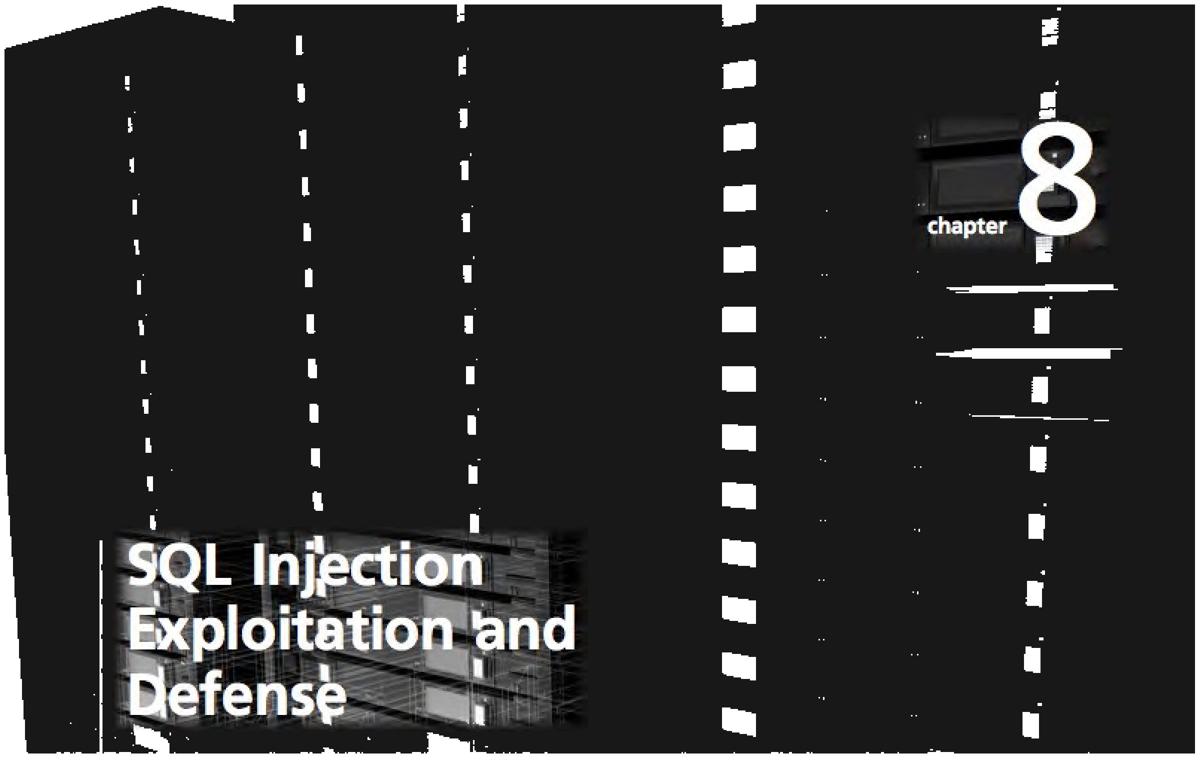

chapter 8

SQL Injection Exploitation and Defense

After reading this chapter and completing the exercises, you will be able to:

- Define SQL injection exploitation
- Identify ways intruders gather information from a network infrastructure
- Describe common strategies for exploiting database infrastructures
- Identify common SQL statements and SQL constructs used to exploit weaknesses
- Apply exploitation for the purpose of identifying infrastructure weaknesses
- Identify defense strategies against SQL injection exploits

Security In Your World

MerryH@K is a self-proclaimed hacking genius of the underground. Notorious for her successful intrusions into the world's most secure systems, she takes great pride in her ability to identify and exploit infrastructure weaknesses. Armed with close to 15 years of experience breaking into private networks, she has obtained a strong understanding of the inner workings of both network architectures and network administrators. Through practice, she has matured her skill for reading basic machine responses and has developed an accurate understanding of the behavior of a typical administrator. In general, MerryH@K's success stems from her refined ability to blindly gather information from foreign systems. Given the extraordinary value of data stored within today's databases, SQL injections and the database infrastructure have become her areas of focus for the last five years. Her latest triumph involved a high-profile company from which she extracted sensitive data, gave herself administrative rights, and ultimately took control of the entire system, which surprisingly began with a single SQL statement created to initiate a delay in the system response. When asked by her peers to describe the most effective technique for extracting data from a secured database, she responds, "Observation."

Achieving success in database intrusion today requires keen observation skills above all other things. Although some database knowledge is necessary, most of what an intruder needs can be found through basic Internet searches and database vendors' Web sites. Intrusion requires the know-how to take a very little bit of information and use it to piece together a larger picture of the target system's construction.

All great security defense strategists attempt to master both the psyche and the behavior of their potential enemies. This is also important for those attempting to defend their infrastructures from SQL exploitation. It can be viewed as a chess match of great proportions. As the intruder is attempting to predict the behavior and strategy of the administrator, the administrator is attempting to predict the behavior and strategy of the attacker. Unfortunately for administrators, the attacker's odds of winning this battle are much greater because security strategies are never foolproof. Whereas security professionals have several reliable applications to help them detect and protect an infrastructure from viruses and intrusion detections, locating and fixing SQL injection vulnerabilities require a great deal of manual configuration, diligent exploration, and constant observation.

Obtaining hands-on knowledge of the various means by which an infrastructure can be exploited is vital to the success of a SQL exploitation defense strategy. This chapter offers insight into the many common and successful techniques intruders similar to

(continued)

MerryH@K use to exploit a system. This chapter, like the previous one, requires the reader to explore and take on the role of a potential attacker who might be working to obtain control over a foreign system.

Please use caution when using the techniques and tools introduced in this chapter. These techniques can cause changes to the database that might interfere with the integrity of the storage system and its interconnected applications. When possible, practice these techniques and tools within a test environment.

