A Forensic Investigation of PL/SQL Injection Attacks in Oracle

1st July 2010

David Litchfield

(david@v3rity.com)

PL/SQL injection vulnerabilities are one of the more commonly found security flaws in the Oracle database server; literally hundreds have been discovered (and fixed) since they first came to prominence in 2001. Some of the more well known PL/SQL injection flaws include those found in CTXSYS.DRILOAD, SYS.DBMS_EXPORT_EXTENSION and the recently patched flaw in DBMS_JAVA. Attackers can exploit PL/SQL injection flaws to trivially gain elevated privileges on the database server and this has been well documented in other papers by the author. This paper, however, asks what does a PL/SQL injection attack look like under the covers after the dust has settled - what evidence of such an attack may be left on the server and how does a breach investigator go about finding it. The tool used during the investigation is V3rity for Oracle.

For the purposes of this paper let's assume there's an attacker using an account called GREMLIN and there's a vulnerable procedure owned by the SYS user and is executable by PUBLIC with the following code:

```
CREATE OR REPLACE PROCEDURE ARB_SQL( SQLQRY VARCHAR) IS
BEGIN
      EXECUTE IMMEDIATE SQLQRY;
END;
```

It is clear from this code that the procedure simply executes the supplied SQL; further it will execute with the privileges of the SYS user providing a clear path for privilege escalation. Whilst intended to be simple, many of the flaws that have been patched in Oracle are very similar in nature.
Typically an attack may involve granting system privileges, granting roles, modifying permissions on objects or the creation of new objects. Once done an attacker may attempt to clean up after themselves which may involve dropping objects and revoking privileges. Such operations are known as DDL (Data Definition Language). For example, an attacker may choose to grant themselves DBA privileges:

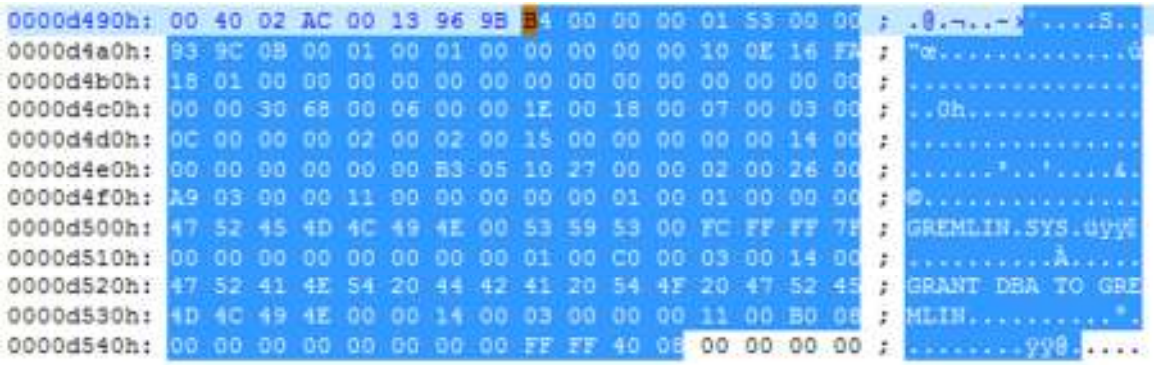
```
SQL> EXEC SYS.ARB_SQL('GRANT DBA TO GREMLIN');
```

Using dump ddl action of V3rity for Oracle on the redo log file after such an attack reveals the following (output trimmed for clarity):

```
C:\Users\david\v3rity>dumpaction redo01.log ddl
<?xml version="1.0"?><LOG>
<FILENAME>redo01.log</FILENAME>
<database_sid>ORCL</database_sid>
<version>10.1</version>
<ltimestamp>04/03/2010 21:30:13</ltimestamp>
<blocksize>512</blocksize>
<nab>136</nab>
<lowscn>760920</lowscn>
<nextscn>761030</nextscn>
<ENTRIES>
```

```
<ENTRY>
<TIMESTAMP>01/07/2010 15:30:15</TIMESTAMP>
<RDRCOFST>0x0000D498</RDRCOFST>
<CHVCOFST>0x0000D4C8</CHVCOFST>
<SESSION_USER>GREMLIN</SESSION_USER>
<CURRENT_USER>SYS</CURRENT_USER>
<SQL_STATEMENT>GRANT DBA TO GREMLIN</SQL_STATETMENT>
<SCHEMA></SCHEMA>
<OBJECT></OBJECT>
</ENTRY>
</ENTRIES>
</LOG>
```

We can see that just after 15:30 on the 1st of July that the session user GREMLIN executed the GRANT DBA TO GREMLIN statement, however the current user is the SYS user. This record can be found 0x0000D498 bytes into the redo log file and opening it in a binary viewer confirms this:



The fact that the session user and the current user are different could be indicative of a PL/SQL injection attack but bear in mind that this is not alway so. Any PL/SQL procedure that executes DDL as the definer of the procedure will appear as so when executed by another user. That said, the fact that they session user and current user are different should draw the investigators attention and it should be analyzed further. Looking at the DDL that has been executed it is clear that a privilege escalation attack has taken place.

