APEX? Aber sicher!
Tipps und Tricks für eine sichere APEX-Umgebung

Carsten Czarski
ORACLE Deutschland B.V. & Co KG
München

Schlüsselworte

APEX, Sicherheit

Einleitung

Mit der zunehmenden Popularität von APEX rückt das Thema *Sicherheit* mehr und mehr in den Vordergrund. Für APEX-Anwendungen ergeben sich dabei zwei Themenbereiche: Zum einen hängt die Sicherheit einer APEX-Anwendung sehr stark davon ab, inwieweit der Entwickler Sicherheitslöcher durch *Cross-Site-Skripting* oder *SQL Injection* gar nicht erst auftreten lässt. Zum anderen ist jede APEX-Anwendung bekanntlich durch ihre Metadaten definiert, die von der APEX-Laufzeitumgebung ausgeführt werden. Insofern gibt es auch für den Betrieb der APEX-Instanz Sicherheitsaspekte zu beachten.

Dieser Artikel und der zugehörige DOAG2013 Vortrag stellen für APEX-Umgebungen relevante Sicherheitsaspekte vor. Der Fokus liegt allein auf APEX – allgemeine Aspekte zum sicheren Betrieb von Oracle-Datenbanken oder die Sicherheitsaspekte des APEX-Webservers würden den Rahmen sprengen und bleiben deshalb außen vor.

APEX Sessionkonzept: Welcher Datenbankuser ist eigentlich angemeldet?

Das APEX-Sessionkonzept sorgt recht häufig für Verwirrung unter Anwendern. Es wird recht schnell deutlich, dass die eigentliche, physikalische Datenbankverbindung als **APEX_PUBLIC_USER** bzw. **ANONYMOUS** erfolgt – müssen diese Accounts doch im Webserver konfiguriert bzw. (für das *PL/SQL Embedded Gateway*) entsperrt werden. Und obgleich diese Accounts nur mit minimalen Privilegien ausgestattet sind, können alle APEX-Anwendungen SQL-Anweisungen erfolgreich ausführen. Zunächst sei also geklärt, wie genau APEX sich auf die Datenbank verbindet und mit welchen Rechten die SQL-Anweisungen jeweils ablaufen.

An einer APEX-Anwendung sind vier Datenbankschemas beteiligt:

1. Das Schema **APEX_XXXXXX** (für APEX 4.2: **APEX_040200**) enthält die "APEX Engine"; es ist recht hoch privilegiert, aber gesperrt. Das Entsperren von **APEX_XXXXXX** ist normalerweise nicht erforderlich und sollte unterbleiben.

2. Das Schema **FLOWS_FILES** enthält die Tabelle für hochgeladene Dateien. Auch dieses Schema ist gesperrt und sollte für Datenbankverbindungen nicht verwendet werden.

3. Das Schema **APEX_PUBLIC_USER** (oder **ANONYMOUS**) ist, wie schon beschrieben, nur mit einem CREATE SESSION-Privileg ausgestattet. Zusätzliche Privilegien sollten diesem Account nicht eingeräumt werden, da alle APEX-Zugriffe über diesen Account erfolgen. Würde man **APEX_PUBLIC_USER** mit einem SELECT-Privileg auf eine Tabelle ausstatten, hätten *alle* APEX-Sitzungen Zugriff auf diese Tabelle.

4. Das *Parsing Schema* der jeweiligen APEX-Anwendung schließlich hält die Datenbankobjekte (Tabellen, Views, PL/SQL) der APEX-Anwendung. APEX stellt sicher, dass alle SQL-Anweisungen mit den Rechten dieses Parsing Schemas ausgeführt werden.

Wird eine APEX-Anwendung im Browser aufgerufen, so erfolgt die "physikalische" Datenbankverbindung als APEX_PUBLIC_USER. APEX-Aufrufe zielen meist auf die Prozeduren **F** (bei einem Seitenabruf) oder **WWV_FLOW.ACCEPT** beim Absenden eines Formulars (*Page Submit*). Die jeweiligen Parameter der Prozedur bestimmten die Details der APEX-Session wie APEX-Anwendung, APEX-Seite, Session-ID und andere.

