Notes on Active Server Pages (ASP) and MS-SQL Server Integration

Prepared by: Richard Grime
Contributor: Chris Roberts
Reviewed: Tom Weil
Introduction

Using Active Server Pages (based on either Microsoft IIS, or Apache running Sun ONE\(^1\) or the Perl Module Apache::ASP\(^2\)), a secure and reliable method provide a database driven website can be produced.

This document provides guidelines for such a system, based on the security standpoint. It is written assuming that the web server is IIS 5.0 and that the database is running on Microsoft SQL Server 2000.

Site Structure

Data can either be entered into / retrieved from the database anonymously, or on a per user basis (authenticated). Under the anonymous regime, security can only apply to the whole database. If working on an authenticated basis, then specific data can be protected and only delivered to the correct user.

For example:

**Web Guestbook – an anonymous database driven website.**
A guestbook is added to a website and driven from an SQL server. Any anonymous user can view the entire guestbook. They can also write a “new” entry into the database.

No “edit” facility is available (as we cannot tell which user wrote which comment), nor can we confirm the identity of the person submitting the new entry is real.

**Web Magazine Subscriptions – an authenticated database driven website.**
To allow subscribers to change their delivery address, a magazine company adds a web page to their website.

The user must log into a secure (HTTPS) website, where their password is checked. They are then given access to their own subscription record. They cannot write over anyone else’s data – but have full access to their own.

Before writing any code or designing any databases, the decision as to which model best fits your website should be taken.

User Authentication

In order to provide an authenticated web site, we must have a mechanism by which we can identify the user.

College websites use the user’s college ICT logon account – the web server (IIS) then checks with the IC Active Directory to confirm that the users’ password is correct.

Once the identity (logon) has been established the user is allowed to continue into the site, and this logon name is stored in a Server Side\(^3\) variable.

\(^1\) http://wwws.sun.com/software/chilisoft/
\(^2\) http://www.apache-asp.org
\(^3\) http://wwws.sun.com/software/chilisoft/
Now the user has logged in, we can use the server side variable as an argument when talking to the database. However, we cannot talk to the database as if we were that user—that is, we cannot assume that users context. To run in the user’s context, we require a trusted connection.

**Trusted Connections**

We can use Windows Integrated Security\(^4\) to pass the logon name and password (credentials) straight through the web server and to the SQL server. This however has a few caveats:

- The web server must use basic (plain text) authentication. The web server cannot pass NTLM hashes to the SQL server for authentication. This means that the website must be based on HTTPS – for this you will require an SSL certificate.

- The SQL database must hold a valid account for that user. This means that either SQL Windows Authentication is being used, or a SQL Authentication account has been created. With either model, specific rights need to be assigned to each user individually. This system is unsuitable for a site which has a lot of users.

Trusted Connections should be used where the database permission structure is very complicated. For simple data retrieval sites, the management overhead involved with trusted connections is too great.

**Stored Procedures vs. Inline Code**

There are two ways of writing out the SQL commands to retrieve and insert data to and from the database. Code can be placed directly inside the web page, or isolated by placing it in a stored procedure.

Stored procedures are highly recommended for the following reasons:

- SQL Code itself is not visible in your web source code. SQL Code can reveal database structures.

- SQL code should always be kept in server side portions of the web page. However, a simple coding mistake could make these commands visible to the client.

- Keeping the code in a stored procedure keeps all the DB schema / operation related code in one place. This makes for far simple database maintenance and management.

Specifically, SQL code, both in terms of connection strings, and the commands themselves should never be placed in the client side portion of the code. Access to this SQL code may lead to a compromise of your data.

---

\(^3\) Server side variables are held on the web server, and cannot be written to by the client. Client side variables on the other hand are held in the browser – and can be changed at will by the user.

\(^4\) The Windows Integrated Security uses the Security Support Provider Interface (SSPI)
Assigning User Permissions

As with all permissions models, they require some planning. The principle is one of least possible. The fewer accounts, with the lowest permissions possible, that you have – the better.

You should also remember which user context you are assuming.

