Defending against Java Deserialization Vulnerabilities

Bay Area OWASP Meetup - September 2016

Luca Carettoni - @lucacarettoni

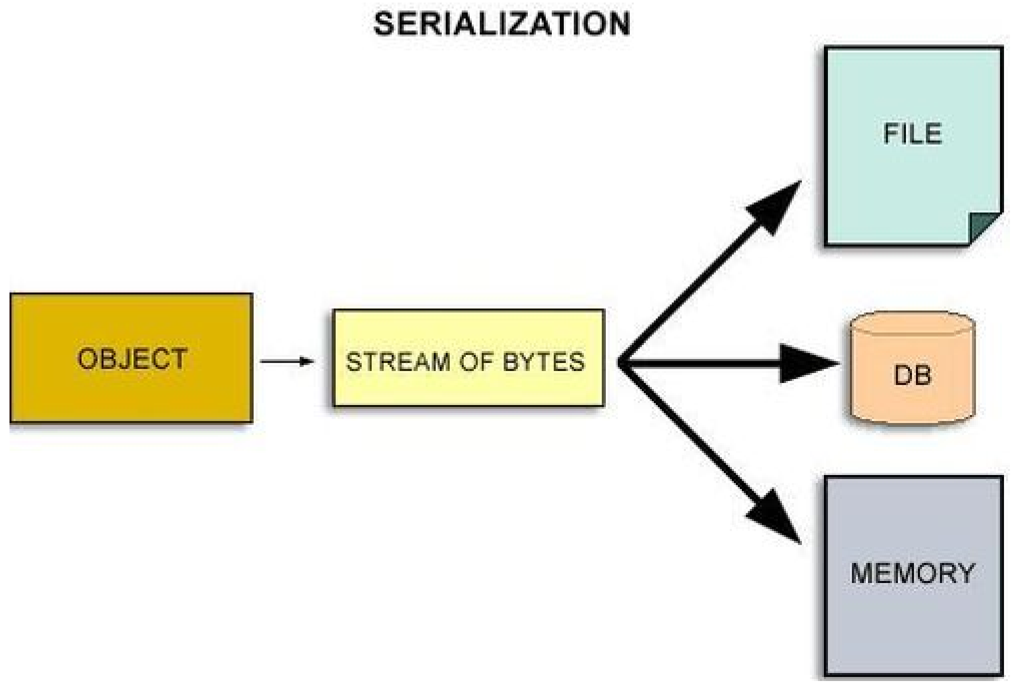
Agenda

This talk is about defense and how to protect your application against this ~~new~~ old class of vulnerabilities.

- Intro to Java Deserialization bugs
- A real-life bug (*SJWC serialized object injection via JSF view state*)
- Discovery
- Defense

Intro to Java Deserialization bugs

From object graph data to byte stream



Serialization in Code

```
//Instantiate the Serializable class
String myPreso = “OWASP Bay Area”;


// Write to disk
FileOutputStream fileOut = new FileOutputStream("serial.data");

// Write object
ObjectOutputStream objOut = new ObjectOutputStream (fileOut);
objOut.writeObject(myPreso);
```

Deserialization in Code

```
// Read from disk
FileInputStream fileIn = new FileInputStream("serial.data");

// Read object
ObjectInputStream objIn = new ObjectInputStream (fileIn);
String myPreso = (String) objIn.readObject();
```

Deserialization in Bytecode

```
[...]

aload ois

invokevirtual Object ObjectInputStream.readObject()

checkcast String

[...]
```

No type safety

Any Serializable class will work until this point.

Callback methods

- Developers can override the following methods to customize the deserialization process
    - **readObject()**
    - **readResolve()**
    - **readObjectNoData()**
    - **validateObject()**
    - **finalize()** Invoked by the Garbage Collector

What if….

1. A remote service accepts Java serialized objects
2. In the classpath of the remote application, there are unrelated classes that are Serializable AND implement one of the callbacks
3. The callback’s method implements something interesting*

* File I/O operations, system commands, socket operations, etc.

