Java反序列化漏洞自动挖掘方法

蚂蚁非攻安全实验室 codeplutos

HackingDay in HangZhou , 2019

目录
CATALOG

关于我

- codeplutos
- 南开大学
- 蚂蚁非攻安全实验室 安全工程师
- Java框架安全研究、前RASP开发、程序分析爱好者
- https://c0d3p1ut0s.github.io/

目录
CATALOG

序列化与反序列化

定义：内存中的对象与可存储在磁盘上的持久化数据相互转换的过程。

各编程语言都存在：

- Java： java.io.Serializable接口、fastjson、jackson、gson
- PHP： serialize()、 unserialize()
- Python： pickle

Java反序列化过程

1. 对象实例化

   - sun.misc.Unsafe#allocateInstance
   - 通过反射调用构造函数

```
0: new           // class com/alipay/Test
3: dup
4: invokespecial // Method "<init>":()V
```

2. 成员变量还原

   - Setter和getter方法
   - 通过反射直接设置
   - 成员变量的处理(例如：PriorityQueue)

```
private void readObject(ObjectInputStream var1) throws IOException, ClassNotFoundException {
        var1.defaultReadObject();
        var1.readInt();
        SharedSecrets.getJavaOISAccess().checkArray(var1, Object[].class, this.size);
        this.queue = new Object[this.size];

        for(int var2 = 0; var2 < this.size; ++var2) {
            this.queue[var2] = var1.readObject();
        }

        this.heapify();
    }
```

Java反序列化漏洞（PriorityQueue）

```
private void readObject(ObjectInputStream var1) throws IOException, ClassNotFoundException {
        //省略
        this.heapify();
    }
```

```
private void heapify() {
        for(int var1 = (this.size >>> 1) - 1; var1 >= 0; --var1) {
            this.siftDown(var1, this.queue[var1]);
        }
    }
```

省略部分调用

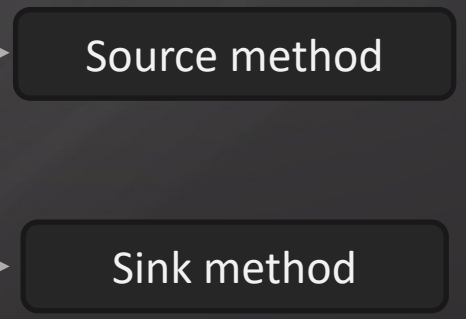
```
public Object transform(Object input) {
  //省略
    Class cls = input.getClass();
    Method method = cls.getMethod(iMethodName, iParamTypes);
    return method.invoke(input, iArgs);
  //省略
    }
```

```
    public Object invoke(Object var1, Object... var2) throws IllegalAccessException,
IllegalArgumentException, InvocationTargetException {
        if (!this.override && !Reflection.quickCheckMemberAccess(this.clazz, this.modifiers)) {
            Class var3 = Reflection.getCallerClass();
            this.checkAccess(var3, this.clazz, var1, this.modifiers);
        }
        MethodAccessor var4 = this.methodAccessor;
        if (var4 == null) {
            var4 = this.acquireMethodAccessor();
        }
        return var4.invoke(var1, var2);
    }
```

Java反序列化漏洞挖掘

寻找一个类，通过构造一个对象，使其在被反序列化时能执行到危险（sink）方法。

```
/*
    Gadget chain:
        ObjectInputStream.readObject()
            PriorityQueue.readObject()
                ...
                    TransformingComparator.compare()
                        InvokerTransformer.transform()
                            Method.invoke()
                                Runtime.exec()
 */
```



Java反序列化漏洞挖掘

寻找一个类，通过构造一个对象，使其在被反序列化时能执行到危险（sink）方法。

```
/*
    Gadget chain:
        ObjectInputStream.readObject()
            PriorityQueue.readObject()
                ...
                    TransformingComparator.compare()
                        InvokerTransformer.transform()
                            Method.invoke()
                                Runtime.exec()
 */
```

- 命题1. 寻找一个类，存在可能的执行路径，从反序列化入口（source）方法执行到危险（sink）方法
- 命题2. 构造这个对象，使危险（sink）方法参数可控。

目录
CATALOG

Java反序列化漏洞挖掘

寻找一个类，通过构造一个对象，使其在被反序列化时能执行到危险（sink）方法。

```
/*
    Gadget chain:
        ObjectInputStream.readObject()
            PriorityQueue.readObject()
                ...
                    TransformingComparator.compare()
                        InvokerTransformer.transform()
                            Method.invoke()
                                Runtime.exec()
 */
```

