

简介

SpringMVC的一个目录穿越漏洞，或者说目录遍历漏洞。SpringMVC是Java开发界最常用的框架之一，当静态资源存放在Windows系统上时，攻击可以通过构造特殊URL导致目录遍历漏洞

影响版本

- Spring Framework 5.0 to 5.0.4.
- Spring Framework 4.3 to 4.3.14
- 已不支持的旧版本仍然受影响

漏洞利用条件

- Server运行于Windows系统上
- 要使用file协议打开资源文件目录

环境搭建&漏洞复现

```
git clone https://github.com/spring-projects/spring-mvc-showcase.git
```

修改spring的版本

```
<properties>
    <java-version>1.8</java-version>
    <org.springframework-
version>5.0.0.RELEASE</org.springframework-version>
    <org.aspectj-version>1.8.1</org.aspectj-version>
</properties>
```

修改配置文件: org\springframework\samples\mvc\config\WebMvcConfig.java
这里的作用主要是配置资源映射，将url的/resources/映射到file协议的resources下

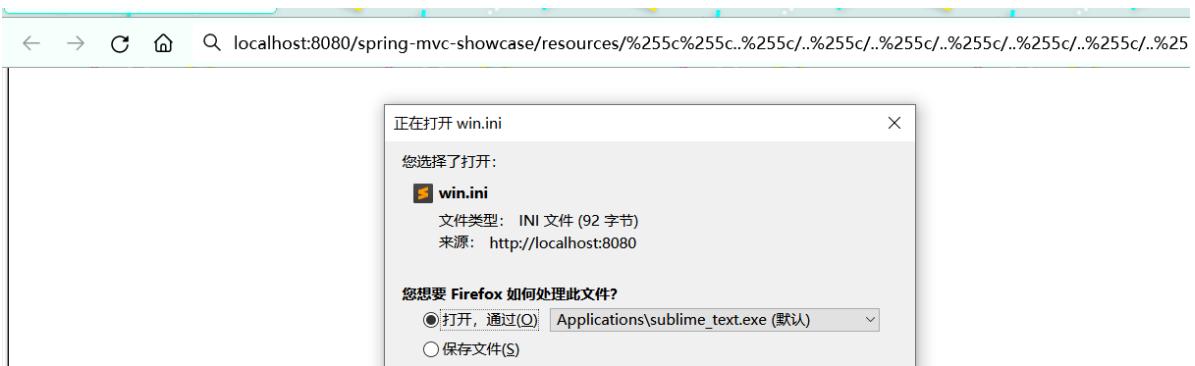
```
@Override
public void addResourceHandlers(ResourceHandlerRegistry
registry) {

    registry.addResourceHandler("/resources/**").addResourceLocations("file:./src/main/resources/", "/resources/");
}
```

使用mvn命令启动: mvn jetty:run

访问url:

```
http://localhost:8080/spring-mvc-
showcase/resources/%255c%255c..%255c/%255c/..%255c/..%255c/..%255c/..%255c/..%255c/..%255c/windows/win.ini
```

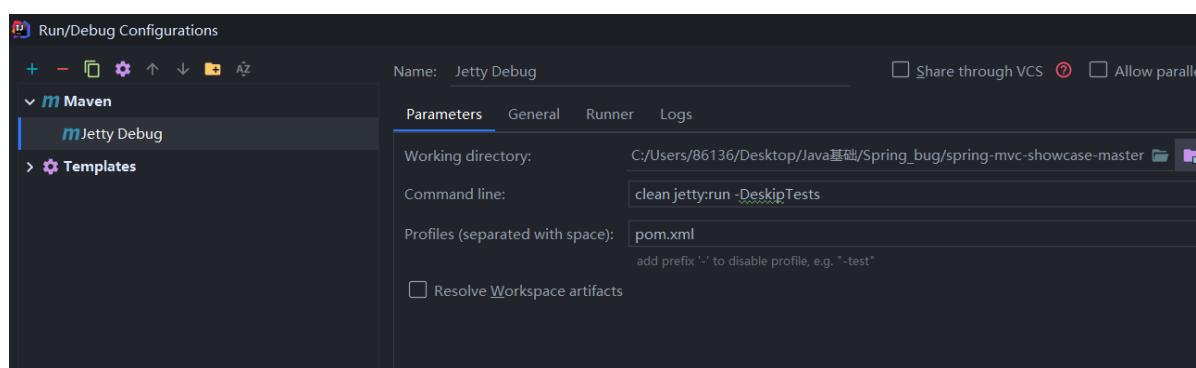


漏洞分析

添加依赖(根据自己的报错来)

```
<!--  
https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api  
-->  
<dependency>  
    <groupId>javax.xml.bind</groupId>  
    <artifactId>jaxb-api</artifactId>  
    <version>2.3.1</version>  
</dependency>
```