Die Prozeduren der APEX-Engine "gehören" dem APEX-Eigentümer **APEX_XXXXXX**. Nun werden die Anwendungsdefinitionen aus dem Repository gelesen und das SQL ermittelt, welches (bspw. zur Darstellung eines Reports) ausgeführt werden soll. An dieser Stelle bedient sich APEX eines Kniffs: Das Oracle-interne PL/SQL-Paket **DBMS_SYS_SQL** arbeitet wie das bekannte **DBMS_SQL**, nimmt jedoch einen zusätzlichen Parameter entgegen. Mit diesem wird das Datenbankschema bestimmt, *mit dessen Rechten* das SQL ausgeführt werden soll. Das Anwendungs-SQL wird zwar von APEX_XXXXXX ausgeführt, faktisch sind aber die Rechte des *Parsing-Schemas* der Anwendung aktiv. Diese Vorgehensweise stellt sicher, dass jede Anwendung nur die Rechte in Anspruch nehmen kann, die dessen *Parsing Schema* eingeräumt wurden. Für den reinen APEX-Betrieb kann das Parsing Schema der Anwendung sogar gesperrt werden (ACCOUNT LOCK), da die Datenbankverbindung ja als APEX_PUBLIC_USER erfolgt.

Sicherheitsaspekte beim Betrieb einer APEX-Instanz

Nach dem Einloggen am Workspace **INTERNAL** gelangt man zu den *Sicherheitsattributen* der APEX-Instanz per Klick auf **Instanz verwalten** und dann auf **Sicherheit**. Abbildung 1 zeigt den sich darauf öffnenden Dialog.

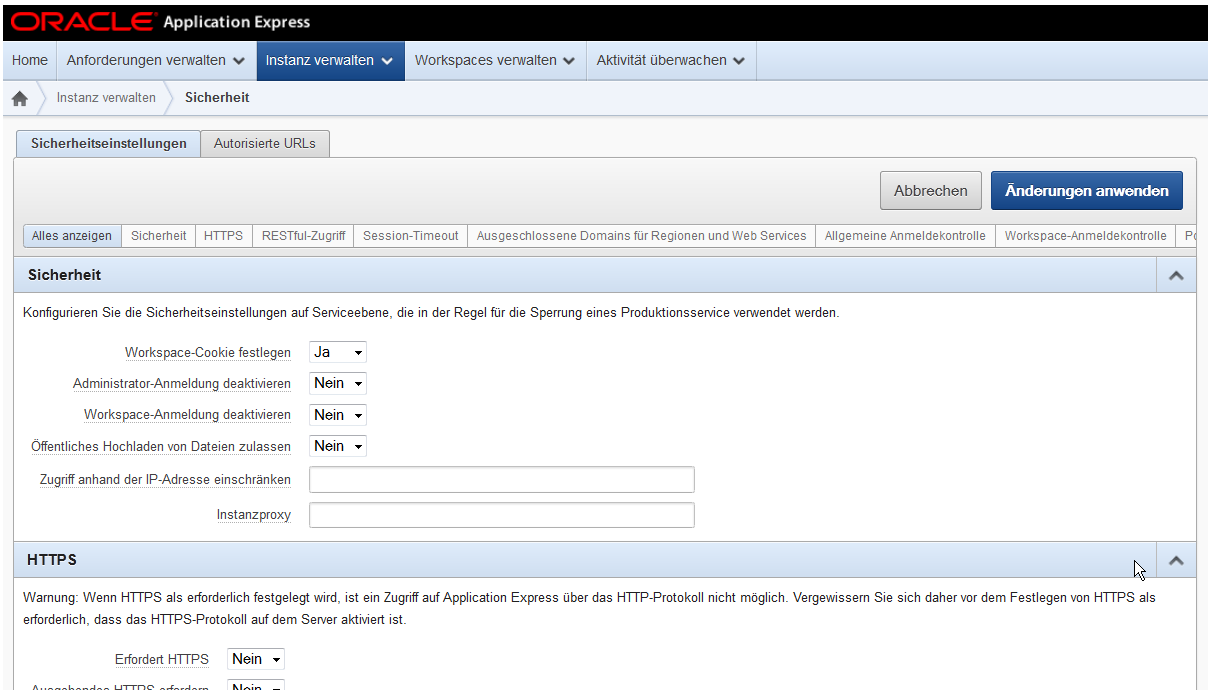

Abbildung 1: Sicherheitsattribute der APEX-Instanz

An dieser Stelle können verschiedene, sicherheitsrelevante Einstellungen für die APEX-Instanz vorgenommen werden. Folgerichtig gelten Sie für *alle* APEX-Anwendungen. Einige Beispiele sind:

- Admin-Login auf Workspace- oder Instanzebene ein- und ausschalten
- Datei-Upload für nicht authentifizierte Nutzer aktivieren oder deaktivieren
- Maximale Dauer einer APEX-Session einstellen
- Verhalten bei fehlerhaftem Login einrichten

Ob man die Einstellungen eher restriktiv oder offen gestaltet, hängt sicherlich von der Art der APEX-Instanz ab (bspw. Entwicklungs- oder Produktionsumgebung). Die von APEX mitgebrachten Standardeinstellungen versuchen einen Kompromiss zwischen Sicherheit und sofortiger Lauffähigkeit: So ist der Datei-Upload für nicht authentifizierte Nutzer verboten. Dagegen wird HTTPS *nicht* erzwungen, denn hierfür braucht es noch Anpassungen am Webserver (SSL-Zertifikat), die man nicht überall voraussetzen kann. Nach Aufsetzen des Servers empfiehlt es sich also, diesen Bereich an die eigenen Bedürfnisse anzupassen. Wenn möglich, sollte der Webserver für SSL aufgesetzt und SSL erzwungen werden; denn Netzwerkverschlüsselung sollte inzwischen Standard sein – auch im Intranet.

Passwort-Policies sollte man anpassen – so ist das Wort "oracle" standardmäßig verboten; gleiches sollte für den eigenen Firmennamen gelten. Generell gilt es, die Passwort-Policy an die Gegebenheiten im Unternehmen anzupassen.

Für die Einstellung des Session-Timeouts sind allgemeine Empfehlungen schwierig; der Default von 8 Stunden dürfte für viele Produktionsumgebungen aber zu lang sein. Hierbei ist zu beachten, dass jeder Anwendungsentwickler diesen Wert in seiner Anwendung übersteuern kann.

Ebenfalls erwähnenswert ist die Möglichkeit, die APEX-Entwicklungsumgebung auf einem Produktionssystem zu deinstallieren. Das geschieht, indem das im APEX-Download enthaltene Skript **apxdevrm.sql** als SYS aufgerufen wird. Anschließend kann die APEX-Instanz nur noch APEX-Anwendungen ausführen; ein Login am Workspace ist nicht mehr möglich. Auch die APEX-Administrationsoberfläche (Workspace INTERNAL) ist dann nicht mehr vorhanden – die APEX-Administration muss dann mit den PL/SQL-Paketen **APEX_INSTANCE_ADMIN** und **APEX_UTIL** erfolgen – Installation oder Upgrade von Anwendungen muss mit SQL*Plus erfolgen.

Der Sicherheitseffekt ist einleuchtend: Da kein Workspace-Login mehr möglich ist, ist es auch nicht mehr möglich, die Anwendung in Produktion – absichtlich oder unabsichtlich – zu ändern.

Sicherheitsaspekte im APEX Workspace

Auch der Administrator eines APEX-Workspace kann Einstellungen zur Sicherheit vornehmen. Dies betrifft im wesentlichen die Nutzerverwaltung. Alle anderen Attribute können auf Ebene der APEX-Anwendung eingerichtet werden.

APEX-Workspaces sind so konzipiert, dass die Nutzerverwaltung dezentral erfolgt; jeder Workspace-Administrator verwaltet seine Nutzer selbst. APEX unterscheidet dabei drei Nutzertypen:

- **Administratoren** haben Vollzugriff auf den Workspace – sie können prinzipiell alle Anwendungen nutzen, bearbeiten und auch den Workspace administrieren (also neue Benutzer anlegen)
- **Entwickler** können Anwendungen erstellen, verändern oder löschen. Auf den Administrationsbereich haben sie keinen Zugriff. Es ist allerdings *nicht* möglich, bestimmte Nutzer nur für bestimmte Anwendungen freizuschalten: Alle Entwickler können auf allen Anwendungen eines Workspace arbeiten.

- **Endbenutzer** dagegen können sich lediglich anmelden und damit eine vorhandene Anwendung (welche die Workspace-Nutzer zur Authentifizierung nutzt) verwenden.

