Erstellen sicherer ASP.NET-Anwendungen

Authentifizierung, Autorisierung und sichere Kommunikation

Kapitel 12 – Datenzugriffssicherheit

J.D. Meier, Alex Mackman, Michael Dunner und Srinath Vasireddy

Microsoft Corporation

Oktober 2002

Auf der Orientierungsseite finden Sie einen Ausgangspunkt und eine vollständige Übersicht zum *Erstellen sicherer ASP.NET-Anwendungen*.

Zusammenfassung

In diesem Kapitel werden Empfehlungen und Anleitungen behandelt, die beim Entwickeln einer sicheren Datenzugriffsstrategie hilfreich sind. Die behandelten Themen umfassen die Verwendung der Windows-Authentifizierung von ASP.NET für Datenbanken, das Sichern von Verbindungszeichenfolgen, das sichere Speichern von Anmeldeinformationen in einer Datenbank, das Schützen vor SQL Injection-Angriffen sowie das Verwenden von Datenbankrollen.

Inhalt

Wenn Sie webbasierte Anwendungen erstellen, ist es wichtig, dass Sie einen sicheren Ansatz verwenden, um auf die Daten zuzugreifen und sie zu speichern. In diesem Kapitel werden einige Hauptthemen des Datenzugriffs behandelt. Es wird Ihnen bei folgenden Aktionen behilflich sein:

- Auswählen zwischen der Authentifizierung durch das Betriebssystem Microsoft® Windows® und der SQL-Authentifizierung, wenn eine Verbindung zu SQL Server™ hergestellt wird.
- Sicheres Speichern von Verbindungszeichenfolgen.
- Entscheiden, ob der Sicherheitskontext des ursprünglichen Aufrufers an die Datenbank übermittelt wird.
- Nutzen des Verbindungspoolings.
- Schützen vor SQL Injection-Angriffen.
- Sicheres Speichern von Anmeldeinformationen in einer Datenbank.

In diesem Kapitel werden auch verschiedene Kompromisse dargestellt, die sich auf die Verwendung von Rollen beziehen, z. B. Rollen in der Datenbank im Vergleich zur Rollenlogik, die auf der mittleren Ebene angewendet wird. Abschließend wird eine Reihe von wichtigen Empfehlungen für den Datenzugriff präsentiert.

Einführen der Datenzugriffssicherheit

In Abbildung 12.1 werden die dem Datenzugriff zugeordneten Hauptsicherheitsthemen dargestellt.

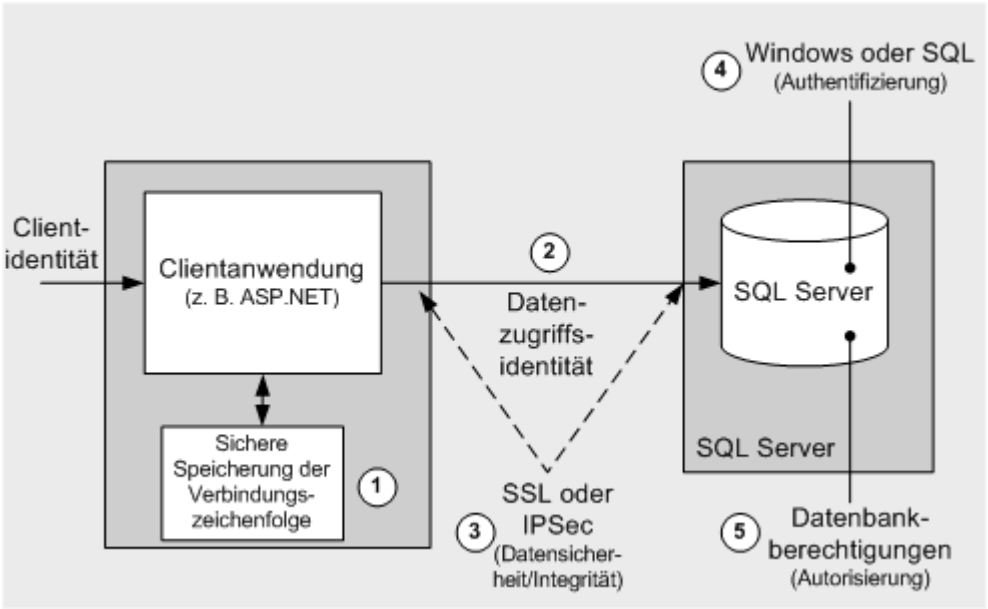

Abbildung 12.1
Hauptsicherheitsthemen für den Datenzugriff

Die in Abbildung 12.1 dargestellten und im gesamten Kapitel beschriebenen Hauptthemen sind nachfolgend zusammengefasst:

1. **Sicheres Speichern von Datenbank-Verbindungszeichenfolgen -** Dies ist besonders von Bedeutung, wenn die Anwendung die SQL-Authentifizierung verwendet, um die Verbindung zu SQL Server oder zu Datenbanken herzustellen, die nicht von Microsoft stammen und die explizite Anmeldeinformationen erfordern. In diese Fällen enthalten die Verbindungszeichenfolgen unverschlüsselte Benutzernamen und Kennwörter.

2. **Verwenden von geeigneten Identitäten für den Zugriff auf die Datenbank -** Der Datenzugriff kann unter Verwendung der Prozessidentität des aufrufenden Prozesses oder mithilfe einer oder mehrerer Dienstidentitäten oder mit der Identität des ursprünglichen Aufrufers (über den Identitätswechsel oder die Delegierung) durchgeführt werden. Die Auswahl wird durch Ihr Datenzugriffsmodell bestimmt – vertrauenswürdiges Subsystem oder Identitätswechsel/Delegierung.
3. **Sichern von Daten, die über das Netzwerk gesendet werden -** Beispielsweise das Sichern von Anmeldeinformationen und vertraulicher Daten, die an und von SQL Server gesendet werden.

Hinweis: Anmeldeinformationen werden nur dann im Netzwerk offen gelegt, wenn Sie die SQL-Authentifizierung verwenden (nicht bei der Windows-Authentifizierung).

SQL Server 2000 unterstützt SSL mit Serverzertifikaten. IPSec kann ebenfalls zum Verschlüsseln des Datenverkehrs zwischen dem Clientcomputer (z. B. ein Web- oder Anwendungsserver) und dem Datenbankserver verwendet werden.

4. **Authentifizieren von Aufrufern in der Datenbank -** SQL Server unterstützt die Windows-Authentifizierung (unter Verwendung von NTLM oder Kerberos) und SQL-Authentifizierung (mithilfe des in SQL Server integrierten Authentifizierungsmechanismus).
5. **Autorisieren von Aufrufern in der Datenbank -** Einzelnen Datenbankobjekten werden Berechtigungen zugeordnet. Die Berechtigungen können zu Benutzern, Gruppen oder Rollen zugeordnet werden.

SQL Server-Gatekeeper

In Abbildung 12.2 werden die hauptsächlichen Gatekeeper für den Datenzugriff auf SQL Server hervorgehoben.

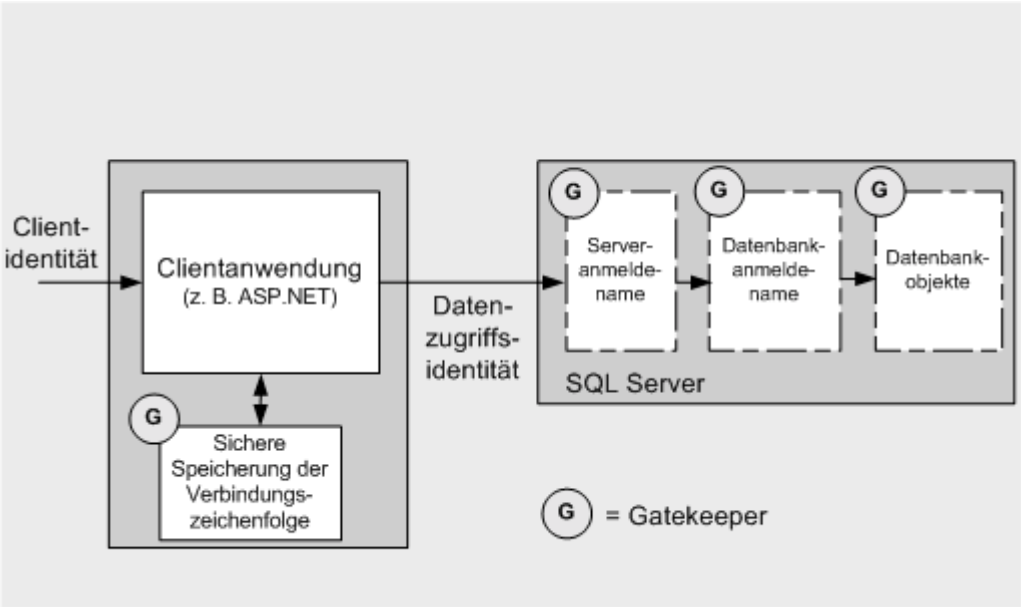

Abbildung 12.2
SQL Server-Gatekeeper

Nachfolgend sind die wichtigsten Gatekeeper aufgeführt:

- Der ausgewählte Datenspeicher, der zum Verwalten der Datenbank-Verbindungszeichenfolge verwendet wird.
- Der SQL Server-Benutzername (durch den Servernamen bestimmt, der in der Verbindungszeichenfolge angegeben ist).

- Der Datenbankanmeldename (durch den Datenbanknamen bestimmt, der in der Verbindungszeichenfolge angegeben ist).
- Einzelnen Datenbankobjekten zugeordnete Berechtigungen.
  Die Berechtigungen können zu Benutzern, Gruppen oder Rollen zugeordnet werden.

Vertrauenswürdiges Subsystem und Identitätswechsel/Delegierung

Der feinstufige Zugriff auf die Datenbank ist einer der zu beachtenden Hauptfaktoren. Sie müssen überlegen, ob Sie die Autorisierung auf Benutzerebene für die Datenbank wünschen (die das Modell von Identitätswechsel/Delegierung erfordert) oder ob Sie die Anwendungsrollenlogik auf der mittleren Ebene Ihrer Anwendung verwenden können, um Benutzer zu autorisieren (erfordert das Modell mit vertrauenswürdigen Subsystemen).

Wenn die Datenbank die Autorisierung auf Benutzerebene erwartet, müssen Sie für den ursprünglichen Aufrufer einen Identitätswechsel durchführen. Obwohl das Modell von Identitätswechsel/Delegierung unterstützt wird, wird empfohlen, das Modell mit vertrauenswürdigen Subsystemen zu verwenden, bei dem der ursprüngliche Aufrufer am IIS-/ASP.NET-Gate überprüft, einer Rolle zugeordnet und dann auf Basis der Rollenmitgliedschaft autorisiert wird. Die Systemressourcen für die Anwendung werden dann unter Verwendung von Dienstkonten auf Anwendungs- oder Rollenebene autorisiert, oder es wird die Prozessidentität der Anwendung verwendet (z. B. das Konto ASPNET).

Diese beiden Modelle sind in Abbildung 12.3 dargestellt.

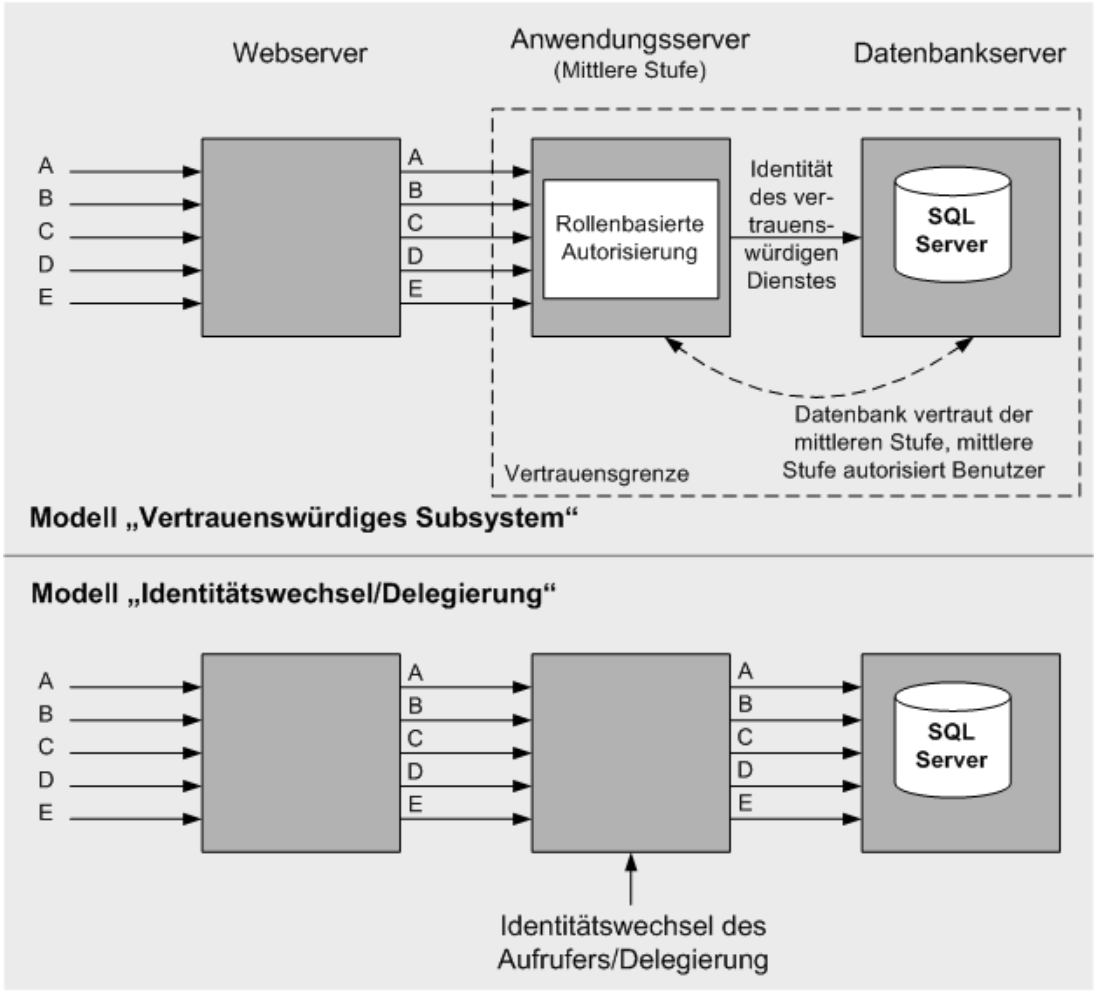

Abbildung 12.3

Das Modell mit vertrauenswürdigen Subsystemen und das Modell mit Identitätswechsel/Delegierung für den Datenbankzugriff

Sie sollten eine Reihe von Hauptfaktoren bedenken, wenn Sie die Verbindung zu SQL Server für den Datenzugriff herstellen. Diese werden unten zusammengefasst und in nachfolgenden Abschnitten näher ausgeführt.

