3/2009

DATABASE SECURITY

Hackers have the ability to attack databases and examine their contents. With a lot of our business being done over the internet, our most personal information is being stored in online databases run by companies we trust. We purchase goods, we pay our bills, and we answer medical questions all which are stored in databases. This information is personal, and could do damage if it ended up in the wrong hands. Digital information security is very important to keep secure. Unlike traditional paper documents, digital information can be changed or wiped out completely with little or no trace left behind. Because of that, people using online databases must have confidence that the data they submit must be secure and safe from malicious users. I personally have had my identity stolen and know the hardships that come with that event.

A study has shown that people fear the security risks of ordering online (3). The people in this study were not against using the internet, in fact they used the internet to look up product details and prices. However when it came time to actually making the purchase, the people went to a physical store. Privacy concerns are the largest barrier for people not participating in e-commerce shopping (4). Large databases have been compromised releasing credit card numbers, physical addresses, email address, and other personal information to the underground criminal community. People have had their identities stolen, unauthorized charges on their credit cards, and have also had their email address signed up for millions of spam messages. In these scenarios, all of the information was stolen from databases, but how were hackers able to do that? How did they gain access? What was done wrong from a security standpoint that allowed the malicious attacks? This paper will answer these questions by describing database security techniques and comparing them to infamous database attacks.

Techniques For Securing Data

Encrypting data is a good method for securing a database that might fall into unwanted hands. However, encryption can really only help when the attacker has physical access to the data. Most companies store historical data backups off site for emergency retrieval. Keeping all of your data in an online and immediately accessible format will be a waste as most of the data is not accessed which will make queries looking for recent data run slower. Because of this, archives are kept offsite just in case they need to be accessed. The offsite data should be stored in an encrypted format. When data is stored offsite, the only protection is physical protection.

Locks and security systems will prevent unwanted people from getting to the physical media, however if that security is breached the data can be restored. If the data is not encrypted, it is very easy for thieves to restore the data and find the information they are looking for. If it is encrypted, it will be a much harder task as they will need to find the encryption keys.

Encrypting live data has its benefits but it also may not be that useful. Most people who have hacked into databases have done so through knowing a password, or maliciously using an application that already has been authenticated through the database. At this point, the user level security has been breached and all data is readily accessible. However, protecting live data with encryption does have the ability to prevent an attack where the data is physically accessible to a criminal. For example, a thief that gains access to the computer storing the database could mirror the drives and take the data for offsite analysis potentially avoiding detection. If that data was encrypted the thief will have a much harder time getting the data off in an encrypted format as opposed to an unencrypted format.

Databases or data collections are now found in the pocket of most people. Cell phones and PDAs contain personal information such as calendars, address book and stored credit card numbers. If this PDA was lost or stolen, the thief could have complete access to all data. If it was not encrypted this could be a very easy task for even the rookie hacker. Any device that contains personal data and has a chance to be lost or stolen should always be encrypted to make it at the very least challenging for the malicious user to obtain your data.

Encryption also has its use for hackers that listen or snoop on the transactions between a client and the database. It is possible for a hacker to capture data sent between a client and a server, or an application and the database. If the data is not encrypted all the results of the query can be easily viewed by the hacker. These results could contain credit card information, addresses, or other personal information. The encrypted data will look like a bunch of garbage if it falls into the wrong hands. The idea of encrypting results can also be taken to the next step by also encrypting the queries. The data between the application and the database can be captured, and if only the results are encrypted, the hacker may not easily see any personal information, however the hacker could gain enough information to easily hack into the database. If the query itself is left unencrypted the database structure and potentially logins/passwords could be obtained and additionally used for unethical purposes.

One technique that online merchants can use to prevent unauthorized use of credit cards is to use a technique known as IP2Location. That is it maps the purchasers ip address to a localized geographic location and makes sure the billing address is in that range. If it is not within in that range, extra steps could be taken to ensure the purchaser is legitimate. For example, a thief could steal/purchase the details relating to a specific account and know what the billing address is. As he fills out the purchase form and sets the billing address to the one on file with the credit card, and the shipping address to his, the system could do a quick check and see if the billing location matches the results of the geographic location determined by the ip address of his computer. If

they are different, the online merchant could require further communication such as a phone call or other validation method to ensure the purchase is not fraudulent.

