XXEBugFind User Guide

04 September, 2014

ABOUT

XXEBugFind is a tool for detecting XML External Entity (XXE) Attacks in Java applications. . Here is a link to the tool's repository: https://github.com/ssexxe/XXEBugFind. The tool is based on Soot version 2.5.0.

Dependencies

The tool needs the following libraries to run,

- Soot-2.5.0.jar – a bytecode analysis framework found at http://www.sable.mcgill.ca/soot. Note that the version 2.5.0 is necessary for our tool to run effectively.
- JUnit-4.10.jar- a testing framework to run our tests

In addition, the Java runtime must be present on the system as this tool is Java based. You can download the tool via Oracle's website.

Getting Started

Download the tool via the GitHub link provided in the about section. Run Apache *Ant* on the target *default* in the build.xml file located at the root directory. Alternatively, open the *XXEBugFind standalone* folder to start using the tool immediately. The folder should contain the following

- XXEBugFind.jar – this is the main application
- lib – this folder contains the libs used by XXEBugFind

Running the tool

To get started, invoke the tool via the command as follows:

Java –jar XXEBugFind.jar –d "path to one or more jar files" – l "path to one or more libs used by the file"

There is a sample application in the tutorial folder called *MyXXETestApp.* It depends on three XML libraries namely JDOM, DOM4J, WoodStox. The dist subfolder contains the binary applications.

XXEBugFind Commands

Usage: -d [-l <value>] [-f <value>] [-o <value>] [-rs <value>] [-rtloc <value>]

Basic commands are shown below:

-d

Use this to specify the jar file or files you want to analyse. You can specify multiple files separated by a semicolon. It is best to always enclose the paths in quotations to prevent error when parsing your commands. Specifying only the –d option is allowed for applications that uses Java's JAXP XML libraries for XML processing and do not depend on any external XML libs. See sample usages below.

```
java –jar XXEBugFind –d "myApplication.jar"
java -jar XXEBugFind -d "myjar1.jar;myjar2.jar" -l
“libpath/myXMLlib1.jar;libpath/myXMLlib2.jar"
```

-l

Specifies one one more libraries your java application is dependent on. Not all have to be included. The ones that are necessary are **custom XML parser libraries used** if any e.g. jdom-2.0.5.jar for JDOM. This is because the tool uses the libraries internally for type resolution. Not specifying the XML libs used may yield wrong results. See sample usage below:

```
java -jar XXEBugFind -d "myjar1.jar;myjar2.jar" -l
“libpath/myXMLlib1.jar;libpath/myXMLlib2.jar"
```

-f

Specifies the format of the output. Currently supported formats are **xml** or **text** (in lowercase) e.g. -f xml

```
java -jar XXEBugFind -d "myjar1.jar" -f xml
```

```
java -jar XXEBugFind -d "myjar2.jar" -f text
```

-o

Specifies the output location. Use this option if you want to save the output in a file. It's best to enclose in quotation marks e.g. -o "/users/sse/outputFile"

```
java -jar XXEBugFind -d "myjar1.jar" -f xml -o "/users/outputFile"
```

-rs

Specifies the location of a custom rule-set to be used by the tool. The rule-set file should be an XML file containing all the vulnerabilities to be checked for. This situation may be suitable for vulnerabilities not covered by our tool. Sample usage:

```
java -jar XXEBugFind -d "myjar.jar" -rs "/users/sse/MyRuleset.xml"
```

-rtloc

Specifies the directory containing rt.jar which is usually found in the java installation directory. This command is only needed when the tool is unable to find this file as it is required to run the tool. E.g. -rtloc "/u/path-to-rt-folder"

Sample Usage: *(Using our test app see Running the tool section)*

```
 java -jar XXEBugFind.jar -d "tutorial/MyXXETestApp/dist/MyXXETestApp.jar"
-l "tutorial/MyXXETestApp/dist/lib/jdom-
2.0.5.jar;tutorial/MyXXETestApp/dist/lib/dom4j-
1.6.1.jar;tutorial/MyXXETestApp/dist/lib/stax2-api-
3.1.1.jar;tutorial/MyXXETestApp/dist/lib/woodstox-core-asl-
4.2.0.jar;tutorial/MyXXETestApp/dist/lib/woodstox-core-lgpl-4.2.0.jar;" -f
text -o output.txt
```

Note: For the above usage to work, it is assumed that the tutorial folder and XXEBugFind.jar are in the same parent folder.