- **Welcher Authentifizierungstyp sollte verwendet werden**? Die Windows-Authentifizierung bietet eine umfangreichere Sicherheit, wobei Firewalls und Probleme mit nicht vertrauenswürdige Domänen die Verwendung der SQL-Authentifizierung erfordern können. Wenn dies der Fall ist, sollten Sie sicherstellen, dass die Verwendung der SQL-Authentifizierung durch Ihre Anwendung so sicher wie möglich erfolgt, wie weiter unten in diesem Kapitel unter "SQL-Authentifizierung" beschrieben wird.
- **Einzelne Benutzerrolle im Vergleich zur mehreren Benutzerrollen -** Muss die Anwendung auf SQL mithilfe eines einzelnen Kontos zugreifen, das in der Datenbank über einen festen Satz an Berechtigungen verfügt, oder sind mehrere (rollenbasierte) Konten in Abhängigkeit vom Benutzer der Anwendung erforderlich?
- **Aufruferidentität -** Muss die Datenbank die Identität des ursprünglichen Aufrufers über den Aufrufkontext erhalten, um entweder die Autorisierung oder die Überwachung durchzuführen, oder können Sie eine bzw. mehrere vertrauenswürdige Verbindungen verwenden und die Identität des ursprünglichen Aufrufers auf der Anwendungsebene übergeben?

  Damit das Betriebssystem die Identität des ursprünglichen Aufrufers übermitteln kann, erfordert sie Identitätswechsel/Delegierung auf der mittleren Ebene. Dadurch wird die Effektivität des Verbindungspoolings erheblich verringert. Das Verbindungspooling ist weiterhin aktiviert, führt jedoch zu vielen kleinen Pools (für jeden einzelnen Sicherheitskontext) und das bei geringer oder keiner Wiederverwendung von Verbindungen.
- **Werden vertrauliche Daten an den und vom Datenbankserver gesendet**? Obwohl die Windows-Authentifizierung bedeutet, dass Sie keine Benutzeranmeldeinformationen über das Netzwerk an den Datenbankserver übergeben, sollten Sie vertrauliche Anwendungsdaten (z. B. Mitarbeiter- oder Gehaltsinformationen) über IPSec oder SSL sichern.

Authentifizierung

In diesem Abschnitt wird beschrieben, wie Clients für SQL Server authentifiziert werden und Sie eine Identität auswählen, die für den Datenbankzugriff innerhalb der Clientanwendungen verwendet wird, bevor die Verbindung zu SQL Server hergestellt wird.

Windows-Authentifizierung

Die Windows-Authentifizierung ist aus folgenden Gründen sicherer als die SQL-Authentifizierung:

- Die Anmeldeinformationen werden für Sie verwaltet und nicht über das Netzwerk übertragen.
- Sie vermeiden das Einbetten von Benutzernamen und Kennwörtern in Verbindungszeichenfolgen.
- Die Anmeldesicherheit wird über Ablaufzeiten für Kennwörter, minimale Länge und Kontensperrung nach mehreren ungültigen Anmeldeanforderungen verbessert. Dadurch wird die Bedrohung durch Verzeichnisangriffe (Wörterbuchangriffe) verringert.

Verwenden Sie die Windows-Authentifizierung in den folgenden Szenarien:

- Sie haben das Modell mit vertrauenswürdigen Subsystemen verwendet und stellen die Verbindung zu SQL Server über eine einzelne feste Identität her. Wenn Sie die Verbindung über ASP.NET herstellen, wird hierbei davon ausgegangen, dass die Webanwendung nicht für den Identitätswechsel konfiguriert ist.

  Verwenden Sie in diesem Szenario die ASP.NET-Prozessidentität oder eine Serviced Component-Identität (abgerufen von dem Konto, das zum Ausführen einer Enterprise Services-Serveranwendung verwendet wurde).

- Sie delegieren den Sicherheitskontext des ursprünglichen Aufrufers absichtlich, indem Sie die Delegierung verwenden (und sind dafür bereit, die Anwendungsskalierbarkeit zu opfern, indem Sie auf das Datenbankverbindungspooling verzichten).

Bedenken Sie die folgenden wichtigen Punkte, wenn Sie die Windows-Authentifizierung verwenden, um die Verbindung zu SQL Server herzustellen:

- Verwenden Sie für das ASP.NET-Prozesskonto das Prinzip der minimalen Rechte. Vermeiden Sie es, dem ASP.NET-Prozesskonto das Recht **Als Teil des Betriebssystems handeln** zu gewähren, um **LogonUser**-API-Aufrufe zu aktivieren.
- Ermitteln Sie, welcher Code zusätzliche Rechte erfordert, und positionieren Sie diesen in Serviced Components, die in prozessexternen Enterprise Services-Anwendungen ausgeführt werden.

Weitere Informationen

Weitere Informationen zum Zugreifen auf Netzwerkressourcen über ASP.NET sowie zum Auswählen und Konfigurieren eines entsprechenden Kontos zum Ausführen von ASP.NET finden Sie in Kapitel 8 "ASP.NET-Sicherheit".

Verwenden der Windows-Authentifizierung

Ihnen bieten sich die folgenden Optionen, wenn Sie die Windows-Authentifizierung zum Herstellen der Verbindung zu SQL Server über eine ASP.NET-Anwendung (oder einen Webdienst oder eine Remotekomponente, für die ASP.NET als Host verwendet wird) verwenden.

- Verwenden der ASP.NET-Prozessidentität.
- Verwenden von festen Identitäten in ASP.NET.
- Verwenden von Serviced Components.
- Verwenden der API **LogonUser** und Durchführen eines Identitätswechsels für eine bestimmte Identität.
- Verwenden der Identität des ursprünglichen Aufrufers.
- Verwenden des anonymen Internetbenutzerkontos.

Empfehlung

Es wird empfohlen, die lokale ASP.NET-Prozessidentität durch Ändern des Kennworts auf dem Server in einen bekannten Wert zu konfigurieren und ein gespiegeltes Konto auf dem Datenbankserver zu erstellen, indem Sie einen lokalen Benutzer mit demselben Namen und demselben Kennwort erstellen. Weitere Einzelheiten zu diesem und anderen Ansätzen finden Sie nachfolgend.

Verwenden der ASP.NET-Prozessidentität

Wenn Sie die Verbindung zu SQL Server direkt über eine ASP.NET-Anwendung (oder Webdienst oder eine Remotekomponente, die von ASP.NET verwaltet wird) herstellen, verwenden Sie die ASP.NET-Prozessidentität. Hierbei handelt es sich um einen häufigen Ansatz, und die Anwendung definiert die Vertrauensgrenzen, d. h. die Datenbank vertraut dem ASP.NET-Konto beim Zugriff auf die Datenbankobjekte.

Ihnen bieten sich drei Optionen:

- Verwenden von gespiegelten lokalen ASPNET-Konten.
- Verwenden von gespiegelten, benutzerdefinierten lokalen Konten.
- Verwenden eines benutzerdefinierten Domänenkontos.

Verwenden von gespiegelten lokalen ASPNET-Konten

Dies ist der einfachste Ansatz, der im Allgemeinen verwendet wird, wenn Sie Eigentümer der Zieldatenbank sind (und die Verwaltung lokaler Datenbankserverkonten steuern können). Mit dieser Option verwenden Sie das lokale ASPNET-Konto mit minimalen Rechten, um ASP.NET auszuführen, und erstellen dann ein dupliziertes Konto auf dem Datenbankserver.

Hinweis: Dieser Ansatz bietet zusätzlich den Vorteil, dass er über nicht vertrauenswürdige Domänen hinweg und durch Firewalls hindurch funktioniert. Der Firewall öffnet möglicherweise nicht ausreichend Ports, um die Windows-Authentifizierung zu unterstützen.

Verwenden von gespiegelten, benutzerdefinierten lokalen Konten

Dieser Ansatz gleicht dem vorherigen Ansatz, ausgenommen der Tatsache, dass Sie nicht das ASPNET-Standardkonto verwenden. Dies bedeutet zweierlei:

- Sie müssen ein benutzerdefiniertes lokales Konto mit den entsprechenden Berechtigungen und Rechten erstellen.
  Weitere Informationen finden Sie unter "Vorgehensweise: Erstellen eines benutzerdefinierten Kontos zum Ausführen von ASP.NET" im Abschnitt "Referenz" dieses Handbuchs.
- Sie verwenden nicht länger das durch den .NET Framework-Installationsprozess erstellte Standardkonto. Ihr Unternehmen besitzt möglicherweise eine Richtlinie, die die Verwendung von Standardinstallationskonten nicht gestattet. Dadurch kann sich die Sicherheit Ihrer Anwendung möglicherweise erhöhen.
  Weitere Informationen finden Sie unter "Sans Top 20, W7 – Accounts with No Passwords or Weak Passwords" (http://www.sans.org/top20.htm, englischsprachig).

Verwenden eines benutzerdefinierten Domänenkontos

Dieser Ansatz gleicht dem vorherigen mit der Ausnahme, dass Sie ein Domänenkonto mit minimalen Rechten anstatt eines lokalen Kontos verwenden. Dieser Ansatz setzt voraus, dass sich die Client- und Servercomputer in derselben oder in vertrauten Domänen befinden. Der Hauptvorteil ist, dass die Anmeldeinformationen nicht computerübergreifend verwendet werden. Die Computer gewähren lediglich dem Domänenkonto den Zugriff. Ebenso gestaltet sich die Verwaltung mit Domänenkonten einfacher.

Implementieren einer gespiegelten ASPNET-Prozessidentität

Sie müssen die folgenden Aktionen durchführen, um gespiegelte Konten zum Herstellen der Verbindung von ASP.NET zu einer Datenbank zu verwenden:

- Verwenden Sie den Benutzer-Manager auf dem Webserver, um das Kennwort des ASPNET-Kontos auf einen bekannten starken Kennwortwert zurückzusetzen.

  Wichtig: Wenn Sie das ASPNET-Kennwort in einen bekannten Wert ändern, stimmt das Kennwort in der LSA (Local Security Authority) auf dem lokalen Computer nicht länger mit dem in der Windows-SAM (Security Account Manager)-Datenbank gespeicherten Kennwort überein. Wenn Sie zum Standardwert **AutoGenerate** zurückkehren möchten, müssen Sie Folgendes durchführen:

  Führen Sie **Aspnet_regiis.exe** aus, um ASP.NET in seine Standardkonfiguration zurückzusetzen. Weitere Informationen finden Sie in der Microsoft Knowledge Base im Artikel Q306005, "HOWTO: Repair IIS Mapping After You Remove and Reinstall IIS" (US). Wenn Sie so vorgehen, erhalten Sie ein neues Konto und eine neue Windows-SID (Security Identifier). Die Berechtigungen für dieses Konto werden auf ihre Standardwerte zurückgesetzt. Daher müssen Sie die Berechtigungen und Rechte explizit erneut zuweisen, die Sie ursprünglich für das alte ASPNET-Konto festgelegt haben.

- Setzen Sie das Kennwort explizit in der Datei **Machine.config** fest.

```
<processModel userName="machine" password="YourStrongPassword"    .
```

  - Sie sollten die Datei **Machine.config** vor unberechtigten Zugriffen schützen, indem Sie Windows-Zugriffssteuerungslisten verwenden. Gewähren Sie z. B. nicht dem anonymen IIS-Internetbenutzerkonto den Zugriff auf **Machine.config**.
- Erstellen Sie auf dem Datenbankserver ein gespiegeltes Konto (mit demselben Benutzernamen und demselben Kennwort).
- Erstellen Sie in der SQL-Datenbank einen Serverbenutzernamen für das lokale ASPNET-Konto, und ordnen Sie diesem Benutzernamen dann ein Benutzerkonto innerhalb der erforderlichen Datenbank zu. Erstellen Sie dann eine Datenbank-Benutzerrolle, fügen Sie den Datenbankbenutzer zur Rolle hinzu, und konfigurieren Sie die entsprechenden Datenbankberechtigungen für die Rolle.
  Weitere Informationen finden Sie weiter unten in diesem Kapitel unter "Erstellen eines Datenbankkontos mit minimalen Rechten".

Herstellen der Verbindung zu SQL Server mithilfe der Windows-Authentifizierung

So stellen Sie die Verbindung zu SQL Server mithilfe der Windows-Authentifizierung her:

- Verwenden Sie in der Clientanwendung eine Verbindungszeichenfolge, die entweder **Trusted Connection=Yes** oder **Integrated Security=SSPI** enthält. Die beiden Zeichenfolgen sind äquivalent und führen beide zur Windows-Authentifizierung (vorausgesetzt, SQL Server ist für die Windows-Authentifizierung konfiguriert). Beispiel:

```
"server=MySQL; Integrated Security=SSPI; database=Northwind"
```

Hinweis: Die Identität des Clients, der die Anforderung erstellt (d. h. der von SQL Server authentifizierte Client), wird vom Threadidentitätswechseltoken (wenn der Thread momentan den Identitätswechsel durchführt) oder vom Prozesstoken des Clients bestimmt.

Verwenden von festen Identitäten in ASP.NET

Bei diesem Ansatz konfigurieren Sie Ihre ASP.NET-Anwendung so, dass für eine angegebene, feste Identität ein Identitätswechsel durchgeführt wird, indem das folgende Element in der Datei **Web.config** verwendet wird:

```
<identity impersonate="true"
          userName="YourAccount"
          password="YourStrongPassword" />
```

Dies wird zur Standardidentität, die beim Herstellen der Verbindung zu Netzwerkressourcen (einschließlich Datenbanken) verwendet wird.

Dieser Ansatz wird für .NET Framework, Version 1.0, aus zwei Gründen nicht empfohlen:

- Benutzernamen und Kennwörter befinden sich unverschlüsselt im Webbereich (d. h. in der Datei **Web.config** in einem virtuellen Verzeichnis).
- ASP.NET (unter Windows 2000) erfordert das Recht **Als Teil des Betriebssystems handeln**. Diese Einschränkung gilt nicht für Microsoft Windows Server 2003.
  Weitere Informationen zu diesem weitreichenden Recht finden Sie im Microsoft Systems Journal, Ausgabe August 99, in der Spalte "Security Briefs" (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmsj99/html/security0899.asp, englischsprachig).

