

XV Konferencja PLOUG
Kościelisko
Październik 2009

Project Lockdown – wybrane elementy i narzędzia

Grzegorz Jakusz-Gostomski
OPITZ CONSULTING Kraków Sp.z o.o.

grzegorz.jakusz-gostomski@opitz-consulting.com

Abstrakt. Project Lockdown jest próbą przedstawienia ustandaryzowanego i uniwersalnego podejścia do problematyki zabezpieczenia baz danych Oracle. Praca Arupa Nanda „Project Lockdown. A phased approach to securing your database infrastructure” stała się punktem wyjścia tego opracowania, którego głównym celem jest zaprezentowanie wybranych technik zwiększenia bezpieczeństwa baz danych. Techniki te zostały wzbogacone o dodatkowe elementy systemu bezpieczeństwa dostępne w Oracle 11g oraz zalecenia przedstawiane w benchmarkach bazodanowych publikowanych przez SANS (SCORE checklist project), CIS (Security Configuration Benchmark for Oracle Database Server 11g v. 1.0.1), Oracle (Oracle Database Security Checklist) czy też DISa dla Departamentu Obrony USA (Oracle Database Security Checklist). Na zakończenie zaprezentowane zostały narzędzia, które pomagają zarówno w ocenie poziomu bezpieczeństwa, jak i w codziennej ochronie systemów bazodanowych.

1. Wstęp

Project Lockdown jest próbą przedstawienia ustandaryzowanego podejścia do zabezpieczenia baz danych Oracle. Pojęcie „Project Lockdown” pierwotnie wywodzi się z pracy Arupa Nanda „*Project Lockdown. A phased approach to securing your database infrastructure*”. W pracy tej autor zaprezentował listę kroków zmierzających do zabezpieczenia środowiska baz danych Oracle w wersji 9-10. Praca ta stała się punktem wyjścia poniższego opracowania, w którym skupiłem się na kilku wybranych zagadnieniach takich jak zabezpieczenie listenera, mechanizmach audytu czy szyfrowaniu danych. Znalazły się tu również dodatkowe elementy systemu bezpieczeństwa dostępne w Oracle 11g oraz zalecenia przedstawiane w dostępnych benchmarkach dla baz danych publikowanych przez SANS (*SCORE checklist project*), CIS (*Security Configuration Benchmark for Oracle Database Server 11g v. 1.0.1*), Oracle (*Oracle Database Security Checklist*) czy też DISa dla Departamentu Obrony USA (*Oracle Database Security Checklist*). Na końcu zaprezentowane zostały narzędzia RepScan (firmy Red-Database-Security GmbH) oraz Hedgehog, które pomagają zarówno w ocenie poziomu bezpieczeństwa, jak i w ochronie baz danych.

Projekt Lockdown składa się z czterech czasowo zdefiniowanych etapów. Każdy z etapów odnosi się odpowiednio do dnia, tygodnia, miesiąca oraz kwartału. Oczywiście nie trzeba ściśle przestrzegać tak narzuconego harmonogramu. Służy to wyłącznie wyodrębnieniu faz o różnych poziomach skomplikowania a więc i o różnych wymaganiach czasowych oraz przedstawieniu pewnej ścieżki działania, która definiuje, jakie kroki należy wykonać przed przystąpieniem do następnego etapu. Przedstawione działania są na tyle uniwersalne i kompleksowe, że można je z powodzeniem zastosować na środowiskach o różnej specyfice działania i złożoności. Operacje te, jak zresztą wszelkie działania mające na celu zabezpieczenie systemów informatycznych, nie powinny być działaniami jednorazowymi. Bezpieczeństwo takich systemów jest, bowiem bytem dynamicznym, podlegającym ciągłym zmianom i modyfikacjom. Biorąc pod uwagę fakt, że nie da się zbudować systemu bezpiecznego w 100%, działania te powinny być na trwale wpisane w pracę i politykę działów odpowiedzialnych za zarządzanie bazami danych.

2. Etap 1

Etap 1, który powinien trwać nie dłużej niż jeden dzień, obejmuje zadania o niskim poziomie skomplikowania. Na zadania te składają się następujące kroki:

- sprawdzenie kont użytkowników i haseł,
- zabezpieczenie słownika danych,
- ograniczenie logowania SYSDBA,
- zabezpieczenie listenera,
- identyfikacja i ograniczenie przywilejów.

Na tym etapie skupię się na zadaniu zabezpieczenia listenera.

2.1. Zabezpieczenie listenera

Pomimo swej prostej budowy (listener składa się z 2 plików wykonalnych oraz 3 plików konfiguracyjnych) stanowi on potężne narzędzie w rękach wprawnego napastnika. Listener nie tylko pełni funkcję *proxy* – procesu nasłuchującego, przekierującego połączenia do bazy danych, ale może także uruchamiać polecenia systemu operacyjnego i w konsekwencji dać pełną kontrolę nie tylko nad bazą danych, ale także nad serwerem, na którym działa. Temat zabezpieczenia listenera

jest często zaniewany przez administratorów, co w połączeniu z licznymi dziurami bezpieczeństwa prowadzić może do poważnych zagrożeń.

Do znanych problemów związanych z listenerem możemy zaliczyć: brak domyślnej blokady konta administracyjnego, przesyłanie haseł otwartym tekstem czy też możliwość logowania się do listenera nie tylko hasłem, ale także hashem tego hasła. Do najczęstszych form ataku na listenera należą *Buffer Overflow* czy ataki typu DoS (*Denial of Service*).

Pierwszym etapem uszczelniania listenera (krok ten powinien rozpoczynać wszelkie prace mające na uwadze bezpieczeństwo systemów) powinna być instalacja odpowiednich patchset'ów oraz Oracle CPU (*critical patch update*). W wersjach wcześniejszych niż 10g, zabezpieczenie listenera sprowadzało się zwykle do utworzenia hasła dostępu. Jeżeli nie mamy ustawionego hasła administracyjnego, każdy może się podłączyć do procesu nasłuchującego i wykonywać komendy bez dodatkowej autoryzacji. Aby ustawić hasło wykonujemy komendę **SET PASSWORD** z poziomu narzędzia lsnrctl (ewentualnie `change_password` a następnie `save_config`) albo umieszczając bezpośrednio w pliku listener.ora wpis **PASSWORD_[listener name] = password** (oczywiście w tym przypadku uwagę zwracamy na prawa dostępu do tego pliku). W Oracle 10g problem ten częściowo rozwiązano umożliwiając jedynie lokalnym administratorom na łączenie się bez hasła do procesu nasłuchującego (a mówiąc dokładniej właścicielom procesu tnslnr). Mimo wszystko zalecanym podejściem jest ustalenie hasła dla listenera bez względu na wersję bazy danych.

