Remote code execution via Java native deserialization

Introduction

- I am not a pen tester. High school dropout, no formal training or education in security.
- Software engineer for 17 years, climatology domain
- Last 5 years focusing on security, mainly Java
- Managed Red Hat's Java middleware security team
- Now an engineering manager for a SDN company
- I love finding new 0day and popping shells!

Outline

- Java (de)serialization
- RCE via XML deserialization
- RCE via native deserialization
- RCE via XML <-> binary mapping vector
- Other InvocationHandlers?
- “Property-oriented programming” and gadgets
- Where lies the vulnerability?

Java (de)serialization

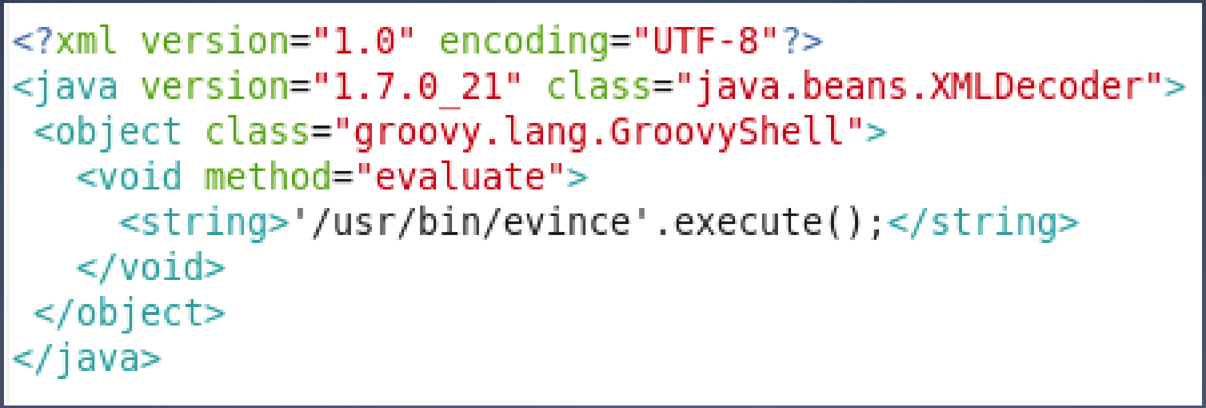
- Java has multiple serialization implementations
- XML serialization: XXE and RCE possible in multiple implementations
- Native serialization: binary data format, with RCE possible depending on what's on the classpath
- Dozer, Kryo and other frameworks
- Common thread: don't deserialize untrusted input (duh!)

RCE – XML deserialization

- Alternative XML-based serialization formats
- JAXB is the standard (no known flaws)
- Other XML serialization libraries exist, and have exposed security issues leading to RCE
- These are commonly used by big applications and XML REST API frameworks
- We'll look at just two examples: XMLDecoder and XStream
- **NOT** reliant on classes implementing Serializable

XMLDecoder

- XMLDecoder's XML format can represent a series of methods that will be called to reconstruct an object
- If XMLDecoder is used to deserialize untrusted input, arbitrary code can be injected into the XML

