Die vielen Vorteile von Apex können die Fachabteilung eines Unternehmens dazu bewegen, eine Fachanwendung zur Personalverwaltung oder für andere sensible Firmendaten zu entwickeln. Spätestens dann kommt die Frage der Sicherheit einer Apex-Anwendung auf. Der Artikel beschreibt die Sicherheitsaspekte der verschiedenen Installationsvarianten von Produktiv-Umgebungen sowie wichtige Apex-Sicherheitskonzepte und deren Konfiguration.

Sicherheitsaspekte einer Apex-Installation

André Schulte, Freiberufler

Eine Apex-Anwendung kann grundsätzlich genauso sicher oder unsicher wie Applikationen basierend auf anderen Technologien betrieben werden. Je beliebter und verbreiteter eine Technik ist, desto größer wird der Personenkreis, der sich damit beschäftigt, Sicherheitslücken zu finden und auszunutzen. Apex ist genauso wie ein JEE- oder .NET-basiertes System „out of the box" nur unzureichend gesichert und muss an Sicherheitsanforderungen angepasst werden.

Das Sicherheitsniveau des Gesamtsystems wird durch das schwächste Glied der Kette bestimmt. Dabei müssen auch das Betriebssystem, die Datenbank und das Netzwerk betrachtet werden. Das Sicherheitskonzept muss eine Sicherheits-Maintenance vorsehen. Diese definiert Aufgaben wie regelmäßige Patches, Durchsicht von Auditlogs oder Überprüfung von Rechten. Um die Komplexität einzuschränken, geht der Artikel nur auf die spezifischen Aspekte einer Apex-Sicherheitskonfiguration für eine Produktivumgebung ein. Zunächst werden allgemeine Bedrohungen einer Apex-Anwendung aufgezeigt, die grundsätzlich für alle Web-Anwendungen relevant sind.

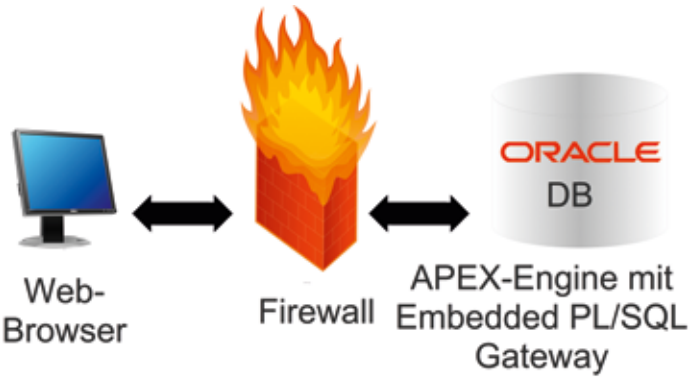

Abbildung 1: Apex-Installation mit Embedded PL/SQL Gateway

Bedrohungen einer Apex-Anwendung

Es gibt folgende Szenarien:

- *SQL-Injection*
  Der Angreifer ändert den SQL-Befehl und injiziert ein geändertes Verhalten. Dies kann zur Zerstörung der Datenbank oder zum Lesen von geschützten Daten führen. SQL-Injection entsteht durch Zeichenkettenkonkatenation zu einem SQL-Befehl, bei der der Angreifer Parameter so übergibt, dass ein in seinem Sinne geänderter SQL-Befehl ausgeführt wird. Schutz bieten Bindevariablen oder, falls dies nicht möglich ist, eine Prüfung der Variablen durch das DBMS_ASSERT-Package.
- *Cross-Site Scripting (CSS)*
  Ein Schadcode wird bei einer Webseite so eingeschleust, dass er auf der Client-Seite ausgeführt wird. Man unterscheidet persistente und nicht persistente Angriffe. Ein Beispiel für einen nicht persistenten Angriff ist eine präparierte URL mit einem Parameter, der per HTTP-GET-Aufruf übergeben wird. Der Browser zeigt den übergebenen Parameter ungeprüft auf der Webseite an und kann beispielsweise einen Script-Aufruf verursachen. Bei einem persistenten Angriff wird der Schadcode zum Beispiel als Blog-Eintrag in der Datenbank gespeichert und als Teil der Webseite angezeigt. Durch die Anzeige im Client-Browser wird der Schadcode ausgeführt und kann beispielsweise JavaScript-Aufrufe verursachen.
- *URL Manipulation (URL-Tampering)*
  Es wird eine URL manipuliert, sodass die Web-Anwendung Inhalte anzeigt, die der Nutzer eigentlich nicht sehen soll.
  In der Regel werden mit HTTP-GET übergebene Parameter abgeändert und beispielsweise eine andere Personen-ID für die Anzeige an den Server gesendet. Gegen solche Angriffe hilft die Session State Protection, die weiter unten im Artikel erläutert wird.
- *Netzwerk*
  Durch das Abhören von Netzwerk-Paketen kann man an Personen-IDs, Passwörter und andere Informationen, nach denen man gezielt sucht, kommen. Auch das Manipulieren oder wiederholte Absenden von Netzwerk-Paketen sowie eine „Man in the Middle"-Attacke müssen beachtet werden. Eine Verschlüsselung der HTTP-Verbindungen mit Secure Sockets Layer (SSL) als HTTPS schützt gegen diese Angriffe. Bei hohen Sicherheitsanforderungen kann dafür ein Zertifikat von einer Zertifizierungsstelle genutzt werden.

