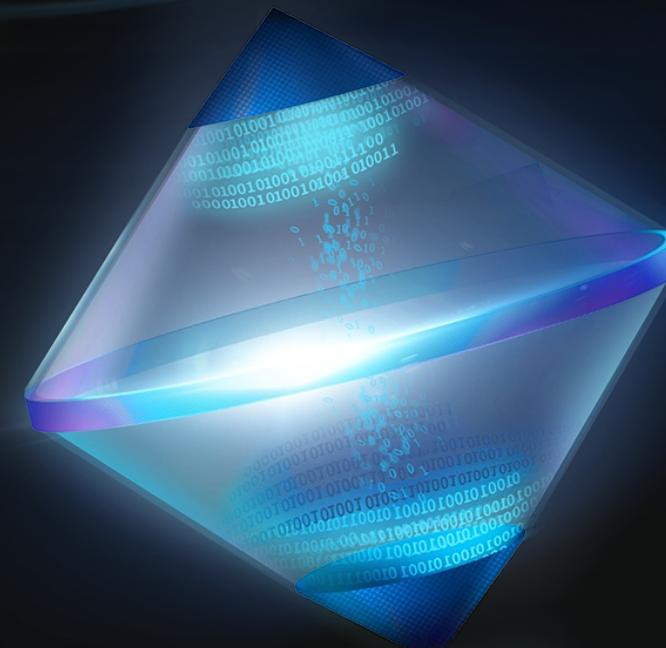


Magic In Java API



Speaker: yemoli、R1ckyZ



yemoli、永信至诚伽玛实验室高级研究员

R1ckyZ、中国科学院大学

 wha1er

 @yemoliSec

 yml-sec.top

 R1ckyZ-WEB

 @iR1ckyZ

 r1ckyz.github.io

Agenda

01

API简介

02

安全影响

03

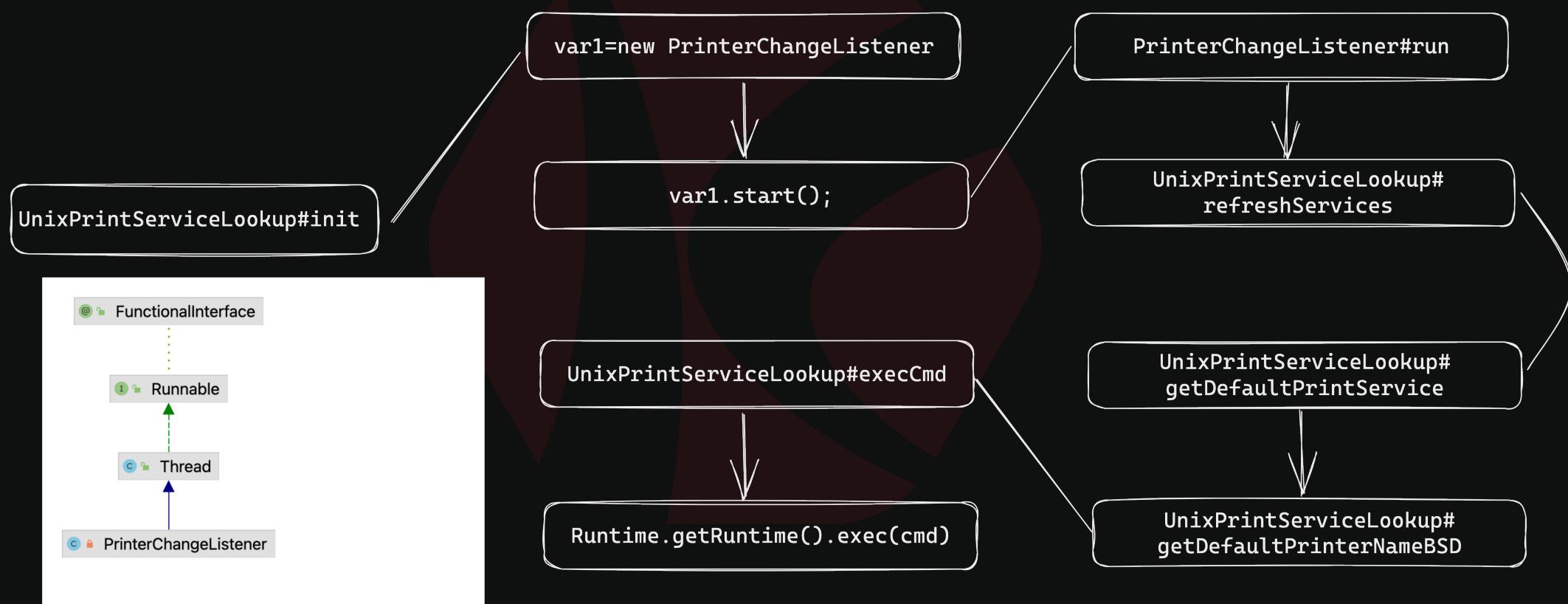
RASP攻防Tricks

04

减缓措施

◆ API简介

JDK中 PrintServiceLookup接口用于提供打印服务的注册查找功能，在linux的JDK中它的实现类叫做UnixPrintServiceLookup 或 PrintServiceLookupProvider（高版本 jdk中）



◆ API简介

execCmd的参数来自于类的属性，在某些特定情况下如果被篡改掉可能存在一些安全风险

```
public class UnixPrintServiceLookup {  
    String[] lpcFirstCom = new String[] {"xxx"};  
    private String getDefaultPrinterNameBSD() {  
        if(cmdIndex == -1) {  
            cmdIndex = getBSDCommandIndex();  
        }  
        String[] var1 = execCmd(this.lpcFirstCom[cmdIndex]);  
        if(var1 != null && var1.length != 0) {  
            return cmdIndex == 1 && var1[0].startsWith("missingprinter") ? null : var1[0];  
        }  
        else {  
            return null;  
        }  
    }  
}
```

安全影响

01

FieldBased mechanism

02

Pojo Class Recovery

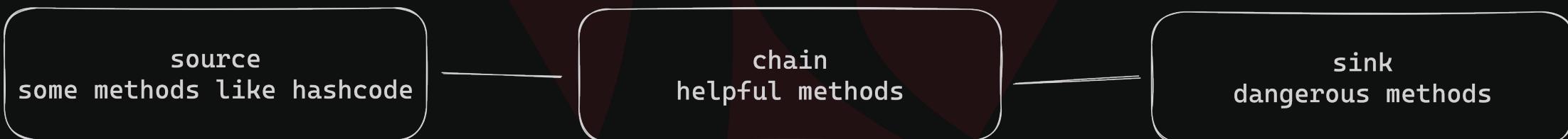
03

Member of gadget chain

◆ How FieldBased mechanism work in deserialisation

- 在某些时候实例化类时调用无参数构造方法
- 还原 Map 类型时，会调用 hashCode 等方法
- 允许反序列化未实现 Serializable 接口的类
- 使用反射来恢复对象的属性