设置Jetty Debug



当外部要访问静态资源时，会调用

`org.springframework.web.servlet.resource.ResourceHttpRequestHandler:handleRequest` 来处理

经过getAttribute方法后会进行一次url解码

```
ResourceHttpRequestHandler > getResource()
filed.class file, bytecode version: 52.0 (Java 8)                                         Download Sources Choose Sources...
protected Resource getResource(HttpServletRequest request) throws IOException {   request: "Request(GET //127.0.0.1:8080/spring-mvc-showcase/resources/%255
String path = (String)request.getAttribute(HandlerMapping.PATH_WITHIN_HANDLER_MAPPING_ATTRIBUTE);   path: "%5c%5c..%5c/%..%5c/%..%5c/%..%5c/%..%5c...
if (path == null) {
    throw new IllegalStateException("Required request attribute '" + HandlerMapping.PATH_WITHIN_HANDLER_MAPPING_ATTRIBUTE + "' is not set");
} else {
    path = this.processPath(path);
    if (StringUtils.hasText(path) && !this.isInvalidPath(path)) {
        if (path.contains("%")) {
            try {
                if (this.isInvalidPath(URLDecoder.decode(path, enc: "UTF-8"))) {
                    if (logger.isTraceEnabled()) {
                        logger.trace(o: "Ignoring invalid resource path with escape sequences [" + path + "].");
                }
            }
        }
    }
}
return new DefaultResourceLoader(this.getClass().getClassLoader(), this.getServletContext()).getResource(path);
}
```

此时path

为 %5c%5c..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/windows/win.ini

后续有processPath和isValidPath方法对path进行处理和校验，其中

`processPath`方法作用是处理斜杠的问题，比如多个斜杠或者丢失斜杠的问题，和这里无关系。程序经过它是path也没有变化。`isValidPath`是需要重点关注的。这里依次将path和解码后的path传入`isValidPath`方法

跟进去看下：

首先禁止了WEB-INF和META-INF等关键路径，然后处理是url（带有:/）的情况，最后是处理带有..的情况，而..正是目录遍历漏洞需要用到的

ResourceHttpRequestHandler → isInvalidPath()

Compiled .class file, bytecode version: 52.0 (Java 8)

```
protected boolean isInvalidPath(String path) {    path: "%5c%5c..%5c/%5c/%5c/%5c/%5c/%5c/%5c/windows/win.ini"
    if (logger.isTraceEnabled()) {
        logger.trace( o: "Applying \"invalid path\" checks to path: " + path);
    }

    if (!path.contains("WEB-INF") && !path.contains("META-INF")) {
        if (path.contains(":/")) {...}

        if (path.contains("..")) {
            path = StringUtils.cleanPath(path);    path: "%5c%5c..%5c/%5c/%5c/%5c/%5c/windows/win.ini"
            if (path.contains("../")) {
                if (logger.isTraceEnabled()) {
                    logger.trace( o: "Path contains \"../\" after call to StringUtils#cleanPath.");
                }
            }
        }
    }
}
```

跟进cleanPath,首先将url解码后的path分割成数组,然后经过两个for循环。这个cleanPath的主要作用就是将/foo/bar/../**这样的路径转换为 /foo/** (可能就是为了满足windows特性吧,想起了一个apache的cve)

StringUtils → cleanPath()

Compiled .class file, bytecode version: 52.0 (Java 8)

```
String[] pathArray = delimitedListToStringArray(pathToUse, delimiter: "/");    pathArray: "%5c%5c..%5c", "..%5c", "..%5c", "..%5c", "+ 6 more"    pathElements: size = 11
List<String> pathElements = new LinkedList();    pathElements: size = 11
int tops = 0;    tops: 0

int i;
for(i = pathArray.length - 1; i >= 0; --i) {
    String element = pathArray[i];    pathArray: "%5c%5c..%5c", "..%5c", "..%5c", "..%5c", "+ 6 more"
    if (!"..".equals(element)) {
        if ("..".equals(element)) {
            ++tops;
        } else if (tops > 0) {
            --tops;
        } else {
            pathElements.add( index: 0, element);
        }
    }
}

for(i = 0; i < tops; ++i) {    tops: 0
    pathElements.add( index: 0, element: "..");
}
```

pathArray

- pathArray = {String[11]@8476}
 - 0 = "%5c%5c..%5c"
 - 1 = "..%5c"
 - 2 = "..%5c"
 - 3 = "..%5c"
 - 4 = "..%5c"
 - 5 = "..%5c"
 - 6 = "..%5c"
 - 7 = "..%5c"
 - 8 = "..%5c"
 - 9 = "windows"
 - 10 = "win.ini"

