PHP代码加密技术

—@swoole郭新华

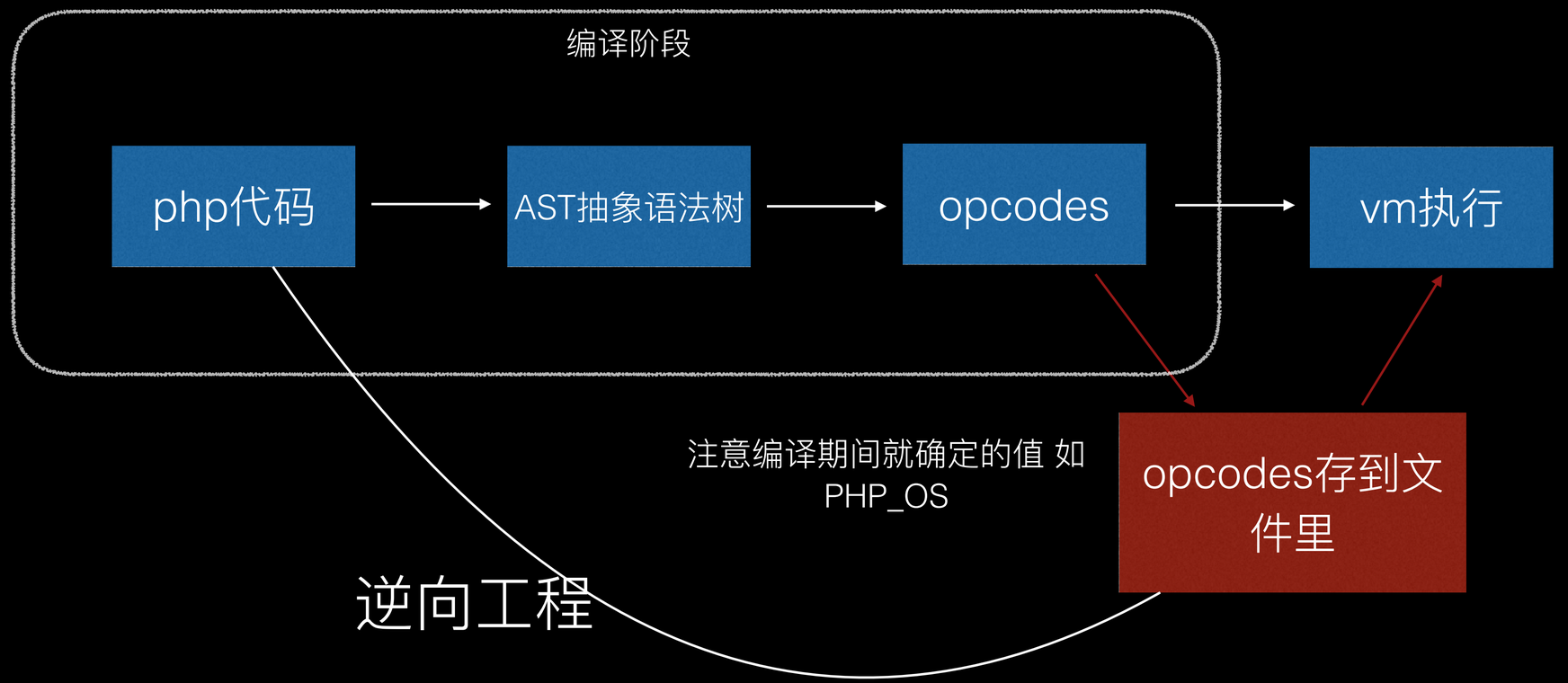
关于我

- pecl官方开发组成员。
- swoole-src项目核心成员。
- 现任车轮互联架构师。
- php-cp、swoole-mysql-proxy、swoole-serialize、swoole-compiler等项目作者。

为什么要加密

- 增强应用的安全性
- 保护知识产权
- 外包公司的延续性
- etc…

一个简单的加密系统


编译阶段
php代码
AST抽象语法树
opcodes
vm执行
注意编译期间就确定的值 如
PHP_OS
opcodes存到文件里
逆向工程

逆向例子

```php
define("PI", 3.1415926);

/**
 * @desc 计算圆周长
 * @param int 半径
 * @return float 周长
 */
function calc_circum($radius) {

    $circum = 2 * PI * $radius;
    return $circum;
}
```

```
function name:  calc_circum
compiled vars:  !0 = $radius, !1 = $circum
-------------------------------------------------------------------
0         RECV                                    !0
1         FETCH_CONSTANT                          ~2      'PI'
2         MUL                                     ~3      2, ~2
3         MUL                                     !1      ~3, !0
4         RETURN                                  !1
```

