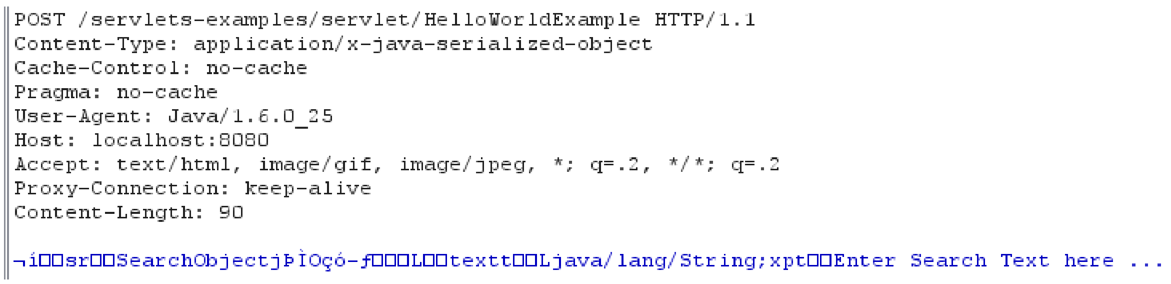
Pentesting Java thick applications with Burp JDSer

Recently I stumbled upon a Java Rich Client pentest project. Fortunately, the communication was made via HTTP, so it was possible to manipulate requests and response with our favorite tool, Burp.

Unfortunately, the app has been transmitting data in serialized Java format. So the intercepted requests and responses look like this:



```
POST /servlets-examples/servlet/HelloWorldExample HTTP/1.1
Content-Type: application/x-java-serialized-object
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.6.0_25
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Proxy-Connection: keep-alive
Content-Length: 90

¬í□□sr□□SearchObjectjÞÌOçó-ƒ□□□L□□textt□□Ljava/lang/String;xpt□□Enter Search Text here ...
```

After a little bit of Google searching, I came across this very well-written article about Java serialization and tried out his tool: BurpDSer. Scratching my head off for a few hours installing dependencies and could not get it to work (there's some problem with IRB Shell not popping up), I began searching for alternative solutions. Luckily I found this excellent SANS blog which outline high level steps to make a Burp Deserialization plugin. So I put together a simple implementation of that idea. I hope it will be helpful for pentesters as well as developers in dealing with serialized Java applications.

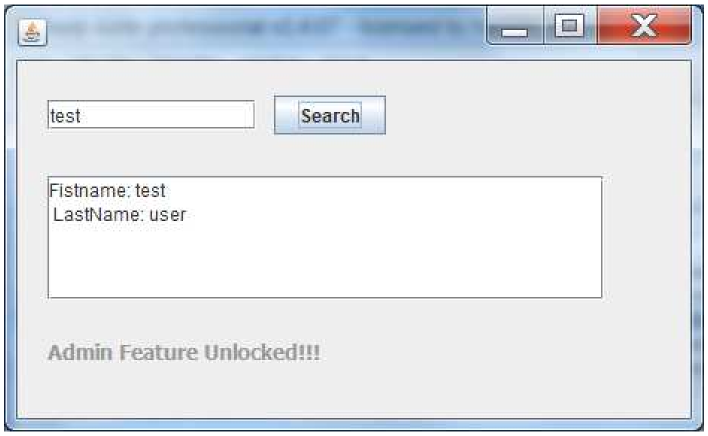
In this blog post, I will cover following information:

1. **Setup Burp proxy**
2. **Inspect Java client**
3. **Run BurpJDSer**
4. **Fuzz for server errors**
5. **Bypass client side controls**
6. **Remediation instructions**

Introduction

BurpJDSer is a Burp plugin that will deserialize/serialize Java request and response to and from XML with the help of Xtream library. BurpJDSer utilizes native Java technology to deserialize/serialize Java request, thus no additional software is required.

Let's consider this dummy Java app that communicates with a servlet via HTTP. It's a very simple search box which sends **SearchObject** to a server. Server responses with a **SearchResult** object back. If it indicates that client has admin privilege, the gray text will become red.



