# XXE defence(les)s in JDK XML parsers

Jonathan Brossard

# XXE defence(les)s in JDK XML parsers

Sergey Gorbaty sergey.gorbaty@salesforce.com
Xiaoran Wang xiaoran.wang@salesforce.com
Hormazd Billimoria hbillimoria@salesforce.com
Jonathan Brossard jbrossard@salesforce.com

Product Security, Salesforce.com, San Francisco, CA, USA

**Abstract.** This article will demonstrate that many Oracle JDK XML parsers are vulnerable to Xml eXternal Entity (XXE) attack. We shall also demonstrate that existing Java defenses against XXE attacks fall short and fail to protect against malicious malformed XML, which results in a parsing error with external entities successfully expanded. Our exploit has the capability of exfiltrating files and launching directory traversals. We present a proof of concept dumping filenames under /tmp directory from a remote server running JDK 7 via untrusted XML files. It is our hope to raise awareness of the industry regarding the dangers of XXE-type of attacks. This should also result in upgrading the best practices for disabling external entity resolution for several XML parsers.

Keywords: Java, XXE, XML, parser.

# 1 Introduction

In the present article we will provide an overview of the standard attack using XML external entities. In the following subsection we will describe the effects of specific malformed XML attack scenario for JDK7-based XML parsers, which was first introduced in [1]. In the later subsection we will introduce defences that are recommended by authoritative sources and demonstrate where they fall short.

## 1.1 Background

XML DTD can be constructed from internal, external and parameter entities. External entities are references to other entities that can be found outside of the current document. When XML parser encounters an external entity URI, it expands the reference to include the content from the external link in the current document. An external entity may reference a file or a URL.

```
<?xml version="1.0" encoding="utf-8"?>
    <!DOCTYPE test [
        <!ENTITY xxe SYSTEM "file:///etc/passwd">
    ]>
<test>&xxe;</test>
```

**Fig. 1.** Simple XXE payload

## 1.2 Introducing the Attack

As it was alluded to in the previous section, the exception thrown by the XML parser may happen during various stages of parsing. JDK XML entity expander always assumes the external entity URL is well formed and will attempt to resolve it. Our first goal is to confirm that external entities are being resolved; therefore, we include an external URL in one entity. Our second goal is to create an entity pointing to an invalid URL, which is containing data resulted from expansion of another external entity. Such entity structure would force the XML parser to resolve some sensitive external entities first and then leak the resolved data using specially crafted URL and finally throw an IOException. The IOException in this case is most often shadowed and thrown only after an attempt to resolve the entity URL at which point the attacker had already received the data via an attacker-controlled DNS resolver.

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE foo [
        <!ENTITY % payload SYSTEM "file:///etc">
        <!ENTITY % dtd SYSTEM "http://externallyhostedentity.com/a.dtdâĂĬ>
        %dtd;
        %release;
    ]><foo>business as usual</foo>
```

**Fig. 2.** XXE with entity cross-references

