Große und kleine Unternehmen und Organisationen verwenden oft eine große Anzahl von (Oracle-) Datenbanken. Mehrere Hundert, wenn nicht gar Tausende Datenbanken sind dabei nicht ungewöhnlich. Um sichere Datenbanken zu gewährleisten, stehen viele Firmen vor dem Problem, diese große Menge an Datenbanken auf Sicherheitsrichtlinien zu überprüfen. Dabei taucht oft die Frage auf, gegen welchen Standard man testen soll oder aus gesetzlichen Gründen testen muss.

Überprüfung von Oracle-Datenbanken bezüglich Sicherheits-Richtlinien

Alexander Kornbrust, Red-Database-Security GmbH

Nachdem man die im Internet verfügbaren Oracle-Security-Richtlinien überprüft hat, werden oft eigene Richtlinien/Baselines erstellt, die der Firma/Organisation und den Datenbank-Installationen besser entsprechen. Der folgende Artikel beschreibt die existierenden Sicherheits-Richtlinien, gibt Hinweise zur Erstellung einer eigenen Richtlinie und weist auf typische Fallstricke bei der Erstellung einer Baseline/Richtlinie/Policy hin.

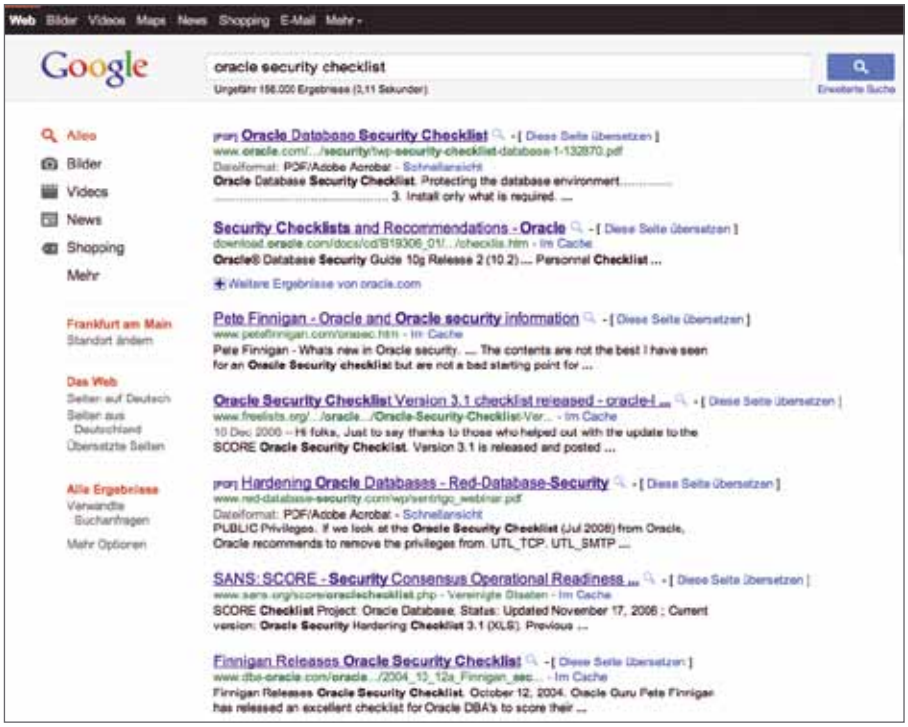

Abbildung 1: Google-Abfrage

- CIS Benchmark for Oracle (9-11)
- DISA Stig (9-11)
- SANS Score (9-11)
- NSA Guide (9)
- Oracle Security Checklist

Existierende Baselines

Die Suche nach den Oracle-Security-Baselines beginnt normalerweise mit Google. Eine Eingabe von „Oracle Security Baseline“, „Oracle Security Checklist“ oder „Oracle Security Policy“ liefert sehr viele Treffer, wobei die meisten sich auf folgende Richtlinien reduzieren lassen (siehe Abbildung 1):