Existing ideas

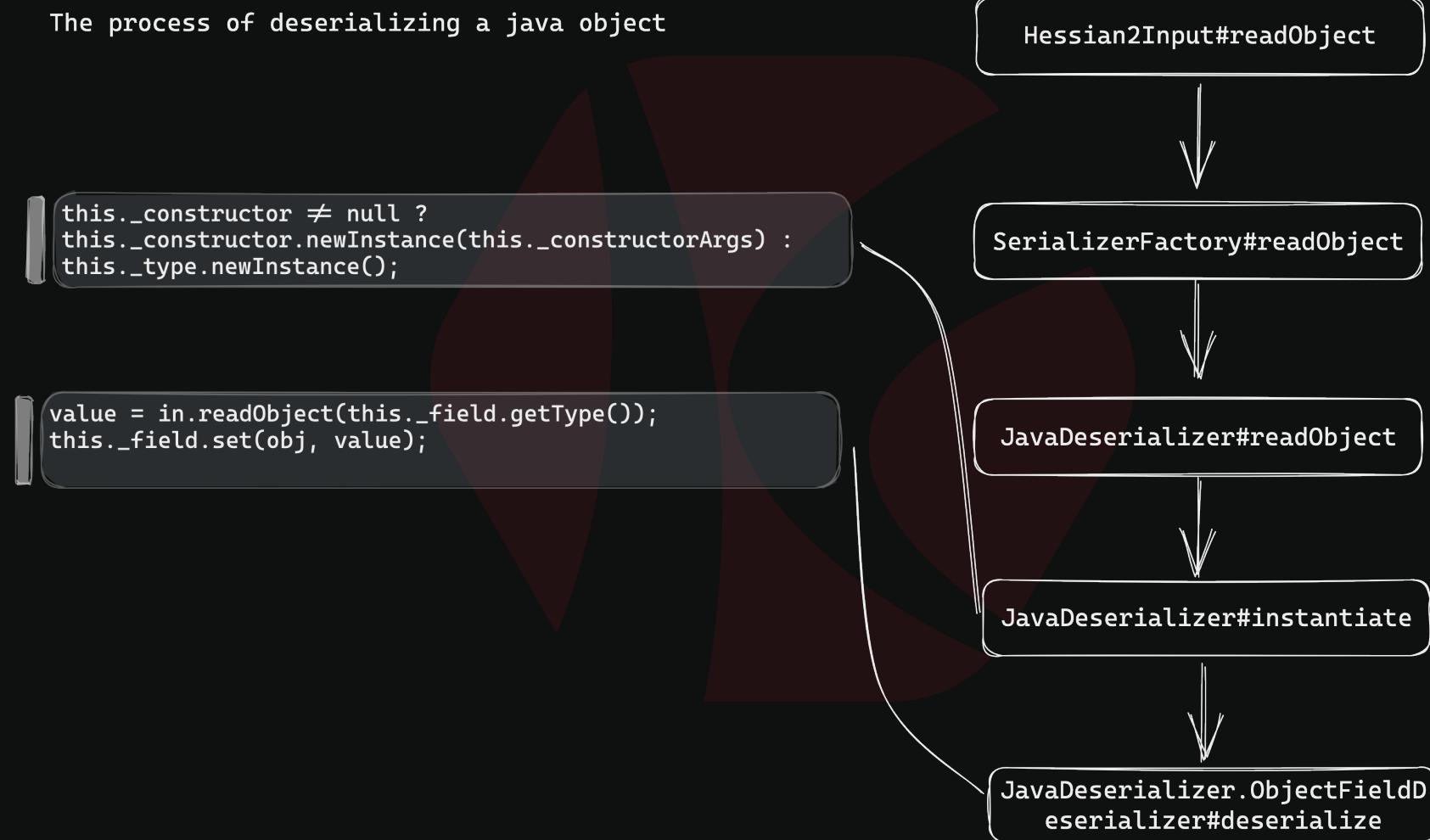


◆ Utilisation under FieldBased mechanism

Serialize Type	Affect
java native serialize	NO
Hessian	YES(version=3.x)
Hessian Lite	YES(version<=3.2.12)
SOFA-Hessian	YES(version<4.0.0)
Kryo	YES
FastJson	YES (FieldBased)
Jackson	YES (FieldBased)
XStream	NO
FST	NO

