

唯品会  
一家专门做特卖的网站



唯品会安全应急响应中心  
VSRC

# 因唯安全 所以信赖

2017唯品会第二届电商安全峰会  
——深度揭秘唯品会信息安全建设实践

苏州

中国·苏州





# 伤口撕开，给你看

唯品会资深信息安全工程师  
沈海涛

# 开年的第二天



提交人

提交时间

2017-02-

漏洞名称

新年礼-VIP 服务器getshell-不吹牛逼

漏洞域名

vip.com

漏洞URL

http://vip.com/



# 序列化，又见序列化

November 6, 2015

# What Do WebLogic, JBoss, Jenkins, OpenNMS, and Your Application Have in Common?

By @breenmachine





# What



- Java 序列化将一个对象转换为流，反序列化则是将对象流转换为实际程序中使用的 Java 对象的过程。序列化可以用于轻量级的持久化、通过 Sockets 进行传输、或者用于 Java RMI。可序列化的对象需要实现 `java.io.Serializable` 接口或者 `java.io.Externalizable` 接口。
- RMI是Remote Method Invocation的简称，是J2SE的一部分，能够让程序员开发出基于Java的分布式应用。一个RMI对象是一个远程Java对象，可以从另一个Java虚拟机上（甚至跨过网络）调用它的方法，可以像调用本地Java对象的方法一样调用远程对象的方法，使分布在不同的JVM中的对象的外表和行为都像本地对象一样。
- 而 RMI 对于远程对象的调用使用了 Java 的序列化和反序列化。

# What

```
public class Staff implements Serializable{  
    private static final long serialVersionUID = -2834558021263531425L;  
    private String name;  
    private int salary;  
  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getSalary() {  
        return salary;  
    }  
    public void setSalary(int salary) {  
        this.salary = salary;  
    }  
}
```

```
public class Staff implements Serializable{  
    private static final long serialVersionUID  
    private String name;  
    private int salary;
```

# What

```
@Test
public void testSerialize() throws Exception {
    Staff staff = new Staff();
    staff.setName("Boss");
    staff.setSalary(100000);
    //将 staff 序列化, 并保存到 staff.ser 文件中
    ObjectOutputStream outputStream =
        new ObjectOutputStream(new FileOutputStream("staff.ser"));
    outputStream.writeObject(staff);
    outputStream.close();
    //从 staff.ser 文件中反序列化对象, 并打印对象
    ObjectInputStream objectInputStream =
        new ObjectInputStream(new FileInputStream("staff.ser"));
    Staff staff2 = (Staff) objectInputStream.readObject();
    System.out.println(staff2);
    objectInputStream.close();
}
```

outputStream.writeObject(staff);  
objectInputStream.readObject();

<terminated> TestStaff.testSerialize [JUnit] C  
[name=Boss, salary=100000]



# Staff.ser



```
00000000: aced 0005 7372 0024 636f 6d2e 7669 7073 ....sr..com.vips
00000010: 686f 702e 7675 6c73 2e73 6572 6961 6c69 hop.vuls.seriali
00000020: 7a61 7469 6f6e 2e53 7461 6666 d8a9 a07f zation.Staff....
00000030: 6f5a 625f 0200 0249 0006 7361 6c61 7279 oZb_...I..salary
00000040: 4c00 046e 616d 6574 0012 4c6a 6176 612f L..namet..Ljava/
00000050: 6c61 6e67 2f53 7472 696e 673b 7870 0001 lang/String;xp..
00000060: 86a0 7400 0442 6f73 73 ..t..Boss
```



# head



```
00000000:  aced 0005 7372 0024 636f 6d2e 7669 7073  ....sr..com.vips
00000010:  686f 702e 7675 6c73 2e73 6572 6961 6c69  hop.vuls.seriali
00000020:  7a61 7469 6f6e 2e53 7461 6666 d8a9 a07f  zation.Staff....
00000030:  6f5a 625f 0200 0249 0006 7361 6c61 7279  oZb_...I..salary
00000040:  4c00 046e 616d 6574 0012 4c6a 6176 612f  L..namet..Ljava/
00000050:  6c61 6e67 2f53 7472 696e 673b 7870 0001  lang/String;xp..
00000060:  86a0 7400 0442 6f73 73          ..t..Boss
```



