Darkness/Airsupply

X'CON2006

目录

- 关于PHP
- 深入了解PHP
- Decode的关键---Opcode
- Opcode Hooker技术介绍
- 分析Zend Optimizer
- 开始Decode
- 饶过混淆技术

About PHP

- 简约不简单
- 跨平台
- 使用广泛

X'con 2006

X'con 2006

Inside PHP

- Php core 与 ZendCore

Web Server Software

php 4

Modules

MySQL | Oracle | ODBC | XML | IMAP | LDAP | Java | COM | More Modules | ...

Database

Zend Engine

Browser

WEB SERVER INTERFACE

FUNCTION MODULES INTERFACE

EXECUTOR

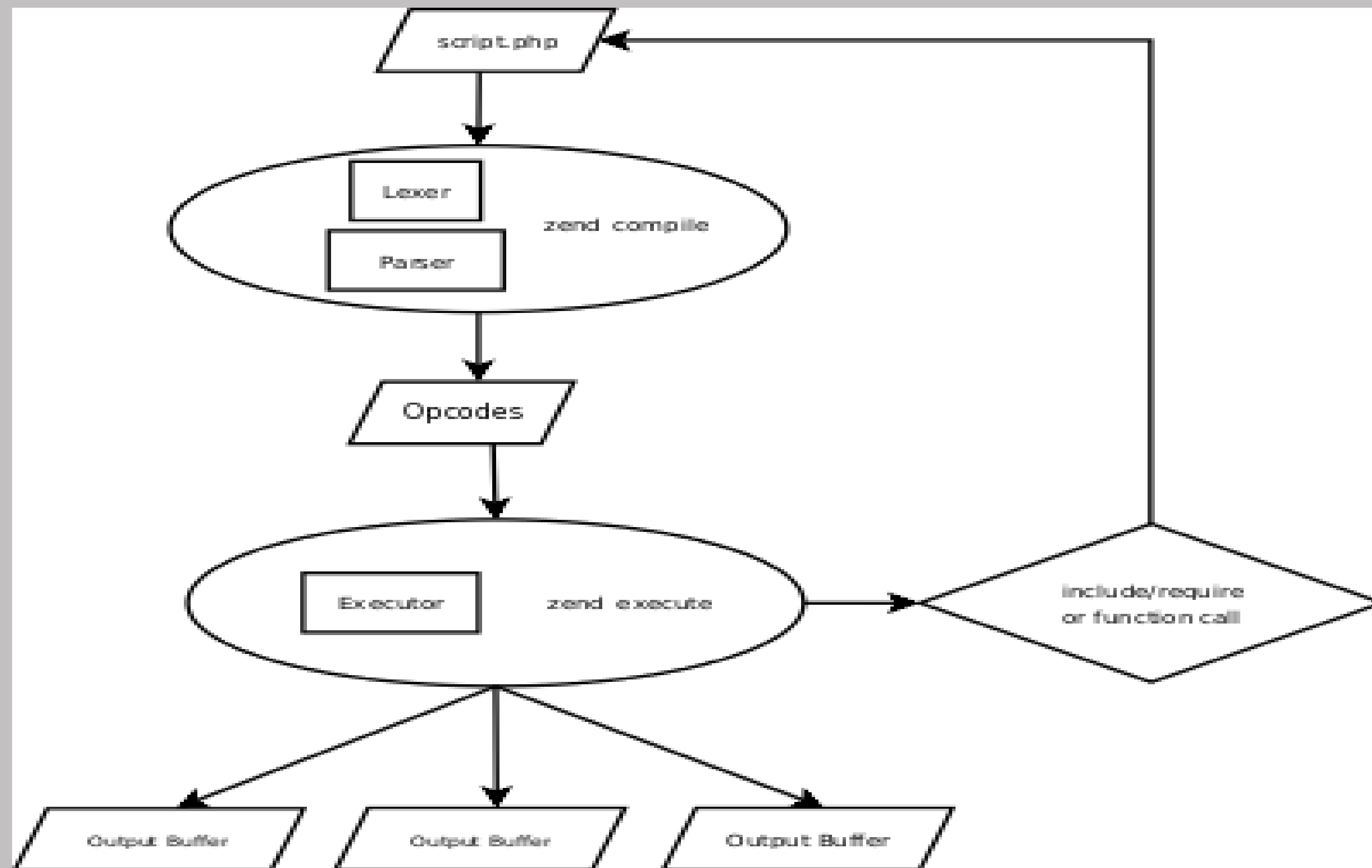
RUN-TIME COMPILER

Web Server Disk

1 2 3 4 5 6

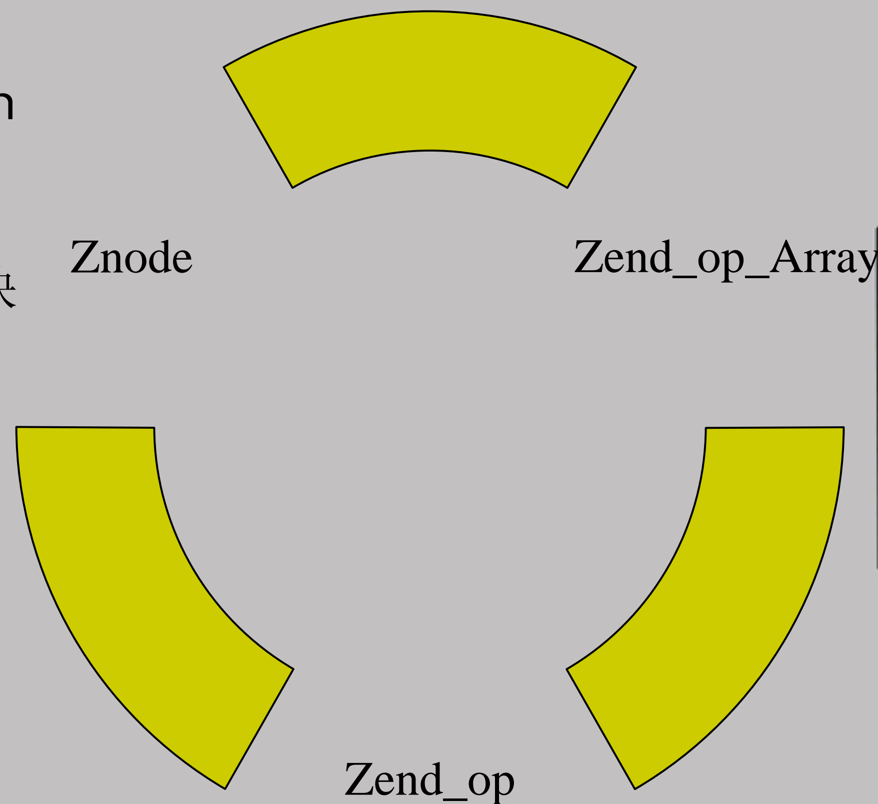
1、把原始的php脚本生成中间代码（op-codes）放在op-arrays里面--对应下图的zend_compile

2、在虚拟机上面执行中间代码（Zend引擎就是个虚拟机）--对应下图的zend_execute

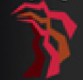


Opcode相应结构

- Opcode由Zend_op_array,Znode,Zend_op3个结构体构成.
- Zend_op_array被ZEND编译PHP文件后传入Zend执行模块中



X'con 2006



```
struct _zend_op_array {
    zend_uchar type;  /* MUST be the first element of this struct! */
    zend_uchar *arg_types;          /* MUST be the second element of
    this struct! */
    char *function_name;        //调用自定义的函数名字
    zend_uint *refcount;
    zend_op *opcodes;   //参见下面
    zend_uint last, size; //最后一个opcode number，即opcode number合 /
    总数.
    zend_uint T;
    zend_brk_cont_element *brk_cont_array;
    zend_uint last_brk_cont;
    zend_uint current_brk_cont;
    zend_bool uses_globals;
    /* static variables support */
    HashTable *static_variables;
    zend_op *start_op;
    int backpatch_count;
    zend_bool return_reference;
    zend_bool done_pass_two;
    char *filename; //文件名
    void *reserved[ZEND_MAX_RESERVED_RESOURCES];
};
```

Zend_op_oparray结构

Zend_op

```
typedef struct _zend_op {
    zend_uchar opcode; //opcode进行什么操作的值。比如为1，进行一个加法操作.在zend_compile.h里面有完整的定义.
    znode result;
    znode op1; //PHP4的opcode
    znode op2; //PHP5 的opcode
    ulong extended_value;
    uint lineno;
} zend_op;
```