Exploitation and Information Gathering

Exploitation is the act of using system vulnerabilities and carefully crafted SQL queries to gather information and subsequently peel away at the infrastructure's security defense for the purpose of gaining access or control of a system. Exploitation does not always result in the control of a system; it depends on the effectiveness of the SQL query injection techniques and the usefulness of the output generated from the system as a result of the SQL query injection techniques. Essentially, the success of exploitation is dependent on the amount of information derived from that system during the exploitation; the more information an attacker can find out about the system, the more freedom they will have within that system. To defend a system from successful exploitation, a security professional must be aware of the means by which information can be derived as well as which information to protect. This section displays the common techniques and exploits used to find different types of information from a system and identify information that those who are exploiting are attempting to obtain.

Information That Aids in Exploitation

As discussed in the previous chapter, inferential exploration is a valuable strategy that can be used by both system administrators and potential intruders for gathering information about a system and its infrastructure vulnerabilities. Locating a weakness is only the first step in the process of intrusion. These weaknesses provide an open door through which intruders can obtain further information about a system and its applications in order to take control. For example, an attacker may obtain access to a particular database, but is not aware of the structure or type of information contained with the tables and records within it. Therefore, the goal is to construct a theoretical picture of the infrastructure, identifying all of the layers and their associated parts. A variety of details, such as the type of Web application, scripting language, Web server, and middleware, combined with the knowledge of the number and type of databases that exist, aid in the goal of obtaining access to the system. Every piece of information helps by providing clues that eventually lead to an intruder developing an understanding of the database schema. The database schema is the overall logical structure of the objects within the database. The schema includes the default stored procedures, tables, views, and users. Obtaining a map or schema of the database subsequently leads to the intruder finding the path to the desired access. This map must be obtained before specific targets can be identified and significant power can be obtained. The end goal of an intruder is often to obtain information residing within the database. Each piece of information that can be gathered about the functionality of the system strengthens the intruder's power over the system, bringing the intruder closer to his or her goal. Therefore, identifying and securing valuable information can be a strong step toward intrusion prevention. This section reviews a few of the most important pieces of information necessary to begin an exploit.

Information About the Database To successfully exploit a given system, an intruder must first obtain information about the database system itself. Without an understanding of the vendor or version of the database, an exploit is nearly impossible to pull off. The vendor and version of the database provide an intruder a great deal of information, allowing more of an opportunity for attack. Furnished with this information, the intruder can infer such things as the SQL language syntax to use to construct injections, the available default procedures that can be called to obtain more information, the way the database processes queries, and the storage mechanisms utilized. A great portion of the schema can be determined as well because each database vendor (e.g., Oracle, Microsoft) contains default tables and objects, so that identifying the vendor and version number can provide an intruder with important information as to the name of the tables in which sensitive information might reside. If an intruder is able to use the information gathered to construct an accurate map of the database scheme, the entire database is in great jeopardy. The vendor provides a jump-start to understanding the database schema of a target database. For example, finding out that the database is Oracle informs the intruder that the SYS user is installed by default and contains the critical system tables and privileges, whereas in Microsoft SQL Server, the database administrator (DBA) can access and control all database items. This is a simple example, but the important thing to understand is that there are variances between database vendors, as well as different versions under the same vendor, that enable intrusion. Finding this information is the first step toward exploitation.

Identifying the Vendor The database vendor can be identified in several different ways. Locating this information is quite easy for the knowledgeable intruder, as many clues are available to help. The following list of clues aid intruders in identifying the database vendor. Keep in mind while reviewing these items that they are only clues; by themselves, none of these clues is reliable enough to draw a conclusion. An intruder often needs to identify more than one of the following clues to conclude with a sufficient level of certainty the vendor and type of database being used:

- *The scripting language*—There are standards and commonalities between the scripting languages being used within Web applications and the underlying database vendor. These standards came to fruition as database vendors began to build relationships with specific languages by including additional support for improved compatibility. Database vendors often lean toward one or two languages, building additional support for these languages to increase the transparency and interoperability. For example, PHP is often used to communicate with MySQL as it offers native PHP drivers allowing for alternative connections to the database through PHP scripting. On the other hand, .NET was developed by Microsoft and from the introduction of SQL Server 2005 until today, a great amount of support for integration with the .NET Framework is included with SQL Server. This relationship offers the highest compatibility and transparency, which has been extended to include ADO.NET. Oracle has offered support for Java since its introduction into the coding world. More recently, Java's acquisition of Sun Oracle has made this relationship even stronger. Therefore, the presence of PHP can indicate that MySQL exists, whereas .NET implies that SQL Server is the underlying database and Java Script used within a Web application can almost provide certainty that the database being used is Oracle.

- *The platform*—The operating system on which the database resides provides clues as to the version of database in place. As discussed in earlier chapters, Microsoft SQL Server is based on the foundation of Microsoft Windows Server 2008, so any indication that the underlying operating system is Microsoft Windows can also be a clue that points toward a SQL Server database. One the other hand, open source operating systems such as Linux and UNIX are often used to support MySQL and Oracle. The platform provides only one small clue as to the underlying database. Many factors unrelated to the database can play a role in choosing an operating system. These factors include cost, complexity, and support, so be aware that the operating system alone is not a reliable indication of the database in use. Every configuration is different and every network, as well as the applications that reside on it, has its own unique and individual needs. Therefore, the platform being used within an infrastructure can only provide supporting evidence when combined with other information. For example, the discovery of Microsoft Windows is hardly an argument for an underlying SQL Server database. Windows is an extremely popular operating system onto which both Oracle and MySQL can be installed, so to make this assumption would be a grave mistake. On the other hand, combining the discovery of a Windows platform with the identification of the usage of .NET within the Web application can make for a much stronger argument that SQL Server is in use.

