RIPS源码精读(二):扫描对象的实例化及token信息的生成

梅子酒m3i / 2018-08-21 02:38:20 / 浏览数 3313 技术文章 技术文章 顶(0) 踩(0)

引言

在main.php的171行附近,rips对Scanner类进行了实例化，并由此进入正式的分析流程.

内容简介

阅读rips关于token分析处理相关的源码,并分析对应的用途及处理逻辑.

Scanner类

首先是调用Scanner类的构造函数

```
$scan = new Scanner($file_scanning, $scan_functions, $info_functions, $source_functions);
```

各参数说明如下:

```
$file_scanning:■■■■■■■■■■

$scan_functions:■■■■■■■■■■■ main.php ■ $_POST['vector'] ■■■■

$info_functions:■ main.php ■ Info::$F_INTEREST ■■,■■■■■■■■■■■■

$source_functions:■ main.php ■ Sources::$F_OTHER_INPUT ■■,■■■■■■■■■■■■
```

Scanner类构造函数分析

Scanner构造函数定义如下:

```
function __construct($file_name, $scan_functions, $info_functions, $source_functions)
```

首先是大量的变量初始赋值:

```
//■■■■■■■■■■
       $this->file_name = $file_name;
       $this->scan_functions = $scan_functions;
       $this->info_functions = $info_functions;
       $this->source_functions = $source_functions;


       //......
```

其中夹杂着Analyzer类的初始化,用于获取php的include_path配置

```
$this->include_paths = Analyzer::get_ini_paths(ini_get("include_path"));
```

紧接着便是根据文件生成token信息

```
$tokenizer = new Tokenizer($this->file_pointer);
$this->tokens = $tokenizer->tokenize(implode('',$this->lines_pointer));
unset($tokenizer);
```

在讲这几行作用之前,要先了解token_get_all函数

token_get_all()函数简单介绍

php手册说明如下

> token_get_all() 解析提供的 source 源码字符，然后使用 Zend 引擎的语法分析器获取源码中的 PHP 语言的解析器代号

函数定义

> array token_get_all (string $source)

示例代码

```
<?php echo 123;>
```

token_get_all()处理语句

```
token_get_all("<?php echo 123;>");
```

处理结果

```
Array
(
   [0] => Array
       (
           [0] => 376
           [1] => <?php
           [2] => 1
       )

   [1] => Array
       (
           [0] => 319
           [1] => echo
           [2] => 1
       )

   [2] => Array
       (
           [0] => 379
           [1] =>
           [2] => 1
       )

   [3] => Array
       (
           [0] => 308
           [1] => 123
           [2] => 1
       )

   [4] => ;
   [5] => Array
       (
           [0] => 378
           [1] => ?>
           [2] => 1
       )

)
```

可以看到,代码被分割成了五段,其中除了第四段之外,每一段都分为三段.

我们设$token=token_get_all(....),那么$token[0]便对应着

```
Array
       (
           [0] => 376
           [1] => <?php
           [2] => 1
       )
```

则$token[0][1]对应<?php

那么下一个问题便是$token[0]对应数组中的三个值,分别代表什么意思,解释如下:

```
Array
       (
           [0] => 376  // token■■
           [1] => <?php  // ■■■■
           [2] => 1  // ■■
       )
```

我们可以使用token_name获得索引所对应的字面常量

```
echo token_name(376);

//result => T_OPEN_TAG

echo token_name(319);

//result => T_ECHO

echo token_name(308);

//result => T_LNUMBER

echo token_name(378)

//result => T_CLOSE_TAG
```

以上便是对token_get_all函数大致介绍

Scanner类中token信息生成分析

回到生成token信息的这几句

```
$tokenizer = new Tokenizer($this->file_pointer);
$this->tokens = $tokenizer->tokenize(implode('',$this->lines_pointer));
unset($tokenizer);
```

```php
function __construct($filename){
   $this->filename = $filename;
}
```