自动化搜索

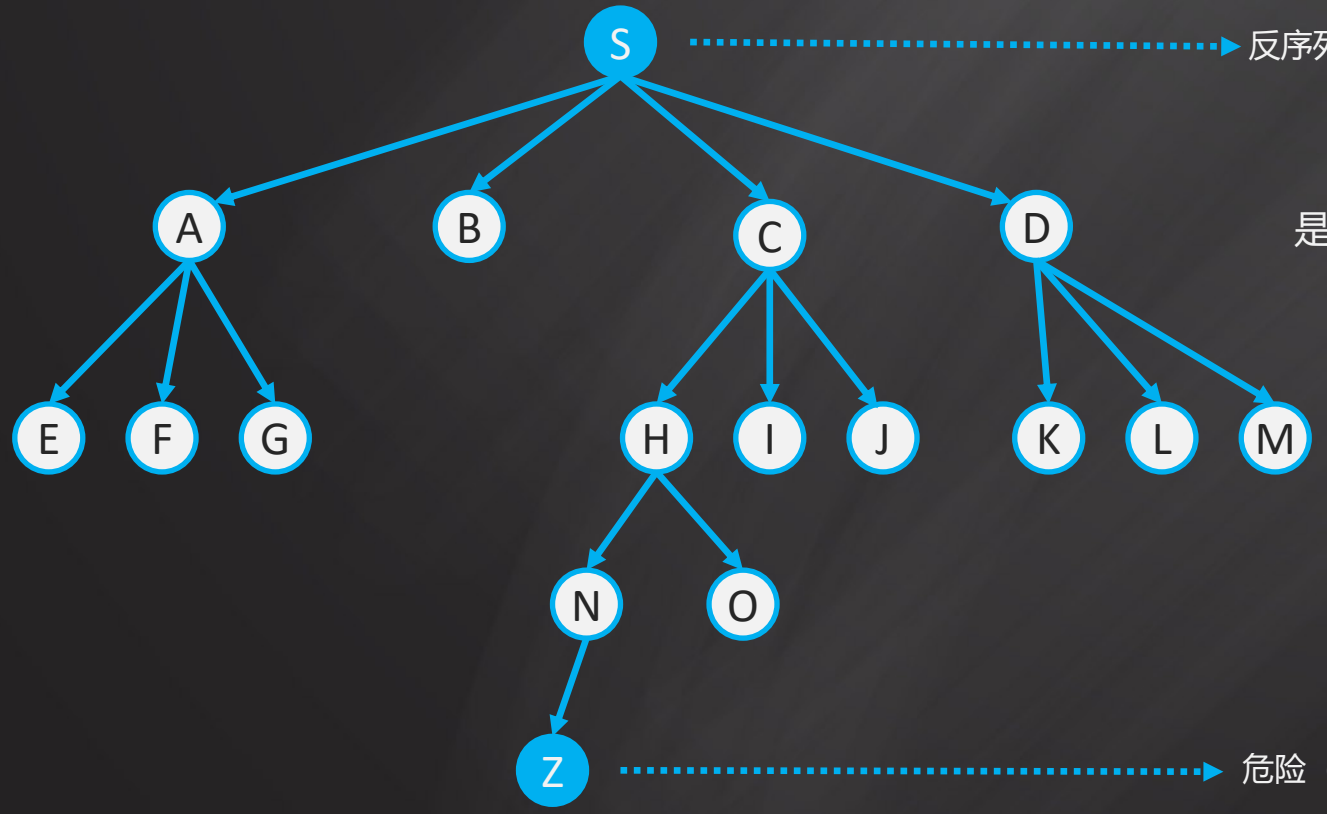
- 命题1. 寻找一个类，存在可能的执行路径，从反序列化入口（source）方法执行到危险（sink）方法