# ClassDesc



```
00000000: aced 0005 7372 0024 636f 6d2e 7669 7073 .....sr..com.vips
00000010: 686f 702e 7675 6c73 2e73 6572 6961 6c69 hop.vuls.seriali
00000020: 7a61 7469 6f6e 2e53 7461 6666 d8a9 a07f zation.Staff....
00000030: 6f5a 625f 0200 0249 0006 7361 6c61 7279 oZb_...I..salary
00000040: 4c00 046e 616d 6574 0012 4c6a 6176 612f L..namet..Ljava/
00000050: 6c61 6e67 2f53 7472 696e 673b 7870 0001 lang/String;xp..
00000060: 86a0 7400 0442 6f73 73 ..t..Boss
```



# ClassData



```
00000000: aced 0005 7372 0024 636f 6d2e 7669 7073 ....sr..com.vips
00000010: 686f 702e 7675 6c73 2e73 6572 6961 6c69 hop.vuls.seriali
00000020: 7a61 7469 6f6e 2e53 7461 6666 d8a9 a07f zation.Staff....
00000030: 6f5a 625f 0200 0249 0006 7361 6c61 7279 oZb_...I..salary
00000040: 4c00 046e 616d 6574 0012 4c6a 6176 612f L..namet..Ljava/
00000050: 6c61 6e67 2f53 7472 696e 673b 7870 0001 lang/String;xp..
00000060: 86a0 7400 0442 6f73 73 ..t..Boss
```

# 结构

```
1  STREAM_MAGIC (2 bytes) 0xACED
2  STREAM_VERSION (2 bytes) 0x0005
3  newObject
4      TC_OBJECT (1 byte) 0x73
5      newClassDesc
6          TC_CLASSDESC (1 byte) 0x72
7          className
8              length (2 bytes) 0x0024 = 36
9              text (36 bytes) com.vipshop.vuls.serialization.Staff
10         serialVersionUID (8 bytes) 0xd8a9a07f6f5a625f = -2834558021263531425L
11         classDescInfo
12             classDescFlags (1 byte) 0x02 = SC_SERIALIZABLE
13             fields
14                 count (2 bytes) 0x0002 = 2
15                 field[0]
16                     primitiveDesc
17                         prim_typecode (1 byte) 0x49 I = integer
18                         fieldName
19                             length (2 bytes) 0x0006 = 6
20                             text (6 bytes) 0x73616c617279 = salary
21                             text (6 bytes) 0x73616c617279 = salary
22                     field[1]
23                         objectDesc
24                             obj_typecode (1 byte) 0x4C L = object
25                             fieldName
26                                 length (2 bytes) 0x0004 = 4
27                                 text (4 bytes) 0x73616c617279 = name
28                                 className1
29                                     TC_STRING (1 byte) 0x74
30                                         length (2 bytes) 0x12 = 18
31                                         text (18 bytes) 0x4c6a6176612f6c616e672f537472693b =Ljava/lang/String;
32                                 classAnnotation
33                                     TC_ENDBLOCKDATA (1 byte) 0x78
34                                     superClassDesc
35                                         TC_NULL (1 byte) 0x70
36                     classdata[]
37                         classdata[0] (4 bytes) 0x000186a0 100000 = salary
38                         classdata[1]
39                             TC_STRING (1 byte) 0x74
40                                 length (2 bytes) 0x0004 = 4
41                                 text (4 bytes) 0x426f7373 Boss
```



# What

```
import java.rmi.Remote;

public interface IMyRmi extends Remote{
    String sayHello(String name) throws RemoteException;
}
```

```
public class MyRmiImpl extends UnicastRemoteObject implements IMyRmi{

    protected MyRmiImpl() throws RemoteException {
        super();
    }

    private static final long serialVersionUID = 4958973123281164759L;

    @Override
    public String sayHello(String name) throws RemoteException {
        return "Hello " + name + " ^^ ";
    }

}
```

# What

```
public class Run {  
  
    public static void main(String[] args) {  
        try {  
            IMyRmi myRemote = new MyRmiImpl();  
            LocateRegistry.createRegistry(6600);  
            Naming.rebind("rmi://127.0.0.1:6600/hello", myRemote);  
            System.out.println("ready.....");  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

ready.....