由于返回的path中没有包含".."，返回false

```
if (path.contains("../")) {
    path = StringUtils.cleanPath(path);
    if (path.contains("../..")) {
        if (logger.isTraceEnabled()) {
            logger.trace("Path contains
\"..\" after call to StringUtils#cleanPath.");
        }
    }
    return true;
}
return false;
```

这里其实并没有发生什么，关键点在后面那次`isInvalidPath`

```
if (this.isInvalidPath(URLDecoder.decode(path, enc: "UTF-8"))) {
```

进入cleanPath,这里将 \ 替换为了 /,然后进行一些替换

然后这里有点问题：

`cleanPath` 的问题在于 `String[] pathArray = delimitedListToStringArray(pathToUse, "/");` 这个是允许空元素存在的，也就是说 `cleanPath` 会把 `//` 当成一个目录，而操作系统是不会把 `//` 当成一个目录的。

Input	cleanPath	Filesystem
/foo/..	/	/
/foo/.../..	/..	/..
/foo//.../	/foo/	/
/foo///.../..	/foo/	/..
/foo///.../..../	/foo/	/.../

最后返回的path中仍然没有`..`,返回false

```
ResourceHttpRequestHandler > isValidPath()
compiled .class file, bytecode version: 52.0 (Java 8)
    }
}

    if (path.contains("../")) {
        path = StringUtils.cleanPath(path);
        if (path.contains("../")) {  path: "//windows/win.ini"
            if (logger.isTraceEnabled()) {
                logger.trace( or "Path contains '../' after call to StringUtils#cleanPath.");
            }
        }
        return true;
    }
}

return false;
```

当我们通过两次检测之后,这里有一个getResourceTransformers()方法,getLocation方法包含配置文件中配置的路径,跟进resoslveResource

The screenshot shows the IntelliJ IDEA IDE with the following details:

- Top Bar:** Shows "ResourceHttpRequestHandler > getResource()".
- Code Editor:** Displays Java code for resolving resources. A red bar highlights the line: "Resource resource = resolveChain.resolveResource(request, path, this.getLocations());". A green bar highlights the line: "if (resource != null && !this.getResourceTransformers().isEmpty()) {". A blue bar highlights the line: "if (logger.isTraceEnabled()) {".
- Toolbars:** Standard Java development toolbar.
- Variables:** Shows variables for the current stack frame:
 - request = {Request@8361} "Request(GET //127.0.0.1:8080/spring-mvc-showcase/resource... View
 - path = "%5c%5c..%5c/%5c/%5c/%5c/%5c/%5c/windows/win.ini"
 - resolveChain = {DefaultResourceResolverChain@9907}
 - resource = {UrlResource@9961} "URL [file:/src/main/resources/%5c%5c..%5c/%5c/%5c..%5c..%5c]"
- Watches:** Shows the value of "this.getLocations()", which is an ArrayList with size 2. The elements are:
 - { } 0 = {UrlResource@9913} "URL [file:/src/main/resources/]"
 - { } 1 = {ServletContextResource@9914} "ServletContext resource .."

进入getResource

```
PathResourceResolver > getResource()
Compiled .class file, bytecode version: 52.0 (Java 8)
    private Resource getResource(String resourcePath, List<? extends Resource> locations) {   resourcePath: "%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini"
        Iterator var3 = locations.iterator();   locations: size = 2

        while(var3.hasNext()) {
            Resource location = (Resource)var3.next();   location: "URL [file:./src/main/resources/]

            try {
                if (this.logger.isTraceEnabled()) {
                    this.logger.trace( o: "Checking location: " + location);
                }
            }

            Resource resource = this.getResource(resourcePath, location);   resourcePath: "%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini" location: "URL [file:./src/main/resources/]
            if (resource != null) {
                return resource;
            }
        }
    }

```

进入createRelative()

```
PathResourceResolver > getResource()
Compiled .class file, bytecode version: 52.0 (Java 8)
    @Nullable
    protected Resource getResource(String resourcePath, Resource location) throws IOException {   resourcePath: "%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini"
        Resource resource = location.createRelative(resourcePath);   location: "URL [file:./src/main/resources/]" resourcePath: "%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini"
        if (resource.exists() && resource.isReadable()) {
            if (this.checkResource(resource, location)) {
                return resource;
            }
        }
    }

```

```
UrlResource > createRelative()
Compiled .class file, bytecode version: 52.0 (Java 8)
    public Resource createRelative(String relativePath) throws MalformedURLException {   relativePath: "%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini"
        if (relativePath.startsWith("/")) {
            relativePath = relativePath.substring(1);
        }

        return new UrlResource(new URL(this.url, relativePath));   url: "file:./src/main/resources/" relativePath: "%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini"
    }

```

经过 `createRelative` 方法拼接后得到

`file:./src/main/resources/%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini`