- *The database response*—Database responses, if present, can provide the most reliable means by which to identify the database. This is true for almost any piece of technology that exists today. It only takes one informative error message to give away the identity of the database being used. This is due to the differences in syntax and error format from one database to another. One apparent way to identify the database from an error message is by using the error code provided. A database error will often include a number or code in hopes of offering the user or administrator with some means by which to obtain further information about the cause of the error; each database vendor has its own convention for numbering errors. Therefore, becoming familiar with the naming convention of the different database vendors provides intruders a way to immediately identify the database. To make matters worse, vendors maintain an area on their Web sites where administrators can search for an error number and find the meaning of the code itself. This is intended to aid in the troubleshooting process, yet if an intruder is unfamiliar with the error code naming convention of each database vendor, he or she can simply search for the corresponding error number on each vendor's site until the correct vendor is found. Table 8-1 provides a list of the vendor Web sites where error code searches can be found. In some cases, error messages include the name of the database, the version number, and the last patch applied to the system. Obviously, generating an error with this information included within the message would be the easiest way to obtain the information, yet these messages are not as prevalent in today's security-conscious society. The amount of information included within a database error is dependent on both the vendor and the type of error. Databases can also be configured to eliminate the return of error messages or to return a generic message for all errors despite the type of error encountered. These circumstances are intended to avoid passing along unwanted information to intruders.

It is important to point out that not all databases will return error messages as described in this section. A database can be configured to allow the Web application to handle errors or to provide a generic message to a user despite the error the database encountered. Ideally, a database is configured to eliminate external error messaging altogether, having no response to an erroneous query, and in turn offering no information to an external user.

When combined, the clues identified in this section can provide an intruder with information about the type or vendor of the database. Having this knowledge, an intruder gains an understanding of how to speak with and manipulate the database to ensure further exploitation and ultimately aids in helping the intruder achieve his or her desired goal.

Vendor	Error code search Web site	Sample
Microsoft SQL Server	www.microsoft.com/technet/support/ee/ee_advanced.aspx	Login failed for user '*DOMAINNAME\ACCOUNTNAME*' (Error code: 18456)
MySQL	http://dev.mysql.com/doc/refman/5.1/en/error-messages-server.html	error 2003: access denied for user @ localhost (100061)
Oracle	www.ora-code.com/	ERROR: ORA-01017 invalid username/password; login denied

Table 8-1 Error code identification

Identifying the Version Identifying the version can be equally as important to an intruder as identifying the vendor. Finding out which version of the database is in use provides an intruder with better insight into the capabilities of the system. Each version of a database management system is different. They can include new features, missing features, or have subtle changes in syntax that could deter the attack, and many new versions are created as a way to patch known security vulnerabilities from previous versions. Therefore, armed with the knowledge of the version of the database system, an intruder will be certain to ensure the correct approach for injection and can potentially identify and take advantages of known vulnerabilities for that version and use these as an advantage toward obtaining control.

With the database vendor identified, locating the version number can be an easy task. Each database management system has standard queries meant for returning the version number of that system. For example, executing the command *SELECT @@VERSION* will return the version of the underlying Microsoft SQL Server as well as the processor, operating system, service pack, and build, which is quite a lot of information for such a small statement! For MySQL, the equivalent statement is *SELECT VERSION()*, and for Oracle, *select * from v$version where banner like* '*Oracle%*'*;* will return the version and build number. Using the injection techniques discussed in the previous chapter, dynamic SQL can be used to send these queries to be processed by the database by injecting the statements as a string parameter within the Web application or from the URL. One of three things will occur:

• *Results returned*—In cases where the application input or output has not been filtered and the application or parameter is expecting a string (the ideal scenario for an intruder), the database will return the results of the injected statement and provide the version number.

- *Error returned*—If a number is expected within the application or as a parameter, as opposed to the string that was input, or the statement is constructed incorrectly, an error will be generated. Surprisingly, depending on the vendor, this error message may also provide the necessary information.
- *Nothing returned*—For scenarios in which the application does filter the input or output or is configured to handle error messages in such a way that messages either do not return anything or return a generic response, nothing will be returned. Under these circumstances, finding the version will be more difficult. Each version of the database will have subtle changes in the way that some statements are constructed. To identify the correct version, the intruder will have to take a trial-and-error approach by choosing the statements that have been subtly changed with each version of that particular database management system. These statements will then need to be injected one by one into the application or URL until the correct syntax returns a result. The correct syntax will then identify the correct version of the database.

Standard statements for identifying the version of a database management system are helpful, yet they are not the only standard statements that exist. Several other standard statements are also available with every database management system. These statements can be used to gather useful information, such as the name, the location, and the language being used on the database. Administrators should familiarize themselves with these statements so as to understand the amount of information that can be gathered during an initial trial at exploitation.

Extracting the Real Data

The information gathered using the techniques discussed in the previous section acts as a building block for an intruder, opening the door of possibilities tremendously. Based on the knowledge obtained about the database, the intruder can make inferences (e.g., SQL syntax, data structure) that will enable the construction of meaningful queries that will provide data to assist in the further development of the intruder's understanding of the database schema. In other words, the exploitation attack is equipped to delve deeper into the system, adding more of an understanding of the data stored within it. Targets can be located and data can be extracted as each layer of integrity is taken apart piece by piece, or bit by bit.

Statement Exploits

Once vulnerability has been exploited and the appropriate syntax has been identified, an endless number of SQL statements can be injected into the database. The intruder has the capability to access the database as a typical user. Restricted only by the privileges of the user for which the queries are being processed, which can be liberal depending on the configuration, thoughtful statements can be constructed for further exploration. This section explores the most common statements constructed for SQL injection attacks as well as the inventive ways they are used to uncover more information about the internal workings of the database. Later sections explore ways privileges can be manipulated and granted to extend the intruder's capabilities, but for this section, it is assumed that the intruder is working under restricted conditions.

