

此文档为《Java代码审计零基础入门到项目实战》配套教材，由【闪石星曜CyberSecurity】出品。

请勿对外泄露，一经发现严肃处理！

课程学习中有任何疑问，可添加好友 Power_7089 寻求帮助，为你答疑解惑。

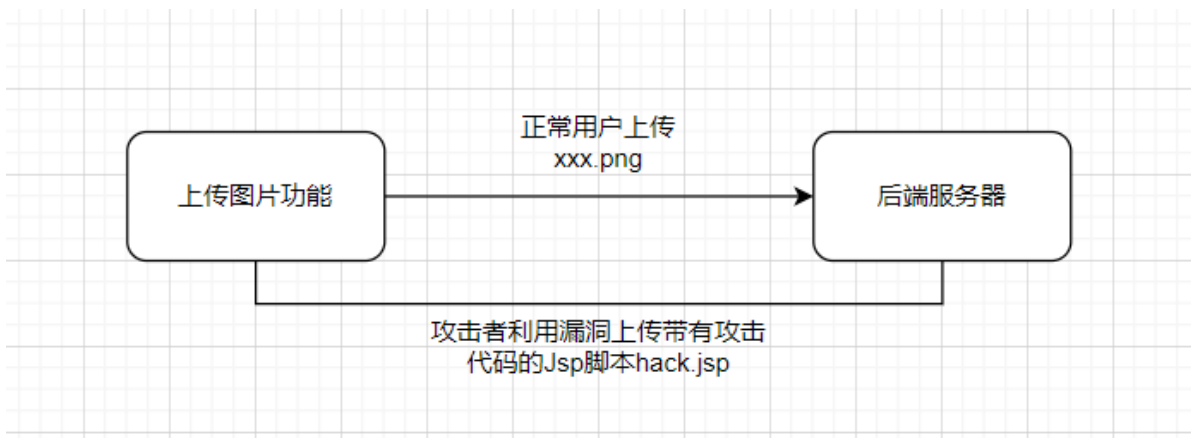
在 Java代码审计之基础知识篇 - 1.3 Java文件操作之文件上传 中我们学到了文件上传的方式，下面讲讲任意文件上传漏洞代码审计。

建议没有看过的朋友回顾下基础，与下面课程内容紧密关联。

一、任意文件上传漏洞

1、什么是任意文件上传漏洞

任意文件上传漏洞常发生在文件上传功能中，由于后端代码中没有严格限制用户上传的文件，导致攻击者可以上传带有恶意攻击代码的JSP 脚本到目标服务器，进而执行脚本，以达到控制操纵目标服务器等目的。



2、任意文件上传漏洞危害

如果目标服务器存在任意文件上传漏洞，服务将会面临巨大风险，包括但不限于：服务器的网页被篡改，网站被上传木马，服务器被远程控制，被安装后门，执行挖矿程序等。

二、任意文件上传漏洞代码审计

在对文件上传功能进行代码审计时我们主要分析整个上传流程对所上传文件做了什么样的操作。进而分析是否能够造成任意文件上传漏洞。

我们比较关注的几点：①、SpringBoot对JSP的限制。②、文件后缀名是否存在白名单。③、文件类型是否存在白名单。④、所保存的路径是否能够解析JSP。⑤、文件头检测。

我们使用 Java代码审计之基础知识篇 - 1.3 Java文件操作之文件上传 课程中 FileUploadDemo 项目代码继续作为我们的演示代码。只不过我们将视角从开发上传功能转换成对上传功能的代码审计，分析是否存在任意文件上传漏洞。

1、SpringBoot 对 JSP的限制

现在大多数项目都是基于SpringBoot架构进行的开发，官方不建议 SpringBoot 使用 JSP，并且做了一些限制。

具体官方文档: <https://docs.spring.io/spring->

[boot/docs/2.1.1.RELEASE/reference/htmlsingle/#boot-features-jsp-limitations](https://docs.spring.io/spring-boot/docs/2.1.1.RELEASE/reference/htmlsingle/#boot-features-jsp-limitations)

懒得打开可以看下面原文:

When running a Spring Boot application that uses an embedded servlet container (and is packaged as an executable archive), there are some limitations in the JSP support.

