前言

之前分析了 Apache Shiro 权限绕过漏洞(CVE-2020-11989)，过了一段时间又出现了新的权限绕过漏洞 (CVE-2020-13933)。应该是在修复的基础上进行了绕过。 CVE-2020-11989 的影响版本为 Apache Shiro < 1.5.3 , CVE-2020-13933 的影响版本为 Apache Shiro < 1.6.0 。

31 web/src/main/java/org/apache/shiro/web/util/WebUtils.java

@@ -109,16 +109,7 @@

Apache shiro 1.5.2

```
 109      * @return the path within the web application
 110      */
 111     public static String getPathWithinApplication(HttpServletRequest request) {
-112         String contextPath = getContextPath(request);
-113         String requestUri = getRequestUri(request);
-114         if (StringUtils.startsWithIgnoreCase(requestUri, contextPath)) {
-115             // Normal case: URI contains context path.
-116             String path = requestUri.substring(contextPath.length());
-117             return (StringUtils.hasText(path) ? path : "/");
-118         } else {
-119             // Special case: rather unusual.
-120             return requestUri;
-121         }
 122     }
 123
```

Apache shiro 1.5.3

```
 109      * @return the path within the web application
 110      */
 111     public static String getPathWithinApplication(HttpServletRequest request) {
+112         return normalize(removeSemicolon(getServletPath(request) + getPathInfo(request)));
 113     }
 114
```

在这个地方 Shiro 对 url 的处理是造成 CVE-2020-11989 的原因之一，Apache Shiro 1.5.3 对此进行的修复。

然后我们查看最新版的 Apache Shiro 1.6.0 发现在 `web/src/main/java/org/apache/shiro/web/filter/InvalidRequestFilter.java` 增加了

```
+ public class InvalidRequestFilter extends AccessControlFilter {
+
+     private static final List<String> SEMICOLON = Collections.unmodifiableList(Arrays.asList(";", "%3b", "%3B"));
+
+     private static final List<String> BACKSLASH = Collections.unmodifiableList(Arrays.asList("\\", "%5c", "%5C"));
+
+     private boolean blockSemicolon = true;
+
+     private boolean blockBackslash = true;
+
+     private boolean blockNonAscii = true;
+
+     @Override
+     protected boolean isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) throws Exception {
+         String uri = WebUtils.toHttp(request).getRequestURI();
+         return !containsSemicolon(uri)
+             && !containsBackslash(uri)
+             && !containsNonAsciiCharacters(uri);
+     }
+
+     @Override
+     protected boolean onAccessDenied(ServletRequest request, ServletResponse response) throws Exception {
+         WebUtils.toHttp(response).sendError(400, "Invalid request");
+         return false;
+     }
+
+     private boolean containsSemicolon(String uri) {
+         if (isBlockSemicolon()) {
+             return SEMICOLON.stream().anyMatch(uri::contains);
+         }
+         return false;
+     }
+
+     private boolean containsBackslash(String uri) {
+         if (isBlockBackslash()) {
+             return BACKSLASH.stream().anyMatch(uri::contains);
+         }
+         return false;
+     }
```

从全局上对分号，反斜杠和非ASCII字符进行了过滤

环境搭建

还是选择 `https://github.com/l3yx/springboot-shiro` 项目进行测试 下载完成后修改一下 `pom.xml` 中 `org.apache.shiro` 所对应的版本号 为 1.5.3 ， 同时将 `LoginController` 中修改为

```
    @GetMapping("/admin/{name}")
     public String admin(@PathVariable String name) {
         return "admin page,hello " + name;
     }
```

原因之后描述，同时为了方便调试，同时在 pom.xml 文件中加入

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-core</artifactId>
    <version>1.5.3</version>
</dependency>
```

生成 war 包，部署于Tomcat

修改 `\apache-tomcat-8.0.52\bin\catalina.bat` 文件

```
if not "%JPDA_ADDRESS%" == "" goto gotJpdaAddress
set JPDA_ADDRESS=127.0.0.1:5005
:gotJpdaAddress
```

`catalina.bat jpda start` 启动，配置 idea 中的远程调试

漏洞分析

Shiro 层

Tomcat 类中的 `org.apache.catalina.core.ApplicationFilterChain` 是用于管理针对请求 request 的过滤器。

> Tomcat 的类 ApplicationFilterChain 是一个 Java Servlet API规范 javax.servlet.FilterChain 的实现，用于管理某个请求 request 的一组过滤器 Filter 的执行。当针对一个 reques 所定义的一组过滤器 Filter 处理完该请求后，组后一个 doFilter() 调用才会执行目标 Servlet 的方法 service(),然后响应对象 response 会按照相反的顺序依次被这些Filter处理，最终到达客户端。

根据调试时显示出的调用链，可以看到先执行到了 shiro 中的 `OncePerRequestFilter` 这个类