<table>
<thead>
<tr>
<th>Model</th>
<th>User Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous</td>
<td>IUSR_&lt;server&gt;, IWAM_&lt;server&gt; or hard coded</td>
</tr>
<tr>
<td>Authenticated – no Trusted</td>
<td>SYSTEM or hard coded</td>
</tr>
<tr>
<td>Authenticated – Trusted Connection</td>
<td>Authenticated User’s Login Name</td>
</tr>
</tbody>
</table>

In this case “hard coded” refers to the user name specified in the SQL Connection string. For example, this connection string connects using the “anon_user” account:

```
Driver={SQL Server}; Server=server_name; Database=database_name; UID=anon_user; PWD=password
```

You can see that the password for this account has to be stored in the Web source – which shows that the SQL code should never be accessible by the client.

Any accounts that are “hard coded” or defaults, like SYSTEM, IUSR or IWAM should have the least amount of permission to do their job.

For trusted connections, the string would look like:

```
Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=Calc;Data Source=NEWServer
```

In this case, no password or user name is stored – but bear in mind the caveats of trusted connections mentioned above.

### User Accounts for Stored Procedures

If you use stored procedures to provide data access for your site (as is recommended), then you can assign a SQL account to each stored procedure. Using the “hard coded” method mentioned above, each stored procedure then executes in a context of least possible permissions – which also fits the recommended security model.

Take the web guestbook example, we could have two stored procedures, say:

1) SP_ListGuestbookEntries
2) SP_WriteNewGuestbookEntry

Stored procedure 1 queries the database, and returns a recordset containing all of the guestbook entries. This only needs to read the database – no write permissions are required.

---

5 IIS uses these two accounts to represent the anonymous user – where <server> is the name of your web server. This is who
Hence, we could create a sql account *user_read* (although names shouldn’t be this obvious!). Once this user exists, we can use the connection string:

```
Driver={SQL Server}; Server=server_name; Database=database_name; UID=user_read; PWD=ReadTheDB
```

Now, for stored procedure 2, this can write only “new” entries to the guest book. Hence, the user (*user_write*) for this would only require *insert* permissions – no read, or update etc, is required.

```
Driver={SQL Server}; Server=server_name; Database=database_name; UID=user_write; PWD=WriteTheDB
```

Note that all these models can be mixed. You may wish to use the “user_read” type scenario just for reading data out of the database. For writing back, you could use a Trusted Connection. This is particularly relevant for the following type of example:

**Web Guestbook with Administrator Feature**

A web guestbook is placed on a site, however the owner of the site would like to be able to remove certain entries via a web form.

*To allow the owner to securely log in as an “Administrator”, we use a trusted connection. This way, we avoid some of the caveats involved with trusted connections.*

The only SQL Account we have to maintain is that of the administrator. As the administrator’s credentials are never written down in the web code – there is no risk of breach.

**Maintaining Stateful Web Pages**

More advanced web sites can use a number of mechanisms to store session information. **Server Side Variables**

These are held at the web server. The client has no direct write access, so they cannot be modified by the browser. Commonly seen as ASP (VBScript) variables.

**Client Side Variables.**

The browser can also hold a number of variables. This should be considered untrusted, as they can be modified by the client. They are commonly seen in JavaScript variables.

If you use Client Side JavaScript code to validate an HTML form before posting back to your database, then you should also double check this validation back at the server. If you do not, then the client will be able to submit data that does not fit your validation rules – which compromises your data integrity.

**Query Strings**

These variables take the form of additions to the URL. While very convenient for programmers, they provide the most easily exploitable path for users. Consider this query string:
http://www.myweb.com/index.asp?sql_command="select user from users"

In this case, the query string called “sql_command” clearly contains some SQL script. Merely by changing this string, a user could access other data. Even slightly less obvious strings such as “user_name=joebloggs” can be modified very easily.

Query Strings should be avoided for anything other than harmless data. They should never contain information which is used to either connect to, or query, a database. They should always be treated as untrusted.

**Application Variables**

As a special variant of Server Side Variables, Application Variables provided the best way of storing usernames and passwords in hard coded connections. For example, the connection string might look like:

Driver={SQL Server}; Server=server_name; Database=database_name;
UID=Application("appvar1"); PWD=Application("appvar2")

The application variables are defined in the Global.ASA (under IIS). This way, you can store all the user names and passwords in a common location – making it easy to change the passwords used on a regular basis.

**File Extensions: Server Side Includes (SSI's)**

When using SSI's in ASP, you should take care to use file extensions that are understood by the ASP interpreter. For example, using the “.inc” file extension would cause its entire contents to be rendered in plain text to the browser.

The same applies for other pages. Do not store files on your servers Web Root that you do not wish to be accessed. Random files like “readme.txt” or “todo.txt” will be accessible to any viewer and may contain sensitive information.