Kolejnym krokiem jest ograniczenie możliwości modyfikacji ustawień listenera. Zmiany takie można zwykle przeprowadzić albo z poziomu samego programu lsnrctl albo przez zmiany w pliku konfiguracyjnym. Jako odpowiedź na szereg udokumentowanych ataków polegających na przekierowywaniu TRC_FILE i LOG_FILE na dowolne lokalizacje w obrębie serwera (dzięki temu możliwość zmiany licznich plików konfiguracyjnych), Oracle wprowadził parametr **ADMIN_RESTRICTION** blokujący możliwość zmiany ustawień z poziomu listenera. Parametr ten instruuje listenera, aby nie przyjmował żadnych poleceń rozpoczynających się od polecenia SET. Wszelkie zmiany powinny być odtąd wykonywane na poziomie pliku konfiguracyjnego. Parametr ten ustawia się w listener.ora jako:

```
ADMIN_RESTRICTION_[listener_name] = ON
```

Dobrym zwyczajem jest również włączenie logowania na poziomie listenera. Dzięki temu uzyskamy dostęp do wielu informacji na temat operacji wykonywanych przez program nasłuchujący, rodzaju połączeń, protokołów i hostów, z których łączono się, a przede wszystkim informacji na temat nieudanych prób połączeń. Log taki dużo mówi nam o ewentualnych atakach i powinien być na bieżąco analizowany. Rejestracje zdarzeń listenera uruchamia się za pomocą komendy **SET LOG_STATUS = ON** (wcześniej możemy ustawić odpowiednio LOG_DIRECTORY oraz LOG_FILE). Inaczej sprawa wygląda z informacjami typu trace. Taki tracing zawiera szczegółowe informacje o działaniu listenera i produkuje olbrzymią ilość danych niekoniecznie potrzebnych w codziennych pracach administracyjnych. Znane są zresztą ataki typu DoS ustawiające poziom tracingu na SUPPORT i powodujące wyczerpanie się dostępnego miejsca i w efekcie unieruchomienie listenera. Na co dzień, jeżeli nie ma potrzeby szczegółowej analizy ruchu sieciowego powinniśmy wyłączyć tracing za pomocą komendy **SET TRC_LEVEL = OFF**.

Kolejną funkcjonalnością procesu listenera jest EXTProc (*external procedures*) umożliwiająca uruchamianie zewnętrznych procedur (napisanych w C, Javie czy w innych językach programowania). Funkcjonalność ta jest rzadko wykorzystywana a stanowi potencjalne zagrożenie dla bezpieczeństwa bazy danych. Głównym źródłem zagrożenia jest agent extproc, który uruchamiany jest z prawami właściciela binariów Oracle i który w imieniu bazy wykonuje zewnętrzną procedurę a jej wyniki przekazuje do bazy. Warto zapamiętać, że w obecnych wersjach Oracle, baza sama potrafi wywołać zewnętrzną procedurę. Dlatego funkcjonalność ta powinna być usunięta z listenera (należy jednak pamiętać o pewnych funkcjonalnościach, które wymagają konfiguracji extproc – np. MTS na Windowsie). Konfiguracja extproc w listenerze zwykle wygląda następująco:

```
LISTENEREXTPROC=
(DESCRIPTION=
(AADDRESS=
(PROTOCOL=ipc)(KEY=extproc)))
SID_LIST_LISTENEREXTPROC=
(SID_LIST=
(SID_DESC=
(PROGRAM=extproc)
(ENVS="EXTPROC_DLLS=ONLY:/home/xyz/mylib.so:/home/abc/urllib.so,
LD_LIBRARY_PATH=/private/xpm/lib:/private/mylibs,
MYPATH=/usr/ucb:/usr/local/packages, APL_ENV_FILE=/apl/conf/env.txt")
(SID_NAME=extproc)
(ORACLE_HOME=/oracle)))
```

W przypadku, gdy potrzebujemy powyższej funkcjonalności, należy tak dostosować parametry listenera, aby zmienna ENVS zawierała jedynie te ścieżki i biblioteki, do których baza danych powinna mieć dostęp.

3. Etap 2

Etap 2 obejmuje zadania możliwe do wykonania w przeciągu jednego tygodnia. Zadania te charakteryzują się nieco wyższym stopniem zaawansowania w porównaniu z krokami etapu 1. Często są to działania, które powinny być przeprowadzane już na etapie instalacji i konfiguracji środowiska bazodanowego. W tej fazie powinniśmy się skupić na następujących tematach:

- usunięcie nie używanych komponentów,
- ograniczenie autentykacji OS,
- zabezpieczenie środowiska SQLPlus,
- weryfikacja przywilejów PUBLIC (ACL),
- sprawdzenie wielkości limitów na przestrzeniach tabel,
- audyt użytkowników.

W tym rozdziale ograniczę się do tematów związanych z zabezpieczeniem środowiska SQL-Plus, przywilejami PUBLIC oraz śledzeniem zachowań użytkowników.

3.1. Zabezpieczenie środowiska SQLPlus

Jedną z cech środowiska SQLPlus umożliwiającą przeprowadzenie potencjalnie niebezpiecznego ataku jest możliwość opuszczenia środowiska poleceniem host. Jako przykład posłużyć może próba wykonania skryptu na zdalnej bazie:

```
sqlplus user/userab@zdaInadb @skrypcik.sql
```

Użytkownik może przerwać wykonywanie powyższego skryptu za pomocą Control-C, ewentualnie każdej innej kombinacji klawiszy, która przerywa działanie programu. Następnie wykorzystując polecenie **host** uzyskuje dostęp do powłoki systemu operacyjnego.

Istnieją dwa sposoby na zablokowanie takich operacji – przez wykorzystanie tzw. profilu użytkownika SQLPlus albo skorzystanie z trybu restrict.

Profile użytkowników sqlplus'a znajdują się w widoku **SQLPLUS_PRODUCT_PROFILE**. Wykonując poniższe polecenie blokujemy wszystkim użytkownikom możliwość wykonywania polecenia **host** z poziomu sqlplus'a.

```
SQL> insert into system.SQLPLUS_PRODUCT_PROFILE values
('SQL*Plus', '%', 'HOST', null, null, 'DISABLED', null, null);
```

Od tego momentu przy próbie wywołania polecenia **host** otrzymamy błąd:

```
SQL> host
```

```
SP2-0544: Command "host" disabled in Product User Profile
```

Powyższe podejście możemy również wykorzystać do wyłączenia innych poleceń dostępnych z SQL*Plusa takich jak **STARTUP**, **EXECUTE**, **CONNECT** czy **VARIABLE** jak i również poleceń SQL wywoływanych z poziomu środowiska SQL*Plus takich jak **ALTER**, **GRANT**, **INSERT**, **AUDIT** czy **TRUNCATE**.

Drugą metodą, mniej restrykcyjną i nieco prostszą (dostępną od wersji Oracle 9i), jest skorzystanie z opcji **restrict** przy wywoływaniu SQL*Plusa.

```
sqlplus -restrict 1 scott/tiger
```

Wówczas w odpowiedzi na wywołanie polecenia **HOST**, dostaniemy:

```
SP2-0738: Restricted command "host" not available
```

I tak przy pomocy poziomu 1 możemy zablokować nie tylko **HOST** ale także **EDIT**, na poziomie 2 blokujemy dodatkowo **SAVE**, **SPOOL** i **STORE** a na poziomie 3 **GET** i **START**.

Najważniejszą różnicą między zaprezentowanymi tu rozwiązaniami jest to, że w przypadku polecenia **restrict** również użytkownik **SYS** jest objęty tymi restrykcjami.