核心—防止逆向

世界上没有破解不了的软件，只有不值得破解的软件。

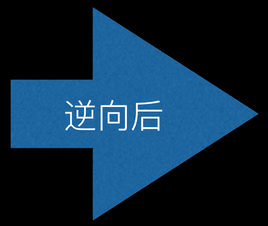
- 提高逆向成本(对黑客)
- 逆向后不可读(对读者)

逆向后不可读—剔除注释

```php
define("PI", 3.1415926);

/**
 * @desc 计算圆周长
 * @param int 半径
 * @return float 周长
 */
function calc_circum($radius) {

    $circum = 2 * PI * $radius;
    return $circum;
}
```




```php
define("PI", 3.1415926);

function calc_circum($radius) {

    $circum = 2 * PI * $radius;
    return $circum;
}
```

函数编译后的结构体

```
struct _zend_op_array {
    ...
    zend_string *doc_comment;
    ...
};
```

- 注意$reflection->getDocComment ()

逆向后不可读—混淆局部变量

```php
define("PI", 3.1415926);

function calc_circum($radius) {

    $circum = 2 * PI * $radius;
    return $circum;
}
```

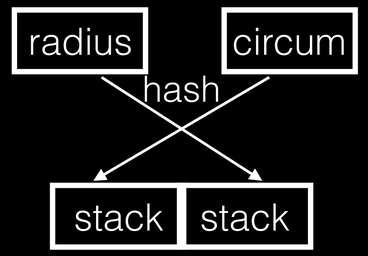
逆向后

```php
define("PI", 3.1415926);

function calc_circum($_423235211) {

    $_423235212 = 2 * PI * $_423235211;
    return $_423235212;
}
```