CIS Benchmark

Der wohl verbreitetste Standard für Oracle-Datenbanken ist „CIS Benchmark“ (siehe http://benchmarks.cisecurity.org/en-us/?route=downloads.browse.category.benchmarks.servers.database.oracle). Dieser basiert zu großen Teilen auf dem Buch „Oracle Security Step-by-Step“, das von Pete Finnigan und vielen weiteren Autoren 2003 erstellt und zwei Jahre später aktualisiert wurde. In der neuen Version (11*g*) wurde die Checkliste von einem Sicherheitsexperten (ohne tiefes Oracle-Wissen) weiterentwickelt und enthält deshalb zahlreiche Fehler, die einem „Oracler“ nicht passieren würden (wie „revoke privilege to user“). Trotzdem ist es ein Vorteil dieses Benchmarks, dass bei vielen Tests die auszuführenden Befehle hinterlegt sind. Die Ausgabe aller zu analysierenden Auditdaten kann je Datenbank mehr als 100 Seiten lang sein, was die Analyse der gewonnenen Daten erschwert. Über den Sinn vieler der Tests (etwa TKProf aus Sicherheitsgründen vom DB-Server löschen, „automated backups should be verified“, „failsafe must be engaged“ …) kann außerdem diskutiert werden. Soll der CIS Benchmark in kommerziellen Programmen verwendet werden, ist zusätzlich eine kostenpflichtige Lizenz vom Center of Internet Security notwendig.

DISA Stig

Die Sicherheitsanforderungen der für die US-Behörden und Drei- oder Vier-Buchstaben-Organisationen (NSA, CIA,

FBI, NASA …) entwickelten Richtlinie (siehe http://iase.disa.mil/stigs/app_security/database/oracle.html) sind generell sehr hoch und oft auf europäische Unternehmen und Organisationen nicht anwendbar.

Ein Vorteil bei den DISA Stigs ist es, dass sowohl die SQL-Befehle als auch die zu erwarteten Ergebnisse beschrieben sind. Auch hier sind die Ausgaben unter Umständen sehr lang (siehe Listing 1).

SANS Score

Der SANS Score für Oracle (siehe http://www.sans.org/score/oraclechecklist.php) basiert wie der CIS Benchmark auf dem Buch „Oracle Security Step-by-Step“, wird jedoch seit geraumer Zeit nicht mehr weiterentwickelt.

NSA Guide

Inzwischen erstellt die NSA (National Security Agency (siehe http://www.nsa.gov/ia/_files/db/oracle9i_guide.pdf) keine eigenen Richtlinien, sondern verweist auf den CIS Benchmark bzw. den DISA Stig für Oracle.

Oracle Security Checklist

Die Checklist von Oracle ist mehr eine allgemeine Zusammenfassung von Hinweisen und Best Practices (siehe http://download.oracle.com/docs/cd/B19306_01/network.102/b14266/checklis.htm).

Zusammenfassend lässt sich sagen, dass die im Internet verfügbaren Security-Richtlinien in der Regel von Security-Experten, aber nicht von Oracle-Experten zusammengestellt werden (was zu Hinweisen wie „Alle Rechte von Public entfernen“ führt), zu viele Informationen liefern und oft nicht direkt in der eigenen Firma/Organisation angewendet werden können. Deshalb wird man in der Regel nicht darum herumkommen, eigene, angepasste Richtlinien zu erstellen.

Erstellen eigener Richtlinien

Bevor man anfängt, eigene Richtlinien zu erstellen, sollte man folgende Punkte diskutieren:

- Wird die Policy für alle Datenbanken benötigt oder sind separate Richtlinien für die verschiedenen Klassen (Test, Development, Pre-Live/Staging, Production) notwendig?
- Soll die Richtlinie für alle oder nur für die neu aufsetzten Systeme gelten?
- Wie soll die Baseline regelmäßig überprüft werden?
- Was passiert bei Verletzungen der Policy? Konsequenzen?
- Wie werden Ausnahmen behandelt? Prozess?