3.2. Przywileje PUBLIC

Trudno zdefiniować pojęcie **PUBLIC**, nie jest ono ani rolą ani użytkownikiem, chociaż w wielu przypadkach jest zarządzane w sposób podobny dla obu. W literaturze nazywane jest zwykle grupą. Grupa **PUBLIC** jest stałą cechą bazy danych Oracle, nie można jej bowiem wyłączyć czy odinstalować. Jest to jedyna grupa w bazie, do której jednocześnie należą wszyscy użytkownicy bazy. Użytkownicy dziedziczą uprawnienia związane z **PUBLIC**, dlatego też należy ostrożnie podchodzić do przydzielania czy odbierania uprawnień do niej przypisanych. Przykładowo nadając **PUBLIC** uprawnienie **SELECT ANY TABLE** a następnie odbierając je, odbieramy je wszystkim użytkownikom bazy (a nie tylko tym, którym za pomocą wcześniejszego polecenia je nadaliśmy), co może wywołać niekorzystne konsekwencje. O ile istnieje możliwość przypisania ról do grupy **PUBLIC**, nie powinniśmy nigdy tego wykorzystywać. W Oracle 10.2 grupie **PUBLIC** nie jest przypisana żadna rola, co szybko można zweryfikować:

```
SQL> select * from dba_role_privs where GRANTEE = 'PUBLIC';
no rows selected
```

Domyślnie, każda instalacja Oracle dostarczana jest z dość sporym zbiorem uprawnień do obiektów nadanych grupie **PUBLIC**. I tak w Oracle 9 obiektów takich jest około 13 tys., w Oracle 10g – 21 tys. a w Oracle 11 około 27 tys. Z jednej strony jest ich na tyle dużo, że niemożliwa staje się dokładna ich analiza, a z drugiej strony bywa i tak, że po wgraniu nowego patch-setu uprawnień **PUBLIC** są powtórnie przywracane. Skupić się, więc powinniśmy jedynie na najbardziej krytycznych funkcjonalnościach i obiektach, do których grupa **PUBLIC** ma dostęp. W szczególności chodzi o uprawnienia **EXECUTE** na takich pakietach systemowych jak **DBMS_RANDOM** (**DBMS_CRYPTO** in 11g), **UTL_FILE**, **UTL_HTTP**, **UTL_SMTP**, **UTL_TCP** czy **UTL_INADDR**, uprawnieniach systemowych czy dostępu do ważnych obiektów **SYS**.

Oracle 11g ogranicza możliwość korzystania z wymienionych pakietów wprowadzając tzw. Fine Grained Access Control on Network Services (w skrócie ACL, zaimplementowany za pomocą pakietu **dbms_network_acl_admin**). Zamiast dawać grupie PUBLIC uprawnienia EXECUTE do pakietów utl_smtp, utl_mail czy utl_inaddr, za pomocą ACL definiuje się, którzy użytkownicy mogą dokonywać wywołań sieciowych i w odniesieniu do których hostów. Aby sprawdzić uprawnienia związane z ACL można skorzystać z widoków **DBA_NETWORK_ACL** oraz **DBA_NETWORK_ACL_PRIVILEGES**.

Jeżeli chodzi o uprawnienia systemowe, to również w tym wypadku powinniśmy kierować się zasadą – żadnych uprawnień dla grupy PUBLIC. O wiele bardziej rozsądnym jest rozwiązanie polegające na stworzeniu roli, przypisaniu jej konkretnych uprawnień systemowych a następnie przypisaniu jej konkretnym użytkownikom. W szczególności powinniśmy kontrolować dostęp do wszelkich uprawnień zawierających ANY w swojej nazwie. Oczywiście równie niebezpieczne są uprawnienia systemowe nie zawierające słowa Any, takie jak ALTER SYSTEM, ALTER DATABASE, AUDIT czy ALTER PROFILE. Za ich pomocą jeszcze bardziej powiększa się wachlarz metod nieautoryzowanego dostępu do bazy danych.

3.3. Audyt użytkowników

Zaufanie do użytkowników to jedna sprawa a kontrolowanie ich zachowań to druga. Tą samą zasadą należy się kierować w odniesieniu do użytkowników korzystających z bazy danych, a w szczególności do użytkowników posiadających specjalne przywileje, a więc takich, którzy łączą się z bazą za pomocą ról SYSDBA czy SYSOPER.

Już od wersji Oracle 9 administratorzy mają do dyspozycji parametr inicjalizacyjny **AUDIT_SYS_OPERATION**, który włącza śledzenie operacji wykonywanych przez użytkowników z rolami SYSDBA albo SYSOPER. W zależności od ustawień parametru AUDIT_TRAIL (os, db, extended, xml), dane z śledzenia będą zapisywane albo w samej bazie (SYS.AUD\$) lub w plikach systemu operacyjnego w formie tekstowej albo xml. Przy tej okazji warto również zwrócić uwagę na domyślnie istniejący auditing rejestrujący logowania użytkowników uprzywilejowanych, uruchomienia i zatrzymania bazy oraz zmiany strukturalne. Pliki z takiego audytu domyślnie trzymane są w \$ORACLE_HOME/rdbms/audit/ora_XXXXX.aud.

Aby aktywować rejestrację logowań użytkowników, należy po ustawieniu parametru AUDIT_TRAIL (i ewentualnie AUDIT_SYS_OPERATIONS) uruchomić audyt za pomocą polecenia AUDIT SESSION (użytkownik taki musi mieć uprawnienie AUDIT SYSTEM). Od tego momentu przy pomocy widoków **DBA_AUDIT_TRAIL** i **DBA_AUDIT_SESSION** uzyskujemy informacje na temat monitorowanych zdarzeń:

```
SQL> select OS_USERNAME, USERNAME, action_name, SESSIONID, RETURNCODE from
dba_audit_session;
```

OS_USERNAM	USERNAME	ACTION_NAME	SESSIONID	RETURNCODE
oraas	DBSNMP	LOGOFF	1656	0
	SYSMAN	LOGON	1657	0
oraas	DBSNMP	LOGOFF	1659	0
oraas	SCOTT	LOGON	1660	1017

Dane takie można później wykorzystać do analizy profili zachowań użytkowników np. logowania nieudane (returncode≠0), próby logowania nieistniejącym użytkownikiem czy złym hasłem (returncode=1017), próby logowania poza godzinami pracy (na podstawie zapisanego czasu operacji) czy też sprawdzenie użytkowników współdzielących konto w bazie danych (logowania z różnych terminali tym samym użytkownikiem).

W przypadku aktywacji audytu należy pamiętać, że dane te zapisywane zostają w tabeli **SYS.AUD\$** w przestrzeni tabel SYSTEM. Efektem zapelnienia się tej przestrzeni będzie zawie-

szenie się bazy (atak DoS), należy, więc pamiętać o regularnym monitoringu tej tabeli oraz ewentualnym usuwaniu czy archiwizowaniu nadmiarowych danych. Istnieje też możliwość przemieszczenia tej tabeli do innej przestrzeni tabel, ale w takim przypadku należy skontaktować się z Oracle Support.