符号表



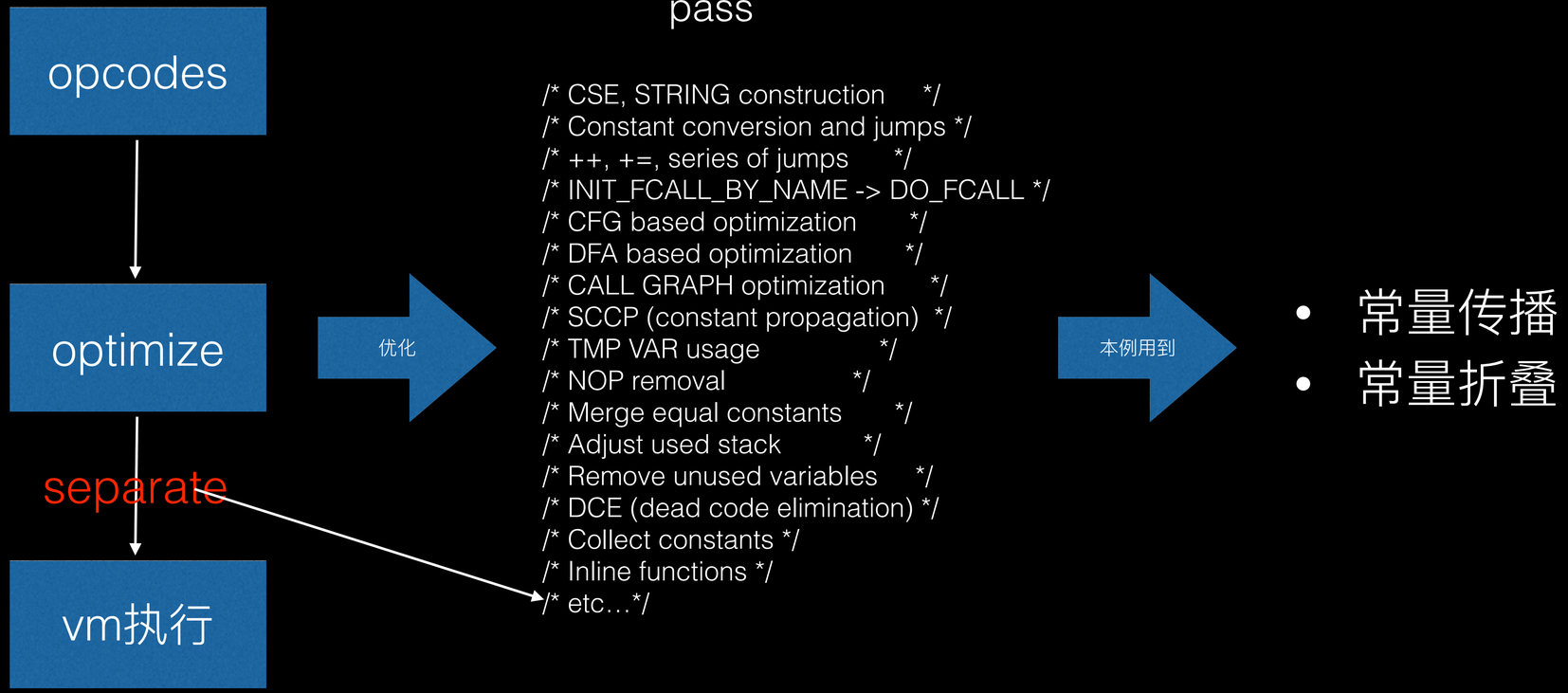
!0 => $radius
!1 => $circum

很多用符号表的情况

- $$var_name
- compact、extract等函数
- include进来的变量
- etc…

Notice: Undefined variable: $_432345234

逆向后不可读—编译优化


opcodes
optimize
separate
vm执行
优化
pass
/* CSE, STRING construction */
/* Constant conversion and jumps */
/* ++, +=, series of jumps */
/* INIT_FCALL_BY_NAME -> DO_FCALL */
/* CFG based optimization */
/* DFA based optimization */
/* CALL GRAPH optimization */
/* SCCP (constant propagation) */
/* TMP VAR usage */
/* NOP removal */
/* Merge equal constants */
/* Adjust used stack */
/* Remove unused variables */
/* DCE (dead code elimination) */
/* Collect constants */
/* Inline functions */
/* etc…*/
本例用到
• 常量传播
• 常量折叠

编译优化-例子

```php
define("PI", 3.1415926);

function calc_circum($_423235211) {

    $_423235212 = 2 * PI * $_423235211;
    return $_423235212;
}
```

常量传播

```php
function calc_circum($_423235211) {

    $_423235212 = 2 * 3.1415926 * $_423235211;
    return $_423235212;
}
```

常量折叠

```php
function calc_circum($_423235211) {

    $_423235212 = 6.28319 * $_423235211;
    return $_423235212;
}
```

- need to migrate from php72

逆向后不可读—什么是内联

```
function calc_circum($_423235211) {

    $_423235212 = 6.28319 * $_423235211;
    return $_423235212;
}

function calc(){
   echo calc_circum(1024);
}
```

内联函数步骤

步骤1过滤

- 函数有静态变量
- 扩展实现的函数
- 递归调用自己
- 函数体过大
- 函数多态
- etc…

步骤2处理指令

- recv、send、return init fcall、do fcall
- etc…

步骤3合并

- copy字面量、变量等信息
- 合并opcodes
- etc…

步骤4调整

- 修复jmp、jmpz、jmpnz等指令的偏移量
- etc…

最终的结果

```php
function calc_circum($_423235211) {

    $_423235212 = 6.28319 * $_423235211;
    return $_423235212;
}

function calc(){
   echo calc_circum(1024);
}
```

注入指令(内联)

优化

```
function name:  calc
----------------------------------------
1         ECHO                    6433.99
2       > RETURN                  null
```

```php
function calc(){
    echo 6433.98;
    return;
}
```

