2006-343: SQL INJECTION ATTACKS AND PREVENTION TECHNIQUES

Mario Garcia, Texas A&M University-Corpus Christi

SQL Injection Attacks and Prevention Techniques

Abstract

Databases introduce a number of unique security requirements for their users and administrators. On one hand, databases are designed to promote open and flexible access to data. On the other hand, it's this same open access that makes databases vulnerable to many kinds of malicious activity [1]. One of the main issues faced by database security professionals is avoiding inference capabilities. Structured Query Language (SQL) injection is a technique used to take advantage of non-validated input vulnerabilities to pass SQL commands through a Web application for execution by a backend database. Attackers take advantage of the fact that programmers often chain together SQL commands with user-provided parameters, and can therefore embed SQL commands inside these parameters. The result is that the attacker can execute arbitrary SQL queries and/or commands on the backend database server through the Web application. In this report we discuss the different SQL injection attacks and prevention techniques available.

Introduction

Within the past decade, the growth of the Database industry and the Internet has revolutionized the way many people interact with information. This rapid proliferation and the cost effectiveness of new key technologies are creating large opportunities for developing large-scale distributed applications. These systems are made up of several interacting components, each of which is pretty much well encapsulated. However, this phenomenal growth has also brought about security concerns since some of the data now being made available on the Internet is sensitive. For example eCommerce, the leading Web-based application is projected to exceeding $1 trillion over the next several years. The strong need for information security is attributed to several factors, including the availability of sensitive information stored in corporations and governments databases to the outside world.

Database Access Control Models

Access control models were developed to primarily address the issues of data availability, secrecy, and confidentiality. These models can be classified as either traditional or recent. Traditional access control models are broadly categorized as discretionary access control (DAC) and mandatory access control (MAC) models. Newer models comprise mechanisms such as role-based access control (RBAC) or task-based access control (TBAC). These mechanisms address the security requirements of a wider range of applications.

Discretionary Access Control (DAC) Model

Discretionary access control is based on the concept of access rights (or privileges) to data objects, and mechanisms (such as SQL GRANT and REVOKE statements) for giving subjects such privileges. A privilege allows a subject to access some data object in a certain manner (e.g., reading and writing the data). All the subjects and objects in the system are enumerated and the access authorization rules for each subject and object are specified. Subjects can be users,

groups, or processes that act on behalf of other subjects. If a subject owns an object, the subject is authorized to grant or revoke access rights on the object to other subjects at his or her discretion. DAC policies are flexible and the most widely used for Web-based applications. However, these policies do not provide high security assurance. For example, DAC allows copying of data from one object to another, which can result in allowing access to a copy of data to a user who does not have access to the original data.

Mandatory Access Control (MAC) Model

Mandatory access control is based on system-wide policies that cannot be changed by individual users. In this model, each data object is assigned a security class, and each user is assigned clearance for a security class, and rules are imposed on reading and writing of database objects by users. An important goal of this model is to control information flow in order to ensure confidentiality and integrity of the information, which is not addressed by DAC models. For example, to ensure information confidentiality in Defense applications, a MAC model can be implemented using a multilevel security mechanism that uses no read-up and no write-down rules, also known as Bell-LaPadula restrictions[2]. These rules are designed to ensure that information does not flow from a higher sensitivity level to a lower sensitivity level. To achieve information integrity, the access rules are formulated as no-read-down and no-write-up. The goal in this case is not to allow the flow of low integrity information to high integrity objects.

Unlike the DAC model, the MAC model provides more robust protection mechanisms for data, and deals with more specific security requirements, such as information flow control policy. However, enforcement of MAC policies is often a difficult task, and in particular for Web-based applications, they do not provide viable solutions because they lack adequate flexibility.

Role-based Access Control (RBAC) Model

Role-based access control (RBAC) models are receiving increasing attention as a generalized approach to access control because they provide several well-recognized advantages over more traditional Access Control Lists (ACL). They have been discussed in details in and in and have, in fact, been formalized by NIST. Since roles are derived from a user's responsibilities and functions within an organization, a role-based model directly supports arbitrary, organization-specific security policies. ACL's on the contrary are tied to particular objects and their maintenance can turn out to be a nightmare for system administrators.