- 命题2. 构造这个对象，使危险（sink）方法参数可控。

手工构造

典型的调用链搜索问题

在静态分析中，这是一个典型的可达性分析问题。



反序列化入口（source）方法

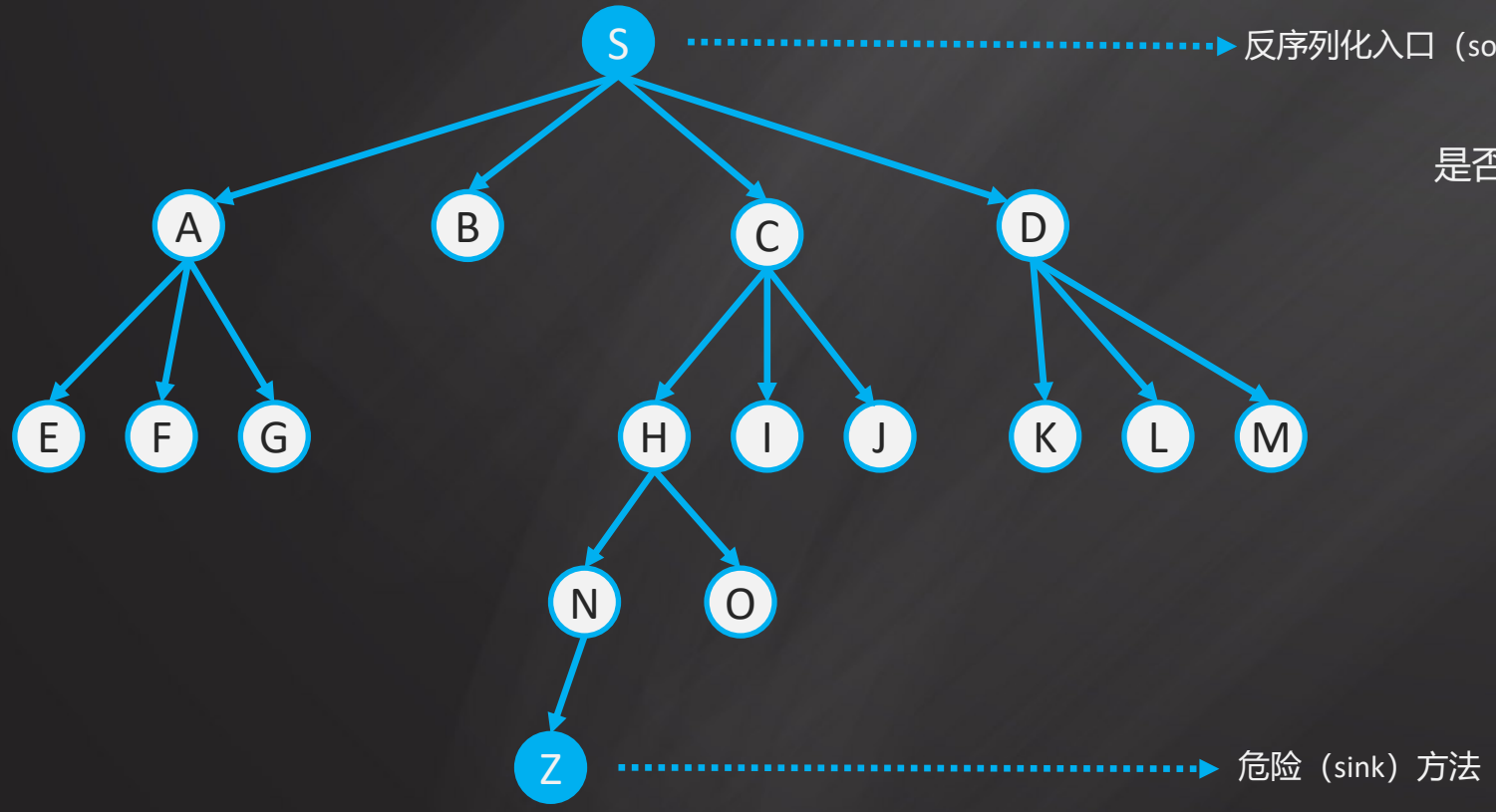
是否可能（may）存在S – C – H – N – Z这样的路径

危险（sink）方法

调用树

典型的调用链搜索问题

可达性分析 - may分析：无需绘制控制流图，只需搜素调用树。



调用树

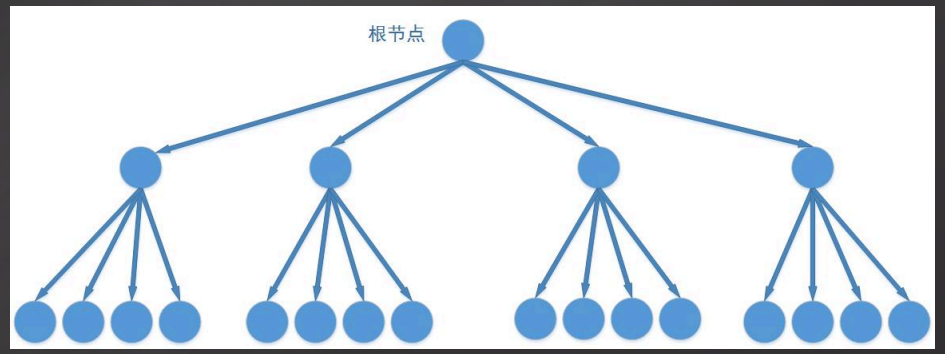
是否可能（may）存在S – C – H – N – Z这样的路径

```
public void doTheWork() {
   Object o = null;
   for (int i=0; i<5; i++) {
      try {
         o = makeObj(i);
      }
      catch (IllegalArgumentException e) {
        e.printStackTrace();
        return;
      }
      finally {
         System.out.println(o);
      }
   }
}
```