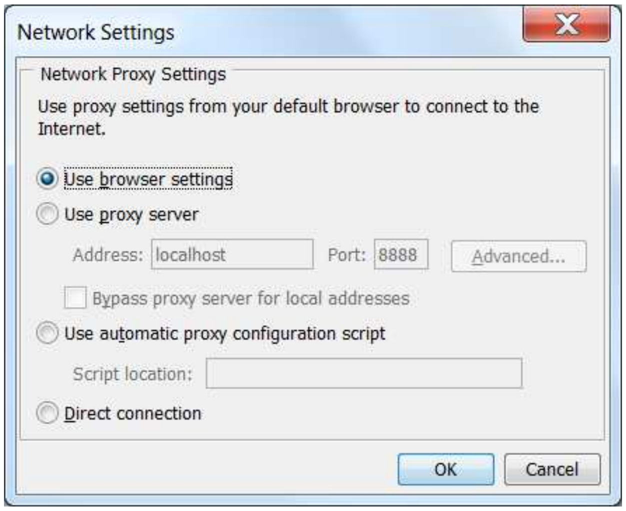
Fistname: test
LastName: user

Admin Feature Unlocked!!!

Figure 1: Sample Client

Step 1: Set up Burp proxy

- If the program is started from the command line (java –jar client.jar), add the following flags: -Dhttp.proxyHost=127.0.0.1 -Dhttp.proxyPort=<Burp port>
- If the program is started from browser (Java Web Applet), make sure JVM set to use browser proxy settings (**Windows Control Panel → Java → Network Settings**) or explicitly set to use Burp proxy.



Network Settings

Network Proxy Settings

Use proxy settings from your default browser to connect to the Internet.

- [✓ Use browser settings]
- Use proxy server
  - Address: localhost Port: 8888
  - Bypass proxy server for local addresses
- Use automatic proxy configuration script
  - Script location:
- Direct connection

Figure 2: Configure Java Network Settings on Windows

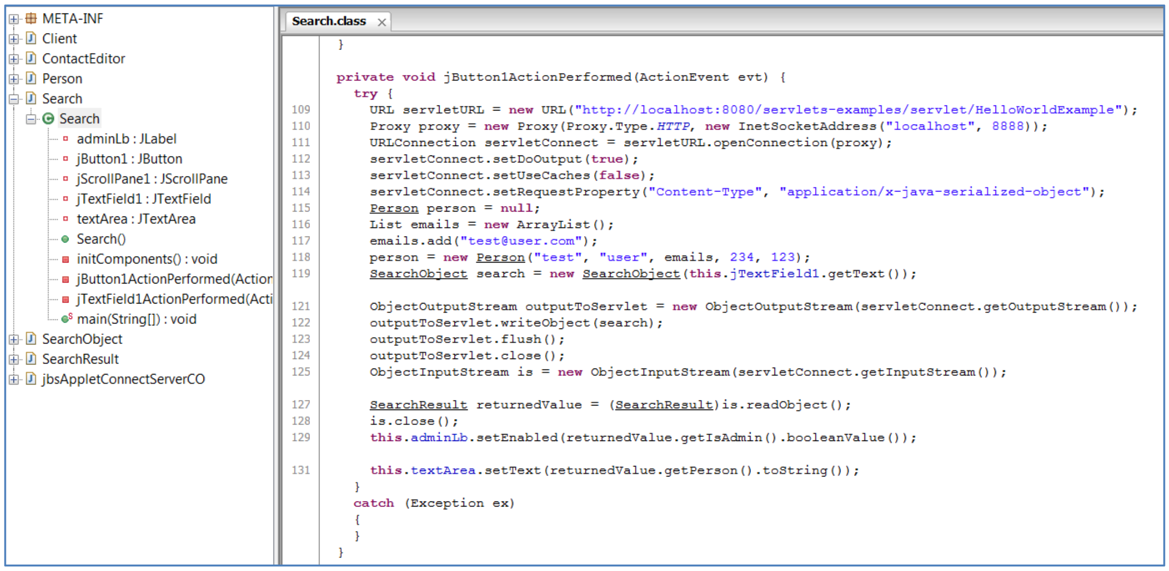
- If the program communicates via HTTPS, import PortSwigger Root Certificate to your favorite browser.