接下来调用tokenize函数,跟进

```
public function tokenize($code){
   $this->tokens = token_get_all($code);
   $this->prepare_tokens();
   $this->array_reconstruct_tokens();
   $this->fix_tokens();
   $this->fix_ternary();
   #die(print_r($this->tokens));
   return $this->tokens;
}
```

可以看出在tokenize调用了多个token分析相关的函数,完成token分析准备、重构等工作

prepare_token()函数分析

跟进$this->prepare_tokens()

```
function prepare_tokens()
{

   for($i=0, $max=count($this->tokens); $i<$max; $i++)
   {
       if( is_array($this->tokens[$i]) )
       {
           if( in_array($this->tokens[$i][0], Tokens::$T_IGNORE) )
               unset($this->tokens[$i]);
           else if( $this->tokens[$i][0] === T_CLOSE_TAG )
               $this->tokens[$i] = ';';
           else if( $this->tokens[$i][0] === T_OPEN_TAG_WITH_ECHO )
               $this->tokens[$i][1] = 'echo';
       }

       else if($this->tokens[$i] === '@')
       {
           unset($this->tokens[$i]);
       }
```

```
        else if( $this->tokens[$i] === '{'
        && isset($this->tokens[$i-1]) && ((is_array($this->tokens[$i-1]) && $this->tokens[$i-1][0] === T_VARIABLE)
        || $this->tokens[$i-1] === ']') )
        {
            $this->tokens[$i] = '[';
            $f=1;
            while($this->tokens[$i+$f] !== '}')
            {
                $f++;
                if(!isset($this->tokens[$i+$f]))
                {
                    addError('Could not find closing brace of '.$this->tokens[$i-1][1].'{}.', array_slice($this->tokens, $i-1,
                    break;
                }
            }
            $this->tokens[$i+$f] = ']';
        }
    }

    // rearranged key index of tokens
    $this->tokens = array_values($this->tokens);
}
```

在prepare_token函数中,大体上是由一个for循环与return语句组成,for循环为prepare_token的主要功能

首先对每个token判断是否为数组,这一判断的依据我们在上面已经提到,随后进入in_array,在第一个in_array中,紧接着是第二个in_array,这一步的主要作用为,通过tok

与第二个in_array处于同一判断级别的条件为判断是否为php的开始(<?=)与闭合标签,若是,则替换为;或echo

与第一个in_array处于同一判断级别的另两个条件为

1. @符号
2. 该token信息为{,且下一token信息存在,■■token■■■■■, ■■token■■■■■■■或■■token■]

在该else if语句中,首先会将本次循环对应的token信息的{替换为[,接着便是while循环,寻找下一个闭合的}符号,如果寻找不到则执行addError

这个else if解释起来较为复杂繁琐,简单来讲便是将$array{xxx}格式的变量转换为$array[xxx]

接下来便是结束循环语句,执行return语句

总结一下prepare_token()函数功能:

去除无意义的符号,统一数组格式为$array[xxx]格式

array_reconstruct_tokens函数分析

在开始这里的分析前,我们先观察数组变量的token结构,php代码:

```
<?php

$array = array();
$array[0] = [1];
$array["meizj"] = ["mei"];
```