调用树搜索实现

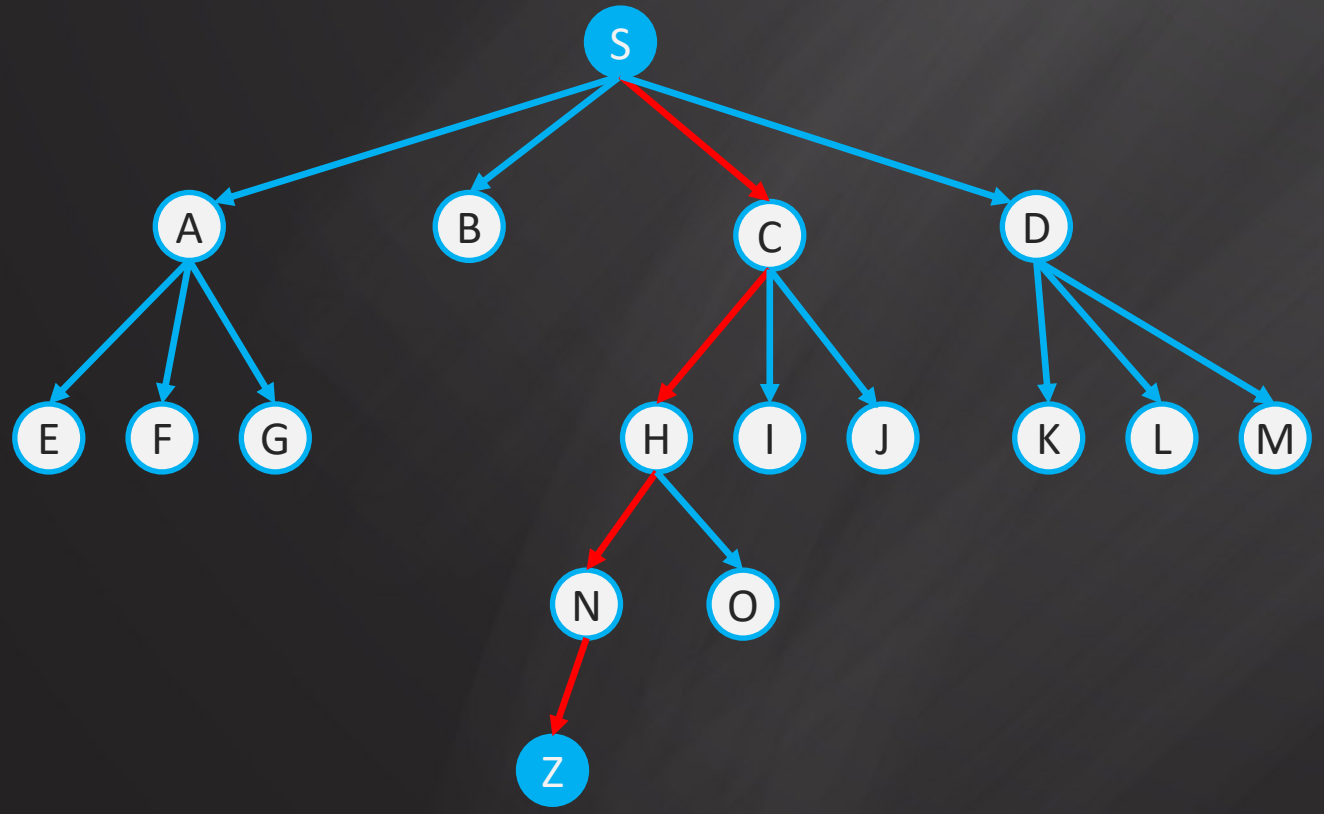
深度优先搜索（DFS） vs 广度优先搜索（BFS）

- 调用路径越长，payload越难构造
- 搜索深度有限
- 等价于搜索一个n叉树（n>100）的前几层
- 调用链的存储



调用树搜索实现

深度优先搜索（DFS）



调用树

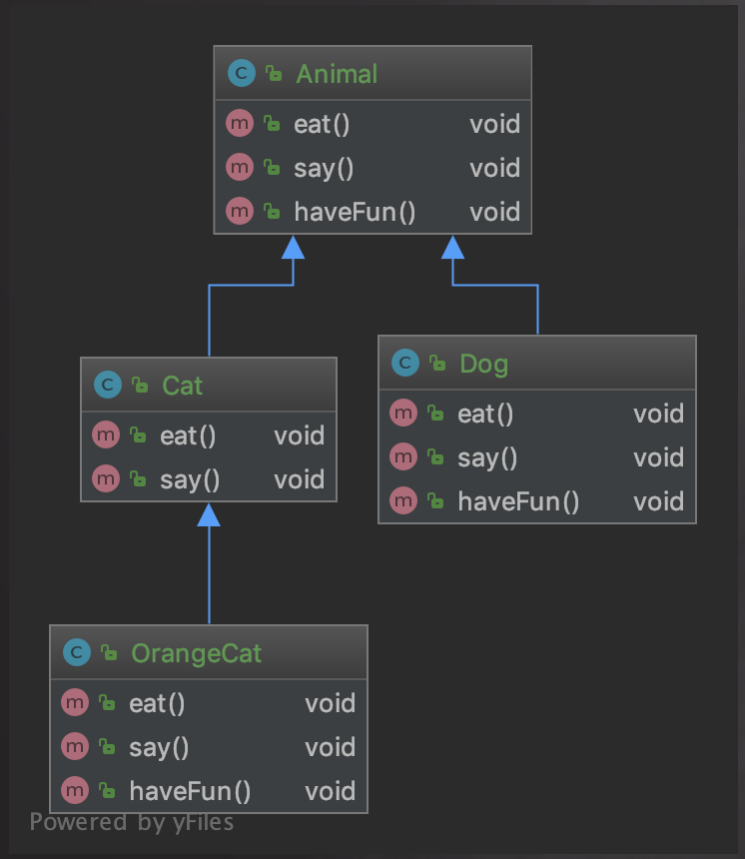