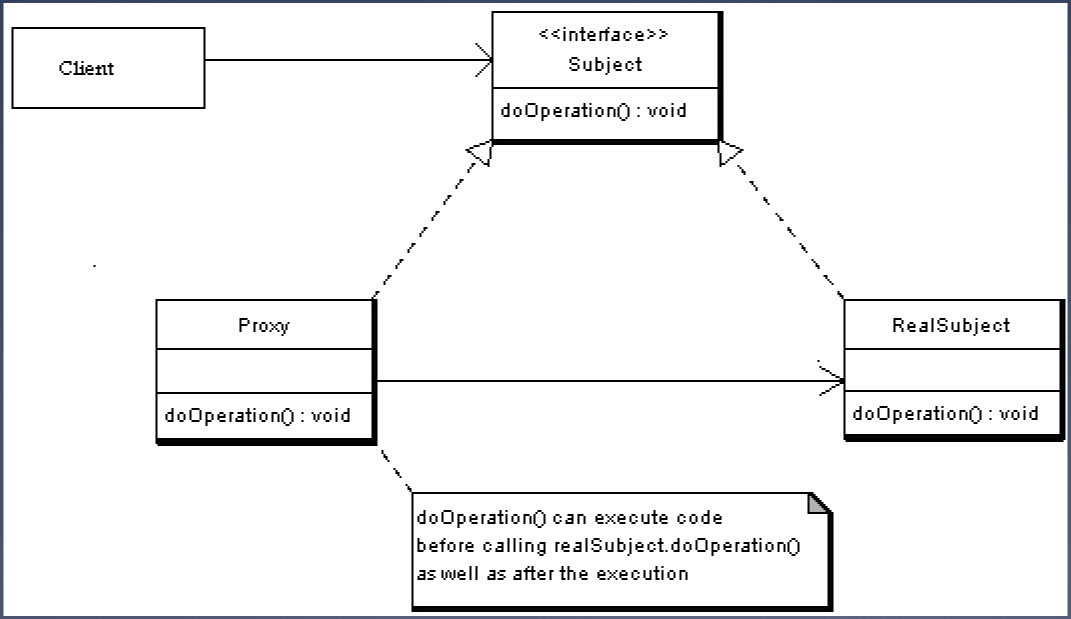


```xml
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.7.0_21" class="java.beans.XMLDecoder">
 <object class="groovy.lang.GroovyShell">
   <void method="evaluate">
     <string>'/usr/bin/evince'.execute();</string>
   </void>
 </object>
</java>
```

- Live demo: Restlet CVE-2013-4221. Fixed by removing vulnerable functionality.

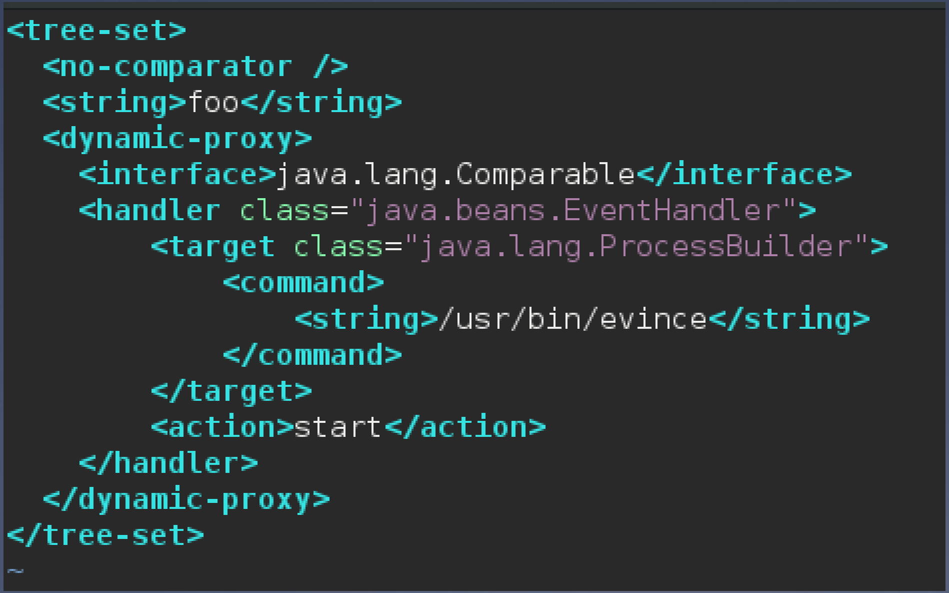
XStream

- Reflection-based deserialization
- Has a special handler for dynamic proxies (implementations of interfaces)
- Spring OXM, Sonatype Nexus, Jenkins affected



XStream

- Attackers can provide XML representing a dynamic proxy class, which implements the interface of a class the application might expect
- Dynamic proxy implements an EventHandler that calls arbitrary code when any members of the deserialized class are called



```
<tree-set>
  <no-comparator />
  <string>foo</string>
  <dynamic-proxy>
    <interface>java.lang.Comparable</interface>
    <handler class="java.beans.EventHandler">
        <target class="java.lang.ProcessBuilder">
            <command>
                <string>/usr/bin/evince</string>
            </command>
        </target>
        <action>start</action>
    </handler>
  </dynamic-proxy>
</tree-set>
~
```