Also consider these intructions from Burp author if the above method fails to intercept HTTP traffic:

http://blog.portswigger.net/2009/04/intercepting-thick-client.html

Step 2: Download and Inspect Client jar

- Download the Java client to your computer. This can be done viewing HTML response from the page that loads Java applet. A sample applet tag that reveals location of the client jar file:
  <applet code="com.example.client" archive="SerializedClient.jar"/>
- Use any Java decompiler to decompile the target jar file. My favorite is JD-GUI from http://java.decompiler.free.fr/
- What to look for: any hardcoded passwords, backdoors, communication methods, SQL queries,etc.



```java
}

private void jButton1ActionPerformed(ActionEvent evt) {
  try {
    URL servletURL = new URL("http://localhost:8080/servlets-examples/servlet/HelloWorldExample");
    Proxy proxy = new Proxy(Proxy.Type.HTTP, new InetSocketAddress("localhost", 8888));
    URLConnection servletConnect = servletURL.openConnection(proxy);
    servletConnect.setDoOutput(true);
    servletConnect.setUseCaches(false);
    servletConnect.setRequestProperty("Content-Type", "application/x-java-serialized-object");
    Person person = null;
    List emails = new ArrayList();
    emails.add("test@user.com");
    person = new Person("test", "user", emails, 234, 123);
    SearchObject search = new SearchObject(this.jTextField1.getText());

    ObjectOutputStream outputToServlet = new ObjectOutputStream(servletConnect.getOutputStream());
    outputToServlet.writeObject(search);
    outputToServlet.flush();
    outputToServlet.close();
    ObjectInputStream is = new ObjectInputStream(servletConnect.getInputStream());

    SearchResult returnedValue = (SearchResult)is.readObject();
    is.close();
    this.adminLb.setEnabled(returnedValue.getIsAdmin().booleanValue());

    this.textArea.setText(returnedValue.getPerson().toString());
  }
  catch (Exception ex)
  {
  }
}
```

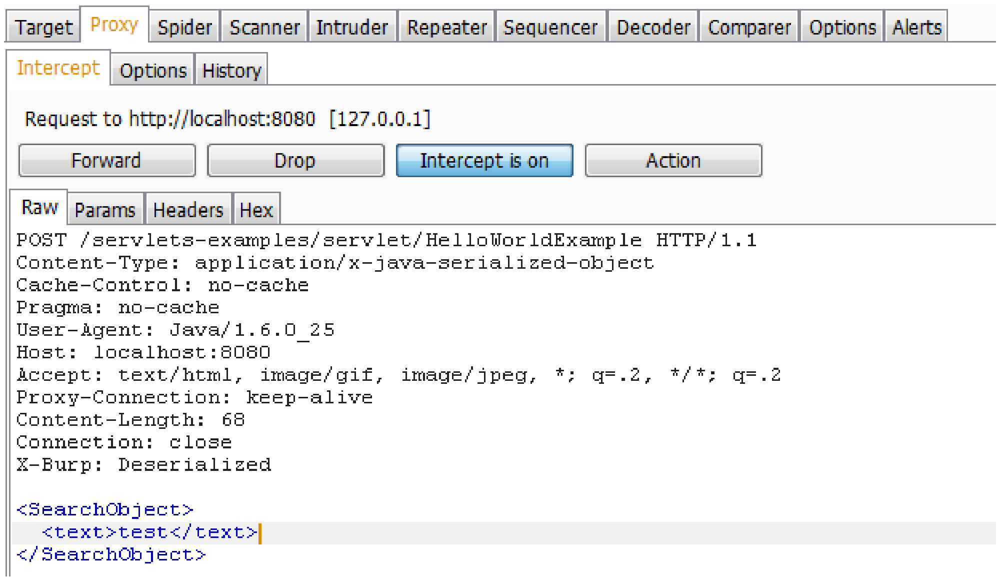
Figure 3: Static code analysis with JD-GUI

Step 3: Run the BurpJDSer plugin

- Download all the executables here: https://github.com/khai-tran/BurpJDSer/tree/master/executables
- Run this command:
  java -classpath burp.jar;burpjdser.jar;xstream-1.4.2.jar;**<client_jar>** burp.StartBurp

