ASP, Security & Perfomance

Christian Hassa

Meine Firma beschäftigt sich mit Microsoft-Technologie und dort insbesondere mit Contentmanagement-Systemen, und in unseren Anfängen 1997 haben wir auch in ASP programmiert. Der ASP-Artikel "Webdatenbank" (PCNEWS-76, Seite 66) gibt eine gute Einführung in die Programmierung von dynamischen Websites. Die Microsoft-Plattform wird jedoch leider oft wegen *Security*-Fragen attackiert, obwohl sie unserer Meinung die kostengünstigste und effizienteste Plattform zur Implementierung von kommerziellen Weblösungen ist, insbesondere mit Microsoft .NET. Ein Grund für die häufigen *Security*-Probleme auf der Microsoft-Plattform ist die Tatsache, dass sie sehr leicht installiert und programmiert werden kann und Anfänger oft mit der Adaption von Tutorials oder Beispielen als Vorlage Websites programmieren, die dann auch tatsächlich in einer Echtumgebung live funktionieren.

Der PCNEWS-Artikel ist so ein Beispiel, das sehr gut als Vorlage für die Erstellung eines dynamischen Websites für einen Anfänger dienen kann. Dabei wurde aber nicht an *Security* gedacht. Jemand, der auf Basis des Artikels eine Website programmiert, wird zuerst einmal froh sein, damit Ergebnisse produzieren zu können und sich ebenfalls keine Gedanken über *Security* machen.

Die größte Sicherheitslücke in dem beschriebenen Kode ist die Art und Weise, wie die Datenbankabfragen erzeugt werden. Dadurch, dass die Eingaben des Benutzers direkt in die SELECT-Query eingefügt werden, kann der Benutzer nicht nur Parameter für die *Query* angeben, sondern die *Query* beliebig modifizieren. Beispielsweise kann aus der *Query*:

```
SQLquery = "SELECT * " & _
     "FROM Redaktion " & _
     "WHERE Redaktion.ID='"+ID+"';"
```

durch die Eingabe von

```
';DELETE * FROM Redaktion;SELECT * FROM Redaktion WHERE
Redaktion.ID='1
```

folgende *Query* ausgeführt werden:

```
SELECT * FROM Redaktion WHERE Redaktion.ID='';
DELETE * FROM Redaktion;
SELECT * FROM Redaktion WHERE Readaktion.ID='1';
```

Das ist jetzt natürlich nur ein phantasieloses Beispiel, aber ich glaube, das Problem wird dadurch deutlich.

Statt dessen sollte man das Format der *ADO-Parameter-Queries* verwenden, wo die Queryparameter mit "?" im Querytext angegeben und danach Parameter-Objekte an die Query angehängt werden. Das sieht ungefähr so aus (Genaueres kann man der MSDN-Dokumentation zum Thema *ADO Command Object* entnehmen):

```
Set objCmd = Server.CreateObject("ADODB.Command")
objCmd.CommandText = "SELECT * FROM Redaktion WHERE Redaktion.ID=?"
objCmd.CommandType = adCmdText
'erzeugt ein ADODB.Connection Objekt und öffnet dieses
Set objConn = GetNewConnection
objCmd.ActiveConnection = objConn
Set objParm1 = objCmd.CreateParameter("ID", adInteger, _
     adParamInput, 4, CInt(Request.QueryString("ID"))
objCmd.Parameters.Append objParm1
Set objRs = objCmd.Execute
```

Neben dem Sicherheitsvorteil, hat diese Methode noch folgende wichtige Vorteile:

1. Bei der Konvertierung von *(untyped)* Benutzereingaben (die *posted* Parameter eines *Webrequests* sind ja nur *untyped name/value pairs*) auf die *strongly typed arguments* einer Datenbank (zum Beispiel bei einem *Datetime field*) gibt es weniger Probleme. Sobald man aus der Benutzereingabe einen Variant vom *Subtype Date* gemacht hat, kann bei der Übergabe des Parameters in die Datenbank nichts mehr passieren (zum Beispiel bei unterschiedlichen *Locale*-einstellungen am Datenbankserver und am Webserver, usw...).

2. *Parameterqueries* sind wesentlich schneller (mindestens um den Faktor 10). Sogar in einer Jet-Datenbank können Sie *Parameterqueries* vorab anlegen und Access errechnet den *Execution Plan* für diese *Queries* vorab. Noch mehr Vorteile ergeben sich natürlich mit einer SQL *Server Engine*, vor allem wenn man *Stored Procedures* verwendet.