Installationsvarianten

Bei der Installation von Apex unterscheidet man die Installation mit dem Embedded PL/SQL Gateway (EPG) sowie die mit dem Oracle HTTP-Server (OHS) und MOD_PLSQL. Darüber hinaus kann man mit dem Apex-Listener installieren.

Beim Embedded PL/SQL Gateway läuft der Oracle-XML-DB-HTTP-Server direkt in der Oracle-Datenbank 11*g*. In dieser Zwei-Tier-Architektur greift der Browser des Nutzers direkt auf die Datenbank zu. Es ergibt sich weniger Netzwerk-Overhead als bei den anderen Varianten und man benötigt keine separate Server-Installation. Von Seiten der Sicherheit aus betrachtet ist die Lösung für Produktionen nicht zu empfehlen, weil der HTTP-Listener und Apex in derselben Datenbank ausgeführt werden. Es ist nicht möglich, sie zu trennen. Die Datenbank kann, wie in Abbildung 1 dargestellt, nicht hinter einer Firewall separiert werden und ist für Angreifer zugänglicher.

MOD_PLSQL ist ein Apache-Webserver-Erweiterungsmodul, mit dem man aus PL/SQL-Packages und Stored Procedures dynamische Webseiten erzeugen kann. Der Vorteil ist die Trennung des Apache-Webservers von der Datenbank, sodass die Datenbank hinter einer Firewall geschützt ist und die Sicherheit erhöht wird. Der Apex-Listener läuft in einem Java-Container. Er verbindet sich über JDBC mit der Datenbank. Listener und Datenbank können durch eine Firewall getrennt werden, damit die gleiche Sicherheit wie bei Nutzung der MOD_PLSQL-Lösung entsteht (siehe Abbildung 2).

Der Apex-Listener hat jedoch noch weitergehende Sicherheitsfunktionen, die ihn zur sichersten Apex-Installationsvariante machen. Es gibt Listen für Allowed- und Blocked-Procedures, die für Prozeduren, Packages und Schema-Namen verwendet werden können. Die „Database Validation Function“ ist eine Funktion in der Datenbank, die prüft, ob eine angefragte Prozedur ausgeführt werden soll. Sie kann einen Cache auf der Listener-Seite nutzen, um den Datenbank-Roundtrip zu verhindern. Bei einer Anfrage wird zuerst die Validate-Funktion geprüft, dann die Allowed-List und zum Schluss die Blocked-List. Erst danach erfolgt der eigentliche Aufruf der Prozedur. Der Apex-Listener beinhaltet außerdem Statistiken, ein Fehler-Tracking-System und eine Protokollierung. Die Protokollierung zeigt Informationen wie geänderte Cache-Einträge der Validate-Funktion oder die Ausführungszeit der Prozedur.

Database Access Descriptor (DAD)

Wenn man keinen Apex-Listener verwendet, dann kann man den Zugriff auf die Datenbank mit dem Database Access Descriptor (DAD) einschränken. Mithilfe der Funktion „wwv_flow_epg_include_mod_local“ kann man per String-Vergleich eine benutzerdefinierte Funktion autorisieren. Die Funktion gibt „true“ für eine erfolgreiche Autorisierung, ansonsten „false“ zurück. Listing 1 zeigt die Autorisierung der Funktion „myfunction“.