W przypadku Oracle 11g, czy to instalując nową bazę przy pomocy asystenta instalacji czy aktualizując poprzednie wersje, domyślnie otrzymujemy włączoną opcję audytu (opcja `extended security`). Audyt ten obejmuje m.in. operacje takie jak **alter any procedure**, **alter profile**, **alter system**, **create any table**, **create user** czy **grant any role** aby wymienić jedynie najważniejsze z nich. Można oczywiście wyłączyć tak szczegółowy monitoring albo przypisując parametrowi `AUDIT_TRAIL` wartość `NONE` albo używając komendy SQL **noaudit**.

4. Etap 3

Wraz z etapem 3 przechodzimy do zagadnień zarówno bardziej skomplikowanych jak i czasochłonnych. Ze względu na możliwe implikacje i wpływ na działanie naszych aplikacji działania te mogą wymagać dokładnych analizy oraz testów.

- Usunięcie haseł ze skryptów
- Kontrola dostępu do obiektów bazy danych
- Ograniczenie dostępu z określonych maszyn

4.1. Kontrola dostępu do obiektów

Kontrola dostępu do krytycznych obiektów, jaki i kontrola zmian jest bardzo często wymagana przez istniejące uregulowania kontrolne (HIPAA, GLB) czy też wytyczne wewnętrznej polityki bezpieczeństwa w firmie. W tej sytuacji ponownie można zastosować rejestrację operacji użytkowników, wykorzystaną już wcześniej do monitorowania profili zachowań użytkowników.

O ile w przypadku śledzenia użytkowników korzystamy z audytu na poziomie sesji, o tyle w przypadku obiektów zastosowany powinien być audyt na poziomie dostępu (*access level*). Oczywiście ze względu na dodatkowy narzut obliczeniowy wprowadzany przez system monitoringu, należy ograniczyć się do najbardziej krytycznych kolumn, tabel i procedur. Monitoring taki uruchamiamy w następujący sposób:

```
AUDIT SELECT ON CZESIU.TEST BY ACCESS
```

Gdy jednak interesuje nas jedynie fakt dostępu do obiektu możemy ustawić śledzenie na poziomie sesji:

```
AUDIT SELECT, INSERT, DELETE ON CZESIU.TEST BY SESSION
```

Od tego momentu za pomocą widoku `DBA_AUDIT_TRAIL` mamy dostęp do dziennika dostępu do danych obiektów. Jeżeli już uzyskamy informacje na temat częstotliwości i sposobu dostępu do określonych obiektów w pewnej jednostce czasu, możemy włączyć bardziej szczegółowy audyt na poziomie dostępu do wybranych obiektu.

```
AUDIT SELECT, INSERT, DELETE ON CZESIU.TEST BY ACCESS
```

Wspomniane wyżej podejście, jako kontrola dostępu do obiektów bazodanowych ma jednak jedną drobną wadę. Obiekty, które chcemy monitorować muszą być dostępne w bazie. W przypadku, gdy monitoring taki objąć powinien również obiekty nowe, nieistniejące jeszcze w momencie włączenia audytu, skorzystać możemy z poniższego polecenia:

```
AUDIT SELECT ON DEFAULT BY SESSION;
```

Od tego momentu wszystkie nowo utworzone obiekty będą podlegały śledzeniu. W widoku **ALL_DEF_AUDIT_OPTS** można sprawdzić, jakie są aktualne ustawienia śledzenia dla nowo tworzonych obiektów

```
SQL> select * from all_def_audit_opts;
ALT AUD COM DEL GRA IND INS LOC REN SEL UPD REF EXE FBK
S/S S/S S/S S/S S/S S/S A/A S/S S/S S/S S/S -/- S/S S/S
```

5. Etap 4

Etap 4, jako najdłuższy, obejmuje zadania, dla których trudno jest zdefiniować stałe ramy czasowe. Wraz ze zmieniającymi się wymaganiami biznesowymi oraz funkcjonalnymi dla bazy danych i aplikacji, zmieniają się i wymagania stawiane przed systemem bezpieczeństwa. Z założenia przyjmuje się, że trwają one około kwartału, aczkolwiek często mogą trwać znacznie dłużej. Na tym etapie skupić się powinniśmy na następujących tematach:

- Fine Grained Auditing
- Virtual Private Database
- Szyfrowanie danych
- Zabezpieczenie kopii bezpieczeństwa

5.1. Virtual Private Database

Oficjalnie termin Virtual Private Database (od wersji Oracle 8, nazywany często również Fine Grained Access Control) używany jest w odniesieniu do tzw. Row-Level Security (RLS, zaimplementowanego za pomocą pakietu **DBMS_RLS**) przy wykorzystaniu mechanizmu kontekstu aplikacji (samo jednak zastosowanie kontekstu aplikacji nie jest wymagane w RLS). Często jednak używa się go na określenie wszelkich technik związanych z tzw. śledzeniem niskopoziomowym bez względu na implementację. Warto również wspomnieć, że pierwotnie VPD wywodził się z specjalnej wersji Oracle – Trusted Oracle – używanej między innymi przez DoD (Department of Defence).

VPD pozwala na ograniczenie dostępu do wybranych kolumn w oparciu o zaimplementowanie polityki bezpieczeństwa, zdefiniowanej przy pomocy PL/SQLa (pakiet **DBMS_RLS**, domyślnie nikt nie ma prawa **EXECUTE** na tym pakiecie). Po krotce proces dostępu do danych kontrolowany jest przez funkcję PL/SQL, która w zależności od otrzymanych wartości zwraca pewien ciąg znaków (ciąg znaków dołączany, jako predykat do klauzuli **WHERE**). Tak stworzona funkcja jest następnie rejestrowana czy też łączona z obiektem, którego ma dotyczyć. Może to być tabela, widok czy synonim. W momencie pojawienia się zapytania do takiego obiektu, wartość zwracana przez powiązaną funkcję jest dodawana do pierwotnego zapytania i w sposób niejawni, transparentnie wpływając na zbiór danych wynikowych.

Istotną cechą VPD jest uzależnienie zwracanego wyniku od kontekstu, w jakim zapytanie zostało wykonane. I tak kontekst możemy zbudować w oparciu o nazwę użytkownika, dział z którego pochodzi czy też np. aplikację z której wykonano dane zapytanie. Przykładem kontekstu może być np. `user_level` ustalany na poziomie wyzwalacza np. za pomocą wcześniej utworzonej procedury `context_pkg.set_user_level_ctx(l_level)`, który później można użyć do ustalania warunków zwracanych przez VPD:

```
SQL> select sys_context('USER_LEVEL_CTX','LEVEL') from dual;
```

Można również skorzystać z już istniejących kontekstów jak np. `IP_ADDRESS`, `CURRENT_USER` czy np. `ISDBA`. Możemy zdefiniować, jakiego typu zapytań dana polityka ma doty-

czyć. Czy mają nią być objęte wszystkie wywołania DML, czy tylko niektóre z nich. Warto tu podkreślić, że dany obiekt może być związany z więcej niż tylko z jedną funkcją. Wówczas wszystkie odpowiednie filtry będą nałożone przezroczyście przed zwróceniem danych do użytkownika.

W przypadku, gdy jednemu z użytkowników chcemy dać pełny dostęp do znajdujących się danych, bez względu na istniejące ograniczenia, możemy użyć uprawnienia **EXEMPT ACCESS POLICY**, które umożliwi posiadaczowi go użytkownikowi na obejście całej struktury VPD.

