How to unwrap PL/SQL

Pete Finnigan, Principal Consultant
Introduction

- My name is Pete Finnigan
  - I specialise in researching and securing Oracle databases
- The PL/SQL wrapping process has particularly interested me for some years
- I wanted to investigate why the method chosen to secure intellectual property written in PL/SQL is weak
- I also felt it was intriguing that Oracle has made it “easy” for anyone to understand how to recover source code in 9i and lower
- I also find it interesting that Oracle has shipped API’s since the beginning of PL/SQL that can be used to unwrap
The agenda

- Oracle’s PL/SQL language – a sample procedure
- How PL/SQL is wrapped, the language internals, the database tables and files used, the events that can reveal information
- Why it is possible to read wrapped code – driven by history and design choice!
- How it is possible to read wrapped code – some sample code shipped by Oracle
- The built in API’s shipped by Oracle
- 10g, the changes
- What can be done to protect PL/SQL source code
Why is there a problem with wrapped PL/SQL

- Intellectual property can be revealed if PL/SQL is unwrapped
- This can include
  - Your own application source code
  - Oracle shipped features hidden by the wrapper
- In 9i and lower wrapped PL/SQL revealed symbols
- Finding SQL injection bugs just became easier
- There are PL/SQL unwrapping tools available
PL/SQL language compilation structure

PL/SQL source code

PL/SQL compiler front end

DIANA – An intermediate language for ADA

Compiler back end

Virtual machine M-Code
DIANA is the key for 9i and lower

- PL/SQL is based on ADA
- DIANA – Descriptive intermediate language for ADA
  - DIANA is an abstract data type
  - DIANA is an intermediate form of PL/SQL programs
  - Intended to communicate between the front end and back ends of a PL/SQL compiler
  - Each defining DIANA entity represents a PL/SQL entity
- Two trees –
  - Abstract syntax tree constructed prior to semantic analysis
  - Attributed tree (the DIANA structure)
IDL – Interface description language

- DIANA is written down as IDL
- What is IDL? – Interface description language – Also derived from ADA
- IDL is stored in the database in 4 dictionary tables
  - IDL_CHAR$, IDL_SB4$, IDL_UB1$ and IDL_UB2$
- Wrapped PL/SQL is simply DIANA written down as IDL
- Oracle say that wrapped PL/SQL is simply encoded
- Therefore the *wrap* program is the front end of a PL/SQL compiler.
- Is wrapped PL/SQL – DIANA – reversible?
<table>
<thead>
<tr>
<th>DIANA – An Intermediate Language for ADA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editors: G. Goos, W.A. Wulf</td>
</tr>
<tr>
<td>A. Evans, Jr and K.J. Butler</td>
</tr>
<tr>
<td>Springer-Verlag</td>
</tr>
<tr>
<td>ISBN : 0387126953</td>
</tr>
<tr>
<td>Revised Edition (December 1983)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quote from page 165:</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Appendix III – Reconstructing the source”</td>
</tr>
<tr>
<td>“One of the basic principals of DIANA is that the structure of the original source program is to be retained in the DIANA representation…..”</td>
</tr>
<tr>
<td>“There is a close correspondence between ADA’s syntax and DIANA’s structural attributes… It is this correspondence that permits source code reconstruction.”</td>
</tr>
</tbody>
</table>
From Oracle’s own documentation

PL/SQL User's Guide and Reference
10g Release 1 (10.1)
Part Number B10807-01

“PL/SQL is based on ADA, as a result PL/SQL uses a variant of DIANA, a tree structured language....”

“It is defined using a meta notation called IDL (Interface Definition Language)....”

“DIANA is used internally by compilers and other tools.....”

“At compile time PL/SQL is translated into M-Code. Both DIANA and M-Code are stored in the database.....”
A Sample PL/SQL procedure – 9i

SQL> connect sys/change_on_install as sysdba
Connected.

SQL> create or replace procedure AA as
    2    begin
    3        null;
    4    end;
    5     /
Procedure created.
SQL>
Save the PL/SQL and wrap the code

SQL> save aa.sql
Created file aa.sql
SQL> exit
{output snipped}
G:\code> wrap iname=aa.sql oname=aa.pls


Copyright (c) Oracle Corporation 1993, 2001. All Rights Reserved.
Processing aa.sql to aa.pls

G:\code>
The wrapped output

create or replace procedure AA wrapped
  0
abcd
{snipped 15 identical lines}
3
7
9200000
1
4
0
1
2 :e:
1AA:
0

What is the meaning of this encoded file? –
Note the highlights – we will see them again
9i and below wrapped PL/SQL weaknesses

SQL> create or replace procedure encode (credit_card in varchar2, str out varchar2) is
2  key varchar2(16):='01234567890ABCDEF';
3  begin
4    null;
5  end;
6  /

Procedure created.