得到的token信息:

```
/Applications/MAMP/htdocs/aaa.php:18:
array (size=35)
 0 =>
   array (size=3)
     0 => int 376
     1 => string '<?php
' (length=6)
     2 => int 1
 1 =>
   array (size=3)
     0 => int 379
     1 => string '
' (length=1)
     2 => int 2
```

```
 2 =>
   array (size=3)
     0 => int 312
     1 => string '$array' (length=6)
     2 => int 3
 3 =>
   array (size=3)
     0 => int 379
     1 => string ' ' (length=1)
     2 => int 3
 4 => string '=' (length=1)
 5 =>
   array (size=3)
     0 => int 379
     1 => string ' ' (length=1)
     2 => int 3
 6 =>
   array (size=3)
     0 => int 366
     1 => string 'array' (length=5)
     2 => int 3
 7 => string '(' (length=1)
 8 => string ')' (length=1)
 9 => string ';' (length=1)
 10 =>
   array (size=3)
     0 => int 379
     1 => string '
' (length=1)
     2 => int 3
 11 =>
   array (size=3)
     0 => int 312
     1 => string '$array' (length=6)
     2 => int 4
 12 => string '[' (length=1)
 13 =>
   array (size=3)
     0 => int 308
     1 => string '0' (length=1)
     2 => int 4
 14 => string ']' (length=1)
 15 =>
   array (size=3)
     0 => int 379
     1 => string ' ' (length=1)
     2 => int 4
 16 => string '=' (length=1)
 17 =>
   array (size=3)
     0 => int 379
     1 => string ' ' (length=1)
     2 => int 4
 18 => string '[' (length=1)
 19 =>
   array (size=3)
     0 => int 308
     1 => string '1' (length=1)
     2 => int 4
 20 => string ']' (length=1)
 21 => string ';' (length=1)
 22 =>
   array (size=3)
     0 => int 379
     1 => string '
' (length=1)
     2 => int 4
 23 =>
   array (size=3)
```

```
    0 => int 312
    1 => string '$array' (length=6)
    2 => int 5
24 => string '[' (length=1)
25 =>
  array (size=3)
    0 => int 318
    1 => string '"meizj"' (length=7)
    2 => int 5
26 => string ']' (length=1)
27 =>
  array (size=3)
    0 => int 379
    1 => string ' ' (length=1)
    2 => int 5
28 => string '=' (length=1)
29 =>
  array (size=3)
    0 => int 379
    1 => string ' ' (length=1)
    2 => int 5
30 => string '[' (length=1)
31 =>
  array (size=3)
    0 => int 318
    1 => string '"mei"' (length=5)
    2 => int 5
32 => string ']' (length=1)
33 => string ';' (length=1)
34 =>
  array (size=3)
    0 => int 379
    1 => string '


' (length=3)
    2 => int 5
```

从第13行开始,出现的token索引为:

308 379 312 318

分别对应的token信息为:

T_LNUMBER:整型
T_WHITESPACE:空格
T_VARIABLE:变量
T_CONSTANT_ENCAPSED_STRING:字符串语法

因此,根据行数与对应token索引的值可以明白键值的类型是可以由T_CONSTANT_ENCAPSED_STRING以及T_LNUMBER来表示的.

有了这层基础,我们才能较好的去分析array_reconstruct_tokens

随后进入array_reconstruct_tokens函数,函数源码如下:

```
function array_reconstruct_tokens()
{
   for($i=0,$max=count($this->tokens); $i<$max; $i++)
   {

       if( is_array($this->tokens[$i]) && $this->tokens[$i][0] === T_VARIABLE && $this->tokens[$i+1] === '[' )
       {
           $this->tokens[$i][3] = array();
           $has_more_keys = true;
           $index = -1;
           $c=2;

           // loop until no more index found: array[1][2][3]
           while($has_more_keys && $index < MAX_ARRAY_KEYS)
           {
```

```
                $index++;

                if(($this->tokens[$i+$c][0] === T_CONSTANT_ENCAPSED_STRING || $this->tokens[$i+$c][0] === T_LNUMBER || $this->t
                {
                    unset($this->tokens[$i+$c-1]);
                    $this->tokens[$i][3][$index] = str_replace(array('"', "'"), '', $this->tokens[$i+$c][1]);
                    unset($this->tokens[$i+$c]);
                    unset($this->tokens[$i+$c+1]);
                    $c+=2;
                // save tokens of non-constant index as token-array for backtrace later
                } else
                {
                    $this->tokens[$i][3][$index] = array();
                    $newbraceopen = 1;
                    unset($this->tokens[$i+$c-1]);
                    while($newbraceopen !== 0)
                    {
                        if( $this->tokens[$i+$c] === '[' )
                        {
                            $newbraceopen++;
                        }
                        else if( $this->tokens[$i+$c] === ']' )
                        {
                            $newbraceopen--;
                        }
                        else
                        {
                            $this->tokens[$i][3][$index][] = $this->tokens[$i+$c];
                        }
                        unset($this->tokens[$i+$c]);
                        $c++;

                        if(!isset($this->tokens[$i+$c]))
                        {
                            addError('Could not find closing bracket of '.$this->tokens[$i][1].'[].', array_slice($this->tokens
                            break;
                        }
                    }
                    unset($this->tokens[$i+$c-1]);
                }
                if($this->tokens[$i+$c] !== '[')
                    $has_more_keys = false;
                $c++;
            }

            $i+=$c-1;
        }
    }
    $this->tokens = array_values($this->tokens);
}
```