Das .NET-Framework, Version 1.1, stellt für dieses Szenario unter Windows 2000 eine Erweiterung bereit:

- Die Anmeldeinformationen werden verschlüsselt.
- Die Anmeldung wird vom IIS-Prozess durchgeführt, damit ASP.NET das Recht **Als Teil des Betriebssystem handeln** nicht erfordert.

Verwenden von Serviced Components

Sie können eine Serviced Component entwickeln, damit diese insbesondere Datenzugriffscode aufnimmt. Bei Serviced Components können Sie auf die Datenbank entweder über das Hosting Ihrer Komponente in einer Enterprise Services (COM+)-Serveranwendung zugreifen, die unter einer bestimmten Identität ausgeführt wird, oder Sie können Code einfügen, der die API **LogonUser** verwendet, um den Identitätswechsel durchzuführen.

Die Verwendung von prozessexternen Serviced Components erhöht die Sicherheit, da die Prozesshops den Angreifern die Arbeit erschweren, insbesondere wenn die Prozesse mit verschiedenen Identitäten ausgeführt werden. Der zweite Vorteil ist, dass Sie Code isolieren können, der vom Rest der Anwendung weiterreichende Rechte erfordert.

Aufrufen von "LogonUser" und Durchführen des Identitätswechsels für eine bestimmte Windows-Identität

Sie sollten **LogonUser** nicht direkt über ASP.NET aufrufen. Unter Windows 2000 fordert dieser Ansatz von Ihnen, dass Sie der ASP.NET-Prozessidentität das Recht **Als Teil des Betriebssystems handeln** gewähren.

Eine bevorzugte Möglichkeit ist es, **LogonUser** außerhalb des ASP.NET-Prozesses aufzurufen, indem Sie eine Serviced Component in einer Enterprise Services-Serveranwendung verwenden, wie bereits beschrieben.

Verwenden der Identität des ursprünglichen Aufrufers

Damit dieser Ansatz funktioniert, müssen Sie die Kerberos-Delegierung verwenden und für den Aufrufer einen Identitätswechsel für die Datenbank durchführen, entweder direkt über ASP.NET oder über eine Serviced Component.

Fügen Sie Folgendes über ASP.NET zu der Datei **Web.config** Ihrer Anwendung hinzu:

```
<identity impersonate="true" />
```

Rufen Sie **CoImpersonateClient** über eine Serviced Component auf.

Verwenden des anonymen Internetbenutzerkontos

Als Variante des vorherigen Ansatzes können Sie für Szenarien, in denen Ihre Anwendung die Formular- oder Passport-Authentifizierung verwendet (impliziert die anonyme IIS-Authentifizierung), den Identitätswechsel in der Datei **Web.config** der Anwendung aktivieren, um das anonyme Internetbenutzerkonto für den Datenbankzugriff zu verwenden.

```
<identity impersonate="true" />
```

Wenn IIS für die anonyme Authentifizierung konfiguriert ist, führt diese Konfiguration dazu, dass der Code der Webanwendung unter Verwendung des Identitätswechseltokens des anonymen Internetbenutzers ausgeführt wird. In einer Webhostumgebung hat dies den Vorteil, dass Sie den Datenbankzugriff mehrerer Webanwendungen einzeln überwachen und nachverfolgen können.

Weitere Informationen

- Weitere Informationen und Implementierungsdetails zum Verwenden der Identität des ursprünglichen Aufrufers finden Sie unter "Übermitteln des ursprünglichen Aufrufers an die Datenbank" in Kapitel 5, "Intranetsicherheit".
- Weitere Informationen zum Konfigurieren von IIS für die Verwendung des anonymen Benutzerkontos finden Sie in Kapitel 8, "ASP.NET-Sicherheit".

Wann kann die Windows-Authentifizierung nicht verwendet werden?

Bestimmte Anwendungsszenarien verhindern möglicherweise die Verwendung der Windows-Authentifizierung. Beispiel:

- Der Datenbankclient und Datenbankserver sind durch einen Firewall getrennt, die eine Windows-Authentifizierung verhindert.
- Die Anwendung muss eine Verbindung zu einer oder mehreren Datenbanken herstellen, wobei mehrere Identitäten verwendet werden.
- Sie stellen die Verbindung zu Datenbanken her, bei denen es sich nicht um SQL Server-Datenbanken handelt.
- Sie verfügen in ASP.NET nicht über eine sichere Möglichkeit, um Code als einen bestimmten Windows-Benutzer auszuführen. Entweder Sie können (oder wollen) den Sicherheitskontext des ursprünglichen Aufrufers nicht weiterleiten, oder Sie möchten eher ein dediziertes Dienstkonto verwenden, anstatt Endbenutzern Anmeldungen zu gewähren.

  Das Angeben eines Benutzernamens und eines Kennworts in der Datei **Machine.config** (über das Element <**processModel**>) oder **Web.config** (über das Element <**identity**>), um den ASP.NET-Workerprozess oder Ihre Anwendung auszuführen, ist weniger sicher, als explizite Schritte zum Schützen der SQL-Standardanmeldeinformationen einzuleiten.

In diesen Szenarien müssen Sie die SQL-Authentifizierung verwenden (oder den eigenen Authentifizierungsmechanismus der Datenbank) und die folgenden Schritte durchführen:

- Schützen der Datenbank-Benutzeranmeldeinformationen auf dem Anwendungsserver.
- Schützen der Datenbank-Benutzeranmeldeinformationen, während diese vom Server zur Datenbank übertragen werden.

Wenn Sie die SQL-Authentifizierung verwenden, gibt es zahlreiche Möglichkeiten, wie Sie die SQL-Authentifizierung sicherer gestalten können. Diese werden im nächsten Abschnitt hervorgehoben.

SQL-Authentifizierung

Wenn Ihre Anwendung die SQL-Authentifizierung verwenden muss, müssen Sie die folgenden wichtigen Punkte bedenken:

- Verwenden Sie ein Konto mit minimalen Rechten, um die Verbindung zu SQL herzustellen.
- Die Anmeldeinformationen werden über das Netzwerk gesendet, deshalb müssen sie gesichert werden.
- Die SQL-Verbindungszeichenfolge (die Anmeldeinformationen enthält) muss gesichert werden.

Typen von Verbindungszeichenfolgen

Wenn Sie die Verbindung zu einer SQL Server-Datenbank mithilfe von Anmeldeinformationen (Benutzername und Kennwort) herstellen, sieht die Zeichenfolge folgendermaßen aus:

```
SqlConnectionString = "Server=YourServer;
                      Database=YourDatabase;
                      uid=YourUserName;pwd=YourStrongPassword;"
```

Wenn Sie die Verbindung zu einer bestimmten Instanz von SQL Server (diese Funktion ist nur in SQL Server 2000 oder höher verfügbar) herstellen müssen, die auf demselben Computer installiert ist, dann sieht die Verbindungszeichenfolge folgendermaßen aus:

```
SqlConnectionString = "Server=YourServer\Instance;
                      Database=YourDatabase;uid=YourUserName;
                      pwd=YourStrongPassword;"
```

Wenn Sie die Verbindung zu SQL Server mithilfe Ihrer Netzwerkanmeldeinformationen herstellen möchten, verwenden Sie das **Integrated Security**-Attribut (oder das **Trusted Connection**-Attribut) und lassen den Benutzernamen und das Kennwort aus:

```
SqlConnectionString = "Server=YourServer;
                       Database=YourDatabase;
                       Integrated Security=SSPI;"
```

– oder –

```
SqlConnectionString = "Server=YourServer;
                      Database=YourDatabase;
                      Trusted_Connection=Yes;"
```

Wenn Sie die Verbindung zu einer Oracle-Datenbank mithilfe expliziter Anmeldeinformationen (Benutzername und Kennwort) herstellen, sieht die Zeichenfolge folgendermaßen aus:

```
SqlConnectionString = "Provider=MSDAORA;Data Source=YourDatabaseAlias;
                      User ID=YourUserName;Password=YourPassword;"
```

Weitere Informationen

Weitere Informationen zum Verwenden von UDL-Dateien (Universal Data Link) für Verbindungen finden Sie in der Microsoft Knowledge Base im Artikel Q308426, "HOW TO: Use Data Link Files with the OleDbConnection Object in Visual C# .NET" (US).

Auswählen eines SQL-Kontos für Verbindungen

Verwenden Sie die integrierten Konten **sa** oder **db_owner** nicht für den Datenzugriff. Verwenden Sie stattdessen Konten mit minimalen Rechten, jedoch mit starkem Kennwort.

Vermeiden Sie die folgende Verbindungszeichenfolge:

```
SqlConnectionString = "Server=YourServer\Instance;
                       Database=YourDatabase; uid=sa; pwd=;"
```

Verwenden Sie Konten mit minimalen Rechten, jedoch mit starkem Kennwort. Beispiel:

```
SqlConnectionString= "Server=YourServer\Instance;
                      Database=YourDatabase;
                      uid=YourStrongAccount;
                      pwd=YourStrongPassword;"
```

Beachten Sie, dass dies nicht das Problem mit dem unverschlüsselten Speichern der Anmeldeinformationen in den **Web.config**-Dateien behandelt. Sie haben bis jetzt nur das Schadensausmaß eingeschränkt, das im Fall eines erfolgreichen Angriffs eintreten könnte, indem Sie ein Konto mit minimalen Rechten verwenden. Wenn Sie die Sicherheit weiter erhöhen möchten, sollten Sie die Anmeldeinformationen verschlüsseln.

Hinweis: Wenn Sie bei der Installation von SQL Server eine Sortierreihenfolge gewählt haben, bei der zwischen Groß-/Kleinschreibung unterschieden wird, wird bei Ihrer Anmelde-ID ebenfalls zwischen Groß-/Kleinschreibung unterschieden.

Übergeben von Anmeldeinformationen über das Netzwerk

Wenn Sie die Verbindung zu SQL Server über die SQL-Authentifizierung herstellen, werden der Benutzername und das Kennwort unverschlüsselt über das Netzwerk übertragen. Dies kann sich als erhebliches Sicherheitsproblem erweisen. Weitere Informationen zum Sichern des Kanals zwischen einer Anwendung oder einem Webserver und einem Datenbankserver finden Sie weiter unten in diesem Kapitel unter "Sichern der Kommunikation".

Sichern von SQL-Verbindungszeichenfolgen

Benutzernamen und Kennwörter sollten in Konfigurationsdateien nicht unverschlüsselt gespeichert werden. Weitere Informationen zum sicheren Speichern von Verbindungszeichenfolgen finden Sie weiter unten in diesem Kapitel unter "Speichern von Datenbank-Verbindungszeichenfolgen".

Authentifizieren anhand von Nicht-SQL Server-Datenbanken

Ein typisches Problem, das beim Herstellen der Verbindung zu Datenbanken auftreten kann, die nicht auf SQL basieren, ist mit Szenarien vergleichbar, in denen Sie die SQL-Authentifizierung verwenden müssen. Sie müssen möglicherweise Anmeldeinformationen explizit bereitstellen, wenn die Zielressourcen die Windows-Authentifizierung nicht unterstützen. Um diese Art von Szenario zu sichern, müssen Sie die Verbindungszeichenfolge sicher verwahren und auch die Kommunikation über das Netzwerk sichern (um das Abfangen von Anmeldeinformationen zu verhindern).

Weitere Informationen

- Weitere Informationen zum Speichern von Datenbank-Verbindungszeichenfolgen finden Sie weiter unten in diesem Kapitel unter Sicheres Speichern von Datenbank-Verbindungszeichenfolgen".
- Weitere Informationen zum Sichern des Kanals zum Datenbankserver finden Sie weiter unten in diesem Kapitel unter "Sichern der Kommunikation".

Autorisierung

SQL Server bietet für die Autorisierung eine Reihe von rollenbasierten Ansätzen. Diese drehen sich um die folgenden drei Rollenarten, die von SQL Server unterstützt werden:

- **Benutzerdefinierte Datenbankrollen** – Diese werden dazu verwendet, um Benutzer mit den gleichen Berechtigungen in der Datenbank zu gruppieren. Mit den Rollen fügen Sie Benutzer- oder Gruppenkonten von Windows zu Benutzerdatenbankrollen hinzu und richten Berechtigungen für einzelne Datenbankobjekte ein (gespeicherte Prozeduren, Tabellen, Sichten usw.).
- **Anwendungsrollen** – Diese sind in der Hinsicht mit Benutzerdatenbankrollen vergleichbar, dass sie beim Einrichten von Objektberechtigungen verwendet werden. Im Gegensatz zu diesen enthalten sie jedoch keine Benutzer oder Gruppen. Stattdessen müssen sie durch eine Anwendung mithilfe einer integrierten gespeicherten Prozedur aktiviert werden. Nach der Aktivierung bestimmen die der Rolle gewährten Berechtigungen die Möglichkeiten der Anwendung für den Datenzugriff.

  Anwendungsrollen ermöglichen es Datenbankadministratoren, ausgewählten Anwendungen den Zugriff auf angegebene Datenbankobjekte zu gewähren. Dies steht im Gegensatz zum Gewähren von Berechtigungen für Benutzer.
- **Feste Datenbankrollen** – SQL Server bietet auch feste Serverrollen, z. B. **db_datareader** und **db_datawriter**. Diese integrierten Rollen sind in allen Datenbanken enthalten und können verwendet werden, um einem Benutzer schnell eine Reihe von auf den Lesevorgang bezogene (und andere häufig verwendete) Berechtigungen in der Datenbank zu gewähren.

Weitere Informationen zu diesen verschiedenen Rollenarten (sowie zu festen Serverrollen, die mit festen Datenbankrollen vergleichbar sind, jedoch auf der Server- und nicht der Datenbankebene angewendet werden) finden Sie in der Onlinedokumentation von SQL Server (http://www.microsoft.com/sql/techinfo/productdoc/2000/books.asp, englischsprachig).

Verwenden mehrerer Datenbankrollen

Wenn Ihre Anwendung über verschiedene Benutzerkategorien verfügt und die Benutzer in den einzelnen Kategorien dieselben Berechtigungen in der Datenbank erfordern, muss die Anwendung mehrere Rollen verwenden.