With Jetty and Tomcat, it should work if you use war packaging. An executable war will work when launched with `java -jar`, and will also be deployable to any standard container. JSPs are not supported when using an executable jar.

Undertow does not support JSPs.

Creating a custom `error.jsp` page does not override the default view for error handling. Custom error pages should be used instead.

如果想要使用JSP也可以, 需要引入相关依赖, 自建WEB-INF,web.xml等操作, 这样一通操作下来失去了SpringBoot的一些特性。当然这也仅是一部分原因。这里我们不过多探讨。可参考

https://blog.csdn.net/weixin_43122090/article/details/103866174

我们在对SpringBoot项目审计时如果想要查看是否对JSP完全解析, 可以从熟悉的 `pom.xml` 文件下手, 查看是否引入了相关依赖。如下所示:

```
<!--用于编译jsp-->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```

探索SpringBoot是否解析JSP目的是如果该项目存在任意文件上传漏洞, 那么我们可以通过上传jspShell代码最大化我们的攻击。

Spring Boot Fat Jar 写文件漏洞到稳定 RCE 的探索

<https://github.com/LandGrey/spring-boot-upload-file-lead-to-rce-tricks>
<https://landgrey.me/blog/22/>
<https://threedr3am.github.io/2021/04/14/JDK8%E4%BB%BB%E6%84%8F%E6%96%87%E4%BB%B6%E5%86%99%E5%9C%BA%E6%99%AF%E4%B8%8B%E7%9A%84SpringBoot%20RCE/>

2、校验文件类型

2.1、文件后缀名校验

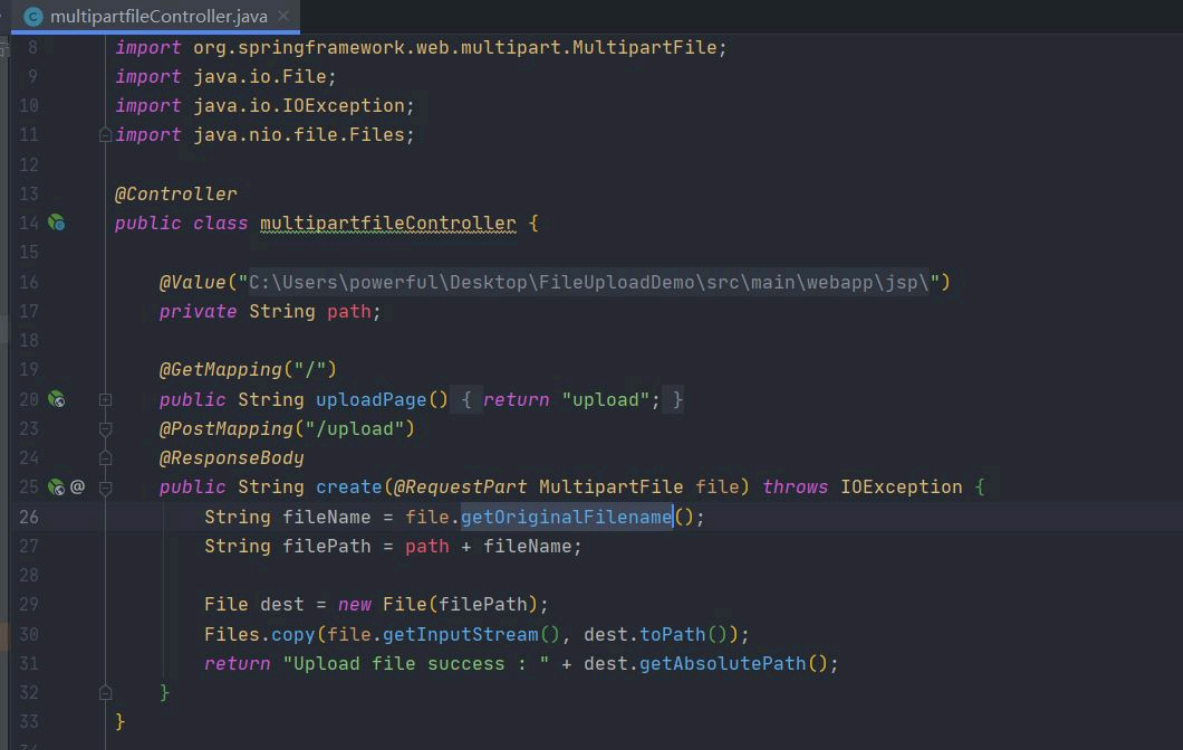
这里我们主要关注后端是否对上传的文件后缀名进行了检查判断。如果在后端对后缀名没有限制。那么就极有可能存在任意文件上传漏洞。

很多项目是通过前端限制文件上传类型。但我们都知道, 不论前端怎么限制, 我们都可以通过抓包的方式修改后缀为敏感文件, 以达到上传Webshell的目的。

下面举个例子。

打开FileUploadDemo项目，打开

src/main/java/com/example/demo/multipartfileController.java 文件，第26行就是通过 getOriginalFilename() 方法获取上传时的文件名。第27行直接和path路径拼接。并没有判断后缀名。如下图所示：