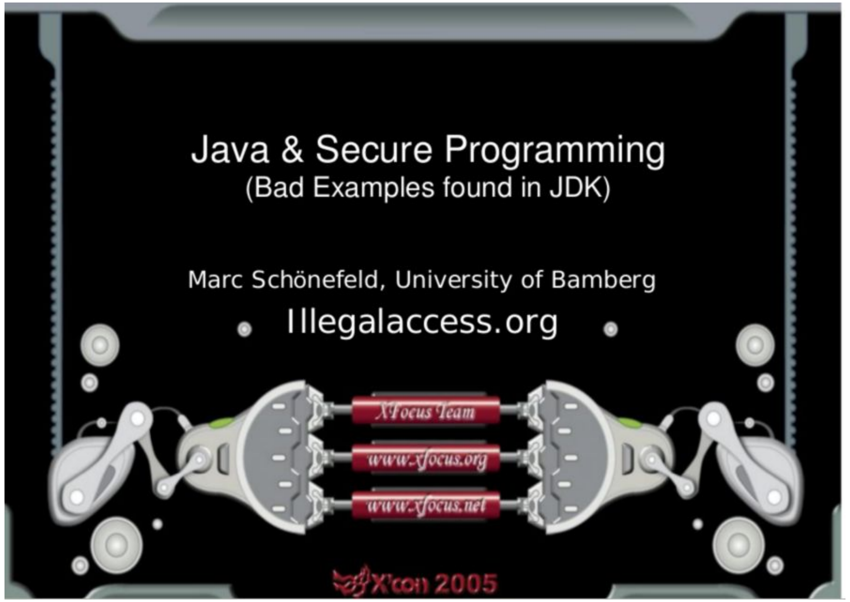
Unlikely?

```
//org.apache.commons.fileupload.disk.DiskFileItem
Private void readObject(ObjectInputStream in) {
    in.defaultReadObject();

    OutputStream output = getOutputStream();
..
    FileInputStream input = new FileInputStream(dfosFile);
    IOUtils.copy(input, output);
    dfosFile.delete();
..
```

The Forgotten Bug Class @matthias_kaiser™

2005 - Marc Schonefeld

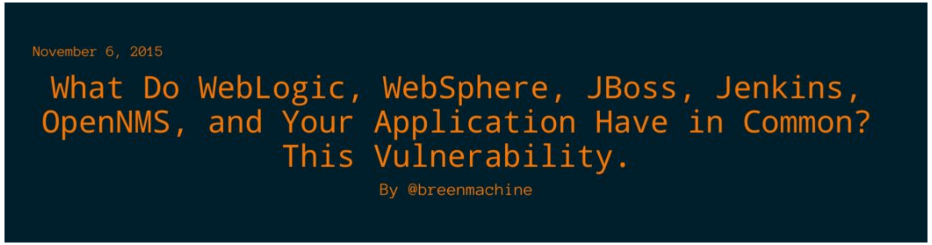


Java & Secure Programming
(Bad Examples found in JDK)

Marc Schönefeld, University of Bamberg
Illegalaccess.org

XFocus Team
www.xfocus.org
www.xfocus.net
X'con 2005

2015 - Steve Breen



November 6, 2015

What Do WebLogic, WebSphere, JBoss, Jenkins, OpenNMS, and Your Application Have in Common? This Vulnerability.

By @breenmachine

And many more...

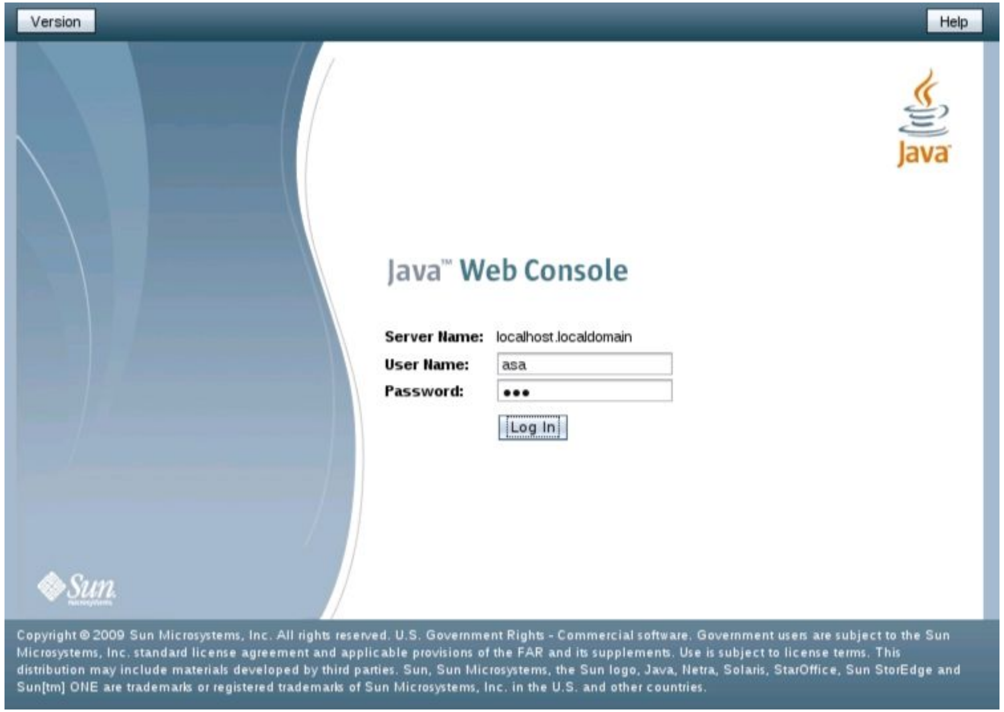
A real-life bug, back from 2010

Sun Java Web Console serialized object injection via JSF view state

Sun Java Web Console

README - SJWC_3.0

“The Sun Java (TM) Web Console is a web application that provides a single point on entry for many of Sun's systems management applications. The console application provides a single-sign on capability and a secure home page for many of Solaris”