代码体积变大
丢失堆栈
修改文件需要重新编译整个项目

逆向后不可读—基于llvm编译成bitcode指令

——————————-分割线——————————

垃圾代码

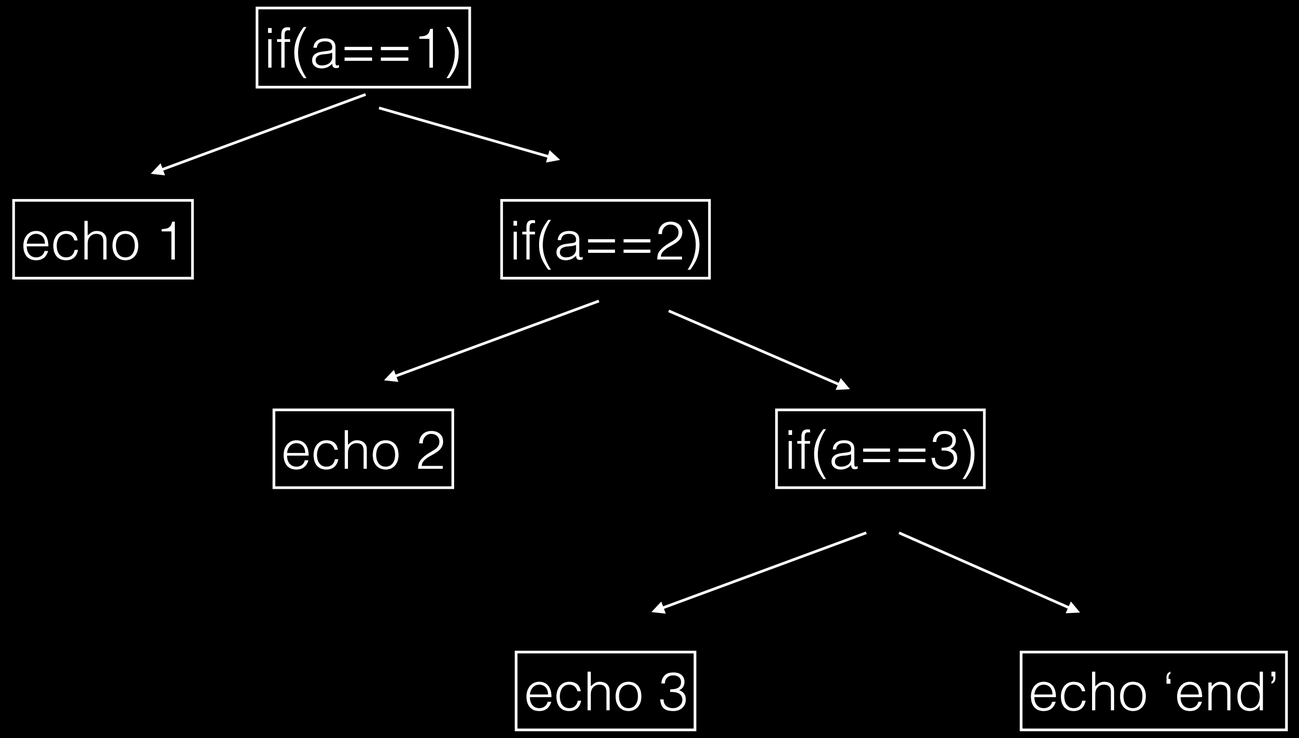
```php
function calc(){
    echo 6433.98;
    return;
}
```

```php
function calc(){
    //垃圾指令
    $_423235211 = 6433.98;
    if($_423235211){
        echo 6433.98;
    }
    return;
}
```

