浅谈渗透江湖之细水柔情

谢公子学安全 今天

以下文章来源于酒仙桥六号部队 ，作者队员编号020

酒仙桥六号部队
知其黑，守其白。 分享知识盛宴，闲聊大院趣事，备好酒肉等你！

这是 **酒仙桥六号部队** 的第 **19** 篇文章。
全文共计4030个字，预计阅读时长12分钟。

1 前言

在渗透测试的江湖里，不只有getshell后在刀光剑影的内网中拿下域控的快意恩仇，更有侧重于业务逻辑的细水柔情。业务安全需要去细腻的考虑方方面面，更偏向于逻辑漏洞的一个思路挖掘。

在接到一个待测试目标站点后，不只要对常规漏洞的点去进行渗透测试，还要对各个功能模块所可能存在的逻辑漏洞进行挖掘。文件上传等多种方式getshell固然可以对站点一剑封喉，但一些严重敏感信息泄露、逻辑设计缺陷跟流程缺陷，虽然细腻不易察觉，却同样招招致命。

2 通用业务逻辑漏洞模块

通用业务逻辑漏洞模块，是各行业共性的业务逻辑安全风险点，在每次的测试过程中也是最常见到的模块，也是测试的重点对象，所以网上介绍此类逻辑漏洞的文章很多，在这里再做下简单的思路点总结：

1.注册模块

- 恶意用户批量注册。
- 用户登录账号、id、昵称身份覆盖。

2.登录认证模块

- 密码爆破。
- 空密码登录。

•登录越权：修改登录请求返回包中的用户id等。

3.找回密码（修改密码）模块

•前端js校验绕过。

•邮箱重置密码链接弱token遍历。

4.验证码模块（图形、滑块、手机、邮箱）

•验证码前端js绕过。

•验证码空参数或删除验证码绕过。

•图形验证码长宽DDOS。

•验证码单一用户无限制发送。

•验证码重复使用。

•验证码低位数爆破。

•验证码回传泄露。

•验证码越权接收。

•验证码重复发送同一值。

5.支付交易（充值、提现、抽奖、优惠券、会员）等多个模块

•金额、数量负值/小数。

•总金额=商品金额+优惠券金额（只校验订单总金额，而不单独校验优惠券金额跟商品金额，可增大优惠券金额）。

•订单参数混淆干扰（在同一个订单内提交两个或多个金额参数，如price=1&price=-1）。

•校验商品总数量不能为负数，而不校验单个数量，可以设置两个商品一个数量为-1，一个数量为2。

•越权使用他人优惠券。

•首充优惠、升级会员等，多台设备同一账号同时进入支付宝微信第三方支付页面，此时签名订单已生成，支付时不会变成其他金额，可依次以优惠价格支付订单。

•小数点精度：0.019=0.02（比如充值0.019元，第三方支付截取到分也就是0.01元，但是系统四舍五入为0.02）。

•int型溢出（超过最大值整数溢出）遍历优惠券id，有可能遍历出测试隐藏的无条件大额优惠券。

•首充、提现、抽奖、领取优惠券等并发：不一定非要用同一个数据包去进行多次并发操作，可用bp等工具拦截客户端数据包，快速多次点击相应客户端按钮，然后停止拦截，并发请求。

3 专项业务逻辑漏洞模块

由于业务逻辑漏洞更侧重的是思路，再加上师傅们超脱银河系的脑洞，各行各业的业务逻辑漏洞被挖掘的越来越多。接下来以我自身案例以及网上分享的逻辑漏洞思路点，给大家汇总一下不同行业所特有的逻辑漏洞风险点：

1.直播

•快速进出房间炸房。

•无限发送点赞协议。

•修改礼物数量，0，小数，负数，特定值（一般情况下为1073741824）。

•修改礼物ID，遍历尝试是否有隐藏ID。

•并发送礼物，抽奖。

•无限创建首次优惠订单，有些首次优惠订单是一个特殊的pid，这种的直接替换pid进行支付。有些是相同的ID，这种的提前创建订单，记录多个订单号在依次修改订单支付。

