

Testing Web Applications for SQL Injection

Sam Shober

SamShober@Hotmail.com

Abstract:

This paper discusses the SQL injection vulnerability, its impact on web applications, methods for pre-deployment and post-deployment testing of the application, and solution implementation suggestions. SQL injection is one of many parameter manipulation attacks that can be executed against Web Applications. The SQL injection vulnerability is easily identified and prevented in most systems, but often is not.

Introduction:

If a web application has a vulnerability that allows a potential attacker access to restricted data it is definitely a security risk, but is it a defect? The answer is 'Yes.' While many security risks are the result of missing patches to the web server's underlying systems; often times those systems contain vulnerabilities that can be prevented through code and testing regardless of the status of the web server.

A vulnerability that websites commonly suffer from is SQL injection. SQL injection, as its name implies, is a technique of inserting SQL statements into an existing web application query for client-supplied data. SQL injection is performed with the intent of exploiting a vulnerable web application.

SQL injection is an application-level vulnerability that allows attackers access despite the presence of SSL keys, authentication levels and other security features. SQL injection compromises the commands used to validate the user's access authentication or update the database. SQL injection is used to permit unauthorized access to restricted data by attackers who have neither a logon ID nor password.

Secure Socket Layer (SSL) traffic offers no protection from a SQL injection attack. This is because SSL encrypts the traffic between the browser and the server. To exploit the SQL injection vulnerability a legitimate session between the browser and the web application is used. In many cases the connection through which the attack is taking place is secured by SSL. An attacker submits requests through the browser, slightly altering the web applications commands to violate the application. Similarly, a firewall will not protect against a SQL injection attack. A SQL injection attack uses a legitimate, established session between a browser and the web server.

Some "Attack Signature" based Intrusion Detection Systems (IDS) are ineffective against SQL injection unless their signatures have been specially configured for input forms on the site. In a rapidly changing environment it may not be possible for the IDS to remain up-to-date. Due to budgetary constraints that limit licenses, companies that have an IDS often configure it to cover public sites only, leaving their internal web applications unprotected and exploitable.

SQL injection is a vulnerability that can exist on any web application that interacts with a relational database. Many websites displaying this vulnerability use Microsoft's popular ASP technology coupled with IIS and a SQL Server backend. Sites constructed with virtually every web application technology have been shown to be vulnerable when used in conjunction with SQL Server, Oracle, MySQL, and other relational database engines. Nearly every web page technology in place is vulnerable to a SQL injection attack if it communicates with a relational database.

There are several types of attacks that may be conducted with SQL Injection. The results of the attacks are as varied as the systems being compromised. In brief the attack types are:

- Authorization bypass attacks– Web based authentications are bypassed and areas that should be restricted are made accessible to the attacker.
- SELECT statement attacks – The attacker modifies existing queries to return data of his/her choosing instead of the data types intended by the system designer.
- INSERT statement attacks – The attacker is able to insert data directly into the system for his/her purposes.
- SQL Server Stored Procedures – The system level stored procedures are called and used to provide an extraordinary level of access to the attacker.

For a comprehensive description of how each of these attack types is used, download the following SQL injection articles.

- **SQL Injection: Are Your Web Applications Vulnerable?**
<http://www.spidynamics.com/whitepapers.html>
- **Advanced SQL injection In SQL Server Applications by Chris Anley**
http://www.nextgenss.com/papers/advanced_sql_injection.pdf

Testing For SQL Injection

SQL injection is easy to recognize as a tester and very easy to protect against in the production environments.

The easiest and first test for a tester to perform when testing for SQL injection is to insert apostrophes (') into form fields. This will yield an ODBC error, ADO recordset error, or a system access denied message. For example:

Ado RECORDSET ERROR	ODBC ERROR MESSAGE
ADODB.Recordset.1 error '80004005' SQLState: 37000 Native Error Code: 911 [MERANT][ODBC Oracle 8 driver][Oracle 8]ORA-00911: invalid character /includes/functions.inc, line 18	Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the character string " AND password =". /secure/bb_login.asp, line 32

A message similar to the example is the first sign to an attacker that the potential exists to retrieve data that might otherwise not be available. Following the receipt of such a

message, test to see what sort of access can be obtained using the specific attacks mentioned later.