IP-Whitelist, HTTPS, File Upload, Session Timeout und Passwörter

Mit der Einstellung „Restrict Access by IP Address“ kann man den Zugang zur Apex-Instanz auf IP-Basis beschränken. Um den Parameter zu ändern, meldet man sich als Apex-Admin über die Web-Oberfläche an und geht auf „Manage Instance -> Security“. Dort kann unter „Restrict Access by IP Address“ eine kommaseparierte Liste, auch unter Nutzung von Wildcards, eingegeben werden.

Mit dem Setzen des „Require HTTPS“-Attributs auf „yes“ aktiviert man das HTTPS-Protokoll. Daten werden verschlüsselt über das Internet zum Browser des Clients übertragen. Die Einstellung kann im INTERNAL-Workspace unter „Manage Instance -> Security“ vorgenommen werden. Durch Konfigurieren des „Allow Public File Upload”-Attributs auf „no“ kann man verhindern, dass nicht authentifizierte Nutzer in Datei-Upload-Steuerelementen Dateien hochladen können. Damit nicht geschlossene Browser beziehungsweise Nutzer, die sich nicht aus der Applikation ausloggen, ein geringeres Sicherheitsrisiko darstellen, ist es sinnvoll, den Parameter „Session Timeout“ festzulegen. Man kann dort angeben, nach wie vielen Sekunden eine ungenutzte Session abläuft und nach wie vielen Sekunden eine Session

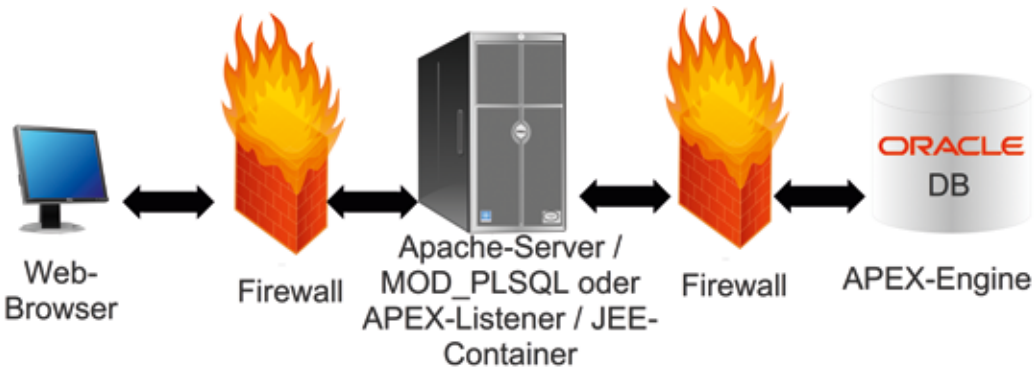

Abbildung 2: Apex-Installation mit MOD_PLSQL und Apex-Listener

```
CREATE OR REPLACE function wwv_flow_epg_include_mod_local( procedure_name in varchar2)
return boolean
is
begin
    if upper(procedure_name) in ('Apex_040100.MYFUNCTION','MYFUNCTION') then
        return TRUE;
    else
        return FALSE;
    end if;
end wwv_flow_epg_include_mod_local;
/
```

Listing 1

```
create table testapp.geheim as select * from (
select object_name, object_type ,
ROW_NUMBER() OVER(PARTITION BY object_type ORDER BY object_type, object_name) myindex
from dba_objects where object_type in (,TABLE', ,FUNCTION', ,PROCEDURE') order by object_type, object_
name) where myindex <= 5
/
```

Listing 2

beendet werden soll. Der Entwickler einer Apex-Applikation kann für die Nutzer eine Passwort-Policy festlegen. Dies ist sinnvoll, damit Nutzer nicht zu einfache Passwörter verwenden. Beispiele für Regeln sind die Verwendung mindestens eines Großbuchstaben oder mindestens einer Zahl und das Vorschreiben einer Mindestlänge des Passworts von sechs Buchstaben. Die Sicherheit kann weiterhin erhöht werden, indem das Wiederverwenden von Passwörtern unterbunden wird. Diese Funktionalität wird unter „sqlplus" als Apex-Schema-Owner (Apex_040100) beispielsweise für sechzig Tage aktiviert: „Apex_INSTANCE_ADMIN.SET_PARAMETER(‚PASSWORD_HISTORY_DAYS',60);".

