HMIN328 : TP Oracle et sécurité

1. Sécurité, identification usager, droits et chiffrement

1.1 Exercice 1

1.2 Question 1

Vous consulterez la vue dba_users pour constater que vous avez effectivement accès à l'image MD5 de chaque mot de passe utilisateur. Quels sont les dispositifs qui pourraient mettre à mal la protection associée à cette clé de hachage ? Quels sont les moyens pour limiter le risque de "piratage" des mots de passe ?

1.3 Question 2

Quelles sont les vues à consulter pour connaître les utilisateurs qui disposent du plus grand nombre de droits sur la base de données ? Proposer des requêtes adaptées à la consultation de ces droits (nom utilisateur, catégories de droits) ?

1.4 Question 3

Expliquer l'intérêt de créer un rôle et d'attribuer les droits associés à un ce rôle à un ensemble d'usagers. Commentez la séquence suivante

```
create role distribue;
grant create session, select any dictionary to distribue;
grant create synonym to distribue;
grant create public database link to distribue;
grant drop public database link to distribue;
grant distribue to public;
```

1.5 Question 4

Expliquer la requête suivante :

```
select SYS_CONTEXT('USERENV','ISDBA') isdba from dual;
```

1.6 Question 5

Deux paquetages PL/SQL nommés dbms_obfuscation_toolkit (pour le plus ancien) et dbms_crypto fournissent différentes fonctionnalités pour le chiffrement. Nous utilisons ici dbms_obfuscation_toolkit. Un exemple permettant de disposer de clés MD5 vous est donné et vous vous approprierez cet exemple en construisant une procédure qui renvoie un clé de hachage pour chaque nom de région (table région).

```
CREATE or REPLACE FUNCTION set_md5(v_in VARCHAR2)
   RETURN VARCHAR2
   IS
   result VARCHAR2(40);
   BEGIN
```

```
    result := UTL_RAW.CAST_TO_RAW(dbms_obfuscation_toolkit.md5(input_string=>v_in));
    RETURN result;
    END;
    /

select set_md5_md5('admirable') from dual;
select length(md5('admirable')) from dual;
```

1.7 Question 6

Le paquetage dbms_obfuscation_toolkit offre aussi des fonctionnalités liées à des méthodes de chiffrement symétriques à l'exemple de la méthode associée à l'algorithme DES (Data Encryption Standard) qui consiste à effectuer des combinaisons, substitutions et permutations entre le texte à chiffrer et une clé (fournie également en entrée), en faisant en sorte que les opérations puissent se faire dans les deux sens (chiffrement et déchiffrement). Les fonctions dbms_obfuscation_toolkit.DESEncrypt et dbms_obfuscation_toolkit.DESDecrypt sont exploitées dans le paquetage donné dans le fichier Securite_dbms_obf.sql.
Vous ajouterez une colonne nommée president à la table Region et vous ferez en sorte de crypter l'information associée au nom de chaque président de région. Vous utiliserez cette information pour l'afficher en clair et sous forme cryptée (construire une procédure PL/SQL et un curseur).

2. Sécurité et injection SQL

2.1 Question 1

Une requête élémentaire qui restitue tous les tuples d'une table vous est donnée. Cette requête est une illustration des principes de l'injection SQL.

```
SELECT username, password FROM dba_users
WHERE username ='HR'
OR '1'='1';
```

Le recours à des variables d'environnement (SQL ou bindParam pour PHP), à des procédures stockées sécurisées (PL/SQL) ou des requêtes paramétrées (PreparedStatement en Java) sont des moyens simples et performants de limiter les effets de l'injection SQL. Un exemple d'utilisation de variable d'environnement vous est donné ci-dessous.

```
variable reg varchar2(10);
execute :reg := '91';
SELECT cheflieu FROM region WHERE region = :reg;
```

Vous exploiterez la variable d'environnement reg pour lui attacher une autre valeur (par. ex la valeur de la région RHONE-ALPES) et ré-interpréter la requête. Dans quel autre contexte, avons nous déjà abordé les variables d'environnement et pourquoi ?