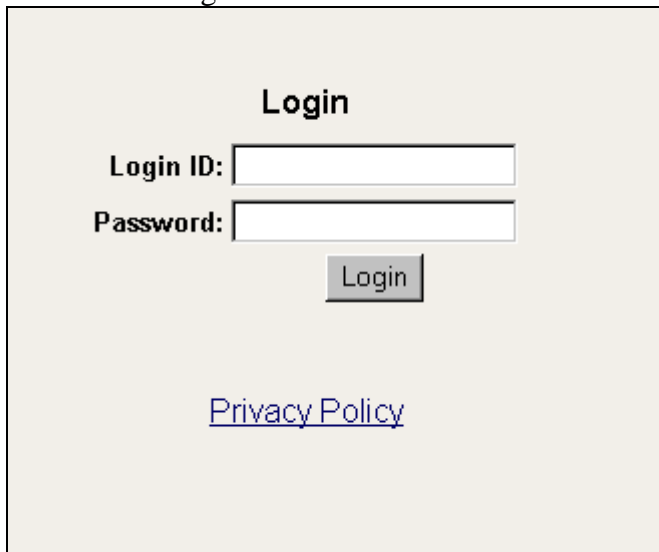
Sometimes servers are configured to not deliver error messages to the end user. Instead, they may return generic pages or redirect the browser back to the home page, for example. This does not mean the test did not reveal a vulnerability, but only that the tester must put in a little more work to identify the vulnerability.

Many applications use client-side JavaScript to encourage the user to input properly formatted data. Because client-side scripts are incredibly easy to bypass, client-side scripting should be considered for correcting client-side usability issues rather than as a serious deterrent to a SQL injection attack. Rather than relying on client-side scripts to perform your security checks, couple the server-side validations with client-side scripts to prevent the average user from making bad inputs, but don't rely on them to protect the application from a determined individual.

Specific Types of Attacks

Testing for Authorization bypass attacks–

An authorization bypass attack is one in which web based authentications are bypassed making restricted areas accessible to the attacker. A standard web authentication form looks something like this.



To perform a SQL injection attack against a typical web-based authentication form the attacker may enter the following into the **Login ID** or the **Password** field.

' or '' = ''

This seemingly simple bit of code entered into the password field would allow an attacker to access the secured part of the site with only a valid user name, thus having access to the legitimate user's data without the user's knowledge. The attacker could commit fraud,

transfer funds, alter purchase orders or perform any activities that an ordinary user would be able to perform.

With ' or ' = ' entered into both fields the user name and the password fields will allow the attacker to access the secured site without a user name or password. Having the injected SQL statement entered into both fields may give the attacker less access or perhaps it could give them administrative access. The rules governing the application determine the type of access and it must be tested.

Testing for SELECT statement attacks –

A SELECT statement attack is one in which the attacker modifies existing queries to return data of his or her choosing instead of the data types intended by the system designer. There are several types of SELECT statement attacks. The types and complexity of SELECT statement attacks is covered thoroughly in the aforementioned white paper. Select statement attacks can reveal a systems databases structure and provide information beyond simple access to secure systems that the authentication bypass attacks provides. Conducting select statement attack requires some patience on the part of the tester or attacker, but patience is rewarded with more valuable information about the company and its customers.

Direct injection attacks

For the first test, append the word "OR" to a parameter's legitimate value. For the second test insert an ampersand or its Unicode equivalent at the tail end of the parameter. If either test produces an error, the server is vulnerable to direct injection attacks.

Quoted injection attacks

For the first test append a single quotation mark, a space and the word "OR" to a parameter's legitimate value. For the second test insert a single quotation mark, a space and an ampersand or its Unicode equivalent at the tail end of the parameter. If either test produces an error, the server is vulnerable to direct injection attacks.