```
8 import org.springframework.web.multipart.MultipartFile;
9 import java.io.File;
10 import java.io.IOException;
11 import java.nio.file.Files;
12
13 @Controller
14 public class multipartfileController {
15
16     @Value("C:\\Users\\powerful\\Desktop\\FileUploadDemo\\src\\main\\webapp\\jsp\\")
17     private String path;
18
19     @GetMapping("/")
20     public String uploadPage() { return "upload"; }
21
22     @PostMapping("/upload")
23     @ResponseBody
24     public String create(@RequestParam MultipartFile file) throws IOException {
25         String fileName = file.getOriginalFilename();
26         String filePath = path + fileName;
27
28         File dest = new File(filePath);
29         Files.copy(file.getInputStream(), dest.toPath());
30         return "Upload file success : " + dest.getAbsolutePath();
31     }
32 }
33
34 }
```

2.2、文件后缀名校验黑白名单

如果后端使用了黑白名单限制后缀名已经初步起到了一定的防护作用。但具体还需要根据实际情况分析。如果是黑名单是否存在遗漏的情况。我会使用模糊字典Fuzz后缀名，看看是否能绕过。推荐一位前辈文章：<https://gv7.me/articles/2018/make-upload-vul-fuzz-dic/>

使用白名单判断后缀是比较安全稳妥的方式。比如下面代码：

```
// 获取文件后缀名
String suffix = fileName.substring(fileName.lastIndexOf("."));
//黑白名单判断
String[] suffixList = {".jpg", ".png", ".jpeg", ".gif"};
```

2.3、MIME type检测

校验文件类型还有一种方式检测MIME Type。也就是我们在请求中常见的 Content-Type 字段。

如果项目中使用MIME type黑白名单检测文件类型，可以分析黑白名单中是否有遗漏的敏感文件类型。

常见的MIME类型：

https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

推荐两篇学习文章：

https://blog.csdn.net/qq_42764468/article/details/121522645
<https://www.cnblogs.com/bojuetech/p/5907910.html>

3、文件名操作

在这里我们关注上传的文件名是否有所改动。

常见的情况是后端直接接受保存我们上传的文件名。

但也常后端自定义不可预测的文件名，比如使用UUID。比如以下代码：