XStream in Jenkins

- Jenkins XML API uses XStream to deserialize input
- Access to XML API -> RCE (but not such a huge deal)
- Live demo: Jenkins
- Solution: blocked DynamicProxyConverter in XStream wrapper class
- Upstream solution: whitelisting, with dynamic proxies excluded by default
- More information: https://securityblog.redhat.com/2014/01/23/java-deserialization-flaws-part-2-xml-deserialization/

RCE – binary deserialization

- Java contains a native serialization mechanism, that converts objects to binary data
- When deserializing, the readObject() and readResolve() methods of the class will be called
- This can lead to vulnerabilities if a class on the classpath has something exploitable in readObject() or readResolve()
- How can an attacker provide binary serialized objects?

RCE – binary deserialization

- Serialization is used as a format for transferring objects over networks, e.g. via REST APIs
- Example #1: RichFaces state (CVE-2013-2165, Takeshi Terada, MBSD)
- Example #2: Restlet REST framework
- Live demo: Restlet PoC
- What kind of issue could exist in readResolve()/readObject() that would be exploitable?

CVE-2011-2894: Spring

- Discovered by Wouter Coekaerts in Spring AOP
- Serializable InvocationHandler exposed
- Allows mapping a proxy to ANY method call on the proxy interface
- Similar exploit to EventHandler, but more complex setup of the serialized object graph
- More info: http://www.pwntester.com/blog/2013/12/16/cve-2011-2894-deserialization-spring-rce/

commons-fileupload

- Component to simplify file uploads in Java apps
- DiskFileItem class implements readObject()
- The readObject method creates a tmp file on disk:
  - tempFile = new File(tempDir, tempFileName);
- tempDir is read from the repository private attribute of the class, exposing a poison null byte flaw (file-writing code is native, now patched)
- An attacker can provide a serialized instance of DFI with a null-terminated full path value for the repository attribute: /path/to/file.txt\0
- commons-fileupload code embedded in Tomcat

Restlet + DFI

- Upload a JSP shell to achieve RCE
- Solution #1: don't deserialize untrusted content
- Solution #2: don't introduce flaws in readObject()/readResolve()
- Solution #3: type checking with look-ahead deserialization (Pierre Ernst): http://www.ibm.com/developerworks/java/library/se-lookahead/index.html
- Or notsoserial: https://tersesystems.com/2015/11/08/closing-the-open-door-of-java-object-serialization/

Dozer XML ↔ Binary Mapper

- Uses reflection-based approach to type conversion
- Used by e.g. Apache Camel to map types
- If used to map user-supplied objects, then an attacker can provide a dynamic proxy
- There must either be an object being mapped to with a getter/setter method that matches a method in an interface on the server classpath, or a manual XML mapping that allows an attacker to force the issue.
- InvocationHandler must be serializable (implements Serializable)
- EventHandler is not

Dozer CVE-2014-9515

- Wouter Coekaerts reported a serializable InvocationHandler in older versions of Spring: CVE-2011-2894
- Using Alvaro Munoz's CVE-2011-2894 exploit, I was able to develop a working Dozer exploit. It is only exploitable if all the aforementioned conditions are met, and vuln Spring JARs are on the classpath
- Live demo: Dozer RCE https://github.com/pentestingforfunandprofit/research/tree/master/dozer-rce
- Reported upstream since Dec 2014, no response: https://github.com/DozerMapper/dozer/issues/217

Other InvocationHandlers

- Any common component is useful, but in the JDK itself means universally exploitable
- Three other InvocationHandlers in Java 7/8:
  - CompositeDataInvocationHandler
  - MbeanServerInvocationHandler
  - RemoteObjectInvocationHandler
- CompositeDataInvocationHandler: forwards getter methods to a CompositeData instance. No use.

