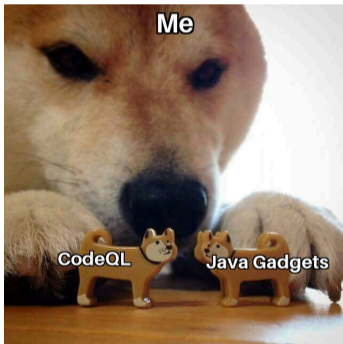




Finding Java deserialization gadgets with CodeQL



5 Juillet 2022

Synacktiv

Hugo VINCENT (@hugow_vincent)



Agenda



- 1 Why this talk
- 2 Java deserialization vulnerability
- 3 CodeQL
- 4 Finding gadgets with CodeQL
- 5 Limitations

Why this talk



- Java deserialisation vulnerabilities still exists
- Finding them becomes more and more difficult
- It's hard to find gadget chains by hand
- Finding a deserialisation vulnerability without a gadget is frustrating
- Since 2017 insecure deserialization is included in the OWASP Top 10

Agenda



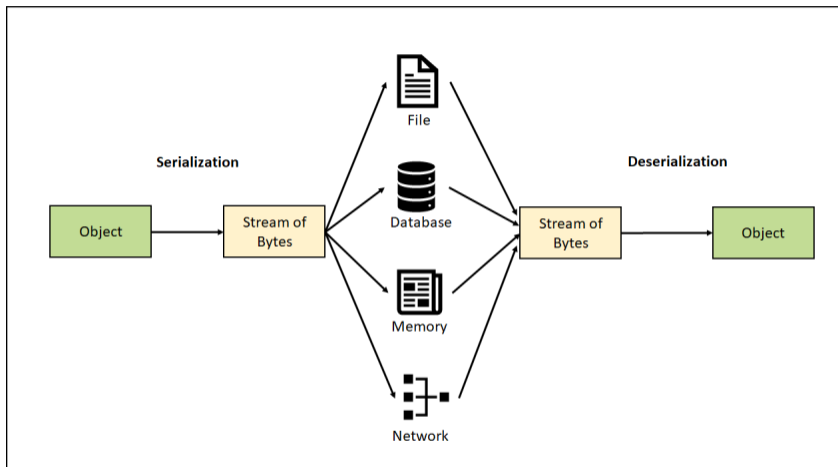
- 1 Why this talk
- 2 Java deserialization vulnerability
- 3 CodeQL
- 4 Finding gadgets with CodeQL
- 5 Limitations



Serialisation

The process of converting an object to a byte stream such that this byte stream can be reverted back to the object

Java deserialization vulnerability



serialisation



■ Reading serialized data from an *ObjectInputStream*

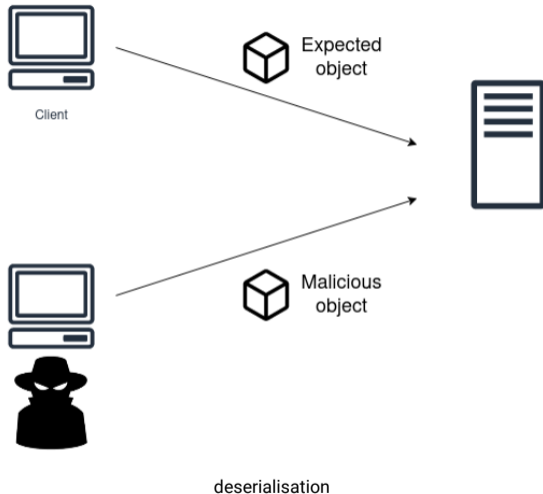
- `readObject`
- `readResolve`
- `readExternal`
- ...

What's the problem?



- Supplying user controlled data to remote method
- No checks are performed during the deserialisation process
- Every *Serializable* class can be supplied in the byte stream and reconstructed
- *Dangerous* methods can be called during the deserialisation process

What's the problem?