然后调用 `resource.exists()` 方法判断文件是否存在

这里会调用 `isFileURL` 对 `url` 进行判断，是否以 `file://` 协议来读取文件，这也是为什么配置静态目录的时候要使用 `file://` 协议。

```
AbstractFileResolvingResource > exists()
Compiled .class file, bytecode version: 52.0 (Java 8)
    public boolean exists() {
        try {
            URL url = this.getURL();   url: "file:src/main/resources/%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini"
            if (ResourceUtils.isFileURL(url)) {   url: "file:src/main/resources/%5c%5c..%5c/.%5c/.%5c/.%5c/%5c/%5c/windows/win.ini"
                return this.getFile().exists();
            } else {
                URLConnection con = url.openConnection();
                this.customizeConnection(con);
                HttpURLConnection httpCon = con instanceof HttpURLConnection ? (HttpURLConnection)con : null;
                if (httpCon != null) {

```

然后调用this.getFile()来获取这个文件对象,来到ResourceUtils#getFile

首先对是否为file://协议又判断了一遍,之后进行了一步最重要的操作 new

```
File(toURI(resourceUrl).getSchemeSpecificPart())
```

The screenshot shows a Java code editor with the following code:

```
ResourceUtils > getFile()  
compiled .class file, bytecode version: 52.0 (Java 8)  
Download Sources Choose  
    }  
    @  
    public static File getFile(URL resourceUrl, String description) throws FileNotFoundException {  
        Assert.notNull(resourceUrl, message: "Resource URL must not be null");  
        if (!"file".equals(resourceUrl.getProtocol())) {  
            throw new FileNotFoundException(description + " cannot be resolved to absolute file path because it does not reside in the file system or a file resource");  
        } else {  
            try {  
                return new File(toURI(resourceUrl).getSchemeSpecificPart());  
            } catch (URISyntaxException var3) {  
                return new File(resourceUrl.getFile());  
            }  
        }  
    }
```

将 `resourceUrl` 转换为 URL 对象，最后调用 `URI` 类的 `getSchemeSpecificPart()` 获取到文件路径，而在 `getSchemeSpecificPart()` 里面是有一次 `decode` 操作的，也就是在这里把 `%5c` 解码成了 \

这里进行返回

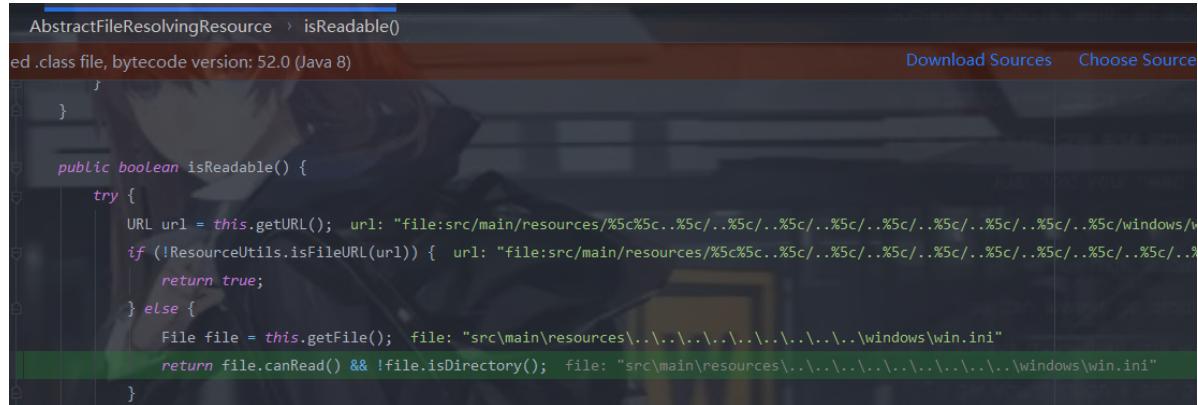
The screenshot shows a Java code editor with the following code:

```
ResourceUtils > getFile()
Compiled .class file, bytecode version: 52.0 (Java 8)
Download Sources Choose Sources

}

public static File getFile(URL resourceUrl, String description) throws FileNotFoundException {
    Assert.notNull(resourceUrl, message: "Resource URL must not be null");
    if (!"file".equals(resourceUrl.getProtocol())) {
        throw new FileNotFoundException(description + " cannot be resolved to absolute file path because it does not reside in the file system");
    } else {
        try {
            return new File(toURI(resourceUrl).getSchemeSpecificPart());
        }
    }
}
```