•刷屏：发言刷屏，分享，点赞等有提示的地方刷屏房间内可以申请的地方进行申请取消操作，看看是否能炸房。

•越权踢人，增加管理员，关闭房间等操作。

•发送的表情是否可以修改长宽（真实案例）。

•加密直播尝试删除页面锁定弹窗对应div标签。

2.外卖

•商品数量，0，负数，小数，特定值，正负数（A为-1，B为2，总值为1）。

•送餐员评价修改，星级，打赏金额（小数，负数）。

•订单商品评价，星级，评论字数，上传图片是否可以自定义格式。

•订单超出送餐地址。

•强行货到付款，取消订单，退款。

•越权操作别人订单，登录。

•优惠购买会员（重复使用优惠购买）。

3.社交论坛

•强行举报（读取本地消息上传那种）。

•强行加好友（一般尝试重发通过好友这条协议）。

•自由修改号码（靓号类）。

•群管理无限禁言越权禁言，踢人，拉黑。

•会员修改金额，数量，无限优惠购买。

•非会员使用会员功能。

4.购物

•购买数量：为0，小数，负数，正负值（A为-1，B为2，总值为1）。

•代金券：并发领取，遍历领取。

•同一个代金券重复使用。

•未满足条件使用代金券。

5.读书/漫画

•打赏金额为负数，小数，特定值（溢出）。

•越权删除评论，登录。

•修改充值金额。

•付费漫画免费看。

•评论图片数量过多会导致客户端加载卡死。

6.音乐

•唱歌类软件修改上传分数等参数。

•付费下载尝试替换下载ID。

•修改付费下载金额。

•F12查看下是否有歌曲地址。

7.网约车

•无限叫车，重复发送协议造成市场混乱。

•修改评价分数。

•修改限时优惠叫车关键参数。

•越权操作其他订单。

8.交易平台