Password processes can also secure a database. Password rules should be enforced such that passwords are difficult to guess and even harder to brute force. Recommended password restrictions are passwords longer than 10 characters, mixed upper and lower case, contain non alphanumeric characters and do not contain a dictionary word. Requirements should be in place to enforce the changing of passwords. Passwords should not remain constant for more than 90 days to remain secure. This limits the amount of time a hacker can attempt to guess or brute force the password. It is also recommended to keep track of the last 5 passwords used and not let a user re-use one of those. This will prevent a user from changing his password to the existing one. Users are creatures of habit(8), and an aware hacker will note this. Password patterns can be determined and are never a good idea.

Another issue with passwords are default passwords. As a database admin, it is important to change any default or empty passwords for system database user accounts after installation. Each database management system is different, but special care should be taken to identify any of these defaults for the particular system. Hackers are already aware of all of these defaults and will attempt to use these first.

User restrictions are also important. Although it varies from system to system, super user accounts should be setup such that they are accessible only from the local machine. This adds a second level of security other than the super user password. The person desiring to use the password not only needs to know what it is, but has to gain physical access to the database server. This also prevents applications from logging in with the super user password when they are run on separate machines. If this is not done, an attack such as SQL injection can be disastrous. Also, the ability to set restrictions must be set. Only certain users should have access to certain parts of the database. Only highly trusted users should have the ability to create or delete tables, and maybe only the database administrator should have the ability to set these restrictions.

A more in depth version of user restrictions involves a complex permissions table. Once a query is submitted to be run, the query is first validated to ensure it is a proper query for the user who wants to run it. The system first checks to see if the user who issued the query is in the set of authorized users. Next, the system checks to see that the fields requested in the query are in the permitted fields list for that user for that query. And finally after the query is run, the returned recordset is checked against the permissions table to be sure the user who initiated the initial query is permitted to see the resulting dataset. These many checks will ensure from beginning to end that the query is not malicious and that the returned data is OK for the user to see. However, setting up the permissions table can be a complex and tedious task.

The physical connectivity of the database server is also important in fighting attacks. The database server should never be directly connected to the internet. All access should be through an access point which will limit the amount of ways a hacker has access to the data. This is similar to physical access to a secure building. If you have 1000 doors it is difficult to monitor all 1000 of them, however if your building has only one door you could focus your security efforts on that one door. In the database world, if you put your database directly connected to the internet it's the same as having a building with 1000 doors – it is very difficult to secure all of them. If you place your web server directly connected to the internet, and your database server on your local network, all malicious attacks must first pass through the web server in order to access the database server. So in effect you have only one path to get access to the database and can watch that path closely.

When setting up a database it is also important to set up correct access logging. Most database management systems allow login successes, failures, or both. At a minimum, failures must be logged. In addition to logging the failures it is important that the logs be examined on a regular basis, or that alerts be sent when failure thresholds are exceeded for the number of failures for a particular account. Also, login success could be a sign of malicious behavior as well. If a particular account is being accessed by several unique computers at the same time, it might be an indication that a username and password have gotten out into the wild. So it is best to log both successes and failures, but the logs need to be filtered and looked at on a regular basis with automatic parsing immediately highlighting any suspicious behavior.

Also, as a database administrator it is very important to be running the latest version of the database server software. Software updates frequently are released and it is a good idea to keep up with them. As the software manufacturers discover vulnerabilities in their products, they will release patches to hide close the holes. With each patch, hackers become aware of issues in software that has not been updates, and it makes it that much easier for them to attack a server with old software. If the latest software is running, at least all the widely known issues are covered.

Another good idea as a database administrator is to protect old data. If a new database server is installed, make sure the existing one is completely taken down. Unused database servers that remain running will likely not be kept up with the latest security patches and area a good target for hackers. Even though they may not contain the latest live data, they could contain clues or passwords to assist hackers in cracking the live server. Also, the data that is living on the old server could contain important information such as credit card numbers or social security numbers which is still valuable to a hacker.