不能太多

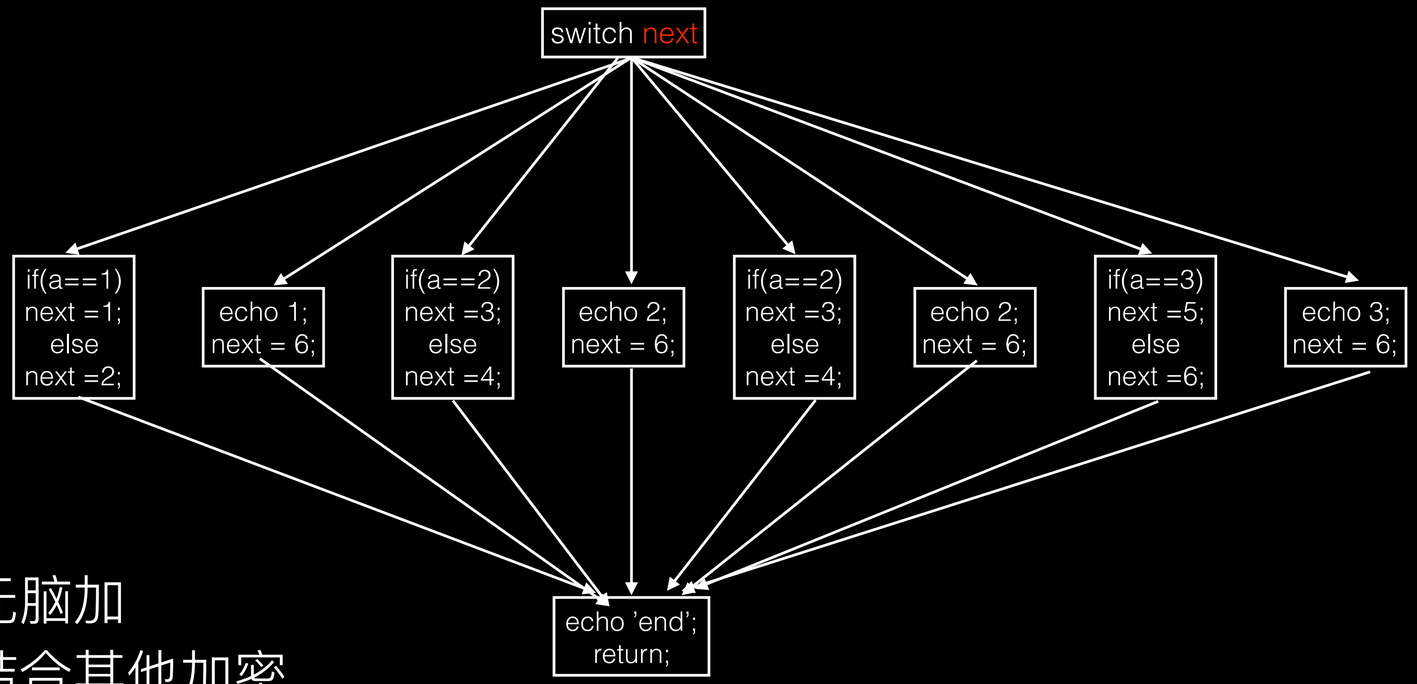
扁平化控制流

```php
if($a==1){
    echo 1;
}else if($a==2){
    echo 2;
}else if($a==3){
    echo 3;
}
echo 'end';
```



扁平化控制流

```php
$next = 0;
while (1) {
    switch ($next) {
        case 0:if ($a == 1) $next = 1; else $next = 2;break;
        case 1:echo 1;$next = 6;break;
        case 2:if ($a == 2) $next = 3; else $next = 4;break;
        case 3:echo 2;$next = 6;break;
        case 4:if ($a == 3) $next = 5; else $next = 6;break;
        case 5:echo 3;$next = 6;break;
        case 6:echo "end";return;
    }
}
```