2.2 Question 2

Un exemple de procédure PL/SQL exploite un curseur explicite (faible) est donné
(repris de http ://www.oratechinfo.co.uk/sql_injection.html)

```
 CREATE OR REPLACE PROCEDURE show_unimportant_data(p_code IN VARCHAR2)
    AS
      c1   SYS_REFCURSOR;
      l_ncc  VARCHAR2(50);
      l_cheflieu VARCHAR2(50);
    BEGIN
```

```
      OPEN c1 FOR 'SELECT ncc, cheflieu FROM region WHERE region = ''' || p_code || '''';
      LOOP
        FETCH c1 INTO l_ncc, l_cheflieu;
       EXIT WHEN c1%NOTFOUND;
       dbms_output.put_line('Region : ' || l_ncc || ' : Nom : ' || l_cheflieu);
     END LOOP;
     CLOSE c1;
   END show_unimportant_data;
   /
-- utilisation classique
EXEC show_unimportant_data('91');

-- utilisation malicieuse
drop table credit_cards;
CREATE TABLE credit_cards ( card_num    VARCHAR2(50),  pin_num  NUMBER(4) );
INSERT INTO credit_cards VALUES ( '000-000-000', '1234' );
INSERT INTO credit_cards VALUES ( '000-000-001', '2345' );
INSERT INTO credit_cards VALUES ( '000-000-002', '2346' );

BEGIN
    show_unimportant_data('91'' UNION SELECT card_num, TO_CHAR(pin_num) FROM credit_cards ' ||
                                      'WHERE ''1''=''1');
    END;
  /
```

Vous utiliserez cet exemple pour construire votre propre exemple de procédure et d'utilisation malicieuse de procédure qui renvoie le nom et le mot de passe (crypté) des utilisateurs de la base de données.

2.3 Question 3

Vous exploiterez le code ci-dessous pour construire une procédure exploitant les variables d'environnement (et donc robuste face à de l'injection SQL) qui prendra en argument un code de région et qui en retournera toutes les noms de département associés.

```
declare
empcur pls_integer;
sqlStmt varchar2(32767);
fonction varchar2(20) :='commercial';
nom varchar2(20);
v_execute pls_integer;
begin
sqlStmt := 'SELECT nom FROM emp WHERE fonction = :fonction';
empcur := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(empcur, sqlStmt, DBMS_SQL.NATIVE);
DBMS_SQL.BIND_VARIABLE(empcur, ':fonction', fonction);
DBMS_SQL.DEFINE_COLUMN_CHAR(empcur, 1, nom, 20);
v_execute := DBMS_SQL.EXECUTE(empcur);
loop
     if DBMS_SQL.FETCH_ROWS(empcur) = 0 then
        exit;
     end if;
     DBMS_SQL.COLUMN_VALUE_CHAR(empcur, 1, nom);
     DBMS_OUTPUT.PUT_LINE('Emp name: '||nom);
   end loop;
   DBMS_SQL.CLOSE_CURSOR(empcur);
 EXCEPTION
```

```
   when others then
        DBMS_SQL.CLOSE_CURSOR(empcur);
        raise_application_error(-20000, 'Unknown Exception Raised: '||sqlcode||' '||sqlerrm);
end;
/
```

2.4 Question 4

L'archive Java WebGoat vous est donnée (sous forme d'une archive Java), vous l'exploiterez pour comprendre quelques uns des mécanismes de l'injection SQL.

3. Audit avec un déclencheur sur l'ensemble de la BD

Cet exemple nécessite des droits administrateurs.

```
create table logon_events (user_l varchar(30), session_id  varchar(30), osuser varchar(30),
host varchar(30), ipaddress varchar(30), prg varchar(30), sysd date);


CREATE OR REPLACE TRIGGER LOGON_EMPLOYEE
AFTER LOGON ON DATABASE
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
IF ( sys_context('userenv','sessionid') != 0 ) then
INSERT INTO logon_events
( SELECT USER,sys_context('userenv','sessionid'),
sys_context('userenv','os_user'),
sys_context('userenv','host'),
sys_context('userenv','ip_address'),program,sysdate
FROM  v$session
WHERE AUDSID = sys_context('userenv','sessionid') );
COMMIT;
END IF;
END;
/

-- connexion de hr et une ligne dans la table
```