Using UNION

UNION statements are very powerful tools of SQL injection attacks. They provide an opportunity for an intruder to attach his or her own queries onto already existing legitimate statements. UNION statements are SQL statements that include the UNION operator, which combines two or more SQL statements from which the output is combined. If a UNION operator is placed at the end of one SQL statement and a SELECT query is added after the UNION statement, then both queries will produce output. For example, imagine a scenario in which a local online specialty grocer allows customers to search and purchase products on its Web site, *www.yum.com*. This site allows patrons to choose from different types of foods, such as dairy, produce, and meat. Each category has its own respective button; when visitors click this button, a new page appears displaying a list of available products for that food category. For example, Dairy's button may link to the following URL:

http://www.yum.com/index.asp?category=dairy

This URL shows the variable as *category* and the value *dairy.* The scripting language will take the output from the button (the preceding URL) and send it as a request to the database. Again, we are using ASP, and so the code to request this information may be similar to this:

```
food_cat = request("category")
sqlstr= "SELECT * FROM products WHERE Food_Category = '"&food_cat&"'"
set rs-conn.execute(sqlstr)
```

In this case, the ASP would create the following SQL statement to be executed by the database:

```
SELECT * FROM products WHERE Food_Category = 'Dairy'
```

In this case, the database would return one or more rows that match the WHERE clause, which in this case is Dairy. No errors are generated for this scenario. Keep in mind that when using SQL, alphanumeric values must be enclosed using single quotes (e.g., 'Dairy').

If a UNION statement is injected after 'Dairy' and a new statement is created as such:

http://www.yum.com/index.asp?category=dairy **union select Table_Name from Information_Schema.Tables- -**

both results will be combined and returned, given the syntax is correct. In this case, if the syntax is correct, the table name from within the Information_Schema.Tables will be returned. The statement Information_Schema.Tables is the location from which a user can view all of the tables from within the SQL Server database. The Table_Name call is essentially asking the database to provide all of the names of the tables in the Information_Schema area. Initiating this procedure will return all tables for a specific database. The double dashes (--) comment the rest of the statement.

The difficulty involved with using the correct syntax for UNION statements is that the data type must be the same for each column that the original statement is returning and the number of columns being requested in the new statement must be the same as the results of the original statement. Therefore, using the previous example, if the original statement has more than one table and it is of a different data type than that of "Table_Name," this statement will fail. Without complete knowledge of what is being returned from the original request, it can take a bit of trial and error to successfully pull this off.

The first step is to try to figure out how many columns are being used in the original query. If error messages are being returned, there is a technique that will help identify this number. The trick is to keep adding null expressions in place of the SELECT statement in the URL until the error messages disappear, such as:

http://www.yum.com/index.asp?category=dairy **union select null from Information_Schema .Tables- -**

http://www.yum.com/index.asp?category=dairy **union select null null from Information_Schema .Tables- -**

http://www.yum.com/index.asp?category=dairy **union select null null null from Information_ Schema.Tables- -**

http://www.yum.com/index.asp?category=dairy **union select null null null null from Information_ Schema.Tables- -**

Depending on the number of columns being requested in the original statement, this can be quite a long process. Yet, if null is accepted (which it is in most modern databases), it is an effective technique. Because null is being used, the data type is irrelevant, ensuring that the focus is on finding the number of columns avoiding data type errors.

Once the number of columns being called from the original request has been determined, the data type necessary for each column can also be determined. Again, this involves trial and error, but can be determined by replacing each column one at a time with a specific data type. For example, assuming that it was discovered that the original statement contained only three columns, the following strategy would determine the correct column from which to extract table names from the Information_Schema.Tables view:

http://www.yum.com/index.asp?category=dairy **union select Table_Name, null, null from Information_Schema.Tables- -**

http://www.yum.com/index.asp?category=dairy **union select null, Table_Name, null from Information_Schema.Tables- -**

http://www.yum.com/index.asp?category=dairy **union select null, null, Table_Name from Information_Schema.Tables- -**

For those data statements in which an error is returned, the incorrect data type exists, so the choice is to either attempt a different data type for that column, or leave it as null and out of the statement altogether, such as:

http://www.yum.com/index.asp?category=dairy **union select Table_Name, null, Table_Name, from Information_Schema.Tables- -**

Automated tools have been developed by both administrators and intruders to assist with this effort. They can be found by searching online. These tools can be used in cases where *null* is not able to be used or when the columns are much too large and the task becomes much too time consuming. As you can see, table names can be extracted using these techniques: *Table_Name* can be replaced with *Column_Name*, whereas *Information_Schema.Tables* can be replaced with *Information_Schema.Columns* to identify and view all of the columns in the database using the same strategy!

Using Conditions

Conditional statements can also be quite handy in providing the intruder with clues adding to the theoretical database schema mapping. Essentially, conditional statements assert that if a certain condition shows true, then a specified action should be taken and if a certain condition shows false, then an alternative action should be taken. This section shows how conditional statements can be used iteratively to expand the intruder's view of the database and allow further collection of information. Conditional statements are greatly advantageous in situations where UNION statements are not allowed.

Using conditional statements as injections is like playing a game of *Twenty Questions* with the database. For example, an intruder can ask the database *Am I the Administrator?* by constructing a statement that asserts if the username for which my queries are being processed is equal to the administrator account for the database, then return a 1 otherwise error. This is the constructed statement for SQL Server:

```
IF ((select user) = 'sa' OR (select user) = 'dbo') select 1 ELSE
select 1/0
```

Because the expression 1/0 cannot be calculated, the statement will generate an error, if the "if" condition is not true. Otherwise, it will return a 1. In cases where applications are configured to never display error messages, the intruder's question, *Am I the Administrator?* can still be answered because a 1 will not be returned to the user. Generating error messages is often not the best approach for an intruder who wants to ensure that his or her actions are transparent to the system and the system administrator. Conditional statements similar to the previous one do not return a great deal of information and it is likely that several statements will be required to gather the desired amount of information about the database scheme. With each statement, an error message is generated and recorded within the error logs of the database. Network and database monitoring systems provide an administrator with the capability to set up alerts under certain conditions. For example, an administrator can create a rule within a system that states *If the number of error messages that the database experiences exceeds the baseline or number of error messages that are typically experienced, alert staff.* If this rule is created within an environment, then an intruder who is generating excessive numbers of error messages to obtain information will be identified through alerts. It is important to point out that the absence of an error message does not indicate the absence of an error; only a returned result indicates error-free processing. Cases where errors are not sent to external users (blind environments) typically indicate that errors are being handled and deterred by the Web application as a safety measure to minimize the impact of a SQL injection, yet these messages are still being logged by the database. Therefore, although conditional messages are effective in nonblind circumstances, they should not be used in situations where a great number of iterative conditional statements are necessary. There are several alternative and less-alarming ways to obtain an answer from a conditional statement.