Für jede Rolle muss ein anderer Berechtigungssatz in der Datenbank erstellt werden. Mitglieder der Rolle "Internetbenutzer" benötigen z. B. möglicherweise die Berechtigung für den Lesezugriff auf die meisten Tabellen in der Datenbank, während Mitglieder der Rolle "Administrator" oder "Operator" möglicherweise Berechtigungen zum Lesen und zum Schreiben benötigen.

Optionen

Um diesen Szenarien zu entsprechen, bieten sich Ihnen zwei hauptsächliche Optionen für die rollenbasierte Autorisierung in SQL Server:

- **Benutzerdefinierte SQL Server-Datenbankrollen** – Diese werden verwendet, um Berechtigungen für den Zugriff auf Datenbankobjekte zu Benutzergruppen zuzuordnen, die innerhalb der Datenbank über dieselben Sicherheitsberechtigungen verfügen.

  Wenn Sie benutzerdefinierte Datenbankrollen verwenden, führen Sie Überprüfungen am Gate durch, ordnen Benutzer zu Rollen zu (z. B. in einer ASP.NET-Webanwendung oder in einer Serviced Component der mittleren Ebene in einer Enterprise Services-Serveranwendung) und verwenden mehrere Identitäten, um die Verbindung zur Datenbank herzustellen, wobei jede einer benutzerdefinierten Datenbankrolle zugeordnet ist.

- **SQL-Anwendungsrollen** – Diese sind in der Hinsicht mit benutzerdefinierten Datenbankrollen vergleichbar, dass sie verwendet werden, wenn Sie Berechtigungen zu Datenbankobjekten zuordnen. Im Gegensatz zu den benutzerdefinierten Datenbankrollen enthalten sie jedoch keine Mitglieder und werden von einzelnen Anwendungen aktiviert, indem diese eine integrierte gespeicherte Prozedur verwenden.

  Wenn Sie Anwendungsrollen verwenden, führen Sie Überprüfungen am Gate durch, ordnen Benutzer zu Rollen zu, stellen die Verbindung zur Datenbank mithilfe einer einzelnen, vertrauenswürdigen Dienstidentität her und aktivieren die entsprechende SQL-Anwendungsrolle.

Benutzerdefinierte Datenbankrollen –

Wenn Sie sich für die Verwendung von benutzerdefinierten Datenbankrollen entschieden haben, müssen Sie Folgendes durchführen:

- Erstellen mehrerer Dienstkonten für den Datenbankzugriff.
- Zuordnen der Konten zu einer benutzerdefinierten Datenbankrolle.
- Einrichten der erforderlichen Datenbankberechtigungen für jede Rolle in der Datenbank.
- Autorisieren der Benutzer in Ihrer Anwendung (ASP.NET-Webanwendung, Webdienst oder Komponente der mittleren Ebene) und anschließendes Verwenden der Anwendungslogik auf der Datenzugriffsebene, um zu ermitteln, mit welchem Konto die Verbindung zur Datenbank hergestellt wird. Dies basiert auf der Rollenmitgliedschaft des Aufrufers.

  Sie können einzelne Methoden deklarativ konfigurieren, um nur die Benutzer zuzulassen, die einer Reihe von Rollen angehören. Dann fügen Sie die imperativen Rollenüberprüfungen im Code der Methode hinzu, um die genaue Rollenmitgliedschaft zu ermitteln, die die zu verwendende Verbindung bestimmt.

In Abbildung 12.4 wird dieser Ansatz veranschaulicht.

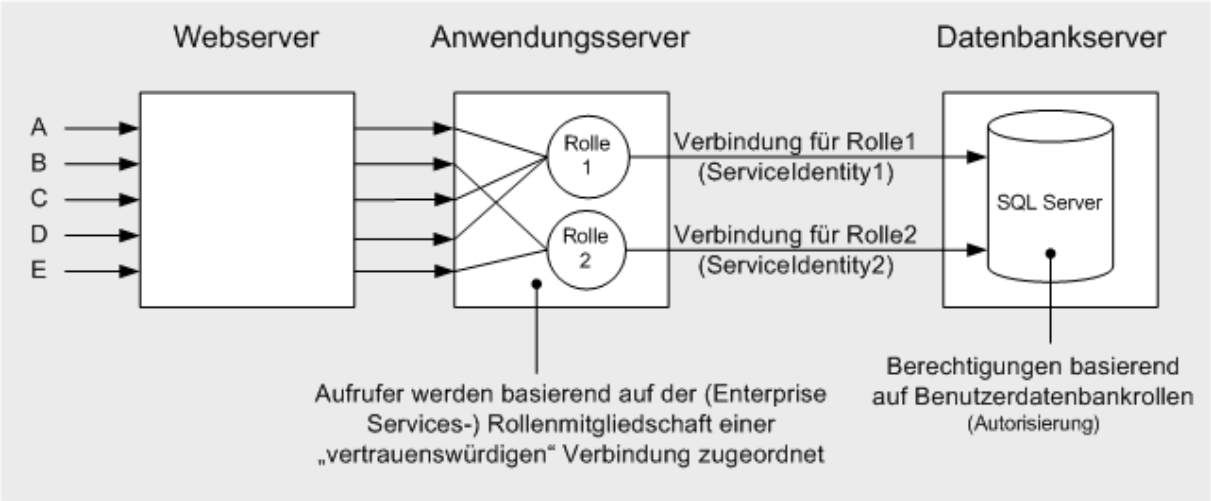

Abbildung 12.4
Herstellen der Verbindung zu SQL Server unter Verwendung mehrerer SQL-Benutzerdatenbankrollen

Damit Sie die bevorzugte Windows-Authentifizierung für dieses Szenario verwenden können, entwickeln Sie Code (mithilfe der API **LogonUser**) in einer "Out of Process"-Serviced Component, um einen Identitätswechsel für eine der Windows-Identitäten aus einer Reihe von Identitäten durchzuführen.

Bei der SQL-Authentifizierung verwenden Sie eine andere Verbindungszeichenfolge (die verschiedene Benutzernamen und Kennwörter enthält), die von der rollenbasierten Logik in Ihrer Anwendung abhängt.

Weitere Informationen

Weitere Informationen zum sicheren Speichern von Datenbank-Verbindungszeichenfolgen finden Sie weiter unten in diesem Kapitel unter "Sicheres Speichern von Datenbank-Verbindungszeichenfolgen".

Anwendungsrollen

Bei SQL-Anwendungsrollen müssen Sie Folgendes durchführen:

- Erstellen Sie ein einzelnes Dienstkonto für den Datenbankzugriff (dies kann das zum Ausführen des ASP.NET-Workerprozesses verwendete Prozesskonto oder eine Enterprise Services-Anwendung sein).
- Erstellen Sie innerhalb der Datenbank einen Satz von SQL-Anwendungsrollen.
- Richten Sie die erforderlichen Datenbankberechtigungen für jede Rolle in der Datenbank ein.
- Autorisieren Sie die Benutzer in Ihrer Anwendung (ASP.NET-Webanwendung, Webdienst oder Komponente der mittleren Ebene), und verwenden Sie anschließend die Anwendungslogik auf der Datenzugriffsebene, um zu ermitteln, welche Anwendungsrolle innerhalb der Datenbank aktiviert wird. Dies basiert auf der Rollenmitgliedschaft des Aufrufers.

In Abbildung 12.5 wird dieser Ansatz veranschaulicht.

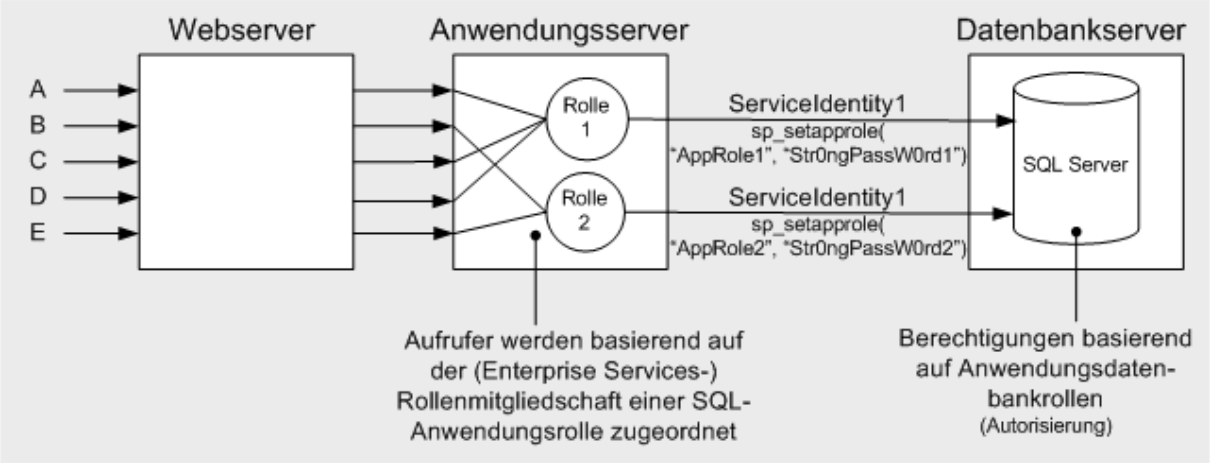

Abbildung 12.5
Verwenden mehrerer SQL-Anwendungsrollen

In Abbildung 12.5 wird die Identität **ServiceIdentity1**, die für den Zugriff auf die Datenbank verwendet wird, aus dem ASP.NET-Workerprozess oder aus einer Prozessidentität der Enterprise Services-Serveranwendung abgerufen.

Bei diesem Ansatz wird dieselbe Dienstidentität (und daher dieselbe Verbindung) zum Herstellen der Verbindung zu SQL Server verwendet. Die SQL-Anwendungsrollen werden basierend auf der Rollenmitgliedschaft des Aufrufers mit der integrierten gespeicherten Prozedur **sp_setapprole** aktiviert. Für diese gespeicherte Prozedur müssen der Rollenname und ein Kennwort bereitgestellt werden.

Wenn Sie diesen Ansatz verwenden, müssen Sie den in den Anmeldeinformationen enthaltenen Rollennamen und das Kennwort sicher speichern. Weitere Informationen und sichere Speicherverfahren finden Sie weiter unten in diesem Kapitel unter "Sicheres Speichern von Datenbank-Verbindungszeichenfolgen".

Einschränkungen von SQL-Anwendungsrollen

Nachfolgend werden die wichtigsten Punkte aufgeführt, derer Sie sich bewusst sein sollten, bevor Sie die Verwendung von SQL-Anwendungsrollen wählen:

- Sie müssen die Anmeldeinformationen für die SQL-Anwendungsrollen verwalten. Sie müssen die gespeicherte Prozedur **sp_setapprole** aufrufen und für jede Verbindung einen Rollennamen und ein Kennwort übergeben. Wenn Sie eine SQL-Anwendungsrolle über verwalteten Code aktivieren, erweist sich das in der Assembly unverschlüsselt eingebettete Kennwort als nicht sicher.
- Anmeldeinformationen für die SQL-Anwendungsrolle werden unverschlüsselt an die Datenbank übergeben. Sie sollten diese im Netzwerk sichern, indem Sie zwischen dem Anwendungsserver und dem Datenbankserver IPSec oder SSL verwenden.
- Nachdem eine SQL-Anwendungsrolle für eine Verbindung aktiviert ist, kann diese nicht mehr deaktiviert werden. Sie verbleibt aktiv, bis die Verbindung geschlossen wird. Sie können für dieselbe Verbindung auch nicht zwischen zwei oder mehreren Rollen wechseln.
- Verwenden Sie die SQL-Anwendungsrollen nur, wenn Ihre Anwendung zum Herstellen der Verbindung zur Datenbank nur eine einzelne feste Identität verwendet. Mit anderen Worten, verwenden Sie diese nur, wenn die Anwendung das Modell mit vertrauenswürdigen Subsystemen verwendet.

  Wenn sich der Sicherheitskontext der Verbindung ändert (wie es der Fall wäre, wenn die Verbindung zur Datenbank mit dem Kontext des ursprünglichen Aufrufers hergestellt wurde), arbeiten die SQL-Anwendungsrollen nicht mit dem Verbindungspooling zusammen.

  Weitere Informationen finden Sie in der Microsoft Knowledge Base im Artikel Q229564, "PRB: SQL Application Role Errors with OLE DB Resource Pooling" (US).

Sichere Kommunikation

In den meisten Anwendungsszenarien müssen Sie die Kommunikationsverbindung zwischen dem Anwendungsserver und der Datenbank sichern. Sie müssen Folgendes gewährleisten:

- **Nachrichtenvertraulichkeit** – Die Daten müssen verschlüsselt werden, um sicherzustellen, dass sie nicht unberechtigt eingesehen werden können.
- **Nachrichtenintegrität** – Die Daten müssen signiert werden, um sicherzustellen, dass sie nicht verändert werden können.

In einigen Szenarien müssen alle zwischen dem Anwendungsserver und dem Datenbankserver übertragenen Daten gesichert werden, während in anderen Szenarien nur ausgewählte Datenelemente gesichert werden müssen, die über bestimmte Verbindungen gesendet werden. Beispiel:

- In einer Intranetanwendung der Personalabteilung sind einige der Mitarbeiterdaten vertraulich, die zwischen dem Client und dem Datenbankserver übergeben werden.
- In Internetszenarien (z. B. sichere Bankanwendungen) sind alle zwischen dem Anwendungsserver und dem Datenbankserver übergebenen Daten vertraulich und müssen gesichert werden.
- Wenn Sie die SQL-Authentifizierung verwenden, sollten Sie auch die Kommunikationsverbindung sichern, um sicherzustellen, dass Benutzernamen und Kennwörter nicht unberechtigt mithilfe von Software zur Netzwerküberwachung abgerufen werden können.

Optionen

Es stehen zwei Möglichkeiten zum Sichern der Netzwerkverbindung zwischen einem Anwendungsserver und dem Datenbankserver zur Verfügung:

- IPSec
- SSL (Mithilfe eines Serverzertifikats auf dem SQL Server-Computer)

Hinweis: Sie müssen SQL Server 2000 ausführen, um die Verwendung von SSL unterstützen zu können. Frühere Versionen unterstützen die Verwendung von SSL nicht. Auf dem Client müssen die Clientbibliotheken von SQL Server 2000 installiert sein.

Auswählen eines Ansatzes

Ob Sie IPSec oder SSL verwenden, hängt von einer Reihe wichtiger Umgebungsfaktoren ab, z. B. von Überlegungen zum Firewall, von der Version des Betriebssystems und von Datenbanken usw.