```
<!ENTITY % nested "<!ENTITY &#x25; release SYSTEM 'jar:%payload;.externallyhostedentity.com!/'>">
%nested;
```

**Fig. 3.** Externally hosted a.dtd

The main reason for having a separate a.dtd is that expanded references cannot be referenced in the same document.

Figures 2 and 3 demonstrate how an attacker may learn if the parser is vulnerable to XXE by monitoring DNS resolver and/or `http://externallyhostedentity.com` HTTP logs. Furthermore, the XML parser may leak the directory content of /etc to the authoritative DNS resolver that holds A-type record for the attacker controlled `http://externallyhostedentity.com`.

## 2 Insufficient Recommendations

Oracle JDK7 documentation contains plethora of information about XML parser configuration. One of the first options that come to developer's mind is to enable *FEATURE_SECURE_PROCESSING* in order to prevent external connections. "*When FEATURE_SECURE_PROCESSING is enabled it is recommended that implementations* restrict *external connections by default*" [2], alas, despite Oracle's recommendation XML parsers do not actually restrict external connections when *FEATURE_SECURE_PROCESSING* is enabled. When it comes to tackling XXE Oracle does not have explicit recommendations but for certain parsers it documents how to turn expansion of XML external entities off while other parses do not have a way to turn it off. OWASP recommendations [3] only cover major JDK parsers and respond to XXE threat by disabling of fetching external DTD altogether on most of them. Long et. all in [4] recommend creating a custom entity resolver for XMLReader but do not cover any others. [5] recommendations provide generic recommendations and only cover *javax.xml.parsers.DocumentBuilderFactory* specifics.

In the following sections we will examine each major JDK XML parser provider and see what capabilities it provides for mitigating XXE.

### 2.1 javax.xml.stream.XMLInputFactory

Using *setProperty* method on this [6] Oracle XML factory, developer can set *javax.xml.stream.isSupportingExternalEntities* property to false. Unfortunately, as we discovered [7] setting this property was not properly functioning and resulted in a 0-day vulnerability, which was addressed in JDK update 7u67.

OWASP recommendation [3] for this parser ignores useful needs for XML DTD and recommends disabling external DTD altogether, which certainly fixes the problem but may not be in line with the business needs.

### 2.2 javax.xml.parsers.DocumentBuilderFactory

OWASP recommendations disabling the DTD but also mentions that if one cannot completely block DTD, she should simply disable the following propertes: `http://xml.org/sax/features/external-general-entities`, `http://xml.org/sax/features/external-parameter-entities`.

Unfortunately, the documentation does not explicitly tell that disabling both of the above properties is required. Disabling `http://xml.org/sax/features/external-general-entities` on its own would have no effect against the attack mentioned earler.

Morgan et. all [5] and OWASP recommendations appear to be very similar.

Oracle documents [8] that *setAttribute* method in this factory can be used to set *XMLConstants.ACCESS_EXTERNAL_DTD* and *XMLConstants.ACCESS_EXTERNAL_SCHEMA* properties, which would allow a developer to restrict protocols that can be used to fetch DTD or schema. The developer should be aware that *jar://* protocol is quite dangerous and should be excluded as it can resolve files and externally hosted websites.

### 2.3 javax.xml.parsers.SAXParserFactory

Oracle documentation [9] for this factory does not include any features that would help us disable external entities processing. OWASP includes recommendations that range from disabling DTD to disabling external entities only. The defenses listed by OWASP are similar to the ones outlined in 2.2.

### 2.4 javax.xml.transform.sax.SAXTransformerFactory and javax.xml.transform.TransformerFactory

Oracle provides in [10] information on how to disable protocols that can be used to fetch external DTD and external entities. Unfortunately, it is not possible to turn off external entities without disabling DTD using *XMLConstants.ACCESS_EXTERNAL_DTD* attribute.

### 2.5 javax.xml.validation.SchemaFactory and javax.xml.validation.Validator

These particular parsers allow a developer to provide a custom external resource resolver using *setResourceResolver* method [11][12]. Unfortunately, the default parameter value of *null* does not result in a safe behavior and is, essentially, a no-op. The developer must supply a proper resolver else the attack will succeed.

An alternative and safe way to mitigate the attack is to utilize *setProperty* method [13][14] with *XMLConstants.*
*ACCESS_EXTERNAL_DTD* as the first parameter and whitelisted protocols as the second.

### 2.6 javax.xml.bind.Unmarshaller and javax.xml.xpath.XPathExpression

As Oracle documentation suggests there isn't a way to modify behavior of *Unmarshaller* in terms of resolving external entities: "*There currently are not any properties required to be supported by all JAXB Providers on Unmarshaller*" [15]. *XPathExpression* simply does not expose any public method to set properties.

The only option to make these two parsers safe available is to parse the XML first, using a different safe parser, and then pass the result in. E.g. for *Unmarshaller* a developer would need to produce a safe *java.xml.transform.Source* and pass it to the *unmarshal* method. And for *XPathExpression* a safely parsed *org.xml.sax.InputSource* needs to be passed to the *evaluate(...)* method [16].

## 3 Conclusion

Each JDK parser has a specific configuration when it comes to preventing XXE attacks. It is important to configure the parser for handling incorrect input as well as the one that is correct but knowingly produces errors.

## References


1. Timur Yunusov, A.O.: Xml out-of-band data retrieval, BlackHat USA. https://media.blackhat.com/eu-13/briefings/Osipov/bh-eu-13-XML-data-osipov-slides.pdf (2013)
2. Oracle: Property XMLConstants.ACCESS_EXTERNAL_DTD. (http://docs.oracle.com/javase/7/docs/api/javax/xml/XMLConstants.html#ACCESS_EXTERNAL_DTD)
3. OWASP: Xml external entity (xxe) processing. (https://www.owasp.org/index.php/XML_External_Entity_%28XXE%29_Processing)
4. Fred Long, Dhruv Mohindra, R.C.D.F.D.S.: The CERT Oracle Secure Coding Standard for Java. Addison-Wesley (2012)
5. Timothy D. Morgan, O.A.I.: Xml schema, dtd, and entity attacks. http://vsecurity.com/download/papers/XMLDTDEntityAttacks.pdf (2014)
6. Oracle: Class xmlinputfactory. (http://docs.oracle.com/javase/7/docs/api/javax/xml/stream/XMLInputFactory.html)
7. Oracle: Oracle critical patch update advisory - october 2014. http://www.oracle.com/technetwork/topics/security/cpuoct2014-1972960.html (2014)
8. Oracle: Function DocumentBuilderFactory.setAttribute(..). (http://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/DocumentBuilderFactory.html#setAttribute(java.lang.String,%20java.lang.Object))
9. Oracle: Function SAXParserFactory.setFeature(..). (http://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/SAXParserFactory.html#setFeature(java.lang.String,%20boolean))
10. Oracle: Function TransformerFactory.setAttribute(..). (http://docs.oracle.com/javase/7/docs/api/javax/xml/transform/TransformerFactory.html#setAttribute(java.lang.String,%20java.lang.Object))
11. Oracle: Function SchemaFactory.setResourceResolver(..). (http://docs.oracle.com/javase/7/docs/api/javax/xml/validation/SchemaFactory.html#setResourceResolver(org.w3c.dom.ls.LSResourceResolver))
12. Oracle: Function Validator.setResourceResolver(..). (http://docs.oracle.com/javase/7/docs/api/javax/xml/validation/Validator.html#setResourceResolver(org.w3c.dom.ls.LSResourceResolver))
13. Oracle: Function SchemaFactory.setProperty(..). (http://docs.oracle.com/javase/7/docs/api/javax/xml/validation/SchemaFactory.html#setProperty(java.lang.String,%20java.lang.Object))
14. Oracle: Function Validator.setProperty(..). (http://docs.oracle.com/javase/7/docs/api/javax/xml/validation/Validator.html#setProperty(java.lang.String,%20java.lang.Object))
15. Oracle: Class Unmarshaller. (http://docs.oracle.com/javaee/7/api/javax/xml/bind/Unmarshaller.html#supportedProps)

16. Oracle: Function XPathExpression.evaluate(..). (http://docs.oracle.com/javase/7/docs/api/javax/xml/xpath/XPathExpression.html#evaluate(org.xml.sax.InputSource))