搜索停止条件：

- 到达指定深度
- 搜索到sink方法

搜索结果保存：

- 使用stack保存路径

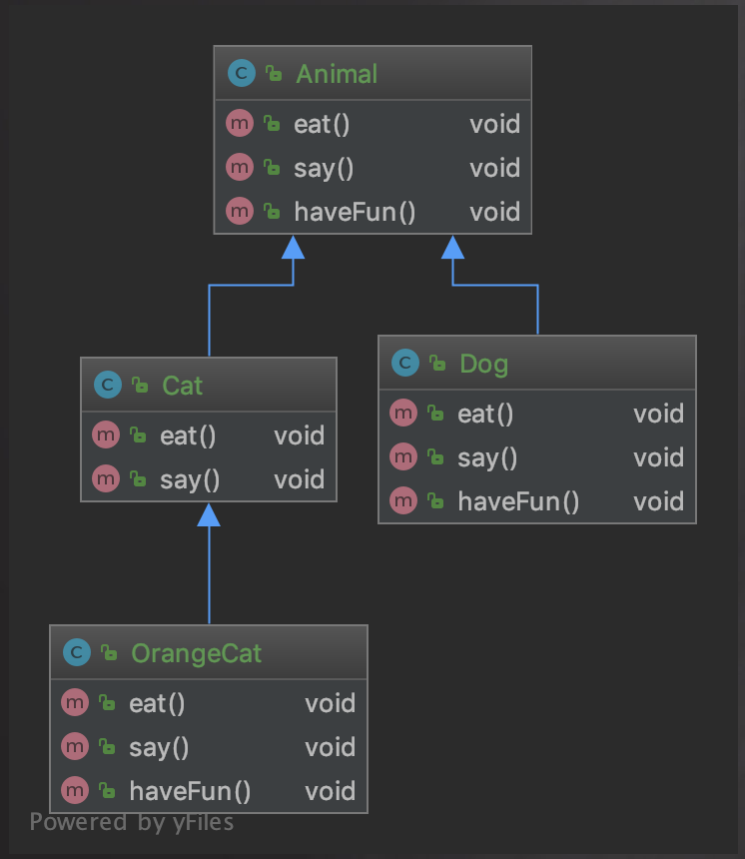
搜索中的多态问题




```
public void doStuff(Animal animal){
        animal.eat();
    }
```