Hinweis: IPSec soll nicht die Sicherheit auf Anwendungsebene ersetzen, sondern wird heute als tiefreichender Verteidigungsmechanismus oder zum Sichern unsicherer Anwendungen, ohne diese zu ändern, sowie zum Sichern von anderen Protokollen als TLS (z. B. SSL) vor Angriffen aus dem Netzwerk verwendet.

Weitere Informationen

- Weitere Informationen zum Konfigurieren von IPSec finden Sie unter "Vorgehensweise: Verwenden von IPSec zum Sichern der Kommunikation zwischen zwei Servern" im Abschnitt "Referenz" dieses Handbuchs.
- Weitere Informationen zum Konfigurieren von SSL finden Sie unter "Vorgehensweise: Verwenden von SSL zum Sichern der Kommunikation mit SQL Server 2000" im Abschnitt "Referenz" dieses Handbuchs.
- Weitere allgemeine Informationen zu SSL und IPSec finden Sie in Kapitel 4, "Sichere Kommunikation".

Herstellen der Verbindung mit minimalen Rechten

Das Herstellen der Verbindung zur Datenbank mit minimalem Recht bedeutet, dass die von Ihnen eingerichtete Verbindung nur über die minimalen Rechte verfügt, die Sie in der Datenbank brauchen. Einfacher gesagt, Sie stellen die Verbindung zur Datenbank nicht unter Verwendung des Kontos **sa** oder des Datenbankeigentümerkontos her. Idealerweise kann das entsprechende, für die Verbindung (die zu einer Identität zusammengefasst werden kann, die eine bestimmte Rolle darstellt) verwendete Konto keine Datensätze in der Datenbank hinzufügen oder aktualisieren, wenn der aktuelle Benutzer nicht dazu autorisiert ist.

Wen Sie die Verbindung zu SQL Server herstellen, muss der von Ihnen gewählte Ansatz die erforderliche Feinstufigkeit unterstützen, die Ihre Datenbankautorisierung erfordert. Sie müssen bedenken, wem die Datenbank vertrauen kann. Sie kann Folgendem vertrauen:

- Der Anwendung
- Den anwendungsdefinierten Rollen
- Dem ursprünglichen Aufrufer

Die Datenbank vertraut der Anwendung

Denken Sie z. B. an eine Finanzanwendung, die Sie für die Verwendung mit Ihrer Datenbank autorisieren. Die Finanzanwendung ist verantwortlich für das Verwalten der benutzerdefinierten Authentifizierung und für den Autorisierungszugriff. In diesem Fall verwalten Sie Ihre Verbindungen über ein einzelnes vertrauenswürdiges Konto (das entweder einem SQL-Benutzernamen oder einem Windows-Konto entspricht, das einem SQL-Benutzernamen zugeordnet ist). Wenn Sie die Windows-Authentifizierung verwenden, bedeutet dies normalerweise, dass die Prozessidentität der aufrufenden Anwendung (z. B. der ASP.NET-Workerprozess oder die Identität einer Enterprise Services-Serveranwendung) die Möglichkeit besitzt, auf die Datenbank zuzugreifen.

Aus der Sicht der Autorisierung ist dieser Ansatz sehr grobstufig, da die Verbindung als Identität ausgeführt wird, die über den Zugriff auf alle Datenbankobjekte und Ressourcen verfügt, die von der Anwendung benötigt werden. Die Vorteile dieses Ansatzes sind, dass Sie das Verbindungspooling verwenden und die Verwaltung vereinfachen können, da Sie nur ein einzelnes Konto autorisieren. Der Nachteil ist, dass alle Benutzer mit denselben Verbindungsrechten ausgeführt werden.

Die Datenbank vertraut verschiedenen Rollen

Sie können Pools aus separaten, vertrauenswürdigen Verbindungen zur Datenbank verwenden, die den von der Anwendung definierten Rollen entsprechen, z. B. eine Verbindung für Kassierer, eine weitere für Manager usw.

Diesen Verbindungen steht es frei, die Windows-Authentifizierung zu verwenden. Der Vorteil der Windows-Authentifizierung ist, dass sie die Verwaltung der Anmeldeinformationen übernimmt und diese nicht über das Netzwerk sendet. Während die Windows-Authentifizierung auf der Prozess- oder Anwendungsebene möglich ist (wenn Sie eine einzelne Verbindung zur Datenbank verwenden), stellen sich ihr durch die Tatsache, dass Sie mehrere Identitäten verwenden müssen (eine pro Rolle), weitere Herausforderungen.

Viele Anwendungen verwenden die **LogonUser**-API, um ein Windows-Zugriffstoken einzurichten. Das Problem dieses Ansatzes besteht aus zweierlei:

- Sie haben jetzt ein Problem mit der Verwaltung der Anmeldeinformationen (die Anwendung muss den Benutzernamen und das Kennwort für das Konto sicher speichern).
- Die **LogonUser**-API erfordert, dass das aufrufende Prozesskonto über das Recht **Als Teil des Betriebssystems handeln** verfügt. Das bedeutet, dass Sie dem ASP.NET-Prozesskonto dieses Recht gewähren müssen, was jedoch nicht empfohlen wird. Eine Alternative ist die Verwendung der SQL-Authentifizierung, bei der Sie wiederum die Anmeldeinformationen auf dem Server und bei der Übertragung im Netzwerk schützen müssen.

Hinweis: Diese **LogonUser**-Einschränkung wird in Windows Server 2003 aufgehoben.

Die Datenbank vertraut dem ursprünglichen Aufrufer

In diesem Fall müssen Sie den ursprünglichen Aufrufer über mehrere Ebenen an die Datenbank übermitteln. Das bedeutet, dass Ihre Clients Netzwerkanmeldeinformationen besitzen müssen, damit sie von einem Computer zum nächsten gelangen können. Dazu ist die Kerberos-Delegierung erforderlich.

Obwohl diese Lösung eine feinstufige Autorisierung in der Datenbank bietet, da Ihnen die Identität des ursprünglichen Aufrufers bekannt ist und Sie für Datenbankobjekte Berechtigungen auf Benutzerbasis einrichten können, wirkt sie sich auf die Leistung und Skalierbarkeit der Anwendung aus. Das Verbindungspooling (obwohl weiterhin aktiv) wird uneffektiv.

Erstellen eines Datenbankkontos mit minimalen Rechten

Die folgenden Schritte werden als einfaches Beispiel bereitgestellt, um Ihnen zu zeigen, wie ein Datenbankkonto mit minimalen Rechten erstellt wird. Während die meisten Datenbankadministratoren bereits mit diesen Schritten vertraut sind, ist dies für viele Entwickler nicht der Fall, die dann möglicherweise auf die Verwendung von **sa** oder eines Datenbankeigentümerkontos zurückgreifen, damit ihre Anwendungen funktionieren.

Dies kann bei einem Wechsel von einer Entwicklungsumgebung zu einer Testumgebung und dem anschließenden Wechsel zu einer Produktionsumgebung zu Problemen führen, da die Anwendung aus einer frei zugänglichen Umgebung in eine umfassender kontrollierte Umgebung gelangt, wodurch die Anwendung möglicherweise nicht mehr ordnungsgemäß funktioniert.

Sie beginnen mit der Erstellung eines SQL-Benutzernamens entweder für ein SQL-Konto oder für ein Windows-Konto (Benutzer oder Gruppe). Anschließend fügen Sie diesen Benutzernamen zu einer Datenbankbenutzerrolle hinzu und ordnen der Rolle dann Berechtigungen zu.

▶ **So richten Sie ein Datenzugriffskonto für SQL ein:**

1. Erstellen Sie ein neues Benutzerkonto, und fügen Sie dieses Konto einer Windows-Gruppe hinzu. Wenn Sie mehrere Benutzer verwalten, verwenden Sie eine Gruppe. Wenn Sie ein einzelnes Anwendungskonto verwenden (z. B. ein dupliziertes ASP.NET-Prozesskonto), fügen Sie das Konto möglicherweise nicht zu einer Windows-Gruppe hinzu.
2. Erstellen Sie einen SQL Server-Benutzernamen für den Benutzer bzw. für die Gruppe.
   a. Starten Sie den Enterprise Manager, suchen Sie dort Ihren Datenbankserver, und erweitern Sie dann den Ordner **Sicherheit**.
   b. Klicken Sie mit der rechten Maustaste auf **Benutzernamen**, und klicken Sie dann auf **Neuer Benutzername**.
   c. Geben Sie den Windows-Gruppennamen in das Feld **Name** ein, und klicken Sie dann auf **OK**, um das Dialogfeld **SQL Server-Anmeldungseigenschaften** zu schließen.
3. Erstellen Sie in der gewünschten Datenbank einen neuen Datenbankbenutzer, der dem SQL Server-Benutzernamen zugeordnet wird.
   a. Verwenden Sie den **Enterprise Manager**, erweitern Sie den Ordner **Datenbanken**, und erweitern Sie dann die gewünschte Datenbank, für die der Benutzername den Zugriff erfordert.
   b. Klicken Sie mit der rechten Maustaste auf **Benutzer**, und klicken Sie dann auf **Neuer Datenbankbenutzer**.
   c. Wählen Sie den zuvor erstellten Anmeldenamen aus.
   d. Geben Sie einen Benutzernamen an.
   e. Konfigurieren Sie die Berechtigungen, wie nachfolgend beschrieben.
4. Gewähren Sie dem Datenbankbenutzer die Berechtigung **Select** für die Tabellen, auf die er zugreifen können muss, sowie die **Exec**-Berechtigungen (Execute) für relevante gespeicherte Prozeduren.

Hinweis: Wenn die gespeicherte Prozedur und die Tabelle Eigentum derselben Person sind und der Zugriff auf die Tabelle nur über die gespeicherte Prozedur erfolgt (und kein direkter Zugriff auf die Tabelle erforderlich ist), reicht es aus, wenn Sie nur der gespeicherten Prozedur die Ausführungsberechtigung gewähren. Der Grund hierfür ist das Konzept der Besitzverkettung. Weitere Informationen finden Sie in der SQL Server-Onlinedokumentation.

---

5. Wenn das Benutzerkonto auf alle Sichten und Tabellen in der Datenbank zugreifen können soll, fügen Sie es zur Rolle **db_datareader** hinzu.

Sicheres Speichern von Datenbank-Verbindungszeichenfolgen

Es gibt eine Reihe von möglichen Positionen und Ansätzen zum Speichern von Datenbank-Verbindungszeichenfolgen, wobei diese hinsichtlich der Sicherheit und der Flexibilität der Konfiguration variieren.

Optionen

In der folgenden Liste werden die hauptsächlichen Optionen zum Speichern von Verbindungszeichenfolgen aufgeführt:

- Verschlüsselung mit DPAPI
- Unverschlüsselt in der Datei **Web.config** oder **Machine.config**
- UDL-Dateien
- Benutzerdefinierte Textdateien
- Registrierung
- COM+-Katalog

Verwenden von DPAPI

Mit Windows 2000 und den neueren Betriebssystemen wurde die Win32® Data Protection API (DPAPI) für das Ver- und Entschlüsseln von Daten eingeführt. DPAPI ist Teil der Cryptography API (Crypto API) und in **Crypt32.dll** implementiert. Die Schnittstelle setzt sich aus zwei Methoden, **CryptProtectData** und **CryptUnprotectData**, zusammen.

Die DPAPI ist insofern besonders nützlich, als dass sich hiermit das Schlüsselverwaltungsproblem erübrigt, das ansonsten mit Anwendungen einhergeht, die Kryptografie einsetzen. Mit der Verschlüsselung können Daten zwar gesichert werden, es müssen jedoch weitere Schritte unternommen werden, um auch die Sicherheit des Schlüssels zu gewährleisten. Die DPAPI verwendet das Kennwort des Benutzerkontos, das die DPAPI-Funktionen aufruft, um den Verschlüsselungsschlüssel abzuleiten. Damit verwaltet das Betriebssystem (und nicht die Anwendung) den Schlüssel.

Warum nicht LSA?

Viele Anwendungen verwenden die LSA (Local Security Authority) zum Speichern von vertraulichen Daten. DPAPI besitzt gegenüber dem LSA-Ansatz die folgenden Vorteile:

- Damit sie die LSA verwenden kann, muss eine Prozedur Administratorrechte besitzen. Dies lässt hinsichtlich der Sicherheit Bedenken aufkommen, da es das mögliche Schadensausmaß drastisch erhöht, das durch einen Angreifer angerichtet werden kann, der den Prozess erfolgreich angreift.
- Die LSA bietet zum sicheren Speichern nur eine begrenzte Anzahl von Slots, von denen viele bereits vom System verwendet werden.

Computerspeicher im Vergleich zum Benutzerspeicher

DPAPI kann entweder mit dem Computer- oder Benutzerspeicher (der ein geladenes Benutzerprofil erfordert) arbeiten. Standardmäßig verwendet die DPAPI den Benutzerspeicher, Sie können jedoch auch festlegen, dass der Computerspeicher verwendet werden soll, indem Sie das Flag CRYPTOPROTECT_LOCAL_MACHINE an die DPAPI-Funktionen übergeben.

Mit der Verwendung des Benutzerprofils wird eine zusätzliche Sicherheitsschicht eingezogen, denn hiermit wird der Zugriff auf die geheimen Daten weiter eingeschränkt. Nur der Benutzer, der die Daten verschlüsselt, kann sie auch wieder entschlüsseln. Allerdings erfordert die Verwendung des Benutzerprofils zusätzlichen Entwicklungsaufwand, wenn die DPAPI von einer ASP.NET-Anwendung verwendet werden soll, denn Sie müssen explizite Schritte zum Laden und Entladen des Benutzerprofils unternehmen (da ASP.NET Benutzerprofile nicht automatisch lädt).

Der Computerspeicheransatz ist einfacher zu entwickeln, da dieser keine Benutzerprofilverwaltung voraussetzt. Wird hierbei jedoch kein zusätzlicher Entropieparameter verwendet, ist dieser Ansatz weniger sicher, da jeder Benutzer des Computers Daten entschlüsseln kann. (Unter Entropie wird in diesem Zusammenhang ein Zufallswert verstanden, der das Dechiffrieren der Geheimdaten schwieriger macht.) Das Problem bei der Verwendung eines zusätzlichen Entropieparameters besteht darin, dass dieser Wert von der Anwendung ebenfalls sicher gespeichert werden muss, woraus sich wiederum ein Schlüsselverwaltungsproblem ergibt.