Derzeit ist der Login am Workspace (und damit das Entwickeln einer APEX-Anwendung) nur mit den im Workspace verwalteten Nutzerkonten möglich. In einem Workspace muss also zumindest für jeden Entwickler ein Nutzerkonto eingerichtet werden.

Sicherheitsaspekte auf Ebene der APEX-Anwendung

Authentifizierung der Endbenutzer

Für jede APEX-Anwendung wird ein *Authentifizierungsschema* festgelegt – es bestimmt, wie sich Endbenutzer an dieser Anwendung anmelden. Beim Erstellen der Anwendung wird meist **Application Express** voreingestellt, was bedeutet, dass man sich mit den im APEX-Workspace hinterlegten Nutzernamen anmelden muss. Was für die Entwicklung sicherlich noch in Ordnung geht, sollte jedoch spätestens in Produktion geändert werden: Zu empfehlen ist die Nutzung zentraler Authentifizierungsdienste im Unternehmen. Ein LDAP-Server sollte nahezu überall vorhanden sein; und die Nutzung eines solchen zur Anmeldung an einer APEX-Anwendung ist denkbar einfach.

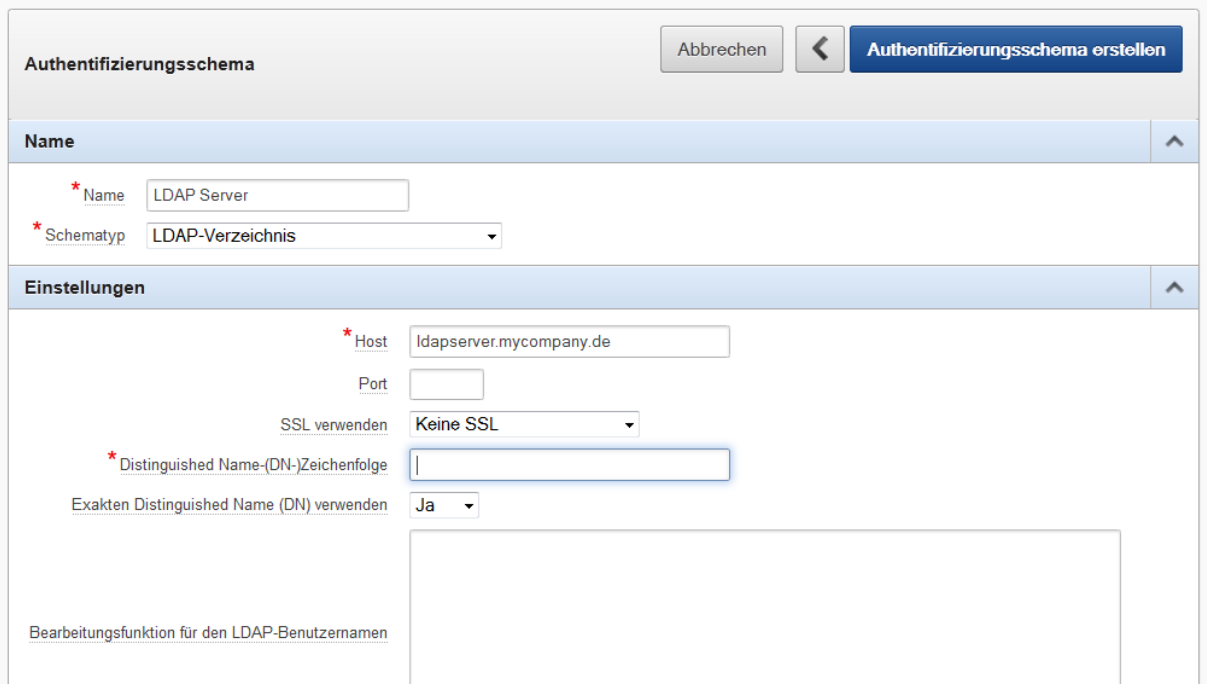

Abbildung 2: Nutzung eines LDAP Servers zur Authentifizierung

Der Vorteil liegt auf der Hand: Passwörter, die man nicht verwaltet, können auch nicht durch Schwächen in der eigenen Anwendung ausgespäht werden – und da die Endbenutzer mit ihrem Standardpasswort arbeiten, sinkt das Risiko unsicher aufbewahrter Passwörter.