Step 4: Inspecting traffic

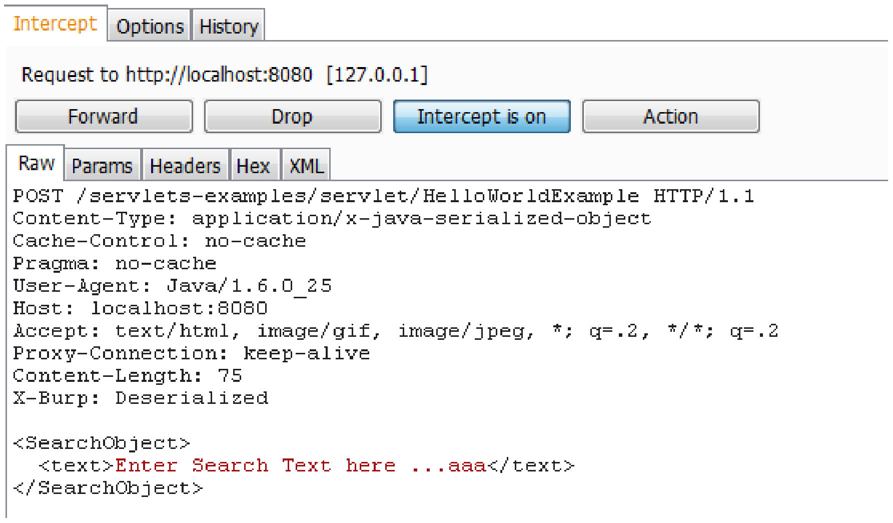
- When setup properly, we should see some traffic captured by Burp.
- Essentially, what the plugin does is to convert serialized request to XML for you to modify, than convert back from XML to serialized object before sending to the server



```
POST /servlets-examples/servlet/HelloWorldExample HTTP/1.1
Content-Type: application/x-java-serialized-object
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.6.0_25
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Proxy-Connection: keep-alive
Content-Length: 68
Connection: close
X-Burp: Deserialized

<SearchObject>
  <text>test</text>
</SearchObject>
```

Figure 4: Sample deserialized request

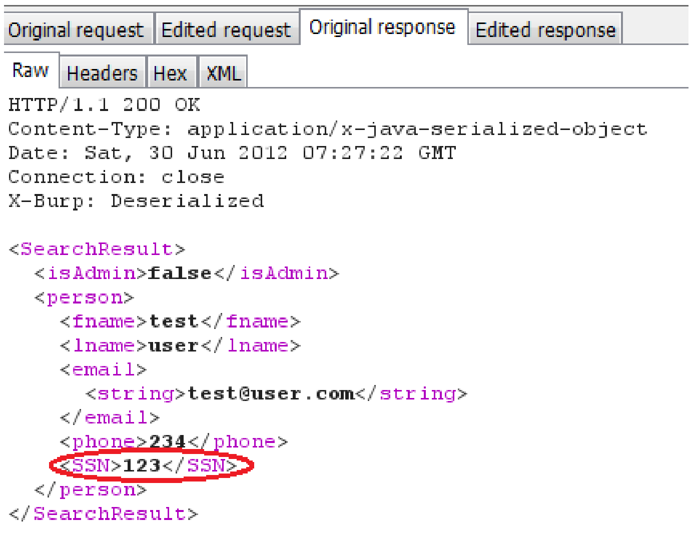
- Similarly for the response, the plugin will deserialize to XML and then serialize back to Java object so that it won't break the client. Thus you won't see pretty XML result in the response



```
POST /servlets-examples/servlet/HelloWorldExample HTTP/1.1
Content-Type: application/x-java-serialized-object
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.6.0_25
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Proxy-Connection: keep-alive
Content-Length: 75
X-Burp: Deserialized

<SearchObject>
  <text>Enter Search Text here ...aaa</text>
</SearchObject>
```