Server Name: localhost.localdomain
User Name: asa
Password: •••

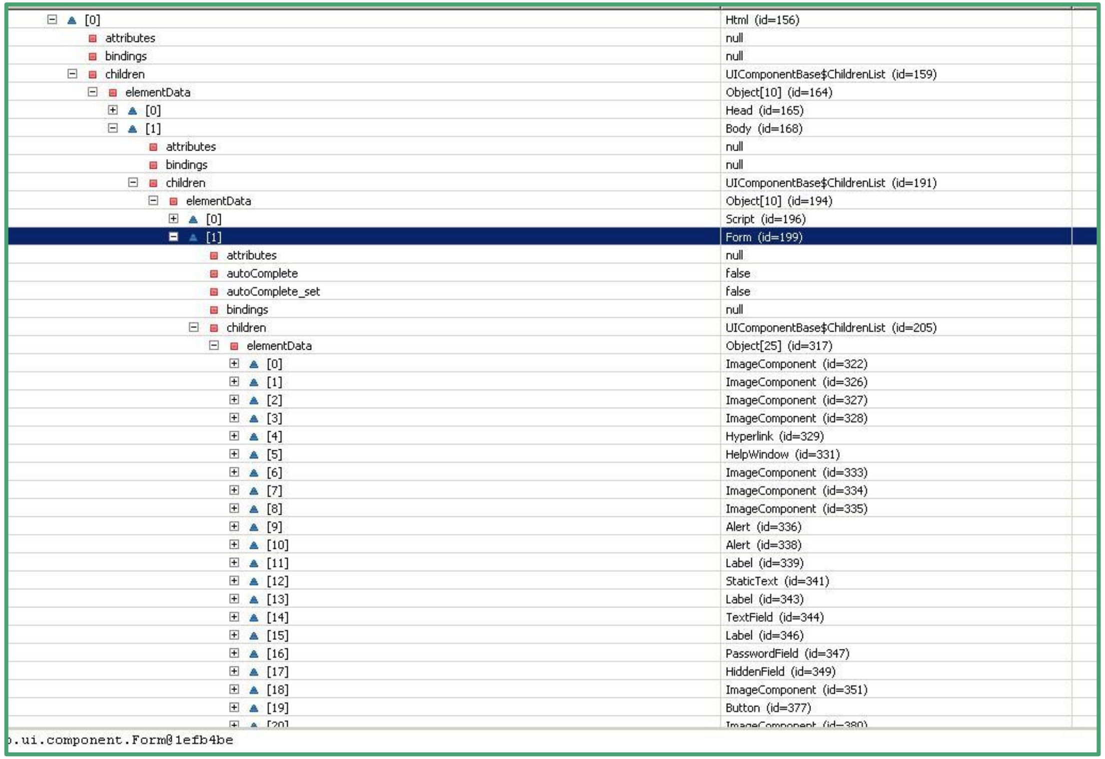
Copyright © 2009 Sun Microsystems, Inc. All rights reserved. U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements. Use is subject to license terms. This distribution may include materials developed by third parties. Sun, Sun Microsystems, the Sun logo, Java, Netra, Solaris, StarOffice, Sun StorEdge and Sun[tm] ONE are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