```
String originalFileName = file.getOriginalFilename();
String extension = originalFileName.substring(originalFileName.lastIndexOf('.'));
String fileName = UUID.randomUUID() + extension;
```

将文件名随机命名可以增加一些攻击利用难度。但并没有直接修复任意文件上传漏洞。

很多时候由于代码编写不规范会将上传后的路径，文件名也会显示在前端。

4、保存路径

在这里我们关注是否保存在本地，保存文件路径是否是在非解析路径，保存路径是否可控。

4.1、是否保存在本地

现在很多项目可以说是都在向云服务器迁移，并且对数据，文件做了隔离。不同的场景应用不同的存储服务器。

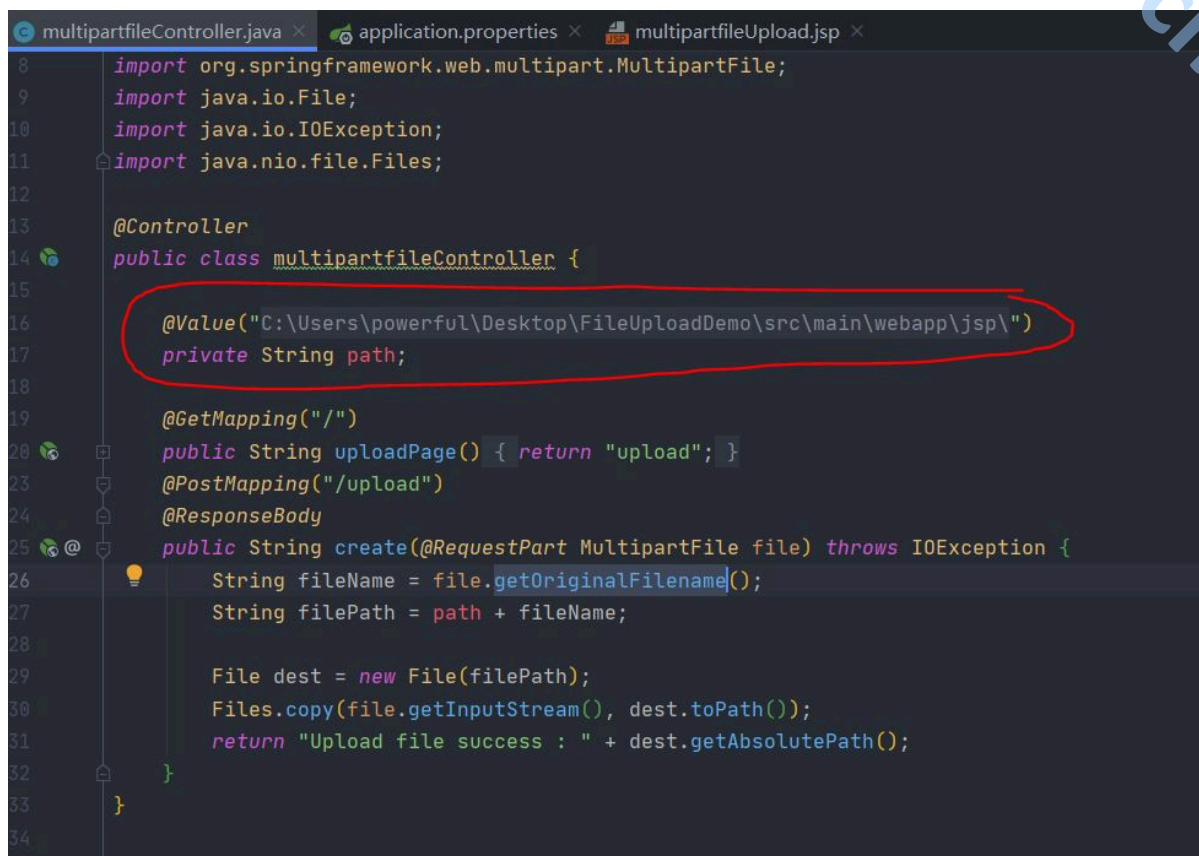
后端在对上传文件保存时无非要么是保存在服务器本地，要么保存在相关云存储服务器。比如：七牛云OSS，阿里OSS等等。如果保存在了OSS上，尽管上传了敏感Webshell脚本，也是无法执行的。

有一些其他骚操作，这里暂且不谈。推荐两篇文章：

<http://pirogue.org/2017/09/29/aliyunoss/>。

下面我们讲讲将文件存储在本地的情况。

判断是否保存在本地很简单，查看配置文件，或者上传文件代码处是否标明等等。比如下图所示：