SQL> save encode.sql

{snipped}

G:\code>wrap iname=encode.sql oname=encode.plb


Copyright (c) Oracle Corporation 1993, 2001. All Rights Reserved.

Processing encode.sql to encode.plb
Hacking wrapped PL/SQL – pre-9i

- The symbol table is visible
- For the previous example it is possible to
  - Deduce the purpose of the procedure
  - Find out the encryption algorithm used using DBA_DEPENDENCIES unless it is implemented internally to the procedure
  - Decrypt Credit Cards – in this case
- Trojans can be planted
- Wrapped source can be modified without un-wrapping
  - Example: Fixed DBMS_OUTPUT limits problem
- SQL injection identification is possible / DDL can be altered
The relationships in 9i

PL/SQL Source

Wrap utility

Wrapped File

create or replace procedure aa
  wrapped
  0
  abcd
  abcd
  abcd
  abcd
  Abcd
  ...

Diana and M-Code
IDL_CHAR$
IDL_UB1$
IDL_UB2$
IDL_SB4$

Source Code
SOURCE$
DBA_SOURCE
ALL_SOURCE
The dictionary tables and views

- **SYS.IDL_CHAR$**
- **SYS.IDL_UB1$**
- **SYS.IDL_UB2$**
- **SYS.IDL_SB4$**
- **SYS.SOURCE$**

---

```sql
SQL> desc idl_ub1$
Name                      Null?    Type
-------------        -------- ----------
OBJ#                NOT NULL NUMBER
PART               NOT NULL NUMBER
VERSION             NUMBER
PIECE#             NOT NULL NUMBER
LENGTH              NOT NULL NUMBER
PIECE              NOT NULL LONG RAW
```

```sql
SQL> desc source$
Name Null? Type
--------------- -------- ----------
OBJ#            NOT NULL NUMBER
LINE            NOT NULL NUMBER
SOURCE          VARCHAR2(4000)
```

---

From $OH/rdbms/admin/sql.bsq

```sql
/* part: 0 = diana, 1 = portable pcode, 2 = machine-dependent pcode */
```
Recursive SQL

- What is recursive SQL? – background supporting SQL needed to execute the submitted statement

- When compiling PL/SQL there are other background SQL statements that need to run as SYS
  - Check for user’s privileges and roles
  - Triggers
  - Retrieving the PL/SQL code to run
  - Indexes

- How can we see the complete picture?
- Using traces, dumps and events
Trace the compilation of PL/SQL

SQL> alter session set events '10046 trace name context forever, level 12';

Session altered.

SQL> create or replace procedure aa is
  2  begin
  3    null;
  4  end;
  5  /

Procedure created.

SQL> alter session set events '10046 trace name context off';

Session altered.

SQL>
Locate the trace file and check the contents

PARSING IN CURSOR #2 len=106 dep=1 uid=0 oct=6 lid=0 tim=465432930704 hv=1545875908 ad='66f37b44'

update idl_ub2$ set piece#:1 ,length=:2 , piece=:3 where obj#:4 and part=:5 and piece#:6 and version=:7
END OF STMT
PARSE #2:c=0,e=42,p=0,cr=0, cu=0,mis=0,r=0,dep=1,og=4,tim=465432930696
BINDS #2:
  bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=08 oacfl2=1 size=24 offset=0
    bfp=04822394 bln=24 avl=02 flg=05
    value=4
bind 1: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=08 oacfl2=1 size=24 offset=0
    bfp=04822364 bln=24 avl=03 flg=05
    value=123
bind 2: dty=25 mxl=4000(4000) mal=00 scl=00 pre=00 oacflg=12 oacfl2=1 size=4000 offset=0
    bfp=04c67ff4 bln=4000 avl=246 flg=09
    value=

Dump of memory from 0x04C67FF4 to 0x04C680EA
4C67FF0 00030000 000D000C 00250011 [..........%.]
4C68000 002A0029 0038002C 003E003A 00000040 [).*.8...>.@...]
4C68010 001D0017 009A0068 00B40055 001100B5 [....h...U.......]
4C68020 00A400B1 004F00B7 00010000 00010001 [.....O........]
4C68030 00010001 00010001 00010001 00010001 [..................]
4C68040 00000001 00010001 000B0001 00010001 [..................]