Initiated delays or time-based responses are alternative means that have been proven to be an effective means to finding an answer to a constructed conditional statement, in both blind and nonblind environments. Time-based response strategies involve the intruder creating statements that either keep the servers busy or delayed for a certain amount of time to indicate the yes-or-no answer to the constructed statement. For example, if a conditional statement asks the server to respond displaying a 1 for both true and false conditions, yet adds a delay of the response (let's say five seconds) on the false condition, then the presence or absence of a pause is what will provide the intruder with the correct answer.

```
IF((select user) = 'sa' OR (select user) = 'dbo') select 1 AND WAIT
FOR DELAY '0:0:5' ELSE select 1
```

Table 8-2 provides the statements used to pause a system for different database vendors.

Platform	Procedure	Comments
Microsoft SQL Server	WAIT FOR DELAY '0:0:5'	Delays a system for five seconds
MySQL	BENCHMARK(100000, encode('Hello')	Causes the system to encode the word *hello* one million times, resulting in a delay; the exact time of the delay can be determined by practicing the commands prior to using them
Oracle	Pg sleep(5)	Delays a system for five seconds

Table 8-2 Initiating server time delays

The disadvantage of using time delays is the delay itself. Keep in mind that the goal is to gather enough information to better understand the database schema. This will require quite a bit of information and can require a number of conditional statements or yes-and-no responses. This alone takes a lot of time. If each conditional statement includes a time delay of five seconds (which might not be enough time to distinguish between normal system delays and the statement-forced delay), the process takes five times as long as without the delays. Therefore, although this is an effective injection alternative for blind scenarios, it adds a lot of time to the overall process.

Another option for using conditional statements to gather information is designing statements to return different results for true statements as for false statements. For example, constructing a statement that returns a 1 if the condition is true and 2 if the condition is false will provide a clear indication of the correct response.

```
IF ((select user) = 'sa' OR (select user) = 'dbo') select 1 ELSE
select 2
```

Although this might seem to be the simplest and most effective choice for gathering information using conditional statements, because there are no error messages or time delays, it is important to be aware of all the strategies that exist to effectively defend against conditional statement injections. This point will become clearer as the chapter continues. Despite the type of conditional statement chosen, the process of gathering information using only yes-or-no responses is quite a laborious and time-consuming task. However, depending on the defense strategies of the database administrator, conditional statements might be the only means by which information can be gathered. Depending on the value of the data stored within the target database, using conditional statements might just be well worth the trouble!

Large-Scale Extraction

As this chapter has stressed thus far, each bit of information gathered from a system provides a building block to obtaining information and the deeper the discoveries, the more capabilities that will result. Therefore, although the strategies discussed in previous sections might seem tedious and time consuming, they are effective at providing intruders with the necessary data to extend their search. As the data scheme becomes more understood and a partial database scheme is created, the strategies presented thus far can be combined to extract data on a much grander scale. Equipped with the basic information collected thus far, the intruder can now delve much deeper into the infrastructure by identifying the number of accessible databases and by extracting the tables and columns from within them. This section explores the strategies used to locate and extract large scales of information from target databases.

Obtaining Database Names The first step to obtaining large amounts of information is identifying a list of accessible databases that reside within the infrastructure. Once these have been discovered, one or several can be targeted and tables and columns can then be extracted. Here, the intruder will need to use the information that has been gathered thus far, as the exact statements injected to find this information will be dependent on the vendor for correct syntax. Table 8-3 displays a list of potential statements that can be used to extract the names of the surrounding accessible database for the current users. These statements can be injected into an application field or added to an application URL as previously described.

Platform	Statement	Comments
Microsoft SQL Server	SELECT name FROM sysdatabases	Returns a list of databases; the *sysdatabases* is installed with SQL Server and contains entries for each major database, which includes the *master*, *model*, *msdb*, and *tempdb*
MySQL	SELECT schema_Name FROM information_schema.schemata	Returns a list of databases; *Information_schema* allows access to the metadata of the databases and *.schemata* is a table that holds information about the databases
Oracle	SELECT global_name FROM global_name	Within Oracle, a user can only maintain one connection to one database, so the only list that can be retrieved is that of the name of the current database

Table 8-3 Database discovery

Obtaining Table Names Once a list of accessible databases is discovered, the next step is to extract the tables within the target database. This is done by finding the table that holds the number and names of all of the tables in that particular database. Each database vendor has a table called sysobjects that will provide this information. Table 8-4 displays the statements that can be constructed and injected into Web applications or URLs to extract the list of database tables.

Obtaining Columns At this point, the intruder has the names of all of the accessible databases, has targeted a database, and has identified the names of the tables within that database. Having these names, the intruder can construct statements to extract the columns

Platform	Statement	Comments
Microsoft SQL Server	SELECT name FROM systables	Returns a list of tables found in the *sys.tables* view of the target database
MySQL	SELECT Column_Name FROM information_schema .tables	Returns a list of databases; *Information_schema* allows access to the metadata of the databases and *.tables* is a table that holds information about the tables
Oracle	Select Table_Name from all_tables;	Returns all of the tables in the database

Table 8-4 **Table identification**

from the data and, subsequently, the data from within these columns. Table 8-5 displays a few statements that can be constructed and injected into Web applications or URLs to extract this data.

Platform	Statement	Comments
Microsoft SQL Server	SELECT name FROM syscolumns	Returns a list of tables found in the *sys.columns* view of the target database
MySQL	SELECT Column_Name FROM information_schema.columns	Returns a list of databases; *Information_schema* allows access to the metadata of the databases and *.columns* is a table that holds information about the columns
Oracle	Select column_name from all_tab_columns;	Returns all of the columns in the database

Table 8-5 **Identifying columns**

If successful, the intruder now has a pretty clear picture of the database schema and the partial mapping can be completed.