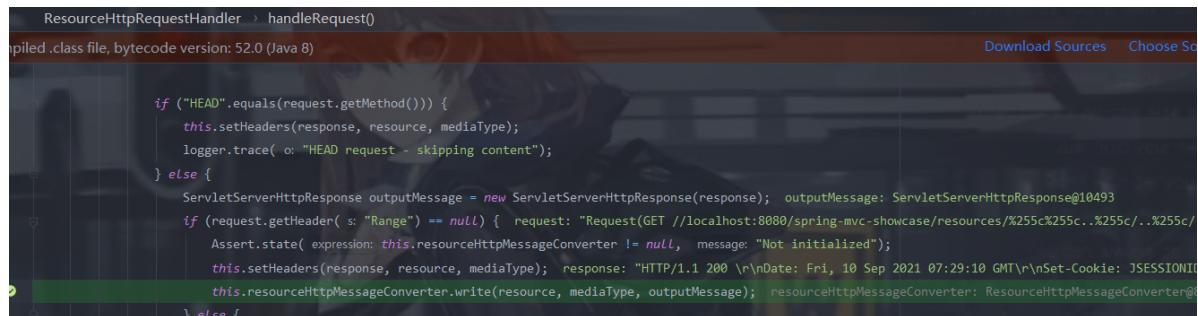
最后返回到 `exists()`，最终返回 `true`，即文件存在。之后调用 `isReadable()` 方法检测该文件是否可读的时候，同样会调用这个 `getFile`，最终返回 `true`，即文件可读。



```
AbstractFileResolvingResource > isReadable()
Compiled .class file, bytecode version: 52.0 (Java 8)
try {
    URL url = this.getURL(); url: "file:src/main/resources/%5c%5c..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/windows/win.ini"
    if (!ResourceUtils.isFileURL(url)) { url: "file:src/main/resources/%5c%5c..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/windows/win.ini"
        return true;
    } else {
        File file = this.getFile(); file: "src/main/resources/../../../../../../../../windows/win.ini"
        return file.canRead() && !file.isDirectory(); file: "src/main/resources/../../../../../../../../windows/win.ini"
    }
}
```

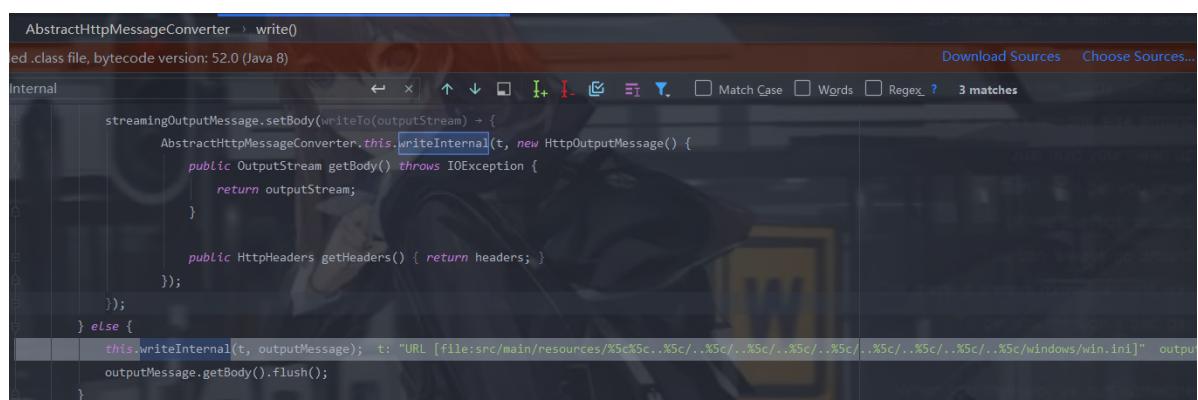
Resource的检测都通过后,来到ResourceHttpRequestHandler#handleRequest方法准备response的内容

跟进write()



```
ResourceHttpRequestHandler > handleRequest()
Compiled .class file, bytecode version: 52.0 (Java 8)
if ("HEAD".equals(request.getMethod())) {
    this.setHeaders(response, resource, mediaType);
    logger.trace( o: "HEAD request - skipping content");
} else {
    ServletServerHttpResponse outputMessage = new ServletServerHttpResponse(response); outputMessage: ServletServerHttpResponse@10493
    if (request.getHeader( s: "Range") == null) { request: "Request(GET //localhost:8080/spring-mvc-showcase/resources/%255c%255c..%255c/..%255c/..%255c/..%255c/..%255c/..%255c/..%255c/windows/win.ini"
        Assert.state( expression: this.resourceHttpMessageConverter != null, message: "Not initialized");
        this.setHeaders(response, resource, mediaType); response: "HTTP/1.1 200 \r\nDate: Fri, 10 Sep 2021 07:29:10 GMT\r\nSet-Cookie: JSESSIONID=...; Path=/; HttpOnly; Secure"
        this.resourceHttpMessageConverter.write(resource, mediaType, outputMessage); resourceHttpMessageConverter: ResourceHttpMessageConverter@10494
    } else {
}
```

