12c Real Application Security
Konzepte, Techniken, Nutzen

Dr. Günter Unbescheid

Database Consult GmbH

Jachenau

Schlüsselworte

Datenbank, Sicherheit, Application Security, Application User, Application Roles

Einleitung

Die Zahl der Web-/Multi-Tier-Anwendungen mit direktem Zugriff auf Oracle-basierte Daten wächst beständig und mit ihnen die Zahl der „externen" End-Benutzer, für die keine maßgeschneiderten Datenbank-Privilegien konfiguriert werden können. Schon seit der Version 8i gibt es Features wie beispielsweise Virtual Private Database, Secure Application Roles (9i) und Client-IDs, die dabei helfen sollten, Security-Anforderungen aus den Applikationen teilweise in die Datenbank zu verlagern und damit zu zentralisieren.

Unter Oracle 12c wurden diese Ansätze unter dem Terminus „Real Application Security" (RAS) weiter ausgebaut und verfeinert. Deklarative Zugriffsregeln und ACLs erlauben es nun Sicherheitsanforderungen noch stärker als bisher von den Applikationen in die Datenbanken zu verlagern.

Der Vortrag zeigt die Konzepte und Techniken die RAS zur Verfügung stellt, um sichere Applikationen zu realisieren.

Die Anforderungen

Enterprise Applikationen, die über Java-basierte Application Server Zugang zu Daten benötigen, die in Datenbanken gespeichert sind, nutzen häufig Connection Pools für ihre Datenbank Zugriffe. Connection Pools dienen in diesem Zusammenhang als Cache von Datenbank-Verbindungen, die in einer Mindest- und Maximalzahl vom Application Server verwaltet und bereitgestellt werden. Applikationen können auf diese Weise mit reduziertem Ressourcenverbrauch eine bereits gestartete Verbindung für ihre Belange kurzfristig reservieren und für ihre Zugriffe nutzen. Für den Zugriff auf Oracle Datenbanken stehen dann unterschiedliche Pool-Implementierungen zur Verfügung, die hier nur kurz erwähnt werden sollen:

- *Implicit Connection Cache*
  Über virtuelle Connections können – neben dem konfigurierten Standard-Benutzer – spezifische Datenbank Benutzer für die Verbindung angefordert werden und nutzen dann die bereitgestellten physischen Verbindungen des Caches. Dieses Feature wird beginnend mit der Version 12.1 nicht mehr weiter entwickelt (*deprecated*).

- *OCI Connection Pooling* – steht im JDBC OCI Kontext zur Verfügung und ermöglicht das Multiplexing unterschiedlicher Sessions über eine reduzierte Anzahl physikalischer Verbindungen.
- *Database Resident Connection Pooling* – baut auf der Seite des Datenbank-Server einen Pool auf der von unterschiedlichen Client-Pools gemeinsam genutzt wird und dabei hilft, die resourcen für die Mindest-Zahl von Pool Verbindungen zu reduzieren.

Wenn wir die im Security-Umfeld üblichen Anforderungen der exakten Privilegierung (*need-to-know* und *least privilege* Prinzipien) auch im Kontext von Web-Applikationen, die auf Datenbanken zugreifen, durchgängig und effizient umsetzen wollen, müssen wir folgendes berücksichtigen

- Die Benutzer der Web-Applikationen (Enduser) sind in der Regel keine klassischen Datenbank-Benutzer (Schema User) mit fest vergebenen Privilegien.
- Sicherheitsanforderungen der Enduser müssen daher dynamisch beim Zugriff über Connection Pools aktivierbar sein, will man nicht einen größtmöglichen Nenner für alle Zugriffe schaffen.
- Trotz der dynamischen Aktivierung müssen Zugriffe zum Zwecke des Auditing und Logging nachvollziehbar und individuellen Personen zuzuordnen sein.

Diese Anforderungen machen eine Erweiterung des klassischen Schema-User und Grant-Modells nötig.