Those numbers look familiar!
SQL> select count(*), 'CHAR$', part, object_type
  2    from idl_char$, dba_objects
  3    where obj# = object_id
  4    and part = 0
  5    group by part, object_type
  6  union
  7  select count(*), 'UB1$', part, object_type
  8    from idl_ub1$, dba_objects
  9    where obj# = object_id
 10    and part = 0
 11  group by part, object_type
 12  union
 13  select count(*), 'UB2$', part, object_type
 14    from idl_ub2$, dba_objects
 15    where obj# = object_id
 16    and part = 0
 17  group by part, object_type
 18  union
 19  select count(*), 'SB4$', part, object_type
 20    from idl_sb4$, dba_objects
 21    where obj# = object_id
 22    and part = 0
 23  group by part, object_type
 24 order by 2

SQL> /

<table>
<thead>
<tr>
<th>COUNT(*)</th>
<th>'CHAR$'</th>
<th>PART</th>
<th>OBJECT_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------</td>
<td>---------</td>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>OPERATOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>PROCEDURE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>TYPE BODY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>SEQUENCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>LIBRARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>FUNCTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>VIEW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>PACKAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>SYNONYM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>TYPE</td>
</tr>
</tbody>
</table>

SQL> /
What IDL was created for procedure ‘AA’?

```sql
SQL> select dbms_rowid.rowid_block_number(rowid) blk,
      2  dbms_rowid.rowid_relative_fno(rowid) fno,
      3  dbms_rowid.rowid_row_number(rowid) rnum,
      4  'CHAR$',part,version,piece#,length
      5  from idl_char$
      6  where obj#=(select obj# from obj$ where name = 'AA')
      7  union
      8  select dbms_rowid.rowid_block_number(rowid) blk,
      9  dbms_rowid.rowid_relative_fno(rowid) fno,
     10  dbms_rowid.rowid_row_number(rowid) rnum,
     11  'UB2$',part,version,piece#,length
     12  from idl_ub2$
     13  where obj#=(select obj# from obj$ where name = 'AA')
     14  union
     15  select dbms_rowid.rowid_block_number(rowid) blk,
     16  dbms_rowid.rowid_relative_fno(rowid) fno,
     17  dbms_rowid.rowid_row_number(rowid) rnum,
     18  'ub1$',part,version,piece#,length
     19  from idl_ub1$
     20  where obj#=(select obj# from obj$ where name = 'AA')
     21  union
     22  select dbms_rowid.rowid_block_number(rowid) blk,
     23  dbms_rowid.rowid_relative_fno(rowid) fno,
     24  dbms_rowid.rowid_row_number(rowid) rnum,
     25  'sb4$',part,version,piece#,length
     26  from idl_sb4$
     27  where obj#=(select obj# from obj$ where name = 'AA')
     28  order by part,piece#
SQL> save rowid.sql
```

<table>
<thead>
<tr>
<th>BLK</th>
<th>FNO</th>
<th>RNUM</th>
<th>CHAR</th>
<th>PART</th>
<th>VERSION</th>
<th>PIECE#</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>49951</td>
<td>1</td>
<td>24</td>
<td>sb4$</td>
<td>0</td>
<td>153092096</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>49951</td>
<td>1</td>
<td>48</td>
<td>sb4$</td>
<td>0</td>
<td>153092096</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>42671</td>
<td>1</td>
<td>21</td>
<td>ub1$</td>
<td>0</td>
<td>153092096</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>35792</td>
<td>1</td>
<td>36</td>
<td>CHAR$</td>
<td>0</td>
<td>153092096</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>50581</td>
<td>1</td>
<td>8</td>
<td>UB2$</td>
<td>0</td>
<td>153092096</td>
<td>4</td>
<td>123</td>
</tr>
<tr>
<td>50581</td>
<td>1</td>
<td>9</td>
<td>UB2$</td>
<td>0</td>
<td>153092096</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>49951</td>
<td>1</td>
<td>50</td>
<td>sb4$</td>
<td>2</td>
<td>153092096</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>42671</td>
<td>1</td>
<td>10</td>
<td>ub1$</td>
<td>2</td>
<td>153092096</td>
<td>1</td>
<td>112</td>
</tr>
<tr>
<td>42671</td>
<td>1</td>
<td>13</td>
<td>ub1$</td>
<td>2</td>
<td>153092096</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

9 rows selected.
Dump the datablocks to find the DIANA

- Why do we need to dump datablocks for the IDL$ tables?