然后依次进入



```
AbstractHttpMessageConverter > write()
Compiled .class file, bytecode version: 52.0 (Java 8)
streamingOutputMessage.setBody(writeTo(outputStream) &gt;
    AbstractHttpMessageConverter.this.writeInternal(t, new HttpOutputMessage() {
        public OutputStream getBody() throws IOException {
            return outputStream;
        }
    }
    public HttpHeaders getHeaders() { return headers; }
));
}
} else {
    this.writeInternal(t, outputMessage); t: "URL [file:src/main/resources/%5c%5c..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/windows/win.ini]" outputMessage: HttpOutputMessage@10495
    outputMessage.getBody().flush();
}
```

```
ResourceHttpMessageConverter > readInternal()
Compiled .class file, bytecode version: 52.0 (Java 8)

@ protected Resource readInternal(Class<? extends Resource> clazz, final HttpInputMessage inputMessage) throws IOException {
    if (this.supportsReadStreaming && InputStreamResource.class == clazz) { supportsReadStreaming: true
        return new InputStreamResource(inputMessage.getBody());
    }
    public String getFilename() { return inputMessage.getHeaders().getContentDisposition().getFilename();
    }
} else if (clazz.isAssignableFrom(ByteArrayResource.class)) {
    byte[] body = StreamUtils.copyToByteArray(inputMessage.getBody());
    return new ByteArrayResource(body);
}
@Nullable
```

```
ResourceHttpMessageConverter > writeInternal()
Compiled .class file, bytecode version: 52.0 (Java 8)
long contentLength = resource.contentLength();
return contentLength < 0L ? null : contentLength;
}

protected void writeInternal(Resource resource, HttpOutputMessage outputMessage) throws IOException, HttpMessageNotWritableException {
    resource: "URL [file:src/main/resources/%5c%5c..%5c/%5c/%5c/%5c/%5c/windows/win.ini"
    this.writeContent(resource, outputMessage);
}
```

```
ResourceHttpMessageConverter > writeContent()
Compiled .class file, bytecode version: 52.0 (Java 8)
this.writeContent(resource, outputMessage);
}

protected void writeContent(Resource resource, HttpOutputMessage outputMessage) throws IOException, HttpMessageNotWritableException {
    resource: "URL [file:src/main/resources/%5c%5c..%5c/%5c/%5c/%5c/%5c/%5c/windows/win.ini"
    try {
        InputStream in = resource.getInputStream();
    }
    try {

```

然后在url.openConnection阶段会进行解码,将%5c解码成/。然后返回文件的
InputStream对象,最终读取内容返回给用户。

```
UrlResource > getInputStream()
Compiled .class file, bytecode version: 52.0 (Java 8)
}

public InputStream getInputStream() throws IOException {
    URLConnection con = this.url.openConnection(); url: "file:src/main/resources/%5c%5c..%5c/%5c/%5c/%5c/%5c/windows/win.ini"
    ResourceUtils.useCachesIfNecessary(con);

    try {
        return con.getInputStream();
    } catch (IOException var3) {

```



```
Handler > openConnection()
Compiled .class file, bytecode version: 52.0 (Java 8)
}

public synchronized URLConnection openConnection(URL var1, Proxy var2) throws IOException {
    var1: "file:src/main/resources/%5c%5c..%5c/%5c/%5c/%5c/windows/win.ini"
    String var4 = var1.getFile(); var4 (slot_3): "src/main/resources\\..\\..\\..\\..\\..\\..\\windows\\win.ini"
    String var5 = var1.getHost(); var5 (slot_4): "src/main/resources/%5c%5c..%5c/%5c/%5c/%5c/windows/win.ini"
    String var3 = ParseUtil.decode(var4); var3 (slot_5): ""
    var3 = var3.replace( oldChar '/', newChar '\\');
    var3 = var3.replace( oldChar '|', newChar ':');
    if (var5 != null && !var5.equals("") && !var5.equalsIgnoreCase( anotherString: "localhost" ) && !var5.equals(".")) {...} else {
        return this.createFileURLConnection(var1, new File(var3));
    }
}
```

总结

感觉流程挺复杂的,这里大概总结一下,一共分三步

1. 在isValidPath方法判断路径是否合法(经过isValidPath后不存在../),但是isValidPath有点问题,可以被绕过
 2. 利用createRelative方法拼接,然后判断文件是否存在并可读(判断时会进行url解码)
 3. 构造response数据,在url.openConnection会进行url解码,然后成功读取我们的文件并返回

注意事项

1、在Spring Framework 大于5.0.1的版本，双编码payload在tomcat、jetty下触发不了，

但是单编码payload可以在jetty下触发，tomcat下失败(因为在默认情况下Tomcat遇到包含%2f()、%5c()的URL直接http 400)

总结：

Spring Framework <=5.0.1

tomcat、jetty:(双编码)