Zend_op 结构

```
typedef struct _znode {
    int op_type; //分为Const/tmp_var/var等
    union {
     zval constant; // 比如说$var的直.zval包括所有php的类型.
     zend_uint var;
     zend_uint opline_num; /*  Needs to be signed */
     zend_uint fetch_type;   //Global 变量与Static变量
     zend_op_array *op_array;
     struct {
          zend_uint var;
          zend_uint type;  //
     } EA;
    } u;
} znode;
```

Znode结构

Opcode的执行

PHP源代码转成opcode后，Zend_Execute_scripts负责执行php脚本.而Zend_Execute_Scripts又由zend api Zend_execute来具体操作opcode.

大概流程如下.

```
switch(EG(active_op_array->opcodes))
/*active_op_array为当前活动的opcode.即Zend_Op_array结构.*/
{
    case Zend_ADD:
        do something;
         go to ZendADD;
    case Zend_SUB:
       do something;
      go to Zend_SUB;
          .....
}
```

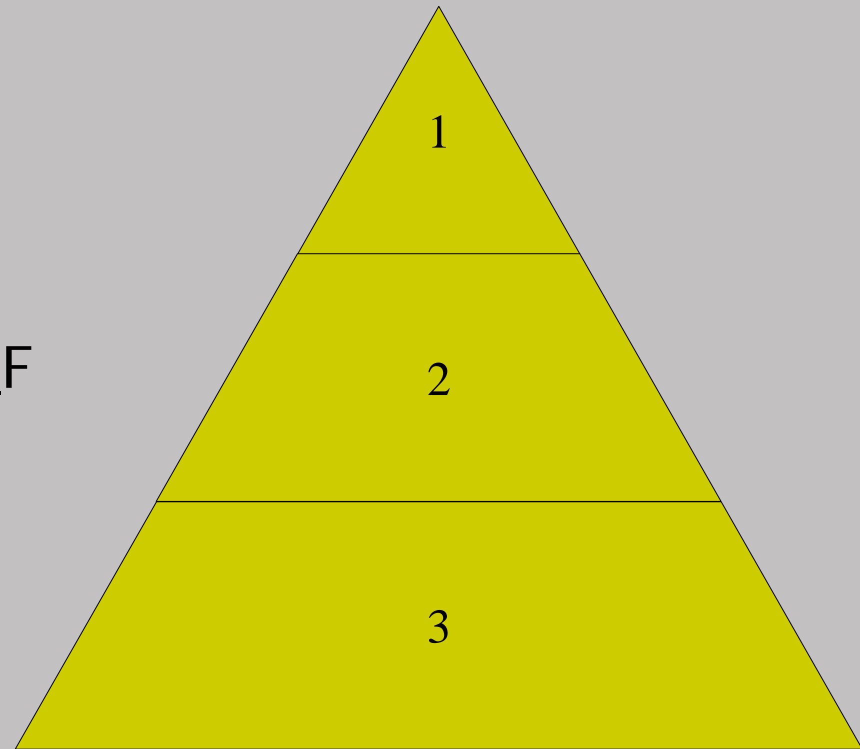
Opcode Hooker

1. 直接修改Zend源代码,在Zend_Execute_Scripts或者Api Zend_Execute处建立自己的HOOK代码 (推荐此方法)
2. 通过PHP扩展建立一个附加Zend_Exerute,使之具有与Zend_Execute一样的功能. 并能接受到Opcode ,Vld(Vulcan Logic Dissassembler)就是这么一个Opcode Hooker.

扩展型Opcode Hooker原理

PHP扩展内核启动优先级别:

1. PHP_MINIT_FUNCTION
2. PHP_RINIT_FUNCTION
3. PHP_*SHUTDOWN_FUNCTION



Zend Guard(Zend Encoder)

PHP官方支持公司出品。具有权威性。占PHP代码加密市场的97%份额。能运行在Zend Optimizer(号称能加速PHP运行的PHP/Zend扩展)上.

介绍:

Zend Guard, formerly known as Zend Guard, protects your commercial PHP 4 and PHP 5 applications from reverse engineering, unauthorized customization, unlicensed use and redistribution.