•钱包并发提现，负数提现。

•使用钱包支付时多个订单并发支付（是否支付金额能大于余额）。

•转账负数，并发转账。

•上架商品突破限制，例如数量，字数。

•替换订单，创建订单号如果订单状态可修改，先进到支付界面，然后将订单修改成更大的金额，然后支付提前进入的支付界面。

•数量修改。

9.快递

•根据距离计算金额时选择近距离，在最终生成订单时进行收货地址修改。

•订单重量修改。

•无验证码限制无限发送上门取件订单。

•快递员评价分数刷分。

•订单遍历。

10.教育

•免费领取课程遍历id/替换收费课程id。

•试看课程抓包查看详情是否返回所有课程链接（会员视频课程同理，会员到期仍可观看或会员权限下可看到专享课程视频链接）。

•自助模拟考试多次重复答题刷分。

•顺序缺陷绕过支付获取课程链接。

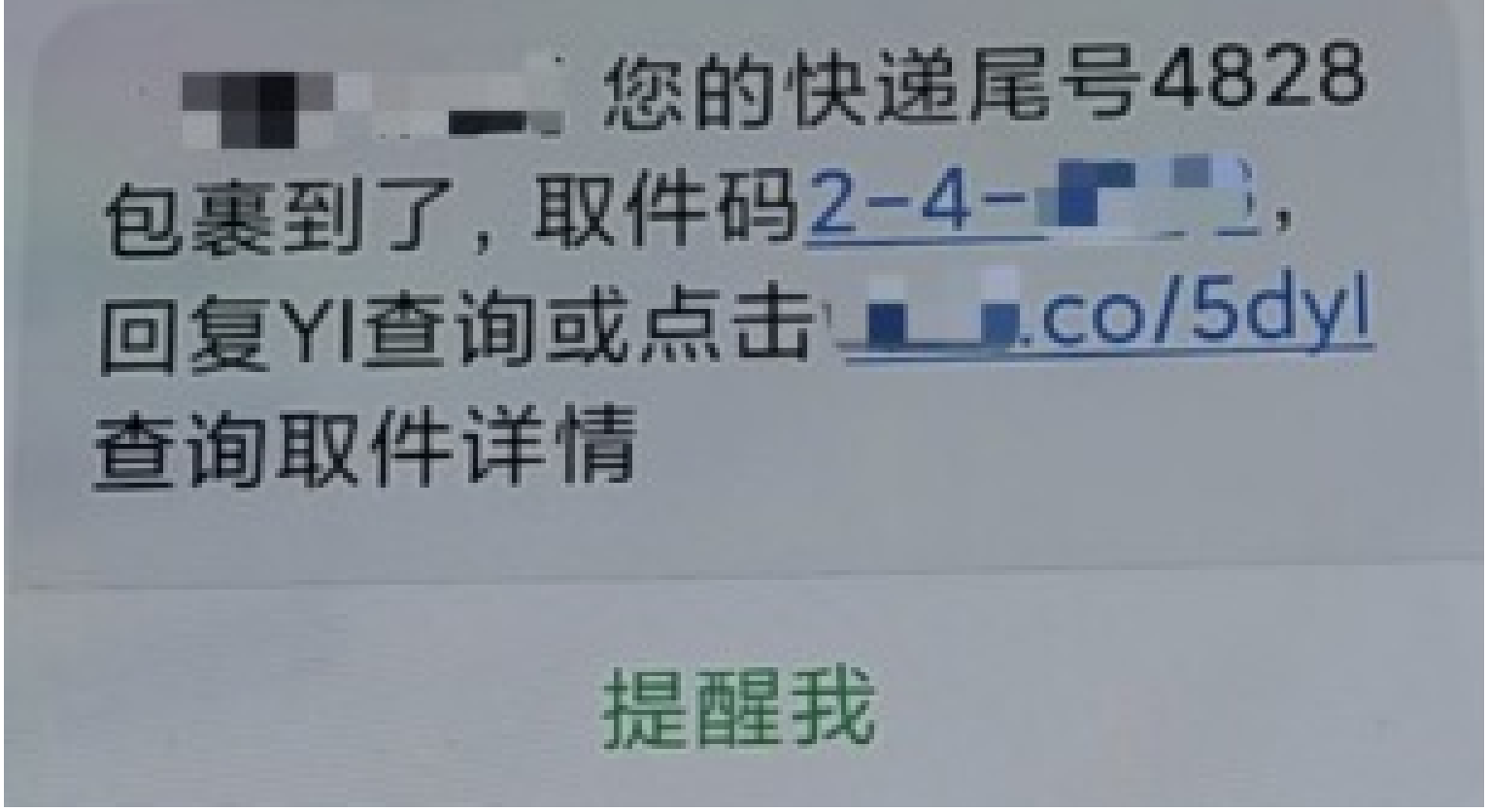
•教师端篡改课时提前结取薪酬。

4 案例分享

现在越来越多的站都采用前后端分离，尤其是微信公众号、小程序这种，很多都是采用纯接口json格式来传递参数信息，但这种纯接口传递参数对于一些像商城、论坛这种数据多，用户交互密切，也很容易传递多余不必要的敏感参数，发生信息泄露。比如查看某个活动排名，在页面只显示用户脱敏的姓名以及手机号等，但抓包查看接口时，却发现此接口返回数据是脱敏前的敏感数据，有些夸张的甚至会返回用户的密码等。基于这种思路，给大家分享两个小小案例。

1.身为一名正义的白帽子，我时常活（qian）跃（fu）在多个行业的业务群里，积极交（shou）流（ji）着上线的最新活动跟业务。而第一个案例，就来源于某个时间某个群内偶然的一条短信。

