Authentifizierung und Autorisierung in Oracle-RDBMS basierten Systemen

Dr. Günter Unbescheid

Database Consult GmbH

Jachenau

Schlüsselworte

Datenbank, Sicherheit, Authentifizierung, Autorisierung

Einleitung

Die effiziente Umsetzung von Security-Anforderungen steht in vielen Projekten mittlerweile sehr weit oben auf den Agenden. Mit jedem Oracle-Release wurden vor diesem Hintergrund die technischen Möglichkeiten weiter ausgebaut, Benutzer zu authentifizieren, um auf dieser Basis ihre Zugriffe und Aktionen im Kontext der Datenbank zu steuern und zu autorisieren. Mittlerweile konkurrieren zahlreiche konzeptionelle Alternativen, angefangen von DB-internen Techniken der Authentifizierung über Password-lose Verfahren bis hin zu externen Verfahren wie die Einbindung von Zertifikaten und Kerberos-Tickets u.v.m. .

Für ein erfolgreiches und verantwortungsbewusstes Security-Design von Datenbank-gestützten Anwendungen aber auch von administrativen Prozessen ist es nach wie vor unerlässlich, die verfügbaren konzeptionellen Alternativen zu bewerten und in die Planung mit einzubeziehen, um zu einer optimalen Lösung zu finden.

Der Vortrag konzentriert sich vor dem Hintergrund des aktuellen Release 12.2 der Datenbank – neben technischen Hinweisen – vor allem auf die strategischen und administrativen Auswirkungen der verfügbaren Design-Alternativen und ist damit für Entwickler und DBAs, die sich mit Security auseinandersetzen, gleichermaßen relevant.

Motivation

In Zeiten der aufkommenden „autonomen“ Datenbanken und zunehmender IT-technischer „Bewölkung“ (*Cloud*) gibt es immer noch einen erheblichen Bedarf, das Thema Datenbank-Sicherheit gründlich zu planen und effizient umzusetzen. Im Zentrum eines jeden Sicherheitskonzeptes stehen die Begriffe und Themen *Authentifizierung* – also der Nachweis oder die eindeutige Verifizierung einer persönlichen oder gruppenspezifischen Identität und über diese die Zuordnung eines gültigen (Datenbank-)Users – und *Autorisierung* – also die Festlegung von Zugriffs- oder Ausführungsrechten auf Datenbank-Objekten. Es versteht sich, dass eine funktionsgerechte Festlegung von Rechten (*need to know principle*) nur auf der Basis einer präzisen, d.h. nach Möglichkeit personenspezifischen Authentifizierung und damit Benutzer-Zuordnung erfolgen kann.

Während die Authentifizierung mit internen Mitteln der Datenbank oder über externe Methoden – wie z.B. Kerberos, RADIUS oder Smartcards – erfolgen kann, wird die Autorisierung in der Regel intern über System- und Objektprivilegien oder Datenbank-Rollen durchgeführt. Gleichwohl können Datenbank-Rollen auch über externe Dienste aktiviert werden.

Authentifizierung

Datenbank-Benutzer und interne Authentifizierung

Im einfachsten Fall werden Datenbank-Benutzer (DBU) zusammen mit einem Passwort in der Datenbank angelegt. DBU lassen sich anlegen als

- *Local User* – in einer Multitenant-Umgebung, d.h. Container Datenbank (CDB), sind dies Benutzer, die in einer Pluggable Database (PDB) registriert sind und dort Zugriffsrechte haben. Local User gleichen Namens lassen sich durchaus in mehreren PDB einer CDB anlegen, weil jede PDB einen eigenen Namespace hat. Sie sind zwar innerhalb der CDB alle unter einer gemeinsamen **`USER_ID`** speichert, haben aber ansonsten eigenständige Metadaten, wie z.B. einen *creation timestamp* und eine *container_id*.

  Auf diese Weise lassen sich namensgleiche, jedoch völlig unabhängige Benutzer, die auch unterschiedliche Passwörter und Zugriffrechte haben können, realisieren.
- *Common User* – werden beginnend mit der Version 12.2 in *CDB common user* und *application common user* unterteilt.