MBeanServerInvocationHandler

- Proxy to an MBean on the server. Potentially useful, e.g. if MBeans used by JBoss Worm are present.
- Problem 1: attacker must specify correct JMX URL
  - Solution 1: JMX is exposed locally on port 1099
  - Solution 2: Brute force JMX URL via Java PID
- Problem 2: attacker cannot control code that is run for any method call, on specific method calls
- EventHandler exploits work no matter which method is invoked on the proxy object. MBeanServerInvocationHandler simply calls the method of the same name on the MBean.

RemoteObjectInvocationHandler

- Proxy to a remote object exported via RMI
- Problem 1: attacker must know details of a remote object exported to the server
  - Solution: JMX registry is exposed via RMI. If JMX is exposed locally on port 1099, the attacker could craft an object instance that points to the JMX RMI URL
- Problem 2: attacker cannot control code that is run for any method call, on specific method calls
- Future work: look for more potentially exploitable InvocationHandlers

Property-oriented programming

- Instantiate a complex object graph whose root node is serializable
- Similar to ROP, exploit conditions in classes on the classpath so deserialization of the object graph lands in execution of arbitrary code
- Shouts to Stefan Esser for considering this in PHP first
- http://www.slideshare.net/frohoff1/appseccali-2015-marshalling-pickles Slides 45 onwards

Gadget: commons-collection

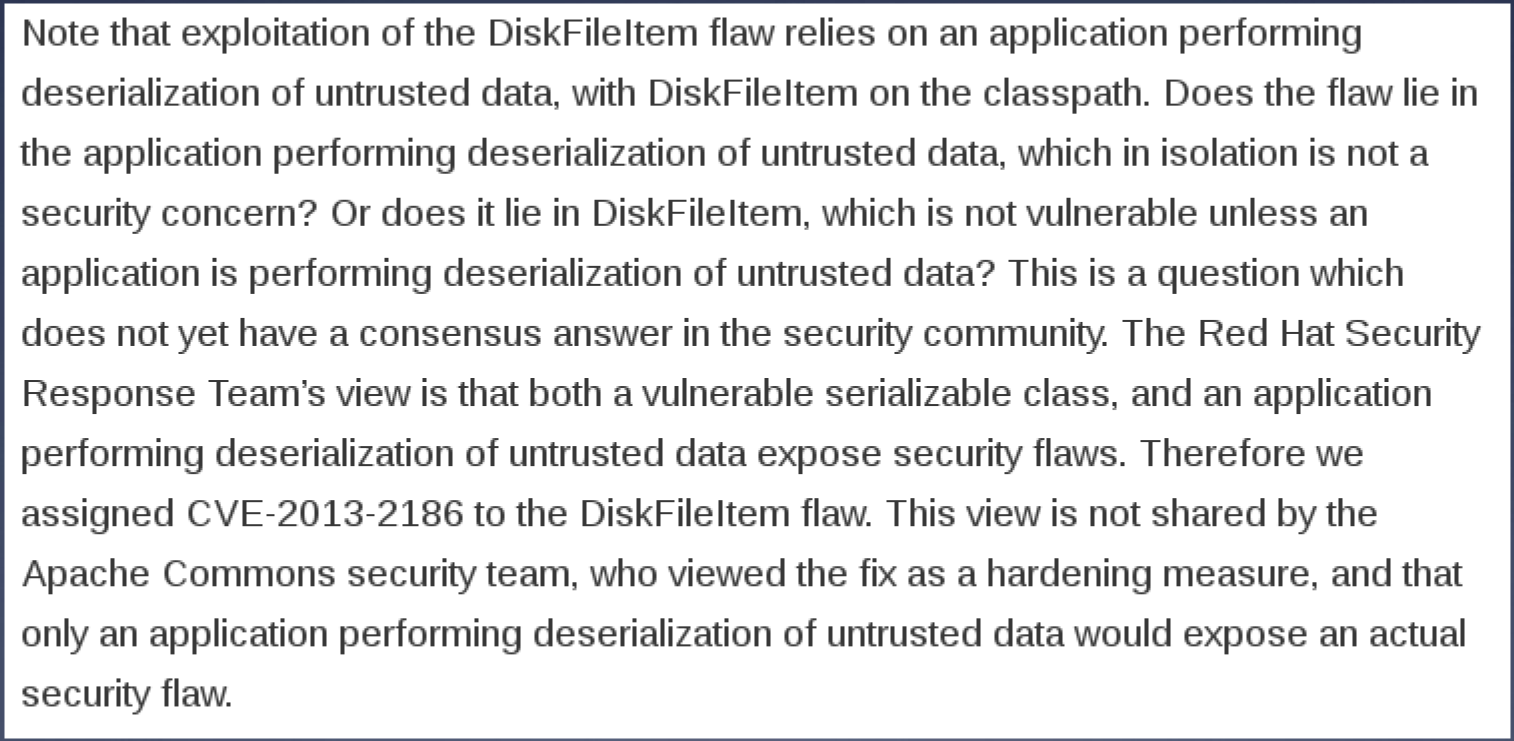
- Serializable InvocationHandler in a library that is almost universally on the classpath
- Presented at AppSecCali and still unpatched: http://www.slideshare.net/codewhitesec/exploiting-deserialization-vulnerabilities-in-java-54707478
- FoxGlove reported multiple vectors for untrusted deserialization in JBoss, WebSphere, Jenkins, WebLogic, etc.: http://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/