SQL> select piece
    2   from sys.idl_ub2$
    3   where obj#=(select obj# from obj$ where name='AA')
    4   and part=0
    5   and piece#=4;

ERROR:
ORA-00932: inconsistent datatypes: expected %s got %s

no rows selected

SQL> alter system dump datafile 1 block 50581;

System altered.

Instead the data must be dumped from the datafile

The contents of the IDL$ tables cannot be selected
### The contents of the block dump for IDL_UB2$

**tab 0, row 8, @0x11b1**

**tl: 271 fb: --H-FL-- lb: 0x1 cc: 6**

| col 0: | 4 | c3 04 05 0a |
| col 1: | 1 | 80         |
| col 2: | 6 | c5 02 36 0a 15 61 |
| col 3: | 2 | c1 05      |
| col 4: | 3 | c2 02 18   |
| col 5: | 246 | 00 00 03 00 0c 00 0d 00 11 00 25 00 29 00 2a 00 2c 00 38 00 3a 00 3e 00 40 |
|       |   | 00 00 17 00 1d 00 68 00 9a 00 55 00 b4 00 b5 00 11 00 b1 00 a4 00 b7 00 4f |

Those values look familiar but in a different order.
### IDL dependencies – (a detour)

```sql
SQL> select distinct owner, name, type
  2      from dba_dependencies
  3 where referenced_name like 'IDL_%'
SQL> /

<table>
<thead>
<tr>
<th>OWNE</th>
<th>NAME</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>ALL_PROBE_OBJECTS</td>
<td>VIEW</td>
</tr>
<tr>
<td>SYS</td>
<td>CODE_PIECES</td>
<td>VIEW</td>
</tr>
<tr>
<td>SYS</td>
<td>INITJVMAUX</td>
<td>PACKAGE BODY</td>
</tr>
<tr>
<td>SYS</td>
<td>ORA_KGLR7_IDL_CHAR</td>
<td>VIEW</td>
</tr>
<tr>
<td>SYS</td>
<td>ORA_KGLR7_IDL_SB4</td>
<td>VIEW</td>
</tr>
<tr>
<td>SYS</td>
<td>ORA_KGLR7_IDL_UB1</td>
<td>VIEW</td>
</tr>
<tr>
<td>SYS</td>
<td>ORA_KGLR7_IDL_UB2</td>
<td>VIEW</td>
</tr>
<tr>
<td>SYS</td>
<td>PARSED_PIECES</td>
<td>VIEW</td>
</tr>
<tr>
<td>SYS</td>
<td>RMJVM</td>
<td>PACKAGE BODY</td>
</tr>
</tbody>
</table>
```
How are IDL tables used?

```sql
SQL> desc code_pieces
Name       Null?    Type
------------ -------- -----
OBJ#        NUMBER   
BYTES       NUMBER   

SQL> set long 1000000
SQL> select text from dba_views
    2 where view_name='CODEPieces'

SQL> /
TEXT

select i.obj#, i.length
from sys.idl_ub1$ i
where i.part in (1,2)
union all
select i.obj#, i.length
from sys.idl_ub2$ i
where i.part in (1,2)
union all
select i.obj#, i.length
from sys.idl_sb4$ i
where i.part in (1,2)
union all
select i.obj#, i.length
from sys.idl_char$ i
where i.part in (1,2)
```
The DIANA and IDL API packages

```
SQL> select text from dba_source
2  where name='PIDL';

package     PIDL is
---------------------------------------------------------------------
-- Persistent IDL datatypes
---------------------------------------------------------------------
  subtype ptnod is binary_integer; -- generic IDL node type
  TRENULL CONSTANT ptnod := 0; -- a NULL node
  subtype ub4 is binary_integer; -- Oracle C type, unsigned byte 4
  subtype ub2 is binary_integer; -- Oracle C type, unsigned byte 2
{Output snipped to 550 lines}
```

```
SQL> select text from dba_source
2  where name='DIANA';

package     diana is
  D_ABORT   constant pidl.ptnty := 1;
  D_ACCEPT  constant pidl.ptnty := 2;
  D_ACCESS  constant pidl.ptnty := 3;
  D_ADDRES  constant pidl.ptnty := 4;
{output snipped to 1596 lines}
```