Online transactions between merchants and credit card validation services rely on trust for security. In this case, there are two parties involved in the data transmission, the store and the credit card company. Each part in the transaction can have some knowledge of the other part and may have access to the other party's systems. The trust aspect is that even though one or both of

the parties could act maliciously they would not because there would be no gain. The trust would be lost and the partnership would be lost.

Bertino et al (6) suggest secure databases must meet 3 requirements - secrecy, integrity, and availability. Secrecy refers to the protection of data against unauthorized disclosure. Integrity refers to prevention of unauthorized and improper data modification. Availability refers to the data being reliable even under malicious access denials making the database system unavailable or unreliable from a performance standpoint.

Another technique for securing data is to limit dynamic requests for data. In this scenario, the data is not requested on demand but published at regular intervals. A client can subscribe to receive updates, and the information contained in the database can be pulled by a central entity and then distributed to the clients who have made a request for the data. This completely separates the data from any malicious attacks. Of course, security measures must be taken such that subscriptions are only handed out to those who should be receiving the data. Another issue with this is that the data may not be available on demand. This could be resolved by the client making a request which will be serviced at some point in the future after a thorough validation of the client's identity, instead of a simple password validation followed by an immediate response.

A recent study has proposed a security solution in which the results are filtered if the proper permissions are not available to the user (7). For example, a user may have permissions to perform a select statement and the security mechanism allows the select to be performed. However, the user may have joined in tables that he does not have permissions to access. Instead of denying the query, the query is allowed to run, and all results are passed back to the security mechanism. At this point, it is the responsibility of the security mechanism to analyze the data and decide which data the user has access to view. Only data the user has access to view will be passed back to the user. This has several advantages. First, there is an intermediate step between the user and the database, so the user never has direct access to the database. Secondly, a dedicated employee can maintain this security mechanism which will alleviate some of the burden of the database team. And finally, since the database system itself is not responsible for performing the security sanity checks, all queries will run much faster. This could leave the data isolated, and several security mechanisms could be placed between different users and the database pushing much of the load to the distributed set of security mechanisms instead of the database. The database would do exactly what it does best - store data.

The timestamp of the data must also be considered when looking to secure data, known as Temporal Data (9). For example, a user may have access to certain data about an object based on its current activity. The human resources personnel could have access to a CIA agents activities when he is in the office, but when he is out on a mission, that information is classified and the HR personnel has no business knowing what he is doing at that moment. In addition, the HR personnel should always have access to the non-secret activities no matter what the current state of the employee is. Pissinou et al go into very complex examples of how this is relevant, but

essentially the concept describes the fact that a users restrictions cannot be locked down to what is taking place right now. The ability to review historical data and the permissions for which data the user has access to should be determined by the permissions available at the time the data was inserted, not based on the permissions of that data today. I think this is a very important concept to grasp because it is very easy for data to fall into the wrong hands if the security does not take that into consideration.

Methods of Attacking a Database

One of the most common ways of hacking into publicly available web based databases is through a technique known as *SQL Injection*. This is a method where websites are left unprotected and as a result an otherwise secure database is vulnerable to attacks. In general, SQL Injection works by using a web form where a user is required to enter information such as a name, or credit card number, and instead of entering the desired information, SQL commands are entered and an attacker can gain access to the entire database.

Using our Employee table created in class as a sample dataset, and a fictitious website that would be used to view data from the database, the process of SQL injection will be outlined. The employee table contains 6 fields – empID, EmpLastName, EmpFirstName, EmpEmail, EmpPhone, and EmpMgrID. On a web page the user is instructed to give his email address to pull up his data. Under normal circumstances the user would also provide a password, however this table does not contain a password field and it is not necessary to illustrate the point. The code from the web page might be something like this:

"select * from employee where EmpEmail = '" + useremailfromwebsite + "'";