Tools & future research

- Ysoserial for finding flaws and aggregating payloads
- Look-ahead deserialization tools
  - PoC by Pierre Ernst @ IBM
  - Notsoserial
  - Serialkiller
- More gadgets, more deserialization vectors
- Gadget entirely in the JDK would be awesome

Where lies the vulnerability?

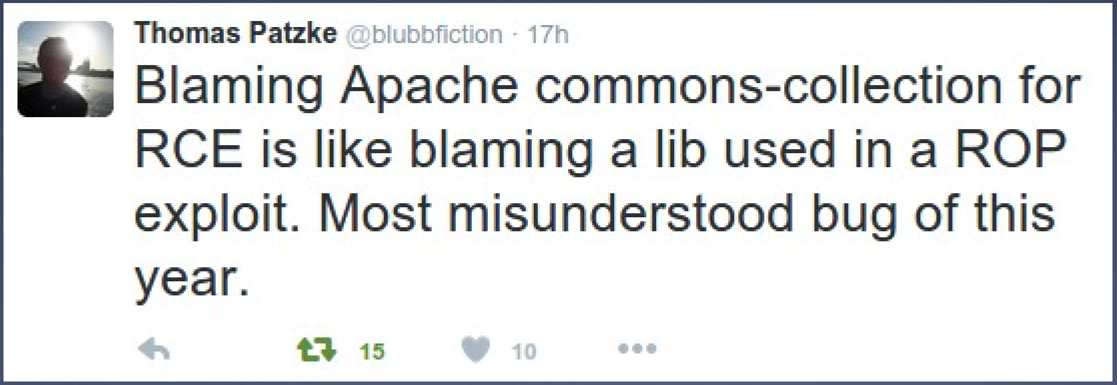
- When at Red Hat, I assigned CVEs to vulnerable classes, and publicly stated:



Note that exploitation of the DiskFileItem flaw relies on an application performing deserialization of untrusted data, with DiskFileItem on the classpath. Does the flaw lie in the application performing deserialization of untrusted data, which in isolation is not a security concern? Or does it lie in DiskFileItem, which is not vulnerable unless an application is performing deserialization of untrusted data? This is a question which does not yet have a consensus answer in the security community. The Red Hat Security Response Team's view is that both a vulnerable serializable class, and an application performing deserialization of untrusted data expose security flaws. Therefore we assigned CVE-2013-2186 to the DiskFileItem flaw. This view is not shared by the Apache Commons security team, who viewed the fix as a hardening measure, and that only an application performing deserialization of untrusted data would expose an actual security flaw.

Where lies the vulnerability?

- I was wrong!
- The vulnerability lies in the application performing deserialization of untrusted data without look-ahead type validation


Thomas Patzke @blubbfiction · 17h

Blaming Apache commons-collection for RCE is like blaming a lib used in a ROP exploit. Most misunderstood bug of this year.

15 10

Questions?