Über LDAP hinaus ist auch die Nutzung vorhandener Infrastrukturen zum Single Sign On möglich.

Sicherheitseinstellungen

Die Sicherheitsattribute einer Anwendung werden im Bereich **Sicherheit** in den **Gemeinsamen Komponenten** eingestellt. Dort lassen sich unter anderem festlegen ...

- **Session-Timeout**: die in der Instanz-Administration global hinterlegten Werte können hier für die Anwendung überschrieben werden
- **Browser Security**: Dort wird festgelegt, ob das Einbetten der Anwendung in einen IFRAME erlaubt ist und ob der Browser die Seiten cachen darf. Dies erfordert natürlich, dass die verwendeten Browser das entsprechend unterstützen, was bei modernen Browsern der Fall ist.
- **Session State Protection**: Wird diese installiert, so werden die APEX-URLs mit einer zusätzlichen Prüfsumme ausgestattet, um URL-Manipulationen zu verhindern. Prinzipiell ist dies sehr zu empfehlen, es bringt aber auch Mehraufwand für den Anwendungsentwickler, denn dieser muss nun bei jeder selbst konstruierten URL die Checksumme mit einbauen (**APEX_UTIL.PREPARE_URL**).

Eine ausführliche Betrachtung aller möglichen Einstellungen würde den Rahmen dieses Artikels sprengen – zu jeder Einstellung enthält APEX teilweise sehr ausführliche Hilfetexte.

Sicherheitsaspekte bei der Anwendungsentwicklung

Den größten Einfluß auf die Sicherheit einer Anwendung haben allerdings nicht die bis hierher besprochenen Einstellungen auf APEX-Instanz-, Workspace- oder Anwendungsebene. Vielmehr ist es der Entwickler, der mit konsequenter, sicherheitsorientierter Entwicklung ganz erheblichen Einfluß auf die Sicherheit der Umgebung hat.

Durchgängigkeit

Um eine Anwendung sicher zu gestalten, ist es immens bedeutsam, dass alle Programmierkonzepte *durchgängig* eingesetzt werden. Das beginnt beim Datenmodell und endet bei der APEX-Oberfläche. Die Entwicklung in (logischen) Schichten ist vielfach bereits gängige Praxis: Der Grundsatz, dass keine Schicht der anderen "trauen" sollte, ist tatsächlich auch sicherheitsrelevant: Denn viele Angriffsvektoren basieren darauf, dass ungültige Werte an die Anwendung übergeben werden, um fehlerhafte Reaktionen zu provozieren. Dadurch kann entweder bösartiger Code zur Ausführung gebracht oder wertvolle Informationen gewonnen werden.

Um dem zu begegnen, empfiehlt sich ein konsequentes logisches Schichtenmodell. Integritätsregeln sollten dabei auf allen Ebenen implementiert werden:

- Auf Tabellenebene setzen *Datenbank-Constraints* durch, dass nur gültige Daten gespeichert werden können.
- Tabellenzugriffe finden per View oder PL/SQL-Paket statt. Auch dort werden alle Parameter, trotz des vorhandenen Constraints, *nochmals* geprüft.
- In der APEX-Anwendungen werden Formulareingaben mit *APEX Validations* geprüft – auf die darunterliegenden Prüfungen verlässt sich die APEX-Anwendung nicht.
- Gegebenenfalls findet im Browser sogar noch eine *clientseitige Validierung* per JavaScript statt – diese ersetzt aber keinesfalls die serverseitigen Validierungen

So wird der Effekt eventueller Fehler minimiert. Es mag nun sein, dass an einer Stelle ein Constraint vergessen wurde und an anderer Stelle ein PL/SQL Parameter nicht korrekt geprüft wird. Um tatsächlich zum Problem zu werden, müsste diese Prüfung nun wirklich *in allen* Schichten vergessen worden sein – was sicherlich nur selten auftreten dürfte.

Das gilt analog auch für die APEX-Komponenten selbst. Enthält eine APEX-Anwendung bspw. eine Schaltfläche, die bei Klick einen PL/SQL-Prozess auslöst, so reicht es nicht aus, nur die Schaltfläche mit einem *Autorisierungsschema* zu schützen – auch der PL/SQL Prozess selbst muss abgesichert

werden – schließlich könnte jemand den nötigen HTTP-Request auch ohne Klick auf die Schaltfläche auslösen.