由于面向对象中多态性的存在，只有在运行时才能确定调用哪个子类的eat方法。

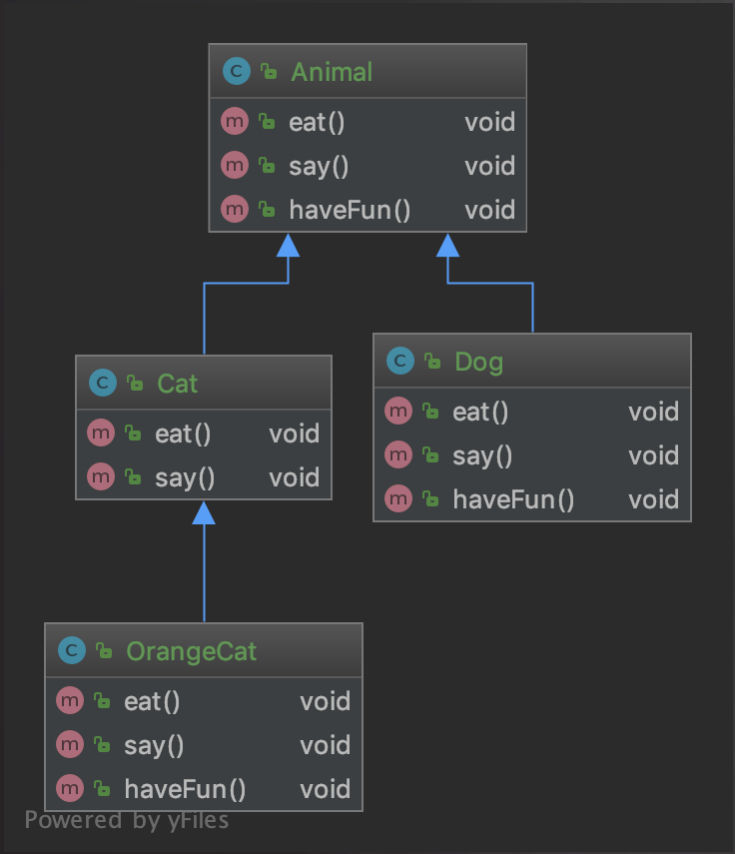
搜索中的多态问题




```
public void doStuff(Cat cat){
    cat.haveFun();
}
```

Dog#haveFun方法可能被调用吗?

搜索中的多态问题




```
public void doStuff(Cat cat){
    cat.haveFun();
}
```

多态的处理：

1. 构建类、接口和方法继承树（双向树）
2. 寻找调用的方法的实现所在类的所有子类集合
3. 在上述集合中寻找调用者类的子类的集合
4. 这些子类中重写的方法即为所有可能调用的方法

路径成环

```
public class CircleChain {
    private Object object;
    private String string;

    public int hashCode() {
        return 31 * object.hashCode() + string.hashCode();
    }
}
```

搜索到CircleChain的hashCode方法时，这个方法调用了Object#hashCode方法，寻找Object的子类会再找到CircleChain类，形成环。

路径爆炸

```
public Node node(int index) {
        Object object = contentList().get(index);
        if (object instanceof Node) {
            return (Node) object;
        }
        if (object instanceof String) {
            return getDocumentFactory().createText(object.toString());
        }
        return null;
    }
```

Java.util.List#get方法
Java.lang.Object#toString方法

```
private class NamespaceIterator implements Iterator {
        private Iterator iter;

        public NamespaceIterator(Iterator iter) {
            this.iter = iter;
        }

        public boolean hasNext() {
            return iter.hasNext();
        }
}
```

