【JS 逆向百例】某公共资源交易网，公告 URL 参数逆向分析

原创 K小哥 K哥爬虫 2022-07-20 17:55 发表于湖北

收录于合集
#JS 逆向百例

44个

声明

本文章中所有内容仅供学习交流，抓包内容、敏感网址、数据接口均已做脱敏处理，严禁用于商业用途和非法用途，否则由此产生的一切后果均与作者无关，若有侵权，请联系我立即删除！

逆向目标

- 目标：某地公共资源交易网
- 主页：aHR0cDovL2dnenkuamNzLmdvdi5jbi93ZWJzaXRlL3RyYW5zYWN0aW9uL2luZGV4
- 接口：aHR0cDovL2dnenkuamNzLmdvdi5jbi9wcm8tYXBpLWNvbnN0cnVjdGlvbi9jb25zdHJ1Y3Rpb24vYmlkZGVyL2JpZFNlY3Rpb24vbGlzdA==
- 逆向参数：链接中的 projectId、projectInfo 参数

逆向过程

抓包分析

链接进入到网站，会发现先转会圈才进入到网页，这里可能就有个渲染加载的过程，打开开发者人员工具，刷新网页，往下滑会看到抓包到了数据返回的接口：aHR0cDovL2dnenkuamNzLmdvdi5jbi9wcm8tYXBpLWNvbnN0cnVjdGlvbi9jb25zdHJ1Y3Rpb24vYmlkZGVyL2JpZFNlY3Rpb24vbGlzdA==，GET 请求，从 preview 响应预览中可以看到当前页面所有公告的信息：

Query String Parameters 中有些参数信息，各类型什么含义后文会详细讲解：

- pageNum：当前为第几页
- pageSize：页面大小
- informationType：公告类型
- projectType：项目类型
- informationName：信息类型

接下来随便点击一条公告，跳转到一个新页面，会发现网页链接变成了这种格式：XXX/index?projectId=XXX&projectInfo=XXX，生成了 projectId 和 projectInfo 两个加密参数，并且经过测试，同一个公告页面这两个加密参数的值是固定的，接下来我们需要尝试找到这两个参数的加密位置。

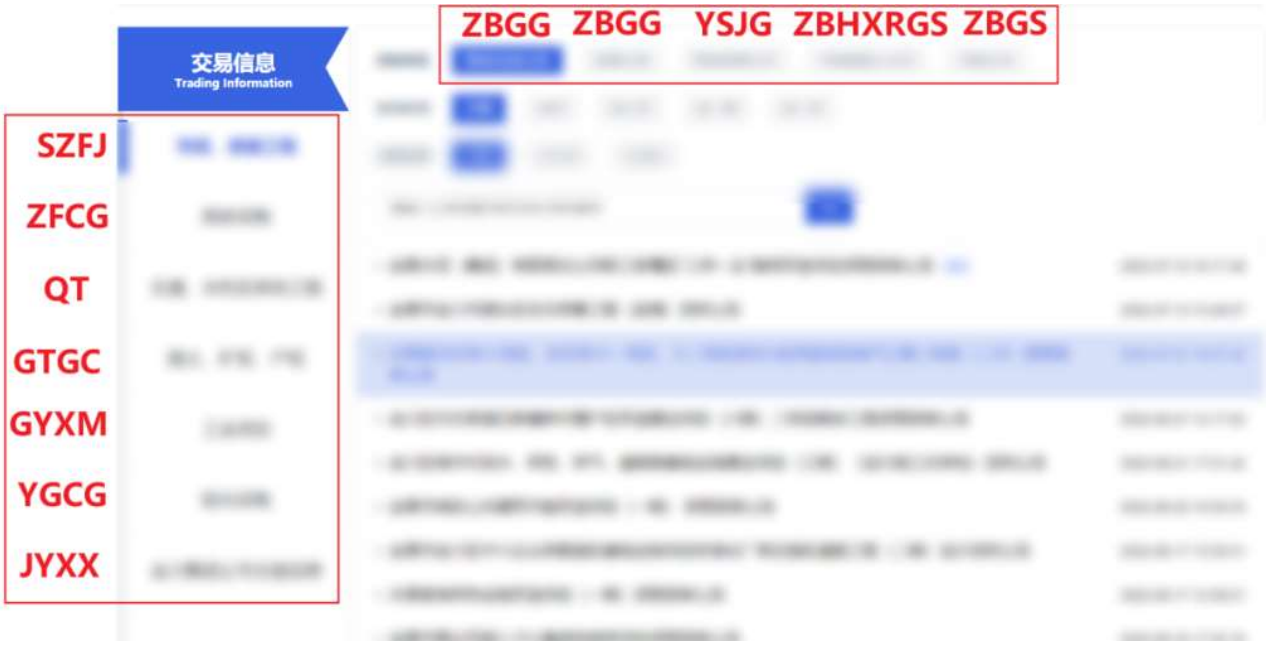
调试分析定位

从主页位置 CTRL + SHIFT + F 全局搜索 projectId 参数，依次对比可以发现，projectId 和 projectInfo 两个加密参数在 chunk-