Danach sollte man überlegen, welche Klassen von Tests man überprüfen will. Dabei wird oft der Fehler gemacht, dass die Gruppen nicht strikt getrennt werden, was später zu Problemen führen kann. Mögliche Klassen von Tests sind:

- Unsichere Konfigurationseinstellungen (wie „UTL_FILE_DIR=*“)
- Unsichere Privilegien (wie „UTL_TCP“ an „PUBLIC“)
- (Absichtlich) schlechte Konfiguration / Hintertür (wie „GRANT DBA TO PUBLIC“)
- Passwörter (wie schwache Passwörter oder Passwörter aus Wörterbuch-Datei)
- Fehlende Patches (letzter CPU oder PSU fehlt)
- Verschlüsselung
- OS-Tests

Nachdem man sich auf die einzelnen Klassen geeinigt hat, kann man diese entsprechend ergänzen und die erwarteten Werte definieren. Dabei sollte es separate Tests anstatt größere Gruppen geben, beispielsweise „ 7.1.1 UTL_TCP an Public gegranted und 7.1.2 UTL_HTTP an Public gegranted“ anstelle von „7.1 – Gefährliche Privilegien von Public entziehen (UTL_TCP, UTL_HTTP ...)“.

Typische Fallstricke in Sicherheits-Richtlinien sind in der Regel zu allgemeine Aussagen wie „Backup muss vorhanden sein“ oder „Use least privilege for Applications“, die dann nicht getestet werden können und die normalerweise alle Benutzer der Richtlinie ignorieren. Weitere Probleme sind fehlende Nummerierungen der einzelnen Tests, die es schwierig machen, einzelne Punkte in einem Dokument zu referenzieren. Auch werden oft Tests, die schwieriger zu implementieren sind – wie beispielsweise „Passwort-Sicherheit“ – einfach weggelassen.

Die folgenden Beispiele zeigen unterschiedliche Implementierungen desselben Tests „Wer hat DBA-Rechte“. Je nach Implementierung sieht das Ergebnis anders aus und benötigt unterschiedlich viel Platz.

Beispiel 1 – einfaches SQL:

```
SQL> select grantee from
dba_role_privs where granted_
role='DBA';
MANI12
SYS
ALEX_ROLE
E1
DEV4
CHECK1
DIRK
ADMALEX
SYSTEM
CREDIT
RS
FLOWS_020100
```

```
DB-DG0014-ORACLE11 (Script)
From SQL*Plus: select username from dba_users where username in
(,ALLUSERS', ,AOLDEMO', ,AQDEMO', ,AQJAVA', ,AQUSER', ,AUC_GUEST',
,BI', ,CTXDEMO', ,DEMO8', ,DEV2000_DEMOS', ,HR', ,IX', ,OE', ,ORA-
BAMSAMPLES', ,PM', ,PORTAL_DEMO', ,PORTAL30_DEMO', ,QS', ,SCOTT',
,SECDEMO', ,SH', ,WK_TEST') or username like ,QS_%';
If any usernames are listed and are not documented in the System
Security Plan and authorized by the IAO, this is a Finding.
```

Listing 1

Beispiel 2 – SQL erzeugt eine Semikolon-separierte Liste:

```
SQL> SELECT SUBSTR (SYS_CON-
NECT_BY_PATH (grantee , ,;'),
     2) csv
  2  FROM (SELECT distinct
grantee , ROW_NUMBER () OVER
(ORDER BY grantee ) rn,
 COUNT (*) OVER () cnt
  3  FROM (select distinct
grantee from dba_role_privs
where granted_role='DBA')
)
  4  WHERE rn = cnt
  5  START WITH rn = 1
  6  CONNECT BY rn = PRIOR rn
     + 1
  7  ;
ADMALEX;ALEX_ROLE;CHECK1;CREDIT
;DEV4;DIRK;E1;MANI12;SYS;SYSTEM
```

Beispiel 3 – SQL erzeugt eine Semikolon-separierte Liste, Default-Werte sind entfernt:

```
SQL> SELECT SUBSTR (SYS_CON-
NECT_BY_PATH (grantee , ,;'),
     2) csv
  2  FROM (SELECT distinct
grantee , ROW_NUMBER () OVER
(ORDER BY grantee ) rn,
 COUNT (*) OVER () cnt
  3  FROM (select distinct
grantee from dba_role_privs
where granted_role='DBA'
and grantee not in (,SYS','SYS
TEM','FLOWS_020100','CTXSYS',
,SYSMAN','BAM','ORASSO','PORTAL
','WKSYS')))
  4  WHERE rn = cnt
  5  START WITH rn = 1
  6  CONNECT BY rn = PRIOR rn
     + 1
  7  ;
ADMALEX;ALEX_ROLE;CHECK1;CREDIT
;DEV4;DIRK;E1;MANI12
```