Software vendors are increasingly writing applications in PHP with the aim of distributing them via download or CD. It is critical that the source code and intellectual property of the applications being distributed is secure, regardless of whether the applications are free, for evaluation purposes or for commercial sale.

The Zend Guard, with its key components of Encoding, Obfuscating and Licensing, make this distribution worry free.

Zend Guard, likes its predecessor Zend Guard, allows Independent Software Vendors (ISVs) and IT managers to safely and confidently distribute and manage the deployment of their PHP applications while protecting their source code.

Zend Guard 4 not only encodes the source code of your application, but also increases source code protection through obfuscation of the various application name components.

Zend Guard 4 is the only product that obfuscates object oriented programs created with PHP 4 and PHP 5.

Zend Optimizer

- ***Zend Optimier*也是一个*HOOK***
  - 他***hook***并替换了原来***Zend***中扫描，编译，执行的一些关键部分。
  - 万幸的是他劫持了 ***zend_ execute,***而没有劫持***zend_execute_scripts***的***.***
  - 这就是***PHP***扩展我不推荐使用的原因

PHP Scripts

zend_loader_file_encoded

zend_current_obfuscation_level

zend_obfuscate_function_name

Zend Optimizer 代码解析模块
(原Zend_Compile)

Level 0

Zend Optimizer 执行模块
(原Zend_Execute)

Vld

看看vld是如何工作的:

```
PHP_RINIT_FUNCTION(vld)
{
        if (VLD_G(active)) {
zend_compile_file = vld_compile_file;
                if (!VLD_G(execute)) {
                        zend_execute       = vld_execute;
                }
        }
        return SUCCESS;
}
```

vld对未encode文件的处理

对于没有使用Zend Guard加密过的代码，vld是能够很好的把opcode还原出来的：

```
 cat /home/mainframe/zend/php/org.php/1.php
<?
$a='test';
print $a;
?>

sapi/cli/php -d vld.active=1
    /home/mainframe/zend/php/org.php/1.php
filename:       /home/mainframe/zend/php/org.php/1.php
function name:  (null)
number of ops:  6
line     #  op                           fetch          ext  operands
-------------------------------------------------------------------------------
  19     0  NOP
         1  ASSIGN                                                 , 'test'
  20     2  NOP
         3  PRINT                                               ~3,
         4  FREE                                                    ~3
  22     5  RETURN                                                   1
```

Vld Vs Encoded

vld hook了compile的过程，所以在生成opcode的时候能够进行处理。但是对于加密过的代码就无能为力了，因为Zend Optimier也hook了这些函数来实现解密的，所以vld能看到的还是加密了的代码。

```
 sapi/cli/php -d vld.active=1 dst.php/1.php
filename:       /home/mainframe/php-4.4.2/dst.php/1.php
function name:  (null)
number of ops:  697
line     #  op                           fetch          ext  operands
-------------------------------------------------------------------------------
   1     0  BEGIN_SILENCE                                    ~0
         1  FETCH_CONSTANT                                   ~1, 'Zend'
         2  END_SILENCE                                      ~0
         3  FREE                                             ~1
   5     4  INIT_STRING                                      ~2
  ..................
```

Api Zend_Execute_Scripts

```
ZEND_API int zend_execute_scripts(int type TSRMLS_DC, zval **retval,
    int file_count, ...)
    {
......
       for (i=0; i<file_count; i++) {
              .......
              }
              EG(active_op_array) = zend_compile_file(file_handle,
    ZEND_INCLUDE TSRMLS_CC);
              zend_destroy_file_handle(file_handle TSRMLS_CC);

              if (EG(active_op_array)) {
                     EG(return_value_ptr_ptr) = retval ? retval :
    &local_retval;
                     zend_execute(EG(active_op_array) TSRMLS_CC);
                     ......
       }
       va_end(files);
       EG(active_op_array) = orig_op_array;
       return SUCCESS;
}
```

X'con 2006

把vld中的vld_dump_oparray(EG(active_op_array));插入到上述的函数里面，如下：

```
EG(active_op_array) = zend_compile_file(file_handle, ZEND_INCLUDE TSRMLS_CC);
            zend_destroy_file_handle(file_handle TSRMLS_CC);
            vld_dump_oparray(EG(active_op_array));
            if (EG(active_op_array)) {
                  EG(return_value_ptr_ptr) = retval ? retval : &local_retval;
                  zend_execute(EG(active_op_array) TSRMLS_CC);
```