■■```array_reconstruct_tokens```■■

■■■■■■■■```if```■■,■■■■■:

1. ■```token```■■■■■
2. ■```token```■■■■■■■■■
3. ■```token```■■■■```token```■■■```[

从这三个条件,我们可以很容易发现这是在寻找■■类型的变量,继续分析

在进入if语句后,将`$this->token[$i][3]`替换为了数组,随后又进行了三次赋值:

```
$has_more_keys = true;
$index = -1;
$c=2;
```

暂时不分析其各自含义,继续向下分析

接下来是一个`while`循环,判断条件有两个:

1. $has_more_keys是否为真
2. $index小于MAX_ARRAY_KEYS

两者需要同时满足,才进入while循环.跟踪MAX_ARRAY_KEYS常量,发现是类似于数组维数的变量,定义如下:

```
define('MAX_ARRAY_KEYS', 10);           // maximum array key $array[1][2][3][4]..
```

进入之后while循环,首先$index变量自增,随后是if语句,判断条件如下:

1. token索引的值需要为数组
2. token索引的值需要为T_CONSTANT_ENCAPSED_STRING,T_LNUMBER,T_NUM_STRING,T_STRING
3. 下一个token对应的值为]

可以判断出,这是在寻找数组的键值部分

进入该if语句后,首先将上一个token信息消除,再将该token的值去掉单双引号存入$this->token[$i+$c][3]位置的数组.

进入该if语句对应的else语句中,与前面取■■■■■index不同,else语句中则是对■■■■index的收集

首先是赋值语句,对token新增了第四个键值,并初始化为数组:

```
$this->tokens[$i][3][$index] = array();
```

接下来对$newbraceopen赋值为1,该变量可理解为[出现的次数.

往下两行是while循环:

```
while($newbraceopen !== 0)
{
   if( $this->tokens[$i+$c] === '[' )
   {
       $newbraceopen++;
   }
   else if( $this->tokens[$i+$c] === ']' )
   {
       $newbraceopen--;
   }
   else
   {
       $this->tokens[$i][3][$index][] = $this->tokens[$i+$c];
   }
   unset($this->tokens[$i+$c]);
   $c++;

   if(!isset($this->tokens[$i+$c]))
   {
       addError('Could not find closing bracket of '.$this->tokens[$i][1].'[].', array_slice($this->tokens, $i, 5), $this->tok
       break;
   }
}
```

有了上一个if的基础,我们可以轻易看出,该while语句作用为将数组的■存储在token信息的第四个键上.

到此为止,array_reconstruct_tokens函数的作用基本明了:

将数组如由$array[]格式转换为$token[i][3]格式表示的数据

fix_tokens()函数分析

■■■■■■■■■■,■```for```■```return```■■■■,■■```for```■■.