JSF ViewState

- JSF ViewState uses Java deserialization to restore the UI state

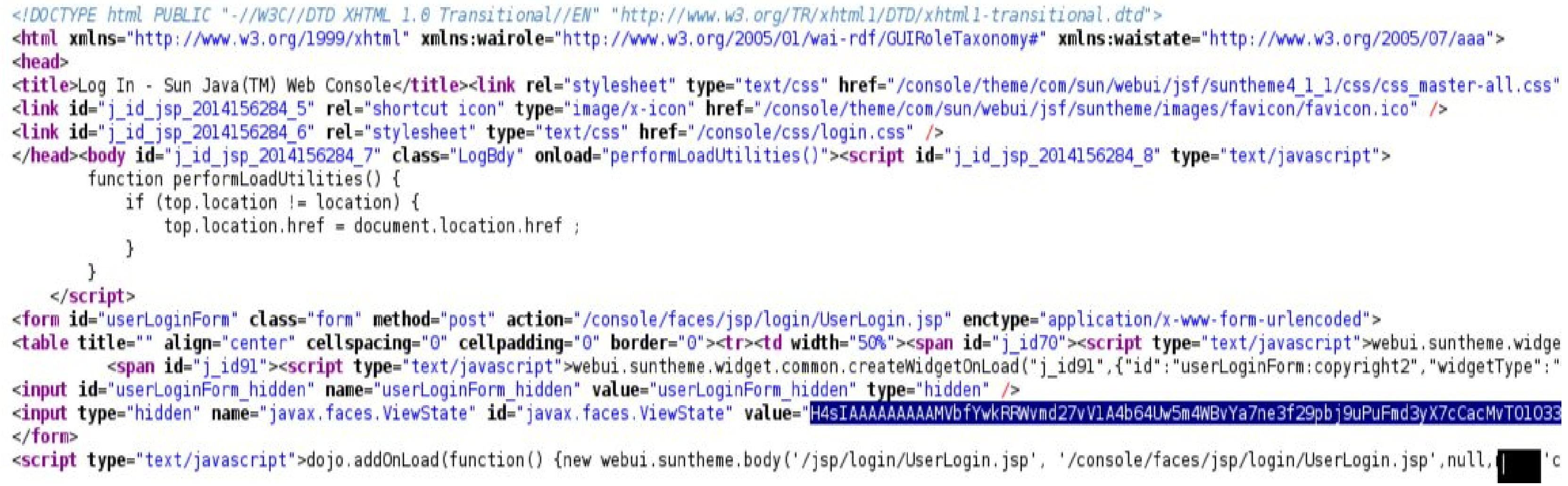
HTML Page

```
<form>
<input type="hidden"
name="javax.faces.ViewState"
value=
</form>
```



```
[0]                                          Html (id=156)
  attributes                                 null
  bindings                                   null
  children                                   UIComponentBase$ChildrenList (id=159)
    elementData                              Object[10] (id=164)
      [0]                                    Head (id=165)
      [1]                                    Body (id=168)
        attributes                           null
        bindings                             null
        children                             UIComponentBase$ChildrenList (id=191)
          elementData                        Object[10] (id=194)
            [0]                              Script (id=196)
            [1]                              Form (id=199)
              attributes                     null
              autoComplete                   false
              autoComplete_set               false
              bindings                       null
              children                       UIComponentBase$ChildrenList (id=205)
                elementData                  Object[25] (id=317)
                  [0]                        ImageComponent (id=322)
                  [1]                        ImageComponent (id=326)
                  [2]                        ImageComponent (id=327)
                  [3]                        ImageComponent (id=328)
                  [4]                        Hyperlink (id=329)
                  [5]                        HelpWindow (id=331)
                  [6]                        ImageComponent (id=333)
                  [7]                        ImageComponent (id=334)
                  [8]                        ImageComponent (id=335)
                  [9]                        Alert (id=336)
                  [10]                       Alert (id=338)
                  [11]                       Label (id=339)
                  [12]                       StaticText (id=341)
                  [13]                       Label (id=343)
                  [14]                       TextField (id=344)
                  [15]                       Label (id=346)
                  [16]                       PasswordField (id=347)
                  [17]                       HiddenField (id=349)
                  [18]                       ImageComponent (id=351)
                  [19]                       Button (id=377)
                  [20]                       ImageComponent (id=380)
.ui.component.Form@1efb4be
```