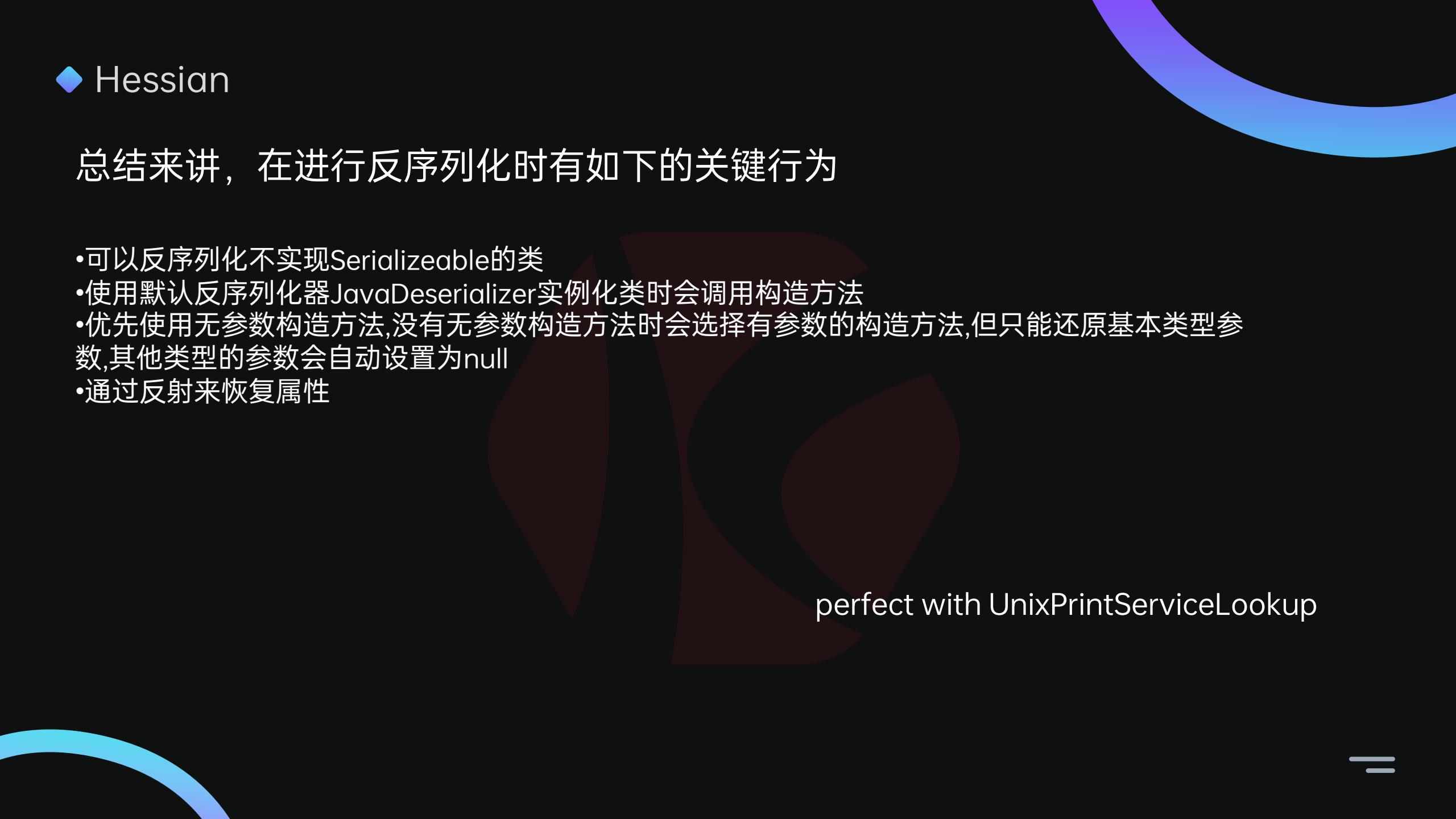
◆ Hessian

Hessian是一种轻量级的二进制序列化协议，3.x版本中其反序列化的大致流程如下



总结来讲，在进行反序列化时有如下的关键行为

- 可以反序列化不实现Serializable的类
- 使用默认反序列化器JavaDeserializer实例化类时会调用构造方法
- 优先使用无参数构造方法,没有无参数构造方法时会选择有参数的构造方法,但只能还原基本类型参数,其他类型的参数会自动设置为null
- 通过反射来恢复属性



perfect with UnixPrintServiceLookup

◆ Why not hessian 4.x ?

```
//3.x use JavaSerializer
protected Serializer getDefaultSerializer(Class cl) {
    if(this._defaultSerializer != null) {
        return this._defaultSerializer;
    }
    else if(!Serializable.class.isAssignableFrom(cl) && !this._isAllowNonSerializable) {
        throw new IllegalStateException("Serialized class " + cl.getName() + " must implement java.io.Serializable");
    }
    else {
        return new JavaSerializer(cl);
    }
}

//4.x use UnsafeSerializer
protected Serializer getDefaultSerializer(Class cl) {
    if(this._defaultSerializer != null) {
        return this._defaultSerializer;
    }
    else if(!Serializable.class.isAssignableFrom(cl) && !this._isAllowNonSerializable) {
        throw new IllegalStateException("Serialized class " + cl.getName() + " must implement java.io.Serializable");
    }
    else {
        return(Serializer)(this._isEnableUnsafeSerializer && JavaSerializer.getWriteReplace(cl) == null ?
UnsafeSerializer.create(cl) : JavaSerializer.create(cl));
    }
}
```

◆ Why not hessian 4.x ?

```
//3.x use JavaDeserializer can call the default constructor
protected Object instantiate() throws Exception {
    try {
        if(_constructor != null) return _constructor.newInstance(_constructorArgs);
        else return _type.newInstance();
    }
    //other code
}
```

```
//4.x use UnsafeDeserializer and instantiating classes with unsafe
protected Object instantiate() throws Exception {
    return _unsafe.allocateInstance(this._type);
}
```

#So in version 4.x you call constructor methods, can't use UnixPrintServiceLookup

◆ Hessian Lite

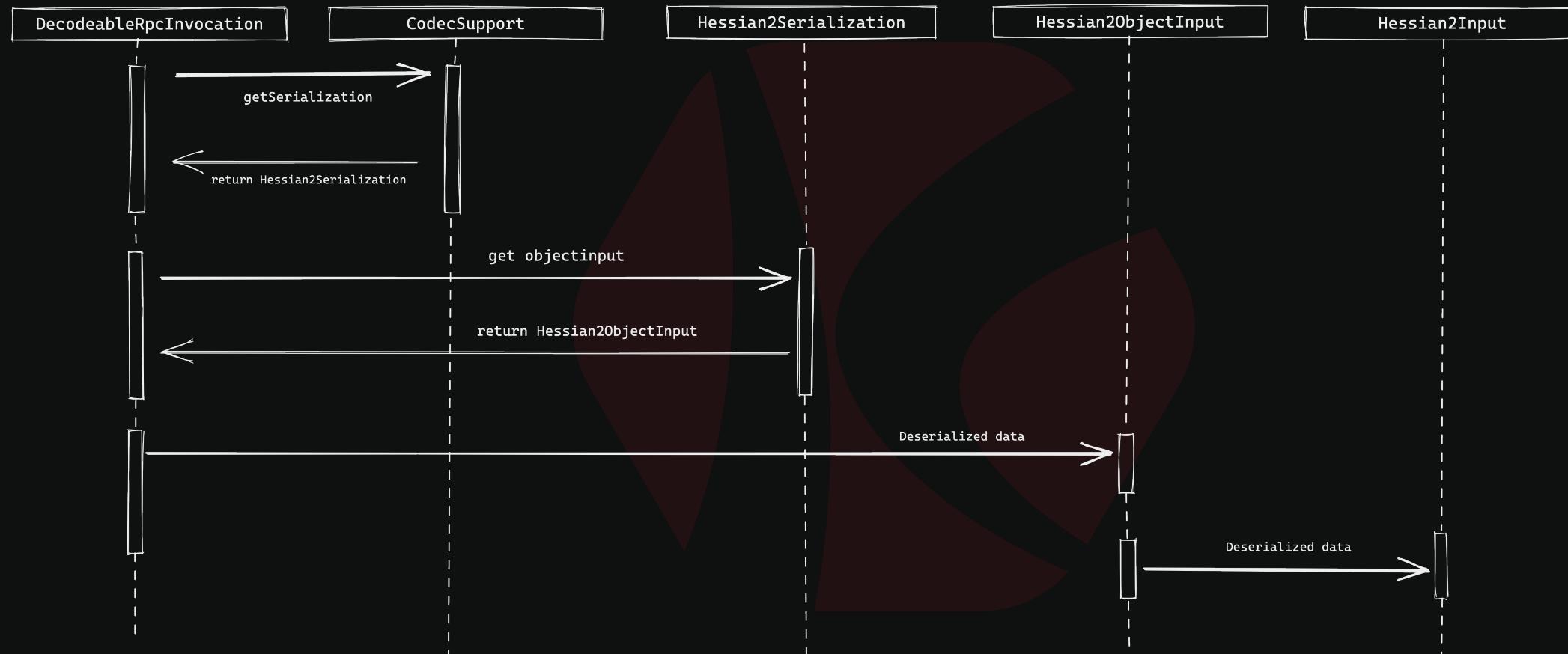
hessian-lite是Apache Dubbo的一个模块，它是基于官方hessian的一个修改版本，在安全性上来说，相比于原始的hessian ,hessian-lite中维护了一个黑名单，用于防御已知的反序列化攻击；

```
#DENY_CLASS
bsh.
ch.qos.logback.core.db.
clojure.
com.alibaba.citrus.springext.support.parser.
com.alibaba.citrus.springext.util.SpringExtUtil.
com.alibaba.druid.pool.
com.alibaba.hotcode.internal.org.apache.commons.collections.functors.
com.alipay.custrelation.service.model.redress.
com.alipay.oceanbase.obproxy.druid.pool.
com.caucho.config.types.
...
```