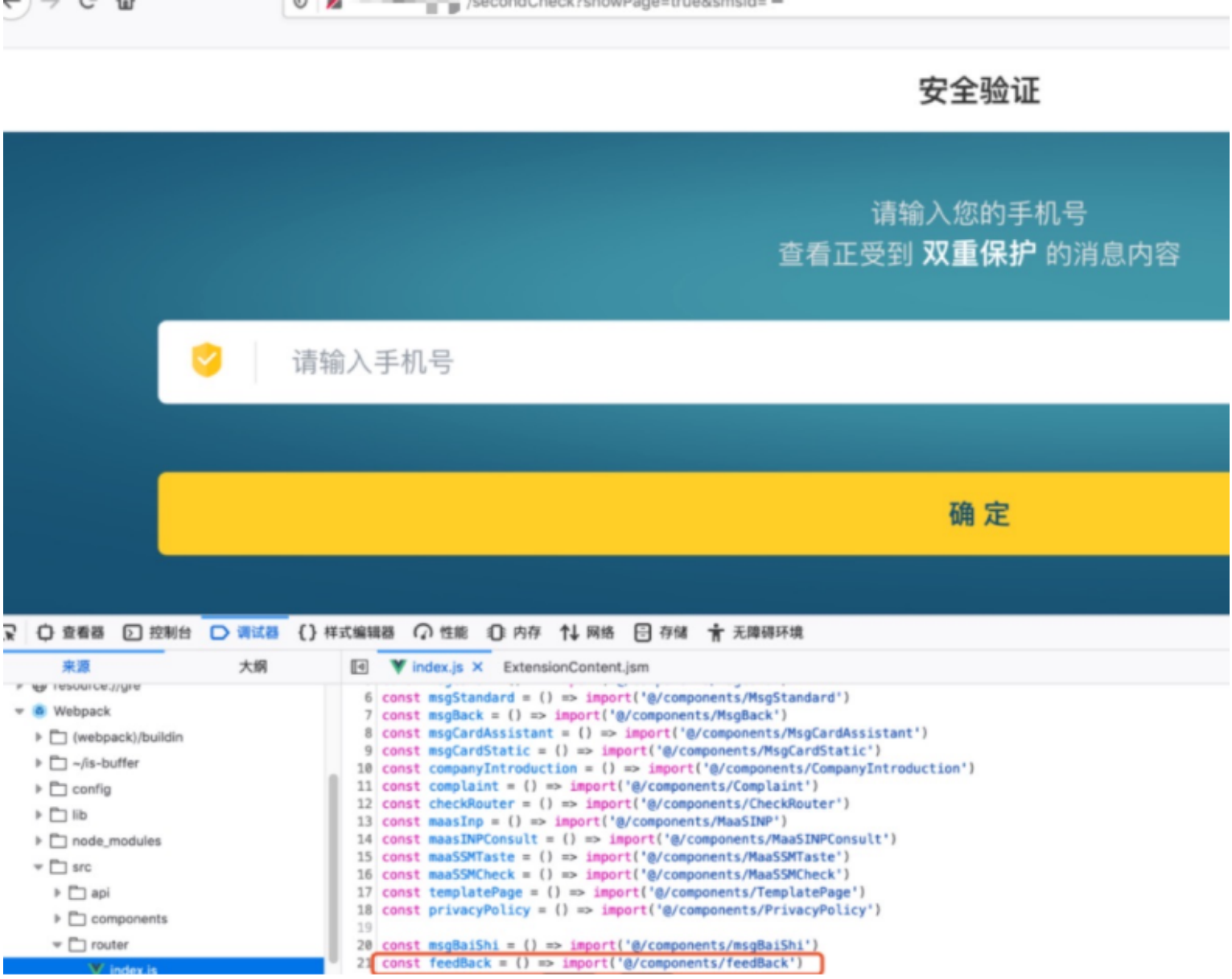
不好意思，放错图了，真实短信来源于某快递公司。



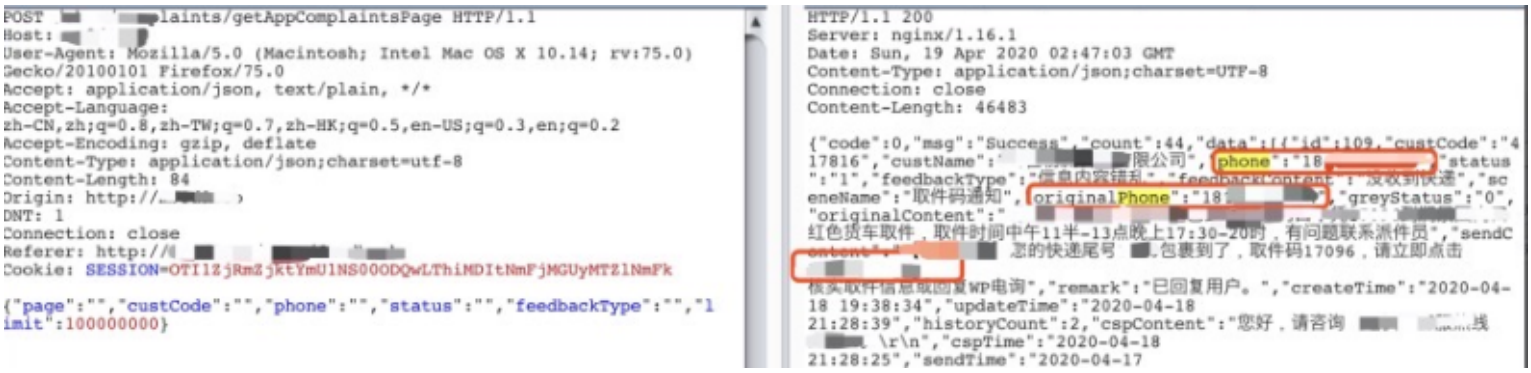
短链接服务是该公司在短信中最新推出的业务，而看到四位数的短链接，相信大家可能跟我一样，最先想到的便是去遍历穷举短链接地址，看是否可以遍历出多个用户的收货信息。