Figure 5: Sample deserialized response

- However switching to Proxy history will reveal deserialized response



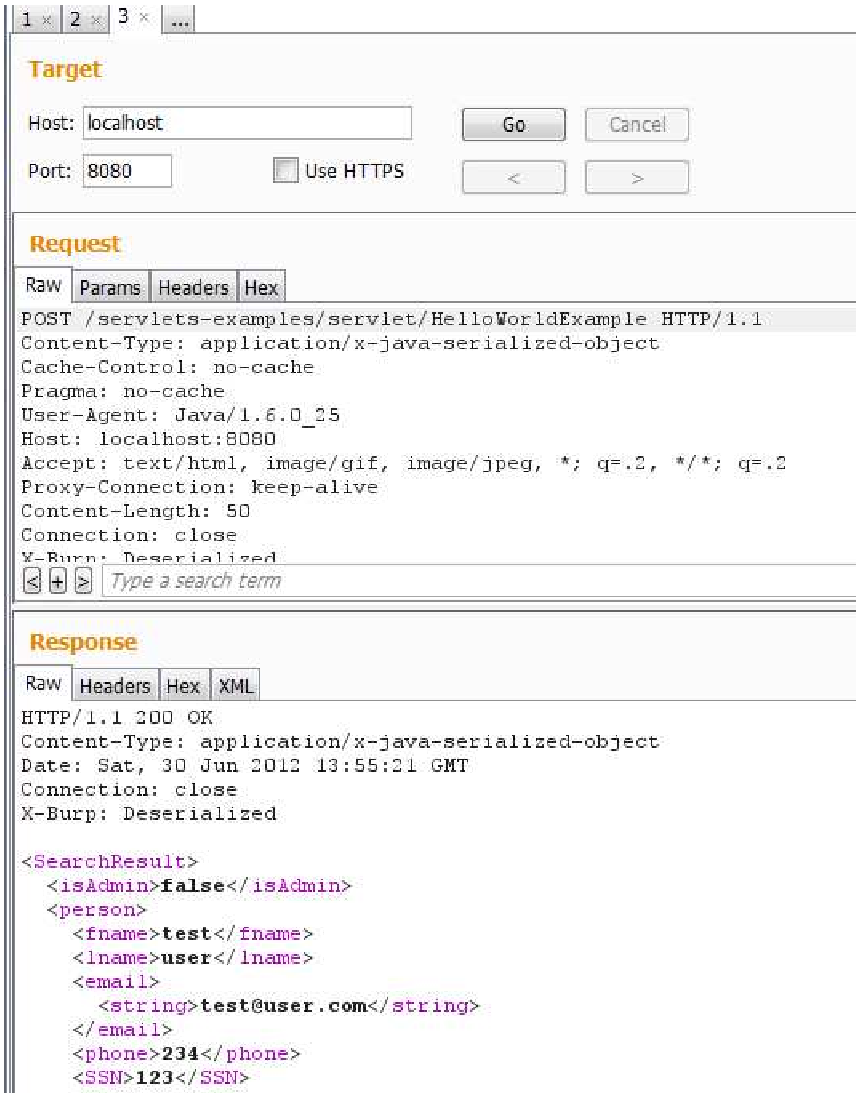
```
HTTP/1.1 200 OK
Content-Type: application/x-java-serialized-object
Date: Sat, 30 Jun 2012 07:27:22 GMT
Connection: close
X-Burp: Deserialized

<SearchResult>
  <isAdmin>false</isAdmin>
  <person>
    <fname>test</fname>
    <lname>user</lname>
    <email>
      <string>test@user.com</string>
    </email>
    <phone>234</phone>
    <SSN>123</SSN>
  </person>
</SearchResult>
```

Figure 6: Hidden SSN passed in XML response

- In this example, SSN is included in the response, but not shown in the client.