  CDB common user sind demnach alle User, die nicht nur im Root Container, sondern auch in allen gegenwärtigen und zukünftigen PDBs inklusive der Application Roots bekannt sind.

  Application common user hingegen werden in den – seit der Version 12.2 verfügbaren – Application Root Containern angelegt und gelten nur im Kontext des betreffenden Application Root Containers und seiner abhängigen Application PDBs.

  Namen von CDB Common User müssen standardmäßig mit „**`C##`**“ oder „**`c##`**“ beginnen. Das Präfix kann allerdings über den init-Parameter **`common_user_prefix`** angepasst werden. Die Groß- und Kleinschreibung ist hierbei nicht relevant. Umgekehrt dürfen Namen von *local user* niemals mit dem spezifizierten Präfix und auch nicht mit dem oben genannten Standard-Präfix beginnen.

  Das Präfix gilt dagegen nicht für Application Common User, die über den entsprechenden Application Root Container angelegt werden. Wichtig ist auch, dass nach dem Anlegen eines solchen Benutzers die betreffende Applikation synchronisiert wird (**`alter pluggable database application <name> sync`**).
- *Non-CDB User* – sind im Grunde *local user* in einer althergebrachten Singletenant- oder Non-CDB-Datenbank.

Namen von Benutzern müssen in ihrem Gültigkeitsbereich stets eindeutig sein und teilen sich mit Rollen den gleichen Namensraum, d.h. können niemals den gleichen Namen wie Datenbank-Rollen haben.

Passwort Algorithmen

Für die interne Authentifizierung werden Passwörter in *one-way* Hash-Form im Data Dictionary gespeichert. Die Speicherung wurde im Laufe der letzten Releases kontinuierlich verbessert:

- Beginnend mit Version 6 wurde zunächst nur DES unterstützt. Ab der Version 11G sind die Passwörter ohne Unterscheidung von Groß- und Kleinschreibung in 3DES gespeichert. Passwort und Benutzername werden hier vor dem Hashing zusammengezogen.
- Ab 11gR1 sind die Passwörter mit Groß- und Kleinschreibungsunterscheidung in SHA1 „gehasht". Zusätzlich und vor der Hash-Operation wird – statt des Benutzernamens – ein SALT-Wert vorangestellt, der mit jedem Wechsel des Passwords neu generiert und zusammen mit dem Hashwert in **`user$.spare4`** gespeichert wird.
- In 12.1.0.1 sind insgesamt 3 Algorithmen möglich: DES, SHA1 und HTTP Digest. Letzterer kommt zur Anwendung, wenn sich XDB User (z.B. im Kontext der EM Express Edition) mit der Datenbank verbinden. Es ist bekanntermaßen ein sehr schwacher Algorithmus.
- Ab Version 12.1.0.2 kommt noch SHA2 dazu, der in Kombination mit PBKDF2 angewendet wird. Ab dieser Version werden die Password Hashes, die im Root Container in **`user$.share4`** gespeichert sind, auch nicht mehr in den PDBs angezeigt. Zusätzlich ist standardmäßig der HTTP Digest Algorithmus ausgeschaltet (siehe auch die **`HTTP DIGEST`** Klausel im **`CREATE`** und **`ALTER USER`** Kommando und die View **`DBA_DIGEST_VERIFIERS`**).

Welche Algorithmen für die Hash-Generierung der Passwörter genutzt werden, hängt von dem Parameter **`SQLNET.ALLOWED_LOGON_VERSION_SERVER`** ab. Die Vorbelegung in Version 12.2 ist **`12`**, was bedeutet, dass 11G und 12C Hashes verfügbar sind. Der Wert **`12a`** hingegen erzeugt nur 12C spezifische SHA2 Hashes.