但当我输入链接查看的时候，发现需要手机号二次认证，想法是美好的，现实却总是残酷的。

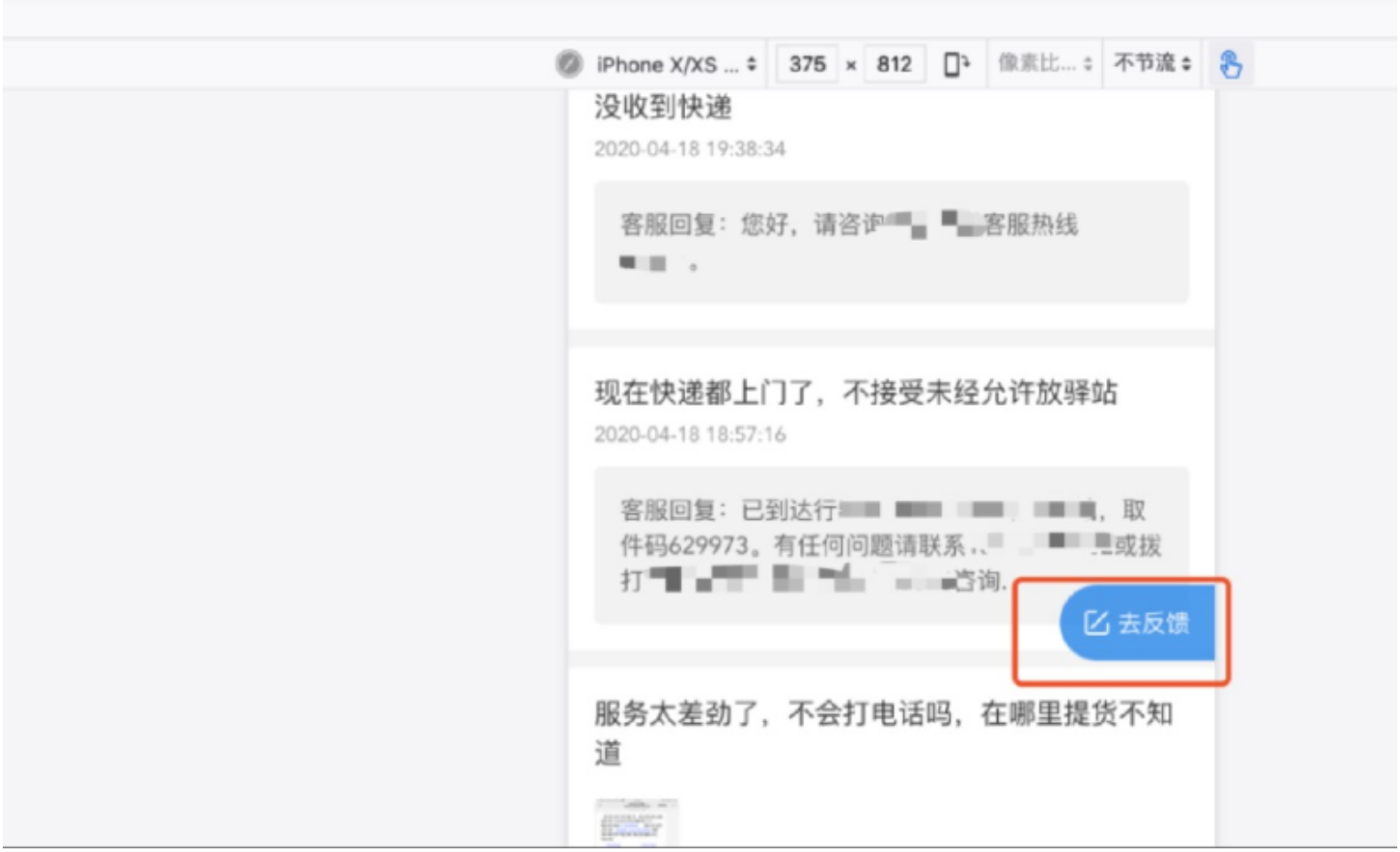
后续查看webpack的router路由时发现一处留言反馈页面。



访问此链接时发现存在接口敏感信息泄露，每个问题反馈都泄露了相应用户的收货信息，包括收货手机号、收货码、收货短链接等。



但可能因为业务刚开通，留言反馈条数并不是很多，此时泄露用户数据有限，后续测试发现在留言反馈处存在一处图片上传点，找不到逻辑漏洞，要是能挖到一处上传也是很香的。



尝试任意文件上传后发现服务器对上传文件强制重命名为.jpg格式，查看路径发现被上传至存储服务器，也无法解析脚本文件，任意文件上传失败。

再次经历现实的毒打果然让我清醒了许多，仔细查看发现在上传处有一个smsId参数。

```
POST /show/complaints/putComplaints HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 12_0 like Mac OS X)
AppleWebKit/604.1.38 (KHTML, like Gecko) Version/12.0 Mobile/15A372
Safari/604.1
Accept: application/json, text/plain, */*
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
token: rWzmU5v1
Content-Type: multipart/form-data;
boundary=--------------------------115997616534247130813060335304
Content-Length: 1897
Origin: [redacted]
DNT: 1
Connection: close
Referer: [redacted]/complaint?smsId=U4I
Cookie: SESSION=OTI1ZjRmZjktYmUlNS00ODQwLThiMDItNmFjMGUyMTZlNmFk
Pragma: no-cache
Cache-Control: no-cache

----------------------------115997616534247130813060335304
Content-Disposition: form-data; name="phone"

13313313322
----------------------------115997616534247130813060335304
Content-Disposition: form-data; name="smsId"

1234
----------------------------115997616534247130813060335304
Content-Disposition: form-data; name="content"

123123
----------------------------115997616534247130813060335304
Content-Disposition: form-data; name="type"

3
----------------------------115997616534247130813060335304
Content-Disposition: form-data; name="file0"; filename="123.jpg"
Content-Type: image/jpeg

◆◆◆◆ JFIFdd◆◆C
```

```
HTTP/1.1 200
Server: nginx/1.16.1
Date: Sun, 19 Apr 2020 03:21:03 GMT
Content-Type: application/json;charset=UTF-8
Connection: close
Content-Length: 29

{"status":0,"desc":"Success"}
```