```
multipartfileController.java × application.properties × multipartfileUpload.jsp ×
8 import org.springframework.web.multipart.MultipartFile;
9 import java.io.File;
10 import java.io.IOException;
11 import java.nio.file.Files;
12
13 @Controller
14 public class multipartfileController {
15
16     @Value("C:\\Users\\powerful\\Desktop\\FileUploadDemo\\src\\main\\webapp\\jsp\\")
17     private String path;
18
19     @GetMapping("/")
20     public String uploadPage() { return "upload"; }
21     @PostMapping("/upload")
22     @ResponseBody
23     public String create(@RequestPart MultipartFile file) throws IOException {
24         String fileName = file.getOriginalFilename();
25         String filePath = path + fileName;
26
27         File dest = new File(filePath);
28         Files.copy(file.getInputStream(), dest.toPath());
29         return "Upload file success : " + dest.getAbsolutePath();
30     }
31 }
32
33 }
```

4.2、是否解析路径

在我们的代码案例中，是将保存的文件保存在了 webapp/jsp 目录下，该目录可以解析 JSP 文件，进而造成了任意文件上传漏洞。

在实际项目中，所保存文件的地址可能是一个不可执行不可解析权限非常低的目录，尽管我们将 WebShell 上传到了目标服务器，那么也因无法解析执行而无功而返。

4.3、路径是否可控

在获取文件名后，大多会进行路径拼接操作。

在这里我们可以检查拼接路径是有相关防护，如果没有限制 `../` 那么极有可能存在目录穿越漏洞。

如果保存图片的地址是非解析目录，我们可以配合目录穿越漏洞操作 WebShell 存储到其他地方，尝试执行。

FileUploadDemo 项目中路径使用了直接拼接的方式，并且没有任何防护，代码如下：

```
String fileName = file.getOriginalFilename();
String filePath = path + fileName;
```

大家可以启动运行该项目自行调试，将上传的文件名改为 `../../../../../../../../test.txt` 后观察结果。

5、文件上传功能关键字

大家在对完整项目进行代码审计时，尽量整理收集尽可能多的信息，确定系统功能点，进而可以针对性的进行代码审计。

在面对一个完整的项目中有我常用以下方式定位上传功能，比如：[查看需求文档](#)，[查看Controller层](#)，部署后通过前端定位功能点，[全局搜索关键字](#) 等等。

下面给出一些文件上传关键字，帮助你快速定位是否存在文件上传功能。

```
File
FileUpload
FileUploadBase
FileItemIteratorImpl
FileItemStreamImpl
FileUtils
UploadHandleServlet
FileLoadServlet
FileOutputStream
DiskFileItemFactory
MultipartRequestEntity
MultipartFile
com.oreilly.servlet.MultipartRequest
.....
```

三、JSP木马

JSP木马也叫作Java WebShell，代码使用JSP（Java Server Pages）编写，只不过是在代码中加入了一些操作文件，访问系统等功能，也就成为了我们口中的JSP木马。

JSP木马又分为一句话木马，小马，大马等。

常用于任意文件上传漏洞处。如果发现目标存在任意文件上传漏洞，我们可以上传JSP木马到目标服务器，以达到控制目标服务器的目的。

JSP木马：

```
https://github.com/theralfbrown/webShell-2/tree/master/jsp
```

webshell管理工具：

```
https://github.com/BeichenDream/Godzilla
https://github.com/rebeyond/Behinder
```

四、任意文件上传漏洞修复

原文：https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html

部分翻译如下：

列出允许的扩展。只允许业务功能的安全和关键扩展

确保在验证扩展名之前应用输入验证。

验证文件类型，不要相信**Content-Type**头，因为它可以被欺骗。

将文件名改为由应用程序生成的文件名

设置一个文件名的长度限制。如果可能的话，限制允许的字符

设置一个文件大小限制

只允许授权用户上传文件

将文件存储在不同的服务器上。如果不可能，就把它们存放在**webroot**之外。

在公众访问文件的情况下，使用一个处理程序，在应用程序中被映射到文件名（`someid -> file.ext`）。

通过杀毒软件或沙盒（如果有的话）运行文件，以验证它不包含恶意数据。

确保任何使用的库都是安全配置的，并保持最新。

保护文件上传免受**CSRF**攻击