Advanced Techniques

Often, filters are used within Web applications as a way to identify and defer an injection. These filters will often use string comparisons to identify dangerous code being input or displayed as output. These filters will search for and block certain known SQL injection characters or statements. Unfortunately, intruders have found several ways to evade and trick these filters. Often, these filters are configured to compare strings or characters that may be found in the application URL or input fields with those common characters and well-known SQL injection keywords. Even the most basic detection systems filter strings such as SELECT and UNION. So, as a way to push injections through filters undetected, intruders use variations of these strings and keywords. The following list describes ways attackers can trick the system or hide from filters:

- *Encoding*—Changing characters from the expected standard (e.g., ASCII) to a different unexpected one (e.g., URL, Hex, Binary) that direct string comparisons will not detect and that can evade basic filters. For example, instead of using a single quote (') within an injection because it is often filtered, the URL encoding standard of a single quote, *%27*, is used instead.
- *Case sensitivity*—If the Web application that is filtering common strings such as SELECT or UNION is case sensitive, variances in case can be used to avoid detection. For example, the word UNION might be found through a basic string comparison, whereas uNiON will not.
- *Breaking it down*—Another way to avoid basic string comparisons from filters is to break the common word up using URL code so that the word is pieced back together once processed. For example, SELECT could be expressed in an injection statement as 'S'+'ELE'+'CT.
- *Using alternatives*—There are several different ways to express statements, so if one injection technique fails and filtering is suspected, alternative words can be used to express the same statement and avoid filters. For example, when using conditional statements, *Case* statements can be used instead of *If .. Then* statements. In case of numbers, 2+2 can be used in place of 4.

These techniques, when used together, can be very powerful. Considering the amount of possible combinations of variations that are possible, they can make detection almost impossible.

Exploitation of Privileges and Passwords

The success of all of the statements described in the previous sections is dependent on the permissions of the user for which the injection statements are being processed within the database. Therefore, if the users for which the injection statements are being executed only have the right to view one database, a query injected with the intention of gaining access to view all of the remote databases will only display the one for which the privileges exist. This can be quite limiting, especially with newer versions of a database management system. As was discussed earlier in the book, a great deal of granularity is offered to administrators when setting up privileges for the database users. Therefore, to apply the techniques in this chapter and gain an understanding of the target database's schema, privileges must be identified and, if possible, increased. This section explores a few of the most common techniques for identifying and obtaining administrator privileges.

Identifying Privileges

Using techniques that were already discussed in this chapter, an intruder is able to locate and read tables and columns within the database. Assuming that the goal is to locate and potentially escalate user privileges, the intruder must first know which privileges are grantable on the system. Therefore, user privilege tables must be located and viewed. Table 8-6 displays the SQL statements that can be constructed and injected into vulnerable Web applications and URLs to identify the available grantable privileges on the database.

Platform	Statement	Comments
Microsoft SQL Server	EXEC sp_helprotect NULL, 'myusername'	Returns a list of all permissions for the username provided in single quotes, for that particular database. It is necessary to change *myusername* to the name of the current user; therefore, the current username must be found first for this statement to be effective
MySQL	SELECT Grantee, privilege_type, is_grantable FROM information_schema .user_priviliges	Returns a list of grantable privileges on the current database; *Information_schema* allows access to the metadata of the databases and *.user_privileges* is a table that holds the grantable privileges
Oracle	Select * from user_sys_privs Select * from user_role_privs Select * from user_tab_privs Select * from user_col_privs	There are four different types of privileges within Oracle: System, Role, Table, and Column. *Sys_privs* holds all current user-grantable system privileges; *Role_privs* holds all current user-grantable roles; *Tab_privs* holds all current user-grantable table privileges; *Col_privs* holds all current user-grantable column privileges

Table 8-6 **Identifying grantable privileges**

Obtaining Passwords

On all database management systems, user passwords are stored using a nonreversible hash within a table for which privileges are needed to access. A **password hash** is a cryptology-encoded string version of a user or system password. Passwords are often stored within a database as a hash to increase security of the password, yet not all passwords are saved as a hash. Some are saved as text strings and pose a great risk to the data integrity. This section focuses on the injection methods by which the text and hash version of stored passwords can be extracted and decrypted for the purpose of escalating one's privileges within a system.

If an intruder uses the techniques discussed throughout this chapter to obtain the names and fields of the tables for which passwords are stored within a database, the intruder can then extract passwords and use them to log in to the system, elevating his or her rights on that system. Whether the password is stored as a hash or a simple text string will determine the ease with which the intruder can put the passwords to use. For example, if a database is stored as a text string in the user table of the database, and an intruder locates the field in which these passwords are stored, a simple SQL statement can be constructed to extract the string value of the field and with very little effort obtain the actual characters of the password. Although password hashes are much more secure, these too can be extracted using injected SQL statements. You can find programs online that are built to automate the processes of decoding hashed passwords, so even the strongest hash does not guarantee safety when stored in a database. Table 8-7 provides examples of SQL statements that can be constructed to retrieve text passwords stored within a system.

Platform	Statement	Comments
Microsoft SQL Server	2000- SELECT name.password FROM master.dbo.sysxlogins 2005- SELECT password_hash FROM sql_logins	In SQL Server 2000, login information is stored in the *sysxlogins* table of the master database; this table was discontinued in later versions and the login information was moved to the *sql_logins* table
MySQL	SELECT user, password FROM mysql.user	Passwords are located in the *mysql.user* table in MySQL
Oracle	SELECT name, password FROM sys.user$ where type#=1	Text passwords can be found in *sys users* in the *system.mgmt_credentials2.table* by default

Table 8-7 **Extracting passwords**

Obtaining Privileges

Obtaining user passwords is one sure way of obtaining higher privileges within a system, yet very often the tables that hold this information require high privileges themselves. More often than not, intruders must find a back-door strategy for viewing the tables that can cause the greatest harm. Unfortunately for database administrators, but fortunately for attackers, these back-door strategies do exist. Unlike many of the strategies that were discussed thus far, obtaining administrator privileges from a system is vendor and version specific, so these strategies vary greatly from one system to the next. This section identifies and explores the vulnerabilities that offer an intruder the potential for escalating privileges.

