Executing SQL as SYS from APPS in Oracle's eBusiness Suite

(and trying to prevent it!)

David Litchfield

7th June 2016

Introduction

This paper presents several methods for the APPS user to execute SQL as SYS in Oracle's eBusiness Suite 12.2 and earlier. As SQL executed from any web-based application executes as the APPS user, these methods present a critical risk to the backend database server and they need to be prevented.

There are a number of PL/SQL packages owned by SYS that can be executed by APPS. There are many more that can be executed by SYSTEM and there are also packages owned by SYSTEM that can be executed by APPS: this indirectly increases the attack surface exposed to APPS by SYS; by exploiting extant flaws in SYSTEM owned packages APPS can gain access to SYS owned packages and from there exploit any weaknesses.

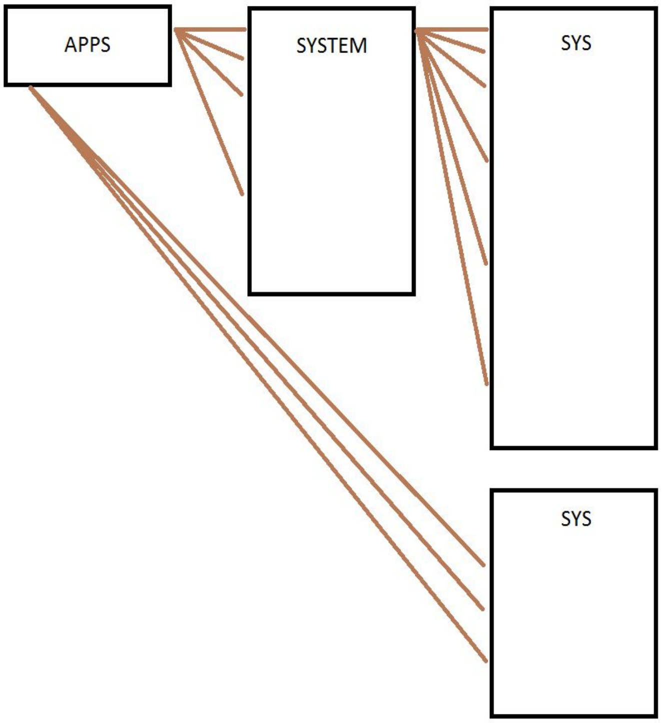

Figure 1: Direct and Indirect Attack Surface

Of course, this chaining might not even be necessary. There are a number of packages owned by SYS that can be executed APPS that allow for the execution of arbitrary SQL. For example, in earlier, unpatched versions of eBusiness Suite, APPS can execute [illegible]. This package contains a function [illegible] that allows SQL to be parsed under a given user ID, including SYS, which can then be executed with that user's privileges:

Direct execution of DBMS_SYS_SQL

```
[illegible]
```

Listing 1: Executing DBMS_SYS_SQL directly

This needs to be "weaponized" if it is to become useful to an attacker exploiting a SQL injection flaw in the web front end. In the exploit below, which can be injected via a web server SQL injection flaw, the code in Listing 1 above is essentially sent to the [illegible] function for execution. This function executes an arbitrary SQL statement and is known in attacker circles as an *auxiliary inject function*; See Appendix A for more auxiliary inject functions available in eBusiness Suite.

```
[illegible]
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

Revoking the execute privilege from APPS on DBMS_SYS_SQL will prevent this particular attack vector. Indeed, it was reported to Oracle in 2014 and they addressed it in April 2016 - see CVE-2016-0697 in http://www.davidlitchfield.com/OracleCPUApril2016.pdf

SQL Injection in SYSTEM.AD_APPS_PRIVATE

In later, patched versions of eBusiness Suite, APPS no longer has the execute privilege on [illegible] but SYSTEM does. So if APPS can exploit a SQL injection flaw in a SYSTEM owned package APPS can again gain access to [illegible]. In the exploit below, APPS uses the [illegible] [illegible] auxiliary inject function to execute the [illegible] procedure whilst simultaneously exploiting a SQL injection flaw in it to execute [illegible] so the chain goes from [illegible] to [illegible] [illegible] to [illegible]

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

SQL injection in SYSTEM.AD_INST

In addition to the SQL injection flaws in AD_APPS_PRIVATE, AD_INST also suffers from SQL injection flaws:

```
[illegible]
          ( [illegible] [illegible]
          [illegible]

[illegible]
C [illegible]
[illegible]
[illegible] [illegible]
[illegible]
```

```
[illegible]
[illegible]                              in_schema[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

This to can be exploited to gain access to DBMS_SYS_SQL.