What's the problem?



```
public class Dummy implements Serializable {  
    private Object privateObject;  
  
    private void readObject(ObjectInputStream in) throws IOException {  
        in.defaultReadObject();  
  
        this.privateObject.dangerousMethod()  
    }  
}
```

deserialisation

A gadget chain



- Using multiple functions in the code to perform other actions
- Same principle as a ROP chain in binary exploitation

The C3PO chain



```
package com.mchange.v2.c3p0.impl;

public class PoolBackedDataSourceBase extends IdentityTokenResolvable implements Referenceable, Serializable {
    [...]
    private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
        [...]
        Object o = ois.readObject();
        if (o instanceof IndirectlySerialized) {
            o = ((IndirectlySerialized) o).getObject();
        }
    }
}
```

c3p0

The C3PO chain



```
package com.mchange.v2.naming;

public class ReferenceIndirector implements Indirector {
    [...]
    private static class ReferenceSerialized implements IndirectlySerialized {
        [...]
        public Object getObject() throws ClassNotFoundException, IOException {
            try {
                [...]
                if (contextName != null)
                    nameContext = (Context) initialContext.lookup(contextName);
            }
        }
    }
}
```

c3p0

The C3PO chain

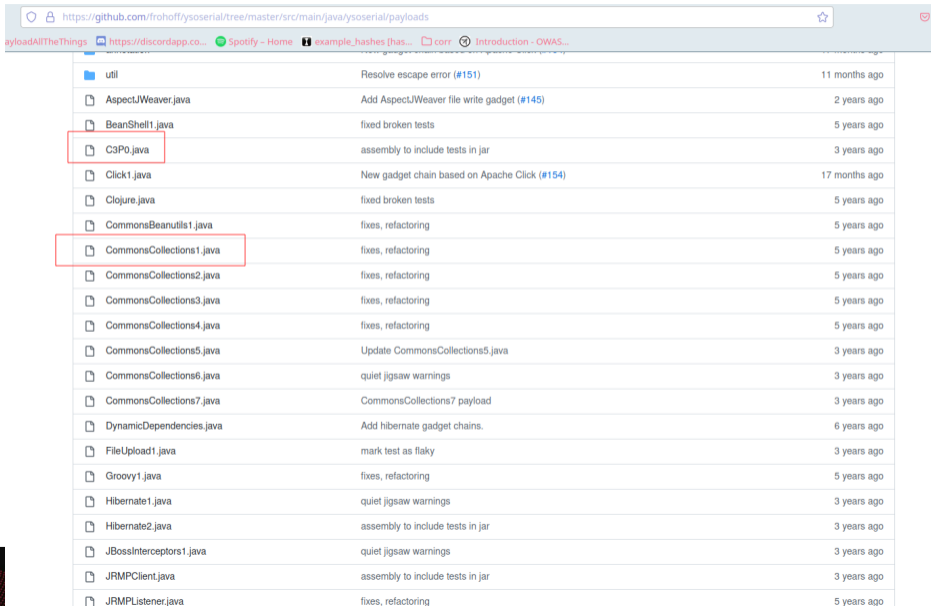


```
PoolBackedDataSourceBase->readObject  
  ReferenceIndirector$ReferenceSerialized->getObject  
    RegistryContext->lookup
```

The Spring1 chain

```
ObjectInputStream.readObject()
  SerializableTypeWrapper.MethodInvokeTypeProvider.readObject()
    SerializableTypeWrapper.TypeProvider(Proxy).getType()
      AnnotationInvocationHandler.invoke()
        HashMap.get()
      ReflectionUtils.findMethod()
    SerializableTypeWrapper.TypeProvider(Proxy).getType()
      AnnotationInvocationHandler.invoke()
        HashMap.get()
    ReflectionUtils.invokeMethod()
      Method.invoke()
        Templates(Proxy).newTransformer()
          AutowireUtils.ObjectFactoryDelegatingInvocationHandler.invoke()
            ObjectFactory(Proxy).getObject()
              AnnotationInvocationHandler.invoke()
                HashMap.get()
            Method.invoke()
              TemplatesImpl.newTransformer()
                TemplatesImpl.getTransletInstance()
                  TemplatesImpl.defineTransletClasses()
                    TemplatesImpl.TransletClassLoader.defineClass()
                      Pwner*(Javassist-generated).<static init>
                        Runtime.exec()
```