java.util.Iterator#hasNext方法

寻找这些方法的实现造成路径爆炸

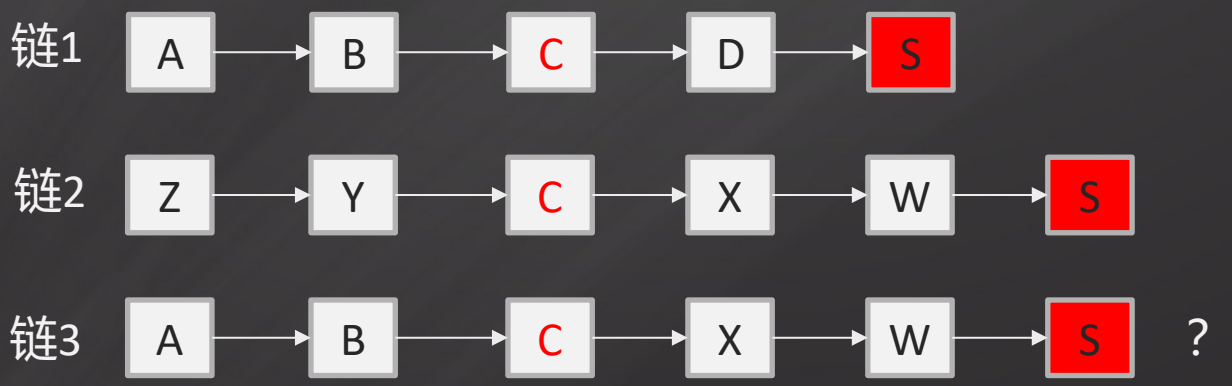
路径爆炸成环问题解决

1. 搜索深度限制（兜底）

2. 已搜索方法缓存
   1. 先缓存、后搜索
   2. 缓存方法signature

3. 人工介入，加入黑名单

4. 调用链缓存



只需要构造C方法执行时的上下文，使其与链2一致即可

工具效果

蚂蚁内部反序列化库

- Classpath：蚂蚁内部常用二方、三方依赖包
- 多条反序列化调用链

Jackson

- Classpath：蚂蚁内部常用二方、三方依赖包
- CVE-2019-12814

目录
CATALOG

Jackson简介

Jackson是一个开源的Java序列化与反序列化工具，可以将java对象序列化为xml或json格式的字符串，或者反序列化回对应的对象，由于其使用简单，速度较快，且不依靠除JDK外的其他库，被众多用户所使用。

Jackson也是Spring MVC默认的json解析库，打开多态之后，jackson会根据json中传入的类名进行反序列化

相比其他后来开发的json解析库来说，jackson有灵活的API，可以很容易根据需要进行扩展和定制。

Jackson历史漏洞

CVE-2017-7525：RCE

CVE-2017-17485：RCE

CVE-2018-14718：RCE

CVE-2019-12086：任意文件读取

CVE-2019-12384：RCE（要求反序列化后再序列化payload）

CVE-2019-12814：XXE

在安全研究人员的努力下，常见库中的gadget越来越少，危害越来越低

CVE-2019-14379：RCE （要求反序列化后再序列化）

Jackson反序列化过程

```
[
    "com.alipay.Person",
    {
        "age": 10,
        "name": "Alice",
        "dna": [
            "com.alipay.Dna",
            {
                "length": 100
            }
        ]
    }
]
```

对象初始化

1. 调用类的无参初始化方法
2. 调用包含一个基础类型参数的构造函数，并且这个参数可控

对象中成员变量赋值

1. 将json看成key-value对，key与field不一定一一对应。
2. 首先看key是否存在setter方法，如果存在setter方法，则会通过反射调用setter方法
3. 否则看在这个类中是否存在与key同名的field，如果存在，则通过反射直接赋值。
4. 否则看是否存在对应的getter方法，且getter的返回值是Collection或者Map的子类，如果满足这个条件，则会调用这个getter方法
5. 如果以上条件都不满足，则抛出异常

反序列化的source method

Jackson反序列化显式调用的方法

- 仅包含一个基本类型参数的构造函数。
- Setter方法
- 返回值是Collection或者Map的子类的getter方法。

反序列化过程中隐式调用的方法

- hashCode
- compare

Jackson反序列化的sink method

命令执行：

- java.lang.reflect.Method#invoke
- javax.naming.Context#lookup
- javax.naming.Context#bind
- java.lang.Runtime#exec
- java.lang.ProcessBuilder#ProcessBuilder

文件读取：

- java.sql.Driver#connect MySQL客户端任意文件读取
- org.xml.sax.XMLReader#parse
- javax.xml.parsers.SAXParser#parse
- javax.xml.parsers.DocumentBuilder#parse