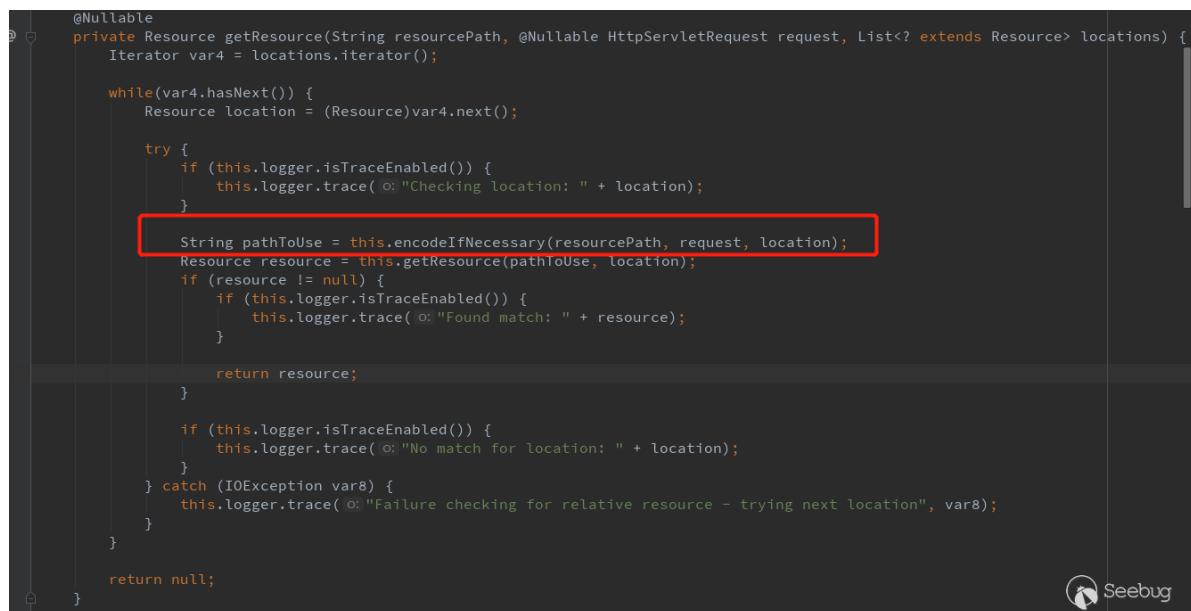
/resources/%255c%255c..%255c/..%255c/..%255c/..%255c/..%255c/..%255c/..%255c/..%255c/windows/win.ini

Spring Framework >5.0.1

jetty:(单编码)

/resources/%5c%5c..%5c/..%5c/..%5c/..%5c/..%5c/..%5c/windows/win.ini

至于为什么双编码不行,仅为在getResource函数中进行了一次url编码



```
    @Nullable
    private Resource getResource(String resourcePath, @Nullable HttpServletRequest request, List<? extends Resource> locations) {
        Iterator var4 = locations.iterator();

        while(var4.hasNext()) {
            Resource location = (Resource)var4.next();

            try {
                if (this.logger.isTraceEnabled()) {
                    this.logger.trace(o:"Checking location: " + location);
                }

                String pathToUse = this.encodeIfNecessary(resourcePath, request, location);
                Resource resource = this.getResource(pathToUse, location);
                if (resource != null) {
                    if (this.logger.isTraceEnabled()) {
                        this.logger.trace(o:"Found match: " + resource);
                    }
                }
            } catch (IOException var8) {
                this.logger.trace(o:"Failure checking for relative resource - trying next location", var8);
            }
        }

        return null;
    }
}
```

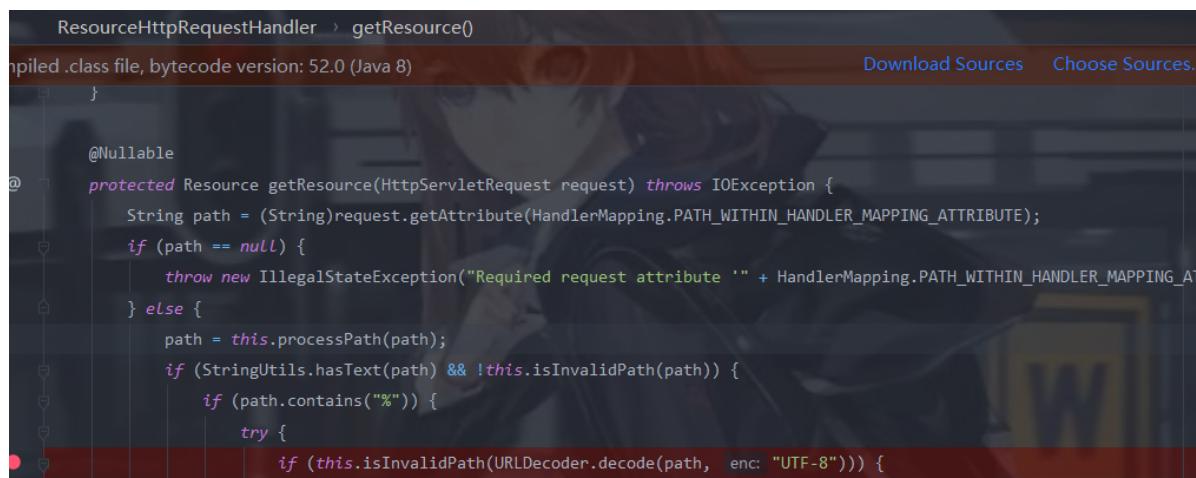
首先我们传入经过两次编码的url,isInvalidPath前进行解码一次,然后getResource函数又编码一次