YSOSERIAL



File Name	Commit Message	Time Ago
util	Resolve escape error (#151)	11 months ago
AspectJWeaver.java	Add AspectJWeaver file write gadget (#145)	2 years ago
BeanShell1.java	fixed broken tests	5 years ago
C3P0.java	assembly to include tests in jar	3 years ago
Click1.java	New gadget chain based on Apache Click (#154)	17 months ago
Clojure.java	fixed broken tests	5 years ago
CommonsBeanutils1.java	fixes, refactoring	5 years ago
CommonsCollections1.java	fixes, refactoring	5 years ago
CommonsCollections2.java	fixes, refactoring	5 years ago
CommonsCollections3.java	fixes, refactoring	5 years ago
CommonsCollections4.java	fixes, refactoring	5 years ago
CommonsCollections5.java	Update CommonsCollections5.java	3 years ago
CommonsCollections6.java	quiet jigsaw warnings	3 years ago
CommonsCollections7.java	CommonsCollections7 payload	3 years ago
DynamicDependencies.java	Add hibernate gadget chains.	6 years ago
FileUpload1.java	mark test as flaky	3 years ago
Groovy1.java	fixes, refactoring	5 years ago
Hibernate1.java	quiet jigsaw warnings	3 years ago
Hibernate2.java	assembly to include tests in jar	3 years ago
JBossInterceptors1.java	quiet jigsaw warnings	3 years ago
JRMPCClient.java	assembly to include tests in jar	3 years ago
JRMPListener.java	fixes, refactoring	5 years ago

Gadget Inspector



- Presented at Black Hat USA 2018 by @ianhaken
- A Java bytecode analysis tool for finding gadget chains
- Works by reconstructing the AST (abstract syntax tree)
- Clojure1 / Jython

Agenda



- 1 Why this talk
- 2 Java deserialization vulnerability
- 3 CodeQL**
- 4 Finding gadgets with CodeQL
- 5 Limitations



- Static code analyser
- build a database by parsing the code to reconstruct the AST
- Analyse the code by making queries on it
- Useful to find vulnerabilities by pattern
- Java/C/C++/C#/Javascript/Python/Swift...
- (partially) open source : <https://github.com/github/codeql>



```
import java

from Method m
where m.hasName("readObject")
select m
```



« 1 / 2 » test.q1 on jdk11u - finished in 0.066 seconds, 392 result count
PMJ

#select v

	#	m
1		readObject
2		readObject
3		readObject
4		readObject
5		readObject
6		readObject
7		readObject
8		readObject
9		readObject
10		readObject
11		readObject
12		readObject
13		readObject
14		readObject
15		readObject
16		readObject
17		readObject

readObject



```
public static String getMD5String(String value) {  
    try {  
        MessageDigest md = MessageDigest.getInstance("MD5");  
        md.update(value.getBytes(StandardCharsets.UTF_8));  
        byte[] digest = md.digest();  
    }  
}
```

- > java/tainted-arithmetic java/tainted-arithmetic 2
- > java/non-ssl-connection java/non-ssl-connection 2
- ✓ java/weak-cryptographic-algorithm java/weak-cryptographic-algorithm 2
 - ⚠ 47 CryptoUtil.java Cryptographic algorithm MD5
 - ⚠ 84 CryptoUtil.java Cryptographic algorithm MD5

weak hash




```
cat Security/CWE/CWE-327/BrokenCryptoAlgorithm.ql
/**
 * @name Use of a broken or risky cryptographic algorithm
 * @description Using broken or weak cryptographic algorithms can allow an attacker to
 *               compromise security.
 * @kind path-problem
 * @problem.severity warning
 * @security-severity 7.5
 * @precision high
 * @id java/weak-cryptographic-algorithm
 * @tags security
 *       external/cwe/cwe-327
 *       external/cwe/cwe-328
 */
```









github / codeql Public

<> Code Issues 488 Pull requests 203 Discussions Actions Projects Security Insights

main codeql / java / ql / src / experimental / Security / CWE /

 aschackmull Merge pull request #9618 from aschackmull/dataflow/deprecate-barrierg... ..