Cross-Site Skripting

Cross-Site-Scripting (XSS)-Angriffe basieren darauf, dass Angreifer HTML- und JavaScript-Code in die normalen Webformulare eingeben. Freitextfelder, wie für Kommentare, stehen hier besonders im Fokus. Werden diese Informationen dann vom Browser eines anderen Nutzers angezeigt, so wird das JavaScript ausgeführt - und dies ist dann das Einfallstor zum Ausspähen von sensiblen Daten. APEX bietet den Schutz vor XSS in seinen Komponenten standardmäßig an. Abbildung 3 zeigt den XSS-Schutz in einem APEX-Bericht. Die standardmäßige Einstellung "**Als Text anzeigen (Sonderzeichen escapen)**" sorgt dafür, dass HTML-Tags maskiert werden – XSS ist nicht möglich. Stellt man dagegen auf **Standardberichtsspalte** um, so werden die Tags nicht mehr maskiert. HTML-Tags werden vom Browser interpretiert und ausgeführt.

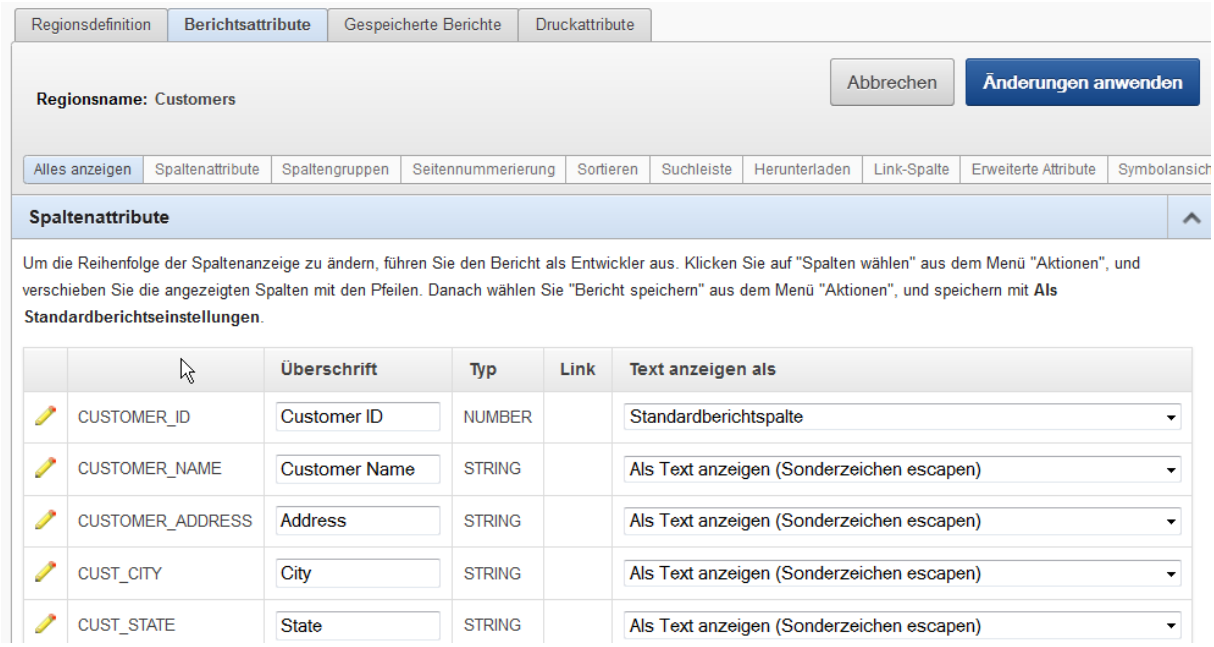

Abbildung 3: APEX-Schutz vor Cross-Site-Skripting ("Sonderzeichen Escapen")

Genau diesen Schutz schalten Entwickler jedoch häufig aus, denn das Maskieren der HTML-Tags sorgt dafür, dass auch gewünschte Formatanweisungen (`<b>`, `<i>`) nicht mehr dargestellt werden. In diesem Fall ist es wichtig, selektiv zu maskieren. Das PL/SQL-Paket **APEX_ESCAPE** kann hier mit seiner *Whitelist* für HTML-Tags weiterhelfen.