Another advantage of RBAC models is their independence from set policies (i.e., they are policy-neutral) in the sense that using role hierarchies and constraints, a wide range of security policies can be implemented, including traditional DAC and MAC, as well as user-specific ones. Administration is also greatly simplified by the use of roles to organize access privileges. For example, if a user moves to a new function within the organization, the user can simply be assigned to the new role and removed from the old one, whereas in the absence of an RBAC model, the user's old privileges would have to be individually revoked, and new privileges would have to be granted.

Other Access Control Models

Other newer and lesser known models include models for Tasks and Workflows for task-intensive and workflow-based systems (typically, most heavily used Web servers fall into this category). These, unlike the previous models, which are all based on the subject-object paradigm, are tailored specifically for content-based information and security policies for task and transaction intensive Web servers. To address the security issues related to task-oriented systems and to effectively serve the unique needs of such systems, researchers in propose a family of task-based access control (TBAC) models that constitutes four models arranged in the form of a hierarchy. The TBAC0 model represents the base model that provides the basic or the minimum facilities, such as tasks, authorization steps, and their dependencies. The TBAC1 model is an extension of TBAC0 that includes the composite authorizations of two or more authorization steps. The TBAC2 model is another extension of TBAC0 that allows both static and dynamic constraints. The TBAC3 model is a consolidated model that has features of both the TBAC1 and TBAC2 models.

The second type includes Agent-based approaches. An *agent* is basically a process that can be characterized by adaptation, cooperation, autonomy, and mobility. This paradigm can be effectively used to provide security features for Web applications. Some agent communication language can be used to negotiate policies during conflicts for secure interoperation among participating policy domains. Agents can be assigned security enforcement tasks at the servers and client machines. Although mobility and adaptability are essential to the efficient use of Internet resources, they pose several security threats. For example, an agent can engage in malicious behavior, thus disrupting normal operation of the host. Similarly, a host may be able to affect the activity of an agent by denying required access to local information resources.

The last type is known as PKI's or Public-Key Infrastructures. This technology is maturing, and the use of PKI certificates is expected to be ubiquitous in the near future. Certificates issued by a PKI facility can be used for enforcing access control in the Web environment. An example is the use of an extended X.509 certificate that carries role information about a user. A certification authority, which acts as a trust center in the global Web environment, issues these certificates .The use of public-key certificates is suitable for simple applications. These techniques can be used to either support a host's access control method by carrying access control information or provide a separate access control mechanism based on trust centers.

Database Attacks

Threats on database security can be grouped into two different categories, physical and logical. Physical threats consist of (but are not limited to) forced disclosure of passwords, destruction of storage devices, power failures, and theft. The most common way to prevent this type of threat is limit the access to the storage devices and put backup and recover procedures in place. Logical threats are unauthorized logical access to information. This is usually through software. Logical threats can result in denial of service, disclosure of information, and modification of data. [3]

Insider Threat

One of the largest threats to a database is a corrupt authorized user .This user can legitimately access confidential information. This information can then be leaked electronically or by some other means such as printout or by word of mouth. There is very little that can be done to prevent this from within the database management system. Mandatory access controls can help a little bit by not allowing a user logged in with classified access to save or copy the data to a location with unclassified access. This type of threat is usually handled by limiting the number of users with that level of access and other complicated procedures.

Login Attacks

Another way to compromise a database is to successfully log in as a legitimate user. This can be done by physically stealing the information or monitoring network traffic for login information. Another attack could involve accessing password lists stored in an operating system. And of course, login information can only be as secure as the password used. If it is easy to crack, there is not much that can be done. Restrictions on the type and form of passwords can help, but does not solve the problem. The database must employ authentication and encryption to ensure that this type of attack is less likely. The web server could be set up to either pass the user authentication information directly to the database or authenticate the user and then use the web server's own authentication information to login to the database. The latter method provides an optimization in that the connection can be cached. This results in an even more vulnerable system because if the web server is compromised then the database is also compromised. Protecting the user's authentication information is important. In general, encryption or a one-time password system could be used [4].

Network Attacks

There are a multitude of possible attacks on a database if it is accessible over a network, even more so if that network is the Internet. A number of precautions can be put in place such as a firewall to protect the database and possibly the Web server .The data sent over the network can be secured by a number of means. A common method on the Internet is the secure socket layer. This would prevent an attacker from just gathering information by watching network traffic. A good method for authenticating with the database will also be necessary. Certificates can also be used in conjunction with databases to ensure authentication. An especially common attack has been the denial of service attack. This type of attack is related more to the Web server allowing access to the database, but can also be mounted against a database itself.