63628500.eb5f8d30.js 中定义，这里是个三目运算，若项目类型相同则执行其后的方法，若不同则往后执行：

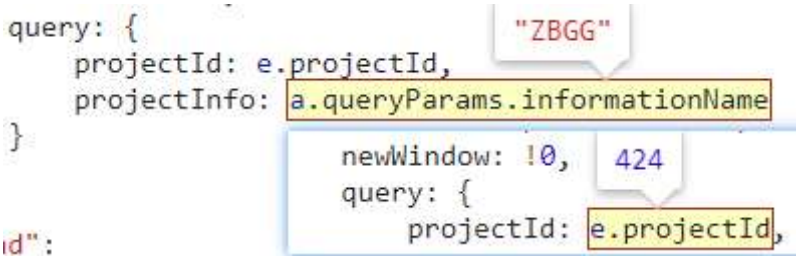
```
242                 case 0:
243                     "ZFCG" == a.queryParams.projectType ? a.parameterTool.encryptJumpPage({
244                         nextPath: "/website/anno/purchase",
245                         newWindow: !0,
246                         query: {
247                             projectId: e.projectId,
248                             projectInfo: a.queryParams.informationName
249                         }
250                     }) : "GTGC" === a.queryParams.projectType ? a.parameterTool.encryptJumpPage({
251                         nextPath: "/website/anno/land",
252                         newWindow: !0,
253                         query: {
254                             projectId: e.projectId,
255                             projectInfo: a.queryParams.informationName
256                         }
257                     }) : a.parameterTool.encryptJumpPage({
258                         nextPath: "/website/anno/index",
259                         newWindow: !0,
260                         query: {
261                             projectId: e.projectId,
262                             projectInfo: a.queryParams.informationName
263                         }
264                     });
265                 case 2:
266                 case "end":
267                     return t.stop()
```

上文代码行判断中出现的 ZFCG、GTGC 是什么意思呢，CTRL + SHIFT + F 全局搜索 ZBGG 参数，在 chunk-043c03b8.34f6abab.js 文件中我们可以找到相应的定义，以下即各自的含义：



在第 267 行，return t.stop() 处打下断点进行调试分析，随便点击一条公告，会发现断点断住，即成功定位，鼠标悬停在 projectId 和 projectInfo 对应的值上，可以知道以下信息：

- projectId ：项目编号
- projectInfo ：信息类型



知道了两个加密参数的具体含义，接下来我们就需要找到其加密位置了，projectId和projectInfo 参数由 a.parameterTool.encryptJumpPage 方法执行，encryptJumpPage 跳转页面加密？这不简直就是明示：