Generell sollte eine Anwendung so beschaffen sein, dass das "nicht-maskieren" von HTML-Tags nur dann stattfindet, wenn man die Inhalte als Entwicker kontrollieren kann. Unkontrollierte Inhalte von Endbenutzern sollten stets maskiert werden.

SQL Injection

Unter SQL Injection versteht man das Aushebeln von Sicherheitsregeln bzw. die Manipulation von Daten durch das Einschleusen (Injizieren) veränderter SQL-Anweisungen. Voraussetzung ist eine verwundbare Stelle in der Anwendung; typischerweise entsteht die dort, wo Nutzereingaben ungeprüft per Zeichenverkettung in eine SQL-Anweisung (Abfrage, DML oder PL/SQL) eingebaut werden.

Der folgende Code ist ein Beispiel für PL/SQL, wie es vielfach in APEX-Berichten genutzt wird: Wenn das Element P1_DEPTNO (könnte eine Auswahlliste sein) gesetzt ist, soll es als Filterkriterium dienen, wenn nicht, soll der Bericht alles anzeigen.

```
declare
  v_sql varchar2(32767);
begin
  v_sql := 'select ename, sal, deptno from emp';
  if v('P1_DEPTNO') is not null then
    v_sql := v_sql || ' where deptno = ' || v('P1_DEPTNO');
  end if;
  return v_sql;
end;
```

Tatsächlich ist ein solcher APEX-Bericht verwundbar gegenüber SQL Injection-Attacken. Denn ein Angreifer könnte dafür sorgen, dass das Formularelement nicht einfach nur eine DEPTNO, sondern ein komplettes SQL-Fragment enthält. Dadurch wird ein völlig anderes SQL ausgeführt als vom Entwickler erwartet – die Sicherheitslücke ist offensichtlich.

Die Vermeidung von SQL Injection-Schwachstellen ist gar nicht so schwierig: Zunächst sollte man versuchen, dynamisches SQL als solches zu vermeiden. In einfach gelagerten Fällen kann der PL/SQL Block problemlos in eine einfache, statische SQL-Abfrage umgeschrieben werden.

```
select ename, sal, deptno
from emp
where (deptno = :P1_DEPTNO or :P1_DEPTNO is null)
```

Es gibt aber auch Situationen, in denen dynamisches SQL nicht vermieden werden kann. So kann es sein, dass die selektierten Spalten oder die Tabellennamen dynamisch sein sollen. In diesen Fällen muss man die Parameter per Zeichenverkettung in das SQL einbauen. Damit dennoch keine Schwachstellen entstehen, sollten die Nutzereingaben vorher überprüft werden - und genau hierzu dient das PL/SQL-Paket **DBMS_ASSERT**. Jeder APEX-Entwickler sollte es kennen und nutzen.

```
declare
  v_sql varchar2(32767);
begin
  v_sql := 'select ename, sal, deptno from emp';
  if v('P1_DEPTNO') is not null then
    v_sql := v_sql ||
      ' where deptno = dbms_assert.enquote_literal('||v('P1_DEPTNO')||');
  end if;
  return v_sql;
end;
```

DBMS_ASSERT.ENQUOTE_LITERAL umfasst den übergebenen Parameter mit einfachen Anführungszeichen und gibt ihn zurück. Enthält der Parameter dagegen Zeichen, die einen SQL Injection-Angriff einleiten, so wird eine Fehlermeldung ausgelöst. Die folgenden SQL-Abfragen zeigen die Arbeitsweise auf.

```
SQL> select dbms_assert.enquote_literal(q'#Meier#') ok from dual;

OK
----------------------------------------------------------------
'Meier'

SQL> select dbms_assert.enquote_literal(q'#Meier' or 1=1#') gefahr
  2  from dual;

FEHLER in Zeile 1:
ORA-06502: PL/SQL: numerischer oder Wertefehler
ORA-06512: in "SYS.DBMS_ASSERT", Zeile 317
ORA-06512: in "SYS.DBMS_ASSERT", Zeile 381
```

Für *Objektnamen* wie Tabellen, Views oder Prozeduren steht die Funktion **ENQUOTE_NAME** bereit. DBMS_ASSERT sollte immer verwendet werden, wenn der PL/SQL-Code in irgendeiner Weise SQL zusammensetzt und dabei Benutzereingaben verwendet. Das ist insbesondere der Fall, wenn ...