■■■```if```■■,■■token■■■■■■■■,■■■if■■.

```if```■■■■■■```while```■■,■■■■```if```■■■■```while```■■■■■■■■■■■■■■■■■■■■■■.

■■```while```■■■■■■■■■■■■■■,■```while```■■■■■■,■■■■■■■■■,■■■■■■,■■■■■■■■■■■■■■■■■```)```,■■■■■■■■■■■■■■■■■token■■:

$this->tokens[$i] = array(T_STRING, 'backticks', $line_nr);
```

■■■■```if```■■■■■```array_merger```■■,■■■■:

```php
$this->tokens = array_merge(
array_slice($this->tokens, 0, $i+1),
array('('),
array_slice($this->tokens, $i+1)
);
```

结合刚刚提到到,将第二个反引号替换为),那么换个角度看,其实也缺失了一个(,为了补齐这个括号,通过使用将token先分段,再插入,再组合的方法达到补齐括号的效果.

因为fix_token的函数过长,因此每个if我都会总结一下作用,那么这个if的作用其实便是:

■ `xxxx` ■■■ xxx()

接下来进入else if.

首先是if语句,进入if语句的条件为:

1. T_IF
2. T_ELSEIF
3. T_FOR
4. T_FOREACH
5. T_WHILE
6. ■■■■■■■■■■■■■■■ $this->tokens[$i+1] === '(' ■■

接下来是一个while语句,结合上面的经验,我们可以知道这其实是在对括号中的内容定位,然而并没有出现记录相关的操作,结合T_IF此类token信息,不难分析出这一步的whil

接着while语句的为一个if语句,相关代码为:

```php
if($this->tokens[$i+$f] === ':')
{
   switch($this->tokens[$i][0])
   {
       case T_IF:
       case T_ELSEIF: $endtoken = T_ENDIF; break;
       case T_FOR: $endtoken = T_ENDFOR; break;
       case T_FOREACH: $endtoken = T_ENDFOREACH; break;
       case T_WHILE: $endtoken = T_ENDWHILE; break;
       default: $endtoken = ';';
   }

   $c=1;
   while( $this->tokens[$i+$f+$c][0] !== $endtoken)
   {
       $c++;
       if(!isset($this->tokens[$i+$f+$c]))
       {
           addError('Could not find end'.$this->tokens[$i][1].'; of alternate '.$this->tokens[$i][1].'-statement.', array_slic
           break;
       }
   }
   $this->wrapbraces($i+$f+1, $c+1, $i+$f+$c+2);
}
```

进入if的条件为:

1. $this->tokens[$i+$f] === ':'

而if语句则是switch语句,分别对应T_IF一类的条件语句,然而再结合前面的$this->tokens[$i+$f] === ':'这个条件则让人有点不解.

这一部分其实是php的替代语法.比如:

```php
<?php if($a<0): ?>
123
<?php endif; ?>
```

替代语法的语法结构与我们常用的语法结构不同这一点十分重要.

在switch语句中,设置了对应不同token的结束符号,而接下来的while语句则是不断寻找对应的结束符号的出现位置.

在最后出现了函数wrapbraces.跟入:

```
function wrapbraces($start, $between, $end)
{
   $this->tokens = array_merge(
       array_slice($this->tokens, 0, $start), array('{'),
       array_slice($this->tokens, $start, $between), array('}'),
       array_slice($this->tokens, $end)
   );
}
```

与上面出现的array_merge作用类似,都是为了补齐语法结构,符合我们平常的使用习惯

```
<?php if($a<0) { ?>
   123
<?php }?>
```

到这一步为止,语法结构补完.

对应的else if语句则为:

```
else if($this->tokens[$i+$f] !== '{' && $this->tokens[$i+$f] !== ';'){
   $c=1;
   while($this->tokens[$i+$f+$c] !== ';' && $c<$max)
   {
       $c++;
   }
   $this->wrapbraces($i+$f, $c+1, $i+$f+$c+1);
}
```

由于我们已经跳过了判断的条件语句,那么此时$token[$i+$f]对应的其实是{,但是可以看到这里的else if判断条件便是■■{ ■■■ ;.