Der folgende Test überprüft, ob das File „tkprof" existiert. Beispiel 1 – CIS Benchmark:

```
ls -la $ORACLE_HOME/bin/tkprof
```

Beispiel 2 – falls nicht existent, wird kein Fehler ausgegeben:

```
 test -e $ORACLE_HOME/bin/tk-
prof &amp;amp;&amp;amp; echo „tkprof
exists"
```

Überprüfen von Baselines

Wenn man eine Baseline erstellt hat, sollte man diese auch regelmäßig überprüfen, um den Status und Verletzungen der Richtlinie feststellen zu können. Dies kann mithilfe von eigenen Programmen, Skripten oder kommerziellen Tools (wie Oracle Gridcontrol, McAfee Security Scanner for Databases) erfolgen. Dazu einige Beispiel-Screenshots, die mit dem McAfee Security Scanner for Databases erstellt wurden. Die gelben und roten Punkte stellen Verletzungen einer Baseline dar (siehe Abbildungen 1).

Meherere Datenbanken

Bei der Überprüfung mehrerer Datenbanken wird man feststellen, dass einige Tests auf (fast) allen Datenbanken eine Verletzung der Regel liefern. In diesem Fall sollte man überlegen, ob der Test so richtig oder ob das gesamte Deployment der Datenbanken zu ändern ist. Zusätzlich sollte man einen Prozess implementieren, der es erlaubt, Verletzungen der Policy wie „REMOTE_OS_AUTHENTITCATION = true" auf SAP-Systemen zu akzeptieren. Dabei sollte sich der DBA diese Verletzung vom Management absegnen lassen (Risk Acknowledgement oder Risk Acceptance).

Auch wenn es anfangs kompliziert klingt, ist die Erstellung einer Baseline nicht so schwierig, sofern man die Sache methodisch angeht und bei der Erstellung der Baseline immer die Implementierung im Auge behält.

Alexander Kornbrust
Red-Database-Security GmbH
ak@red-database-security.com

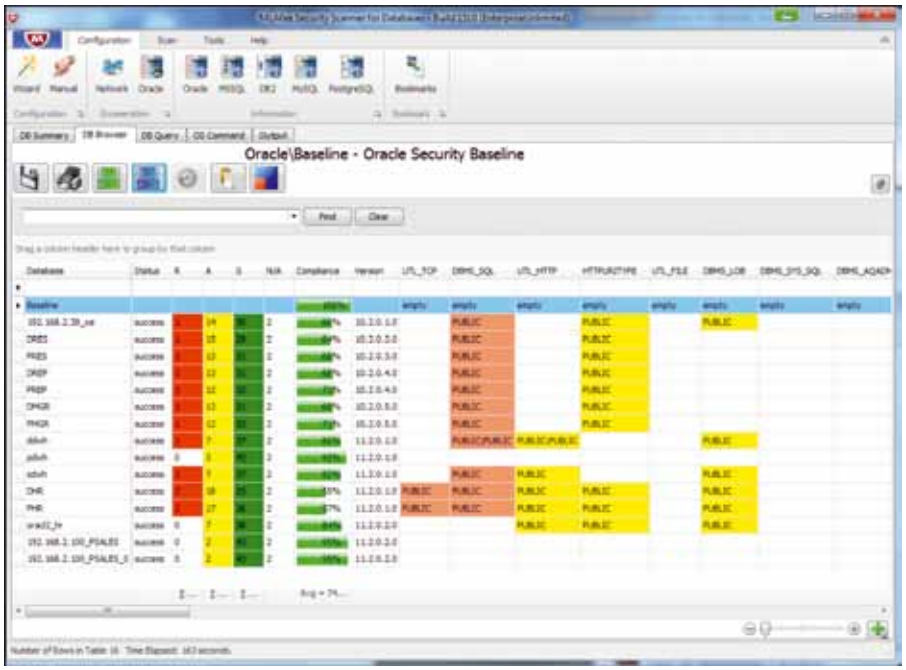

Abbildung 1: Verletzungen der Baseline einer Datenbank