3. Der Code ist leserlicher und damit besser wartbar.

Weitere Dinge, die man erwähnen sollte:

1. Eine Access-Datenbank ist am Webserver sehr ungeeignet. Mit MSDE stellt Microsoft eine SQL Server basierende *Data Engine* zur Verfügung, die kostenlos ist, und über Access wunderbar administriert werden kann. Im Gegensatz zur häufig landläufig angetroffenen Meinung kann MSDE beliebig viele parallele *Connections* verwalten, lediglich ab 4 **gleichzeitig** ausgeführten *Queries* wird die Ausführung weiterer *Queries* gequeued, aber nicht verhindert. Bei dieser Auslastung hat man meistens schon eine Last von 100 Benutzern die online sind, und Access ist bei dieser Auslastung sowieso schon überfordert. MSDE ist technisch ident zur *SQL Server Engine* und kann kostenlos eingesetzt werden. Neben der zuvor beschriebenen Performancebremse fehlt lediglich der *Enterprise Manager* (samt dazugehörigen SQLDMO und SQLNS) sowie einige Replikationsfeatures. Wichtige Features wie Transaktionen und *Stored Procedures* stehen aber zur Verfügung.

2. `Application.Lock` sollte eigentlich nur verwendet werden, wenn man in den *Application State* schreibt, und nicht, um das Locking für eine Datenbank zu synchronisieren. Der *Application State* sollte überhaupt nur sehr sparsam verwendet werden, da man sich sonst um die Synchronisation kümmern muss. Am ehesten eignet sich der *Application State*, um während des `Application_OnStart` Events in einen *Readonly Cache* zu initialisieren. In diesem Zusammenhang könnte man auch noch die Evente `Sesion_OnStart` und `Application_OnStart` erwähnen.

3. Um das *ADO Connectionpooling* effizient arbeiten zu lassen, empfiehlt es sich, die *Connection* möglichst schnell explizit wieder freizugeben, spätestens am Ende der ASP-Seite mit `Set objConn = Nothing`. Außerdem ist es eine gute Angewohnheit, alle instantierten Objekte am Ende der Seite explizit wieder auf `Nothing` zu setzen.

4. Bei komplexerer Logik empfiehlt es sich, die Businesslogik in VB COM Objekte zu kapseln und mit ASP lediglich die Presentationsschicht der Webapplikation zu implementieren (entsprechend des Konzepts einer 3-tier Anwendung). Während der Umstieg von VBScript auf VB nicht besonders schwierig ist, ergeben sich dadurch folgende Vorteile:

   a. Das mächtigere VB Programmiermodell (*strongly typed Variablen*, ...)
   b. *Performance*
   c. Wiederverwendbarkeit der Businesslogik durch Kapselung in COM-Komponenten
   d. Die Businesslogik wird nur kompiliert übergeben und kann nicht ohne den Sourcecode modifziert werden
   e. Verwendung von COM+ Services z.B. für Transaktionen (das geht zwar auch von ASP aus, aber das macht nicht wirklich Sinn)
   f. VB-Komponenten können über den Objektkontext direkt auf die ASP-Objekte zugreifen

5. Für die Verwendung des Session Objekts sind folgende Hinweise wichtig:
   a) Man soll darin keine *Apartment Threaded Objekte* (=alle VB6 Objekte) speichern, da diese *Threadaffinity* verursachen

   b) Die *Session* sollte nur sparsam verwendet werden, da sie Ressourcen am Server belegt
   c) *Sessionstate* kann abgeschalten werden, wenn er nicht verwendet wird (bringt *Performance*)
   d) Für ein *Scaling out Szenario* ist das ASP Session Objekt nicht geeignet, da nicht über mehrere Webserver synchronisiert werden kann

   e) Daher ist der meiste *Session State* besser in einer Datenbank aufgehoben

Zuletzt sollte man der Vollständigkeit halber auch noch auf die neue Microsoft.NET Plattform hinweisen, die mit ASP.NET eine viel komfortablere und mächtigere Programmierumgebung zur Verfügung stellt als ASP. Für Redaktionssysteme ist in den letzten Jahren auch die Verwendung von XML/XSL üblich geworden, da dadurch eine bessere Trennung von Design und Content möglich wird.