```
"http-apr-8080-exec-6"@9,772 in group "main": RUNNING
doFilter:110, OncePerRequestFilter (org.apache.shiro.web.servlet)
internalDoFilter:240, ApplicationFilterChain (org.apache.catalina.core)
doFilter:207, ApplicationFilterChain (org.apache.catalina.core)
doFilterInternal:100, RequestContextFilter (org.springframework.web.filter)
doFilter:119, OncePerRequestFilter (org.springframework.web.filter)
internalDoFilter:240, ApplicationFilterChain (org.apache.catalina.core)
doFilter:207, ApplicationFilterChain (org.apache.catalina.core)
doFilterInternal:93, FormContentFilter (org.springframework.web.filter)
doFilter:119, OncePerRequestFilter (org.springframework.web.filter)
internalDoFilter:240, ApplicationFilterChain (org.apache.catalina.core)
doFilter:207, ApplicationFilterChain (org.apache.catalina.core)
doFilter:128, ErrorPageFilter (org.springframework.boot.web.servlet.support)
access$000:66, ErrorPageFilter (org.springframework.boot.web.servlet.support)
doFilterInternal:103, ErrorPageFilter$1 (org.springframework.boot.web.servlet.support)
doFilter:119, OncePerRequestFilter (org.springframework.web.filter)
doFilter:121, ErrorPageFilter (org.springframework.boot.web.servlet.support)
internalDoFilter:240, ApplicationFilterChain (org.apache.catalina.core)
doFilter:207, ApplicationFilterChain (org.apache.catalina.core)
doFilterInternal:201, CharacterEncodingFilter (org.springframework.web.filter)
doFilter:119, OncePerRequestFilter (org.springframework.web.filter)
internalDoFilter:240, ApplicationFilterChain (org.apache.catalina.core)
doFilter:207, ApplicationFilterChain (org.apache.catalina.core)
invoke:212, StandardWrapperValve (org.apache.catalina.core)
invoke:94, StandardContextValve (org.apache.catalina.core)
invoke:496, AuthenticatorBase (org.apache.catalina.authenticator)
invoke:141, StandardHostValve (org.apache.catalina.core)
```

在 shiro 中 `org.springframework.web.filter.OncePerRequestFilter` 这个类是其他的所有 filter 的父类，所有的 filter 的 doFilter 方法都是调用的这个类中的 doFIlter 方法。

首先调用 `getAlreadyFilteredAttributeName()` 为过滤器标记，然后判断过滤器是否已经调用过，是否未为当前请求启用。 `org.apache.shiro.web.servlet.OncePerRequestFilter#doFilter`

```
107    public final void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)  request: RequestFacade@9689  response: ErrorPageFilter$ErrorWrapperResponse@9806  filterChain:
108            throws ServletException, IOException {
109        String alreadyFilteredAttributeName = getAlreadyFilteredAttributeName();  alreadyFilteredAttributeName: "shiroFilterFactoryBean.FILTERED"
110        if ( request.getAttribute(alreadyFilteredAttributeName) != null ) {
111            log.trace("Filter '{}' already executed.  Proceeding without invoking this filter.", getName());
112            filterChain.doFilter(request, response);
113        } else //noinspection deprecation
114            if (/* added in 1.2: */ !isEnabled(request, response) ||
115                /* retain backwards compatibility: */ shouldNotFilter(request) ) {
116            log.debug("Filter '{}' is not enabled for the current request.  Proceeding without invoking this filter.",
117                    getName());
118            filterChain.doFilter(request, response);
119        } else {
120            // Do invoke this filter...
121            log.trace("Filter '{}' not yet executed.  Executing now.", getName());
122            request.setAttribute(alreadyFilteredAttributeName, Boolean.TRUE);  alreadyFilteredAttributeName: "shiroFilterFactoryBean.FILTERED"
123
124            try {
125                doFilterInternal(request, response, filterChain);  request: RequestFacade@9689  response: ErrorPageFilter$ErrorWrapperResponse@9806  filterChain: ApplicationFilterChain@9691
126            } finally {
127                // Once the request has finished, we're done and we don't
128                // need to mark as 'already filtered' any more.
129                request.removeAttribute(alreadyFilteredAttributeName);
130            }
131        }
132    }
133
134    /**
```

然后调用 `doFilterInternal` 方法，跟进后可以看到执行的是 `public abstract class AbstractShiroFilter extends OncePerRequestFilter` 中的 `doFilterInternal` 方法

org.apache.shiro.web.servlet.AbstractShiroFilter#doFilterInternal

```java
protected void doFilterInternal(ServletRequest servletRequest, ServletResponse servletResponse, final FilterChain chain)  servletRequest: RequestFacade@9689  servletResponse: ErrorPageFilter$ErrorWrapperResponse@9848  cha
        throws ServletException, IOException {

    Throwable t = null;  t: null

    try {
        final ServletRequest request = prepareServletRequest(servletRequest, servletResponse, chain);  request: ShiroHttpServletRequest@10935  servletRequest: RequestFacade@9689
        final ServletResponse response = prepareServletResponse(request, servletResponse, chain);  response: ErrorPageFilter$ErrorWrapperResponse@9848  servletResponse: ErrorPageFilter$ErrorWrapperResponse@9848  chain: Ap

        final Subject subject = createSubject(request, response);  subject: WebDelegatingSubject@10936  request: ShiroHttpServletRequest@10935  response: ErrorPageFilter$ErrorWrapperResponse@9848

        //noinspection unchecked
        subject.execute(new Callable() {  subject: WebDelegatingSubject@10936
            public Object call() throws Exception {
                updateSessionLastAccessTime(request, response);
                executeChain(request, response, chain);
                return null;
            }
        });
    } catch (ExecutionException ex) {
        t = ex.getCause();
    } catch (Throwable throwable) {
        t = throwable;
    }

    if (t != null) {
        if (t instanceof ServletException) {
            throw (ServletException) t;
```

跟进函数 executeChain

org.apache.shiro.web.servlet.AbstractShiroFilter#executeChain

