BEZPIECZEŃSTWO BAZ DANYCH (CZ. 2)

ORACLE ADVANCED SECURITY – OCHRONA PRZED ESKALACJĄ UPRAWNIEŃ KLASY ENTERPRISE

W poprzednim artykule („IT Professional" 7/2015) opisane zostały techniki zabezpieczania przed eskalacją uprawnień poprzez użycie narzędzia Database Vault, za pomocą którego stworzyliśmy REALM, który nie pozwalał na dostęp do danych nawet użytkownikowi z rolą SYSDBA. W kolejnym artykule z cyklu przyglądamy się możliwościom szyfrowania całych przestrzeni tabel za pomocą opcji Oracle Advanced Security.

Kamil Stawiarski

Oracle Database Vault jest narzędziem zapewniającym zaawansowaną ochronę wrażliwych danych aplikacji przed nieautoryzowanym działaniem osób postronnych, wynikającym z eskalacji uprawnień lub pozyskania dostępu za pomocą metod socjotechnicznych do konta bazodanowego z wysokimi uprawnieniami (np. do roli SYSDBA). Z punktu widzenia aplikacji implementacja tego mechanizmu jest w większości przypadków transparentnym rozwiązaniem, pozwalającym na budowanie chronionych obszarów (tzw. REALM), bez konieczności czasochłonnego i kosztownego modyfikowania kodu źródłowego systemu.

> PODSTAWOWE CECHY DATABASE VAULT

Database Vault zapewnia blokadę dostępu do wrażliwych danych poprzez interfejs bazodanowy nawet w przypadku nieuprawnionego dostępu do systemu operacyjnego. Dzięki temu ochrania on również przed skutkami potencjalnych błędów, jakie mogą zostać popełnione przez początkujących administratorów. Opisywane rozwiązanie wbudowane jest w binaria bazy danych w wersji Enterprise Edition, a jego wdrożenie jest stosunkowo proste, niezależnie od wykorzystywanej wersji bazy danych czy systemu operacyjnego.

Cechą charakterystyczną mechanizmu jest fakt, że nawet użytkownik z rolą SYSDBA nie ma możliwości bezpośredniego dostępu do obiektów, chronionych REALM-em:

```
[oracle@rico ~]$ sqlplus sys@hrdb as sysdba
SQL*Plus: Release 12.1.0.2.0 Production on Tue Aug 11 13:44:52 2015
Copyright (c) 1982, 2014, Oracle.  All rights reserved.
Enter password:
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics,
Oracle Database Vault and Real Application Testing options

SQL> select * from hr.employees;
select * from hr.employees
                 *
ERROR at line 1:
ORA-01031: insufficient privileges

SQL> show user
USER is "SYS"
SQL>
```

Implementacja mechanizmu Database Vault automatycznie chroni nas również przed technikami eskalacji uprawnień, które zostały opisane w scenariuszu przedstawionym w jednym z poprzednich artykułów („IT Professional" 5/2015, s. 46), polegającym na użyciu mechanizmu PREPROCESSOR obiektu EXTERNAL TABLE:

```
SQL> select * from x$_cmd;
select * from x$_cmd
```

```
*
ERROR at line 1:
ORA-29913: error in executing ODCIEXTTABLEOPEN callout
ORA-29400: data cartridge error
KUP-04094: preprocessing cannot be performed if Database Vault is installed
```

Jest to jednak prawdą jedynie dla użytkowników nieposiadających roli SYSDBA. Użytkownik z rolą SYSDBA nie zobaczy powyższego komunikatu i będzie mógł nawet swobodnie skompilować kawałek dowolnego kodu (napisanego np. w C++):

```
SQL> select * from x$_cmd;
 TXT
--------------------------------------------------------------------------------
g++         /tmp/read_data.cpp    -o /tmp/read_data
```

> NIEAUTORYZOWANY DOSTĘP DO DANYCH

Możemy łatwo zauważyć, że dalsza eskalacja uprawnień oraz pozyskanie nieautoryzowanego dostępu do zabezpieczonych danych nie są proste do realizacji, a próby nadużyć wymagają rozpoczęcia pracy już z bardzo wysokim poziomem uprawnień, zapewniającym – w ten czy inny sposób – możliwość wykonywania poleceń systemu operacyjnego jako użytkownik należący do odpowiednich grup autoryzacji dla binariów bazy danych Oracle.