Ein kurzer Rückblick

Real Application Security (RAS) erweitert und ergänzt konsequent einige Features, die in älteren Versionen bereits zur Zugriffssteuerung in Web-Applikationen verfügbar waren und die im Folgenden kurz zusammengefasst werden sollen:

- *Client-Identifier (CI)*
  CI sind beliebig und explizit zu setzende Zeichenketten die eine bestehende Session zusätzlich identifizieren helfen. Einmal gesetzt stehen sie in diversen Kontexten wie v$session und den Auditing-Views zur Verfügung. Sie lassen sich auch in bestehenden Sessions setzen und anpassen.
- *Secure Application Roles (SAR)*
  SAR werden so angelegt, dass eine namentlich genannte PL/SQL-Programmeinheit für ihre Aktivierung verantwortlich zeichnet. Innerhalb dieser Prozedur lassen sich dann sinnvolle Plausibilitäten einbauen, bevor die genannte Rolle aktiviert wird. SAR-Rollen können niemals direkt gegrantet werden.
- *Virtual Private Database (VPD)*
  auch bekannt unter dem Terminus Row-Level Security regelt den Zugriff auf Tabellen über Policies, die implizite Where-Bedingungen implementieren, die beim Zugriff eingeschleust werden. Die Policies können auch mit Spalten verknüpft werden und so eine zusätzliche Where-Bedingung generieren, wenn auf sensible Spalten zugegriffen wird.

RAS: Komponenten und Architektur

Real Application Security erweitert die oben genannten Features und stellt so ein Framework zur Verfügung mit dem sich die sogenannte *end-to-end* Security in Multitier-Applikationen aber auch 2-Tier Applikationen umsetzten lässt. Die Datenbank wird so zum zentralen Security-Server für alle in ihrem Kontext gespeicherten und im Rahmen des Frameworks geschützten Daten.

Das Framework ist derzeit mit folgenden Komponenten umgesetzt:

- 32 Metadaten-Tabellen (`XS$%`) im Schema `SYS`.
- 30 `DBA_XS` und 13 `USER_XS_` Views
- 4 vordefinierten `XS_` Datenbank Rollen
- 18 `XS_` und 17 DBMS_XS PL/SQL-Packages
- Java Packages `oracle.security.xs` und `oracle.security.xs.ee.session` zur Einbindung von Java/JDBC-basierten Programmen.
- Diverse vordefinierte Application User und Application Roles
- Die Apex-Anwendung RASADMIN mit einer grafischen Oberfläche zur Verwaltung des Frameworks in der Datenbank.
- Skripte zum Aufbau einer kleinen Demo-Umgebung, inklusive einer Java-Klasse.

Die Objekte von RAS sind Bestandteil der Enterprise Edition ohne zusätzliche Optionen und Packs. Sie sind mit der Installation jeder Datenbank-Instanz verfügbar – mit Ausnahme von RASADMIN, das in einer Apex 5.0 Umgebung explizit heruntergeladen und installiert werden muss.

Alle wesentlichen Konfigurationsschritte zur Absicherung von Applikationen werden im Kontext der Datenbank über die aufgelisteten APIs erledigt. Erst danach werden über Java-APIs die JDBC-Calls der Applikationen angepasst. Die Grundkonfiguration läuft zusammengefasst beispielsweise wie folgt ab (Reihenfolge):

- Für den Zugriff auf die betroffenen Datenbank-Objekte, z.B. Tabellen werden Datenbank-Rollen wie gewohnt angelegt und mit Privilegien ausgestattet.
- Über die RAS-APIs werden nun Application Roles angelegt. Die vorstehend angelegte(n) Datenbank-Rolle(n) werden über `grant` Befehle diesen Application Roles zugeordnet.
- Die zum implementierenden Enduser (*principals*) können nun über die RAS-APIs als Application User angelegt und die zuvor angelegten Application Roles ihnen zugeteilt werden. Den Application Usern können Passwörter zugewiesen werden.
- Über *Security Klassen* werden RAS Privilegien festgelegt
- *Access Control Lists* (ACL) ordnen mit ihren Einträgen (*access control entries*) RAS-Privilegien den vorstehend definierten Application Roles zu.
- *Security Policies* verbinden dann die ACLs mit Zeilenfiltern (*realm constraints*) und Spaltenfiltern (*column constraints*)
- Schließlich werden die Policies den Datenbank-Objekten, also beispielsweise den betroffenen Tabellen, beigestellt.

Das so entstandene komplexe Objektgeflecht kann nun – und sollte auf jeden Fall (!) – über ein eigenes API validiert werden. Application User können sich dann entweder direkt mit der Datenbank unter Angabe ihres Passwortes verbinden oder werden per *attach* Operation an eine bereits bestehende Session, z.B. des Session Pools gekoppelt. Die Daten können dann gemäß den Policies gelesen und geschrieben werden.