```java
    protected void executeChain(ServletRequest request, ServletResponse response, FilterChain origChain)
            throws IOException, ServletException {
        FilterChain chain = getExecutionChain(request, response, origChain);
        chain.doFilter(request, response);
    }
```

跟进函数 getExecutionChain

org.apache.shiro.web.servlet.AbstractShiroFilter#getExecutionChain

```java
    protected FilterChain getExecutionChain(ServletRequest request, ServletResponse response, FilterChain origChain) {
        FilterChain chain = origChain;

        FilterChainResolver resolver = getFilterChainResolver();
        if (resolver == null) {
            log.debug("No FilterChainResolver configured.  Returning original FilterChain.");
            return origChain;
        }

        FilterChain resolved = resolver.getChain(request, response, origChain);
        if (resolved != null) {
            log.trace("Resolved a configured FilterChain for the current request.");
            chain = resolved;
        } else {
            log.trace("No FilterChain configured for the current request.  Using the default.");
        }

        return chain;
    }
```

跟进其中的 getChain

org.apache.shiro.web.filter.mgt.PathMatchingFilterChainResolver#getChain

```java
public FilterChain getChain(ServletRequest request, ServletResponse response, FilterChain originalChain) {  request: ShiroHttpServletRequest@10996  response: ErrorPageFilter$ErrorWrapperResponse@10993  originalChain: App
    FilterChainManager filterChainManager = getFilterChainManager();  filterChainManager: DefaultFilterChainManager@9715
    if (!filterChainManager.hasChains()) {  filterChainManager: DefaultFilterChainManager@9715
        return null;
    }

    String requestURI = getPathWithinApplication(request);  request: ShiroHttpServletRequest@10996

    // in spring web, the requestURI "/resource/menus" ---- "resource/menus/" bose 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
    if(requestURI != null && !DEFAULT_PATH_SEPARATOR.equals(requestURI)
            && requestURI.endsWith(DEFAULT_PATH_SEPARATOR)) {
        requestURI = requestURI.substring(0, requestURI.length() - 1);
    }


    //the 'chain names' in this implementation are actually path patterns defined by the user.  We just use them
    //as the chain name for the FilterChainManager's requirements
    for (String pathPattern : filterChainManager.getChainNames()) {
        if (pathPattern != null && !DEFAULT_PATH_SEPARATOR.equals(pathPattern)
                && pathPattern.endsWith(DEFAULT_PATH_SEPARATOR)) {
            pathPattern = pathPattern.substring(0, pathPattern.length() - 1);
        }

        // If the path does match, then pass on to the subclass implementation for specific checks:
        if (pathMatches(pathPattern, requestURI)) {
            if (log.isTraceEnabled()) {
                log.trace("Matched path pattern [" + pathPattern + "] for requestURI [" + Encode.forHtml(requestURI) + "].  " +
                        "Utilizing corresponding filter chain...");
```

跟进方法 `getPathWithinApplication`

`org.apache.shiro.web.filter.mgt.PathMatchingFilterChainResolver#getPathWithinApplication`

```
163     protected String getPathWithinApplication(ServletRequest request) {  request: ShiroHttpServletRequest@10996
164         return WebUtils.getPathWithinApplication(WebUtils.toHttp(request));  request: ShiroHttpServletRequest@10996
165     }
166 }
```

`WebUtils#getPathWithinApplication` 修复了之前 shiro 1.5.2 所存在的 url 双编码绕过问题。但是我们可以注意到最后的返回值是 `/admin/`

`org.apache.shiro.web.util.WebUtils#getPathWithinApplication`

```
111 @   public static String getPathWithinApplication(HttpServletRequest request) {  request: ShiroHttpServletRequest@9634
112         return normalize(removeSemicolon(getServletPath(request) + getPathInfo(request)));  request: ShiroHttpServletRequest@9634
113     }
114
115     /**
116      * Return the request URI for the given request, detecting an i
117      * URL if called within a RequestDispatcher include.
118      * <p>As the value returned by <code>request.getRequestURI()</c
119      * decoded by the servlet container, this method will decode it
120      * <p>The URI that the web container resolves <i>should</i> be
121      * containers like JBoss/Jetty incorrectly include ";" strings
122      * in the URI. This method cuts off such incorrect appendices.
123      *
124      * @param request current HTTP request
125      * @return the request URI
```

```
Evaluate
Expression:
ize(removeSemicolon(getServletPath(request) + getPathInfo(request)))
Use Ctrl+Shift+Enter to add to Watches
Result:
result = "/admin/"
value = {char[7]@10727}
hash = 0
```

`org.apache.shiro.web.util.WebUtils#getServletPath` 返回值为 `/admin/;whippet`