- eine "PL/SQL Function Returning SQL Query" als Berichtsquelle,
- EXECUTE IMMEDIATE oder OPEN ... FOR im PL/SQL Code oder
- die APEX Substitution Syntax (&PX_ELEMENT.) im SQL oder PL/SQL Code

... verwendet wird.

Überwachung der Anwendungssicherheit mit dem APEX Dictionary

APEX ist bekanntlich durch Metadaten getrieben: Alle Eigenschaften einer APEX-Anwendung finden sich im APEX Repository wieder und können mit Hilfe der *APEX Dictionary Views* betrachtet werden. Dies kann natürlich sehr gut zur Qualitätssicherung und vor allem auch zum Auffinden potenzieller Sicherheitslücken verwendet werden.

So zeigt die folgende Abfrage alle APEX-Berichte an, bei denen wenigstens für eine Berichtsspalte der Cross-Site-Skripting Schutz deaktiviert wurde.

```
SQL> select application_id, page_id, region_name
  2  from apex_application_page_rpt_cols
  3  where display_as = 'Standard Report Column'
  4  order by 1,2;

APPLICATION_ID    PAGE_ID REGION_NAME
-------------- ---------- ---------------------------------
           100          7 Orders for this Customer
           100         20 Product Image
           100         29 Items for Order #&P29_ORDER_ID.
             :          : :
```

Möchte man wissen, welche Anwendungen ihre Benutzer selbst verwalten (und nicht mit dem LDAP-Server arbeiten), so hilft diese Abfrage weiter.

```
SQL> select application_id, AUTHENTICATION_SCHEME_TYPE
  2  from apex_applications

APPLICATION_ID AUTHENTICATION_SCHEME_TYPE
-------------- ----------------------------------------
           149 Oracle Application Server Single Sign-On
           140 Application Express
           185 No Authentication (using DAD)
             : :
```

Zu guter Letzt: APEX Plugins

APEX Plugins kommen mehr und mehr in Mode – schließlich bietet der öffentlicher Server **apex-plugin.com** eine Fülle, teilweise sehr nützlicher, Plugins an, mit denen man seine APEX-Anwendung erweitern und gleichzeitig viel Arbeit sparen kann. Allerdings bedeutet das Nutzen eines Plugins de-facto die Nutzung von Code, den man nicht selbst programmiert hat. Es ist zumindest im Bereich des Möglichen, dass ein Plugin auch Schadcode enthält ...

Die einzige Lösung ist, Plugins vor Nutzung genau zu untersuchen. Da aller Code eines APEX Plugins offen und einsehbar ist, ist dies auch ohne weiteres möglich. In größeren Umgebungen kann es sinnvoll ein, eine *Whitelist* freigegebener Plugins zu pflegen – APEX Entwickler dürfen eben nur diese nutzen. Das Durchsetzen einer solchen Regel ist wiederum mit dem APEX Dictionary möglich: Die folgende Abfrage darf nur freigegebene Plugins zurückliefern, ansonsten verstößt die Anwendung gegen die Regeln.

```
SQL> select application_id, name from apex_appl_plugins;

APPLICATION_ID NAME
-------------- --------------------------------------------
           100 COM.ORACLE.APEX.MASKED_FIELD
           101 COM.ORACLE.APEX.TIMER
           110 COM.ORACLE.APEX.SIMPLE_CHECKBOX
           106 COM.ORACLE.DE.CCZARSKI.DYNAMICQUICKPICKS
```

Weitere Informationen

[1] Deutschsprachige APEX Community
http://tinyurl.com/apexcommunity

[2] Deutschsprachige DBA Community
http://tinyurl.com/dbacommunity

[3] Blog "SQL und PL/SQL in Oracle"
http://sql-plsql-de.blogspot.com

Kontaktadresse:

Carsten Czarski
ORACLE Deutschland B.V. & Co KG
Riesstr. 25, 80992 München

Telefon: +49 (0) 89 1430 2116
E-Mail: carsten.czarski@oracle.com
Blog: http://sql-plsql-de.blogspot.com
Twitter: @cczarski