Once the tester knows which type of Select attack the system is vulnerable he/she can:

- Use a number of the subspecies of this attack type to retrieve data using parenthetical operators to create sub queries.
- Use the UNION clause to create and retrieve data from a different guessed table.
- Use a LIKE clause to retrieve data that ordinarily would not be delivered by the server.
- Use AND or WHERE statements to return additional columns.
- Insert break statements using the semicolon (;) to execute his/her queries.

For a SQL server database, a tester can insert a commented entry into the SQL statement using a semicolon followed by two hyphens (;--). In Oracle commented entries can be used and the tester may consider using the DUAL function to retrieve

data. A tester should not be as concerned with what can be retrieved, but more concerned with whether or not it can be retrieved. If the tester can violate the system, even slightly, it is a serious defect and needs to be corrected as an attacker may detect it also.

For the specifics of conducting each type of attack refer to the white papers mentioned earlier.

Testing for INSERT statement attacks –

An insert statement attack is one in which the attacker is able to insert data directly into the system for his or her purposes. To test for the INSERT vulnerability, manipulate the arguments in the VALUES clause to retrieve other data. Do this using a select sub-query. For example, take a registration form that sends an email to the registrant when the form is completed in addition to displaying the information on the confirmation page.

Fill out the form like this:

Name: ' + (SELECT TOP 1 FieldName FROM TableName) + '
Email: someone@email.com
Phone: 333-333-3333

That would make the SQL statement look like this:

```
INSERT INTO TableName VALUES (" + (SELECT TOP 1 FieldName  
FROM  
TableName) + ", "someone@email.com", '333-333-3333')
```

When the confirmation page or email is generated the sub-query will return the record asked for, not the input that was submitted. An attacker would continue to complete the registration page altering only the record number in the sub-query, thus scrolling through the data one record at a time. Be aware that unless it is specifically limited to return only one record, the query will return an error. If querying an oracle database use the ROWNUM function rather than TOP.

Testing for SQL Server Stored Procedure attacks –

An out-of-the-box install of Microsoft SQL Server has over one thousand stored procedures. If SQL injection is working on a web application that uses SQL Server as its backend, depending on the permissions of the web application's database user, these stored procedures can be used to pull off some remarkable feats.

The first thing to know about stored procedure injection is that there is a good chance the stored procedure's output will not be visible in the same way values get returned with regular injection. Depending on the goals, there may not be a need to get data back at all. It may be possible to inject spurious data into the database with the intention of disrupting normal business. If necessary, an attacker or a tester can find another means of getting the data. Inserted data may show up in regular correspondence, billing statements or email, showing the success of the injection. A

tester of course may have access to the database and can perform queries separately to see if the tests were successful.

Procedure injection can be just as easy as a regular query injection. Procedure injection should look something like this:

```
simplequoted.asp?city=Seattle';EXEC master.dbo.xp_cmdshell  
'cmd.exe dir c:
```

Notice how a valid argument is supplied at the beginning followed by a single quote to interrupt the query. A semicolon is used to terminate the original system-sponsored SQL query. The system then executes the stored procedure. The final argument to the stored procedure has no closing quote.

The example above will satisfy the syntax requirements inherent in most systems vulnerable to SQL injection attacks. When executing this, the tester can include parentheses, additional WHERE statements, etc. There usually is no column matching or data types to worry about, making it possible to exploit the vulnerability in a manner similar to working with applications that do not return error messages.

There are two standard SQL Server stored procedures that should be included in testing:

MASTER.DBO.XP_CMDSHELL takes a single argument. This command should be executed at SQL Server's user level. It is not likely to be available unless the web application has SQL Server admin access.

SP_MAKEWEBTASK takes a query and builds a web page containing its output. Note that a UNC pathname can be used as an output location. This means that the output file can be placed on any system connected to the Internet that has a publicly writeable share on it.

In Oracle databases the equivalents to stored procedures are Views.

Evaluating the results of testing for SQL injection

If a tester receives a database server error message of some kind as a result of attempting SQL injection, the injection was definitely successful insofar as it demonstrates the system's vulnerability. However, database error messages aren't always obvious. A tester should look in every possible place for evidence of successful injection, just like a would-be attacker.