Revoking the EXECUTE privilege from SYSTEM on DBMS_SYS_SQL would go a long way to preventing abuse. However, this still leaves the attack surface presented by packages in the SYS schema that are directly executable by APPS.

We could revoke the execute privileges from APPS on AD_INST and AD_APPS_PRIVATE too but given APPS has the EXECUTE ANY PROCEDURE privilege we'd need to revoke that privilege too.

SQL Injection in SYS.AD_ZD_SYS

AD_ZD_SYS is owned by and executes with SYS privileges. It is executable by APPS. It suffers from multiple SQL injection flaws. If we examine the source we find the LOG procedure:

```
[illegible]                                                      X_MESSAGE
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
.
.
[illegible]
[illegible]

'          [illegible]            [illegible]
[illegible]
'          [illegible]
           [illegible]
           [illegible]            x_message[illegible]
           [illegible]
```

```
                  [illegible]
[illegible]
```

If we look at what calls the LOG procedure we see, for example,

```
[illegible]                          X_STATUS[illegible]
[illegible]
[illegible]
[illegible]
  [illegible] [illegible]       [illegible]                          x_status)
.
.
```

As we can see, [illegible] S is passed to [illegible] without validation and then concatenated in a dynamic INSERT statement as part of [illegible] . [illegible] is controlled by the user.

This can be trivially exploited. As a simple PoC:

```
[illegible]
[illegible]
[illegible]
```

This will execute the [illegible] [illegible] function as SYS.

Revoking the execute privilege from APPS will help prevent this as an attack vector; however, as SYSTEM also has the execute privilege it needs to be revoked from SYSTEM too, otherwise an APPS to SYSTEM to SYS chain can be used.

Summary

Whatever SYS can do, APPS can do too if there is a SQL injection flaw in a SYS owned package executable by APPS. Further, whatever SYSTEM can access in the SYS schema so too can APPS if there is a SQL injection flaw in a SYSTEM owned package executable by APPS. *Indeed, given APPS has the EXECUTE ANY PROCEDURE privilege any vulnerable object in any non-SYS schema will give APPS access to all the SYS packages that user has access to. (For example, XDB can execute SYS.DBMS_PDB_EXEC_SQL.)*

The question is how does one limit exposure? Sure, we can revoke execute privileges, we can revoke the EXECUTE ANY PROCEDURE privilege and so on but this is just spot fixing. Unless you really embark on a project that strips down APPS to the bare bones, you really need to consider other options such as a database firewall. And just food for further thought, given the architecture of eBusiness Suite even if we’re left with a perfect situation where no lateral or vertical privilege movement is possible, given APPS owns all the key data we’re still left with that to manage.

Appendix A

Auxiliary inject functions are functions that execute an arbitrary SQL statement and therefore allow the execute of any SQL including DML and DDL even from a SELECT statement. (This is achieved by specifying the [illegible] pragma in the declaration block)

APPS.ASG_CUSTOM_PVT.EXEC_CMD
This function returns a VARCHAR.
Example:

[illegible]

[illegible]

APPS.WIP_MASS_LOAD_UTILITIES.DYNAMIC_SQL
This function returns a NUMBER.
Example:

[illegible]

[illegible]

[illegible]

APPS.MSC_GET_NAME.EXECUTE_SQL_GETCOUNT
This function returns a NUMBER.
Example:

[illegible]

[illegible]

APPS.BSC_UPDATE_UTIL.EXECUTE_IMMEDIATE
This function returns a NUMBER.
Example:

[illegible]

[illegible]

Note: does not exist in EBS R12

APPS.PSB_WS_ACCT1.DSQL
This function returns a NUMBER.
Example:

[illegible]

[illegible]

Note: does not exist in EBS R12

Note: [illegible] and [illegible] [illegible] can be used in DML inject points (their code issues a savepoint)