Abbildung 1 zeigt die Verbindungen der oben genannten Objekte:

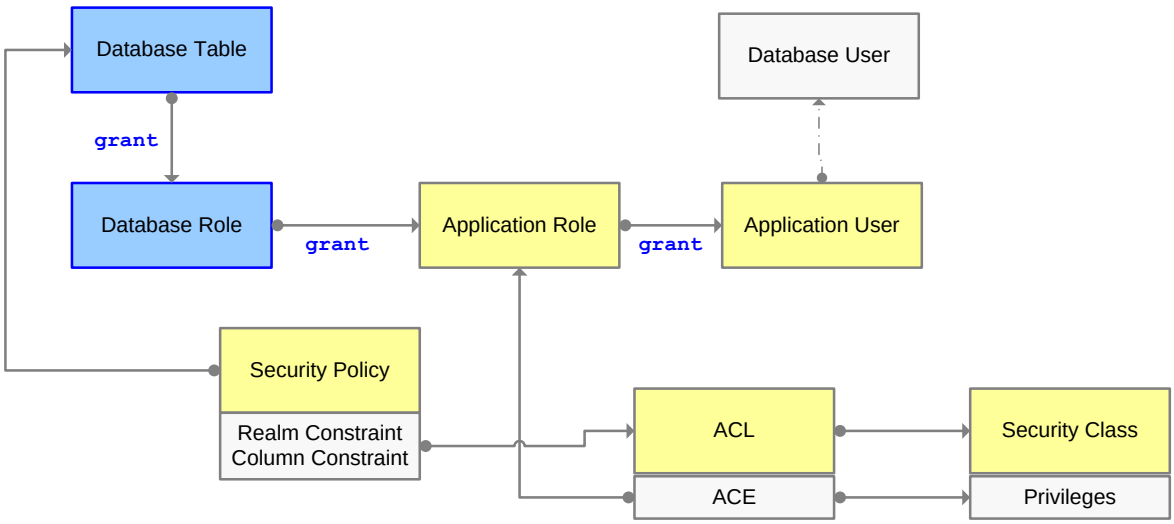

Abbildung 1: RAS Objekt-Beziehungen

Die nachfolgende Abbildung 2 skizziert die Gesamt-Architektur

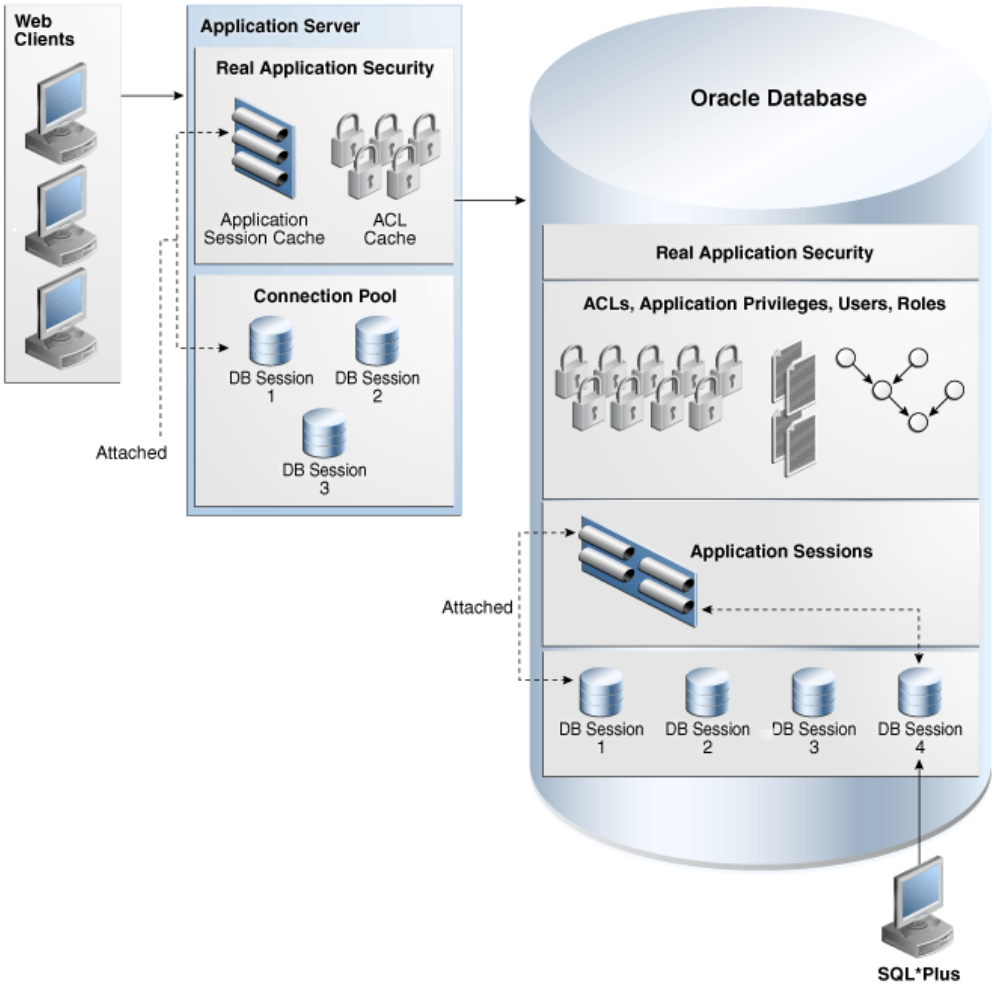

Abbildung 2: RAS Gesamt-Architektur (Quelle Oracle)

Einige der Hauptkomponenten sollen in den folgenden Abschnitten detaillierter beschrieben werden:

- Application User (AU)

  Sind „Schema-los“ können aber sehr wohl im Kontext eines Schemas angelegt werden, was dem Setzen von `CURRENT_SCHEMA` entspricht und keinesfalls Privilegien zuteilt. Im Falle einer Anmeldung müssen die User ein Passwort haben, das auch über Regeln (`create profile`) abgesichert werden kann. Sie können sich damit direkt an der betreffenden Datenbank anmelden.

  Darüber hinaus lassen sich für AU Start- und Endzeiten bestimmen, sowie Proxy User festlegen, die einen definierten Target User über eine Switch .

  AU dürfen nicht namensgleich mit Datenbank Usern sein.

- Application Roles (AR)

  Reguläre AR können nur an AU oder andere AR über das API „gegrantet“ werden nicht aber an Datenbank User oder andere Datenbank Rollen. Wohl aber können – umgekehrt – Datenbank Rollen per `grant` an AR vergeben werden.

  Neben den regulären existieren auch dynamische AR, die nicht vergeben, sondern nur über Application Code aktiviert werden können. Hierbei lassen sich die Aktivierungsdauer in Minuten sowie das Umfeld (scope) einstellen, je nachdem ob die Rolle nach einem *detach* und erneuten *attach* der Session noch aktiv sein soll oder nicht.