Using this statement as an example, the output can vary based on what the user enters. For example, if the user types in *SFETTERS@yahoo.com*, which is a valid entry in our table, the row for that particular email will be returned and displayed to the user. If the user entered *hhh@aaabbbccc.com* , which is an entry that is not contained in the table, the user would see an error indicating the email address was not found. Now lets see what happens if the user is actually a hacker and enters *a' or 'b'='b*. Now when this string is used to build our query above, the actual query will look something like this:

"select * from employee where EmpEmail = 'a' or 'b'='b'";

When executed, this will return all rows from our Employee table. And if the website is structured to display all results, and our employee table contained a password field the hacker now would not only have all information about all employees but also the password for each employee such that he could login and assume the identity of any employee. Since the

evaluation of' *'b'='b'* is always true, this statement will always return all rows. Alternatively, if this query was used to actually *login* to the website, and the web software only checked to see if at least 1 row was returned, the hacker has now logged into the website and can now start making purchases under an account that belongs to someone else.

How about not only a hacker, but a malicious user who wishes to destroy data. He could enter the following for the email address *a';drop table Employee;select * from employee where EmpEmail = 'a.* If this was allowed to be inserted into our SQL query string, the hacker could have just wiped out all our data, assuming the database user account the web page logged in with had the permissions to do so.

The previous examples show how a hacker can take control of a SQL Injection vulnerable database, but some of them depended on the fact that the hacker knew something about the database structure. In order to drop the table Employee, the hacker must have first known that a table employee existed. Using SQL Injection it is also possible to find different table names. On approach to finding the database structure is to use a sub query to guess the table names. For example, entering *a' and 1=select count(*) from users;*-- will give different results depending on if a users table actually exists. Also note in this example, the "--" escape sequence was used to tell the SQL interpreter to ignore anything else that may have been added to the query by the website. This is another way hackers can use SQL Injection to their advantage.

These examples have shown how a user can gain access and use select and drop statements. Once the access has been gained, a hacker can use any statements that the web database user has access to. For example, insert and update statements can be used to create/modify existing accounts. Using SQL Injection in conjunction with database specific vulnerabilities, a hacker can also take over the host database computer and wreak all kinds of havoc. Different server software has different capabilities, but take MS SQL Server for example. This database package supports the command xp_cmdshell which allows commands to act on the computer hosting the database package. These act on the computer itself, and not on the database which allows the hacker to have complete control over the host machine.

The capabilities of a hacker using SQL Injection are very scary, however there are ways to prevent a SQL Injection attack. By following the methods in the following sections the database will be more secure, but by no means are they a guarantee of a secure database. Hackers are always finding loopholes.

One method to prevent SQL Injection is to validate the data the user entered. Prior to building and executing the SQL Query string, parse the user input and look for suspicious characters. This is the section where the database is most vulnerable because it is where the user has direct access through a public website to act maliciously. Look through the data and pick out characters that could indicate problems. For example, if the user was asked to enter a username and password and one of the entries contained spaces, which goes against the requirements when

creating a username and password, stop there and prompt the user to re-enter the information. The same goes for the sing quote (‘) character. Eliminate this character from the valid character set when creating usernames and passwords and then it can be checked right away when an attacker is entering data. Analyze all potential control characters and SQL commands to make sure they are not being used incorrectly. The most strict policy would be to not allow any SQL commands as passwords or as part of passwords and then during validation these can be checked and thrown out before an actual SQL statement is run.

Another method to reduce the effects of SQL Injection attacks is to use views instead of tables. If the attacker can figure out table names based on his attack, it is much better if the table he has discovered is actually a view. Although this will not do very much for the viewing of data, the underlying structure of the database has remained hidden. Attacks such as using the DROP command, or insert or update will be of no use to the attacker as they server no purpose in the world of views.

A very important security measure can be taken to lock out the web database user from allowing a SQL injection attack to ruin a database. For example, it is probably not necessary for the database account used by the web server to need to drop tables. So remove the ability to drop tables from that user. It might even be better to have different web based accounts when accessing the database. One that has read only access to everything and will be used to validate logins and passwords, which is the most common pathway SQL Injection hackers like to take, and another that can have permissions to update (only used once a user has actually logged in). Another user could also be created that had insert permissions in a very limited scope to add new users.

