PeteFinnigan.com Limited
Oracle Security

Secure Coding (PL/SQL)

Securely coding Applications in PL/SQL

PeteFinnigan.com Limited
Oracle Security

Legal Notice

Oracle Database Security Presentation

Published by
PeteFinnigan.com Limited
9 Beech Grove
Acomb
York
England, YO26 5LD

Agenda

- History
- Common attacks on PL/SQL
- Example Hack! – keep in real
- Secure coding in PL/SQL
- Protecting PL/SQL
- Adding license features in PL/SQL

The Problem Space

- Secure coding in PLSQL
  - Manifested in insecure existing code
  - Insecure continuing development practices
  - Often code can provide an easy access to attackers
  - Either remotely (via web or forms based applications)
  - Or locally via database users exploiting poor code
- Coding Security features in PL/SQL
  - Problem squared (**problem*problem**)
  - If you code security features (VPD, OLS, Encryption, Password Functions, Application security....) you must secure this code
    - Secure coding
    - Plus security controls
    - **Code protection, stop theft, running, reading**

http://www.oracle.com/us/support/assurance/coding/index.html

History

- Oracles alerts and CPU's have been littered with PL/SQL bugs
- Oracle started to fix their bugs
- Oracle test their code (Fuzz, static analysis, manual audit)
- Oracle train their developers in secure coding
- http://www.petefinnigan.com/weblog/archives/00001153.htm
- DBMS_ASSERT used extensively plus binds for dynamic code with database objects (not "objects" but tables etc)

What About Customer Code?

- Oracle have fixed hundreds (more?) PL/SQL bugs
- They have training, tools, testing, standards and more
- BUT usually we have not!
- We are 10 years behind Oracle in PL/SQL secure coding
- Most likely
  - We have simple security bugs not found in Oracle code now
  - We use dangerous interfaces
  - We don’t check/audit/test our code for security issues
  - We create open DML/DDL/SQL interfaces
- Not good!

Common Problems (1)

- Injection is the most famous (SQL, PL/SQL, Javascript..)
- Not a web phenomenon as some think
- Just as easy in SQL*Plus, in fact easier
- I wrote the first papers in 2003 (http://www.petefinnigan.com/orasec.htm)
- Three main modes, in-band, out of band, inference
- Order of attack, first, second, third, more...
- Possible because of concatenation
- Input from parameters, database, even session
- Inject SQL, DDL, functions, cursor injection, snarfing

Common Problems (2)

- We write code that accesses the filesystem
- We write code that accesses the networking
- We use dangerous packages – jobs, scheduler...
- We integrate with C or java
- We leak data
  - Passwords – hard coded ALTER USER...IDENTIFIED BY...
  - Networking
  - Encryption keys

Common Problems (3)

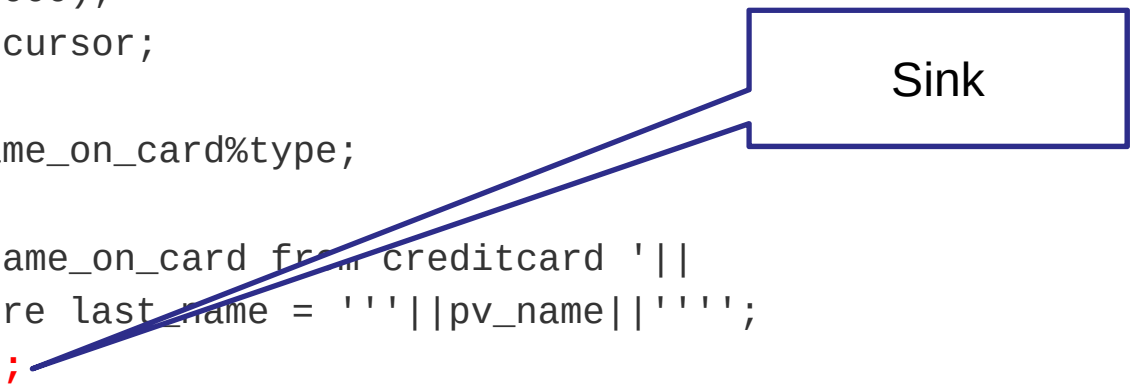
- PL/SQL must also be protected (theft, running, reading)
- Privileges must be controlled
- **Access to the schema means all bets off**
- Package could be intercepted and parameters stolen
- Definer rights code is dangerous as it runs as the owner
- Invoker rights is not totally safe
- Test access rights with my scripts; who_can.. who_has..