Die Komplexität von Datenbank Passwörtern sollte auf jeden Fall über Password Profiles geregelt werden, die als Schaltstelle für die Komplexität und Lebenszeit dienen. Zu beachten ist, dass die Regeln über **`ALTER USER IDENTIFIED BY VALUES`** übergangen werden können, indem Hash Werte direkt in das Data Dictionary eingeschleust werden.

Ähnlich den Benutzern können auch Password Profiles als *local* oder *common profile* angelegt werden. Auch hier ist das Präfix des Namens und die Klausel **`container=all`** entscheidend für den Typ. Die Zuweisungsmöglichkeiten sind im Prinzip beliebig: ein *common user* kann in einer PDB ein *local profile* bekommen oder umgekehrt ein *local user* ein *common profile*.

Die generellen Vor- und Nachteile von Datenbank Usern mit interner Authentifizierung können wie folgt zusammengefasst werden:

+ Standard-Lösung, die mit „Bordmitteln" umzusetzen ist
+ Im aktuellen Release verbesserte und sicherere Hash-Algorithmen und Komplexitätsregeln
– Simple, nachvollziehbare Passwörter und/oder keine Komplexitätsregeln erleichtern das Hacken.
– Kein Auditing und keine Log-Analysen lassen Angriffe unbemerkt.
– Datenbank-zentrisch, d.h. es entsteht viel Administrationsaufwand bei einer großen Menge von Datenbanken und aktuellen Anpassungen von Passwörtern und Usern (**`drop/create`**).

Administrative User

Administrative User können über Betriebssystem-Gruppen oder Passwörter authentifiziert werden. Zusätzlich lassen sich auch zentralisierte Verfahren, wie z.B. Verzeichnisdienste, einsetzen.

Die Authentifizierung über Betriebssystem Gruppen ist für lokale Connects sinnvoll. Bei Multitenant Datenbanken kann damit nur der Zugang zum Root Container geregelt werden. Administratoren der PDBs werden nur über intern gespeicherte Passwörter authentifiziert.

Für Administrative User, die über Passwörter authentifiziert werden und privilegierte Operationen im Kontext der Datenbank auch von remote ausführen sollen, wie z.B. das Starten und Stoppen, oder Backup- und Recovery Operationen, müssen Passwörter auch bei geschlossener Datenbank verifizierbar sein. Aus diesem Grund werden sie nicht nur im Data Dictionary sondern auch in externen Password Files (Parameter **`remote_login_passwordfile`**) gespeichert. Die Datei wird über das Werkzeug **`orapwd`** im Verzeichnis **`$ORACLE_HOME/dbs`** erzeugt. Im aktuelle Release 12.2 wurde die Funktionalität wie folgt erweitert:

- Speicherung der Datei nicht nur im Filesystem, sondern auch in einer ASM-Diskgruppe
- Admin Passwörter sind Case sensitiv und genügen den Regeln des Default Profiles der Datenbank
- Auch externe User, wie z.B. Kerberos User, können im Password File erfasst und entsprechend privilegiert werden.

Im allgemeinen können Password Files für Administratoren nur empfohlen werden, wenn diese von remote authentifiziert werden müssen, um hoch privilegierte Aktionen auszuführen. Da die Datei Instanz-spezifisch ist (im Modus **`exclusive`**) entstehen auch hier hohe administrative Aufwände, um Passwörter in großen Umgebungen aktuell zu halten bzw. anzupassen.

Für eine zentralisierte Verwaltung administrativer User bieten sich unterschiedliche Verfahren von *strong authentication* an:

- Benutzern die über Kerberos authentifiziert wurden, können im Password File des Formats 12.2 auf Basis ihres Principal Name privilegiert werden (**`SYSDBA`**, **`SYSOPER`** etc.).
- Ebenso lässt sich ein Directory Service – vornehmlich Oracle Internet Directory (OID) oder Oracle Unified Directory (OUD) – einsetzen, um administrative User (**`SYSDBA`** und **`SYSOPER`**) über Passworte zu authentifizieren. Siehe in diesem Zusammenhang die init-Parameter **`LDAP_DIRECTORY_SYSAUTH`** und **`LDAP_DIRECTORY_ACCESS`**.
- Schließlich kann die administrative Authentifizierung und Privilegierung auch über SSL und entsprechende Zertifikate erfolgen. Dies kann auch über den Weg von OID und OUD geschehen.