Another very good solution to avoid SQL Injection attacks is called *SQLrand*. This was a study done at Penn State University and is not a standard way of avoiding SQL Injection attacks, but it is a very good solution. In this method, the webserver contains a key. All SQL statements such as insert, select, drop, group, are appended with a key. The SQL statement is then forwarded to a validation proxy prior to being sent to the database. The validation proxy verifies that the statement is built correctly and with the correct keys. For example a typical statement prior to verification with a key of 123 might look like the following:

select123 * from123 user where123 userid=1

The proxy also knows the key and will decode the SQL statement using the key into the format that the database server expects. This method makes it very difficult for an attacker to structure information entered in the correct format. The attacker would have to know that this method was being used and what the key was to have any success. The proxy would throw out any queries with incorrect or missing keys.

Successful Database Attacks

SQL Injection is the most common form of attack. Although it seems simple, it has been used very recently in the largest database attack known to have taken place so far. On August 19,2009 Albert Gonzales was indicted for conspiring to hack into computer networks and stealing data relating to more than 130 MILLION credit and debit cards. This indictment is the largest alleged credit and debit card breach ever charged in the Unites States. The corporate victims in this indictment include Heartland Payment Systems, 7-Eleven and Hannaford Brothers Co. Inc.

Not only did Gonzales and his partners gain access to the card information, but they also sold the information on a website. In addition to selling information relating to credit cards, they also had access and distributed information pertaining to identification. Stolen drivers license numbers, social security numbers, names, street addresses, and email addresses were all sold to the highest bidder. They also used the information they obtained for their own benefit.

Court documents outline the methods used by Gonzales and his cohorts. The method typically began by them looking for unsecure corporate wireless networks. Once their computer was on the network, they would then use SQL injection to obtain access to the card processing databases. This opened a door to the computer, and also according to court documents, they then installed malicious software on the database computer which opened a back door for them to enter as they needed. Although the court documents did not go into specific detail, a method such as the *xp_cmdshell* command described earlier in this paper could have been used to execute whatever program they needed to have running. Also, prior to installing their custom malware program, they ran at least 20 different anti-virus programs against their software to ensure they would not be detected.

Although they attacked several companies such as Marshalls, TJ Maxx, and 7-Eleven directly, the attack on Heartland Payment Systems potentially caused the most damage. Heartland is one of the largest payment processing companies in the US and processes payments for over 250,000 businesses. Gonzales and his buddies using SQL Injection as an access method were able to place malicious software on Heartland's systems that could sniff transactions, including all data contained on the magnetic stripe of a credit card, as they were sent from businesses to Heartland for validation. This software ran undetected for some time and could have taken some or all of the information used in its 100 million transactions performed each month. Reports of fraudulent activity on many different cards started appearing in 2008 with all having the similar trait of transactions performed through Heartland. Heartland called the US Secret Service and hired two breach forensics teams to investigate their systems, and only at that point was the malicious piece of software discovered.

Outside of hackers, another concern for database security is employees. In 2005, Timothy Curly, was employed at American Express as a Database Analyst. During his stint he had access to all kinds of transaction and account data and stole information on thousands of credit cards for his

personal benefit. Although the database could be secure from all types of hacking attempts it is scary to think that an inside job at a trusted company could destroy your credit.

With all that has happened as far as credit card theft goes, a standard, *Payment Card Industry Data Security Standard*, has been developed in order to ease any concerns that companies dealing with payment transactions are not secure. The current version of the standard is 1.2 and it specifies 12 requirements for compliance placed into 6 logically related control objectives:

Build and Maintain a Secure Network

Requirement 1: Install and maintain a firewall configuration to protect cardholder data
Requirement 2: Do not use vendor-supplied defaults for system passwords and other security parameters

Protect Cardholder Data

Requirement 3: Protect stored cardholder data
Requirement 4: Encrypt transmission of cardholder data across open, public networks

Maintain a Vulnerability Management Program

Requirement 5: Use and regularly update anti-virus software
Requirement 6: Develop and maintain secure systems and applications