```
137 @   private static String getServletPath(HttpServletRequest request) {  request: ShiroHttpServletRequest@9634
138         String servletPath = (String) request.getAttribute(INCLUDE_SERVLET_PATH_ATTRIBUTE);  servletPath: null
139         return servletPath != null ? servletPath : valueOrEmpty(request.getServletPath());  servletPath: null  request: ShiroHttpServletRequest@9634
140     }
141
142 @   private static String getPathInfo(HttpServletRequest request) {
143         String pathInfo = (String) request.getAttribute(INCLUDE_PATH_INFO_ATTR
144         return pathInfo != null ? pathInfo : valueOrEmpty(request.getPathInfo(
145     }
146
147 @   private static String valueOrEmpty(String input) {
148         if (input == null) {
149             return "";
150         }
```

```
Evaluate
Expression:
valueOrEmpty(request.getServletPath())
Use Ctrl+Shift+Enter to add to Watches
Result:
result = "/admin/;whippet"
value = {char[15]@10728}
hash = 124734075
```

`org.apache.shiro.web.util.WebUtils#getPathInfo` 返回 `""`

```
@   private static String getPathInfo(HttpServletRequest request) {  request: ShiroHttpServletRequest@9634
        String pathInfo = (String) request.getAttribute(INCLUDE_PATH_INFO_ATTRIBUTE);  pathInfo: null
        return pathInfo != null ? pathInfo : valueOrEmpty(request.getPathInfo());  pathInfo: null  request: ShiroHttpServletRequest@9634
    }

@   private static String valueOrEmpty(String input) {
        if (input == null) {
            return "";
        }
        return input;
    }

    /**
```

```
Evaluate
Expression:
valueOrEmpty(request.getPathInfo())
Use Ctrl+Shift+Enter to add to Watches
Result:
result = ""
value = {char[0]@10729}
hash = 0
```

`org.apache.shiro.web.util.WebUtils#removeSemicolon` 将 `;` 及其之后的全部删除

```
247 @   private static String removeSemicolon(String uri) {  uri: "/admin/;whippet"
248         int semicolonIndex = uri.indexOf(';');  semicolonIndex: 7
249         return (semicolonIndex != -1 ? uri.substring(0, semicolonIndex) : uri);  semicolonIndex: 7  uri: "/admin/;whippet"
250     }
251
252     /**
253      * Return the context path for the given request, detecting an include req
254      * URL if called within a RequestDispatcher include.
255      * <p>As the value returned by <code>request.getContextPath()</code> is <i
256      * decoded by the servlet container, this method will decode it.
257      *
258      * @param request current HTTP request
```

```
Evaluate
Expression:
(semicolonIndex != -1 ? uri.substring(0, semicolonIndex) : uri)
Use Ctrl+Shift+Enter to add to Watches
Result:
result = "/admin/"
value = {char[7]@10732}
hash = 0
```

回到函数 `org.apache.shiro.web.filter.mgt.PathMatchingFilterChainResolver#getChain` 继续向下执行

如果请求的不是 `/`，就去除末尾的 `/` ，返回值就是 `/admin`

```
private static final String DEFAULT_PATH_SEPARATOR = "/";
```

```
97  public FilterChain getChain(ServletRequest request, ServletResponse response, FilterChain originalChain) {  request: Shir
98      FilterChainManager filterChainManager = getFilterChainManager();  filterChainManager: DefaultFilterChainManager@9636
99      if (!filterChainManager.hasChains()) {
100         return null;
101     }
102
103     String requestURI = getPathWithinApplication(request);  requestURI: "/admin"  request: ShiroHttpServletRequest@9634
104
105     // in spring web, the requestURI "/resource/menus" ---- "resource/menus/" bose can access the resource
106     // but the pathPattern match "/resource/menus" can not match "resource/menus/"
107     // user can use requestURI + "/" to simply bypassed chain filter, to bypassed shiro protect
108     if(requestURI != null && !DEFAULT_PATH_SEPARATOR.equals(requestURI)
109             && requestURI.endsWith(DEFAULT_PATH_SEPARATOR)) {
110         requestURI = requestURI.substring(0, requestURI.length() - 1);  requestURI: "/admin"
111     }
112
```

接着根据 `filterChainManager.getChainNames()` 获取的拦截器进行匹配

```
116     for (String pathPattern : filterChainManager.getChainNames()) {  pathPattern: "/admin/*"  filterChainManager: DefaultFilterChainManager@9636
117         if (pathPattern != null && !DEFAULT_PATH_SEPARATOR.equals(pathPattern)
118                 && pathPattern.endsWith(DEFAULT_PATH_SEPARATOR)) {
119             pathPattern = pathPattern.substring(0, pathPattern.length() - 1);
120         }
121
122         // If the path does match, then pass on to the subclass implementation for specific checks:
123         if (pathMatches(pathPattern, requestURI)) {  pathPattern: "/admin/*"  requestURI: "/admin"
124             if (log.isTraceEnabled()) {
125                 log.trace("Matched path pattern [" + pathPattern + "] for requestURI [" + Encode.forHtml(requestURI) + "].  " +
126                         "Utilizing corresponding filter chain...");
127             }
128             return filterChainManager.proxy(originalChain, pathPattern);
129         }
130     }
131
132     return null;
133 }
```

`org.apache.shiro.web.filter.mgt.PathMatchingFilterChainResolver#pathMatches`