Załóżmy jednak, że atak następuje z wewnątrz przedsiębiorstwa i jeden z pracowników lub kontrahentów, mających możliwość wywołania odpowiednich poleceń OS z poziomu bądź to SSH, bądź poprzez mechanizmy bazodanowe, chce uzyskać dostęp do danych wrażliwych. Celem takiego ataku będą bezpośrednio binarne pliki danych, składające się na przestrzenie tabel, w których znajdują się faktyczne tabele. Każdy obiekt bazodanowy posiada swój unikalny identyfikator, który możemy odczytać z perspektywy DBA_OBJECTS:

```
  1  select data_object_id
  2  from dba_objects
  3  where object_name='EMPLOYEES'
  4* and owner='HR'
SQL> /

DATA_OBJECT_ID
--------------
         91753
```

Możemy również sprawdzić, w jakim pliku danych, znajdującym się na systemie plików, tabela EMPLOYEES zawierająca wrażliwe dane pracowników przechowuje swoje bloki danych:

```
  1  select file_name
  2  from dba_data_files d
  3  where exists (select 1
  4                from dba_extents e
  5                where e.file_id=d.file_id
  6                and   e.segment_name='EMPLOYEES'
  7*               and   e.owner='HR')
SQL> /

FILE_NAME
--------------------------------------------------------------------------------
/u01/app/oracle/oradata/rico/hrdb/example01.dbf
```

Każdy plik danych jest złożony z bloków, których struktura (w uproszczeniu) składa się z nagłówka oraz faktycznych danych. Znając dokładną binarną strukturę bloku i mając podstawową wiedzę z zakresu programowania, można stworzyć kod np. w C++, który będzie w stanie dostać się bezpośrednio do bloków danych i sczytać z nich dane. Wstępna postać takiego kodu mogłaby wyglądać następująco:

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

main (int argc, char * argv[])
{
  string fileName;
  cout << "Enter file path: " ;
        cin >> fileName;
  ifstream datafile(fileName.c_str(), ios::binary);
  int numBlocks=0;
  if (datafile.is_open())
  {
    int objIdOffset=24;
    int blockSize=8192;
    int numOfRowsOffset=54;
    int ktbbhictOffset=36;
    int objIdSearch;
    cout << "What data_object_id: ";
    cin >> objIdSearch;
    //objIdSearch=91753;
    int i=0;
    while(!datafile.eof())
    {
     datafile.seekg(i*blockSize);
     char blockType[1];
     int  blockTypeI;
     datafile.read(blockType,1);
     blockTypeI=(int) blockType[0];
     if(blockTypeI==6)
     {
       datafile.seekg(i*blockSize+objIdOffset);

       char objId[4];
```

```
      int  objIdI;
      datafile.read(objId,4);
      memcpy(&objIdI,objId,sizeof(objIdI));
      if(objIdI==objIdSearch)
      {
        char ktbbhictCount[1];
        int  ktbbhictCountI;
        datafile.seekg(i*blockSize+ktbbhictOffset);
        datafile.read(ktbbhictCount,1);
        ktbbhictCountI=(int) ktbbhictCount[0];
        char numOfRows[1];
        int  numOfRowsI;
        datafile.seekg(i*blockSize+numOfRowsOffset+24*ktbbhictCountI);
        datafile.read(numOfRows,1);
        numOfRowsI=(int) numOfRows[0];
        if (numOfRowsI>0)
        {
          numBlocks++;
          cout << "Block number: " << i << "  \
          num of rows: " << numOfRowsI << endl;
        }
      }
    }
    i++;
  }
 }
 datafile.close();
 cout << numBlocks << " blocks found " << endl;
}
```

Powyższy kod skanuje wskazany plik danych pod kątem bloków zawierających wiersze obiektu o wskazanym DATA_OBJECT_ID:

```
[oracle@rico ~]$ ./read_data
Enter file path: /u01/app/oracle/oradata/rico/hrdb/example01.dbf
What data_object_id: 91753
Block number: 206  num of rows: 98
Block number: 207  num of rows: 9
2 blocks found
```

Z poziomu użytkownika HR możemy łatwo sprawdzić, że informacja jest prawidłowa:

```
SQL> conn hr/hr@hrdb
Connected.
SQL> select dbms_rowid.rowid_block_number(rowid), count(1)
  2  from employees
  3  group by dbms_rowid.rowid_block_number(rowid);

DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)   COUNT(1)
---------------------------------- ----------
                               207          9
                               206         98
```

Nietrudno więc wyobrazić sobie rozszerzenie funkcjonalności kodu, które pozwoli na podejrzenie danych w tabeli. Wystarczy uzbroić się w wiedzę z zakresu dokładnej budowy bloku Oracle, spędzić trochę czasu na analizie binarnej zawartości bloku i odkryć metodę lokalizacji offsetów poszczególnych wierszy, binarnej budowy wiersza oraz algorytmów składowania poszczególnych typów danych.

Z wiadomych przyczyn nie opublikujemy całego kodu, który na to pozwala – program taki byłby bronią obosieczną, gdyż z jednej strony pozwalałby na odzyskanie danych z uszkodzonych plików, ale z drugiej byłby narzędziem pozyskiwania danych z bazy bez pozostawiania śladów w narzędziach audytowych oraz z pominięciem mechanizmów takich jak Oracle Database Vault.

Jaki jest więc sposób na zabezpieczenie się przed odczytem danych bezpośrednio z pliku? Naturalnie jest to szyfrowanie plików danych.

> OCHRONA ORACLE ADVANCED SECURITY

Możliwość zaszyfrowania całych przestrzeni tabel pojawiła się wraz z opcją Oracle Advanced Security, którą można dokupić do bazy Enterprise Edition. Opis całego pakietu można znaleźć w dokumentacji Oracle dostępnej na stronie tinyurl.com/OAS-doku.

W przykładowym algorytmie skupimy się na funkcjonalności TDE (Transparent Data Encryption), a konkretnie na tablespace encryption. Ogólna zasada szyfrowania przestrzeni tabel jest prosta – na poziomie bazy danych generujemy tak zwany Master Encryption Key, który służy do szyfrowania i deszyfrowania kluczy TDE Tablespace Encryption Key wykorzystywanych do szyfrowania i deszyfrowania danych w bloku należącym do tej przestrzeni tabel (patrz **schemat obok**).

STWORZENIE „PORTFELA”

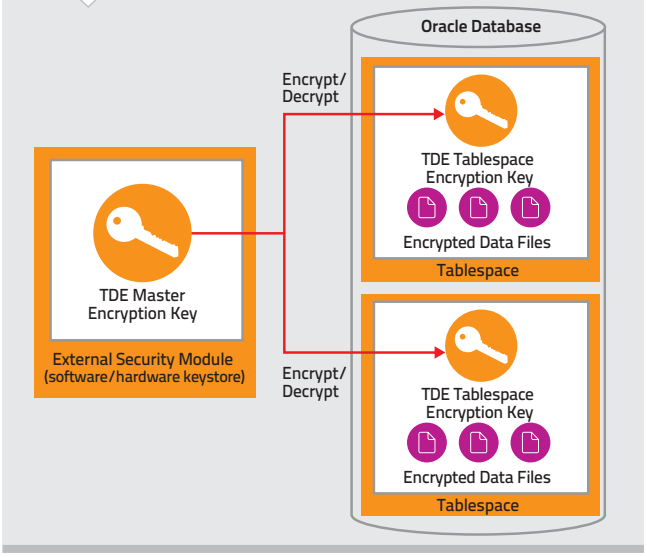
Pierwszym krokiem dla wygenerowania zaszyfrowanej przestrzeni tabel jest stworzenie pliku, w którym będzie przechowywany Master Encryption Key. W pierwszym kroku musimy zdefiniować ścieżkę do pliku wallet w pliku sqlnet.ora znajdującym się w lokalizacji $ORACLE_HOME/network/admin:

```
[root@rico ~]# mkdir -p /etc/oracle/wallet/rico
[root@rico ~]# chown -R oracle:oinstall /etc/oracle/
[root@rico ~]# su - oracle
[oracle@rico ~]$ cd $ORACLE_HOME/network/admin
[oracle@rico admin]$ vim sqlnet.ora
[oracle@rico admin]$ cat sqlnet.ora
# sqlnet.ora Network Configuration File: \
  /u01/app/oracle/product/12.1.0/dbhome_1/network/admin/sqlnet.ora
# Generated by Oracle configuration tools.

NAMES.DIRECTORY_PATH= (TNSNAMES, EZCONNECT)

ENCRYPTION_WALLET_LOCATION=
```

OGÓLNA ZASADA SZYFROWANIA PRZESTRZENI TABEL




```
(SOURCE=
 (METHOD=FILE)
  (METHOD_DATA=
   (DIRECTORY=/etc/oracle/wallet/rico)))
```

Następnie tworzymy plik odpowiedzialny za przechowywanie kluczy, czyli tzw. wallet:

```
SQL> administer key management create keystore '/etc/oracle/wallet/rico' \
identified by ZyrafyWchodzaDoSzafy;

keystore altered.
```

```
SQL> show user
USER is "SYS"
SQL> administer key management set keystore open identified by ZyrafyWchodzaDoSzafy;

keystore altered.
```

Po wykonaniu tej operacji wallet zostanie stworzony we wskazanej lokalizacji oraz otworzony przez instancję.