Sun Java Web Console - Login Page ViewState



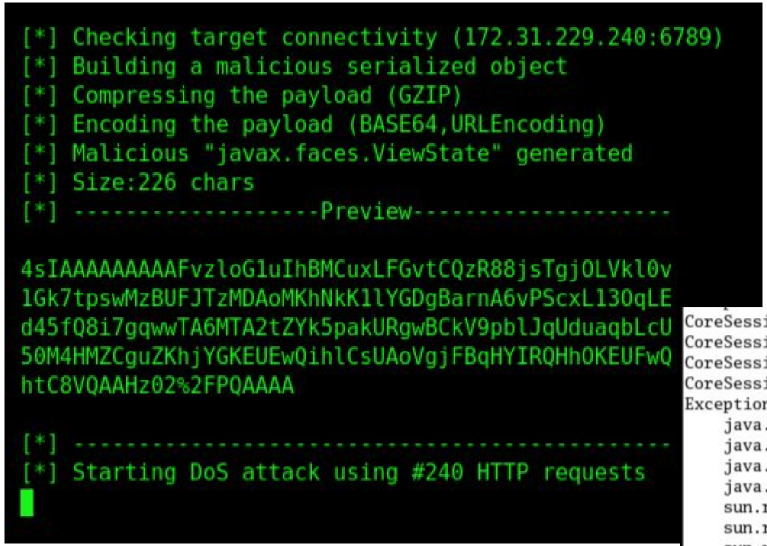
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:wairole="http://www.w3.org/2005/01/wai-rdf/GUIRoleTaxonomy#" xmlns:waistate="http://www.w3.org/2005/07/aaa">
<head>
<title>Log In - Sun Java(TM) Web Console</title><link rel="stylesheet" type="text/css" href="/console/theme/com/sun/webui/jsf/suntheme4_1_1/css/css_master-all.css"
<link id="j_id_jsp_2014156284_5" rel="shortcut icon" type="image/x-icon" href="/console/theme/com/sun/webui/jsf/suntheme/images/favicon/favicon.ico" />
<link id="j_id_jsp_2014156284_6" rel="stylesheet" type="text/css" href="/console/css/login.css" />
</head><body id="j_id_jsp_2014156284_7" class="LogBdy" onload="performLoadUtilities()"><script id="j_id_jsp_2014156284_8" type="text/javascript">
        function performLoadUtilities() {
            if (top.location != location) {
                top.location.href = document.location.href ;
            }
        }
    </script>
<form id="userLoginForm" class="form" method="post" action="/console/faces/jsp/login/UserLogin.jsp" enctype="application/x-www-form-urlencoded">
<table title="" align="center" cellspacing="0" cellpadding="0" border="0"><tr><td width="50%"><span id="j_id70"><script type="text/javascript">webui.suntheme.widge
          <span id="j_id91"><script type="text/javascript">webui.suntheme.widget.common.createWidgetOnLoad("j_id91",{"id":"userLoginForm:copyright2","widgetType":"
<input id="userLoginForm_hidden" name="userLoginForm_hidden" value="userLoginForm_hidden" type="hidden" />
<input type="hidden" name="javax.faces.ViewState" id="javax.faces.ViewState" value="H4sIAAAAAAAAAMVbfYwkRRWvmd27vV1A4b64Uw5m4WBvYa7ne3f29pbj9uPuFmd3yX7cCacMvTO1033
</form>
<script type="text/javascript">dojo.addOnLoad(function() {new webui.suntheme.body('/jsp/login/UserLogin.jsp', '/console/faces/jsp/login/UserLogin.jsp',null,[redacted]'c
```

- ViewState saved client-side only
  - javax.faces.STATE_SAVING_METHOD=”client” before SJWC < 3.1
- No encryption

A good bug

- Attractive target, as SJWC was the admin web interface for Solaris
- At the time of discovery (Jan 2010), I created a Proof-of-Concept using a known gadget based on Hashtable collisions (Crosby & Wallach, 2003)
    - https://www.ikkisoft.com/stuff/SJWC_DoS.java
- Back then, I had no idea about the infamous Apache Common Collections gadget (Gabriel Lawrence, Chris Frohoff)
    - /opt/sun/webconsole/private/container/shared/lib/commons-collections.jar
- However, I was able to leverage an Expression Language (EL) Injection-like to perform arbitrary file read
- Soon after, SJWC started using server-side ViewState
    - “Beware of Serialized GUI Objects Bearing Data” July 2010, Black Hat Vegas

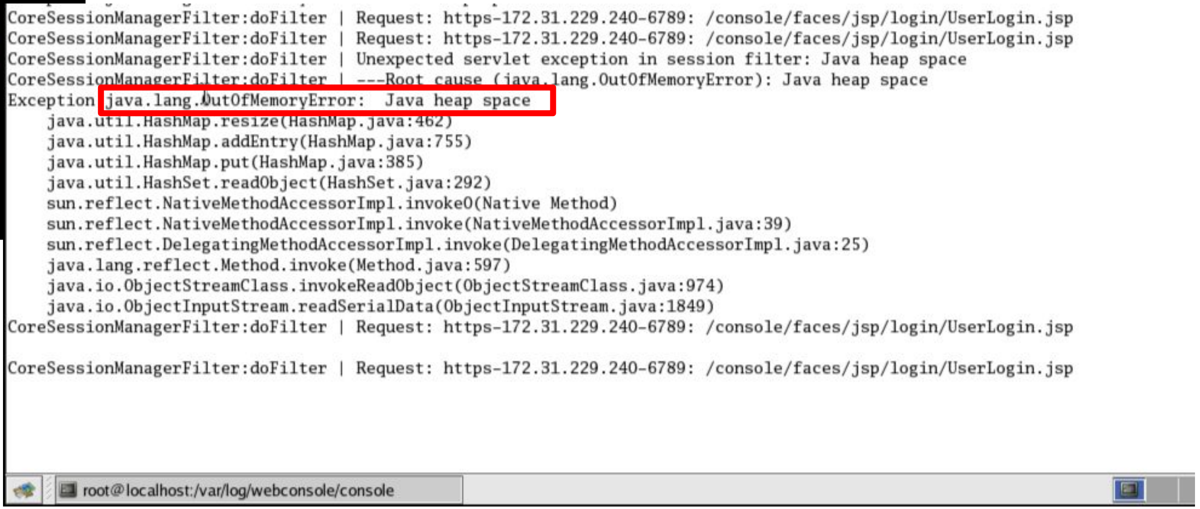
In practice