Demo

- Show creditcard
- Show decryption function
- Show cannot be accessed by orauser, describe
- Desc orablog.cust
- Exec cust('Finnigan)
- Exec cust(')
- Exec cust('x'' union select username from all_users--')
- Exec cust('x'' union select name_on_card||ccdec(pan) from orablog.creditcard--')
- **We can exec a function not allowed and also read data not allowed – any access point in a schema can be used to read any data in that schema**
- **This example is different to normal exploits that "grant dba to..."**

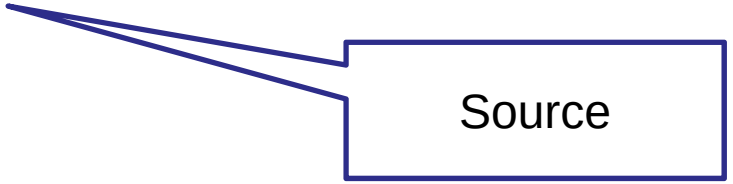
Finding Security Issues - sink

```
create or replace procedure cust(pv_name in varchar2) is
    lv_stmt varchar2(2000);
    type c_ref is ref cursor;
    c c_ref;
    name creditcard.name_on_card%type;
Begin
    lv_stmt:='select name_on_card from creditcard '||
                  'where last_name = '''||pv_name||'''';
    open c for lv_stmt;
    loop
         fetch c into name;
         if(c%notfound) then
                  exit;
         end if;
         dbms_output.put_line('name:=['||name||']');
    end loop;
    close c;
end;
/
```



Finding Security Issues - Source

```
create or replace procedure cust(pv_name in varchar2) is
    lv_stmt varchar2(2000);
    type c_ref is ref cursor;
    c c_ref;
    name creditcard.name_on_card%type;
Begin
    lv_stmt:='select name_on_card from creditcard '||
                  'where last_name = '''||pv_name||'''';
    open c for lv_stmt;
    loop
         fetch c into name;
         if(c%notfound) then
                  exit;
         end if;
         dbms_output.put_line('name:=['||name||']');
    end loop;
    close c;
end;
/
```



Finding Security Issues - Problem

```
create or replace procedure cust(pv_name in varchar2) is
    lv_stmt varchar2(2000);
    type c_ref is ref cursor;
    c c_ref;
    name creditcard.name_on_card%type;
Begin
    lv_stmt:='select name_on_card from creditcard '||
                  'where last_name = '''||pv_name||'''';
    open c for lv_stmt;
    loop
         fetch c into name;
         if(c%notfound) then
                  exit;
         end if;
         dbms_output.put_line('name:=['||name
    end loop;
    close c;
end;
/
```

Problem, need to follow data from source to sink plus check for filters, assignments and more; plus the problem ('||') is in the string assignment not the sink

Wider Issue

- The “source” is a wider issue as we need to understand who can execute the code – who_can_access.sql
- We need to know which packages/views etc use the vulnerable code – dep.sql + who_can_access.sql
- The bigger issue is definer rights code – select authid from dba_procedures;
- Definer rights code means we can exploit any other code in that schema (i.e. Run it) and access any data in that schema

Reviewing Code

- We can use new_code_a.sql to find sinks
  - Execute immediate
  - Dbms_sql
  - Dbms_sys_sql
  - Open for
- We can use new_code.sql to find “problems”
  - strings, concat(), || etc
- SQL>@new_code ‘||’
- Can limit to a single schema
- Focus on definer rights code

Reviewing Code (2)

There are other sources besides parameters!!! For instance SQL

Does not show private func/proc

- Find sources – start with same packages/schema

```
SQL> select object_name,package_name,argument_name from dba_arguments
  2  where data_type='VARCHAR2' and owner='ORABLOG';

OBJECT_NAME                    PACKAGE_NAME                   ARGUMENT_NAME
------------------------------ ------------------------------ ------------------
DECRYPT                        ORABLOG_CRYPTO
ENCRYPT                        ORABLOG_CRYPTO                 CC
MONTHNAME
DAYNAME
CUST                                                          PV_NAME
CHAR_LENGTH                                                   IN_CHAR
CCEN                                                          CC
CCDEC

8 rows selected.

SQL>
```

Tainted data!

Reviewing Code (3)

- The hard part is mapping sources in packages/procedures/functions to concatenated strings and then sink points
- Data has to be “flowed” from entrance to variable to variable to concat statement to sink
- Further analysis is then needed when a vulnerable source is found
  - Who can access that package/procedure/function
  - Who can change the table sourced data
  - Who can change the Session sourced data
- Proper flow analysis is needed – Fortify et-al are options

Secure Coding Practice

- Identify vulnerable code – see above!
- Fix all occurrences not just those located
- Define secure coding standards – Oracle, Feuerstein, O'Reilly
- Train your developers in your standards
- Don't use ||, concat(), do use dbms_assert, filter (white list not black list)
- Use bind variables where possible
- Manually check code – code review
- Simple SQL like my new_code.sql and new_code_a.sql