Inference Control

The users of a database can use information that they have access to and possibly some other supplementary (external) information to infer information that they do not have access to. In more explicit terms, data at a high security level can be inferred from data at a lower security level. This can be a very difficult threat to prevent. This threat is usually associated with statistical databases. Information about individuals can be inferred from answers to allowed statistical queries on the data. A naive approach would be to move the lower level data to a higher level. Only the minimum amount of the lower level data needed to prevent the inference should be moved to the higher security level. Although, this usually results in much if not all of

the lower level data being reclassified at a higher security level. This solution is usually not acceptable. Other techniques can be used such as query restriction, data perturbation, and output perturbation. Query restriction involves requiring a minimum number of rows to be part of the query. This does not really solve the problem but increases the number of queries necessary to infer any confidential information. A common solution is to audit such query patterns in hopes that such activity will be detected before the confidential information is compromised [5.]

Trojan Horses

Trojan horses are corrupt software applications that leak confidential information. These applications are part of the normal use of a system, but have been modified to copy or send sensitive information to unauthorized locations or users. An application that has a Trojan horse must be installed on the system. This could be done by the attacker or by an administrator that did not realize that there was a Trojan horse in the application. The corrupted application will operate as expected for all practical purposes. But it will be doing some additional illegal functions as well. [6.]

Types of Countermeasures

There are several Countermeasures for database security attacks. They are briefly described below:

A view is a description of how data is to be retrieved from the underlying tables. It does not store data but it is treated as though if it were a table in SQL statements. Views are used to perform the following actions:

- Limit the rows accessible to a user (row-level security)
- Limit specific columns accessible to a user (column-level security)
- Pre-join several tables (removing the requirement that the user understand the complexity of joining tables)

Stored Procedures

A stored procedure is a program written in SQL that is stored in a compiled form within the database. Stored procedures can optionally take arguments and/or return values. They can be used to perform table updates as well as retrieve data from tables. Table triggers, applications, or users can execute stored procedures. Stored procedures are usually written and owned by the DBA. It is important to understand the relationship between the owner of a stored procedure and the user who executes the procedure. In general, stored procedures execute on behalf of the user, but use the privileges of the creator. The DBA should own all stored procedures [7], [8].

Developing an Audit Plan

Database auditing is the monitoring and recording of activities occurring within a database. Usually there are two main reasons for auditing:

- Security auditing – to determine if someone is attempting to break into your system
- Performance auditing – to determine why the system is slow

When it is determined that auditing is needed, the next step is to decide where the audit information will be stored. The operating system usually can support an audit trail stored outside the database.

Triggers

Triggers are stored programs that are associated with a table. A trigger is executed when the event on which it is based occurs. The events that will "fire" a trigger are commands that perform INSERT, UPDATE, or DELETE actions.

Backing Up and Recovering the Database.

Even with rigorous security, a company can be vulnerable to data loss. One of the best approaches a company could follow is to have a backup procedure in place to help ensure that the system can be successfully recovered. There are several different forms of backups that can be performed to ensure the recovery of data to a database. They are:

- Cold backups – performed with the database shut down
- Hot backups – performed with the database up
- Logical backups – exports with the database up

Hot and cold backups make copies of database files and store them to another disk on the system or to tape. When certain database files become damaged, they can be replaced by the copies made during the backup. A cold backup, also known as image backup, is performed when the database is down and there are no active processes within the database. The backup process, referred to as a file-level copy, then copies each database file either to another disk or to tape. In the event of data loss, the database can be recovered to the condition it was in when the copy was made. When performed in conjunction with archive logging, the database can be recovered up to a "point in time". The disadvantage of file-level copying is that it is very difficult to recover an individual table easily.

A hot backup is performed while the database remains up and running. Essentially, it performs the same actions as a cold backup, providing recovery up to a "point in time". As with cold backups, individual tables cannot easily be recovered. Logical backups (Exports) enable to capture all of the objects owned by a particular user, or a specific table or set of tables. Unlike a hot or cold backup, which captures complete data files, an export captures random pieces of the database or a complete copy of the entire database to one compressed file. Export files allow moving or copying a database from one system to another.

SQL Injection Attacks

A database application is a program that provides clients with access to data. There are many variations of this type of application, ranging from the expensive enterprise-level Microsoft SQL Server to the free and open source mySQL. Regardless of the flavor, most database server applications have several things in common.

Database applications use the same general programming language known as SQL. This language, also known as a fourth-level language due to its simplistic syntax, is at the core of how a client communicates its requests to the server. Using SQL in its simplest form, a programmer can select, add, update, and delete information in a database.