```
[*] Checking target connectivity (172.31.229.240:6789)
[*] Building a malicious serialized object
[*] Compressing the payload (GZIP)
[*] Encoding the payload (BASE64,URLEncoding)
[*] Malicious "javax.faces.ViewState" generated
[*] Size:226 chars
[*] -------------------Preview--------------------

4sIAAAAAAAAAFvzloG1uIhBMCuxLFGvtCQzR88jsTgjOLVkl0v
1Gk7tpswMzBUFJTzMDAoMKhNkK1lYGDgBarnA6vPScxL13OqLE
d45fQ8i7gqwwTA6MTA2tZYk5pakURgwBCkV9pblJqUduaqbLcU
50M4HMZCguZKhjYGKEUEwQihlCsUAoVgjFBqHYIRQHhOKEUFwQ
htC8VQAAHz02%2FPQAAAA

[*] ----------------------------------------------
[*] Starting DoS attack using #240 HTTP requests
```



```
CoreSessionManagerFilter:doFilter | Request: https-172.31.229.240-6789: /console/faces/jsp/login/UserLogin.jsp
CoreSessionManagerFilter:doFilter | Request: https-172.31.229.240-6789: /console/faces/jsp/login/UserLogin.jsp
CoreSessionManagerFilter:doFilter | Unexpected servlet exception in session filter: Java heap space
CoreSessionManagerFilter:doFilter | ---Root cause (java.lang.OutOfMemoryError): Java heap space
Exception java.lang.OutOfMemoryError:  Java heap space
    java.util.HashMap.resize(HashMap.java:462)
    java.util.HashMap.addEntry(HashMap.java:755)
    java.util.HashMap.put(HashMap.java:385)
    java.util.HashSet.readObject(HashSet.java:292)
    sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    java.lang.reflect.Method.invoke(Method.java:597)
    java.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:974)
    java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1849)
CoreSessionManagerFilter:doFilter | Request: https-172.31.229.240-6789: /console/faces/jsp/login/UserLogin.jsp

CoreSessionManagerFilter:doFilter | Request: https-172.31.229.240-6789: /console/faces/jsp/login/UserLogin.jsp
```

Discovery

Code Review - Entry Points

Look for occurrences of:

- java.io.ObjectInputStream.**readObject()**
- java.io.ObjectInputStream.**readUnshared()**

And perform manual review to determine whether they use user-supplied data