In performing test evaluations the thing to do is search through the entire source of the returned page for phrases like "ODBC", "SQL Server", "Syntax", etc. Additional details on the nature of the error can be in hidden input, comments, etc. To locate this information, view the page source code using the web browser's "View Source" and check the response headers. Many web applications on production systems give error messages with absolutely no information in the body of the response, but contain

database error message in a header. These kinds of features are built into the application for debugging and QA purposes, and not removed or disabled before release.

A tester should look not only on the immediately returned page, but in linked pages as well. In some cases web applications that return generic error message pages in response to a SQL injection attack will have a link to another page. That page may give the full SQL Server error message. Again, this is probably a feature intended for use in debugging, but it can easily be put to use by attackers to determine the success or failure of their efforts.

Another thing the tester should watch out for is a 302-page redirect. Some servers are configured not to return error messages. A attack may be successful even if an ODBC error message is not sent back. Often a server will return a properly formatted, seemingly generic error message page stating that there was "an internal server error" or a "problem processing your request."

Some web applications are built so that in the event of an error of any kind, the client is returned to the site's main page. When the server returns an HTTP 500 Error page back, chances are that injection is successful. Many sites have a default HTTP 500 Internal Server Error page that claims that the server is down for maintenance, or that politely asks the user to email their request to their support staff. It can be possible to take advantage of these sites using stored procedure techniques, which were discussed earlier.

Defenses against SQL injection attacks

Defending against SQL injection attacks should incorporate all of the following:

1. Test the application thoroughly and employ the use of a vulnerability scanner in the QA environment as well as in the production environment. A good vulnerability scanner will test the forms and input parameters of the web application for its susceptibility to a SQL injection based attack. In addition to checking for SQL Injection, a vulnerability scanner will test the application for a wide range of other attack types including: parameter manipulation, Unicode attacks, directory traversal and a host of other techniques used by attackers. Some scanners even allow for the development of checks unique to the application through the use of an Integrated Development Environment type editor.
2. Use server-side validation checks that validate the data from the input fields, checking for conformity to intended as well as valid data inputs for the field. These checks are necessary to ensure that the client-side checks created for speed and usability have not been bypassed. Server-side checks are also useful for verifying that cookies have not been manipulated in an attempt to obtain access to systems and data.
3. Educate the testers and developers on web application security. Personnel don't need to be IT security gurus, but they should be familiar with the terminology and

be able to competently execute attacks against systems to identify vulnerabilities before attackers do.

4. Perform client-side checks to prevent inadvertent submission of illegal character types and to improve the usability of the web application. A determined attacker can easily bypass client-side data checks. These data checks add a measure of polish to the web application that is necessary when competing in the marketplace for customers.
5. Capture and redirect all ODBC or server related error messages to either a generic HTTP 404 error message page, or the default page for the site. While using 403 redirects can be defeated, other defenses may not always be applied. Programmers may inadvertently neglect to add the other defenses needed when designing new pages or redesigning old pages for the application. Having a generic message serve up any time the server tries to return an HTTP status other than 200 OK prevents the attacker who bypasses the applications other defenses from absolutely knowing what happened and whether or not they succeeded. The tester needs to confirm the message is sanitized by using a browser set to show non-friendly http error messages

Bad message	Good message
Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft][ODBC Microsoft Access Driver] Syntax error in string in query expression 'user = '''. /login1.asp, line 10	Thank you, ' for visiting our site! Click here to return to the sites home page

Conclusion:

SQL injection is an easily diagnosed web application vulnerability once the tester knows the basic techniques. The vulnerability, if it exists, can be exploited in many ways and provide access to sensitive data through the normal traffic channels, sometimes without getting flagged by an intrusion detection systems. There are a number of different types of SQL injection attacks and they can be used successfully against almost any relational database system. Defenses to SQL injection can be implemented fairly easily, and multiple layers of defense should be implemented to keep a determined attacker from being successful with this avenue of attack.