A more advanced attack may involve direct updates to the data dictionary using UPDATEs, INSERTs or DELETEs. These are DML operations (Data Manipulation Language). For example, rather than doing performing the GRANT we've just looked at, the same objective can be achieved by performing a direct INSERT on the SYSAUTH$ table, the table that keeps track of role assignments:

```
SQL> EXEC SYS.ARB_SQL('INSERT INTO SYS.SYSAUTH$ (GRANTEE#, PRIVILEGE#,
SEQUENCE#) VALUES(66,4,(SELECT MAX(SEQUENCE#)+1 FROM SYS.SYSAUTH$))');
```

Dumping the redo log with v3rity using the RED[O] option to dump 5.1 operations we have the following entry:

```
<ENTRY>
<TIMESTAMP>01/07/2010 19:02:55</TIMESTAMP>
<SCN>0x0000.000C16D7</SCN>
<RDRCOFST>0x0030C610</RDRCOFST>
<XID>0003.0003.000007BF</XID>
<OBJECT_ID>91</OBJECT_ID>
<OPC>11.1</OPC>
<USER_ID>0</USER_ID>
</ENTRY>
```

Here we can see an INSERT has been performed against an object with ID 91. This is the ID for SYSAUTH$. The user that performed the action is SYS - the user with the user ID of 0. In other words, the user ID of the attacker is not logged but rather SYS. The follwing is a dump of the redo log using the ALTER SYSTEM DUMP LOGFILE command confirming the user ID is 0.

```
REDO RECORD - Thread:1 RBA: 0x000095.00001863.0010 LEN: 0x01a8 VLD: 0x05
SCN: 0x0000.000c1700 SUBSCN:  1 07/01/2010 19:02:55
CHANGE #1 TYP:0 CLS:21 AFN:2 DBA:0x00800029 SCN:0x0000.000c16d8 SEQ:  1
OP:5.2
ktudh redo: slt: 0x0003 sqn: 0x000007bf flg: 0x0012 siz: 128 fbi: 0
            uba: 0x0080041a.03b3.23    pxid:  0x0000.000.00000000
CHANGE #2 TYP:0 CLS:22 AFN:2 DBA:0x0080041a SCN:0x0000.000c16d7 SEQ:  3
OP:5.1
ktudb redo: siz: 128 spc: 4954 flg: 0x0012 seq: 0x03b3 rec: 0x23
            xid:  0x0003.003.000007bf
ktubl redo: slt: 3 rci: 0 opc: 11.1 objn: 91 objd: 91 tsn: 0
Undo type:  Regular undo        Begin trans    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
             0x00000000  prev ctl uba: 0x0080041a.03b3.20
prev ctl max cmt scn:  0x0000.000c0ecc  prev tx cmt scn:  0x0000.000c0ecf
txn start scn:  0xffff.ffffffff  logon user: 0  prev brb: 8389651  prev
bcl: 0 KDO undo record:
KTB Redo
op: 0x04  ver: 0x01
op: L  itl: xid:  0x0002.026.000003a9 uba: 0x00800550.01a4.56
                      flg: C---    lkc:  0     scn: 0x0000.000b9c95
KDO Op code: DRP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x004002ac  hdba: 0x004002a9
itli: 1  ispac: 0  maxfr: 4863
tabn: 0 slot: 20(0x14)
CHANGE #3 TYP:2 CLS: 1 AFN:1 DBA:0x004002ac SCN:0x0000.000c15c1 SEQ:  1
OP:11.2
KTB Redo
op: 0x01  ver: 0x01
op: F  xid:  0x0003.003.000007bf    uba: 0x0080041a.03b3.23
KDO Op code: IRP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x004002ac  hdba: 0x004002a9
itli: 1  ispac: 0  maxfr: 4863
tabn: 0 slot: 20(0x14) size/delt: 13
```

```
fb: --H-FL-- lb: 0x1  cc: 3
null: ---
col  0: [ 2]  c1 43
col  1: [ 2]  c1 05
col  2: [ 3]  c2 0b 45
```

This is problematic because, *without corroborating evidence*, it's difficult to tie this to GREMLIN other than the fact that 66 is their user ID. Whilst there is a transaction ID there is nothing to link this to a session ID and therefore the user. To do this would require other logs such as web server logs if such an attacker was launched via a web application, or the audit trail if auditing is enabled which could show who was logged on at the time of the attack. One should note that only those actions taken by a user logged on as SYS are logged when AUDIT_SYS_OPERATIONS is set to TRUE. In other words, any SQL executed indirectly by SYS via a stored procedure will not be logged.

[1] http://www.databasesecurity.com/HackingAurora.pdf