..

 CWE-016	add all remaining explicit this
 CWE-020	Java: Update Log4J models with provenance information.
 CWE-036	Java: remove duplicated class
 CWE-073	Merge pull request #9618 from aschackmull/dataflow/deprecate-barrierg...
 CWE-078	Deduplicate shared body of regular and experimental versions of `java...
 CWE-089	patch upper-case acronyms to be PascalCase

bb

Agenda



- 1 Why this talk
- 2 Java deserialization vulnerability
- 3 CodeQL
- 4 Finding gadgets with CodeQL**
- 5 Limitations

Finding gadgets with CodeQL



- The source
- The sink
- The path

The Sink



Sink methods are the dangerous methods that we want to reach. We can define them in CodeQL like this :

- *RuntimeExec* the CodeQL class name
- extends the *Method* class
- defined in the *java.lang* package
- in the *Runtime* class
- method name is *exec*

```
private class RuntimeExec extends Method {  
  RuntimeExec(){  
    hasQualifiedName("java.lang", "Runtime", "exec")  
  }  
}
```

The Sink

```
from MethodAccess ma
where ma.getMethod() instanceof RuntimeExec
select ma
```

```
    this.hashCode();
    return false;
}

@Override
public int hashCode() {
    try {
        Runtime.getRuntime().exec("id");
    } catch (IOException e) {
        e.printStackTrace();
    }
    return 0;
}
```

sink

The Sink

```
class DangerousMethod extends Callable {
    DangerousMethod(){
        this instanceof ExpressionEvaluationMethod or
        this instanceof ReflectionInvocationMethod or
        this instanceof RuntimeExec or
        this instanceof URL or
        this instanceof ProcessBuilder or
        this instanceof Files or
        this instanceof FileInputStream or
        this instanceof FileOutputStream or
        this instanceof EvalScriptEngine or
        this instanceof ClassLoader or
        this instanceof ContextLookup or
        this instanceof OGNLEvaluation or
        this instanceof DriverManagerMethods or
        this instanceof System
    }
}
```

The Sink



We want all the methods that calls a *DangerousMethod*, so we look for *MethodAccess* of dangerous methods, and we use the enclosing callable as a result :

```
private class CallsDangerousMethod extends Callable {  
  
    CallsDangerousMethod(){  
  
        exists(MethodAccess ma | ma.getMethod() instanceof DangerousMethod and ma.  
            getEnclosingCallable() = this)  
  
    }  
  
}
```

The Sink

```
from Callable c
where c instanceof CallsDangerousMethod
select c
```

```
public class VulnClass implements VulnInterface, Serializable {
    public void vulnMethod(){
        Class clazz = Object.class;
        Object o = new Object();
        try {
            Method m = clazz.getMethod("toString");
            m.invoke(o, null);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

#	
1	vulnMethod
2	hashCode

sink

The Source



A source is a method that we can call to start the gadget chain, the first obvious one is `readObject` but there are other methods like :

- `readObjectNoData`
- `readResolve`
- `readExternal`
- ...

The Source



There are other methods that have been used in other known chains that we can add :

- hashCode
- equals
- compare
- ...

The Source

```
1) private void readObject(java.io.ObjectInputStream s)
2)     throws IOException, ClassNotFoundException
3)     {
4)         s.defaultReadObject();
5)         [...]
6)         table = new Entry<?,?>[length];
7)         [...]
8)         for (; elements > 0; elements--) {
9)             K key = (K)s.readObject();
10)            V value = (V)s.readObject();
11)            reconstitutionPut(table, key, value);
12)
13)         }
14)         [...]
```

The Source