```
IMyRmi service = (IMyRmi) Naming.lookup("rmi://127.0.0.1:6600/hello");  
String s = service.sayHello("zhangsan");  
System.out.println(s);
```

Hello zhangsan ^\_^

```
IMyRmi myRemote = new MyRmiImpl();  
LocateRegistry.createRegistry(6600);  
Naming.rebind("rmi://127.0.0.1:6600/hello", myRemote);
```

```
import java.rmi.Naming;  
  
public class MyRmiClient {  
  
    public static void main(String[] args) {  
        try {  
            IMyRmi service = (IMyRmi) Naming.lookup("rmi://127.0.0.1:6600/hello");  
            String s = service.sayHello("zhangsan");  
            System.out.println(s);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```



# How



- 敏感信息泄露
- 信息伪造
- 拒绝服务
- 代码执行



# 敏感信息泄露



0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
ac	ed	00	05	73	72	00	24	63	6f	6d	2e	76	69	70	73	草..sr.\$com.vips[
68	6f	70	2e	76	75	6c	73	2e	73	65	72	69	61	6c	69	hop.vuls.seriali
7a	61	74	69	6f	6e	2e	53	74	61	66	66	d8	a9	a0	7f	zation.Staff丞..[
6f	5a	62	5f	02	00	02	49	00	06	73	61	6c	61	72	79	oZb_...I..salary
4c	00	04	6e	61	6d	65	74	00	12	4c	6a	61	76	61	2f	L..namet..Ljava/
6c	61	6e	67	2f	53	74	72	69	6e	67	3b	78	70	00	01	lang/String;xp..
86	a0	74	00	04	42	6f	73	73								?t..Boss□



# 信息伪造



Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	ac	ed	00	05	73	72	00	24	63	6f	6d	2e	76	69	70	73	草..sr.\$com.vips[
00000010	68	6f	70	2e	76	75	6c	73	2e	73	65	72	69	61	6c	69	hop.vuls.seriali
00000020	7a	61	74	69	6f	6e	2e	53	74	61	66	66	d8	a9	a0	7f	zation.Staff丞..[
00000030	6f	5a	62	5f	02	00	02	49	00	06	73	61	6c	61	72	79	oZb_...I..salary
00000040	4c	00	04	6e	61	6d	65	74	00	12	4c	6a	61	76	61	2f	L..namet..Ljava/
00000050	6c	61	6e	67	2f	53	74	72	69	6e	67	3b	78	70	00	03	lang/String;xp..
00000060	0d	40	74	00	04	42	6f	73	73								.@t..Boss

```
<terminated> TestStaff.testSpoofSerialize  
[name=Boss, salary=200000]
```

# 拒绝服务

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	11	11	11	05	73	72	00	24	63	6f	6d	2e	76	69	70	73	...sr.\$com.vips
00000010	68	6f	70	2e	76	75	6c	73	2e	73	65	72	69	61	6c	69	hop.vuls.seriali
00000020	7a	61	74	69	6f	6e	2e	53	74	61	66	66	d8	a9	a0	7f	zation.StaffZK..[
00000030	6f	5a	62	5f	02	00	02	49	00	06	73	61	6c	61	72	79	oZb_...I..salary
00000040	4c	00	04	6e	61	6d	65	74	00	12	4c	6a	61	76	61	2f	L..namet..Ljava/
00000050	6c	61	6e	67	2f	53	74	72	69	6e	67	3b	78	70	00	03	lang/String;xp..
00000060	0d	40	74	00	04	42	6f	73	73								.@t..Boss

## Failure Trace

```
java.io.StreamCorruptedException: invalid stream header: 11111105
    at java.io.ObjectInputStream.readStreamHeader(Unknown Source)
    at java.io.ObjectInputStream.<init>(Unknown Source)
    at com.vipshop.vuls.serialization.TestStaff.testDosSerialize(TestStaff.java:46)
```

# 命令执行

```
@Test
public void testEvilSerialize() throws FileNotFoundException, IOException, ClassNotFoundException{

    //从calc.ser 文件中反序列化对象，并打印对象
    ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream("calc.ser"));
    Staff staff2 = (Staff) objectInputStream.readObject();
    System.out.println(staff2);
    objectInputStream.close();
}
```



其中 calc.ser 是使用 ysoserial 工具生成的 CommonsCollections 的命令执行序列化对象，生成方法：

```
java -jar ysoserial-0.0.4-all.jar CommonsCollections1 "calc" > calc.ser
```

此时，我们运行这个测试用例的时候，会直接弹出计算器。



# Why



- 序列化的流是遵循标准协议格式的
- `ObjectInputStream` 默认不提供校验流的 API
- 可以反序列化任意当前 `classloader` 能加载的类
- 即使最后类强转失败，对象依然会被创建
- RMI 默认监听到所有网卡上