```
150 protected boolean pathMatches(String pattern, String path) {  pattern: "/admin/*"  path: "/admin"
151     PatternMatcher pathMatcher = getPathMatcher();  pathMatcher: AntPathMatcher@10745
152     return pathMatcher.matches(pattern, path);  pathMatcher: AntPathMatcher@10745  pattern: "/admin/*"  path: "/admin"
153 }
```

`org.apache.shiro.util.AntPathMatcher#matches`

`org.apache.shiro.util.AntPathMatcher#match`

`org.apache.shiro.util.AntPathMatcher#doMatch`

```java
    protected boolean doMatch(String pattern, String path, boolean fullMatch) {  pattern: "/admin/*"  path: "/admin"  fullMatch: true
        if (path.startsWith(this.pathSeparator) != pattern.startsWith(this.pathSeparator)) {  path: "/admin"
            return false;
        }

        String[] pattDirs = StringUtils.tokenizeToStringArray(pattern, this.pathSeparator);  pattern: "/admin/*"
        String[] pathDirs = StringUtils.tokenizeToStringArray(path, this.pathSeparator);

        int pattIdxStart = 0;
        int pattIdxEnd = pattDirs.length - 1;
        int pathIdxStart = 0;
        int pathIdxEnd = pathDirs.length - 1;

        // Match all elements up to the first **
        while (pattIdxStart <= pattIdxEnd && pathIdxStart <= pathIdxEnd) {
            String patDir = pattDirs[pattIdxStart];
            if ("**".equals(patDir)) {
                break;
            }
            if (!matchStrings(patDir, pathDirs[pathIdxStart])) {
                return false;
            }
            pattIdxStart++;
            pathIdxStart++;
        }

        if (pathIdxStart > pathIdxEnd) {
            // Path is exhausted, only match if rest of pattern is * or **'s
            if (pattIdxStart > pattIdxEnd) {
                return (pattern.endsWith(this.pathSeparator) ?
```

`pattDirs` 的最后一位是 `*` 所以会返回 false

```java
        for (int i = pattIdxStart; i <= pattIdxEnd; i++) {  i: 1  pattIdxStart: 1  pattIdxEnd: 1
            if (!pattDirs[i].equals("**")) {  pattDirs: String[2]@10812  i: 1
                return false;
            }
```

没有匹配到会返回 null

```java
    for (String pathPattern : filterChainManager.getChainNames()) {
        if (pathPattern != null && !DEFAULT_PATH_SEPARATOR.equals(pathPattern)
                && pathPattern.endsWith(DEFAULT_PATH_SEPARATOR)) {
            pathPattern = pathPattern.substring(0, pathPattern.length() - 1);
        }

        // If the path does match, then pass on to the subclass implementation for specific checks:
        if (pathMatches(pathPattern, requestURI)) {
            if (log.isTraceEnabled()) {
                log.trace("Matched path pattern [" + pathPattern + "] for requestURI [" + Encode.forHtml(requestURI) + "].  " +  requestURI: "/admin"
                        "Utilizing corresponding filter chain...");
            }
            return filterChainManager.proxy(originalChain, pathPattern);  filterChainManager: DefaultFilterChainManager@9539  originalChain: ApplicationFilterChain@9277
        }
    }

    return null;
}
```

匹配到的话会指向 `ProxiedFilterChain`

> 1.路径匹配：pathMatches(pathPattern, requestURI)，默认的Fliter逐个与请求URI进行匹配；2、代理FilterChain：ProxiedFilterChain。如果匹配不上，那么直接走servlet的FilterChain，否则先走shiro的代理FilterChain（ProxiedFilterChain），之后再走servlet的FilterChain

继续单步执行

```java
protected FilterChain getExecutionChain(ServletRequest request, ServletResponse response, FilterChain origChain) {  request: ShiroHttpServletRequest@9525  response: ErrorPageFilter$ErrorWrapperResponse@9522  origChain: App
    FilterChain chain = origChain;  chain: ApplicationFilterChain@9277

    FilterChainResolver resolver = getFilterChainResolver();  resolver: PathMatchingFilterChainResolver@9322
    if (resolver == null) {
        log.debug("No FilterChainResolver configured.  Returning original FilterChain.");
        return origChain;
    }

    FilterChain resolved = resolver.getChain(request, response, origChain);  resolved: null  resolver: PathMatchingFilterChainResolver@9322  request: ShiroHttpServletRequest@9525  response: ErrorPageFilter$ErrorWrapperResp
    if (resolved != null) {
        log.trace("Resolved a configured FilterChain for the current request.");
        chain = resolved;  resolved: null
    } else {
        log.trace("No FilterChain configured for the current request.  Using the default.");
    }

    return chain;  chain: ApplicationFilterChain@9277
}
```

最后返回 `ApplicationFilterChain` 相当于并没有执行 `Filter`

此时就相当于已经绕过了 shiro 的权限验证，可以直接访问到需要权限目录下的文件，但是有时会返回这样的界面，是因为 Spring 并没有匹配到相对应的页面。

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Sep 25 16:04:12 CST 2020
There was an unexpected error (type=Not Found, status=404).
No message available

Spring层

`chain.doFilter(request, response);`

```
446     protected void executeChain(ServletRequest request, ServletResponse response, FilterChain origChain)  request: ShiroHttpServletRequest@9525  response: ErrorPageFilter$ErrorWrapperResponse@9522  origChain: ApplicationFilter
447             throws IOException, ServletException {
448         FilterChain chain = getExecutionChain(request, response, origChain);  chain: ApplicationFilterChain@9277  origChain: ApplicationFilterChain@9277
449         chain.doFilter(request, response);  chain: ApplicationFilterChain@9277  request: ShiroHttpServletRequest@9525  response: ErrorPageFilter$ErrorWrapperResponse@9522
450     }
451 }
452
```

接下来的调用栈如图