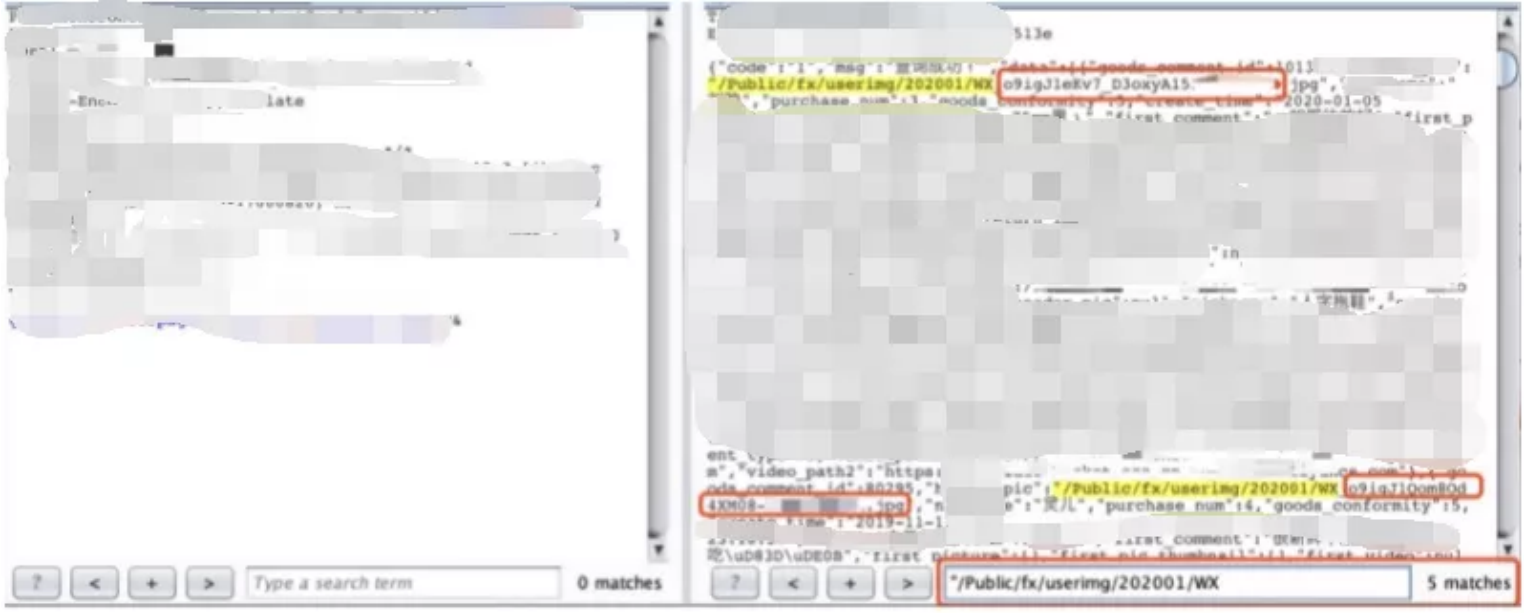
当对此参数进行修改时，发现在服务端会直接根据此smsId获取用户的源手机号（即收货手机号）、取货码、短链接并写入同一个问题反馈详情中。

```
POST [redacted]complaints/getAppComplaintsPage HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:75.0)
Gecko/20100101 Firefox/75.0
Accept: application/json, text/plain, */*
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/json;charset=utf-8
Content-Length: 78
Origin: [redacted]
DNT: 1
Connection: close
Referer: [redacted]

{"page":"","custCode":"","phone":"","status":"","feedbackType":"","l
imit":1}
```