Durch die Nutzung von zentralisierten Verfahren im Zusammenspiel mit OUD und OID kann auf Password Files verzichtet werden.

Proxy User

Proxy User fungieren als Stellvertreter für einen Proxy Client User. Authentifiziert wird der Proxy User und nicht der Proxy Client, in dessen Schema-Kontext der Proxy User agieren kann. In jedem Fall muss der Proxy Client ein lokaler Datenbank User sein und über **`grant`** ein explizites **`connect through`** autorisiert worden sein. Es gibt eine Reihe von Variationen:

- Der Proxy agiert mit allen Rollen des Clients
- Der Proxy agiert mit einer Teilmenge der Rollen des Clients

Beide Varianten können mit einer **`proxy only connect`** Klausel kombiniert werden. In diesem Fall kann sich der Proxy Client niemals direkt, sondern nur über den Proxy User mit der Datenbank verbinden.

Proxy User werden sowohl von den Tools, wie z. B. Datapump, als auch in Verbindung mit Wallets unterstützt, wie bei der Nutzung eines *external password store*. Ebenso lassen sie sie sich in Verbindung mit Kerberos Authentifizierungen einsetzten. Auch wenn Enterprise User in Verzeichnisdiensten wie OID oder OUD zur zentralen Benutzerverwaltung eingesetzt werden, können *proxy permissions* im Verzeichnisdienst definiert werden und auf einen Proxy Client in der Datenbank als Target verweisen.

Ganz allgemein sind Proxy User sehr nützlich, wenn Verbindungen zu einem Datenbank Schema hergestellt werden müssen, ohne das Passwort des betreffenden Users bekanntgeben zu müssen.

Authentifizierung über das Betriebssystem

Die Authentifizierung von (nicht administrativen) Datenbank Benutzern kann auch über das Betriebssystem erfolgen, indem OS Benutzer über ein festgelegtes Präfix (`os_authent_prefix`) einem Datenbank User zugeordnet werden. Die Authentifizierung bei der Datenbank erfolgt dann ohne Angabe eines Passworts. Diese Methode kann sinnvoll sein, wenn auf dem Datenbank-Server persönliche Benutzer eingerichtet wurden und für diese eine lokale Authentifizierung ohne Passwort in der Datenbank sinnvoll erscheint. Wesentlich sicherer für eine Authentifizierung ohne Passwort sind zweifelsohne Kerberos oder Zertifikate/SSL.

Die Authentifizierung über das Betriebssystem steht auch im Kontext von CDBs und ihren PDBs zur Verfügung. Dort allerdings nur für sogenannte Common User, sofern das Präfix (`os_authent_prefix`) auf den für Common User gültigen Wert `C##` eingestellt wurde.

Secure Sockets Layer (SSL) und Tansport Layer Security (TLS)

SSL und sein Nachfolger TLS ist ein Protokoll zur abgesicherten Kommunikation von Client- und Serversystemen, das auf einer Kombination von symmetrischen sowie Public und Private Keys aufbaut. Auf diese Weise unterstützt es sowohl die Verschlüsselung als auch Authentifizierung von Daten.

Um TLS für die Authentifizierung von Datenbank Benutzern einzusetzen – und gleichzeitig den Netzwerk-Verkehr zu verschlüsseln – sind folgende Schritte notwendig (stark verkürzt):

- Erstellung von Zertifikaten für die Client und Serverseite
- Speicherung der Zertifikate in einem Wallet oder alternativ in einem Hardware Security Modul (HSM)
- Konfiguration des Listeners (`listener.ora`) und Network Profiles (`sqlnet.ora`) – Port-Nummern, Protokoll/Authentication Service , Wallet Location.
- Analoge Arbeiten auf der Client-Seite (ohne die Listener Konfiguration)
- In der Datenbank: Einrichten eines externen Users (`identified externally`) mit dem *distinguished name* des Client Zertifikats.