Krytycznym elementem wdrażania polityki VPD jest identyfikacja wrażliwych tabel oraz kolumn, do których chcielibyśmy ograniczyć dostęp. Często potrzeba do tego nie tylko wiedzy z zakresu działania baz danych, ale i wiedzy na temat budowy danej aplikacji. VPD uwalnia programistów od implementacji mechanizmu kontroli dostępu do poufnych danych na poziomie aplikacji. W ten sposób otrzymujemy do dyspozycji efektywną i wydajną platformę kontroli dostępu.

Oprócz ewidentnych zalet zastosowania VPD musimy wziąć pod uwagę pewne ograniczenia z niego wypływające. VPD działa na zasadzie przepisania czy też przemodelowania zapytania, wpływając w ten sposób na plan jego wykonania a w konsekwencji na wydajność takiego zapytania (często, bowiem wymagana jest ponowna analiza składniowa i semantyczna takiego zapytania). Jeżeli w naszej bazie używamy replikacji, wówczas użytkownik przeprowadzający taką replikację musi mieć nieograniczony dostęp do tabel źródłowych, w przeciwnym razie replikowana zostanie jedynie część danych. Używają tzw. ścieżek bezpośrednich przy wykonywaniu eksportu lub importu danych, musimy pamiętać, aby dezaktywować VPD na podległych obiektach. Eksport ścieżką bezpośrednią (direct path export) omija warstwę VPD, dlatego też opcja **DIRECT=Y** jest ignorowana i eksport wykonywany jest ścieżką konwencjonalną.

5.2. Szyfrowanie danych

W bazie danych Oracle mamy do dyspozycji dwie metody szyfrowania danych. Pierwsza z nich oparta jest o dostępne pakiety programistyczne, jako API do wykorzystania we własnych aplikacjach. Mowa tu o pakiecie **dbms_obfuscation_toolkit**, który dostępny jest już od wersji Oracle 8, a który oferuje szyfrowanie w oparciu o DES (z max kluczem 56 bitów) i DES3 (Oracle 9) oraz pakiet **dbms_crypto**, który pojawił się w wersji 10g i który wspiera m.in. AES, MD5 czy SHA-1. Drugie rozwiązanie to wykorzystanie istniejących funkcjonalności wbudowanych w silnik bazy danych takich jak TDE (*transparent data encryption*) czy TTE (*transparent tablespace encryption*). O ile w przypadku pakietów użytkownik dysponuje większą elastycznością w zakresie konfiguracji i szyfrowania, o tyle jednak więcej wysiłku potrzeba przy implementacji i zarządzaniu taką infrastrukturą. W tym rozdziale skupię się jedynie na drugiej z tych funkcjonalności.

Baza danych Oracle oferuje mechanizmy ochrony danych takie jak autentykacja, autoryzacja czy audyt jednak dotychczas brakowało w niej narzędzi zabezpieczających pliki danych znajdujących się na poziomie systemu operacyjnego. Wystarczyło chociażby użyć narzędzia **STRINGS**, aby uzyskać swobodny dostęp do jawnych informacji zawartych w takich plikach. Trwało to aż do czasu wprowadzenia TDE (*transparent data encryption*). TDE umożliwia proste i szybkie szyfrowanie poufnych danych bez potrzeby dodatkowych wysiłków związanych z zarządzaniem całą infrastrukturą szyfrowania. Wystarczy jedynie utworzyć repozytorium (zwane dalej wallet), aby móc transparentnie szyfrować i odszyfrowywać dane. Pojedynczy klucz szyfrujący jest użyty bez względu na liczbę szyfrowanych kolumn. Klucze te są następnie szyfrowane przy użyciu klucza głównego (*master key*), który dla celów bezpieczeństwa znajduje się w zewnętrznym wallecie. W celu użycia TDE niezbędnym jest uprawnienie **ALTER SYSTEM** oraz hasło do repozytorium. Jeżeli nie posiadamy jeszcze repozytorium (czyli nie posiadamy master key), tworzymy je następująco:

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY password
```

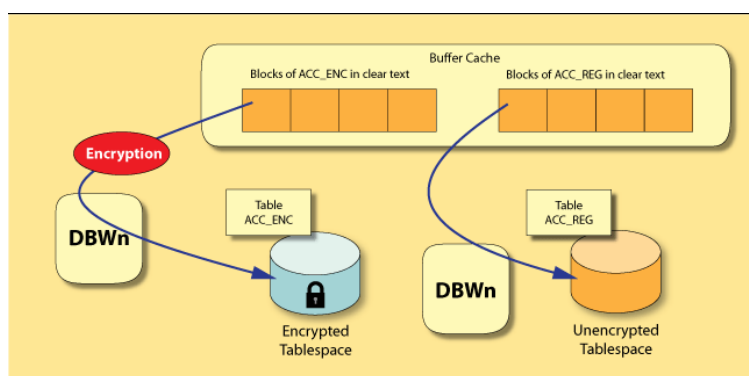
Dokonać tego można również za pomocą narzędzia **mkwallet** jak i przy pomocy Oracle Wallet Manger'a. Poleceniem tym tworzymy klucz główny, który będzie odtąd służył do szyfrowania naszych danych. Klucz ten stworzony zostanie w domyślnym repozytorium (chyba, że inaczej ustawiliśmy parametr **ENCRYPTION_WALLET_LOCATION** w pliku `sqlnet.ora`) i pozostanie do naszej dyspozycji aż do czasu zatrzymania bazy. Po ponownym uruchomieniu bazy, aby móc ponownie korzystać z TDE, czy innymi słowy, mieć dostęp do uprzednio zaszyfrowanych danych, musimy otworzyć nasz wallet:

```
ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY password
```

Pomimo licznych zalet i korzyści wynikających z zastosowania tej technologii, należy również wspomnieć o zagrożeniach i ograniczeniach wynikających z jej użycia. Nie wszystkie kolumny mogą być w jednakowym stopniu szyfrowane jak np. typu BLOB. Indeksy na takich kolumnach również nie są wydajne, w szczególności zapytania typu LIKE, które w normalnych okolicznościach używają skanowania zakresowego indeksów (*range scan*) to w przypadku szyfrowania wymuszają zastosowanie skanowania całych tabel (*full table scan*). Szyfrowanie danych jest operacją dodatkowo obciążającą jednostkę CPU, dlatego też należy znaleźć kompromis między ilością zaszyfrowanych danych a akceptowalnym spadkiem wydajności. Krytyczną sprawą jest również zarządzanie głównym kluczem szyfrującym oraz jego hasłem dostępu. Jeżeli zgubimy taki klucz dane automatycznie stają się niedostępne, również te znajdujące się w kopiach zapasowych. Problem stanowią mogą też tzw. kopie-duchy (ang. *ghost copies*) związane z naszymi danymi, a znajdujące się w formie jawnej w plikach danych. Są one wynikiem licznych operacji na danych, które mają miejsce w ciągu całego cyklu życia pliku danych (jako wynik fragmentacji, sortowań, kopiowania czy przenoszenia danych w obrębie przestrzeni tabel). Jedynie ostatnia, aktualna wersja danych jest trzymana w nich w formie zaszyfrowanej. Wówczas przy pomocy zwykłych hex-edytatorów wartości 'ghost-copies' mogą zostać odczytane.

W Oracle 11g mamy dodatkowo do dyspozycji nową metodę szyfrowania danych a mianowicie *Transparent Tablespace Encryption* (zwaną dalej TTE, a będąca wariacją wspomnianej wcześniej TDE). Pomimo wielu cech wspólnych łączących obie te technologie TTE jest bardziej wydajną i dojrzałą metodą szyfrowania danych na poziomie całych przestrzeni tabel. Również i w tym wypadku ma zastosowanie master key, aczkolwiek nie jest on tożsamy z tym stosowanym w TDE.

Widok **V\$ENCRYPTED_TABLESPACES** pokazuje informacje na temat wszystkich zaszyfrowanych przestrzeni tabel. Również dane znajdujące się w segmentach tymczasowych (jako wynik operacji np. SORT albo JOIN), UNDO czy w redo logach są szyfrowane.



Rys. 1. Schemat działania TTE

W przypadku TTE bloki danych, które są zapisywane na dysk przez proces DBWR, są w locie szyfrowane. Odwrotnie dzieje się przy odczycie bloków danych z dysku, wówczas proces serwera odszyfrowuje dane. Operacje typu eksport ścieżką bezpośrednią, czy wszelkie bezpośrednie modyfikacje wykonują operacje szyfrowania w tle. Ważną cechą TTE jest fakt, że dane znajdujące się w buforach pamięci przechowywane są w postaci jawnej. Wszystkie metody dostępu do danych

takie jak np. skanowanie indeksów, łączenia czy klucze obce mają miejsce na poziomie buforów danych, co oznacza, że nie ma tu żadnych ograniczeń funkcjonalnych czy też dodatkowego narzutu obliczeniowego w porównaniu z dostępem do danych niezakodowanych (plan wykonania zapytań nie zmienia się).

Dzięki temu możliwym staje się skanowanie zakresowe indeksów (w porównaniu ze skanowaniem pełnym redukuje nawet to 90% odczytów typu *consistent gets*). TTE zdejmuje również ograniczenie szyfrowania dla kluczy obcych czy indeksów innych niż B-tree. Jak pokazują testy, TDE ma również większe niż TTE zapotrzebowanie na przestrzeń dyskową, sięgające nawet do 50% (od 1 do nawet 52 bajtów dodatkowej przestrzeni – w tym 20 bajtów dla danych kontrolnych, do 16 bajtów dopełnienia oraz dodatkowe 16 bajtów dla opcji SALT). Ograniczeniem TTE jest to, że klucz główny nie może być redefiniowany, jak jest to możliwe przy TDE.

Mówiąc o szyfrowaniu nie sposób pominąć jego wpływu na wydajność bazy danych. A wpływ taki może być dość znaczny. Dostępne testy wydajności opierające się na czasie odpowiedzi operacji DML na przestrzeniach tabel skonfigurowanych dla algorytmów 3DES i AES prezentują spore przyrosty. Bazując na przykładzie zaczerpniętym z książki „*Oracle 11g New Features*” wydawnictwa Osborne – czasy odpowiedzi mogą być większe nawet o 30%. Załadowanie testowej tabeli 1752 tys. wierszy na zwykłej przestrzeni tabel trwało 21:30 minut, a przypadku 3DES już 28:30 minut oraz 28:54 w przypadku algorytmu AES. Dlatego też w pewnych sytuacjach lepszym rozwiązaniem może okazać się szyfrowanie jedynie odpowiednich kolumn z wrażliwymi danymi.

5.3. Zabezpieczenie backupu

Piętą achillesową zabezpieczeń środowisk bazodanowych często są kopie zapasowe. Do wersji Oracle 10g jedyną możliwością zabezpieczenia backupu (poza fizyczną kontrolą dostępu do nośników danych) było zastosowanie oprogramowania firm trzecich (lub np. Oracle Secure Backup). Od wersji 10g administratorzy mają do dyspozycji kilka możliwych sposobów rozwiązania tej kwestii. RMAN w wersji 10g dysponuje 3 możliwymi trybami szyfrowania dla kopii zapasowych - tryb transparentny, oparty o hasło albo tryb łączony.

Tryb transparentny bazuje na TDE i wymaga oczywiście dostępu do repozytorium, w którym znajduje się nasz klucz główny. Z tego powodu nadaje się on najlepiej do tworzenia kopii zapasowych na tym serwerze, na którym istnieje już wallet. Szyfrowanie aktywowane jest za pomocą polecenia:

```
RMAN> configure encryption for database on;
```

Jest to ustawienie na poziomie globalnym, jednakże dla indywidualnych przestrzeni tabel możemy włączyć je przy pomocy polecenia SET ENCRYPTION ON FOR TABLESPACE TEST. Dodatkowo można zmodyfikować algorytm szyfrowania:

```
RMAN> CONFIGURE ENCRYPTION ALGORITHM 'AES192';
```

Należy pamiętać, że korzystając z tego trybu, wallet cały czas musi być dostępny. W przeciwnym razie otrzymamy komunikat o błędzie:

```
ORA-19913: unable to decrypt backup
```

```
ORA-28365: wallet is not open
```

Metoda druga wymaga wydania dodatkowej dyrektywy, która ustala hasło dla kopii zapasowych. Metoda ta jest bardziej odpowiednia w przypadku, gdy kopie bezpieczeństwa chcemy wykonywać na innych serwerach, lub kopiować dane przez sieć w inne lokalizacje. Kopie takie zabezpieczone są jedynie podanym hasłem. Aby uaktywnić ją wykonujemy komendę:

```
RMAN> SET ENCRYPTION ON IDENTIFIED BY password ONLY;
```

Metoda łączona reprezentuje zalety obydwu wymienionych wyżej rozwiązań. W przypadku, gdy wallet jest otwarty, ma zastosowane TDE, w przeciwnym wypadku RMAN zastosuje podane

hasło. Dzięki temu tryb ten jest idealny do wykonywania kopii zapasowych, które następnie mogą być odtwarzane poza środowiskiem źródłowym (np. poza serwerami firmowymi)

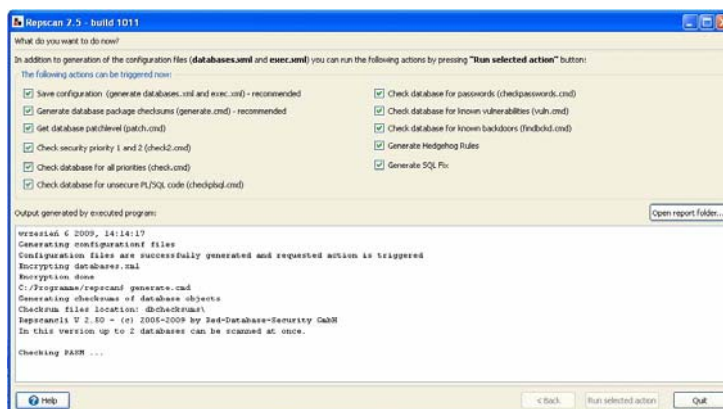
Należy wspomnieć także o kilku ograniczeniach związanych z stosowaniem TDE w Oracle 10g. Szyfrowanie w RMAN jest dostępne od wersji 10gR2 z poziomem COMPATIBLE ustawionym minimalnie na 10.2.x.x. Jedynie dane w tzw. backup set'ach podlegają szyfrowaniu, kopie typu *image* nie mogą być szyfrowane. Nawet po zmianie klucza głównego istnieje możliwość odtworzenia kopii bezpieczeństwa. Dla każdej operacji backupu generowany jest nowy klucz sesji. Klucz taki zakodowany jest przy użyciu albo klucza głównego bazy danych, podanego hasła albo obydwu naraz. W przypadku utraty hasła, bezpowrotnie tracimy możliwość przywrócenia takiego backupu (właściwość, co by nie mówić oczekiwana, w przypadku wykradzenia kopii zapasowych). Dodatkowy narzut obliczeniowy, związany z koniecznością zaszyfrowania danych, można próbować rozwiązać przez utworzenie dodatkowego kanału.

6. Narzędzia

Na rynku istnieje liczna grupa rozwiązań wspomagających administratorów baz danych zarówno w codziennej pracy jak i podczas przeprowadzanych audytów bezpieczeństwa. Opitz-Consulting, obok narzędzi takich jak Toad, SqlPlus, SQLDeveloper, Nagios, PASM czy Grid Control używa narzędzi do oceny istniejącego poziomu bezpieczeństwa oraz służących do zabezpieczenia środowisk bazodanowych. Do tej grupy należą m.in. RepScan oraz Hedgehog.

6.1. Repscan

RepScan jest narzędziem często wykorzystywanym jako narzędzie pierwszego kontaktu, czy pierwszej oceny poziomu bezpieczeństwa bazy danych. Może też być wykorzystany, jako narzędzie regularnych przeglądów systemu bezpieczeństwa.



Rys. 2. Interfejs RepScan

RepScan jest narzędziem stworzonym przez firmę Red Database Security, a obecnie dystrybuowany przez firmę Sentrigo i OPITZ CONSULTING. RepScan jest zaawansowanym skanerem bezpieczeństwa dedykowanym dla baz danych Oracle. W tej chwili RepScan (wersja 2) przeprowadza około 3000 różnego rodzaju analiz, co stawia go w ścisłej czołówce tego typu rozwiązań na świecie. Wersja 3 z poszerzonym zakresem analiz będzie już w krotce dostępna.

Z jego pomocą możemy zidentyfikować m.in. słabe i domyślne hasła (również w komponentach takich jak APEX, OAS, OID), niezabezpieczone luki oraz istniejące backdoory, brakujące łąty w bazie danych czy modyfikacje na jej obiektach (przy pomocy SHA2 checksum). Posiada również moduł do przeszukiwania istniejących obiektów pod kątem wrażliwych danych (hasła czy numery kart kredytowych). RepScan jest narzędziem, które dowolnie możemy konfigurować oraz

dostosowywać do własnych potrzeb np. przez tworzenie własnych pluginy (PL/SQL oraz C++) do analizy haseł w naszych aplikacjach. Poza tym jest prosty w instalacji i obsłudze oraz niezwykle szybki w działaniu.

In PASM the following rules violations were found:

Test	Priority	Handle	Description
CPU April 2009 is missing. Last patch: N/A	Critical	PATCH044	Apply CPU April 2009 Security Patches for 11.1.0.7 (Unix/Linux: 8290478)
[1] APEX installations vulnerable	High	APEX001	Vulnerable APEX installation found. Please remove outdated APEX schema.
Payload	High	BACKD0069	*. Please run/check the backdoor report for additional information.
DBMS_XMLGEN granted to PUBLIC	High	CONF151	The package DBMS_XMLGEN is granted to PUBLIC. This package allows users to run SQL statements against multiple tables in one command.

In PASM the following weak passwords in database were found:

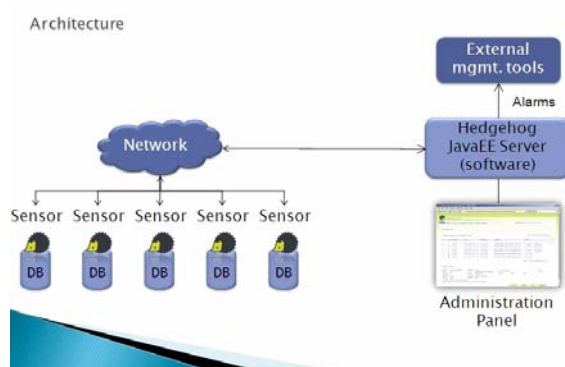
User name	Password	Status	Type	PW Type
APEX_PUBLIC_USER	*** WEAK ***	Open	Oracle SHA1	shared password
DBSNMP	*** WEAK ***	Open	Oracle SHA1	shared password
SYS	*** WEAK ***	Open	Oracle SHA1	shared password
SYSMAN	*** WEAK ***	Open	Oracle SHA1	shared password
ATHLONAM	*** WEAK ***	Open	Oracle DES	years at the end

Rys. 3. Przykład raportu końcowego RepScan'a

Raporty z wykonanych działań przedstawiane są w formie XML, i jak widać na powyższych grafikach są bardzo czytelne. Ciekawą funkcjonalnością jest również tworzenie poleceń SQL, które pomagają w zabezpieczeniu lub likwidacji zidentyfikowanych zagrożeń. Ścisła integracja z narzędziem Hedgehog umożliwia automatyczne generowanie zbioru reguł, które następnie będą monitorowane przez tą aplikację.

6.2. Hedgehog

Hedgehog izraelskiej firmy Sentrigo jest narzędziem typu IDS (*Intrusion Detection System*) przeznaczonym dla baz danych (od Oracle 8.1.7, od MS SQL-Server 2000, SyBase). Jego głównym zadaniem jest nie tylko monitorowanie aktywności baz danych, pod kątem ataków lub pojawiających się nadużyć, ale również przeciwdziałanie ewentualnym zagrożeniom.



Rys. 4. Architektura Hedgehog

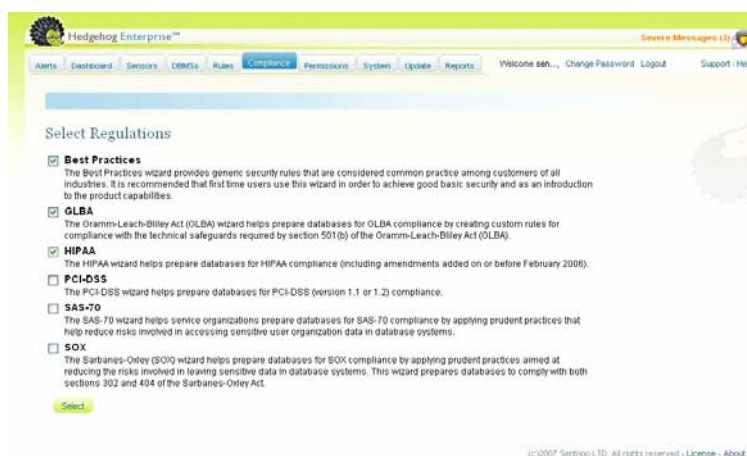
Hedgehog składa się z 2 głównych komponentów, mianowicie z serwera i sensorów. Server to aplikacja sieciowa napisana w Javie EE, wraz z wewnętrzną bazą danych (może zostać migrowana do Oracle lub MS-SQL Servera). Serwer może działać pod kontrolą systemu Windows, Solaris, AIX, HP-UX i oczywiście Linuxa. Sensory to procesy/ agenci działające po stronie monitorowanych baz danych (konstrukcją przypominające Intelligent Agent w GC) i komunikujące się z głównym serwerem przy pomocy streamingu XML po SSL. Serwer, który może działać również jako klaster, może jednocześnie zarządzać kilkunastoma sensorami. Zgodnie z danymi producenta, aplikacja ta generuje zwiększone o 5% zapotrzebowanie na moc obliczeniową.