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks an incredible number of systems on the internet are susceptible to this form of attack. Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can be almost totally prevented. In this report we would be discussing various examples of SQL Injection Attacks and methods of preventing them [5,6,7,8,9,10].

Example 1 of SQL Injection Attacks

Microsoft SQL Server has its own dialect of SQL, which is called Transact SQL, or TSQL for short. The power of TSQL can be exploited in a number of ways to show how SQL injection attacks work. Consider the following query, which is based on the users table

```
Select username from users where username='' having 1=1
```

A system administrator can easily make a login.asp page to query the database using the following login credentials:

```
Username: ' having 1=1 --
Password: [Anything]
```

This SQL query causes ASP to spit the following error to the browser:

```
Microsoft OLE DB Provider for SQL Server (0x80040E14)
Column 'users.userName' is invalid in the select list because it is not contained in an
aggregate function and there is no GROUP BY clause./login.asp
```

Now it appears that this error message now tells the unauthorized user the name of one field from the database being validated: users.userName. Using the name of this field, it is possible to use SQL Server's LIKE keyword to login with the following credentials:

```
Username: ' or users.userName like 'a%' --
Password: [Anything]
```

Once again, this results in an injected SQL query being performed against our users table:

```
select userName from users where userName='' or users.userName like 'a%' --' and
userPass=''
```

When the users table was created, a user whose userName field was admin and userPass field was wwz04ff was also created. Logging in with the username and password shown above uses SQL's like keyword to get the username, the query grabs the userName field of the first row whose userName field starts with a, which in this case is admin.

Example 2

SQL Server amongst other databases separates queries with a semi-colon. The use of a semi-colon allows multiple queries to be submitted as one batch and executed sequentially, for example:

```
select 1; select 1+2; select 1+3;
```

This would return three record sets. The first would contain the value 1, the second the value 3, and the third the value 4, etc. So, if we logged in with the following credentials:

```
Username: ' or 1=1; drop table users; --
Password: [Anything]
```

then, the query would execute in two parts. Firstly, it would select the userName field for all rows in the users table. Secondly, it would delete the users table, meaning that in the next login the following error will be observed:

```
Microsoft OLE DB Provider for SQL Server (0x80040E37)
Invalid object name 'users'. /login.asp, line 16
```

Example 3

Many web sites use the default system account (sa) user when logging into SQL Server from their ASP scripts or applications. By default, this user has access to all commands and can delete, rename, and add databases, tables, triggers, etc.

One of SQL Server's most powerful commands is SHUTDOWN WITH NOWAIT, which causes SQL Server to shutdown, immediately stopping the Windows service. To restart SQL server after this command is issued, you need to use the SQL service manager or some other method of restarting SQL server.

Once again, this command can be exploited by looking at the login example:

```
Username: '; shutdown with nowait; --
Password: [Anything]
```

This would make our login.asp script run the following query:

select userName from users where userName=''; shutdown with nowait; --' and userPass=''

If the user is setup as the default account, or the user has the required privileges, then SQL server will shut down and will require a startup before it will function again.

SQL Server also includes several extended stored procedures, which are basically special C++ DLL's that can contain powerful C/C++ code to manipulate the server, read directories and the registries, delete files, run the command prompt, etc. All extended stored procedures exist under the master database and are prefixed with "xp_".

There are several extended stored procedures that can cause permanent damage to a system, and using our login form with an injected command as the username, like this:

Username: '; exec master..xp_xxx; --
Password: [Anything]

This can be executed as an extended stored procedure. All that is needed is to pick the appropriate extended stored procedure and replace xp_xxx with its name in the sample above. For example, if IIS was installed on the same machine as SQL Server (which is typical for small one/two man setups), then the administrator could restart it by using the xp_cmdshell extended stored procedure (which executes a command string as an operating-system command) and IIS reset. All that is needed is to enter the following user credentials into the getlogin.asp page:

Username: '; exec master..xp_cmdshell 'iisreset'; --
Password: [Anything]

Which would send the following query to SQL Server:

select userName from users where userName=''; exec master..xp_cmdshell 'iisreset'; --' and userPass=''

SQL Injection Prevention Techniques

The first thing that should be done in all database servers is to assign a strong password to the DBA account. Second, the DBA should create user accounts and assign these users to specific activities or databases. For example, if a user account has been set up for the Users database, which didn't have access to the Master database, any attempt at using extended stored procedures (such as xp_cmdshell) will not work.