```
SQL> !ls /etc/oracle/wallet/rico
ewallet.p12

SQL> desc V$ENCRYPTION_WALLET
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 WRL_TYPE                                           VARCHAR2(20)
 WRL_PARAMETER                                      VARCHAR2(4000)
 STATUS                                             VARCHAR2(30)
 WALLET_TYPE                                        VARCHAR2(20)
 WALLET_ORDER                                       VARCHAR2(9)
 FULLY_BACKED_UP                                    VARCHAR2(9)
 CON_ID                                             NUMBER

SQL> select WRL_PARAMETER, status from V$ENCRYPTION_WALLET;

WRL_PARAMETER
--------------------------------------------------------------------------------
STATUS
------------------------------
/etc/oracle/wallet/rico/
OPEN_NO_MASTER_KEY
```

Plik wallet nie zawiera jeszcze żadnego klucza, na co wskazuje status OPEN_NO_MASTER_KEY.

Następnym krokiem jest więc stworzenie klucza typu MASTER:

```
SQL> administer key management set key identified by "ZyrafyWchodzaDoSzafy" with backup;

keystore altered.

SQL> select WRL_PARAMETER, status from V$ENCRYPTION_WALLET;

WRL_PARAMETER
--------------------------------------------------------------------------------
STATUS
------------------------------
/etc/oracle/wallet/rico/
OPEN
```

ZASZYFROWANIE PRZESTRZENI TABEL

Po wykonaniu powyższych operacji jesteśmy gotowi do stworzenia zaszyfrowanej przestrzeni tabel, do której można będzie przenieść tabele zawierające wrażliwe dane. Do wyboru mamy następujące algorytmy szyfrowania:

- 3DES168
- AES128
- AES192
- AES256

Stworzenie zaszyfrowanej przestrzeni tabel jest bardzo proste:

```
SQL> create tablespace secure_data
  2  datafile '/u01/app/oracle/oradata/rico/hrdb/secure_data01.dbf'
  3  size 128m
  4  autoextend on next 64m
  5  maxsize 1g
  6  encryption using 'AES256'
  7  default storage (encrypt);

Tablespace created.
```

Bloki wszystkich obiektów znajdujących się w tej przestrzeni tabel zostaną zaszyfrowane, a ich odczytanie będzie możliwe tylko i wyłącznie wtedy, gdy wallet będzie otworzony – w przeciwnym wypadku otrzymamy komunikat:

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

Oczywiście utrata pliku wallet nie będzie należeć do najszczęśliwszych chwil w życiu administratora bazy danych Oracle. Dlatego zdecydowanie należy uwzględnić lokalizację tego pliku w polityce backupu.

UWAGA NA AMM

Obiekty znajdujące się w zaszyfrowanej przestrzeni tabel nie będą mogły być już celem ataków z poziomu systemu plików. Próba dostania się do bloku danych skończy się oglądaniem kalejdoskopu nic nieznaczących bajtów. Bez pliku wallet otworzonego przez bazę danych pliki przestrzeni tabel jako takie są bezużyteczne.

Należy jednak pamiętać o pewnej cesze – blok danych jest deszyfrowany w momencie wykonywania operacji odczytu przez bazę danych i jeśli zostanie zbuforowany w buffer cache, znajduje się tam już w postaci jawnej. Jeśli nasza baza danych działa w systemie operacyjnym Linux i mamy uruchomiony mechanizm AMM (Automatic Memory Management), zawartość pamięci współdzielonej (SGA) jest reprezentowana przez pliki na urządzeniu /dev/shm. W takim przypadku znowu pojawia się możliwość przechwycenia wrażliwych danych z poziomu systemu operacyjnego poprzez lokalizację odpowiednich plików pamięci i skanowanie ich wspomnianym już narzędziem napisanym w C++:

```
[oracle@rico shm]$ ls ora_rico_346* | awk '{system("strings " $0 " | grep Steven && echo " $0)}'
Steven
Steven
ora_rico_346619910_62
Steven
Steven
ora_rico_346619910_68
[oracle@rico shm]$ ~/read_data
Enter file path: /dev/shm/ora_rico_346619910_62
What data_object_id: 92723
Block 877: 9 rows (3 ITL slots)
Block 878: 98 rows (3 ITL slots)
2 blocks found
```

Z tego też powodu sugerujemy rezygnację z AMM na rzecz ASMM, jeśli chcemy mieć pewność zabezpieczenia przed tego typu nadużyciem z poziomu systemu operacyjnego.

Autor jest posiadaczem tytułu Oracle Certified Master, zaproszonym do programu Oracle ACE.

ORACLE ADVANCED SECURITY A „REKOMENDACJA D"

W 2013 roku Komisja Nadzoru Finansowego stworzyła dokument o nazwie „Rekomendacja D", który dotyczył zarządzania obszarami technologii informacyjnej i bezpieczeństwa środowiska teleinformatycznego w bankach. Dokument ten składa się z 22 rekomendacji – produkt Oracle Advanced Security pokrywa następujące obszary:

- 9.13 – ochrona poufności informacji przekazywanej,
- 9.14 – ochrona wycofywanych komponentów,
- 10.6 – szyfrowanie poufnych danych,
- 15.19 – zabezpieczenie kopii awaryjnych.