```
getHandlerInternal:363, AbstractHandlerMethodMapping (org.springframework.web.servlet.handler)
getHandlerInternal:110, RequestMappingInfoHandlerMapping (org.springframework.web.servlet.mvc.method)
getHandlerInternal:59, RequestMappingInfoHandlerMapping (org.springframework.web.servlet.mvc.method)
getHandler:395, AbstractHandlerMapping (org.springframework.web.servlet.handler)
getHandler:1234, DispatcherServlet (org.springframework.web.servlet)
doDispatch:1016, DispatcherServlet (org.springframework.web.servlet)
doService:943, DispatcherServlet (org.springframework.web.servlet)
processRequest:1006, FrameworkServlet (org.springframework.web.servlet)
doGet:898, FrameworkServlet (org.springframework.web.servlet)
service:622, HttpServlet (javax.servlet.http)
service:883, FrameworkServlet (org.springframework.web.servlet)
service:729, HttpServlet (javax.servlet.http)
internalDoFilter:292, ApplicationFilterChain (org.apache.catalina.core)
doFilter:207, ApplicationFilterChain (org.apache.catalina.core)
doFilter:52, WsFilter (org.apache.tomcat.websocket.server)
internalDoFilter:240, ApplicationFilterChain (org.apache.catalina.core)
doFilter:207, ApplicationFilterChain (org.apache.catalina.core)
executeChain:449, AbstractShiroFilter (org.apache.shiro.web.servlet)
```

> Spring 在 Tomcat 中运行时需要提供对 Servlet 规范的支持，因为 Tomcat 时基于 Servlert 规范的 web 容器。 DispatcherServlet 是 Servlet 规范的具体实现。在 web 开发过程中，启动 Tomcat 容器时会根据其 Servlet 规范启动 Spring 实现的 DispatcherServlet ，这样就驱动了 Spring 的运行。

DispatcherServlet 在将请求映射到处理器时，调用了 `getHandler`

`org.springframework.web.servlet.handler.AbstractHandlerMapping#getHandler`

```
394     public final HandlerExecutionChain getHandler(HttpServletRequest request) throws Exception {
395         Object handler = getHandlerInternal(request);
396         if (handler == null) {
397             handler = getDefaultHandler();
398         }
399         if (handler == null) {
400             return null;
401         }
```

跟进 getHandlerInternal

org.springframework.web.servlet.handler.AbstractHandlerMethodMapping#getHandlerInternal

```
362     protected HandlerMethod getHandlerInternal(HttpServletRequest request) throws Exception {
363         String lookupPath = getUrlPathHelper().getLookupPathForRequest(request);
364         request.setAttribute(LOOKUP_PATH, lookupPath);
365         this.mappingRegistry.acquireReadLock();
366         try {
367             HandlerMethod handlerMethod = lookupHandlerMethod(lookupPath, request);
368             return (handlerMethod != null ? handlerMethod.createWithResolvedBean() : null);
369         }
370         finally {
371             this.mappingRegistry.releaseReadLock();
372         }
373     }
```

通过 getLookupPathForRequest 获取请求的绝对路径

org.springframework.web.util.UrlPathHelper#getLookupPathForRequest(javax.servlet.http.HttpServletRequest)

```
166     public String getLookupPathForRequest(HttpServletRequest request) {  request: ShiroHttpServletRequest@9525
167         // Always use full path within current servlet context?
168         if (this.alwaysUseFullPath) {
169             return getPathWithinApplication(request);
170         }
171         // Else, use path within current servlet mapping if applicable
172         String rest = getPathWithinServletMapping(request);  request: ShiroHttpServletRequest@9525
173         if (!"".equals(rest)) {
174             return rest;
175         }
176         else {
177             return getPathWithinApplication(request);
178         }
179     }
180
181     /**
182      * Variant of {@link #getLookupPathForRequest(HttpServletRequest)} t
183      * automates checking for a previously computed lookupPath saved as
184      * request attribute. The attribute is only used for lookup purposes
```

```
Evaluate
Expression:
getPathWithinServletMapping(request)
Use Ctrl+Shift+Enter to add to Watches
Result:
result = "/admin/;whippet"
  value = {char[15]@10704}
  hash = 124734075
```

org.springframework.web.util.UrlPathHelper#getPathWithinServletMapping

```
215     public String getPathWithinServletMapping(HttpServletRequest request) {  request: ShiroHttpServletRequest@11039
216         String pathWithinApp = getPathWithinApplication(request);  pathWithinApp: "/admin/;whippet"
217         String servletPath = getServletPath(request);  servletPath: "/admin/;whippet"
218         String sanitizedPathWithinApp = getSanitizedPath(pathWithinApp);  sanitizedPathWithinApp: "/admin/;whippet"
219         String path;  path: null
220
221         // If the app container sanitized the servletPath, check against the sanitized version
222         if (servletPath.contains(sanitizedPathWithinApp)) {
223             path = getRemainingPath(sanitizedPathWithinApp, servletPath, ignoreCase: false);  sanitizedPathWithinApp: "/admin/;whippet"
224         }
225         else {
226             path = getRemainingPath(pathWithinApp, servletPath, ignoreCase: false);  pathWithinApp: "/admin/;whippet"  servletPath: "/admin/;whippet"
227         }
228
229         if (path != null) {
230             // Normal case: URI contains servlet path.
231             return path;  path: null
232         }
233         else {
234             // Special case: URI is different from servlet path.
```

跟进函数 getPathWithinApplication

org.springframework.web.util.UrlPathHelper#getPathWithinApplication