我们将鼠标悬停在 a.parameterTool.encryptJumpPage 上，跟进到方法生成的 js 文件 app.3275fd87.js 中去瞅瞅：

以上我们可以清晰地知道下面两个参数的具体含义：

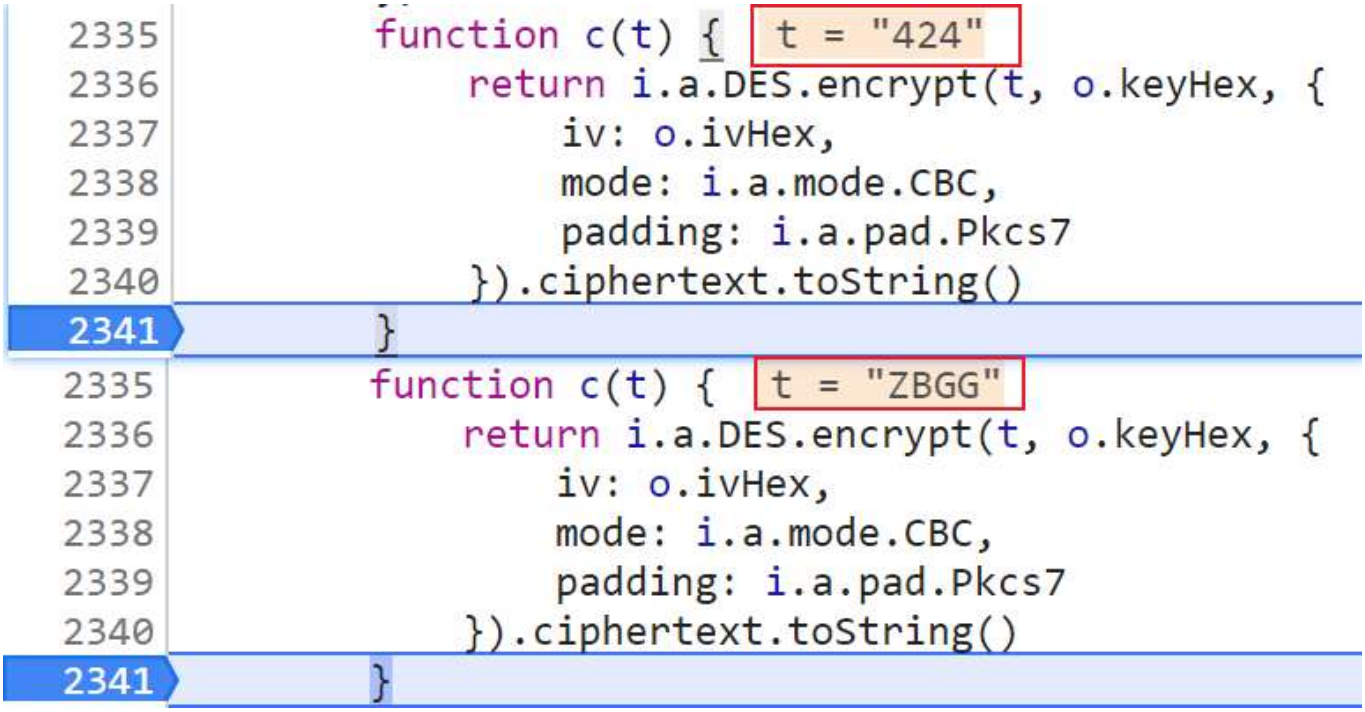
- query：加密数据（ projectId 和 projectInfo）
- nextPath：路由跳转位置

在第 2389 行打断点进行调试分析，从下图可以知道，projectId 和 projectInfo 参数在 a 中被加密了：

进一步跟踪 a 的位置，往上滑可以看到第 2335 行到 2356 行是很明显的 DES 加密：

```
2335        function c(t) {
2336            return i.a.DES.encrypt(t, o.keyHex, {
2337                iv: o.ivHex,
2338                mode: i.a.mode.CBC,
2339                padding: i.a.pad.Pkcs7
2340            }).ciphertext.toString()
2341        }
2342        function l(t) {
2343            var n;
2344            t = i.a.DES.decrypt({
2345                ciphertext: i.a.enc.Hex.parse(t)
2346            }, o.keyHex, {
2347                iv: o.ivHex,
2348                mode: i.a.mode.CBC,
2349                padding: i.a.pad.Pkcs7
2350            });
2351            try {
2352                n = JSON.parse(t.toString(i.a.enc.Utf8))
2353            } catch (e) {
2354                n = t.toString(i.a.enc.Utf8)
2355            }
2356            return n
```

但具体是哪个函数部分对 query 中的 projectId 和 projectInfo 参数进行了加密还不得而知，我们继续打断点调试分析，在 2341 行打断点时发现，projectId 参数对应的值 424，projectInfo 参数对应的值 ZBGG，都在 function c(t) 中进行了处理，证明此处就是关键的加密位置：




```
function c(t) {
    return i.a.DES.encrypt(t, o.keyHex, {
        iv: o.ivHex,
        mode: i.a.mode.CBC,
        padding: i.a.pad.Pkcs7
    }).ciphertext.toString()
}
```

分析这段关键的加密代码：

- iv：ivHex 十六进制初始向量
- mode：采用 CBC 加密模式，其是一种循环模式，前一个分组的密文和当前分组的明文异或操作后再加密
- padding：采用 Pkcs7 填充方式，在填充时首先获取需要填充的字节长度 = 块长度 - （数据长度 % 块长度），在填充字节序列中所有字节填充为需要填充的字节长度值
- ciphertext.toString()：将加密后的密文，以十六进制字符串形式返回