Session State Protection

Mit der Verwendung von HTTPS können Pakete unterwegs nicht mehr manipuliert werden. Der Nutzer kann aber vor dem Absenden in der Browserzeile einen Parameter, etwa für eine Personen-ID oder eine Mandanten-ID, manipulieren. Apex speichert Session-Parameter serverseitig und persistent in der Datenbank. Aus diesem Grund sollte man sensitive Daten in der Session in verschlüsselter Form speichern. Im Application Builder kann, wie in Abbildung 3 dargestellt, unter „Shared-Components -> Session State Protection" über einen Wizard der Schutz auf verschiedenen Ebenen aktiviert werden.

Man unterschiedet zwischen „Application Item Protection" zum Schutz individueller Items und „Page Access Protection", um die ganze Seite zu schützen. Bei „Page Protection" wählt man im Wizard für die Seite „Arguments Must Have Checksum". Weitere Einstellungen sind „No Arguments Allowed", „No URL Access" oder „Unrestricted". Eine mit „Arguments Must Have Checksum" geschützte Seite hat beim Aufruf in der URL als letzten Paramter „cs=<checksum>". Der Nutzer kann die anderen Parameter in der URL sehen, aber eine Änderung wird an der falschen Checksumme erkannt und führt zu einem Fehler. Auf diese Weise wird das sogenannte „URL-Tampering" verhindert.

Session State Protection und VPD

Eine Session State Protection schützt vor Manipulationen von Seiten des Clients. Ein SQL-Befehl, der eine Sicherheitslücke bei der Session State Protection findet, wird auf der Datenbank ungehindert ausgeführt. Ein weiteres Problem ist, dass eine komplexe Anwendung aufgrund eines Entwicklungsfehlers mit einem der Reports ungewollt zu viele Daten anzeigt. Die Qualitätssicherung hat den Report aber nie mit den Parametern des Anwenders, der die Lücke ausnutzt, aufgerufen. Um bei einer Lücke in der Session State Protection oder bei einem Anwendungsfehler dennoch sensible

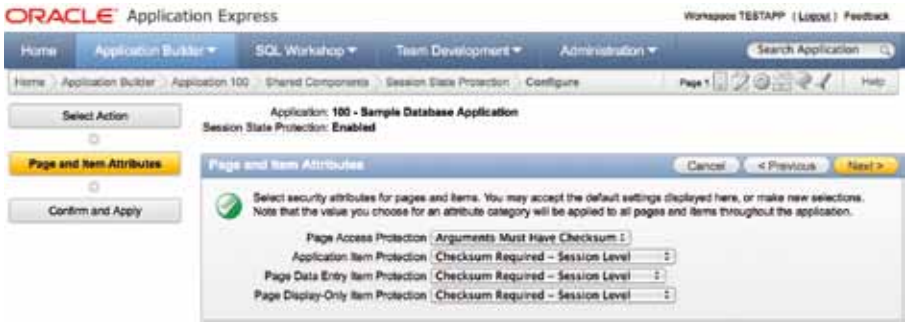

Abbildung 3: Session State Protection gegen URL-Manipulation

```
Grant EXECUTE ON DBMS_RLS TO TESTAPP; (als Nutzer sys)
Grant CREATE ANY CONTEXT TO TESTAPP; (als Nutzer sys)

create or replace context apex_context using vpd_context
/
create or replace package vpd_context is
  procedure logonUser(userid varchar2);
end vpd_context;
/

create or replace package body vpd_context is
  procedure logonUser(userid varchar2) is
  begin
    dbms_session.set_context(‚apex_context‘,‘userid‘, userid);
  end logonUser;
end vpd_context;
/

create or replace function pol_geheim(obj_owner in varchar2, obj_name in varchar2) return varchar2
as
  predicate varchar2(200) := ‚1=2‘;
begin
   if SYS_CONTEXT(‚apex_context‘,‘userid‘)= ‚ADMIN‘ then
     predicate := ‚‘;
   elsif SYS_CONTEXT(‚apex_context‘,‘userid‘)= ‚TABLE_ADMIN‘ then
     predicate := ‚OBJECT_TYPE=‘‘TABLE‘‘‘;
   elsif SYS_CONTEXT(‚apex_context‘,‘userid‘)= ‚PROCEDURE_ADMIN‘ then
     predicate := ‚OBJECT_TYPE=‘‘PROCEDURE‘‘‘;
   end if;
   return predicate;
end;
/
 BEGIN
DBMS_RLS.ADD_POLICY(
object_schema => ‚testapp2‘,
object_name => ‚geheim‘,
policy_name => ‚secure_geheim‘,
policy_function => ‚pol_geheim‘,
statement_types => ‘SELECT’);
END;
/
```

Listing 3

Daten zu schützen, kann zusätzlich zur Session State Protection die Virtual Private Database (VPD) genutzt werden. VPD ist nur mit der Enterprise Edition der Datenbank möglich und erzwingt den Schutz der Daten direkt bei der Ausführung des SQL-Befehls in der Datenbank.