Ciekawym jest, że sensory analizują bazę na poziomie struktur pamięci (w przypadku Oracle na poziomie Shared Memory). Nie ma, więc potrzeby tworzenia dedykowanego użytkownika w bazie danych i, co więcej, generowania dodatkowego ruchu po stronie serwera. Sensory te na bieżąco analizują aktywność bazy danych, pod kątem m.in. uruchamianych poleceń, łączących się użytkowników czy aplikacji i klasyfikują te zachowania jako bezpieczne lub nie w oparciu o dostępne reguły. Zachowania uznane, jako podejrzane są dokładniej analizowane. W razie wykrycia zdarzenia, które spełnia warunki danej reguły, generowany jest alarm oraz wywoływana jest odpowiednia reakcja (zerwanie sesji czy przesunięcie użytkownika do kwarantanny).

Sensory działają w oparciu o reguły. Hedgehog dostarczany jest z kilkoma grupami reguł, a każda z takich reguł składa się z zdarzenia wyzwalającego (*trigger*), które pociąga za sobą pewną akcję (*action*). Hedgehog składa się z czterech warstw reguł:

- *vPatch* – czy też wirtualny patching, są to reguły opierające się na rozpoznanych i opublikowanych lukach w systemach zabezpieczeń (*Vulnerability in package CTXSYS*, *Sensitive access to External Job Component*), wzorcach i sposobach ataków (*SQL Injection* przy użyciu *DBMS_METADATA*, *Buffer Overflow in RESTORE function*) czy np. analizie zachowań podejrzanych aplikacji (*Assessment tool detection /Checkpwd*). Reguły te są wbudowane i nie można ich zmieniać czy dopasować.
- *compliance template* – szablony gotowych reguł obejmujących zdarzenia wymagane czy też zalecane do rejestracji przez regulacje takie jak HIPAA, GLBA, PCI-DSS czy SOX
- *Community Best Practices* – zbiór reguł dostarczony przez firmę Sentrigo, oparty na doświadczeniu i sugestiach doświadczonych użytkowników i ekspertów

Reguły użytkownika – reguły stworzone na potrzeby danego środowiska czy aplikacji przez użytkownika (np. Monitoruj operacje DML wobec tabeli CREDIT_CARD)



Rys. 5. Hedgehog. Reguły wbudowane

Ciekawą funkcjonalnością jest *vPatch*, który de facto, zabezpiecza naszą bazę danych bez konieczności instalowania niezbędnych Oracle CPU czy innych zalecanych łatek (skądinąd wielce zalecane działanie). Biorąc pod uwagę jego możliwości, oraz czas, jaki często upływa od momentu wykrycia luki w systemie zabezpieczeń do opublikowania oficjalnego patcha, warto zainwestować w takie rozwiązanie, które bez dodatkowych wysiłków (testy, przestoje, certyfikacje dla środowisk itd.) zabezpiecza naszą bazę. Przykładowo, już 48 godzin od publikacji CPU April 2009, firma Sentrigo udostępniła reguły *vPatch* dla wszystkich luk i zagrożeń w nim się znajdujących. Tak, więc w 48 godzin od godziny ‘zero’ dysponujemy działającym i zabezpieczonym systemem. Produktem nieco podobnym w działaniu do omówionego tu HH jest Oracle Audit Vault. Służy on jednak głównie, jako centralne repozytorium, przechowujące dane z audytów z podległych baz

danych. Nie ma tu możliwości definiowania reguł czy reakcji na wykryte zagrożenia, nie wspominając o wirtualnym patchowaniu. OAV jest zresztą narzędziem wymagającym większego nakładu sił przy instalacji, konfiguracji i zarządzaniu.

7. Zakończenie

Project Lockdown jest jedną z licznych prób stworzenia uniwersalnego schematu postępowania zmierzającego do budowy i utrzymania bezpiecznego środowiska bazodanowego. Postępowanie jednak takie musi być zbudowane na i mocno związane z istniejącą polityką bezpieczeństwa. Kluczowym elementem w każdej polityce bezpieczeństwa jest zrozumienie wymagań stojących przed systemem bezpieczeństwa. Aby zastosowane środki bezpieczeństwa realizowały swoją funkcję ochronną, należy najpierw zrozumieć istniejące zagrożenia, potencjalnych napastników, motywacje, którymi się kierują, ich cele oraz możliwe metody użyte do osiągnięcia tych celów. Polityka bezpieczeństwa powinna być, bowiem inna w przypadku banku, firmy farmaceutycznej czy sklepu internetowego. Bez dokładnego zrozumienia takich zagrożeń i znalezienia odpowiedzi na takie pytania, jak – co i dlaczego powinniśmy chronić, może się okazać, że użyte środki i zabezpieczenia mają zastosowanie do zagrożeń, które de facto albo nie istnieją, albo nie mogą być zniwelowane przy ich pomocy. Zabezpieczenie bazy danych przed szkodliwym działaniem użytkowników wewnętrznych wymaga podjęcia zupełnie innych kroków aniżeli próba sprostania wymaganiom stawianym przez regulacje kontrolne takie jak HIPPA czy SOX. Zanim, więc oficjalnie stwierdzimy, że szyfrowanie danych jest najlepszym sposobem ochrony istniejącego środowiska w ramach naszej polityki bezpieczeństwa, zadajmy sobie pytanie czy na pewno go potrzebujemy, jak również, jakie dane i przed kim powinny być chronione.

Bibliografia

- Alapati S.: OCP Oracle Database 11g. New Features for Administrators Exam Guide, Oracle Press
- Finningan P.: Oracle Security Masterclass, OUGF Conference 2009, May 14'th 2009
- Freeman R.: Oracle Database 11g. New Features, Oracle Press
- Knox D.: Effective Oracle Database 10g Security by Design, Oracle Press
- Kornbrust A.: Oracle Security Technology Day: Oracle Security – Trends und Ausblick für 2009
- Litchfield D.: The Oracle Hacker's Handbook: Hacking and Defending Oracle, John Wiley & Sons 2007
- Oracle Database Security Checklist, an Oracle White Paper, June 2008
- SANS Institute: Oracle Database Security Checklist
- Sentriego: Practical Guide to Database Security & Compliance
- Shaul J, Ingram A.: Practical Oracle Security, Syngress
- Nanda A.: Project Lockdown. A phased approach to securing your database infrastructure, OTN, May 2008
- Nanda A.: Encrypting Tablespace, Oracle Magazine January/ February 2009