```
$ egrep -r "readObject\(|readUnshared\("
```

Code Review - Gadgets

- This is the interesting (and complex) part of exploiting Java deserialization vulnerabilities
- As a defender, assume that there are multiple game-over gadgets available in the classpath
    - For example, SJWC uses 58 dependency JARs

- If you want to learn more on how to discover gold and silver gadgets:
    - Marshalling Pickles - Gabriel Lawrence, Chris Frohoff
    - Java Deserialization Vulnerabilities, The Forgotten Bug Class - Matthias Kaiser
    - Surviving the Java serialization apocalypse - Alvaro Muñoz, Christian Schneider
    - Ysoserialpayloads - https://github.com/frohoff/ysoserial/tree/master/src/main/java/ysoserial/payloads

Discovery with no code...

- Decompile :)
- Magic bytes in the network traffic
  - 0xAC 0xED
  - rO0
  - FvzFgDff9
  - ...
- Passive and active tools
  - https://github.com/DirectDefense/SuperSerial
  - https://github.com/johndekroon/serializekiller
  - <--ADD your favourite web scanner vendor HERE-->

Defense

Things that do NOT work

- Patching Apache Commons
- Removing dependencies from the classpath
- Black-listing only
- Using a short-lived Java Security Manager during deserialization

Your best option. All other mitigations are suboptimal.

Do not use serialization when receiving untrusted data.

It’s 2016, there are better options.

Option #1 - Add authentication

- Add a layer of authentication to ensure that Java serialization can be invoked by trusted parties only
  - At the network layer, using client-side TLS certs
  - At the application layer, encryption/signing of the payload

Pro	Cons
● Network layer solutions can be implemented with no application changes (e.g. stunnel)	● Additional operational complexity ● If enc/dec is implemented by the application, secure keys management is crucial ● Trusted parties can still abuse the application

Option #2 - Use Java Agent-based solutions

- Install a Java Agent solution to perform JVM-wide validation (blacklisting/whitelisting)
  - https://github.com/Contrast-Security-OSS/contrast-rO0
  - https://github.com/kantega/notsoserial

Pro	**Cons**
● No application changes ● Easy to deploy and use	● Performance hit ● In certain environment, not usable (e.g. software engineer with no access to the underlying JVM container)

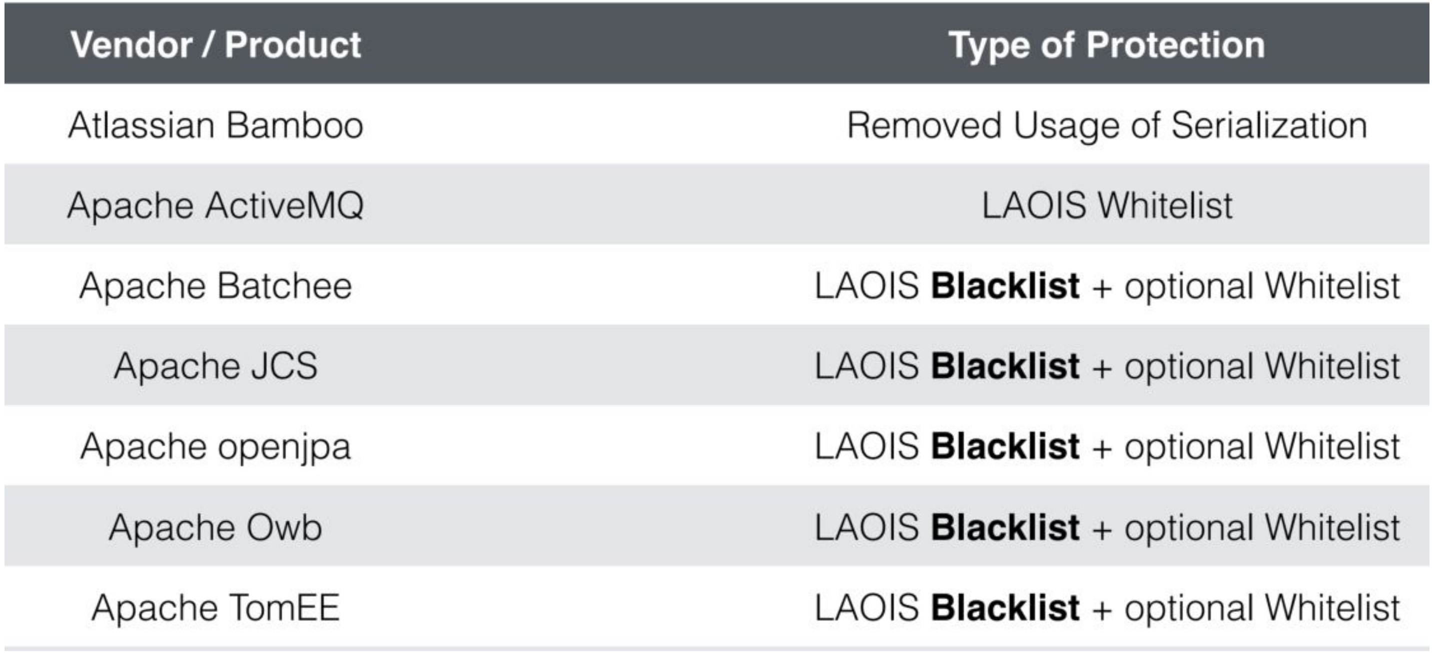
Option #3 - Use safe ObjectInputStream implementation

- Replace calls to ObjectInputStream with calls to a safe implementation
  - Based on look-ahead techniques
  - https://github.com/ikkisoft/SerialKiller
  - https://github.com/Contrast-Security-OSS/contrast-rO0 (SafeObjectInputStream)

Pro	Cons
● Full control for developers	● Requires re-factoring ● To be bulletproof*, whitelisting must be used (which requires profiling, good understanding of the app)

* Still affected by DoS gadgets

Mitigations in real-life



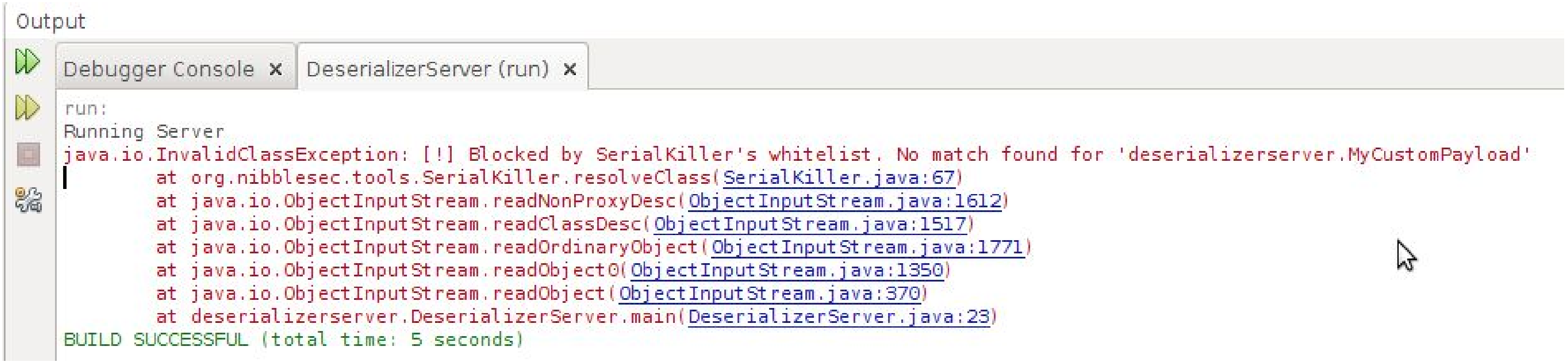
Vendor / Product	Type of Protection
Atlassian Bamboo	Removed Usage of Serialization
Apache ActiveMQ	LAOIS Whitelist
Apache Batchee	LAOIS **Blacklist** + optional Whitelist
Apache JCS	LAOIS **Blacklist** + optional Whitelist
Apache openjpa	LAOIS **Blacklist** + optional Whitelist
Apache Owb	LAOIS **Blacklist** + optional Whitelist
Apache TomEE	LAOIS **Blacklist** + optional Whitelist

Full credit to Alvaro Muñoz and Christian Schneider

SerialKiller

SerialKiller is an easy-to-use look-ahead Java deserialization library to secure application from untrusted input.

https://github.com/ikkisoft/SerialKiller