Understanding the output

Consider the sample output below:

3 variant(s) of xxe vulnerabilities found

XXE Variant-1 due to using SAXBuilder API. See detail:

org.jdom2.input.SAXBuilder.build(java.io.InputStream);.

1 occurrence(s) at:

* class: myjdom.PrettyPrinter method: public static void main(java.lang.String[]) at line 24

Reason: A call to SAXBuilder.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true); should be made before using SAXBuilder.parse(...) methods to prevent possible XXE attacks. Note that for JDOM version 2.0.5, the setting SAXBuilder.setFeature("http://xml.org/sax/features/external-general-entities", false); does not work and so is disregarded as a mitigation procedure. If this is not the case in a later version, you may decide passing a user-defined ruleset into the application

Exploitation route(s)

* [myjdom.PrettyPrinter: void main(java.lang.String[])] --> [org.jdom2.input.SAXBuilder: org.jdom2.Document build(java.io.InputStream)]

XXE Variant-2 due to using XMLStreamReader API. See detail:

javax.xml.stream.XMLStreamReader.next();.

1 occurrence(s) at:

* class: stax.StAXExample method: public static void main(java.lang.String[]) throws javax.xml.stream.XMLStreamException at line 37

Reason: A call to XMLInputFactory.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES, Boolean.FALSE); or XMLInputFactory.setProperty(XMLInputFactory.SUPPORT_DTD, Boolean.FALSE); should be made before using the StAX parser created from them i.e., before using XMLStreamReader.next() method

Exploitation route(s)

* [stax.StAXExample: void main(java.lang.String[])] --> [javax.xml.stream.XMLStreamReader: int next()]

XXE Variant-3 due to using SAXReader API. See detail:

org.dom4j.io.SAXReader.read(java.io.InputStream);.

1 occurrence(s) at:

* class: dom4j.Dom4JExample method: public org.dom4j.Document parse() throws org.dom4j.DocumentException, org.xml.sax.SAXException at line 31

Reason: A call to SAXReader.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true); or SAXReader.setFeature("http://xml.org/sax/features/external-general-entities", false); should be made before using parsers created from them i.e., before using SAXReader.read(...) methods

Exploitation route(s)

```
* [dom4j.Dom4JExample: void main(java.lang.String[])] --> [dom4j.Dom4JExample: org.dom4j.Document parse()]
--> [org.dom4j.io.SAXReader: org.dom4j.Document read(java.io.InputStream)]
```

Note: The above is a sample output when the sample usage involving MyXXETestApp is run.

The tool gives the number of variants of XXE vulnerabilities detected, number of occurences and exploitation routes on how attackers may reach them. Consider the extract below:

```
XXE Variant-1 due to using SAXBuilder API. See detail:

org.jdom2.input.SAXBuilder.build(java.io.InputStream);.

1 occurrence(s) at:

* class: myjdom.PrettyPrinter method: public static void main(java.lang.String[]) at line 24

Reason: A call to SAXBuilder.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true); should be
made before using SAXBuilder.parse(...) methods to prevent possible XXE attacks. Note that for JDOM version
2.0.5, the setting SAXBuilder.setFeature("http://xml.org/sax/features/external-general-entities", false); does not
work and so is disregarded as a mitigation procedure. If this is not the case in a later version, you may decide
passing a user-defined ruleset into the application

Exploitation route(s)

* [myjdom.PrettyPrinter: void main(java.lang.String[])] --> [org.jdom2.input.SAXBuilder: org.jdom2.Document
build(java.io.InputStream)]
```

Examing the extract above in detail we make the following deductions

- It ishows that the tool found 1 variant due to using the SAXBuilder API. The SAXBuilder is the XML parsing class for the JDOM library.
- The location of the vulnerability is in class myjdom.PrettyPrinter and method of the class public static void main(java.lang.String[]) which is located at line 24
- An exploitation route for an attacker is shown as [myjdom.PrettyPrinter: void main(java.lang.String[])] --> [org.jdom2.input.SAXBuilder: org.jdom2.Document build(java.io.InputStream)] .The implication of this is that an attacker may be able to exploit this vulnerability from outside the application since the main() method is known as an entry point into most application.