模拟执行

这里直接引用 JS，使用 nodejs 里面的加密模块 crypto-js 来进行 DES 加密，调试过程中提示哪个函数未定义，就将其定义部分添加进来即可，改写后的完整 JS 代码如下：

```
var CryptoJS = require('crypto-js');

o = {
```

```
    keyHex: CryptoJS.enc.Utf8.parse(Object({
        NODE_ENV: "production",
        VUE_APP_BASE_API: "/pro-api",
        VUE_APP_CONSTRUCTION_API: "/pro-api-construction",
        VUE_APP_DEV_FILE_PREVIEW: "/lyjcdFileView/onlinePreview",
        VUE_APP_FILE_ALL_PATH: "http://www.lyjcd.cn:8089",
        VUE_APP_FILE_PREFIX: "/mygroup",
        VUE_APP_LAND_API: "/pro-api-land",
        VUE_APP_PREVIEW_PREFIX: "/lyjcdFileView",
        VUE_APP_PROCUREMENT_API: "/pro-api-procurement",
        VUE_APP_WINDOW_TITLE: "XXXXXX",
        BASE_URL: "/"
    }).VUE_APP_CUSTOM_KEY || "54367819"),
    ivHex: CryptoJS.enc.Utf8.parse(Object({
        NODE_ENV: "production",
        VUE_APP_BASE_API: "/pro-api",
        VUE_APP_CONSTRUCTION_API: "/pro-api-construction",
        VUE_APP_DEV_FILE_PREVIEW: "/lyjcdFileView/onlinePreview",
        VUE_APP_FILE_ALL_PATH: "http://www.lyjcd.cn:8089",
        VUE_APP_FILE_PREFIX: "/mygroup",
        VUE_APP_LAND_API: "/pro-api-land",
        VUE_APP_PREVIEW_PREFIX: "/lyjcdFileView",
        VUE_APP_PROCUREMENT_API: "/pro-api-procurement",
        VUE_APP_WINDOW_TITLE: "XXXXXX",
        BASE_URL: "/"
    }).VUE_APP_CUSTOM_IV || "54367819")
};

function c(t) {
    return CryptoJS.DES.encrypt(t, o.keyHex, {
        iv: o.ivHex,
        mode: CryptoJS.mode.CBC,
        padding: CryptoJS.pad.Pkcs7
    }).ciphertext.toString()
}

// 测试
// console.log(c('ZBGG'))
// ff15d186c4d5fa7a
```