```
run:
Running Server
java.io.InvalidClassException: [!] Blocked by SerialKiller's whitelist. No match found for 'deserializerserver.MyCustomPayload'
	at org.nibblesec.tools.SerialKiller.resolveClass(SerialKiller.java:67)
	at java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1612)
	at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1517)
	at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1771)
	at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1350)
	at java.io.ObjectInputStream.readObject(ObjectInputStream.java:370)
	at deserializerserver.DeserializerServer.main(DeserializerServer.java:23)
BUILD SUCCESSFUL (total time: 5 seconds)
```

How to protect your application with SerialKiller

1. Download the latest version of the SerialKiller's Jar
   a. This library is also available on Maven Central
2. Import SerialKiller's Jar in your project
3. Replace your deserialization ObjectInputStream with SerialKiller
4. Tune the configuration file, based on your application requirements

In practice 1/2

```
// Read from disk
FileInputStream fileIn = new FileInputStream("serial.data");

// Read object
ObjectInputStream objIn = new ObjectInputStream (fileIn);
String myPreso = (String) objIn.readObject();
```

In practice 2/2

```
// Read from disk
FileInputStream fileIn = new FileInputStream("serial.data");

// Read object
ObjectInputStream objIn = new  SerialKiller(fileIn, "/etc/sk.conf");
String myPreso = (String) objIn.readObject();
```

SK’s configuration 1/2

SerialKiller config supports the following settings:

- **Refresh:** The refresh delay in milliseconds, used to hot-reload the configuration file
- **BlackList:** A Java regex to define malicious classes
  - Provides a default configuration against known gadgets
- **WhiteList:** A Java regex to define classes used by your application
- **Profiling:** To trace classes being deserialized
- **Logging**: Java’s core logging facility

SK’s configuration 2/2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <refresh>6000</refresh>
    <mode>
       <profiling>true</profiling>
    </mode>
    <logging>
       <enabled>true</enabled>
       <logfile>/tmp/serialkiller.log</logfile>
    </logging>
    <blacklist>
       [...]
       <!-- ysoserial's Spring1 payload  -->
       <regexp>org\.springframework\.beans\.factory\.ObjectFactory$</regexp>
    </blacklist>
    <whitelist>
       <regexp>.*</regexp>
    </whitelist>
</config>
```

SerialKiller v0.4 Demo

Thanks!

Luca Carettoni - @lucacarettoni