Hinweis: Wenn Sie DPAPI mit dem Computerspeicher verwenden, ist die verschlüsselte Zeichenfolge für einen angegebenen Computer bestimmt, und daher müssen Sie die verschlüsselten Daten auf jedem Computer generieren. Sie können die verschlüsselten Daten nicht auf andere Computer in einer Farm oder in einem Cluster kopieren.

Wenn Sie DPAPI mit dem Benutzerspeicher verwenden, können Sie die Daten auf einem beliebigen Computer mit einem servergespeicherten Benutzerprofil entschlüsseln.

DPAPI-Implementierungslösungen

In diesem Abschnitt werden zwei Implementierungslösungen präsentiert, die Ihnen zeigen, wie DPAPI von ASP.NET-Webanwendungen verwendet wird, um eine Verbindungszeichenfolge zu sichern (oder beliebige vertrauliche Daten). In diesem Abschnitt werden die folgenden Implementierungslösungen beschrieben:

- **Verwenden von DPAPI über die Enterprise Services -** Diese Lösung ermöglicht es Ihnen, DPAPI mit dem Benutzerspeicher zu verwenden.
- **Direktes Verwenden von DPAPI über ASP.NET -** Bei dieser Lösung können Sie DPAPI mit dem Computerspeicher verwenden, wodurch sich diese Lösung einfacher entwickeln lässt, da DPAPI direkt über eine ASP.NET-Webanwendung aufgerufen werden kann.

Verwenden von DPAPI über die Enterprise Services

Eine ASP.NET-Webanwendung kann DPAPI nicht aufrufen und dann den Benutzerspeicher verwenden, da hierzu ein Benutzerprofil geladen sein muss. Das normalerweise zum Ausführen von Webanwendungen verwendete Konto ASPNET ist ein nicht interaktives Konto und verfügt als solches über keine Benutzerprofile. Außerdem wird der Thread der Webanwendung als momentan authentifizierter Benutzer ausgeführt, der sich von einer Anforderung zur nächsten ändern kann, wenn die ASP.NET-Anwendung einen Identitätswechsel durchführt.

Hieraus ergibt sich für eine ASP.NET-Webanwendung, die die DPAPI verwenden möchte, die folgende Problemstellung:

- Aufrufe der DPAPI von einer ASP.NET-Anwendung, die unter dem ASPNET-Standardkonto ausgeführt wird, schlagen fehl. Der Grund hierfür ist, dass das ASPNET-Konto über keine Benutzerprofile verfügt, da es nicht für interaktive Anmeldungen verwendet wird.
- Wenn eine ASP.NET-Webanwendung für einen Identitätswechsel ihrer Aufrufer konfiguriert ist, verfügt der Thread der ASP.NET-Anwendung über ein zugeordnetes Threadidentitätswechseltoken. Die Anmeldesitzung, die mit diesem Identitätswechseltoken verbunden ist, ist eine Netzwerkanmeldesitzung (die auf dem Server verwendet wird, um den Aufrufer zu repräsentieren). Anmeldesitzungen im Netzwerk führen nicht dazu, dass Benutzerprofile geladen werden.

Sie können eine Serviced Component erstellen (in einer prozessexternen Enterprise Services (COM+)-Serveranwendung), um die DPAPI aufzurufen, damit dieses Problem beseitigt wird. Sie können sicherstellen, dass das Konto, mit dem die Komponente ausgeführt wird, ein Benutzerprofil besitzt und Sie einen Win32-Dienst verwenden können, um das Profil automatisch zu laden.

Hinweis: Es ist möglich, die Verwendung eines Win32-Dienstes zu vermeiden, indem Aufrufe von Win32-Profilverwaltungsfunktionen (**LoadUserProfile** und **UnloadUserProfile**) innerhalb der Serviced Component positioniert werden.

Dieser Ansatz hat zwei Nachteile. Erstens wirken sich Aufrufe dieser APIs auf Anforderungsbasis erheblich auf die Leistung aus. Zweitens erfordern diese APIs, dass der aufrufende Code auf dem lokalen Computer Administratorrechte besitzt, wodurch das Prinzip der minimalen Rechte für das Enterprise Services-Prozesskonto nicht befolgt wird.

Abbildung 12.6 zeigt die Enterprise Services DPAPI-Lösung.

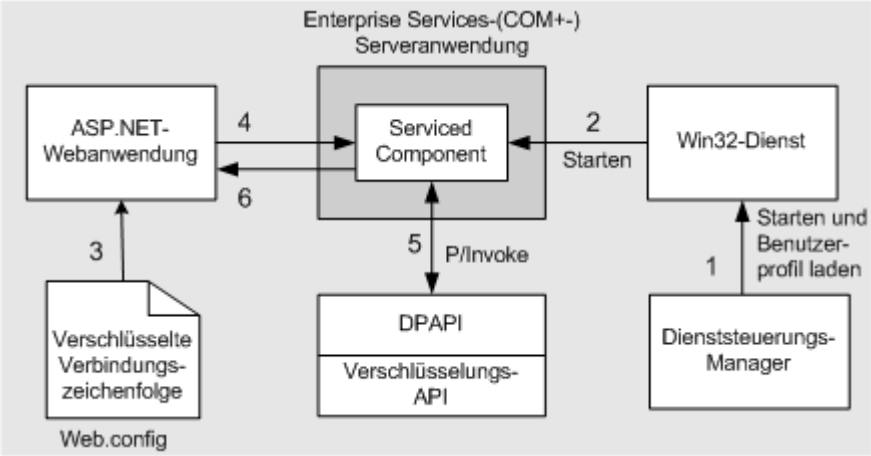

Abbildung 12.6
Die ASP.NET-Webanwendung verwendet eine COM+-Serveranwendung für die Interaktion mit der DPAPI

Nachstehend die Abfolge der Ereignisse zur Laufzeit wie in Abbildung 12.6:

1. Der Dienststeuerungs-Manager von Windows startet den Win32-Dienst und lädt automatisch das Benutzerprofil, das mit dem Konto verbunden ist, unter dem der Dienst ausgeführt wird. Unter dem gleichen Windows-Konto wird auch die Enterprise Services-Anwendung ausgeführt.
2. Der Win32-Dienst ruft eine Startmethode für die Serviced Component auf, mit der die Enterprise Services-Anwendung gestartet und die Serviced Component geladen wird.
3. Die Webanwendung ruft die verschlüsselte Zeichenfolge aus der Datei **Web.config** ab. Sie können die verschlüsselte Zeichenfolge mithilfe des Elements <**appSettings**> in der Datei **Web.config** speichern, wie unten gezeigt. Dieses Element unterstützt beliebige Paare aus Schlüssel und Wert.

```
<configuration>
 <appSettings>
  <add key="SqlConnString"
       value="AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAABcqc/xCHxki3" />
 </appSettings>
</configuration>
```

Sie können die verschlüsselte Zeichenfolge über die folgende Codezeile abrufen:

```
string connString = ConfigurationSettings.AppSettings["SqlConnString"];
```

Hinweis: Sie können die verschlüsselten Verbindungszeichenfolgen in der Datei **Web.config** oder **Machine.config** speichern. Die Datei **Machine.config** wird dabei bevorzugt, da sie sich in einem Systemverzeichnis außerhalb eines virtuellen Verzeichnisses befindet. Dies wird eingehender im nächsten Abschnitt, "Verwenden der Datei **Web.config** und **Machine.config**", erläutert.

4. Die Anwendung ruft eine Methode für die Serviced Component auf, um die Verbindungszeichenfolge zu entschlüsseln.
5. Die Serviced Component arbeitet mit der DPAPI zusammen, um unter Verwendung von **P/Invoke** die Win32-DPAPI-Funktionen aufzurufen.
6. Die entschlüsselte Zeichenfolge wird an die Webanwendung zurückgegeben.

Hinweis: Schreiben Sie eine Hilfsanwendung, die die Verbindungszeichenfolgen übernimmt und die Methode **EncryptData** der Serviced Component aufruft, um die verschlüsselte Zeichenfolge abzurufen, damit Sie diese verschlüsselten Verbindungszeichenfolgen in der Datei **Web.config** an erster Stelle speichern können. Es ist von wesentlicher Bedeutung, dass Sie diese Hilfsanwendung ausführen, während Sie mit demselben Konto angemeldet sind, das Sie zum Ausführen der Enterprise Services-Serveranwendung verwenden.

Direktes Verwenden von DPAPI über ASP.NET

Wenn Sie den Computerspeicher verwenden (und die DPAPI-Funktionen mit dem Parameter CRYPTPROTECT_LOCAL_MACHINE aufrufen), können Sie die DPAPI-Funktionen direkt über eine ASP.NET-Webanwendung aufrufen (da hierzu kein Benutzerprofil erforderlich ist).

Da Sie den Computerspeicher verwenden, besitzt jedoch jedes Windows-Konto, das sich auf diesem Computer anmelden kann, Zugriff auf die vertraulichen Daten. Ein Ansatz zur Risikominderung ist das Hinzufügen der Entropie, wodurch jedoch die zusätzliche Schlüsselverwaltung erforderlich wird.

Erwägen Sie die folgenden Optionen als Alternativen zur Verwendung der Entropie mit dem Computerspeicher:

- Verwenden von Windows-Zugriffssteuerungslisten, um den Zugriff auf verschlüsselte Daten einzuschränken (unabhängig davon, ob die Daten im Dateisystem oder in der Registrierung gespeichert sind).
- Erwägen Sie die feste Kodierung des Entropieparameters in der Anwendung, um das Problem der Schlüsselverwaltung zu vermeiden.

Weitere Informationen

- Weitere Informationen zum Erstellen einer DPAPI-Bibliothek zur Verwendung mit .NET-Webanwendungen finden Sie unter "Vorgehensweise: Erstellen einer DPAPI-Bibliothek" im Abschnitt "Referenz" dieses Handbuchs.
- Eine detaillierte Führung durch die Implementierung, die Ihnen zeigt, wie DPAPI direkt über ASP.NET verwendet wird, finden Sie unter "Vorgehensweise: Verwenden von DPAPI (Computerspeicher) von ASP.NET aus" im Abschnitt "Referenz" dieses Handbuchs.
- Eine detaillierte Führung durch die Implementierung, die Ihnen zeigt, wie DPAPI direkt über Enterprise Services verwendet wird, finden Sie unter "Vorgehensweise: Verwenden von DPAPI (Benutzerspeicher) von ASP.NET aus mit Enterprise Services" im Abschnitt "Referenz" dieses Handbuchs.
- Weitere Informationen zum Windows-Datenschutz mit DPAPI finden Sie in MSDN im Artikel "Windows Data Protection" (englischsprachig).

Verwenden der Datei "Web.config" und "Machine.config"

Es wird nicht empfohlen, unverschlüsselte Kennwörter in der Datei **Web.config** zu speichern. **HttpForbiddenHandler** schützt die Datei standardmäßig davor, von unberechtigten Benutzern gedownloadet und angezeigt zu werden. Dennoch können Benutzer, die direkten Zugriff auf die Ordner besitzen, in denen die Konfigurationsdateien gespeichert werden, weiterhin den Benutzernamen und das Kennwort anzeigen.

Die Datei **Machine.config** wird im Vergleich zu **Web.config** als sicherere Speicherposition betrachtet, da sie sich im Systemverzeichnis (mit Zugriffssteuerungslisten) und außerhalb des virtuellen Verzeichnisses einer Webanwendung befindet. Sperren Sie die Datei **Machine.config** immer unter Verwendung von Zugriffssteuerungslisten.

Weitere Informationen

Weitere Informationen zum Sichern der Datei **Machine.config** finden Sie in Kapitel 8, "ASP.NET-Sicherheit".

Verwenden von UDL-Dateien

Der OLE DB .NET-Datenprovider unterstützt UDL-Dateinamen in seiner Verbindungszeichenfolge. Verwenden Sie innerhalb der Verbindungszeichenfolge den Eintrag **File Name=Name.udl**, um auf eine UDL-Datei zu verweisen.

Wichtig: Diese Option steht nur zur Verfügung, wenn Sie den OLE DB .NET-Datenprovider verwenden, um die Verbindung zur Datenbank herzustellen. Der SQL Server .NET-Datenprovider verwendet keine UDL-Dateien.

Es wird nicht empfohlen, UDL-Dateien in einem virtuellen Verzeichnis zusammen mit anderen Anwendungsdateien zu speichern. Sie sollten sie außerhalb der virtuellen Verzeichnishierarchie einer Webanwendung speichern und die Datei oder den übergeordneten Ordner mithilfe von Windows-Zugriffssteuerungslisten sichern. Sie sollten auch erwägen, UDL-Dateien auf einem vom Betriebssystem getrennten logischen Datenträger zu speichern, um diese vor möglicher Dateivereinheitlichung und Fehlern bei der Verzeichnisdurchquerung zu schützen.

Feinstufigkeit von Zugriffssteuerungslisten

UDL-Dateien (bzw. beliebige Textdateien) bieten eine zusätzliche Feinstufigkeit, wenn Sie, verglichen mit der Verwendung der Datei **Machine.config**, Zugriffssteuerungslisten anwenden. Die der Datei **Machine.config** zugeordneten Standard-Zugriffssteuerungslisten gewähren den Zugriff für eine Vielzahl von lokalen und Remotebenutzern. Die Datei **Machine.config** besitzt z. B. die folgenden Standard-Zugriffssteuerungslisten:

```
MachineName\ASPNET:R
BUILTIN\Users:R
BUILTIN\Power Users:C
BUILTIN\Administrators:F
NT AUTHORITY\SYSTEM:F
```

Dagegen können Sie die UDL-Datei Ihrer eigenen Anwendung umfangreicher vor Zugriffen schützen. Sie können z. B. den Zugriff auf Administratoren, das Systemkonto und das ASP.NET-Prozesskonto (das den Lesezugriff erfordert) einschränken, wie nachfolgend gezeigt:

```
BUILTIN\Administrators:F
MachineName\ASPNET:R
NT AUTHORITY\SYSTEM:F
```

Hinweis: Da UDL-Dateien extern von beliebigen ADO.NET-Clientanwendungen geändert werden können, werden Verbindungszeichenfolgen, die Verweise auf UDL-Dateien enthalten, bei jedem Öffnen der Verbindung analysiert. Dies kann sich auf die Leistung auswirken, und daher wird hinsichtlich der Leistung empfohlen, dass Sie eine statische Verbindungszeichenfolge verwenden, die keine UDL-Datei enthält.