Creating Custom Rulesets

Rulesets allow the extension of XXEBugFind for XML parser libraries not supported. Due to the time frame, XXEBugFind currently supports JDOM, SAX, DocumentBuilder, StAX, JAXB, DOM4J xml parsers. We will now extend this tool to support a custom XML parser library called Piccolo.A sample ruleset is shown in the appendix section. But first we explain our ruleset.

A rule-set is an XML file essentially consists of Vulnerability Definition Items, VDIs. A VDI contains the definition of a vulnerable parser method and all the possible mitigation approaches (method calls and settings) that will make it safe. Each mitigation approach has one or more mitigation spoilers which if called after the mitigation method renders the mitigation ineffective. Mitigation spoilers are usually incorrect configurations by the developer or oversight. The sample VDI shown in the appendix has the following properties:

- A method definition of void javax.xml.parsers.SAXParser.parse(java.io.InputStream, org.xml.sax.helpers.DefaultHandler);

- 1 mitigations method and it’s correct mitigation parameters along with their mitigation spoilers
    - javax.xml.parsers.SAXParserFactory.setFeature(“http://xml.org/sax/features/external-general-entities”, false);
        - It’s solution description
        - Mitigation type: FACTORY
            - mitigation spoiler: javax.xml.parsers.SAXParserFactory.setFeature(“http://xml.org/sax/features/external-general-entities”, true);

We run the command as shown below

```
java -jar XXEBugFind.jar -d "tutorial/MyXXETestApp/dist/MyXXETestApp.jar" -l
"tutorial/MyXXETestApp/dist/lib/Piccolo.jar" -rs piccolo-ruleset.xml -f text -o output-piccolo.txt
```

Output is

```
1 variant(s) of xxe vulnerabilities found
XXE Variant-1 due to using Piccolo API. See detail:
com.bluecast.xml.Piccolo.parse(java.lang.String);.
1 occurrence(s) at:
* class: piccolotest.MyPiccoloExample method: public static void
nonSecurePicoloXMLParse(java.lang.String) throws org.xml.sax.SAXException, java.io.IOException
at line 26
Reason: A call to XMLReader.setFeature("http://apache.org/xml/features/disallow-doctype-decl",
true); or XMLReader.setFeature("http://xml.org/sax/features/external-general-entities", false); should
be  made before using parsers created from them i.e., before using XMLReader.parse(...) methods
Exploitation route(s)
* [myxxetestapp.MyXXETestApp: void main(java.lang.String[])] --> [piccolotest.MyPiccoloExample:
void nonSecurePicoloXMLParse(java.lang.String)] --> [com.bluecast.xml.Piccolo: void
parse(java.lang.String)]
```

Appendix

Sample Ruleset

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<VulnerabilityDefinitionItems>

    <VulnerabilityMitigationItem>

      <methodDefinition>

        <ClassName>javax.xml.parsers.SAXParserFactory</ClassName>

        <MethodName>setFeature</MethodName>

        <MethodParameter>

          <ParameterType>java.lang.String</ParameterType>

        </MethodParameter>

        <MethodParameter>

          <ParameterType>boolean</ParameterType>

        </MethodParameter>

      </methodDefinition>

      <ParameterValue>

        <ParameterType>java.lang.String</ParameterType>

        <ParameterValue>"http://xml.org/sax/features/external-general-entities"</ParameterValue>

      </ParameterValue>

      <ParameterValue>

        <ParameterType>boolean</ParameterType>

        <ParameterValue>false</ParameterValue>

      </ParameterValue>

      <MitigationType>FACTORY</MitigationType>

      <SolutionDescription>A call to SAXParserFactory.setFeature("http://apache.org/xml/features/disallow-doctype-decl",
true); or SAXParserFactory.setFeature("http://xml.org/sax/features/external-general-entities", false); should be  made before
using parsers created from them i.e., before using SAXParser.parse(...) methods</SolutionDescription>

      <MitigationSpoiler>

        <ParameterValue>

          <ParameterType>java.lang.String</ParameterType>

          <ParameterValue>"http://xml.org/sax/features/external-general-entities"</ParameterValue>

        </ParameterValue>

        <ParameterValue>

          <ParameterType>boolean</ParameterType>

          <ParameterValue>true</ParameterValue>

        </ParameterValue>

      </MitigationSpoiler>
```

```
        </VulnerabilityMitigationItem>
    </VulnerabilityDefinitionItem>
</VulnerabilityDefinitionItems>
```