Source code available in $ORACLE_HOME/rdbms/admin/pipidl.sql and pidian.sql
**DIANA Utilities - $OH/rdbms/admin/diutil.sql**

```sql
SQL> desc diutil
PROCEDURE ATTRIBUTE_USE_STATISTICS
<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Type</th>
<th>In/Out</th>
<th>Default?</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBUNIT_NODE</td>
<td>BINARY_INTEGER</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE_COUNT</td>
<td>BINARY_INTEGER</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE_LIMIT</td>
<td>BINARY_INTEGER</td>
<td>OUT</td>
<td></td>
</tr>
</tbody>
</table>

PROCEDURE GET_D
<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Type</th>
<th>In/Out</th>
<th>Default?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>VARCHAR2</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>USR</td>
<td>VARCHAR2</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>DBNAME</td>
<td>VARCHAR2</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>DBOWNER</td>
<td>VARCHAR2</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>STATUS</td>
<td>BINARY_INTEGER</td>
<td>IN/OUT</td>
<td></td>
</tr>
<tr>
<td>NOD</td>
<td>BINARY_INTEGER</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>LIBUNIT_TYPE</td>
<td>NUMBER</td>
<td>IN</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>LOAD_SOURCE</td>
<td>NUMBER</td>
<td>IN</td>
<td>DEFAULT</td>
</tr>
</tbody>
</table>
```

{snipped}
Dumpdiana – a script to dump the DIANA

- `$ORACLE_HOME/rdbms/admin/dumpdian.sql`
- Not installed by default
- Run the script as SYS
- There are two bugs to fix – remove the lines REM ----
- Ensure DIANA, PIDL and DIUTIL PL/SQL packages are installed as well
- Run for sample ‘AA’ procedure as SYS – (output to trace) :-

  SQL> exec sys.dumpdiana.dump(aname => 'AA');

  PL/SQL procedure successfully completed.

  SQL>
A DIANA tree dump – (Goos/Wulf - pages 137 – 144)

Diana node

PD1(4): D_COMP_U [  
  L_SRCPOS : row 1 col 1  
  A_CONTEXT :  
]  

PD2(4): D_CONTEXT [  
  L_SRCPOS : row 1 col 1  
  AS_LIST : < >  
]  

PD3(4): D_S_BODY [  
  L_SRCPOS : row 1 col 1  
  A_UNIT_B :  
]  

PD4(4): DI_PROC [  
  L_SRCPOS : row 1 col 11  
  L_SYMREP : AA,  
  S_SPEC : PD5^^(4),  
  S_BODY : PD8^^(4),  
  S_LOCATI : 0,  
  S_STUB :  
  S_FIRST : PD4^^(4),  
  {output snipped}
]  

PD5(4): D_P_ [  
  L_SRCPOS : row 1 col 1  
  A_HEADER :  
]  

C_OFFSET : 0,  
C_FRAME_ : 255,  
C_ENTRY_ : 1,  
S_FRAME :  
A_UP : PD3^(4),  
S_LAYER : 1,  
L_RESTRICT_REFERENCES : 32,  
A_METH_FLAGS : 0,  
SS_PRAGM_L :  
S_INTRO_VERSION : 0,  
A_PARALLEL_SPEC :  
C_VT_INDEX : 0,  
C_ENTRY_PT : 1  

Code attrib

Structural attribute

Lexical attribute

Semantic attribute
Attributed structured tree

- This is the Block section
- The PD?(?) syntax can also be seen on page 151 of Goos / Wulf book
- Each node has variable number of attributes dependant on node type
- Some of which are nodes
- L_SRCPOS is mandatory for all DÏANA nodes – ADA included
- LX_COMMENT as well
Reconstructing PL/SQL source from DIANA - 1

- Block syntax for PL/SQL

```
Block_statement ::= [block_simple_name]
[declare
  declarative part]
begin
  sequence of statements
[exception
  exception handler {exception handler}]
end [block_simple_name] ;
```

- Diana Rules

```
block => as_item : DS_ITEM,
  as_stm : D_STMT,
  as_alter : DS.Alter;
```

- See page 166 – Goos / Wulf et al
An alternate DIANA dump

{output snipped}

```
PD3(4) : D_S_BODY: [  
    SRCPOS: row 1 col 1  
    A_D_: PD4(4) : DI_PROC: [...]  
    A_HEADER: PD5(4) : D_P_: [...]  
    A_BLOCK_: PD8(4) : D_BLOCK: [...]  
    A_UP: PD1(4) : <reference to D_COMP_U (262145)>  
]

PD4(4) : DI_PROC: [  
    SRCPOS: row 1 col 11  
    L_SYMREP: text: 'AA'  
{output snipped}
```

```
SQL> exec sys.dumpdiana.dump(aname => 'AA',print_format => 1);  
PL/SQL procedure successfully completed.  
SQL>
```
Goos / Wulf et al page 167

Declare

<DS_ITEM>

Begin

<DSSTM>

Exception

<DS_ALTER>

End;