在3.2.12以及之前的版本中,可以反序列化没有继承Serializable的类，同时在黑名单中也不存在对sun.print的包的过滤，且默认的反序列化器为JavaDeserializer，因此可以完全符合我们对UnixPrintServiceLookup的利用要求

◆ Lead to CVE-2022-39198

dubbo通信时默认使用hessian-lite进行反序列化，流程如下



◆ Lead to CVE-2022-39198

根据上述流程可以发现完全符合UnixPrintServiceLookup的利用条件；
对此官方修复方案是在黑名单中限制了包名，同时不允许实例化未继承Serializable的类

org.springframework.orm.	125 org.springframework.orm.
org.springframework.transaction.	126 org.springframework.transaction.
org.yaml.snakeyaml.tokens.	127 org.yaml.snakeyaml.tokens.
	128 + ognl.
pstore.shaded.org.apache.commons.collections.	129 pstore.shaded.org.apache.commons.collections.
	130 + sun.print.
sun.rmi.server.	131 sun.rmi.server.
sun.rmi.transport.	132 sun.rmi.transport.
weblogic.ejb20.internal.	133 weblogic.ejb20.internal.

```
417 +         if (!Serializable.class.isAssignableFrom(cl)
418 +             && !_isAllowNonSerializable) {
419 +             throw new IllegalStateException("Serialized class " + cl.getName() + " must
420 +               implement java.io.Serializable");
421 + }
```

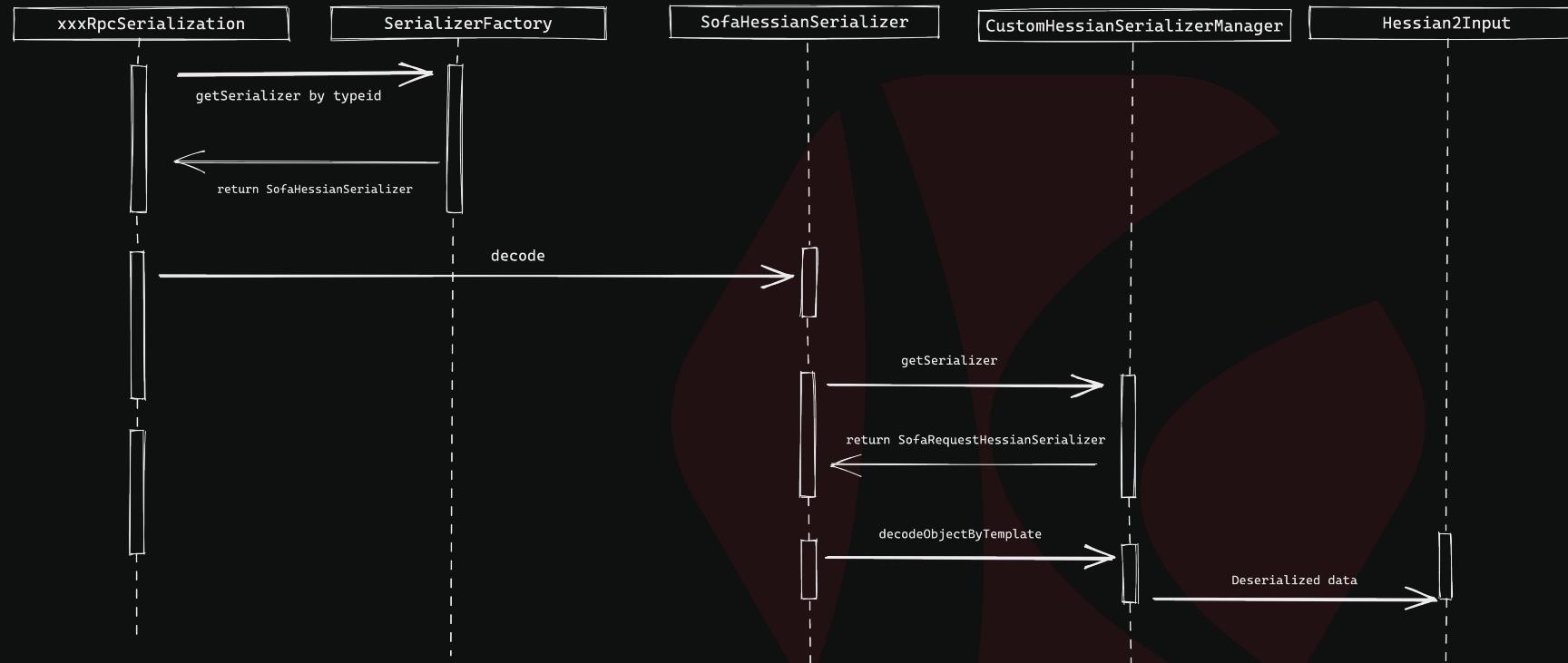
◆ SOFA-Hessian

SOFA-Hessian也是一个基于原生 Hessian 的改进版本, 同样的, 在3.x版本中, 其默认的反序列化器为 JavaDeserializer, 且 sun.print 包并不在其 serialize.blacklist 黑名单中, 所以在3.x版本中对于 SOFA-Hessian 也可以直接利用 UnixPrintServiceLookup