Der Client kann sich nun ohne Passwort mit dem ihm zugeordneten Datenbank User verbinden.

Diese Art der Authentifizierung ist überall dort von Vorteil, wo

- Eine etablierte PKI-Infrastruktur existiert und
- Individuelle Daten User gefordert sind

Das TLS-Modell kann durch die Einbeziehung eines Verzeichnis-Dienstes (in Form von OID oder OUD) erweitert werden. In diesem Fall kann das Client-Zertifikat in den Verzeichnis Dienst geladen werden (im Kontext des dort erfassten Enterprise Users). Über den Verzeichnisdienst wird dann die Zuordnung zu einem Datenbank Benutzer geregelt.

Kerberos

Kerberos ist ein Authentifizierungsdienst für Computernetze, der mit Hilfe eines bei der Authentifizierung erhaltenen General-Tickets (TGT) Service Tickets für Kerberos-Dienste im Netz ausstellt. Die Nutzung dieser Dienste ist dann ohne weitere Authentifizierung möglich. Für die Nutzung von Kerberos muss – verkürzt – folgendes eingerichtet werden:

- Anlegen eines Service Users für die Datenbank im Kerberos Center (KDC) (in Windows Domains ist KDC in AD integriert)
- Konfiguration des Datenbank Servers und Installation eines Keyfiles – generiert aus dem Service User. Unter 12.2 existiert ein automatisches KDC Discovery; `krb5.conf` muss hier auf den Clients nicht explizit ausgebracht werden.
- Anlegen eines externen Datenbank Users (`identified externally`) mit dem User Principal Name des über Kerberos authentifizierten Endbenutzers.

Nach der einmaligen Anmeldung des Endbenutzers und dem damit verbundenen Erhalt des TGT wird im Hintergrund das Service Ticket für die Zieldatenbank gelöst und der Benutzer kann sich ohne Angabe eines Passwortes mit seinem externen Benutzer verbinden.

Kerberos kann auch zur Authentifizierung von Enterprise Users (EUS, s.u.) eingesetzt werden (s.u.).

Enterprise User (EUS)

Benutzer, die in einem zentralen Verzeichnisdienst – wie z.B. OID und OUD – gespeichert werden und für Datenbank Authentifizierungen aber auch Autorisierungen bereitstehen, werden Enterprise User genannt. Für die Authentifizierung dieser User stehen folgende Verfahren zur Verfügung:

- über ein Passwort, das in diesem Fall im Verzeichnisdienst vorgehalten und verifiziert wird
- über ein Kerberos Ticket, das die Datenbank verifiziert
- über ein Zertifikat, das in dem Verzeichnisdienst abgelegt ist.

Im Anschluss an die Authentifizierung werden die Enterprise User beim Connect spezifischen Datenbank Usern (`identified globally`) zugewiesen. Für die Zuweisung können folgende Verfahren genutzt werden:

- *Private Schema/Exclusive Schema* – in diesem Fall übernimmt letztendlich die Datenbank die Zuordnung über den Distinguished Name zu genau einem globalen User.
- *Shared Schema* – Das “Mapping” zwischen Enterprise Usern und global shared usern kann auf zwei Arten erfolgen
  - *Entry-Level* – hierbei wird der DN eines Enterprise Users, der einen beliebigen distinguished name haben kann, explizit auf ein shared Schema einer Datenbank „gemappt“.
  - *Subtree-Level* – hierbei wird ein kompletter Unterbaum eines Directory Baums auf das Shared Schema „gemappt“.

Um mit EUS arbeiten zu können müssen die verfügbaren Zieldatenbanken im Verzeichnisdienst registriert werden. Wenn das geschehen ist, bietet EUS sehr komfortable Möglichkeiten einer zentralisierten Benutzerverwaltung. Sowohl OID als auch OUD bieten die Möglichkeit, Benutzerdaten aus anderen Verzeichnisdiensten zu integrieren (*Proxy*/*Chaining*). Die Authentifizierung über Passwörter erfordert in diesem Fall zusätzliche Konfigurationsschritte.