Step 5: Fuzzing for server error with Repeater and Intruder

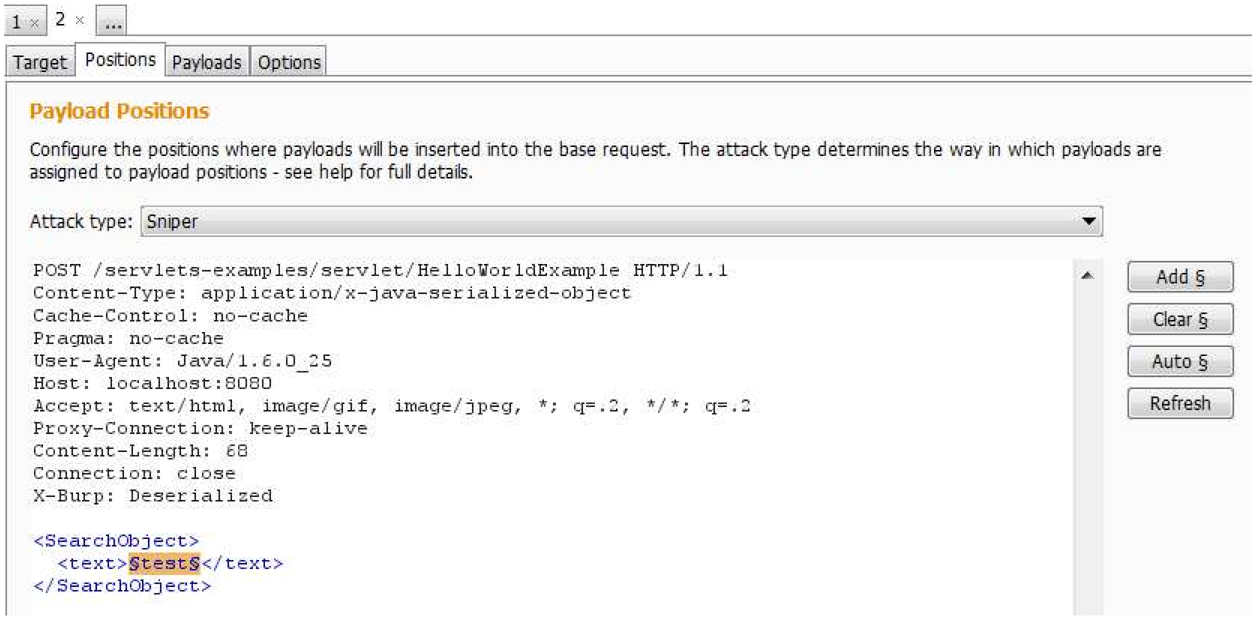
- This plugin will also perform conversion of Java object when processed in Burp Repeater/Intruder for your own convenience
- Fuzz for server vulnerabilities such as SQL Injection, Command Injection, Authorization Bypass, etc.



```
POST /servlets-examples/servlet/HelloWorldExample HTTP/1.1
Content-Type: application/x-java-serialized-object
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.6.0_25
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Proxy-Connection: keep-alive
Content-Length: 50
Connection: close
X-Burp: Deserialized
```

```
HTTP/1.1 200 OK
Content-Type: application/x-java-serialized-object
Date: Sat, 30 Jun 2012 13:55:21 GMT
Connection: close
X-Burp: Deserialized

<SearchResult>
  <isAdmin>false</isAdmin>
  <person>
    <fname>test</fname>
    <lname>user</lname>
    <email>
      <string>test@user.com</string>
    </email>
    <phone>234</phone>
    <SSN>123</SSN>
```

Figure 7: Fuzzing with Burp Repeater



Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

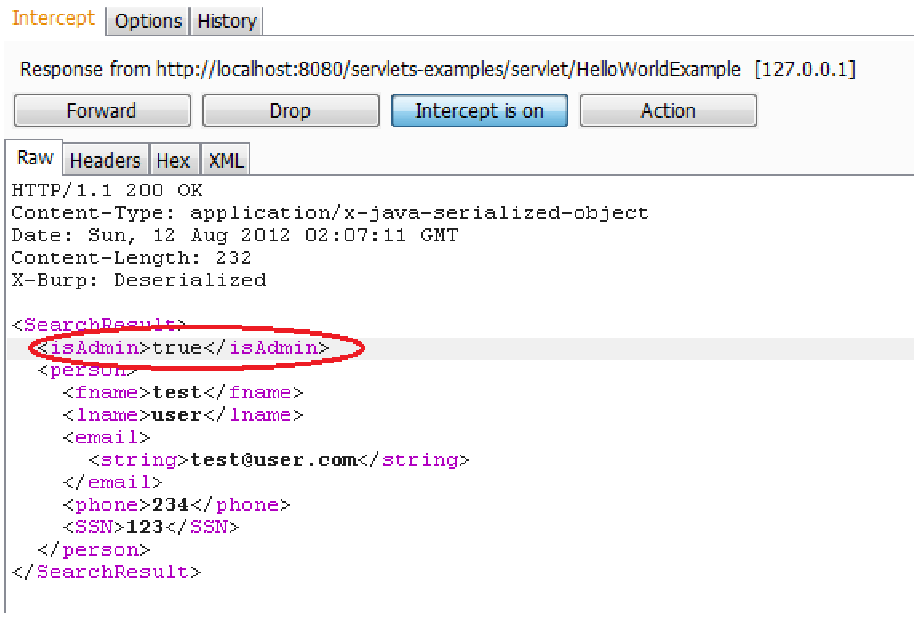
```
POST /servlets-examples/servlet/HelloWorldExample HTTP/1.1
Content-Type: application/x-java-serialized-object
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.6.0_25
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Proxy-Connection: keep-alive
Content-Length: 68
Connection: close
X-Burp: Deserialized

<SearchObject>
  <text>§test§</text>
</SearchObject>
```