当我们进入第二阶段的时候,我们传递的url是经过两次url编码的,但是它只进行一次url解码,所以无法找到文件

1. 在isValidPath方法判断路径是否合法(经过isValidPath后不存在./),但是isValidPath有点问题,可以被绕过
2. 利用createRelative方法拼接,然后判断文件是否存在并可读(判断时会进行url解码)
3. 构造response数据,在url.openConnection会进行url解码,然后成功读取我们的文件并返回

漏洞修复

这个漏洞产生的主要原因就是因为isValidPath中的cleanPath **会把 //当成一个目录**,所以修复时重写了这里的processPath方法



```
ResourceHttpRequestHandler > getResource()
Compiled .class file, bytecode version: 52.0 (Java 8) Download Sources Choose Sources.

    }

    @Nullable
    protected Resource getResource(HttpServletRequest request) throws IOException {
        String path = (String)request.getAttribute(HandlerMapping.PATH_WITHIN_HANDLER_MAPPING_ATTRIBUTE);
        if (path == null) {
            throw new IllegalStateException("Required request attribute '" + HandlerMapping.PATH_WITHIN_HANDLER_MAPPING_ATTRIBUTE + "' is not present");
        } else {
            path = this.processPath(path);
            if (StringUtils.hasText(path) && !this.isValidPath(path)) {
                if (path.contains("%")) {
                    try {
                        if (this.isValidPath(URLDecoder.decode(path, enc: "UTF-8"))) {
```

修复前:

ResourceHttpRequestHandler → processPath()

Compiled .class file, bytecode version: 52.0 (Java 8) [Download Sources](#)

```
protected String processPath(String path) {
    boolean slash = false;

    for(int i = 0; i < path.length(); ++i) {
        if (path.charAt(i) == '/') {
            slash = true;
        } else if (path.charAt(i) > ' ' && path.charAt(i) != 127) {
            if (i != 0 && (i != 1 || !slash)) {
                path = slash ? "/" + path.substring(i) : path.substring(i);
                if (logger.isTraceEnabled()) {
                    logger.trace( o: "Path after trimming leading '/' and control characters: " + path);
                }
            }
        }
    }

    return path;
}

return path;
}
```

修复后：

```
protected String processPath(String path) {  
    path = StringUtils.replace(path, oldPattern: "\\", newPattern: "/");  
    path = this.cleanDuplicateSlashes(path);  
    return this.cleanLeadingSlash(path);  
}
```

版本大于5.0.1之后就只能用单编码了,这里进来会调用getAttribute解码一次,然后传入processPath

The screenshot shows the JD-GUI Java decompiler interface. The title bar reads "ResourceHttpRequestHandler > getResource()". Below it, "Decompiled .class file, bytecode version: 52.0 (Java 8)" is displayed. On the right, there are buttons for "Download Sources" and "Choose Sources...". The main window shows the Java code for the `getResource` method. The code uses `@Nullable` annotations and handles `HttpServletRequest` to get a path attribute. It then processes the path through a `processPath` method and checks if it's valid using `isValidPath`. The code is annotated with Javadoc and includes several `else` blocks and conditionals.

```
    @Nullable
    protected Resource getResource(HttpServletRequest request) throws IOException {
        String path = (String)request.getAttribute(HandlerMapping.PATH_WITHIN_HANDLER_MAPPING_ATTRIBUTE);
        if (path == null) {
            throw new IllegalStateException("Required request attribute '" + HandlerMapping.PATH_WITHIN_HANDLER_MAPPING_ATTRIBUTE + "' is not set");
        } else {
            path = this.processPath(path);
            if (StringUtils.hasText(path) && !this.isValidPath(path)) {
                if (this.isInvalidEncodedPath(path)) {
                    throw new IllegalStateException("Path '" + path + "' is invalid encoded path");
                }
            }
        }
    }
```

这样就不存在将//当成目录了

参考

<https://xushao.ltd/post/cve-2018-1271-fen-xi/#%E7%AE%80%E4%BB%8B>

https://paper.seebug.org/665/#_1

<https://blog.spoock.com/2018/05/30/cve-2018-1271/>