  Für die Zuteilung von AR existieren, wie auch für AU, effektive Start- und Enddaten außerhalb derer die User und Rollen nicht ansprechbar sind.

- Application Sessions (AS)

  AS verbrauchen weit weniger Ressourcen als Standard Datenbank-Sessions, sie halten keine eigenen Cursor und Transaktionskontexte und können dem entsprechend schnell aktiviert und deaktiviert werden. Die Aktivierung erfolgt über eine entsprechende PL/SQL-Api oder Java-Klasse die aus dem Kontext einer – niedrig privilegierten Verbindung heraus aufgerufen wird.

- Access Control Lists (ACL) und Application Privileges (AP)

  Im Gegensatz zu den fest definierten Systemprivilegien der Datenbank, wie z.B. `create session`, sind Application Privilegien namentlich frei definierbar und müssen, um wirksam zu werden, über ACLs und Security Policies mit Objekten der Datenbank verbunden werden. Aggregierte Privilegien erlauben darüber hinaus die Bündelung diverser Application Privilegien.

  Aggregierte Application Privilegien dürfen nicht mit Application Rollen verwechselt werden. Letztere binden – über ACLs – Privilegien an Application User und sorgen damit für deren Aktivierung.

- Security Class (SC)

  Security Klassen bündeln Application Privilegien und erlauben es ACLs, sich auf sie zu beziehen und sich der in ihrem Kontext definierten Privilegien zu bedienen. SC können kaskadierend angelegt werden, d.h. ihre Privilegien an andere SC vererben.

- Security Policy (SP)

  SPs binden Application Privileges an Row- oder Column Constraints, welche die Ergebnismengen in Bezug auf Zielobjekte steuern, z.B. durch die Generierung von WHERE-Klauseln.