Secure Coding – Cont'd

- Professional tools – expensive
- Fuzzing – dangerous - http://www.slaviks-blog.com/wp-content/uploads/2009/01/fuzzor.sql
- Don't use dangerous packages
- Don't access the OS, network
- Don't hard code data such as passwords and keys
- Ensure that access is limited to the code source
- Ensure run time access is limited
- A whole schema must be secure otherwise its not secure

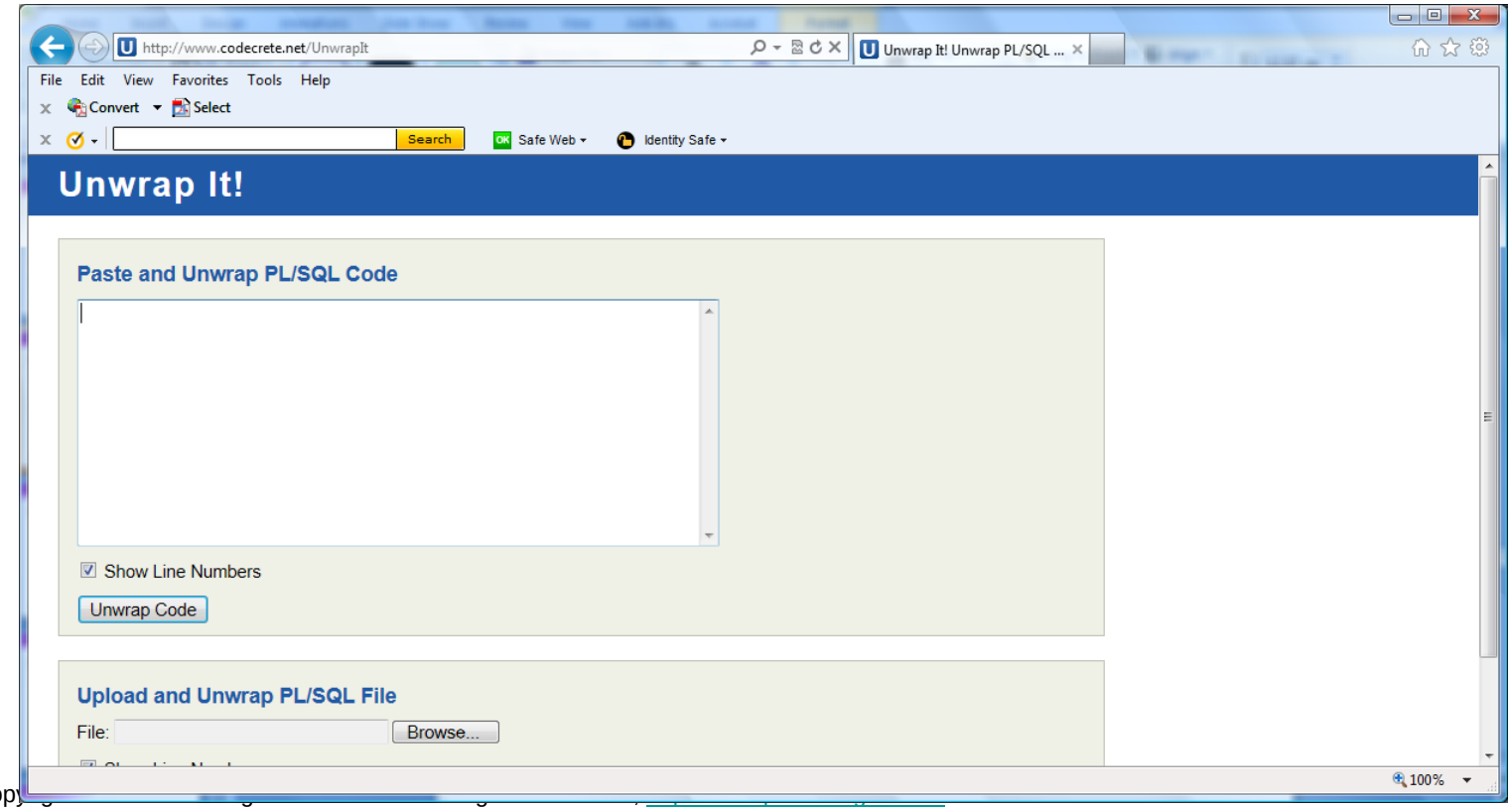
Protecting PL/SQL

- There are two issues to solve:
  - Stopping understanding of IPR or theft of IPR
  - Stopping code being stolen and run elsewhere
- The problem with database code is anyone can read it
- The problem with database code is that anyone can steal it and try and run it elsewhere
- Solutions therefore should stop:
  - Reading
  - Theft and/or un-authorised running
- Implies
  - Solution to remove meaning – minimal solutions available
  - Licensing type features – none available

Should protect our code?