Jackson反序列化漏洞搜索结果

com.mysql.cj.jdbc.NonRegisteringDriver#connect(String, Properties)--> com.mysql.cj.jdbc.admin.MiniAdmin#MiniAdmin(String, Properties)--> com.mysql.cj.jdbc.admin.MiniAdmin#MiniAdmin(String)	CVE-2019-12086
com.sun.jndi.toolkit.url.GenericURLContext#lookup(String)--> javax.naming.InitialContext#lookup(String)--> com.sun.rowset.JdbcRowSetImpl#connect()--> com.sun.rowset.JdbcRowSetImpl#setAutoCommit(boolean)	CVE-2017-7525
javax.xml.parsers.SAXParser#parse(InputSource, DefaultHandler)--> org.mortbay.xml.XmlParser#parse(InputSource)--> org.mortbay.xml.XmlConfiguration#XmlConfiguration(String)	无
com.sun.xml.internal.fastinfoset.sax.SAXDocumentParser#parse(InputSource)--> org.apache.xalan.processor.TransformerFactoryImpl#newTemplates(Source)--> org.jdom.transform.XSLTransformer#XSLTransformer(Source)--> org.jdom.transform.XSLTransformer#XSLTransformer(String)	CVE-2019-12814

目录
CATALOG

与gadgetinspector比较

基于源代码搜索	基于字节码的搜索
无污染跟踪	有污染跟踪
深搜	广搜
有多态处理	部分多态处理 / 调用链多态处理不完全导致遗漏
缓存包含sink method的子链	所有节点仅遍历一次 / 调用链遗漏

优化

```
private void readObject(ObjectInputStream var1) throws IOException, ClassNotFoundException {
        //省略部分代码
        String var10;
        Object var11;
        for(Iterator var8 = var4.entrySet().iterator(); var8.hasNext(); var7.put(var10, var11)) {
            Entry var9 = (Entry)var8.next();
            var10 = (String)var9.getKey();
            var11 = null;
            Class var12 = (Class)var6.get(var10);
            if (var12 != null) {
                var11 = var9.getValue();
                if (!var12.isInstance(var11) && !(var11 instanceof ExceptionProxy)) {
                    var11 = (new AnnotationTypeMismatchExceptionProxy(var11.getClass() + "[" + var11 +
"]")).setMember((Method)var5.members().get(var10));
                }
            }
        }
        AnnotationInvocationHandler.UnsafeAccessor.setType(this, var3);
        AnnotationInvocationHandler.UnsafeAccessor.setMemberValues(this, var7);
    }
```

Var11.toString方法的隐式调用

1. 隐式调用单独分析

2. 基于字节码的自动挖掘

污染（taint）跟踪

定义：将感兴趣的变量进行标记（污染源），随着程序的运行，这些数据随之流动，更多的变量受污染源的影响，污染扩散；然后可以在一些敏感函数（sink method）处检查污染情况，判断sink method调用的上下文是否受污染源的影响。

静态污染跟踪

- Control Flow Analysis
- Call Graph
- Data Flow Analysis

动态污染跟踪

- Shadow Memory
- Instrument

gadgetInspector中的污染跟踪

静态污染跟踪

污染传播：

- 如果一个类是攻击者可控的，那么所有成员变量都是被污染的。

Control Flow Analysis：

- 一个方法中的每个分支都是可达的。

Data Flow Analysis：

- 方法的参数与方法的返回值的关系。

Call Graph：

- 方法的参数与调用的子方法参数的关系

```
public class FnDefault {
private FnConstant f;
public Object invoke(Object arg) {
  return arg != null ? arg :
  f.invoke(arg);
  }
}
```

题外话：污染跟踪的应用

- SAST
- IAST
- RASP

雷神众测认证白帽专属奖励

活动时间：2019年9月7日-11月30日
本次活动参与方式为网上提交漏洞
提交网址： security.alipay.com

丰厚奖励：
新白帽向AFSRC提交漏洞，首个有效漏洞**2倍奖金**（AFSRC1.5倍，雷神众测补贴0.5倍）举个例子：如在AFSRC确认的首个严重漏洞为1100个蚂蚁金币，可获得22000元（AFSRC发放16500，雷神众测发放5500）
注意：漏洞提交标题要写**#我是新人#**否则无法享受新人奖励哦。
新白帽在漏洞详情第五项【你的推荐人】里填写“雷神众测”，首个确认漏洞，将根据不同等级获得以下礼品：

非攻实验室招人啦

非攻安全实验室（N/A Lab）是成立一年有余的年轻团队，归属于蚂蚁金服支付宝事业群，秉承墨子非攻思想， 坚守正义，致力于让公司、整个生态更加安全。团队深耕业务线，在业务安全的一线迎接最高的安全挑战。

招聘安全专家，BASE杭州或成都
懂Java，懂工程
懂数据，懂算法
懂安全，懂业务

如果你有至少两项匹配，那我们这里是你大展才华的地方!

联系hb187361@antfin.com

Thank you

HackingDay in HangZhou , 2019