Implement Strong Access Control Measures

Requirement 7: Restrict access to cardholder data by business need-to-know
Requirement 8: Assign a unique ID to each person with computer access
Requirement 9: Restrict physical access to cardholder data

Regularly Monitor and Test Networks

Requirement 10: Track and monitor all access to network resources and cardholder data
Requirement 11: Regularly test security systems and processes

Maintain an Information Security Policy

Requirement 12: Maintain a policy that addresses information security

Database Security in the Workplace

With the threat of security risks very high, many companies have taken the path of hiring a Chief Information Security Officer (CISO) who is responsible for the security of all data owned by the company. The FBI reported that companies had a 90% chance of having a computer or network security breach in 2009. In 2007 security breaches averaged $14 million per incident and also caused those companies to lose 2.6% of their customers (1). The role of a CISO is primarily a business professional who can integrate security skills. Although a very educated person might

accept the role of a CISO, the education must continue. New attack strategies or software vulnerabilities can come up at any time, and a good CISO will continue with education in order to stay on top of all the technology changes.

The role of the CISO is also to keep a good public image. It has been shown that internet privacy concerns can have an effect on a person's willingness to participate in an electronic market and disclose personal information.(2). This makes it even more important to not only have a secure business, but to also emphasize the fact. If a new web based store front is created, it is important to not only have good products, but to also show that the site is secure. If this is not done, new customers may shy away due to the fact they are unsure if the data they enter will be secure or not.

Conclusion

This paper has demonstrated that database attacks are a very real and current problem. Many millions of credit cards have recently been stolen in the largest known credit card security breach. There are many different ways to secure a database, including encryption, user permissions, software updates, and physical separation. Companies now are employing a Chief Information Security Officer to maintain the data security for the company. With all these ways to ensure the safety of data, online transactions still contain some amount of risk.

1. The Chief Information Security Officer: Analysis of the Skills Required For Success;Journal Of Computer Information Systems Spring 2008 Vol 48 Issue 3pages 15-19)

2. Ba, S. L., & Pavlou, P. A. 2002. Evidence of the effect of trust building technology in electronic markets: Price premiums and buyer behavior. ***MIS Quarterly***, 26(3): 243-268.
3. Barua, A., Konana, P., Whinston, A. B., & Yin, F. 2001. Driving E-business excellence. ***Mit Sloan Management Review***, 43(1): 36-+.
4. Wang, Y. D., & Emurian, H. H. 2005. An overview of online trust: Concepts, elements, and implications. ***Computers in Human Behavior***, 21(1): 105-125.
5. Maurer, Ueli 2004 The Role of Cryptography in Database Security. ACM SIGMOD 1-58113-859-8/04/06
6. Bertino, Elisa. Sandhu, Ravi. Database Security - Concepts, Approaches, and Challenges. IEEE Transactions on Dependable and Secure Computing Vol2 No. 1 January-March 2005
7. Pan, Leon. Using Criterion-Based Access Control for Multilevel Database Security. International Journal of Information Technology Application 1:1(2008) 10-16
8. Skovra, Robert. Framing the Corporate Security Problem: The Ecology of Security. Issues in Informing Science and Information Technology Volume 4 2007 45-52
9. Pissinou Niki., Makki, kia., Park, E.K. Towards a Framework for Integrating Multilevel Secure Models and Temporal Data Models. CIKM 94 ACM 280-287

5.
6.

http://en.wikipedia.org/wiki/Payment_Card_Industry_Data_Security_Standard

$(ISC)^2$

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.4549&rep=rep1&type=pdf

Append key to sql key words such that an attacker could not guess the key

Hackers:

CardSystems Solutions http://www.ftc.gov/opa/2006/02/cardsystems_r.shtm SQL Injection

Heartland Payment Systems (Albert Gonzales) http://www.justice.gov/opa/pr/2009/August/09-crm-810.html SQL Injection

TJX Companies. (Albert Gonzales)

Laws:

http://en.wikipedia.org/wiki/Payment_Card_Industry_Data_Security_Standard

$(ISC)^2$

Identity theft advice: http://www.ftc.gov/bcp/edu/microsites/idtheft/

Generic sites:
http://www.databreaches.net/?p=8248