▶ **So erstellen Sie eine neue UDL-Datei:**

1. Verwenden Sie den Windows Explorer, und wechseln Sie zu dem Ordner, in dem Sie die UDL-Datei erstellen möchten.
2. Klicken Sie mit der rechten Maustaste in den Ordner, wählen Sie den Eintrag **Neu** aus, und klicken Sie dann auf **Textdokument**.
3. Stellen Sie einen Dateinamen mit der Dateierweiterung UDL bereit.
4. Doppelklicken Sie auf die neue Datei, um das Dialogfeld **UDL-Eigenschaften** anzuzeigen.

Weitere Informationen

Weitere Informationen zum Verwenden von UDL-Dateien des Entwicklungsprogramms Microsoft C#® finden Sie in der Microsoft Knowledge Base im Artikel Q308426, "HOW TO: Use Data Link Files with OleDbConnection Object in VC#" (US).

Verwenden benutzerdefinierter Textdateien

Viele Anwendungen verwenden benutzerdefinierte Textdateien, um Verbindungszeichenfolgen zu speichern. Wenn Sie diesem Ansatz folgen, berücksichtigen Sie die folgenden Empfehlungen:

- Speichern Sie benutzerdefinierte Dateien außerhalb der virtuellen Verzeichnishierarchie der Anwendung.
- Erwägen Sie, die Dateien auf einem vom Betriebssystem getrennten logischen Datenträger zu speichern, um sie vor möglicher Dateivereinheitlichung und Fehlern bei der Verzeichnisdurchquerung zu schützen.
- Schützen Sie die Datei mit einer eingeschränkten Zugriffssteuerungsliste, die dem Prozesskonto Ihrer Anwendung den Lesezugriff gewährt.
- Vermeiden Sie es, die Verbindungszeichenfolge unverschlüsselt in der Datei zu speichern. Stattdessen sollten Sie die DPAPI verwenden, um eine verschlüsselte Zeichenfolge zu speichern.

Verwenden der Registrierung

Sie können einen benutzerdefinierten Schlüssel in der Windows-Registrierung verwenden, um die Verbindungszeichenfolge zu speichern. Diese gespeicherten Informationen können entweder in der Registrierungsstruktur HKEY_LOCAL_MACHINE (HKLM) oder HKEY_CURRENT_USER (HKCU) eingetragen werden. Für Prozessidentitäten, die nicht über Benutzerprofile verfügen (z. B. das Konto ASPNET), müssen diese Informationen in HKLM gespeichert werden, damit sie vom ASP.NET-Code abgerufen werden können.

Wenn Sie diesen Ansatz verwenden, sollten Sie Folgendes durchführen:

- Verwenden von Zugriffssteuerungslisten, um den Registrierungsschlüssel unter Verwendung der Datei **Regedt32.exe** zu schützen.
- Verschlüsseln der Datei, bevor sie gespeichert wird.

Weitere Informationen

Weitere Informationen zum Verschlüsseln von Daten beim Speichern in der Registrierung finden Sie unter "Vorgehensweise: Speichern von verschlüsselten Verbindungszeichenfolgen in der Registrierung" im Abschnitt "Referenz" dieses Handbuchs.

Verwenden des COM+-Katalogs

Wenn Ihre Webanwendung Serviced Components umfasst, können Sie Verbindungszeichenfolgen im COM+-Katalog als Konstruktorzeichenfolgen speichern. Diese können einfach verwaltet (mit dem Tool für Komponentendienste) und leicht vom Komponentencode abgerufen werden. Die Enterprise Services rufen die **Construct**-Methode eines Objekts unmittelbar nach dem Erstellen einer Instanz des Objekts auf und übergeben die konfigurierte Erstellungszeichenfolge.

Der COM+-Katalog bietet keine hohe Sicherheit, da die Daten nicht verschlüsselt werden. Er bietet jedoch im Vergleich zu Konfigurationsdateien durch den zusätzlichen Prozesshop eine etwas höhere Sicherheit.

Um den Zugriff auf den Katalog über das Tool für Komponentendienste zu vermeiden, beziehen Sie nur die Liste mit den gewünschten Benutzern in die Rollen **Administrator** und **Reader** in der Anwendung **System** ein.

Im folgenden Beispiel wird gezeigt, wie eine Objektkonstruktor-Zeichenfolge aus einer Serviced Component abgerufen wird:

```
[ConstructionEnabled(Default="Default Connection String")]
public class YourClass : ServicedComponent
{
  private string _ConnectionString;
  override protected void Construct(string s)
  {
    _ConnectionString = s;
  }
}
```

Sie können Code zum Verschlüsseln der Verbindungszeichenfolge hinzufügen, bevor diese gespeichert und dann innerhalb der Serviced Component wieder entschlüsselt wird, um die Sicherheit zu erhöhen.

Weitere Informationen

- Weitere Informationen zum Verwenden von Verbindungszeichenfolgen finden Sie in der Microsoft Knowledge Base im Artikel Q271284, "HOWTO: Access COM+ Object Constructor String in a VB Component" (US).
- Ein vom .NET Framework SDK bereitgestelltes vollständiges Codebeispiel finden Sie im Objektkonstruktorbeispiel, das sich unter der folgenden Adresse befindet: **\Programme\Microsoft Visual Studio .NET\FrameworkSDK\Samples\Technologies\ComponentServices\ObjectConstruction**.

Authentifizieren von Benutzern anhand einer Datenbank

Wenn Sie eine Anwendung erstellen, die Benutzeranmeldeinformationen anhand eines Datenbankspeichers überprüfen muss, bedenken Sie die folgenden Punkte:

- Speichern Sie unidirektionale Kennworthashes (mit einem zufälligen Salt-Wert).
- Vermeiden Sie SQL Injection beim Überprüfen von Benutzeranmeldeinformationen.

Speichern von unidirektionalen Kennworthashes (mit Salt-Wert)

Webanwendungen, die die Formularauthentifizierung verwenden, müssen häufig Benutzeranmeldeinformationen (einschließlich der Kennwörter) in einer Datenbank speichern. Aus Sicherheitsgründen sollten Kennwörter (unverschlüsselt oder verschlüsselt) nicht in der Datenbank gespeichert werden.

Sie sollten das Speichern verschlüsselter Kennwörter vermeiden, da der Aufwand für die Schlüsselverwaltung erhöht wird. Sie können das Kennwort durch die Verschlüsselung sichern, müssen dann aber bedenken, wie der Verschlüsselungsschlüssel gespeichert wird. Wenn der Schlüssel unberechtigten Personen zugänglich wird, kann ein Angreifer alle Kennwörter in Ihrem Datenspeicher entschlüsseln.

Nachfolgend ist der bevorzugte Ansatz aufgeführt:

- **Speichern eines unidirektionalen Hashes des Kennworts -** Berechnen Sie den Hash erneut, wenn das Kennwort überprüft werden muss.
- **Kombinieren Sie den Kennworthash mit einem Salt-Wert (einer kryptografisch starken, zufälligen Zahl) -** Durch die Kombination von Salt-Wert und Kennworthash verringern Sie die Bedrohung durch Verzeichnisangriffe.

Erstellen eines Salt-Wertes

Der nachfolgende Code zeigt, wie ein Salt-Wert mithilfe der Generierung von Zufallszahlen erstellt wird, die von der Klasse **RNGCryptoServiceProvider** im Namespace **System.Security.Cryptography** bereitgestellt wird.

```
public static string CreateSalt(int size)
{
  RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
  byte[] buff = new byte[size];
  rng.GetBytes(buff);
  return Convert.ToBase64String(buff);
}
```

Erstellen eines Hashwertes (mit Salt-Wert)

Das folgende Codefragment zeigt, wie Sie einen Hashwert aus einem bereitgestellten Kennwort und dem Salt-Wert generieren:

```
public static string CreatePasswordHash(string pwd, string salt)
{
  string saltAndPwd = string.Concat(pwd, salt);
  string hashedPwd =
        FormsAuthentication.HashPasswordForStoringInConfigFile(
                                              saltAndPwd, "SHA1");
  return hashedPwd;
}
```

Weitere Informationen

Die ausführlichen Implementierungsdetails dieses Ansatzes finden Sie unter "Vorgehensweise: Verwenden der Formularauthentifizierung mit SQL Server 2000" im Abschnitt "Referenz" dieses Handbuchs.

SQL Injection-Angriffe

Wenn Sie die Formularauthentifizierung anhand einer SQL-Datenbank verwenden, sollten Sie die in diesem Abschnitt beschriebenen Maßnahmen ergreifen, um SQL Injection-Angriffe zu vermeiden. SQL Injection beschreibt das Übergeben zusätzlichen (unberechtigten) SQL-Codes in eine Anwendung, der dabei normalerweise an den regulären SQL-Code angehängt wird, der in der Anwendung enthalten ist. Alle SQL-Datenbanken sind in unterschiedlichem Maße anfällig für SQL Injection, wobei der Fokus in diesem Kapitel auf SQL Server gerichtet ist.

Sie sollten besonders auf die Möglichkeit für potenzielle SQL Injection-Angriffe achten, wenn Sie Benutzereingaben verarbeiten, die Teile eines SQL-Befehls bilden. Wenn Ihr Authentifizierungsschema auf dem Überprüfen von Benutzern anhand einer SQL-Datenbank basiert (z. B. wenn Sie die Formularauthentifizierung anhand von SQL Server verwenden), dann müssen Sie sich vor SQL Injection-Angriffen schützen.

Wenn Sie SQL-Zeichenfolgen aus ungefilterten Eingaben bilden, kann Ihre Anwendung Ziel böswilliger Benutzereingaben sein (denken Sie daran, dass Sie niemals Benutzereingaben vertrauen sollten). Das Risiko besteht darin, dass ein böswilliger Benutzer SQL-Befehle an Ihre geplanten SQL-Anweisungen anhängen kann (mithilfe von Escapezeichen), wenn Sie Benutzereingaben in eine Zeichenfolge einfügen, die zu einer ausführbaren Anweisung wird.

Die Codefragmente in den folgenden Abschnitten verwenden die Datenbank **Pubs**, die mit SQL Server bereitgestellt wird, um Beispiele für SQL Injection zu veranschaulichen.

Das Problem

Wenn Sie Benutzereingaben oder andere unbekannte Daten in Datenbankabfragen einbeziehen, kann Ihre Anwendung für SQL Injection-Angriffe anfällig sein. Beispielsweise sind die beiden nachfolgenden Codefragmente für Angriffe anfällig.

- Sie erstellen SQL-Anweisungen aus ungefilterten Benutzereingaben.

```
SqlDataAdapter myCommand = new SqlDataAdapter(
          "SELECT au_lname, au_fname FROM authors WHERE au_id = '" +
          Login.Text + "'", myConnection);
```

- Sie rufen eine gespeicherte Prozedur auf, indem Sie eine einzelne Zeichenfolge erstellen, die ungefilterte Benutzereingaben einbezieht.

```
SqlDataAdapter myCommand = new SqlDataAdapter("LoginStoredProcedure '" +
                                Login.Text + "'", myConnection);
```

Aufbau eines Injection-Angriffs mit SQL Script

Wenn Sie ungefilterte Benutzereingabewerte (wie oben veranschaulicht) in Ihrer Anwendung akzeptieren, kann ein böswilliger Benutzer Escapezeichen verwenden, um eigene Befehle anzuhängen.

Stellen Sie sich eine SQL-Abfrage vor, die vom Benutzer eine Eingabe in Form einer Rentenversicherungsnummer erfordert, z. B. *172-32-xxxx*, die zu einer Abfrage wie der Folgenden führt:

```
SELECT au_lname, au_fname FROM authors WHERE au_id = '172-32-xxxx'
```

Ein böswilliger Benutzer kann den folgenden Text in das Eingabefeld der Anwendung eingeben (z. B. in ein Textfeldsteuerelement).

```
' ; INSERT INTO jobs (job_desc, min_lvl, max_lvl) VALUES ('Important Job', 25, 100)  -
```

In diesem Beispiel wird die Anweisung INSERT eingefügt (es kann aber jede Anweisung ausgeführt werden, die für das Konto zugelassen ist, das zum Herstellen der Verbindung zu SQL Server verwendet wird). Der Code kann besonders schädlich sein, wenn das Konto Mitglied der Rolle **sysadmin** ist (dies ermöglicht Shellbefehle, die über **xp_cmdshell** ausgeführt werden können) und SQL Server unter einem Domänenkonto ausgeführt wird, das Zugriff auf andere Netzwerkressourcen hat.

Die o. a. Befehle führen zu der folgenden kombinierten SQL-Zeichenfolge:

```
SELECT au_lname, au_fname FROM authors WHERE au_id = '';INSERT INTO jobs (job_desc,
min_lvl, max_lvl) VALUES ('Important Job', 25, 100)  --
```

In diesem Fall beendet das Zeichen **'** (einfaches Anführungszeichen), das die unberechtigte Eingabe anführt, das aktuelle Zeichenfolgenliteral in Ihrer SQL-Anweisung. Es schließt die aktuelle Anweisung nur, wenn das folgende analysierte Token als Fortsetzung der aktuellen Anweisung keinen Sinn macht, während es jedoch als Beginn einer neuen Anweisung geeignet ist.

```
SELECT au_lname, au_fname FROM authors WHERE au_id = ' '
```

Das Zeichen **;** (Semikolon) teilt SQL mit, dass Sie eine neue Anweisung beginnen, auf die dann der unberechtigte SQL-Code folgt:

```
; INSERT INTO jobs (job_desc, min_lvl, max_lvl) VALUES ('Important Job', 25, 100)
```

Hinweis: Das Semikolon ist nicht unbedingt erforderlich, um SQL-Anweisungen voneinander zu trennen. Dies hängt vom Anbieter oder von der Implementierung ab, wird jedoch nicht von SQL Server erwartet. Nachfolgender Code wird z. B. von SQL Server als zwei separate Anweisungen analysiert:

```
SELECT * FROM MyTable DELETE FROM MyTable
```

Abschließend weist die Zeichenfolge **- -** (doppelter Strich) SQL an, den Rest des Textes zu ignorieren, wodurch in diesem Fall das abschließende Zeichen **'** (einfaches Anführungszeichen) ignoriert wird (das ansonsten zu einem SQL-Analysefehler führen würde).

Der vollständige Text, den SQL als Ergebnis der o. a. Anweisung ausführt, lautet wie folgt:

```
SELECT au_lname, au_fname FROM authors WHERE au_id = '' ; INSERT INTO jobs (job_desc,
min_lvl, max_lvl) VALUES ('Important Job', 25, 100) --'
```

Die Lösung

Sie können die folgenden Ansätze verwenden, um SQL sicher über Ihre Anwendung aufzurufen.