Wie bereits erwähnt wurde können auch Proxy User auf diese Weise zugewiesen werden.

EUS kann auch im Kontext von PDBs genutzt werden, wobei jede PDB im Verzeichnisdienst im Großen und Ganzen wie eine reguläre Non-CDB behandelt wird. Allerdings muss das Standard-Verzeichnis für Wallets genutzt werden.

Identitätsnachweis in Multitier Umgebungen

In Umgebungen, die über Web Applikationen auf Oracle Systeme zugreifen werden Benutzer oftmals im Kontext der Applikation authentifiziert. Der Zugriff auf die Datenbank erfolgt hier über *Connection Pools* mit einem konfigurierten statischen User. Wenn in diesen Umgebungen die Identität des Endbenutzers erhalten bleiben soll und gleichzeitig der betreffende Endbenutzer nicht als Datenbank-Benutzer eingerichtet wurde, ermöglichen *Client Identifier* (CI) eine Markierung der betreffenden Session. CI sind beliebig und explizit zu setzende Zeichenketten die eine bestehende Session zusätzlich identifizieren helfen. Die Identifier werden explizit über das Paket **`DBMS_SESSION`** oder entsprechende OCI Calls gesetzt und stehen dann im Session und Auditing-Kontext zur Verfügung. Wenn sie im Rahmen eines *global application context* gesetzt werden, sind sie sogar Session- und Instanz-übergreifend (RAC) verfügbar.

Autorisierung

Autorisierung durch Privilegien

Privilegien regeln erlaubte Aktionen innerhalb und im Kontext von Datenbanken. Es können im wesentlichen drei Arten von Privilegien unterschieden werden:

- *Administrative Privilegien* – wie beispielsweise **`SYSDBA`** und **`SYSBACKUP`** – bündeln erlaubte Aktionen für spezifische administrative Aufgabenkomplexe, wie beispielsweise Backup Aktionen. Administrative Privilegien sind im Grunde vordefinierte administrative Rollen.
- *Systemprivilegien* – wie beispielsweise **`create table`** oder **`select any table`** – erlauben bestimmte Aktionen, wie das Anlegen von Tabellen oder das Lesen jedweder Tabelle in allen verfügbaren Schemata.
- *Objektprivilegien* – wie beispielsweise **`select`** auf Tabelle T im Schema S – erlauben bestimmte Aktionen im Kontext von Datenbank-Objekten, wie beispielsweise Views, Tabellen oder Prozeduren.

Privilegien werden Benutzern oder Rollen (mit Ausnahme von administrativen Privilegien) zugesprochen. Die Zuteilung (**`grant`**) kann bei Multitenant Datenbanken in zwei unterschiedlichen Kontexten erfolgen:

- *Commonly granted* – das Privileg wird über die Klausel **`CONTAINER=ALL`** zugeteilt und steht in jedem verfügbaren Container (auch in zukünftigen) zur Verfügung.
- *Locally granted* – das Privileg wird über die Klausel **`CONTAINER=CURRENT`** (*default* Wert) zugeteilt und steht nur in dem Container zur Verfügung, in dem es zugeteilt wurde.

Vor dem Hintergrund von *Common* und *Local Usern* und ebensolchen Rollen können leicht vielschichtige Gültigkeitsmuster entstehen, z.B. wenn ein *Common User* ein Systemprivileg lokal in nur einer PDB zugeteilt bekommt usw..

Rollen

Rollen bündeln System- und Objekt-Privilegien. Rollen können auch hierarchisch angelegt werden, indem sie weitere Rollen enthalten. Neben den bereits während der Installation angelegten vordefinierten Rollen, wie beispielsweise `DBA` oder `AUDIT_ADMIN`, können beliebige Benutzerrollen angelegt werden. Rollen teilen sich dabei den Namensraum mit Benutzern, können also niemals wie diese benannt werden.