```
263     public String getPathWithinApplication(HttpServletRequest request) {  request: ShiroHttpServletRequest@9525
264         String contextPath = getContextPath(request);  contextPath: "/shiro"
265         String requestUri = getRequestUri(request);  request: ShiroHttpServletRequest@9525
266         String path = getRemainingPath(requestUri, contextPath,  ignoreCase: true);
267         if (path != null) {
268             // Normal case: URI contains context path.
269             return (StringUtils.hasText(path) ? path : "/");
270         }
271         else {
272             return requestUri;
273         }
274     }
275
276     /**
```

跟进函数 `getRequestUri` `org.springframework.web.util.UrlPathHelper#getRequestUri`

```
344     public String getRequestUri(HttpServletRequest request) {  request: ShiroHttpServletRequest@9525
345         String uri = (String) request.getAttribute(WebUtils.INCLUDE_REQUEST_URI_ATTRIBUTE);  uri: "/shiro/admin/%3bwhippet"
346         if (uri == null) {
347             uri = request.getRequestURI();
348         }
349         return decodeAndCleanUriString(request, uri);  request: ShiroHttpServletRequest@9525  uri: "/shiro/admin/%3bwhippet"
350     }
351
352     /**
353      * Return the context path for the given request, detecting an i
354      * URL if called within a RequestDispatcher include.
355      * <p>As the value returned by {@code request.getContextPath()}
356      * decoded by the servlet container, this method will decode it.
357      * @param request current HTTP request
358      * @return the context path
359      */
```

```
Evaluate
Expression:
decodeAndCleanUriString(request, uri)
Use Ctrl+Shift+Enter to add to Watches
Result:
result = "/shiro/admin/;whippet"
value = {char[21]@10711}
hash = 0
```

`decodeAndCleanUriString` 对 url 进行了解码处理

`org.springframework.web.util.UrlPathHelper#decodeAndCleanUriString`

```
459     private String decodeAndCleanUriString(HttpServletRequest request, String uri) {  request: ShiroHttpServletRequest@9525  uri: "/shiro/admin/;whippet"
460         uri = removeSemicolonContent(uri);
461         uri = decodeRequestString(request, uri);  request: ShiroHttpServletRequest@9525
462         uri = getSanitizedPath(uri);
463         return uri;  uri: "/shiro/admin/;whippet"
464     }
```

此处存在一个问题,因为利用 `;` 就可以直接绕过 shiro 的权限验证，但是为什么在直接使用 `;` 时会返回 404 错误，在 spring 中不能找到该页面

`org.springframework.web.util.UrlPathHelper#decodeAndCleanUriString`

```
56      /**
57       * Decode the supplied URI string and strips any extraneous portion after a ';'.
58       */
59      private String decodeAndCleanUriString(HttpServletRequest request, String uri) {  request: ShiroHttpServletRequest@10988  uri: "/shiro/admin/;whippet"
60          uri = removeSemicolonContent(uri);  uri: "/shiro/admin/;whippet"
61          uri = decodeRequestString(request, uri);
62          uri = getSanitizedPath(uri);
63          return uri;
64      }
65
66      /**
67       * Decode the given source string with a URLDecoder. The encoding will be taken
68       * from the request, falling back to the default "ISO-8859-1".
69       * <p>The default implementation uses {@code URLDecoder.decode(input, enc)}.
70       * @param request current HTTP request
71       * @param source the String to decode
72       * @return the decoded String
73       * @see WebUtils#DEFAULT_CHARACTER_ENCODING
74       * @see javax.servlet.ServletRequest#getCharacterEncoding
75       * @see java.net.URLDecoder#decode(String, String)
```

```
Evaluate
Expression:
removeSemicolonContent(uri)
Use Ctrl+Shift+Enter to add to Watches
Result:
result = "/shiro/admin/"
value = {char[13]@11003}
hash = 0
```

`decodeAndCleanUriString` 会先将 `url` 中 `;` 后面的数据进行分割然后再进行 url 解码

解决了这个小小的问题，又产生了一个大大的疑问，在 Apache Shiro权限绕过漏洞分析(CVE-2020-11989) 一文中师傅所利用的 POC 为 `/;/test/admin/page` 如果是这样的话，从 `;` 进行分割，最后得出来的应该是一直去请求 `/` 页面，不应返回权限下的页面，这个问题暂且放下，继续向下分析。

然后回到函数

`org.springframework.web.servlet.handler.AbstractHandlerMethodMapping#getHandlerInternal`