Figure 8: Fuzzing with Burp Intruder

Step 6: Test for client-side authorization bypass

The plugin also has support for serializing requests/responses from XML to Java format. This may come in handy in case you need to bypass client check or enable hidden features of the client. Below is an example of how to do this

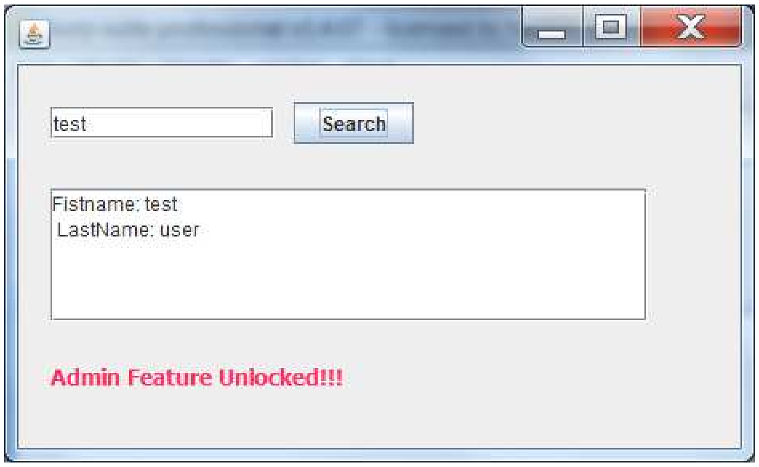
- Intercept server response
- Find and modify hidden parameter to access hidden privileges. In this example, I set isAdmin parameter to 'true' in order to bypass client-side restriction



Response from http://localhost:8080/servlets-examples/servlet/HelloWorldExample [127.0.0.1]

```
HTTP/1.1 200 OK
Content-Type: application/x-java-serialized-object
Date: Sun, 12 Aug 2012 02:07:11 GMT
Content-Length: 232
X-Burp: Deserialized

<SearchResult>
  <isAdmin>true</isAdmin>
  <person>
    <fname>test</fname>
    <lname>user</lname>
    <email>
      <string>test@user.com</string>
    </email>
    <phone>234</phone>
    <SSN>123</SSN>
  </person>
</SearchResult>
```

Figure 9: Intercept & modify Burp response



Fistname: test
LastName: user

Admin Feature Unlocked!!!

Figure 10: Voila! Admin feature is now unlocked

Remediation path

- Don't rely on Java serialization as a method of encryption
- Don't include redundant data in the response, even if it's not displayed on client GUI
- Obfuscate your code: choose a obfuscator that also encrypts String constants such as ZelixClassMaster
- Validate input serialized data
- Consider sealing and signing serialized objects with **javax.crypto.SealedObject**

References

The source code and executable are available at: https://github.com/khai-tran/BurpJDSer. Feel free to leave comments there. Thanks Scott and Antti for your feedbacks on the tool.

You may also want to check out other tools of the same category: JAVA Snoop, BurpDSer.

http://xstream.codehaus.org/

http://pen-testing.sans.org/blog/2011/10/18/tips-for-fat-client-web-app-and-mobile-pen-testing-serialized-object-communication-using-the-burp-suite

http://www.ibm.com/developerworks/java/library/j-5things1/index.html

Happy hacking!