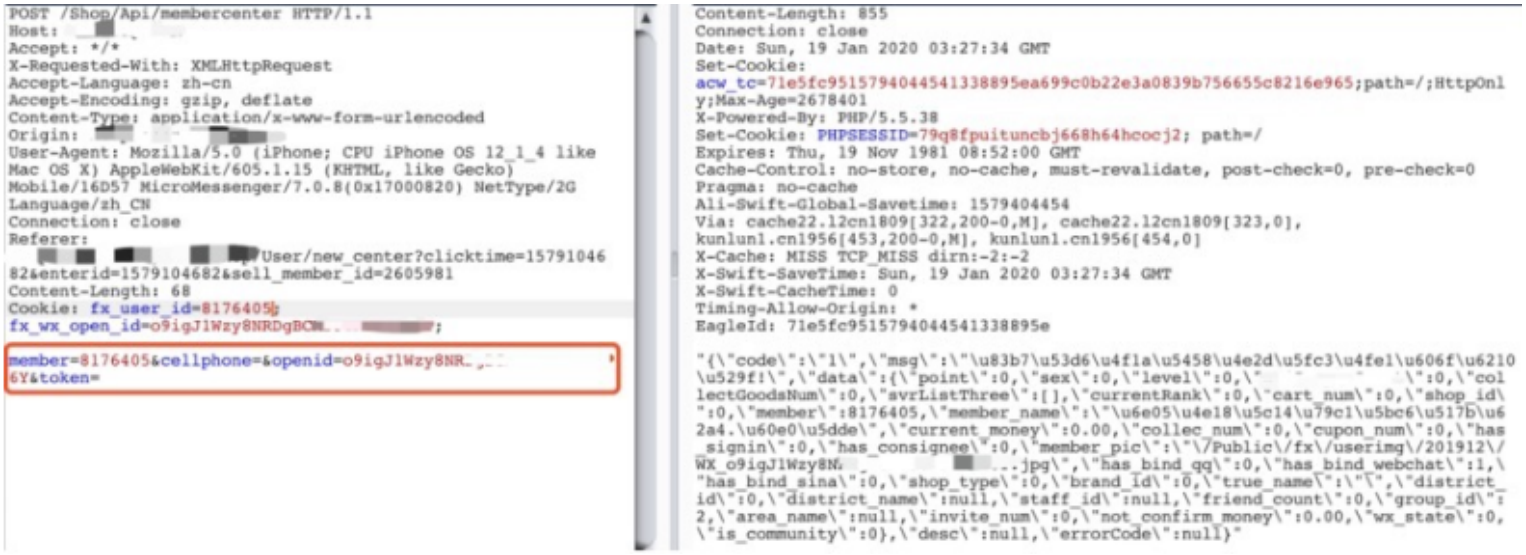
```
HTTP/1.1 200
Server: nginx/1.16.1
Date: Sun, 19 Apr 2020 03:07:31 GMT
Content-Type: application/json;charset=UTF-8
Connection: close
Set-Cookie:
SESSION=NWRkYWViNzMtNjFiNy00N2U4LWI4MTgtODhmMWIwZWMzZWMz;
Path=/show/; HttpOnly; SameSite=Lax
Content-Length: 802

{"code":0,"msg":"Success","count":47,"data":[{"id":112,"custCode":"4
17816","custName":"","phone":"13313313322","status":"0","feedbackTy
pe":"信息内容错乱","feedbackContent":"123123","picture1":"[redacted]
[redacted]/45ee6ffa95282f6c581bb9c00dbc2fc7.jpg","sceneNa
me":"取件码通知","originalPhone":"15[redacted]","greyStatus":"0","orig
inalContent":"[redacted]1A19:20前凭[redacted]4387到[redacted]
[redacted]电话1[redacted]","sendContent":"[redacted]
取件码4387，回复34查看详情或点击[redacted]/1234
快速取件","remark":"","createTime":"2020-04-19
11:00:26","updateTime":"2020-04-19
11:00:25","historyCount":0,"cspContent":"","cspTime":"","sendTime":
"2020-03-25 15:33:17","customerServiceReplies":[]}]}
```

当通过上传点修改遍历此smsId参数，再结合之前的接口敏感信息泄露，即可获取所有用户收货手机号及取货码等信息，极大危害用户的财产安全。

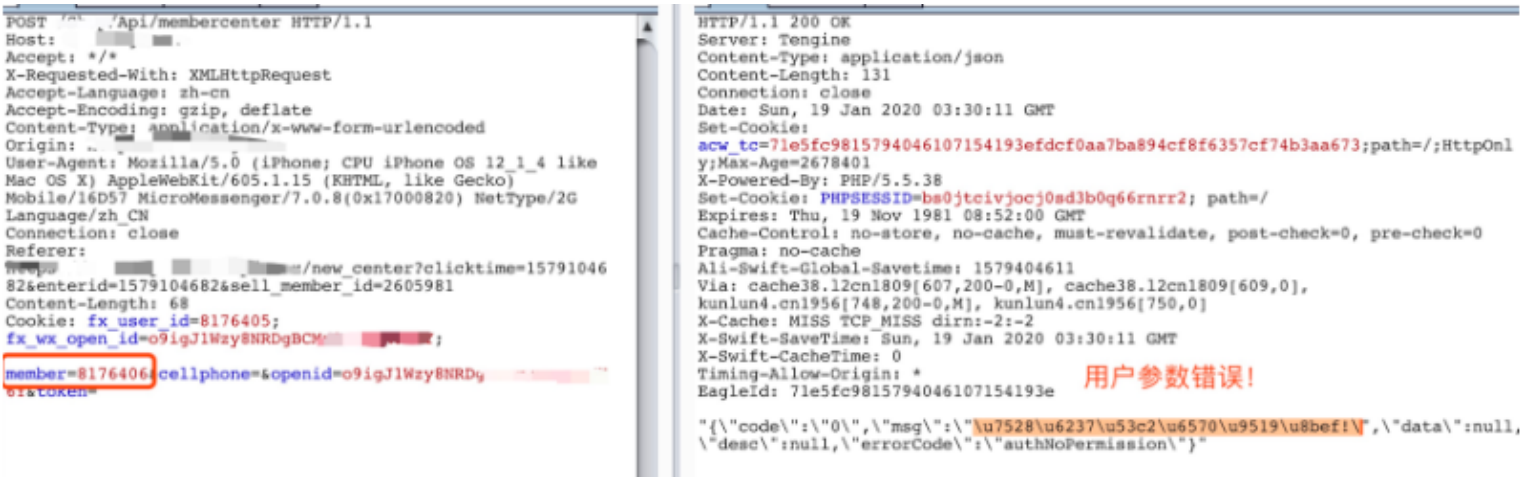
2.第二个案例来源于微信公众号，在测试微信公众号时，相信大家对openid并不陌生，openid是用户在当前公众号下的唯一标识（‘身份证’），就是说通过这个openid，就能区分在这个公众号下具体是哪个用户，若是公众号服务端只根据openid来进行用户的身份校验，则类似于会话固定，因为用户授权给微信公众号，openid是固定不变的，一旦泄露，就意味着用户的所有信息及操作可被盗用。在测试某公众号商城时，又是不经意间发现每个商品的用户评价处，返回的用户头像地址中，存在以微信的openid命名的图片名称。