# 缓解措施



- 对序列化的流数据进行加密
- 在传输过程中使用 TLS 加密传输
- 对序列化数据进行完整性校验
- 对 RMI 监听的 IP 进行限制



# 信息泄露



```
public class Staff implements Serializable, ObjectInputValidation{  
    private static final long serialVersionUID = -2834558021263531425L;  
    private String name;  
    private transient int salary;
```

```
<terminated> TestStaff.testSerialize [JUnit] C:\Pr  
[name=Boss, salary=0]
```

# 对象校验

## 1) 针对序列化对象的属性

可以通过实现 `validateObject` 方法来进行对象属性值的校验。

步骤如下：实现 `ObjectInputValidation` 接口并重写 `validateObject` 方法；实现 `readObject` 方法，并注册 `Validation`。

```
public class Staff implements Serializable, ObjectInputValidation{
    private static final long serialVersionUID = -2834558021263531421L;
    private String name;
    private transient int salary;
    @Override
    public void validateObject() throws InvalidObjectException {
        if (salary < 0) {
            throw new InvalidObjectException("invalid salary");
        }
    }
    private void readObject(java.io.ObjectInputStream in) throws Exception{
        in.registerValidation(this, 0);
        in.defaultReadObject();
    }
}
```

```
@Override
public void validateObject() throws InvalidObjectException {
    if (salary < 0) {
        throw new InvalidObjectException("invalid salary");
    }
}
```

☰ Failure Trace

! java.io.InvalidObjectException: invalid salary

# 对象校验

## 2) 针对整个对象伪造的

上面的方法虽然能够对字段进行验证，但其验证时机是在读取流之后，所以只能对正常的序列化对象进行验证，对于畸形或者恶意序列化对象来说无能为力。


在序列化的流中，对象的描述是先于数据的，这就给了我们读取流完成之前进行验证的机会。

方法是，通过重写 `ObjectInputStream` 的 `resolveClass()` 方法来实现。这样，我们需要自定义一个对象流读取类继承自 `ObjectInputStream`，重写 `resolveClass()` 方法，然后反序列化的时候使用自己的类。

```
public class SecObjectInputStream extends ObjectInputStream{

    public SecObjectInputStream(InputStream in) throws IOException {
        super(in);
    }

    @Override
    protected Class<?> resolveClass(ObjectStreamClass streamClass)
        throws IOException, ClassNotFoundException {
        if (!streamClass.getName().equals(Staff.class.getName())) {
            throw new InvalidClassException("invalid object");
        }
        return super.resolveClass(streamClass);
    }
}
```



# 反序列化流程

## ObjectInputStream

- 4. Resolve classes of stream  
**resolveClass()**
- 5. Deserialize objects

## Serializable Class

- 6. Restore object member fields
  - **readObject(ObjectInputStream)**
  - **readObjectNoData()**
- 7. Eventually replace restored object
  - **readResolve()**
- 8. Optionally validate object
  - **validateObject()**

## Application Code

- 1. Get bytes
- 2. Initialize ObjectInputStream
- 3. Read object from stream
  - **ois.readObject()**

## Garbage Collector

- 11. Call **finalize()** on GC

# 对象校验

```
@Test
public void testSecSerialize() throws FileNotFoundException, IOException, ClassNotFoundException{

    //从 calc.ser 文件中反序列化对象，并打印对象
    ObjectInputStream objectInputStream = new SecObjectInputStream(new FileInputStream("calc.ser"));
    Staff staff2 = (Staff) objectInputStream.readObject();
    System.out.println(staff2);
    objectInputStream.close();
}
```

s Properties Servers Data Source Explorer Snippets Problems Console Progress Search Call Hierarchy JUnit Debug

fter 0.015 seconds

1 Errors: 1 Failures: 0

SecSerialize [Runner: JUnit 4] (0.000 s)

Failure Trace

java.io.InvalidClassException: invalid object

有人利用这个原理写了一个反序列化命令执行的防护工具，可以参考 <https://github.com/ikkisoft/SerialKiller>。

# RMI 监听内网 IP

```
public class Run {  
  
    public static void main(String[] args) {  
        try {  
            IMyRmi myRemote = new MyRmiImpl();  
            LocateRegistry.createRegistry(6600);  
            Naming.rebind("rmi://127.0.0.1:6600/hello", myRemote);  
            System.out.println("ready.....");  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
IMyRmi myRemote = new MyRmiImpl();  
LocateRegistry.createRegistry(6600);  
Naming.rebind("rmi://127.0.0.1:6600/hello", myRemote);
```

```
C:\>netstat -an | findstr 6600  
TCP        0.0.0.0:6600          0.0.0.0:0             LISTENING  
TCP        [::]:6600            [::]:0                 LISTENING
```