Brute Force Attacks Brute force essentially means to use every possible combination. Brute force attacks often involve the act of obtaining passwords or specific pieces of information through iterative trial and error. For example, a brute force attack on a lock combination would involve several attempts of each number of the lock in each order possible until the safe is cracked, so to say. Brute force attack complexity depends on the information that the intruder is attempting to obtain. For example, if the combination of a lock is only three numbers in length and only the numbers 1, 2, and 3 can be used, then the attack will be quick and successful. With this said, brute force attacks are often more successful if a great deal of background knowledge is obtained prior to the attack. For example, information such as the length of a desired administrative password as well as the available characters allowed within the password both can support a brute force attack. It often also involves stages. A brute force attack on an administrative password might begin with a brute force attack on a user password, so as to gain access to database tables from which it is necessary to gather information. Brute force attacks are not uncommon approaches to successfully elevating privileges.

Automated Tools Several tools can be found online that are created for gathering information about a database to assist in taking control through privilege elevation. As was mentioned earlier in the chapter, blind attacks can be very time consuming and laborious, and much like brute force attacks, they involve iterative tasks for obtaining information from a system that provides little to no response. In addition, several tools are available online that are created to automate the process of SQL injection. Some are intended to speed up or decrease the complexity of brute force attacks and others are created specifically for attacks on certain platforms. Either way, these tools can be very dangerous and offer an intruder a way to attack several system vulnerabilities at once.

OPENROWSET Any of the SQL statement strategies provided earlier in this chapter can be used to escalate privileges because they can be used to gather information from a database. The possibilities for injecting conditional statements are endless and certainly not limited to the examples provided in the chapter. With a little ingenuity, anything is possible. One specific procedure worth noting is OPENROWSET. OPENROWSET is a common procedure available in SQL Server and can be used for escalating privileges within SQL Server. OPENROWSET allows a user the ability to remotely connect to the database to retrieve information as a user different from that defined by the Web technology. OPENROWSET was developed to provide administrators an alternative way to access tables in a remote server one time, using their database credentials. The requirement of OPENROWSET is that the credentials provided to log in remotely must match those on the database for which the connection resides. Therefore, a username and password must be obtained to use OPENROWSET, as mentioned previously, but OPENROWSET does not have a time-out for failed attempts at logging in. Therefore, an intruder can use brute force strategies and tools to obtain the necessary credentials. Once the credentials are achieved, OPENROWSET offers a remote connection to the database in a way that data can be manipulated and changed in bulk, providing the necessary door for an intruder to achieve full control of the system.

E-Mail Most modern database systems provide an e-mail function that automatically sends messages via e-mail to administrators or users. These messages can be alerts or general responses from the server. For example, in large environments to better support database users, the database will automatically generate an e-mail that includes a user's password if the password was lost or forgotten. An attacker can take advantage of this system by creating malicious code to redirect or copy the information being sent. Although the information that the intruder will receive is unpredictable, it does provide a channel for escalating database privileges and obtaining sensitive information.

Defending Against Exploitation

Now that vulnerabilities have been identified through the diligent attempts of exploit against one's own infrastructure, vital information has been obtained. For administrators, the only question that remains is *How do we defend our systems?* By now, it is clear that the answer involves much more than code and user restriction alone. Becoming aware of the weaknesses of an infrastructure is the first line of defense, as awareness alone arms administrators by providing them with areas on which to focus their defense. A strong defense includes meticulous system monitoring and thoughtful action. There isn't a simple and quick answer to what is necessary, and SQL injection defense isn't an exact science. As this section explores the strategies for limiting the potential of a SQL injection attack, it is important to understand that this is an ongoing effort and that not one strategy identified here will work alone. The most effective approach is to combine all of these strategies, creating a very strong force to be reckoned with.

Using Bond Parameters

Although this chapter has identified several alternative methods to accomplish a SQL injection attack, the most common intrusions involve malicious SQL statements that are inputted as parameters. **Bond parameters** are placeholders for which to bind user input. Once bonded, the input is placed into an already constructed SQL statement and sent to the database to be processed. Statements are predefined, so dynamic SQL is not necessary, and because of the way that the statements are constructed, the user input has nothing to do with the SQL statement parsing. The user data is treated as data alone and cannot affect the SQL statement, so if the data is an injection, it isn't processed by the database in a malicious way. This is a very effective defense, yet bond parameters cannot be used for every user variable, only data values.

Sanitizing Data

By this point, it should be clear that some type of input filter needs to be applied. Although this chapter identified ways input and output filtering can be defeated, security requires multiple layers of defense. Each layer adds complexity for the intruder, and blocking well-known characters and keywords plays an essential role in limiting the intruder early on. It is important to note that *word filtering* is not synonymous with *word blocking*. **Word blocking** refers to the act of blocking keywords that are not allowed to be used as input within a Web application or a Web URL. **Word filtering** is a balance of blocking those known keywords that are not allowed to be used as input within a Web application or Web URL, and identifying those keywords that are allowed to be used as input within a Web application or Web URL. It is much better to place the majority of one's effort into defining only those words or characters that can be inputted in a specific field, than to spend hours fruitlessly attempting to identify every single variation and combination of potentially dangerous items that could come across the network. Therefore, when possible, limit and mask data fields. This is not to say that blocking is not necessary; common keywords should still be blocked, just to a reasonable extent.

Restricting and Segregating Databases

As was discussed several times throughout this book, granularity is the key to setting permissions. At this point, it should be clear that privileges on the database should be assigned with the greatest granularity available, yet what is often overlooked are the permissions of the users on the Web server. Because the injections will be processed with the privileges of the Web application, the Web server should have the most restricted privileges possible. The ideal strategy is to provide the Web application with rights only to access limited tables on an isolated server until the appropriate credentials are supplied. At that point, some of the restrictions would be lifted, corresponding with the user's rights for whom credentials were provided. Related to this, the administrator should design the infrastructure so that the Web server and database server are segregated. Therefore, if users are not configured properly on the Web server, it will not affect their capabilities on the database management system.