不能无脑加

需要结合其他加密

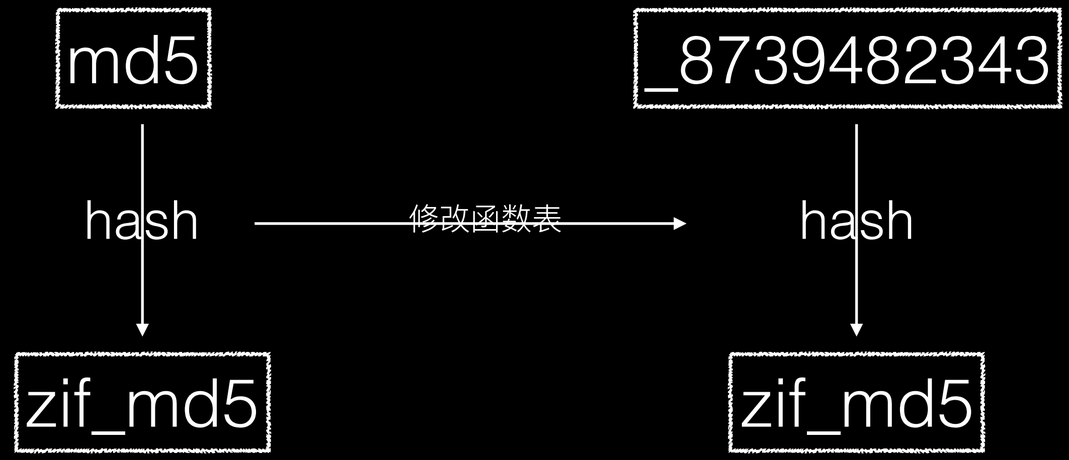
内置函数名替换

```
echo md5('test');
```

修改字面量

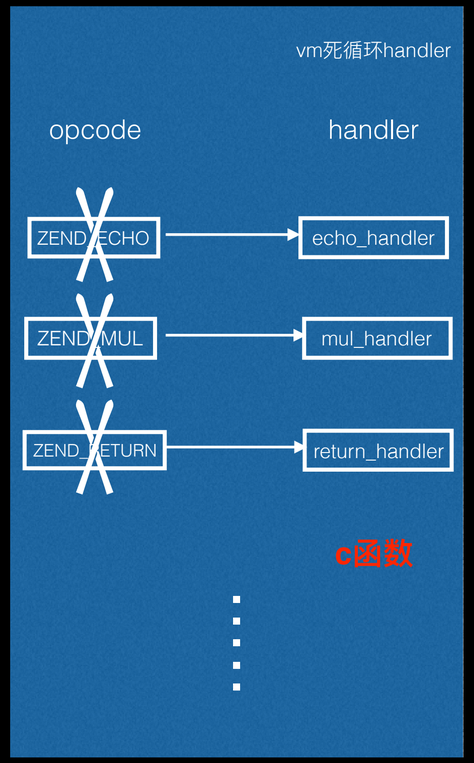
```
echo _8739482343('test');
```

函数表



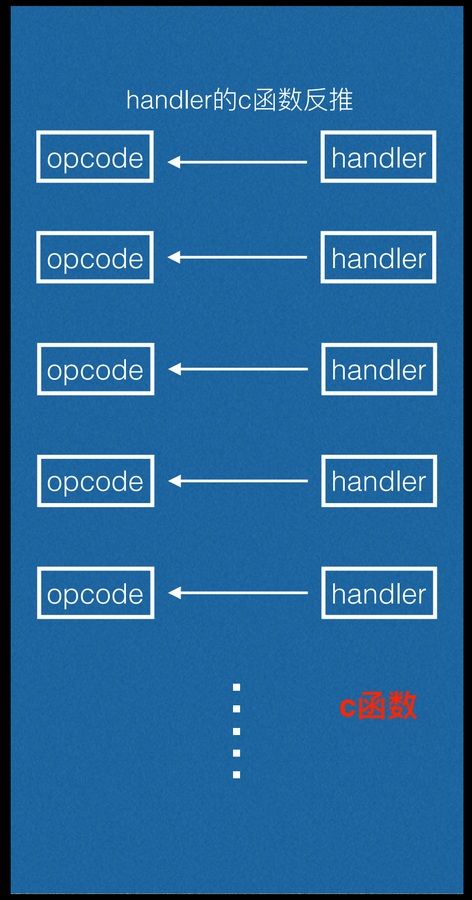
缺点：报错堆栈里面的函数名是乱码

删除/混淆opcode 只保留handler



有些opcode不能删除/混淆

根据handler反推opcode



虚拟机保护技术(vmp)

- 接管zend_vm
- 自定义指令集
- 自定义数据结构
- 自己实现对应的handler

花指令

```php
<?php

echo 1;
echo 2;
echo 3;
echo 4;
```

```
----------------------------------------
0      ECHO                    1
1      ECHO                    2
2      JMPZ                    !0($a), ->4
3      ECHO                    3
4      ECHO                    4
5      RETURN                  1
```

my_jmpz_handler

```php
echo 1;
echo 2;
if($a){
  echo 3;
}
echo 4;
```

运行时解密

加密字面量

```php
<?php

echo "hello world!";
```

加密

```php
echo "LNKEJOINLZNSD";
```

my_echo_handler

```php
echo "LNKEJOINLZNSD";
```

输出前解密

```php
echo "hello world!";
```

输出后加密

```php
echo "LNKEJOINLZNSD";
```

不能太多

其他

- 反调试
- 防篡改
- 预防破解
- 判断指针是否被拦截
- 加壳

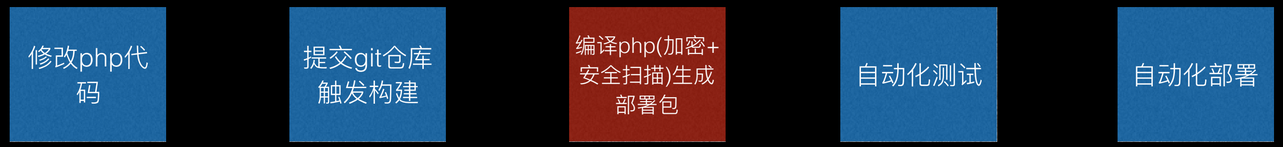
总结

- 报错信息和加密选择后者
- 增加破解难度的空间是无限的
  最难的是加密和性能的权衡

swoole compiler定位—----应用安全工具

代码加密+数据加密+源码漏洞扫描

集成swoole compiler到发布系统



thanks