而在4.x的版本中, 其反序列化器选择上同Hessian一样做出了改动, 因此无法使用

◆ Safety Case

某开源rpc框架依赖了SOFA-Hessian 3.x， Server端解析客户端传入数据的流程如下

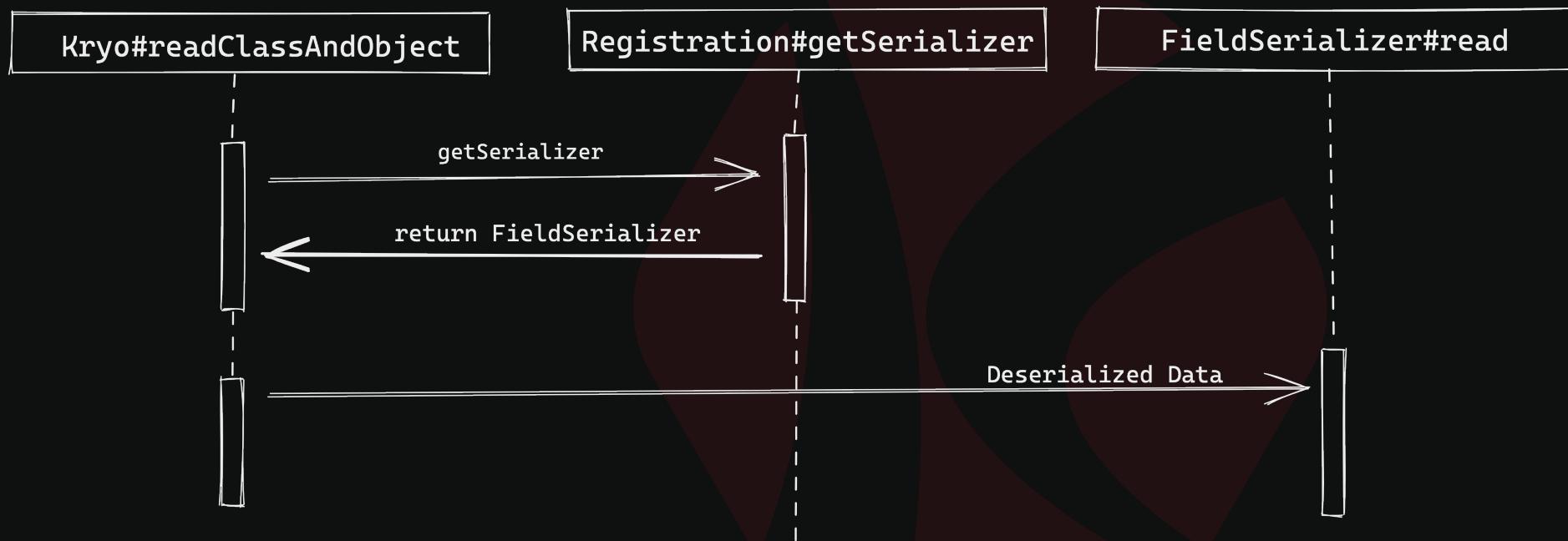


在进行通信解析客户端传入参数时，会使用SOFA-Hessian进行反序列化，当构造好恶意的Object类型参数时，就可以将UnixPrintServiceLookup封装到数据中，在反序列化时执行命令，由于在反序列化前Server端并没有校验客户端调用的方法是否存在，因此无需知道远程的方法名也不影响反序列化恶意数据。



◆ Kryo

Kryo是一种Java对象序列化协议，它可以快速地序列化和反序列化Java对象，并且生成的序列化数据比其他Java序列化协议更小，其反序列化的流程如下

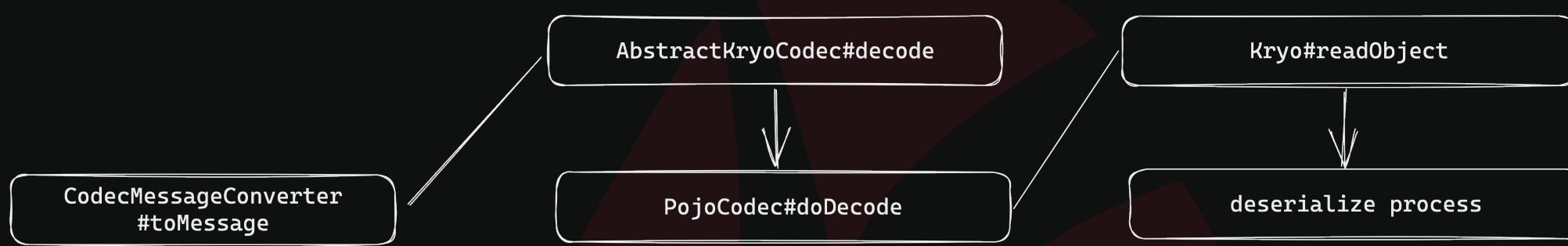


kryo在反序列化类时当类不在预设的缓存中时会使用FieldSerializer进行反序列化，
FieldSerializer实例化类时仅会使用类的无参数的构造方法，当类没有无参数的构造方法
时无法实例化，实例化后利用反射来恢复属性。



◆ Lead to CVE-2020-5413

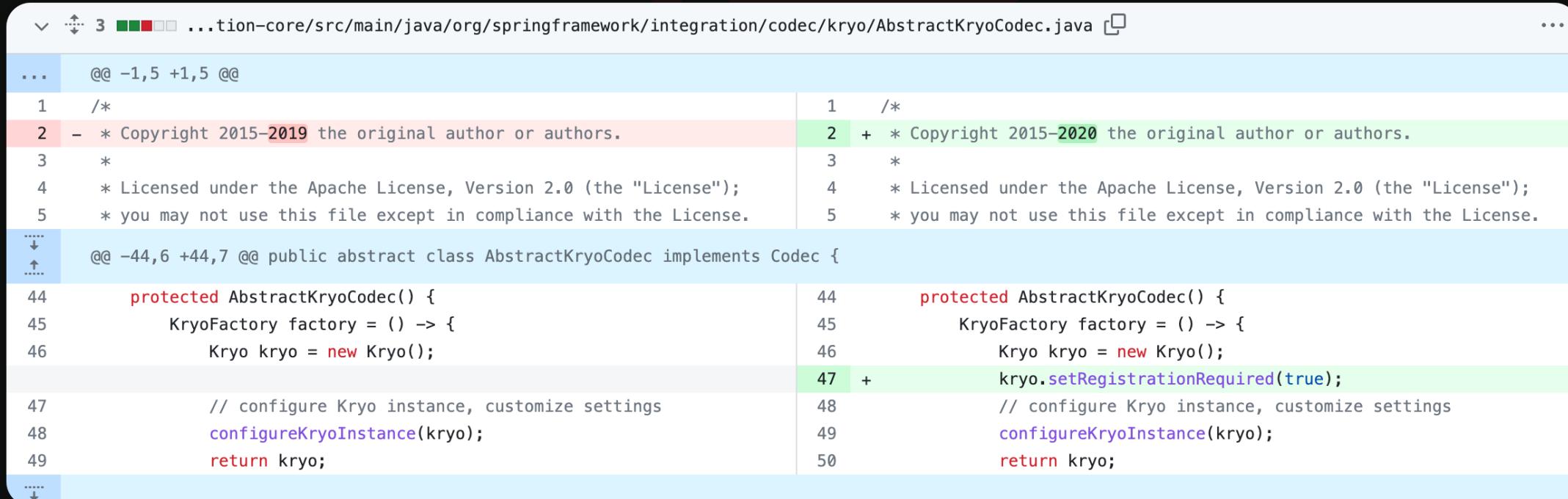
Spring Integration是一种消息控制框架，其CodecMessageConverter在对消息进行转换时会使用到kryo进行反序列化，具体流程如下



在5.2.7及以前的版本中，类无需注册即可使用kryo进行反序列化，因此在这里可以反序列化任意类，在反序列化时会默认用到FieldSerializer来恢复类和属性，因此这里符合使用UnixPrintServiceLookup的条件，无需任何依赖即可完成RCE利用

◆ Lead to CVE-2020-5413

官方对于该问题的修复逻辑主要是开启了kryo的类注册机制，因此在高版本中当消息接收方没有注册我们的恶意类时，是无法进行利用的



The screenshot shows a code diff interface comparing two versions of the `AbstractKryoCodec` class. The left column shows the original code, and the right column shows the patched code. The changes are as follows:

- Line 2: Copyright notice updated from "2015-2019" to "2015-2020".
- Line 44: A new line `kryo.setRegistrationRequired(true);` is added.

```
@@ -1,5 +1,5 @@
1  /*
2 - * Copyright 2015-2019 the original author or authors.
3 *
4 * Licensed under the Apache License, Version 2.0 (the "License");
5 * you may not use this file except in compliance with the License.
@@ -44,6 +44,7 @@ public abstract class AbstractKryoCodec implements Codec {
44     protected AbstractKryoCodec() {
45         KryoFactory factory = () -> {
46             Kryo kryo = new Kryo();
47             // configure Kryo instance, customize settings
48             configureKryoInstance(kryo);
49             return kryo;
50         }
51     }
52 }
```

◆ FastJson(FieldBased)

当FastJson开启Autotype和FieldBased配置时，将JSON转换为JAVA对象的流程如下

实例化类

获取目标类的默认无参数构造函数，通过该函数实例化类

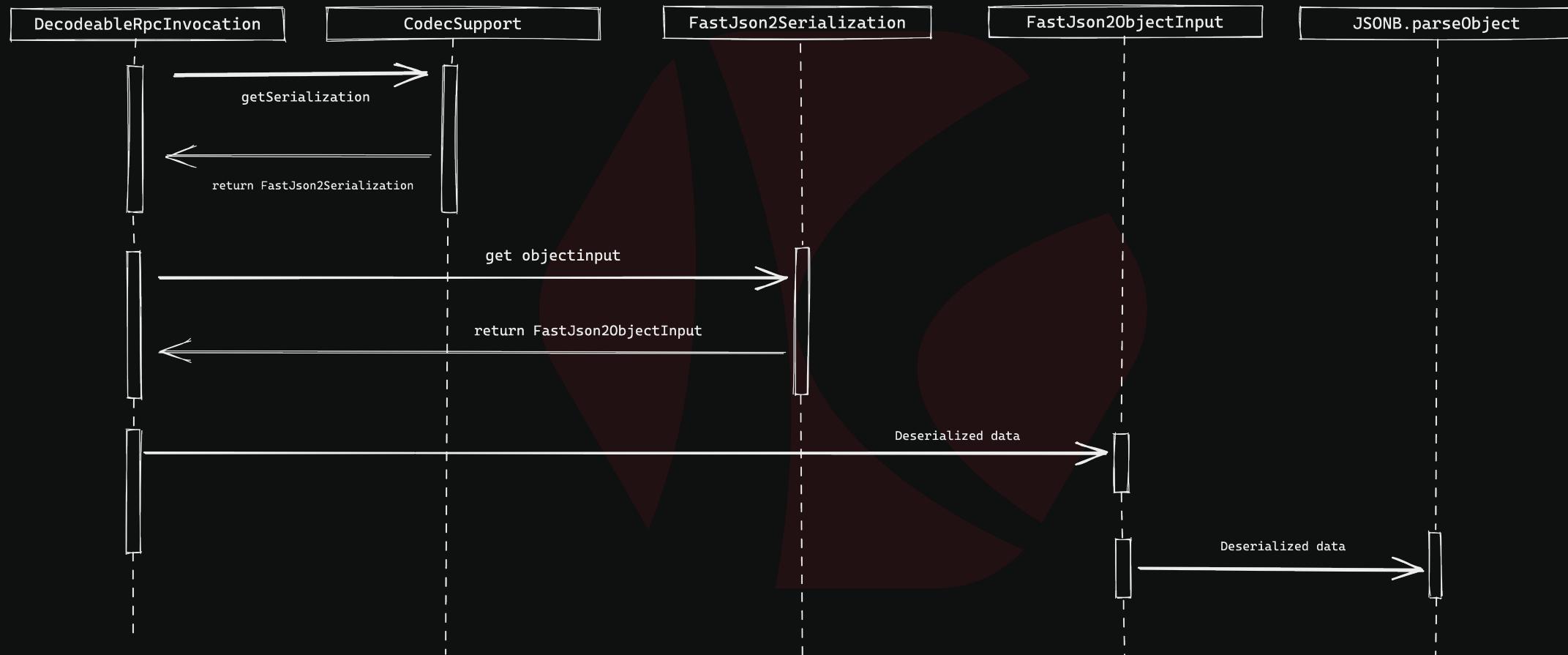
如目标类没有默认无参构造函数，尝试获取调用其它构造函数

恢复属性

1. 获取目标类的所有字段
2. 解析 JSON 字符串, 匹配字段
3. 字段匹配成功, 反射设置值
4. 字段匹配失败, 忽略该属性
5. 反序列化完成, 返回类实例

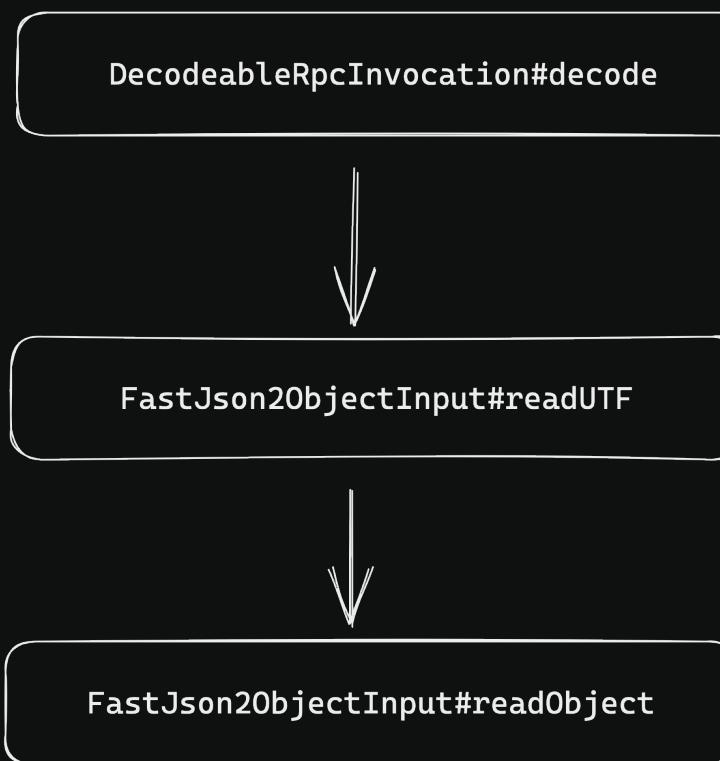
◆ Lead to CVE-2023-23638[1] version: 3.1.0-3.1.4