在后续的测试中发现，服务端根据fxwxopenid与fxuserid来识别用户，但fxwxopenid与fxuserid必须一一对应时才可进行用户相应操作。如下接口查看用户个人信息时，服务端会从session中获取当前用户的fxwxopenid与fxuserid值并赋值给要查询的参数，并对用户这两个参数的一致性进行校验，当校验成功后才会执行查询，若不一致则会提示用户参数错误：



当不一致时提示用户参数错误：



由于商品评价处泄露的用户的openid，现在只需要知道用户的userid值即可完全代替用户进行任意操作，虽然可以通过爆破用户的userid去跟openid一一对应，但是userid值巨大，爆破难度很高。

在fuzz大法尝试获取用户名接口也未能奏效后，场面一度陷入僵局，我甚至想到了放弃来掩饰菜的尴尬。

正当我准备退出登录时，看见了联系客服按钮，想到在客户获取会话时，一般会获取当前用户的id或用户名信息，点击后果然看到了一处可以根据userid来查询用户昵称的接口，这可能就是传说中的皂滑弄人吧。

```
POST [redacted] /Api/get_username HTTP/1.1
Host: [redacted]
Accept: */*
X-Requested-With: XMLHttpRequest
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
Content-Type:
application/x-www-form-urlencoded;
charset=UTF-8
Origin: https://[redacted]
User-Agent: Mozilla/5.0 (iPhone; CPU
iPhone OS 12_2 like Mac OS X)
AppleWebKit/605.1.15 (KHTML, like Gecko)
Mobile/15E148
MicroMessenger/7.0.8(0x17000820)
NetType/WIFI Language/zh_CN
Connection: close
Referer:
https://[redacted]_detail?
or_id=4067436&supply_invoice_need_order_id=
4067865&sell_member_id=2605336
Content-Length: 17
Cookie:

member_id=8471211
```

```
HTTP/1.1 200 OK
Server: Tengine
Content-Type: application/json
Content-Length: 41
Connection: close
Date: Thu, 16 Jan 2020 06:14:36 GMT
Set-Cookie:
acw_tc=d38ec4a815791552759413789ed751d670b6c3e1933d2723036004dba0;
path=/;HttpOnly;Max-Age=2678401
X-Powered-By: PHP/5.5.38
Set-Cookie: PHPSESSID=4ksj85blnjnffsdf916pe5pgm5; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0,
pre-check=0
Pragma: no-cache
Ali-Swift-Global-Savetime: 1579155276
Via: cache2.l2cn1809[93,200-0,M], cache2.l2cn1809[93,0],
kunlun10.cn256[182,200-0,M], kunlun10.cn256[184,0]
X-Cache: MISS TCP_MISS dirn:-2:-2
X-Swift-SaveTime: Thu, 16 Jan 2020 06:14:36 GMT
X-Swift-CacheTime: 0
Timing-Allow-Origin: *
EagleId: d38ec4a815791552759413789e

[{"nickname":"\u81ea\u5728\u72ec\u884c"}]
```

因为在商品评价处，用户昵称也会进行展示，所以只需要进行遍历用户的id来查询用户昵称并保存下来，通过用户昵称将userid值与openid值关联起来，就可以获得用户的身份令牌为了验证漏洞存在，写了个小脚本遍历了一小部分用户名跟id值。

最后只筛选了三个用户的openid跟用户userid以及昵称作为确认漏洞存在，而当把用户名称全部遍历出来，再与商品评价处的用户userid以及openid一一对应，便可获取大量用户的身份令牌。

5 总结

本文更多的是针对于业务逻辑测试点的一些思维分享，不能局限束缚在这些测试点中。熟悉相应系统业务流程，虽然会花费一定时间，但磨刀不误砍柴工，业务逻辑方面的漏洞最重要的是在熟悉业务流程的基础上，发散思维，能够比开发多想一层，那漏洞便会离你更近一分。

一名优秀的渗透测试工程师，既要武能漫游内网拿域控，又可以文能熟悉业务找逻辑。文武兼备，刚柔并济，祝大家早日笑傲江湖。