It is easy to see the close relationship between PL/SQL and DIANA

Then it is easy to see how PL/SQL can be reconstructed from DIANA
Mapping IDL to DIANA

- Take the node names from the DIANA tree or line dump
- Use the DIANA package constants
- Convert dec numbers to Hex
- These hex numbers are familiar?
- Wrap file / idl / diana dumps are all the same
- Hence wrap format is DIANA
Simple tree structure

D_COMP_U

DS_BODY

DI_PROC
L_symrep = ‘AA’

D_P_

D_BLOCK

DS_PARAM

D_NULL_S

DS_ITEM

DS_STMT

DS_ALTER
DIANA utilities - pstub

SQL> variable a varchar2(2000);
SQL> variable b varchar2(2000);
SQL> exec sys.pstub('AA',NULL,:a,:b);

PL/SQL procedure successfully completed.

SQL> print :b

B

-----------------------------------------------

procedure AA is begin stproc.init('begin AA; end;'); stproc.execute; end;
procedure AA is begin stproc.init('begin AA; end;'); stproc.execute; end;
procedure AA is begin stproc.init('begin AA; end;'); stproc.execute; end;

SQL>
DIANA utilities - subptxt

SQL> variable a varchar2(2000);
SQL> exec sys.subptxt('AA',NULL,NULL,:a);

PL/SQL procedure successfully completed.

SQL> print :a

A
-----------------------------------------------

procedure AA;

SQL>
PSTUB and SUBPTXT

- PSTUB and SUBPTXT are demonstration programs that use the IDL and DIANA API’s.
- PSTUB is used to allow the calling of V2 PL/SQL in the server from V1 PL/SQL client tools such as Forms.
- SUBPTXT allows the describing of PL/SQL.
- Both read DIANA and not PL/SQL source code.
- Pistub.sql and the library diutil.sql are the only public example programs to use the DIANA and PIDL packages.
- Diutil.exprtext (private function) is an excellent example of how to use DIANA and PIDL package calls.
Writing a PL/SQL un-wraper

- To create an unwrapping tool we need
  - To understand the relationship between DIANA and PL/SQL language constructs
  - A way to parse the DIANA in the correct order – API calls?
  - A way to read and understand the DIANA node types – API calls?
  - A way to read variable attributes for each node and to read their type and value – API calls

- Mapping PL/SQL to DIANA for some language constructs can be done using test programs and dumpdiana
Limitations of a PL/SQL API based un-wrapper

- A comprehensive PL/SQL un-wrapper can be written using the IDL and DIANA PL/SQL package API’s
- The $OH/rdbms/admin/diutil.sql file indicates how
- PIDL API’s do not emit the complete DIANA
- The DIANA for the body of procedures and functions is not available via the dumpdiana, PIDL, DIANA interfaces (see the next slide)
- The DIANA dump misses PL/SQL in the block section. Local variables are also not included
- It could be possible to write a complete un-wrapper in PL/SQL and read the DIANA from SYS.SOURCE$
PL/SQL API limitations

SQL> create or replace procedure ah (i in number, j out varchar2) is
2  begin
3  if i = 7 then
4     j := 3;
5  else
6     j := 4;
7  end if;
8  end;
9  /

Procedure created.

SQL> exec dumpdiana.dump(aname => 'AH',print_format => 1);

PL/SQL procedure successfully completed.
Enumerating DIANA nodes and attributes

SQL> exec attrib(23);
Node Type D_COMP_U
Num Attributes 9
0: 9:A_CONTEX:1: REF 1
1: 40:A_UNIT_B:1: REF 1
2: 62:AS_PRAGM:1: REF 1
3: 114:SS_SQL:30: REF 0
4: 113:SS_EXLST:30: REF 0
5: 111:SS_BINDS:30: REF 0
6: 41:A_UP:1: REF 0
7: 138:A_AUTHID:2: REF 0
8: 142:A_SCHEMA:2: REF 0

PL/SQL procedure successfully completed.