# RMI 监听内网 IP

```
public class RunOnLocal {  
  
    public static void main(String[] args) {  
        try {  
            IMyRmi myRemote = new MyRmiImpl();  
            RMIClientSocketFactory csf = new RMIClientSocketFactory() {  
  
                @Override  
                public Socket createSocket(String host, int port) throws IOException {  
                    return new Socket("127.0.0.1", 6600);  
                }  
            };  
            RMIServerSocketFactory ssf = new RMIServerSocketFactory() {  
  
                @Override  
                public ServerSocket createServerSocket(int port) throws IOException {  
                    return new ServerSocket(6600, 0, InetAddress.getByName("127.0.0.1"));  
                }  
            };  
  
            // LocateRegistry.createRegistry(6600);  
            LocateRegistry.createRegistry(6600, csf, ssf);  
            Naming.rebind("rmi://127.0.0.1:6600/hello", myRemote);  
            System.out.println("ready.....");  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
LocateRegistry.createRegistry(6600, csf, ssf);  
Naming.rebind("rmi://127.0.0.1:6600/hello", myRemote);  
System.out.println("ready.....");
```

```
C:\>netstat -an | findstr 6600  
TCP    127.0.0.1:6600          0.0.0.0:0              LISTENING  
TCP    127.0.0.1:6600          127.0.0.1:57137        ESTABLISHED  
TCP    127.0.0.1:57137        127.0.0.1:6600         ESTABLISHED
```





# 个性化



- 服务器监控
- 框架提供标准化功能
- 扫描器增加规则
- 网关增加规则
- 配置 Java 安全策略



# 参考文献



- 《Java序列化示例教程》 <http://www.importnew.com/14465.html>
- 《Serialization in Java – Java Serialization》 <http://www.journaldev.com/2452/serialization-in-java>
- 《Surviving the Java Serialization Apocalypse》 <https://www.slideshare.net/cschneider4711/surviving-the-java-deserialization-apocalypse-owasp-appseceu-2016>
- 《Java Object Serialization》 <http://docs.oracle.com/javase/8/docs/technotes/guides/serialization/index.html>
- 《对象序列化流协议》 <http://docs.oracle.com/javase/6/docs/platform/serialization/spec/protocol.html>
- 《Java深度历险（十）——Java对象序列化与RMI》 <http://www.infoq.com/cn/articles/cf-java-object-serialization-rmi/>
- 《怎么做才能让Java 序列化机制 更安全》 <http://www.myexception.cn/software-architecture-design/1463050.html>
- 《Look-ahead Java deserialization》 <https://www.ibm.com/developerworks/library/se-lookahead/>
- 《SerialKiller》 <https://github.com/ikkisoft/SerialKiller>

好了，伤口基本愈合了。

## VSRC 感谢有你！

小川 谢谢你！

谢谢你在新年伊始的第一天第一时间，发现并及时告知了我们这1枚唯品会集团的核心业务的高质量漏洞！你的这次发现，不仅帮助我们及时抑制住了企业进一步的危害和损失，更是从一定程度上加强并完善了我们自身安全工作的建设。

根据VSRC3.0新规评定：

特奖励额外**税后50000元人民币**的现金奖励！

你要不要来撒盐

# Venus

## 基础框架

配置管理

监控

安全

单元测试

模型映射

...

## 组件/资源客户端框架

数据库

缓存

消息

Zookeeper

...

## 服务框架

OSP

REST

...

## 工具

Codegen

Gradle

...

# 感谢您的倾听！

唯品会  
一家专门做特卖的网站



唯品会安全应急响应中心  
VIP Security Response Center



微信号：VIP\_SRC  
官方网站：<http://sec.vip.com>  
微信公众号：唯品会安全应急响应中心  
漏洞接收邮箱：[sec@vipshop.com](mailto:sec@vipshop.com)

唯品会安全应急响应中心  
我们致力于保护用户信息安全  
我们积极营造更加安全的  
线上电商购物平台