Any user-entered variable should be stripped of several key characters that are required for SQL injection. This includes the following

' , / \ * & ( ) $ % ^ @ ~ ´ ? .

This could include more depending on the needs of the script. By adding the following to the previously listed script, any attempt at SQL injection would be made impossible:

```
username = replace (username, ', '')
username = replace (username, ";", "")
```

This would remove all single quotes and semicolons from the query string and would have turned the inject string into the following:

```
"Seth update table name set password=hacker where username=seth"
```

This string would have been interpreted as garbage by the SQL server and would have been rejected, thus stopping the SQL injection attack. Ensure that all accounts have strong passwords. There is no excuse for a DBA account to have a blank password. Even if the server is for testing purposes, if a hacker can access that server, he can also access any other computer connected to the SQL server's local network.

Finally, some changes can be made to the registry and SQL server configurations that will tighten security by removing or limiting extended procedures and other rarely used functions. This includes removing the xp_cmdshell stored procedure, or at least renaming it. In addition, REGEDIT can be used to adjust the value of the following to 1:

```
 HKEY_LOCAL_MACHINES\Software\Microsoft\Microsoft SQL Server\
<Instance Name>\Providers\DisallowAdhocAccess to 1
Or if using the default,
HKEY_LOCAL_MACHINES\Software\Microsoft\MSSQLServer\
MSSQL\DisAllowAdhocAccess
```

This disables all OLE DB ad-hoc queries from the SQL server. Don't allow the use of symlinks to tables. (This can be disabled with the --skip-symbolic-links option.) Make sure that the only Unix user with read or write privileges in the database directories is the user that mysqld runs as.

Don't grant the PROCESS or SUPER privilege to non-administrative users. The output of mysqladmin processlist shows the text of the currently executing queries, so any user who is allowed to execute that command might be able to see if another user issues an UPDATE user SET password=PASSWORD('not_secure') query. mysqld reserves an extra connection for users who have the SUPER privilege (PROCESS before MySQL 4.0.2), so that a MySQL root user can log in and check server activity even if all normal connections are in use. The SUPER privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

Don't grant the FILE privilege to non-administrative users. Any user that has this privilege can write a file anywhere in the filesystem with the privileges of the mysqld daemon! To make this a bit safer, files generated with SELECT ... INTO OUTFILE will not overwrite existing files and are writable by everyone. The FILE privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file

into a database table. This could be abused, for example, by using LOAD DATA to load `/etc/passwd' into a table, which then can be displayed with SELECT.

If the DNS is not trusted, use IP numbers rather than hostnames in the grant tables. In any case, people should be very careful about creating grant table entries using hostname values that contain wildcards!

If a restriction is needed in the number of connections allowed to a single account, it can be done by setting the max_user_connections variable in mysqld. The GRANT statement also supports resource control options for limiting the extent of server use allowed to an account.

Conclusion

This report covers the different SQL Injection attacks and the methods of preventing them stating the importance of building secure software and databases. As application coding and security plays a vital role in database security DBA's and application builders should be cautious about upcoming security issues and attacks. In future a new security trend is to provide multiple layers of security within a computing environment. These layers can include multiple firewalls between the Internet and the organization and even firewalls within an organization to protect high-value assets.

Bibliography

1. SPI Dynamics Inc 2002 "SQL Injection - Are Your Web Applications Vulnerable "
2. Adesh Rampat, " Implementing database security and integrity "27 Feb 2001
3. Mitchell Harper "SQL Injection Attacks: Are You Safe " May 2002
4. Stuart McDonald, "SQL Injection: Modes of Attack, Defense, and Why It Matters " Source:http://www.governmentsecurity.org
5. Imperva Inc, "SQL Injection" Source:http://www.imperva.com/
6. Imperva Inc, "Blindfolded SQL Injection" Source:http://www.imperva.com
7. "SQL injection Basic Tutorial" Source:http://comsec.governmentsecurity.org/
8. Anil John "SQL Injection Protection", May 2004 Source:http://www.ensight.org/
9. Seth Fogie, Cyrus Peikari, Prentice Hall, "SQL Server Attacks: Hacking, Cracking, and Protection Techniques"
10. Michael Otey, "Injection Protection",SQL Server Magazine, March 2004
11. Kevin Spett, "Blind SQL Injection", October 2004, Source:http://www.securitydocs.com/