Ähnlich den Benutzern können auch Rollen in CDBs in unterschiedlichen Kontexten existieren:

- *common roles* – werden im Root Container angelegt und sind in allen (auch zukünftigen) PDBs existent
- *local roles* – existieren nur in spezifischen PDBs, können nur dort genutzt werden und nur lokal zugeteilte Privilegien und Rollen enthalten.

Unabhängig vom Kontext der Rolle ist die Art ihrer Aktivierung. Hier existieren folgende Möglichkeiten in Bezug auf User:

- Eine Rolle wird dem Benutzer zugeteilt (`grant`) und ist standardmäßig aktiv, wenn dieser eine Session startet.
- Eine Rolle wird zugeteilt ist aber nicht standardmäßig aktiv. Sie muss vielmehr explizit über `set role` eingeschaltet werden. Wenn die Rolle zusätzlich über ein Passwort geschützt ist, muss dieses für die Aktivierung angegeben werden.
- Eine *secure application role* kann nur über ein vorher spezifiziertes PL/SQL Paket oder eine Prozedur mit *invoker's rights* aktiviert werden. Die Codierung eine Passwortes ist daher nicht nötig. Ebenso ein `grant`. Das zugeordnete Programmpaket kann dann beliebig komplexe Plausibiliäten abprüfen.
- Globale Rollen (`identified globally`) lassen sich nicht direkt Benutzern zuteilen. Sie werden vielmehr über Verzeichnisdienste im Rahmen von EUS aktiviert. In OID oder OUD werden hierbei sogenannte Enterprise Rollen definiert, die mit globalen Rollen auf Zieldatenbanken verknüpft werden. Nach der erfolgreichen Authentifizierung des Benutzers wird im Verzeichnisdienst die zugeteilte Enterprise Rolle bestimmt und beim Connect an der Datenbank die ihre zugeordnete globale Rolle aktiviert. Auf diese Weise lassen sich in *shared schemas* unterschiedliche Privilegien in Abhängigkeit vom aktuellen Enterprise User aktivieren. Es versteht sich, dass in solchen Umgebungen die feste Zuteilung von Privilegien an die Datenbank Benutzer wenig sinnvoll ist.
- Es ist ebenfalls möglich, das Betriebssystem für die Zuteilung von Rollen (z.B. über Gruppennamen) heranzuziehen. Diese Option ist nicht empfehlenswert und wird hier nicht diskutiert.
- Rollen können auch PL/SQL Programmeinheiten zugewiesen werden (*code based access control, CBAC*). Die für die Programmeinheit relevanten Privilegien werden dann bei der Ausführung derselben aktiviert. Dies macht beispielsweise Sinn bei *invoker's rights* Prozeduren.

Sonderfälle der Autorisierung

Systeme mit erhöhten Sicherheitsanforderungen erfordern oftmals feinere Zugriffskontrollen als das durch Grants auf Tabellen und Views möglich ist. Bei der *Label Security* – einer Form von *mandatory access control* – sind neben Grants auch sogenannte Label für den Zugriff verantwortlich. Diese Label werden sowohl einzelnen Rows als auch Benutzern zugeordnet und sind hierarchisch angelegt. So sieht beispielsweise ein Benutzer mit der *clearance sensitive* nur die Rows einer Tabelle X, die seinem oder einem rangniederen Label entsprechen.

Im Falle von Database Vault (DV) werden Zugriffsregel im Rahmen von *security realms* festgelegt. Auf diese Weise lassen sich herkömmliche System- und Objekt-Privilegien übersteuern (z.B. durch Ausschalten von `select any table`. DV kann eingesetzt werden, um Administratoren den Zugriff auf Benutzerdaten zu verwehren.

Real Application Security

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. Real Application Security (RAS) stellt vor diesem Hintergrund 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 besteht – neben Metadaten-Tabellen und PL/SQL Packages – aus Java Packages zur Einbindung in Java/JDBC-basierte Programme.

Im Zentrum von RAS stehen Application User und Application Roles. Die Funktionsweise und die Konfigurationsschritte von RAS lassen sich wie folgt kurz skizzieren:

- 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.

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