Security-Conscious Database Design

One goal of this chapter was to provide administrators with an outsider's view of their infrastructure. Great insight can be gained from this, so use what you learned. When designing the data infrastructure, consider where things are stored and how this is viewed by an outsider. Does the organization give away too many clues? Consider how objects are named; what information does the name give to an external visitor? Try changing default names on sensitive data whenever possible and do not use object views for critical database objects. It can also be beneficial, if the resources are available, to create a honeypot environment. A **honeypot** is a fake environment that includes false data to mislead intruders who are attempting to gather information about the database.

Diligent Monitoring

Monitoring is equally as important as all of the other security techniques available. Through logs and monitors, an experienced security professional can identify a potential attack. For example, conditional statements that rely on errors will generate a great number of database connection pool errors, so tracking this will show indications of real-time attacks. Automated tools used in brute force attacks increase resource usage, so recording database queries per minute and database requests per minute can also be an accurate alert of potential attack. Use information gathered through inferential observation to create baselines. Use the baselines to create thresholds that sound an alarm in situations where resources are exceeded. If configured correctly, the network management logs and tools can be a significantly valuable security defense tool.

Chapter Summary

- SQL injections are used to identify and exploit weaknesses within a database management system.
- Through inferential explorations and vulnerability exploitation, intruders are able to obtain enough information to obtain a picture of the infrastructure as well as the database schema itself.
- An intruder who has obtained an accurate database schema of a particular database is aware of how the data is structured and is able to find any information that is desired.
- The more information a potential intruder can gain from a system or infrastructure, the better equipped he or she is to obtain control. Valuable information about a database includes its vendor, operating system, supporting applications, and version.
- Standards exist for how systems are configured. For example, certain Web applications are better supported when combined with certain scripting languages or database applications. These standards help intruders make assumptions and gather information to aid in their attack.
- Database errors often include the vendor's name. Therefore, databases that return errors provide intruders with a clear view as to what system they are trying to attack.
- Union statements are powerful components for an attack. They provide intruders with the capability to append malicious code onto legitimate SQL statements.
- Conditional statements can be used in several ways to help intruders obtain access and subsequently gain control over a system. Using conditional statements is the way unauthorized individuals ask the database yes-or-no type questions.
- Union and conditional statements combined offer a powerful strategy to obtaining control of a system using SQL injection. With these statements, privileges can be identified, data can be retrieved, and a theoretical database schema can be created.

- Intruders can evade Web application filters by using alternative encoding, case, and words that are unexpected.
- Privileges can be escalated by obtaining passwords through brute force attacks, automated tools, e-mail redirection, and OPENROWSET.
- Bond parameters defend against exploitation by ensuring that malicious SQL statements are not processed.
- Sanitizing data is an important component to keeping Web application data legitimate and the database secure. Sanitizing data can be done by applying filters and blocking words.
- Segregation of the database servers will ensure that they can be closely monitored and secured.
- When designiug a database infrastructure, you should be considerate of what can be seen from outside the network. For example, tables and critical database items should be named so as to not identify their contents to outsiders.

Key Terms

bond parameters Placeholders to bind user input.

exploitation The act of using system vulnerabilities and carefully crafted SQL queries to gather information and subsequently peel away at the infrastructure's security defense for the purpose of gaining access or control of a system.

honeypot A fake environment that includes false data to mislead intruders who are attempting to gather information about the database.

password hash A cryptology-encoded string version of a user or system password.

word blocking The act of blocking keywords that are not allowed to be used as input within a Web application or a Web URL.

word filtering A balance of blocking those known keywords that are not allowed to be used as input within a Web application or Web URL, and identifying those keywords that are allowed to be used as input within a Web application or Web URL.

Review Questions

1. List and explain at least two ways that an intruder can identify the vendor of a database.
2. Identify the SQL statement for identifying the version of an Oracle, MySQL, and Microsoft SQL Server database management system.
3. Define a UNION statement and explain how it can be used to exploit a system.
4. Explain the constraints of the UNION statement. Provide an example to support your answer.
5. Provide an example of a UNION statement used in conjunction with a SQL injection.

6. Define a conditional statement and explain how it can help an intruder gather information from a system.
7. List the SQL statement to obtain the list of all database names within a Microsoft SQL Server envrionment.
8. List and explain at least two ways that intruders can defeat Web application filters.
9. Identify at least two ways that privileges can be escalated or obtained by an intruder from within the system.
10. Explain how the use of bond parameters can help defend against Web application exploitation.

Case Projects

Case Project 8-1: Defending Against Web Application Exploitation

Create a plan detailing steps that should be taken to defend against SQL injection exploitation within Web applications.

Case Project 8-2: Time Delays

Provide a scenario where time delays can be used to gain information from a system and provide an example to support your statement.

Case Project 8-3: Oracle Vulnerability

Using Oracle's Web site *www.Oracle.com* and the Internet, locate at least one security vulnerability in a current version of an Oracle application.

Case Project 8-4: Microsoft SQL Server Vulnerability

Using the SQL Server Web site *www.microsoft.com/sqlserver/2008/en/us/* and the Internet, identify at least one vulnerability in a current version of a Microsoft SQL Server application.

Case Project 8-5: MySQL Vulnerability

Using My Server's Web site *www.mysql.com/* and the Internet, identify at least one vulnerability in a current version of a MySQL application.

Hands-On Projects

Hands-On Project 8-1: Exploitation Awareness

Nathan is a security consultant for Tyler & Haley financial, a large mortgage lending company in New York City. He has been hired to raise the company DBA's awareness about SQL injections.

1. Nathan is giving a speech on the four steps of exploitation. What four steps do you anticipate him including within his speech?

2. Nathan is planning to describe at least three ways a database management system can be identified. What three ways for identifying a DBMS should Nathan cover?
3. Nathan plans to provide a few examples of SQL statements that can be used to gather information externally. Provide two examples of SQL statements that Nathan can provide.
4. What suggestions do you expect Nathan to provide for securing the company databases against SQL injections?