```java
    protected HandlerMethod getHandlerInternal(HttpServletRequest request) throws Exception {  request: ShiroHttpServletRequest@9525
        String lookupPath = getUrlPathHelper().getLookupPathForRequest(request);  lookupPath: "/admin/;whippet"
        request.setAttribute(LOOKUP_PATH, lookupPath);
        this.mappingRegistry.acquireReadLock();
        try {
            HandlerMethod handlerMethod = lookupHandlerMethod(lookupPath, request);  lookupPath: "/admin/;whippet"  request: ShiroHttpServletRequest@9525
            return (handlerMethod != null ? handlerMethod.createWithResolvedBean() : null);
        }
        finally {
            this.mappingRegistry.releaseReadLock();
        }
    }
```

跟进 lookupHandlerMethod

org.springframework.web.servlet.handler.AbstractHandlerMethodMapping#lookupHandlerMethod

```java
    protected HandlerMethod lookupHandlerMethod(String lookupPath, HttpServletRequest request) throws Exception {  lookupPath: "/admin/;whippet"  request: ShiroHttpServletRequest@11039
        List<Match> matches = new ArrayList<>();
        List<T> directPathMatches = this.mappingRegistry.getMappingsByUrl(lookupPath);
        if (directPathMatches != null) {
            addMatchingMappings(directPathMatches, matches, request);
        }
        if (matches.isEmpty()) {
            // No choice but to go through all mappings...
            addMatchingMappings(this.mappingRegistry.getMappings().keySet(), matches, request);
        }

        if (!matches.isEmpty()) {
            Match bestMatch = matches.get(0);
            if (matches.size() > 1) {
                Comparator<Match> comparator = new MatchComparator(getMappingComparator(request));
                matches.sort(comparator);
                bestMatch = matches.get(0);
                if (logger.isTraceEnabled()) {
                    logger.trace( o: matches.size() + " matching mappings: " + matches);
                }
                if (CorsUtils.isPreFlightRequest(request)) {
                    return PREFLIGHT_AMBIGUOUS_MATCH;
                }
                Match secondBestMatch = matches.get(1);
                if (comparator.compare(bestMatch, secondBestMatch) == 0) {
                    Method m1 = bestMatch.handlerMethod.getMethod();
                    Method m2 = secondBestMatch.handlerMethod.getMethod();
                    String uri = request.getRequestURI();
                    throw new IllegalStateException(
                            "Ambiguous handler methods mapped for '" + uri + "': {" + m1 + ", " + m2 + "}");
                }
            }
```

跟进其中的 addMatchingMappings

org.springframework.web.servlet.handler.AbstractHandlerMethodMapping#addMatchingMappings

```java
    private void addMatchingMappings(Collection<T> mappings, List<Match> matches, HttpServletRequest request) {
        for (T mapping : mappings) {
            T match = getMatchingMapping(mapping, request);
            if (match != null) {
                matches.add(new Match(match, this.mappingRegistry.getMappings().get(mapping)));
            }
        }
    }
```

```
getMatchingPatterns:257, PatternsRequestCondition (org.springframework.web.servlet.mvc.condition)
getMatchingCondition:243, PatternsRequestCondition (org.springframework.web.servlet.mvc.condition)
getMatchingCondition:241, RequestMappingInfo (org.springframework.web.servlet.mvc.method)
getMatchingMapping:95, RequestMappingInfoHandlerMapping (org.springframework.web.servlet.mvc.method)
getMatchingMapping:59, RequestMappingInfoHandlerMapping (org.springframework.web.servlet.mvc.method)
addMatchingMappings:428, AbstractHandlerMethodMapping (org.springframework.web.servlet.handler)
```

org.springframework.web.servlet.mvc.condition.PatternsRequestCondition#getMatchingCondition

```java
    public PatternsRequestCondition getMatchingCondition(HttpServletRequest request) {
        if (this.patterns.isEmpty()) {
            return this;
        }
        String lookupPath = this.pathHelper.getLookupPathForRequest(request, HandlerMapping.LOOKUP_PATH);
        List<String> matches = getMatchingPatterns(lookupPath);
        return !matches.isEmpty() ? new PatternsRequestCondition(new LinkedHashSet<>(matches), other: this) : null;
    }
```

`org.springframework.web.servlet.mvc.condition.PatternsRequestCondition#getMatchingPatterns`

```
255     public List<String> getMatchingPatterns(String lookupPath) {  lookupPath: "/admin/;whippet"
256         List<String> matches = null;  matches: null
257         for (String pattern : this.patterns) {  pattern: "/admin/{name}"
258             String match = getMatchingPattern(pattern, lookupPath);  match: "/admin/{name}"  pattern: "/admin/{name}"  lookupPath: "/admin/;whippet"
259             if (match != null) {  match: "/admin/{name}"
260                 matches = matches != null ? matches : new ArrayList<>();
261                 matches.add(match);
262             }
263         }
264         if (matches == null) {
265             return Collections.emptyList();
266         }
267         if (matches.size() > 1) {
268             matches.sort(this.pathMatcher.getPatternComparator(lookupPath));
269         }
270         return matches;
```

此时我们可以注意到 `/admin/{name}` 与 `/admin/;whippet` 能够匹配成功。 会返回`/admin/;whippet` 的页面，此时的 name 值为 `;wippet` ，如果之前我们并没有修改代码，而是固定的页面的话 访问 `/admin/page` 自然是与 `admin/;page` 匹配不上的。

参考文章

CVE-2020-13933: Apache Shiro 权限绕过漏洞分析
shiro源码篇 - shiro的filter，你值得拥有