Ein Beispiel zeigt, wie VPD mit Apex genutzt werden kann. Der grundsätzliche Gebrauch ist in der MyOracleSupport Note (ID 1352641.1) „Master Note For Oracle Virtual Private Database“ erläutert. Im Beispiel werden ein neuer Workspace namens „TESTAPP“ und in dem TESTAPP-Schema die Tabelle „geheim“ angelegt, die fünf Tabellen-, Funktions- und Prozedurnamen der Datenbank aus dba_objects enthalten soll. Listing 2 zeigt den SQL-Befehl, der mit „sqlplus“ als „sys“ ausgeführt wird.

In Apex wird in der neuen Applikation ein einfacher Report für die Tabelle angelegt, bei dem die gesamte Tabelle mit folgendem SQL-Befehl selektiert wird: „select * from geheim“. In Apex sind die drei Nutzer „admin“, „table_admin“ und „procedure_admin“ für die Applikation angelegt. Der Nutzer „admin“ soll später alle Daten sehen können, „table_admin“ beziehungsweise „procedure_admin“ jedoch nur die Tabellen und Prozeduren aus der Tabelle „geheim“. Um VPD nutzen zu können, werden im TESTAPP-Schema ein Kontext angelegt, die Logon-Prozedur „vpd_context.logonUser“ für das Setzten des Kontext erstellt sowie die Policy-Funktion „pol_geheim“ für die Einhaltung der genannten Regeln implementiert (siehe Listing 3).

Die Funktion „ADD_POLICY“ aktiviert die Regel. Beim Einloggen als Nutzer „admin“ bekommt man aber noch keine Daten angezeigt, da der Kontext von Apex noch nicht gesetzt wird. Mit Apex kann man unter „Application Builder -> Edit Application Properties -> Security“ im Feld „Initialization PL/SQL Code“ die Funktion für das Setzten des Kontext konfigurieren: „vpd_context.logonUser(v(‘APP_USER’))“. Apex ersetzt „v(‘APP_USER’)“ durch den tatsächlich einzuloggenden

Object Name	Object Type	Myindex
ADD_JOB_HISTORY	PROCEDURE	1
APEX	PROCEDURE	2
APEX	PROCEDURE	3
APEX_ADMIN	PROCEDURE	4
APEX_ADMIN	PROCEDURE	5

1 - 5

Object Name	Object Type	Myindex
ACCESS$	TABLE	1
ACTION_TABLE	TABLE	2
ALERT_QT	TABLE	3
ANFRAGE	TABLE	4
ANFRAGEDATEN	TABLE	5

1 - 5

Abbildung 4: Reportdaten, links „procedure_admin“, rechts „table_admin“

Nutzer. Wenn man sich als Nutzer „table_admin“ oder „procedure_admin“ anmeldet und den Report aufruft, sieht man unterschiedliche Daten (siehe Abbildung 4). Die Regel wird allerdings nicht durch Apex, sondern durch die Datenbank bei der Ausführung des SQL-Befehls erzwungen.

Fazit

Die allgemeinen Bedrohungen einer Apex-Anwendung sind vielfältig. Genauso kreativ wie der Angreifer sollte daher auch das Sicherheitskonzept zur Abwehr von Angriffen sein. Ein ganzheitliches Konzept muss alle System-Komponenten berücksichtigen. Bereits bei der Entwicklung der Anwendung müssen Bedrohungen wie die SQL-Injection berücksichtigt und Bindevariablen genutzt werden. Eine Basis-Apex-Härtung umfasst die richtige Installationsvariante und das Setzen der Apex-Sicherheitsparameter. Der Aufwand für diese Maßnahmen ist überschaubar und der Sicherheitsgewinn groß. Weitere Maßnahmen zur Erhöhung der Sicherheit sind mit höherem Aufwand verbunden. Der Einsatz von Virtual Private Database erfordert beispielsweise eine Enterprise-Edition-Lizenz. Diese Lösung ist sinnvoll als zusätzlicher Schutz zur Verhinderung der Einsicht in sensitive Daten unterhalb der Anwender (nicht dem Datenbank-Administrator).

André Schulte
contact@andschulte.de