在dubbo的 3.1.x 版本中，新增了 fastjson2 反序列化器，反序列化数据的流程如下



◆ Lead to CVE-2023-23638[1] version: 3.1.0-3.1.4

在dubbo中默认会使用hessian进行反序列化，但攻击者可以通过consumer更改Serializeid来更改反序列化器的类型，虽然在高版本的dubbo存在着对反序列化id的校验检查，但在检查之前还存在着一个readUTF来读取dubbo的版本号，在Fastjson2反序列化器中readUTF会调用readObject来进行反序列化，整体流程如下



```
public < T > T readObject(Class < T > cls) throws IOException {
    updateClassLoaderIfNeed();
    int length = readLength();
    byte[] bytes = new byte[length];
    int read = is.read(bytes, 0, length);
    if(read != length) {
        throw new IllegalArgumentException("deserialize failed. expected read length: " + length + " but actual read: " + read);
    }
    return(T) JSONB.parseObject(bytes, Object.class, JSONReader.Feature.SupportAutoType,
        JSONReader.Feature.UseDefaultConstructorAsPossible, JSONReader.Feature.UseNativeObject, JSONReader.Feature.FieldBased);
}
```

可以看到在调用fastjson2解析数据时开启了AutoType和FieldBased，可以利用UnixPrintServiceLookup 执行命令，值得注意的是，该种方法适用性更强，无需知道远程服务的服务名及参数类型即可利用；



◆ Lead to CVE-2023-23638[1] version: 3.1.0-3.1.4

对此问题，官方关闭了Autotype,同时不允许反序列化不实现Serializable的类

```
12 ...json2/src/main/java/org/apache/dubbo/common/serialize/fastjson2/FastJson2ObjectInput.java ...
+-- @@ -32,11 +32,15 @@ public class FastJson2ObjectInput implements ObjectInput {
32
33     private final Fastjson2CreatorManager fastjson2CreatorManager;
34
35     private volatile ClassLoader classLoader;
36     private final InputStream is;
37
38 -    public FastJson2ObjectInput(Fastjson2CreatorManager fastjson2CreatorManager, InputStream
+    public FastJson2ObjectInput(Fastjson2CreatorManager fastjson2CreatorManager,
   in) {
39
40         this.fastjson2CreatorManager = fastjson2CreatorManager;
41
42         this.classLoader = Thread.currentThread().getContextClassLoader();
43         this.is = in;
44         fastjson2CreatorManager.setCreator(classLoader);
+-- @@ -107,8 +111,9 @@ public <T> T readObject(Class<T> cls) throws IOException {
107             if (read != length) {
108                 throw new IllegalArgumentException("deserialize failed. expected read length: "
+ length + " but actual read: " + read);
109             }
110 -            return (T) JSONB.parseObject(bytes, Object.class,
+ JSONReader.Feature.SupportAutoType,
111                 JSONReader.Feature.UseDefaultConstructorAsPossible,
112                 JSONReader.Feature.UseNativeObject,
113                 JSONReader.Feature.FieldBased);
114     }
32
33     private final Fastjson2CreatorManager fastjson2CreatorManager;
34
35 +    private final Fastjson2SecurityManager fastjson2SecurityManager;
36 +
37     private volatile ClassLoader classLoader;
38     private final InputStream is;
39
40 +    public FastJson2ObjectInput(Fastjson2CreatorManager fastjson2CreatorManager,
+                               Fastjson2SecurityManager fastjson2SecurityManager,
+                               InputStream in) {
41         this.fastjson2CreatorManager = fastjson2CreatorManager;
42         this.fastjson2SecurityManager = fastjson2SecurityManager;
43         this.classLoader = Thread.currentThread().getContextClassLoader();
44         this.is = in;
45         fastjson2CreatorManager.setCreator(classLoader);
46
+-- @@ -111,9 +115,11 @@ public <T> T readObject(Class<T> cls) throws IOException {
111             if (read != length) {
112                 throw new IllegalArgumentException("deserialize failed. expected read length: "
+ length + " but actual read: " + read);
113             }
114 +            return (T) JSONB.parseObject(bytes, Object.class,
+                               fastjson2SecurityManager.getSecurityFilter(),
115                 JSONReader.Feature.UseDefaultConstructorAsPossible,
116                 JSONReader.Feature.ErrorOnNoneSerializable,
117                 JSONReader.Feature.UseNativeObject,
118                 JSONReader.Feature.FieldBased);
119     }
```

◆ Pojo Class Recovery

通常应用在恢复Pojo类时为了兼容无setter方法的属性会使用反射来直接恢复属性，当其恢复类实例时如使用默认的构造方法，即可满足我们的要求

◆ Lead to CVE-2023-23638[2]

当Dubbo服务提供者收到一个泛化调用请求时，GenericFilter会对请求进行处理，将请求中的参数转换为对应的Java对象，并将其传递给实际的服务实现类进行处理，在dubbo泛化调用时将其模式设置为raw.return后，在还原Pojo类时将使用默认的构造方法，并使用反射来恢复属性

```
private static Object realize0(Object pojo, Class <? > type, Type genericType, final Map < Object, Object > history) {  
    //code  
    dest = newInstance(type);  
    //code  
    Field field = getField(dest.getClass(), name);  
    //code  
    field.set(dest, value);  
}
```

◆ Lead to CVE-2023-23638[2]

其泛化调用时也有着自己的黑名单，位于`serialize.blockedlist`，在3.1.4版本前，`sun.print`包不在黑名单的范围内，因此这里符合利用`UnixPrintServiceLookup`的条件，利用泛化调用时的类实例化来执行命令；为了修复该问题，官方在后续的版本中对类实例化使用了黑白名单结合的过滤形式；使得后续的版本中整体的安全性大大提升。

```
158 org.mortbay.jetty.  
159 org.mozilla.javascript  
160 org.objectweb.asm.  
161 org.osjava.sj.  
162 org.python.core  
163 org.quartz.  
164 org.slf4j.  
165 org.springframework.  
166 - org.yaml.snakeyaml.tokens.directiveToken  
167 - sun.rmi.server.unicastref  
    ⊖
```

```
181 org.mortbay.jetty.  
182 + org.mortbay.log.  
183 org.mozilla.javascript  
184 org.objectweb.asm.  
185 org.osjava.sj.  
186 org.python.core  
187 org.quartz.  
188 org.slf4j.  
189 org.springframework.  
190 + org.thymeleaf.  
191 + org.yaml.snakeyaml.tokens.  
192 + pstore.shaded.org.apache.commons.collections.  
193 + sun.print.  
194 + sun.rmi.server.  
195 + sun.rmi.transport.  
196 + weblogic.ejb20.internal.  
197 + weblogic.jms.common.
```

◆ Lead to Flink RCE

当向/v1/jobs路由发送文件上传的数据包时，会对上传的数据进行反序列化操作，起始点位于
org.apache.flink.runtime.rest.handler.job.JobSubmitHandler#loadJobGraph

```
private CompletableFuture<JobGraph> loadJobGraph(JobSubmitRequestBody requestBody, Map < String, Path > nameToFile)
throws MissingFileException {
    final Path jobGraphFile = getPathAndAssertUpload(requestBody.jobGraphFileName, FILE_TYPE_JOB_GRAPH, nameToFile);
    return CompletableFuture.supplyAsync(
        () -> {
            JobGraph jobGraph;
            try(ObjectInputStream objectIn = new ObjectInputStream(jobGraphFile.getFileSystem().open(jobGraphFile))) {
                jobGraph = (JobGraph) objectIn.readObject();
            }
            catch(Exception e) {
                throw new CompletionException(new RestHandlerException("Failed to deserialize JobGraph.", HttpStatus.BAD_REQUEST, e));
            }
            return jobGraph;
        }, executor);
}
```