Oracles Wrap 10g >

- We can use wrap.exe
- It can be unwrapped - http://www.codecrete.net/UnwrapIt



Oracle Wrap pre 10g

```
SQL> @unwrap_c

unwrap_c: Release 1.4.0.0.0 - Production on Wed Oct 10 09:01:35 2012
Copyright (c) 2004 - 2012 PeteFinnigan.com Limited. All rights reserved.

NAME OF OBJECT TO CHECK                 [P1]: TEST_PROC1
OWNER OF OBJECT TO CHECK               [SYS]:
TYPE OF THE OBJECT               [PROCEDURE]:
OUTPUT METHOD Screen/File                [S]:
FILE NAME FOR OUTPUT              [priv.lst]:
OUTPUT DIRECTORY [DIRECTORY  or file (/tmp)]:

create or replace procedure TEST_PROC1( PV_NUM in NUMBER,
 PV_VAR in VARCHAR2, PV_VAR3 in out INTEGER) is
 L_NUM NUMBER:=3;
 L_VAR NUMBER;
 J NUMBER:=1;
 LV VARCHAR2(32767);
procedure NESTED( PV_LEN in out NUMBER) is
 X NUMBER;
begin
```

Oracle Wrap

- 10g and higher wrap is not good, algorithm is weak – Unix compress, base64 and look up
- Oracle9i wrap is harder to unwrap
- https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Finnigan.pdf
- Unwrappers are available
  - rewrap
  - unwrap10
  - softdream
  - online sites
  - http://sourceforge.net/projects/plsqlunwrapper/
  - Plus many private ones but ,mostly 10g are available not 9i

Protect IPR

- So what can we do?
- Wrap code
- Obfuscate code
- Add license type features
- Add tamperproofing
- Add watermarks or birthmarks
- We can have code check itself!
- More?

Obfuscation

- Obfuscate the PL/SQL code
  - PFCLObfuscate (http://www.pfclobfuscate.com/2012/04/welcome-to-pfclobfuscate/) – compact, character sets, length, comment removal, controls, string obfuscation, scripting, code insert, hide packages, much more features...
  - http://krisrice.blogspot.co.uk/2012/02/sql-developer-31-and-obfuscation.html - SQL Developer - simple variable obfuscation, base64 binary values, long
  - Semantic Designs – PL/SQL obfuscator, obfuscates variables, compact, comment removal - http://www.semdesigns.com/products/obfuscators/PLSQLObfuscationExample.html

Stop Theft

- License features
  - Limit how code will run
- Tamperproofing
  - Detect if code has been modified
  - Checksum
  - Skype as an example
  - Watermarking
  - Uniquely identify all releases to detect who lost the code!

License Features

- License features could have many forms
- No one is doing this – except me?
- Types
  - Time/ date based
  - Place – DBID, DBNAME, Network adaptor, Server, hardware, number of CPU's...
  - Person based
  - Context based – where in code,
  - Privilege based/enabled
  - Combinations of course
  - i.e. Run on Tuesday between 6 and 8 pm when user is "FRED" and role "BLOB" is enabled and DB is PROD and ....

Tamperproofing

- We can use many techniques
- Checksums simplest; test canary values
- Code can checksum itself / cross check
- Stack based checks – code runs in right place

```
-- test rules here to ensure this is called from sqlexec code
owa_util.who_called_me(lv_owner,lv_name,lv_lineno,lv_caller_t);
dbms_output.put_line('owner ['||lv_owner||']');
dbms_output.put_line('name ['||lv_name||']');
dbms_output.put_line('lineno ['||lv_lineno||']');
dbms_output.put_line('caller_t ['||lv_caller_t||']');
if(lv_owner='XXEXEC' and lv_name='READ' and lv_lineno=5 and lv_caller_t='PROCEDURE') then
        dbms_session.set_role('secapp');
else
        raise_application_error(-20001, 'You are not authorised to connect.');
end if;
```

Security Solutions Implemented in PL/SQL

- The ultimate issue
- If a security solution is in PL/SQL
  - Password function
  - VPD predicate function
  - FGA handler function
  - Encryption ...
- We must use the techniques described
- Protect IPR, Tamperproof, Control permissions
- Protect Source code

Finally My Own Research

- Using unwrapping for good not bad
- Take your PL/SQL
- Add license code (currently manual – auto soon)
- Add tamper code
- Obfuscate to hide meaning
- Wrap with 9i – undoc param allows new SQL code, newer PL/SQL has to be dynamic or not protected
- Stops unwrappers working
- Most secure PL/SQL? – **I think so**
- **Risk: Support / optimisations? – use carefully?**

Conclusions

- PL/SQL can be exploited
- Learn to code securely
- Audit your own PL/SQL for weaknesses
- Protect your IPR
- Stop theft with license ideas
- Gather it all up in security features in PL/SQL

Questions?

Any Final Questions?

Secure Coding (PL/SQL)

Securely coding Applications in PL/SQL