SQL>

- See attrib.sql - Also at http://www.petefinnigan.com/attrib.sql
- Uses PIDL to enumerate DIANA nodes and attributes
Creating a real PL/SQL un-wrapper

- Can a complete un-wrapper be written? – Of course, yes
  - There are at least 4 unwrapping tools that I know of
- The complete PL/SQL and SQL grammars are needed - [http://www.antlr.org/grammar/1107752678378/PLSQLGrammar.g](http://www.antlr.org/grammar/1107752678378/PLSQLGrammar.g)
  - Also see “PL/SQL user reference guide”
- It is necessary to understand all DIANA nodes and to map those to PL/SQL – this is not fully documented (partly it is documented as ADA / DIANA)
- It is necessary to understand the wrap file format and to extract the DIANA nodes and attributes from it
- It may be possible to disassemble M-Code back to PL/SQL
- The symbols are embedded in the M-Code
Keywords

SQL> desc v$reserved_words

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYWORD</td>
<td></td>
<td>VARCHAR2 (64)</td>
</tr>
<tr>
<td>LENGTH</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

SQL> select count(*) from v$reserved_words;

<table>
<thead>
<tr>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>809</td>
</tr>
</tbody>
</table>

SQL>
Showing the PL/SQL M-Code as assembler

```
SQL> create or replace procedure ab as
  2  ae number:=1;
  3  begin
  4    ae:=ae+1;
  5  end;
  6  /

Procedure created.

SQL> alter session set events '10928 trace name context forever, level 1';

Session altered.

SQL> exec ab;

PL/SQL procedure successfully completed.

SQL> alter session set events '10928 trace name context off';

Session altered.
```

SIEMENS
The M-Code assembler

Entry #1
00001: ENTER 4, 0
<source not available>
00007: XCAL 1, 1
Entry #1
SYS.AB: 00001: ENTER 76, 0
SYS.AB: 00007: INFR DS[0]+96
Frame Desc Version = 2, Size = 22
# of locals = 2
TC_SSCALARi: FP+4, d=FP+12
TC_SSCALARi: FP+8, d=FP+44
[Line 2] ae number:=1;
SYS.AB: 00012: CVTIN HS+0 =1=, FP+4
[Line 4] ae:=ae+1;
SYS.AB: 00017: CVTIN HS+0 =1=, FP+8
SYS.AB: 00022: ADDN FP+4, FP+8, FP+4
SYS.AB: 00029: RET
00012: RET

- PL/SQL source is shown
- When wrapped – source not available – is shown
- M-Code is mapped to PL/SQL line numbers
- This implies that the line and column details are held in the M-Code
Native compilation and initialisation parameters

- PL/SQL can be natively compiled
- There are a number of initialisation parameters – “sho parameter” in SQL*Plus
- It is possible in some versions to use the native compilation to hack Oracle
- It could be possible to inject PL/SQL code via native compilation
- The generated C Code is M-Code VM calls for each instruction
Some sample code – getting started

SQL> set serveroutput on size 1000000
SQL> exec unwrap('AA');
Start up
Root Node :262145
Root code (hex) :23
Root Type :D_COMP_U
--
A_UNIT_B Node :262147
A_UNIT_B Type :D_S_BODY
A_UNIT_B code (hex) :104
--
A_D_ Node :262148
A_D_ Type :DI_PROC
A_D_ code (hex) :154
--
A_HEADER Node :262149
A_HEADER Type :D_P_
A_HEADER code (hex) :85

- See unwrap.sql (also on http://www.petefinnigan.com/unwrap.sql)
- Test program to
  - Familiarise with the API’s
  - Walk the DIANA nodes
  - Read attributes
- It works! Next, work out the PL/SQL that should be emitted for each node or node group
PL/SQL code generation

- **DS_BODY**
  - **DI_PROC** = ‘AA’
  - **D_P_** = params
    - **DS_PARAM**
  - **D_BLOCK**
    - **DS_ITEM** – local variable
    - **DS_STM**
      - **D_NULL_S**
    - **DS_ALTER**

  “CREATE %{} END;
  l_symrep => PROCEDURE ‘AA’
  {not implemented}
  {not implemented}
  “IS” “BEGIN” %{} “EXCEPTION” %{}
  “END;”
  {not implemented}
  No output
  NULL;
  {not implemented}
A proof of concept un-wrapper

SQL> set serveroutput on size 1000000
SQL> exec unwrap_r('AA');
Start up
CREATE OR REPLACE
PROCEDURE AA
IS
BEGIN
NULL;
END;
/

PL/SQL procedure successfully completed.

SQL>

- Unwrap_r.sql – also available from http://www.petefinnigan.com/unwrap_r.sql
- Implements the code generation to create PL/SQL from DIANA for a simple procedure
- Uses a simple recursive descent parser
create or replace procedure unwrap_r(aname varchar2)
is
  root sys.pidl.ptnod;
  status sys.pidl.ub4;
procedure recurse (n sys.pidl.ptnod) is
  seq sys.pidl.ptseqnd;
  len integer;
begin
  if(pidl.ptkin(n) = diana.d_comp_u) then
    recurse(diana.a_unit_b(n));
  elsif (pidl.ptkin(n) = diana.d_s_body) then
    dbms_output.put_line('CREATE OR REPLACE ');
    recurse(diana.a_d_(n));
    recurse(diana.a_header(n));
    recurse(diana.a_block_(n));
    dbms_output.put_line('END;');
    dbms_output.put_line('/');
  {output snipped}
10g – Different but the same?