- Verwenden Sie die Auflistung **Parameter**, wenn Sie die SQL-Anweisungen erstellen.

```
SqlDataAdapter myCommand = new SqlDataAdapter(
        "SELECT au_lname, au_fname FROM Authors WHERE au_id= @au_id",
        myConnection);

SqlParameter parm = myCommand.SelectCommand.Parameters.Add(
                                            "@au_id",
                                            SqlDbType.VarChar, 11);
parm.Value= Login.Text;
```

- Verwenden Sie die Auflistung **Parameter**, wenn Sie eine gespeicherte Prozedur aufrufen.

```
// AuthorLogin is a stored procedure that accepts a parameter named Login
SqlDataAdapter myCommand = new SqlDataAdapter("AuthorLogin", myConnection);
myCommand.SelectCommand.CommandType = CommandType.StoredProcedure;
SqlParameter parm = myCommand.SelectCommand.Parameters.Add(
                                "@LoginId", SqlDbType.VarChar,11);
parm.Value=Login.Text;
```

  Wenn Sie die Auflistung **Parameter** verwenden, wird die Eingabe als Liberal behandelt, unabhängig davon, was ein böswilliger Benutzer als Eingabe einbezieht. Ein weiterer Vorteil bei der Verwendung der Auflistung **Parameter** ist es, dass Sie Typ- und Längenüberprüfungen erzwingen können. Werte, die sich außerhalb des Bereichs befinden, lösen eine Ausnahmebedingung aus. Dies ist ein gutes Beispiel für eine tief greifende Abwehr.

- Filtern Sie SQL-Zeichen aus Benutzereingaben. In der folgenden Methode wird gezeigt, wie Sie sicherstellen, dass ein in einer einfachen SQL-Vergleichsanweisung (ist gleich, kleiner als, größer als) verwendetes Zeichenfolgenliteral sicher ist. Dies erfolgt dadurch, dass ein in der Zeichenfolge verwendetes Anführungszeichen durch ein weiteres Anführungszeichen außer Kraft gesetzt wird. In einem SQL-Zeichenfolgenliteral werden zwei aufeinanderfolgende Anführungszeichen als Instanz des Anführungszeichens innerhalb der Zeichenfolge behandelt und nicht als Trennzeichen.

```
private string SafeSqlLiteral(string inputSQL)
{
  return inputSQL.Replace("'", "''");
}
…
string safeSQL = SafeSqlLiteral(Login.Text);
SqlDataAdapter myCommand = new SqlDataAdapter(
       "SELECT au_lname, au_fname FROM authors WHERE au_id = '" +
       safeSQL + "'", myConnection);
```

Weitere empfohlene Vorgehensweisen

Nachfolgend finden Sie einige zusätzliche Maßnahmen, mit denen Sie die Möglichkeit einschränken können, dass Ihr Code unberechtigt geändert wird und mit denen Sie auch das Ausmaß eines potenziellen Schadens einschränken können:

- Verhindern Sie ungültige Eingaben am Gate (die Front-End-Anwendung), indem Sie die Größe und den Typ der Eingabe einschränken. Durch die Einschränkung der Größe und des Typs der Eingabe verringern Sie die Möglichkeit, dass Schaden angerichtet werden kann. Wenn das Suchfeld Ihrer Datenbank z. B. eine Länge von elf Zeichen besitzt und nur aus numerischen Zeichen besteht, sollten Sie eine Eingabe in dieser Form erzwingen.
- Führen Sie SQL-Code mit einem Konto aus, das minimale Rechte besitzt. Dadurch wird das Ausmaß möglicher Schäden erheblich verringert.

  Wenn ein unberechtigter Benutzer z. B. eine SQL-Anweisung so verändert hat, dass diese versucht, eine Tabelle über den Befehl DROP aus der Datenbank zu entfernen, die SQL-Verbindung jedoch ein Konto verwendet, das nicht über die entsprechenden Berechtigungen verfügt, schlägt der SQL-Code fehl. Dies ist ein weiterer Grund dafür, nicht das Konto **sa** oder das Konto des Datenbankeigentümers für die SQL-Verbindungen Ihrer Anwendung zu verwenden.
- Zeigen Sie dem Endbenutzer nicht die von der Datenbank ausgelösten SQL-Fehler an, wenn im SQL-Code eine Ausnahmebedingung auftritt. Protokollieren Sie die Fehlerinformationen, und zeigen Sie nur benutzerfreundliche Informationen an. Dadurch wird verhindert, das unnötige Details offen gelegt werden, die einem Angreifer weiterhelfen könnten.

Schützen von Anweisungen für den Mustervergleich

Wenn Eingaben innerhalb eines Zeichenfolgenliterals in einer LIKE-Klausel verwendet werden sollen, besitzen Zeichen, bei denen es sich nicht um das Anführungszeichen handelt, eine bestimmte Bedeutung für den Mustervergleich.

In einer LIKE-Klausel bedeutet das Zeichen **%** z. B. "Übereinstimmung mit Null oder mehr Zeichen". Damit diese Zeichen in der Eingabe als Literalzeichen ohne besondere Bedeutung behandelt werden, müssen sie ebenfalls außer Kraft gesetzt werden. Wenn sie nicht auf besondere Weise behandelt werden, kann die Abfrage falsche Ergebnisse liefern. Ein nicht außer Kraft gesetztes Zeichen für den Mustervergleich kann auch die Indexerstellung verhindern, wenn es am Anfang einer Zeichenfolge steht.

Für SQL Server sollte die folgende Methode verwendet werden, um zulässige Eingaben sicherzustellen:

```
private string SafeSqlLikeClauseLiteral(string inputSQL)
{
  // Make the following replacements:
  // '  becomes  ''
  // [  becomes  [[]
  // %  becomes  [%]
  // _  becomes  [_]

  string s = inputSQL;
  s = inputSQL.Replace("'", "''");
  s = s.Replace("[", "[[]");
  s = s.Replace("%", "[%]");
  s = s.Replace("_", "[_]");
  return s;
}
```

Überwachung

Die Überwachung von Anmeldevorgängen ist in SQL Server nicht standardmäßig aktiviert. Sie können dies entweder über den SQL Server Enterprise Manager oder über die Systemregistrierung konfigurieren. Das Dialogfeld in Abbildung 12.7 zeigt an, dass die Überwachung sowohl für erfolgreiche als auch für fehlgeschlagene Anmeldeversuche aktiviert ist.

Die Protokolleinträge werden in die SQL-Protokolldateien geschrieben, die sich standardmäßig im Verzeichnis **C:\Programme\Microsoft SQL Server\MSSQL\LOG** befinden. Sie können ein beliebiges Textprogramm verwenden, um diese anzuzeigen, z. B. den Editor von Windows.

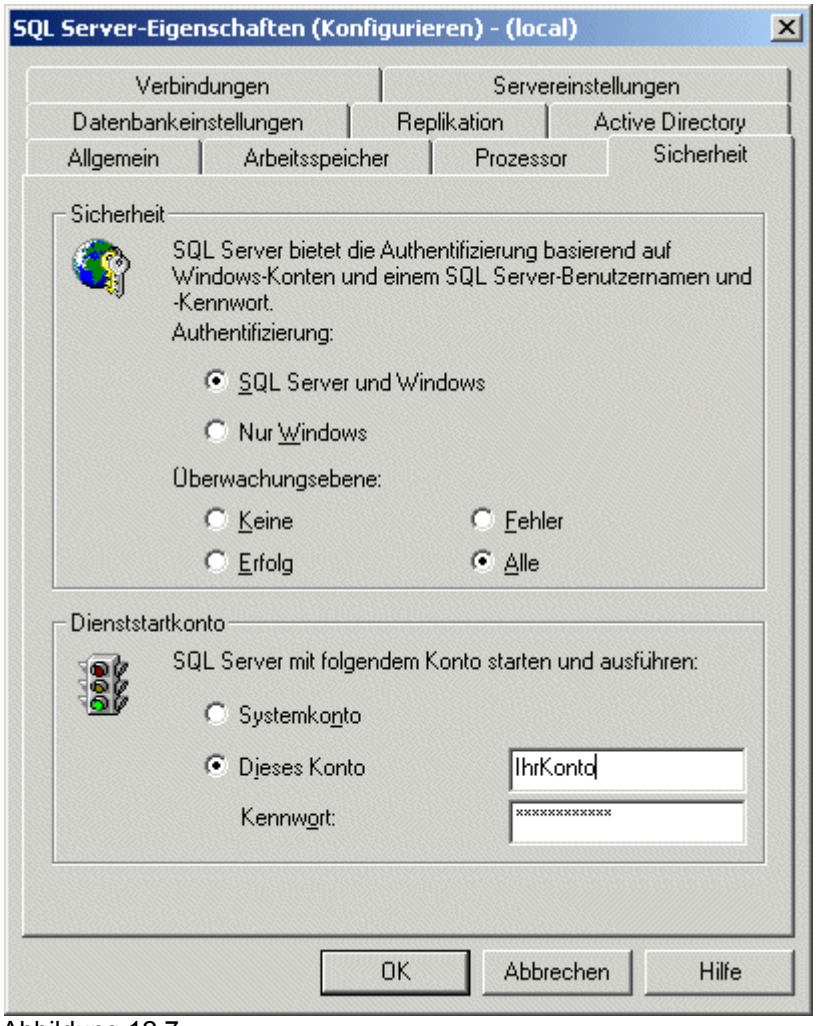

Abbildung 12.7

*Dialogfeld **SQL Server-Eigenschaften** mit den Einstellungen für die Überwachungsebene*

Sie können die Überwachung für SQL Server auch in der Systemregistrierung aktivieren. Wenn Sie die Überwachung für SQL Server aktivieren möchten, erstellen Sie den folgenden **AuditLevel**-Schlüssel in der Systemregistrierung, und legen Sie als Wert einen der nachfolgend angegebenen REG_DWORD-Werte fest:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\AuditLevel
```

Sie können einen der folgenden Werte auswählen, mit denen Sie die gewünschte Detailebene bestimmen können.

3 – Erfasst sowohl erfolgreiche als auch fehlgeschlagene Anmeldeversuche

2 – Erfasst nur fehlgeschlagene Anmeldeversuche

1 – Erfasst nur erfolgreiche Anmeldeversuche

0 – Erfasst keine Anmeldungen

Es wird empfohlen, dass Sie die Überprüfung auf fehlerhafte Anmeldeversuche aktivieren, da Sie so ermitteln können, wenn ein Brute-Force-Angriff auf einen SQL Server versucht wird. Die Auswirkungen der Protokollierung von fehlgeschlagenen Überwachungsversuchen auf die Leistung sind minimal, wenn Sie nicht angegriffen werden. In diesem Fall müssen Sie es sowieso erfahren.

Sie können auch ein Skript für SQL Database Management Objects (DMO, Datenbankverwaltungsobjekte) erstellen. Das folgende Codefragment zeigt einen VBScript-Beispielcode:

```
Sub SetAuditLevel(Server As String, NewAuditLevel As SQLDMO_AUDIT_TYPE)
    Dim objServer As New SQLServer2
    objServer.LoginSecure = True     'Use integrated security
    objServer.Connect Server        'Connect to the target SQL Server
    'Set the audit level
    objServer.IntegratedSecurity.AuditLevel = NewAuditLevel
    Set objServer = Nothing
End Sub
```

Gemäß der SQL Server-Onlinedokumentation lauten die Mitglieder des Aufzählungstyps SQLDMO_AUDIT_TYPE wie folgt:

```
SQLDMOAudit_All      3  Log all authentication attempts regardless of success
                        or failure
SQLDMOAudit_Failure  2   Log failed authentication
SQLDMOAudit_Success  1   Log successful authentication
SQLDMOAudit_None     0   Do not log authentication attempts
```

Prozessidentität für SQL Server

Führen Sie SQL Server mithilfe eines Domänenkontos mit minimalen Rechten aus. Wenn Sie SQL Server installieren, haben Sie die Möglichkeit, den SQL Server-Dienst unter Verwendung des lokalen Kontos SYSTEM oder mit einem angegebenen Konto auszuführen.

Verwenden Sie nicht das Konto SYSTEM oder ein Administratorkonto. Verwenden Sie stattdessen ein Domänenkonto mit minimalen Rechten. Sie müssen diesem Konto keine speziellen Rechte gewähren, da dem Konto bei der Installation (oder über den SQL Server Enterprise Manager, wenn Sie den SQL-Dienst nach der Installation erneut konfigurieren) die erforderlichen Rechte gewährt werden.

Zusammenfassung

Nachfolgend finden Sie eine Zusammenfassung, in der die Empfehlungen für den Datenzugriff in .NET-Webanwendungen hervorgehoben sind:

- Verwenden Sie nach Möglichkeit für SQL Server die Windows-Authentifizierung.
- Verwenden Sie in der Datenbank Konten mit minimalen Rechten.
- Verwenden Sie zum Ausführen von ASP.NET/Enterprise Services lokale Konten mit minimalen Rechten, wenn Sie die Verbindung zu SQL Server herstellen.
- Wenn Sie die SQL-Authentifizierung verwenden, führen Sie die folgenden Schritte durch, um die Sicherheit zu erhöhen:
  - Verwenden Sie benutzerdefinierte Konten mit starken Kennwörtern.
  - Schränken Sie die Berechtigungen der einzelnen Konten in SQL Server unter Verwendung von Datenbankrollen ein.
  - Fügen Sie Zugriffssteuerungslisten zu Dateien hinzu, in denen Verbindungszeichenfolgen gespeichert werden.
  - Verschlüsseln Sie Verbindungszeichenfolgen.
  - Erwägen Sie die Verwendung der DPAPI zum Speichern von Anmeldeinformationen.
- Wenn Sie für SQL die Formularauthentifizierung verwenden, treffen Sie Maßnahmen, um SQL Injection-Angriffe zu vermeiden.
- Speichern Sie zur Überprüfung von Benutzern nicht die Benutzerkennwörter in Datenbanken. Speichern Sie Kennworthashes mit einem Salt-Wert, anstatt unverschlüsselte oder verschlüsselte Kennwörter zu verwenden.
- Schützen Sie vertrauliche Daten, die an oder von SQL Server über das Netzwerk gesendet werden.
  - Die Windows-Authentifizierung schützt Anmeldeinformationen, jedoch keine Anwendungsdaten.
  - Verwenden Sie IPSec oder SSL.