Die folgende Abbildung zeigt die Abhängigkeiten von Security Klassen, Security Policies und ACLs/ACEs.

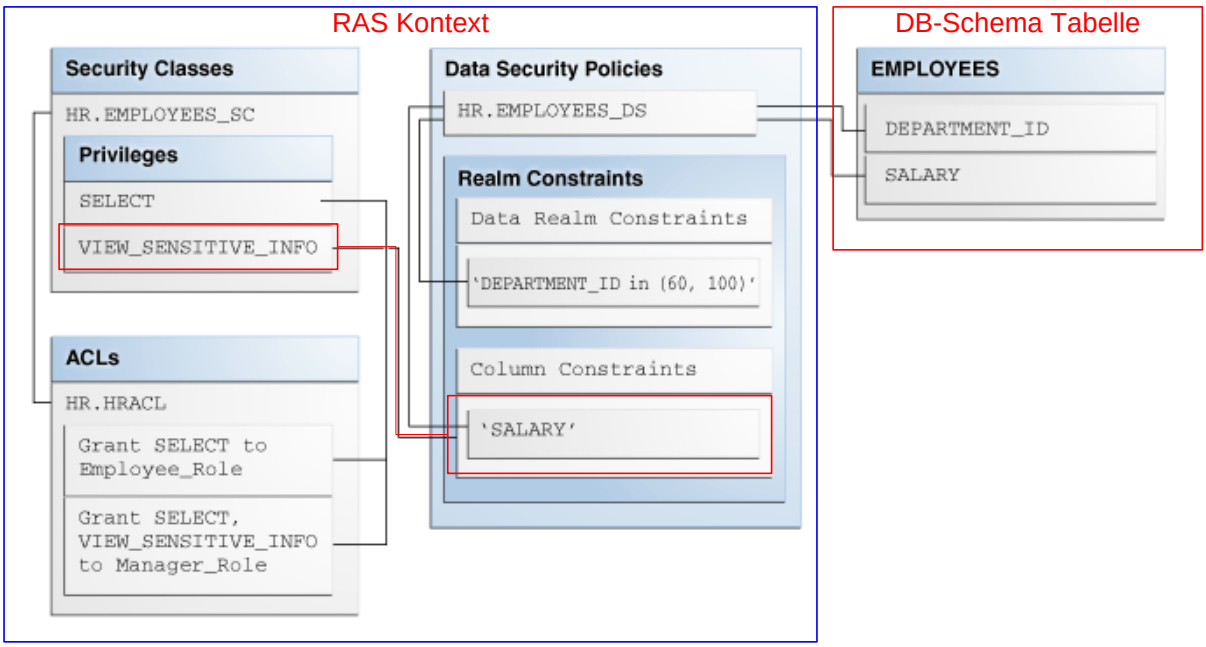

Abbildung 3: Security Classes, -Policies, Tabellen und ACLs (Quelle Oracle)

Implementierung

Zahlreiche Implementierungsbeispiele mit der zugehörigen Syntax finden sich in den Folien dieses Vortrags.

Auditing

RAS Events können über Policies des Unified Audit Trails in 12c aufgezeichnet werden. Hierzu existieren zahlreiche RAS spezifische Audit Optionen. Eigene DBA_XS_AUDIT_TRAIL Views stehen für die Auswertung zur Verfügung.

Lizensierung

RAS erfordert die Enterprise Edition ohne zusätzliche Optionen.

Zusammenfassung

- RAS bietet umfangreiche Möglichkeiten das Thema Application Security in Zieldatenbanken zu implementieren und geht weit über die Features älterer Releases hinaus
- Das Design der Regeln und deren Implementierung ist äußerst komplex und daher fehleranfällig. Die verfügbaren Validierungen bieten hier eine sinnvollen Unterstützung.
- Die Möglichkeiten von RAS sind prinzipiell gut dokumentiert. Einige Features, wie beispielsweise die Einbindung externer Principals, kommen in der Dokumentation zu kurz.

- Wünschenswert wären auch Funktionen, welche die komplexen Regeln ex- und importieren könnten, um sie in unterschiedlichen Systemen nutzen zu können.

Kontaktadresse:

Dr. Günter Unbescheid

Database Consult GmbH

Laich 9 1/9

D-83676 Jachenau

Telefon:	+49 (0) 8043 1010
E-Mail	g.unbescheid@database-consult.de
Internet:	www.database-consult.de