然后重新编译后执行：

```
mainframe@(none) ~/php-4.4.2 $ sapi/cli/php dst.php/2.php
filename:       /home/mainframe/php-4.4.2/dst.php/2.php
function name:  (null)
number of ops:  2
line     #  op                           fetch          ext  operands
-------------------------------------------------------------------------------
   2     0  SUB                                                'aaaa'
   4     1  BW_XOR                                               1
```

原始的结果：

```
mainframe@(none) ~/php-4.4.2 $ sapi/cli/php org.php/2.php
filename:       /home/mainframe/php-4.4.2/org.php/2.php
function name:  (null)
number of ops:  2
line     #  op                           fetch          ext  operands
-------------------------------------------------------------------------------
   2     0  ECHO                                               'aaaa'
   4     1  RETURN                                              1
```

实际内容：

```
<?php
echo "aaaa";
?>
```

上面的结果证实了前面的想法，确实可以从这里突破进行还原。

深入

从前面的结果我们发现vld实现的还不是很准确，op类型对应的不对，需要进行修复。同时，在输出的时候，vld只是输出简单的结果，我们必须把opcode转化成php代码的形式，比如ECHO类型的就是echo，SUB就是-，等等，最后以完整的php代码形式输出。

```
sapi/cli/php  dst.php/2.php
0:<40> ZEND_ECHO                    ->0, 'aaaa', ->0
1:<62> ZEND_RETURN                   ->0, 1, ->0
<?
  echo 'aaaa';
?>
```

混淆技术

最近新出的Zend Guard4使用了混淆技术，我们看下实际的情况。

```
sapi/cli/php  dst.php/3.php
        0: <59> ZEND_INIT_FCALL_BY_NAME            ->0,
    ->0, 'kg-yo'
        1: <61> ZEND_DO_FCALL_BY_NAME              $0,
    'kg-yo', ->0
        2: <62> ZEND_RETURN                    ->0, 1, ->0
<?
  kg-yo ();
?>
```

实际的代码是：

```
<?php
phpinfo();
?>
```

饶过混淆

- 把混淆算法逆向(我不是王小云!!!)
- 利用 zend_obfuscate_function_name预先把所有函数生成对应表

利用zend_obfuscate_function_name输出所有混淆过的php standard模块函数例子:

```
<?

$module = 'standard';
$functions = get_extension_funcs("standard");

foreach($functions as $func) {
    printf( "%s();\r\n“,
zend_obfuscate_function_name($func));

}
?>
```

新的困难

自定义函数和类被也被混淆!!!

```
<?
  function o}xm ()
  {
    print 'aaa';
  }
  phpinfo ();
  echo 'aaaa';
  o}xm ();
?>
```

原始代码是：

```
<?
  function abcd ()
  {
    print 'aaa';
  }
  phpinfo ();
  echo 'aaaa';
  abcd ();
?>
```

解决方法

用户自定义函数没有对应表，没有办法使用前面的方法来修复。我们可能有三种方法

- 生成完php代码后，手工编辑把自定义的函数修复回来
  (因为是自定义的，手工修改不影响执行)
- 动态修复
  (遇到用户自定义的函数，动态修改成一个随件函数，然后把结果保存到一张临时对应表中，再下一次遇到是就可以查询临时表来统一结果。)
- 利用zend_obfuscate_function_name 来穷举
  (推荐完美主义者使用)

总结

- Api Zend_execute_scripts没有被Hook.

  (目前看来Zend_execute_scripts不会被Zend公司Hook,除非PHP不开源,或者重写Zend代码)

- Optimizer脆弱的验证机制

  (无疑Optimizer的加密算法是强壮的,但是我们可以在他解密后小作修改进行还原)

参考

Any Question?

参考:

Http://lxr.php.net

http://www.derickrethans.nl/vld.php

http://cn.php.net/manual/zh/zend.php

感谢

要是没有B105 Bar那8折的啤酒,
我们恐怕难以写出此议题

最后欢迎到:

irc.0x557.net: 9940 (SSL)

Channel: #segfault

技术交流