```
1) private void reconstitutionPut(Entry<?,?>[] tab, K key, V value)
2)     throws StreamCorruptedException
   {
   [...]
3)     int hash = key.hashCode();
   [...]
4)     for (Entry<?,?> e = tab[index] ; e != null ; e = e.next) {
5)         if ((e.hash == hash) && e.key.equals(key)) {
6)             throw new java.io.StreamCorruptedException();
           }
       }
   }
   [...]
```

The source

```
class Source extends Callable{
    Source(){
        getDeclaringType().getASupertype*() instanceof TypeSerializable and (
            this instanceof MapSource or
            this instanceof SerializableMethods or
            this instanceof Equals or
            this instanceof hashCode or
            this instanceof Compare or
            this instanceof ExternalizableMethod or
            this instanceof ObjectInputValidationMethod or
            this instanceof InvocationHandlerMethod or
            this instanceof MethodHandlerMethod or
            this instanceof GroovyMethod
        )
    }
}
```

The path



- Finding a path between the *source* and the *sink*

```
public void A(){
    B()
}

public void B(){
    C()
}

public void C(){
    dangerousMethod()
}

public void readObject(ObjectInputStream in) {
    A()
}
```

The path



```
A.pollyCalls(B)
```

```
A.pollyCalls(B)  
B.pollyCalls(C)  
...
```



■ Recursion

```
private class RecursiveCallToDangerousMethod extends Callable {
    RecursiveCallToDangerousMethod(){

        this instanceof CallsDangerousMethod or
        exists(RecursiveCallToDangerousMethod unsafe | this.polyCalls(unsafe))

    }
}
```



```
1) java.util.PriorityQueue.readObject()
2)     java.util.PriorityQueue.heapify()
3)     java.util.PriorityQueue.siftDown()
4)     java.util.PriorityQueue.siftDownUsingComparator()
5)     org.apache.click.control.Column$ColumnComparator.compare()
6)     org.apache.click.control.Column.getProperty()
7)     org.apache.click.control.Column.getProperty()
8)     org.apache.click.util.PropertyUtils.getValue()
9)     org.apache.click.util.PropertyUtils.getObjectPropertyValue()
10)    java.lang.reflect.Method.invoke()
11)    com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl.
      getOutputProperties()
```


real world use case



- Click1
- ROME
- Hibernate1
- Mojarra
- WildFly1



```
File: WildFlyDataSource.java
113:     private void readObject(java.io.ObjectInputStream in) throws IOException,
        ClassNotFoundException {
114:         in.defaultReadObject();
115:         jndiName = (String) in.readObject();
116:
117:
118:         try {
119:             InitialContext context = new InitialContext();
120:
121:             DataSource originalDs = (DataSource) context.lookup(jndiName);
[...]
```



- Wildfly is a Java application server, with more than 10000 Java classes.
- A pull request was made on ysoserial
- The *WildFlyDataSource* class is part of the *org.jboss.as.connector* package and is bundled inside the WildFly GitHub repository.

Agenda



- 1 Why this talk
- 2 Java deserialization vulnerability
- 3 CodeQL
- 4 Finding gadgets with CodeQL
- 5 Limitations

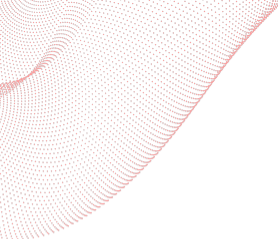
Limitations



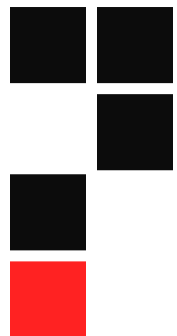
- Need to have the source code of the library/project
- Need to be able to compile the project
- You can analyse one project at a time

The END

■ <https://github.com/synacktiv/QLInspector>



ANY QUESTIONS?



THANKS FOR YOUR ATTENTION

 **SYNAKTIV**