这里可以使用org.apache.flink.api.common.state.StateDescriptor中的readObject来选择反序列化器进行二次反序列化，当选择PojoSerializer反序列化器时，会使用无参构造函数来创建对象，同时使用反射的方式来恢复属性

◆ Lead to Flink RCE

```
//PojoSerializer
public T deserialize(DataInputView source) throws IOException {
    //code
    target = createInstance();
    //code
    Object field = fieldSerializers[i].deserialize(source);
    fields[i].set(target, field);
    //code
}

public T createInstance() {
    if(clazz.isInterface() || Modifier.isAbstract(clazz.getModifiers())) {
        return null;
    }
    try {
        T t = clazz.newInstance();
        initializeFields(t);
        return t;
    }
    catch(Exception e) {
        throw new RuntimeException("Cannot instantiate class.", e);
    }
}
```

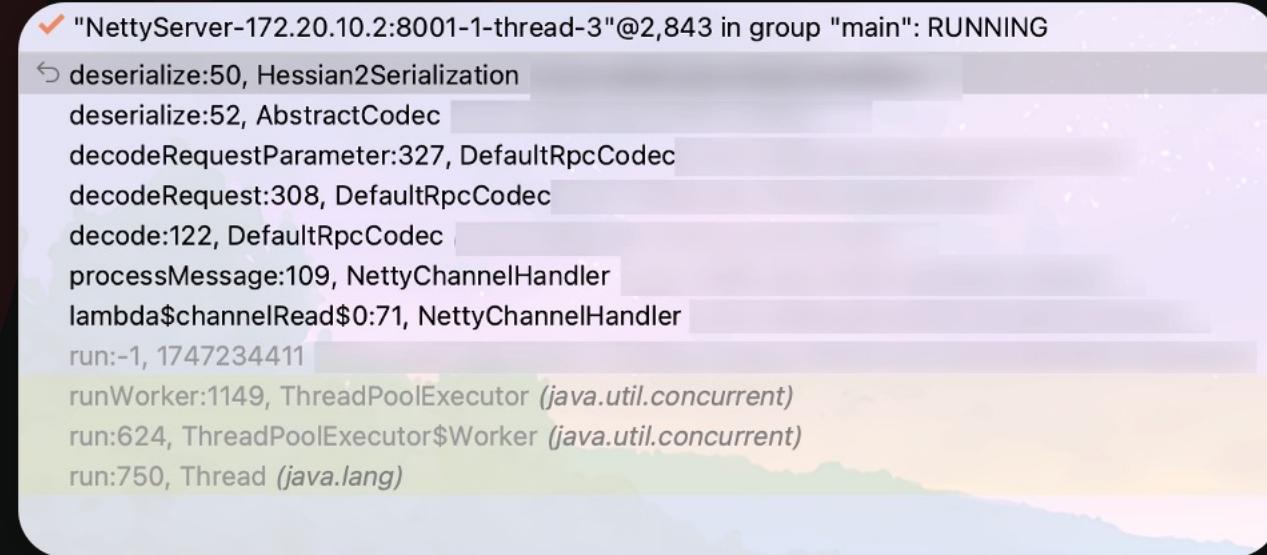
在这里完全适用UnixPrintServiceLookup 的利用条件，将看似“鸡肋”的反序列化变得不再鸡肋

- ◆ Member of gadget chain

由于UnixPrintServiceLookup中执行命令的方法会由getDefaultPrintService方法调用到，且该方法为public方法，因此在反序列化中可以充当getter的gadget的角色，在实际利用中，通常会结合Fastjson、Rome等组件结合使用

◆ Safety Case

某开源的分布式远程服务调用(RPC)框架中，在其进行远程方法调用的过程中，会默认使用hessian2进行数据序列化与反序列化，同时在反序列化前并不会对远程是否存在客户端所请求的方法进行校验；



```
✓ "NettyServer-172.20.10.2:8001-1-thread-3"@2,843 in group "main": RUNNING
  ↵ deserialize:50, Hessian2Serialization
    deserialize:52, AbstractCodec
    decodeRequestParameter:327, DefaultRpcCodec
    decodeRequest:308, DefaultRpcCodec
    decode:122, DefaultRpcCodec
    processMessage:109, NettyChannelHandler
    lambda$channelRead$0:71, NettyChannelHandler
    run:-1, 1747234411
    runWorker:1149, ThreadPoolExecutor (java.util.concurrent)
    run:624, ThreadPoolExecutor$Worker (java.util.concurrent)
    run:750, Thread (java.lang)
```

◆ Safety Case

即使不知道远程服务端的方法，也可以进行反序列化的利用，通过查询其依赖可以发现存在着fastjson的依赖；由于其没有反序列化安全过滤，可以直接通过fastjson中的JSONObject.toString来调用UnixPrintServiceLookup的getDefaultPrintService方法，进而执行命令；

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.83</version>
</dependency>
```



归源 · 智变

Returning to the source · Intelligent transformation

RASP攻防Tricks

◆ RASP攻防Tricks

根据UnixPrintServiceLookup实例化时新建线程的特性，对RASP的检测规则能够轻松的绕过，
demo如下

```
UnixPrintServiceLookup lookup = new UnixPrintServiceLookup();
Field lpcFirstCom = UnixPrintServiceLookup.class.getDeclaredField("lpcFirstCom");
lpcFirstCom.setAccessible(true);
lpcFirstCom.set(lookup,new String[]{"touch /tmp/pwned","touch /tmp/pwned","touch /tmp/pwned"});
```

◆ RASP攻防Tricks

上面的操作中显式的使用了反射来修改值，假如存在着一些漏洞场景时，可以有如下更隐蔽的利用
如在spel + fastjson的场景下，可以无感的使用 fastjson特性来恢复属性

```
T(com.alibaba.fastjson.JSON).parseObject('{"lpcFirstCom":["touch /tmp/pwned","touch /tmp/pwned","touch /tmp/pwned"]}',  
T(sun.print.UnixPrintServiceLookup),  
T(com.alibaba.fastjson.parser.Feature).SupportNonPublicField)
```

减缓措施

◆ 减缓措施

对于 RPC服务来说

- 对泛化调用进行严格的类型检查
- 对通信时使用的序列化器使用黑白名单配置
- 使用安全性较高的序列化器

对于其他应用来说

- 对用户可控的类实例化操作进行严格校验和检查
- 对危险类对象的恢复进行检查



归源·智变
Returning to the source · Intelligent transformation

感谢您的观看！

THANK YOU FOR YOUR WATCHING