VUE_APP_WINDOW_TITLE 对应值内容经过脱敏处理，经测试，不影响结果输出

完整代码

GitHub 关注 K 哥爬虫，持续分享爬虫相关代码！欢迎 star ！

https://github.com/kgepachong/

以下只演示部分关键代码，不能直接运行！ 完整代码仓库地址：

https://github.com/kgepachong/crawler/

本案例代码：

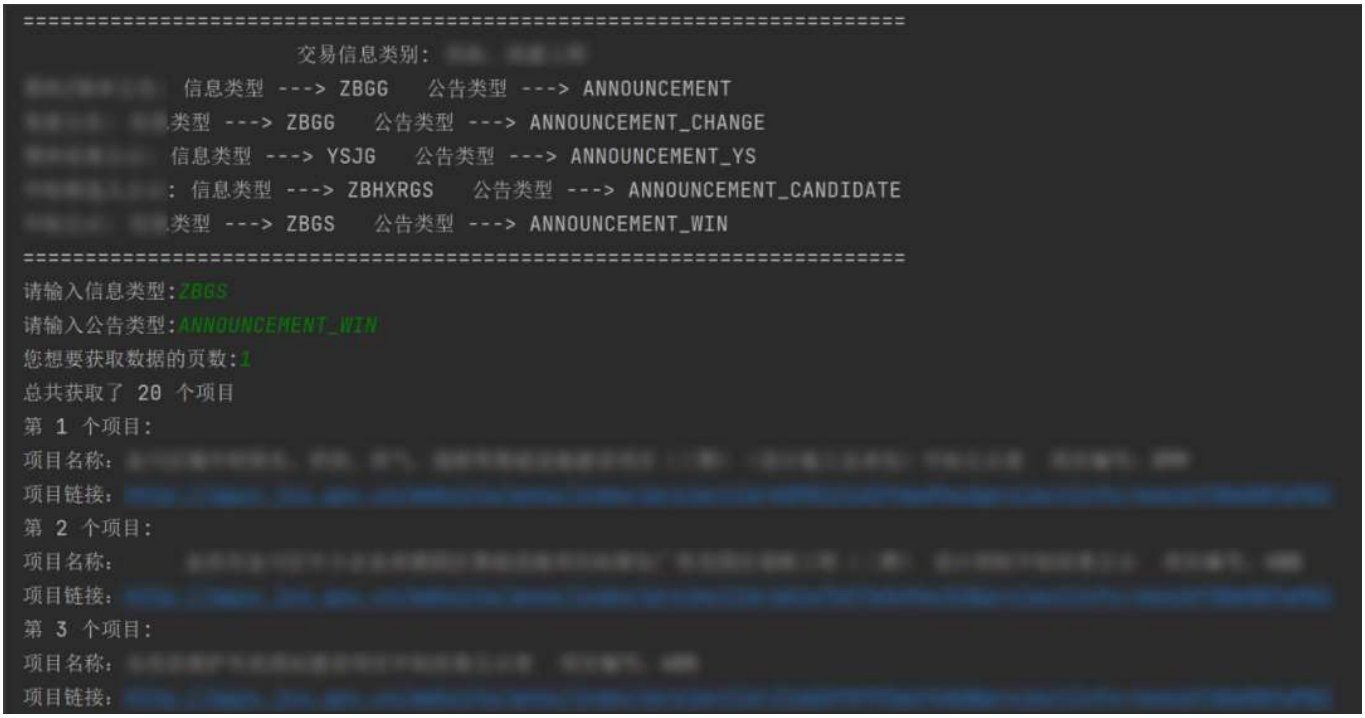
https://github.com/kgepachong/crawler/tree/main/ggzy_jcs_gov_cn

```
# =======================
# --*-- coding: utf-8 --*--
# @Author  : 微信公众号：K哥爬虫
# @FileName: ggzy.py
# @Software: PyCharm
# =======================


import urllib.parse
import execjs
import requests


url = '脱敏处理，完整代码关注 https://github.com/kgepachong/crawler/'


def encrypted_project_id(id_enc):
    with open('ggzy_js.js', 'r', encoding='utf-8') as f:
        public_js = f.read()
        project_id = execjs.compile(public_js).call('Public', id_enc)
```

```
    return project_id


def encrypted_project_info(info_enc):
    with open('ggzy_js.js', 'r', encoding='utf-8') as f:
        public_js = f.read()
        project_info = execjs.compile(public_js).call('Public', info_enc)
    return project_info


def get_project_info(info_name, info_type):
    index_url = '脱敏处理，完整代码关注 https://github.com/kgepachong/crawler/'
    urlparse = urllib.parse.urlparse(index_url)
    project_info = urllib.parse.parse_qs(urlparse.query)['informationName'][0]
    return project_info


def get_content(page, info_name, info_type):
    headers = {
        "Connection": "keep-alive",
        "Pragma": "no-cache",
        "Cache-Control": "no-cache",
        "Accept": "application/json, text/plain, */*",
        "User-
Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4246
        "Referer": "脱敏处理，完整代码关注 https://github.com/kgepachong/crawler/",
        "Accept-Language": "zh-CN,zh;q=0.9"
    }
    url_param = "脱敏处理，完整代码关注 https://github.com/kgepachong/crawler/"
    params = {
        "pageNum": page,
        "pageSize": "20",
        "releaseTime": "",
        "search": "",
        "informationType": info_type,
        "departmentId": "",
        "projectType": "SZFJ",
        "informationName": info_name,
        "onlyCanBidSectionFlag": "NO"
    }
    response = requests.get(url=url_param, headers=headers, params=params)
```

```
    return response


def main():
    print("脱敏处理，完整代码关注 https://github.com/kgepachong/crawler/")
    info_name = input("请输入信息类型:")
    info_type = input("请输入公告类型:")
    page = input("您想要获取数据的页数:")
    get_content(page, info_name, info_type)
    response = get_content(page, info_name.upper(), info_type.upper())
    num = int(page) * 20
    print("总共获取了 %d 个项目" % num)
    for i in range(20):
        title = response.json()['rows'][i]['content']
        query_id = response.json()['rows'][i]['projectId']
        query_info = get_project_info(info_name.upper(), info_type.upper())
        project_id_enc = encrypted_project_id(str(query_id))
        project_info_enc = encrypted_project_info(query_info)
        project_url = '%s?projectId=%s&projectInfo=%s' % (url, project_id_enc, project_info_enc)
        print("第 %d 个项目:" % (i+1) + "\n" + "项目名称：%s  项目编号：%d " % (title, query_id) + "\n"
%s" % project_url)


if __name__ == '__main__':
    main()
```

代码实现效果：



喜欢作者

1 人喜欢

收录于合集 #JS 逆向百例 44

上一篇
【JS 逆向百例】某网站加速乐 Cookie 混淆逆向详解

下一篇
人均瑞数系列，瑞数 4 代 JS 逆向分析

写下你的留言

精选留言

小小白 来自山东

K哥哥啥时候更新rs6啊，血求

K哥爬虫(作者)

不会6哇，小小白dddd

来自浙江

已阅，占位等大佬

零 来自湖北

K哥真的yyds 发太快了学的跟不上了