- **New**
  - A new wrap mechanism has been provided
  - The contents of symbol table are no longer visible
  - The encryption involves base64
  - 10gR2 provides the ability to wrap from within the database using DBMS_DDL
  - There is a new optimizing compiler for PL/SQL

- **Old**
  - The IDL$ tables still contain DIANA and M-Code
  - The DIANA, PIDL, DIUTIL and DUMPDIANA packages are still available
  - It is still possible to reverse simple procedures using the API’s
The 10g wrapped procedure

SQL> select text from dba_source where name='AA';

TEXT
------------------------------------------------------------------------
--------
procedure aa wrapped
a000000
1
abcd
{identical output snipped}
abcd
7
21 55
tpZtVM0u71C31uX+QfYfxhNmy+Awg5nnm7+fMr2ywFy49cO1dIvAwDL+0oabmYEILYvAgcct
yaam9+Lntg==

- This is base64 character set
- Using base64 decode does not reveal the source
- The symbol table is not visible
Create procedure and check IDL use in 10g

```
SQL> create or replace procedure aa is
     2   begin
     3   null;
     4   end;
     5  /

Procedure created.
```

- The same sample procedure
- Wrap with 10g `wrap`
- Roughly the same IDL is created in the database as 9i

```
SQL> save aa.sql replace
Wrote file aa.sql
SQL> !wrap iname=aa.sql oname=aa.pls
SQL> @aa.pls
Procedure created.
```

```
BLK  FNO  RNUM   'CHAR PART  VERSION  PIECE#  LENGTH  
----- ----- ----- ------- ------- ------- ------- 
49722  1   22  sb4$   0      167772160   0       14 
49722  1   23  sb4$   0      167772160   1       2  
24966  1   23  sb4$   0      167772160   2       3  
46407  1   23  sb4$   0      167772160   3       5  
52973  1   23  sb4$   0      167772160   4      131 
52973  1   23  sb4$   0      167772160   5      10  
49722  1   24  sb4$   2      167772160   0       18 
15481  1   24  sb4$   2      167772160   1       174 
15481  1   24  sb4$   2      167772160   2       1  
```

9 rows selected.
Simple unwrapping PL/SQL in 10g

SQL> exec dumpdiana.dump(aname => 'AA');
user: SYS
PL/SQL procedure successfully completed.
SQL> @unwrap_r
Procedure created.
SQL> exec unwrap_r('AA');
Start up
CREATE OR REPLACE
PROCEDURE AA
IS
BEGIN
NULL;
END;
/
PL/SQL procedure successfully completed.

- Running dumpdiana creates the same DIANA tree dump trace file as 9i
- Running the proof of concept un-wrapper still works in 10g
- The wrap process in 10g is different though
Protecting PL/SQL based intellectual property

- Can you protect PL/SQL based intellectual property?
- Write PL/SQL as packages; DIANA is not stored in the database
- 9i and 10g wrap mechanisms have both been cracked and un-wrappers are available but not to most people
- Don’t ship source code to the server
- 10g affords better protection because the symbol tables are not visible and the DIANA cannot be read from SOURCE$ but the mechanism is not as strong as 10g
- Protect database structures such as IDL_CHAR$, IDL_UB1$, IDL_UB2$, IDL_SB4$, SOURCE$, ALL_SOURCE, DBA_SOURCE
- Use the scripts from http://www.petefinnigan.com/tools.htm to confirm who can access tables and views
Scripts used

- Rowid.sql – lists the contents of the IDL$ tables
- Idl.sql – lists the IDL contents for all parsed objects
- Unwrap.sql – test program to walk the DIANA nodes
- Unwrap_r.sql – Proof of concept PL/SQL unwrapper
- Ah.sql – test program
- Aa.sql – test program
- Attrib.sql – dumps DIANA types and attributes
- All scripts are available on [http://www.petefinnigan.com](http://www.petefinnigan.com) – add the script name to the URL
Questions and Answers

- Any Questions, please ask
- Later?
  - Contact me via email peter.finnigan@siemens.com
  - Or via my website http://www.petefinnigan.com
Insight Consulting
Siemens Communications

Security, Compliance, Continuity and Identity Management

SIEMENS