此类代码如下:

```
if($a==1) echo 1;
```

于是在这个else if语句里出现了while循环用以寻找这个语句块的结尾,并通过$this->wrapbraces来补齐语法结构.

再跟入下一个

```
else if(
$this->tokens[$i][0] === T_ELSE
&& $this->tokens[$i+1][0] !== T_IF
&& $this->tokens[$i+1] !== '{')
{
   $f=2;
   while( $this->tokens[$i+$f] !== ';' && $f<$max)
   {
       $f++;
   }
   $this->wrapbraces($i+1, $f, $i+$f+1);
}
```

语法结构基本一样,根据条件判断,该语句是用来补全else结构的{.

再往下依然是else if,代码如下:

```
else if( $this->tokens[$i][0] === T_SWITCH && $this->tokens[$i+1] === '(')
{
   $newbraceopen = 1;
   $c=2;
   while( $newbraceopen !== 0 )
   {
       if( $this->tokens[$i + $c] === '(' )
       {
           $newbraceopen++;
       }
       else if( $this->tokens[$i + $c] === ')' )
       {
           $newbraceopen--;
       }
       else if(!isset($this->tokens[$i+$c]) || $this->tokens[$i + $c] === ';')
       {
```

```
            addError('Could not find closing parenthesis of switch-statement.', array_slice($this->tokens, $i, 10), $this->toke
            break;
        }
        $c++;
    }
    // switch(): ... endswitch;
    if($this->tokens[$i + $c] === ':')
    {
        $f=1;
        while( $this->tokens[$i+$c+$f][0] !== T_ENDSWITCH)
        {
            $f++;
            if(!isset($this->tokens[$i+$c+$f]))
            {
                addError('Could not find endswitch; of alternate switch-statement.', array_slice($this->tokens, $i, $c+1), $thi
                break;
            }
        }
        $this->wrapbraces($i+$c+1, $f+1, $i+$c+$f+2);
    }
}
```

该else if语句进入的条件为switch语句,根据前面的经验,我们可以知道第一个while语句是用来寻找swicth的条件值,而下面的

```
if($this->tokens[$i + $c] === ':')
    {
        $f=1;
        while( $this->tokens[$i+$c+$f][0] !== T_ENDSWITCH)
        {
            $f++;
            if(!isset($this->tokens[$i+$c+$f]))
            {
                addError('Could not find endswitch; of alternate switch-statement.', array_slice($this->tokens, $i, $c+1), $thi
                break;
            }
        }
        $this->wrapbraces($i+$c+1, $f+1, $i+$c+$f+2);
    }
```

则是用来寻找switch语句的结尾并使用{}包裹,使之形成一个代码块.

继续看向下一个else if块:

```
else if( $this->tokens[$i][0] === T_CASE )
{
    $e=1;
    while($this->tokens[$i+$e] !== ':' && $this->tokens[$i+$e] !== ';')
    {
        $e++;

        if(!isset($this->tokens[$i+$e]))
        {
            addError('Could not find : or ; after '.$this->tokens[$i][1].'-statement.', array_slice($this->tokens, $i, 5), $thi
            break;
        }
    }
    $f=$e+1;
    if(($this->tokens[$i+$e] === ':' || $this->tokens[$i+$e] === ';')
    && $this->tokens[$i+$f] !== '{'
    && $this->tokens[$i+$f][0] !== T_CASE && $this->tokens[$i+$f][0] !== T_DEFAULT)
    {
        $newbraceopen = 0;
        while($newbraceopen || (isset($this->tokens[$i+$f]) && $this->tokens[$i+$f] !== '}'
        && !(is_array($this->tokens[$i+$f])
        && ($this->tokens[$i+$f][0] === T_BREAK || $this->tokens[$i+$f][0] === T_CASE
        || $this->tokens[$i+$f][0] === T_DEFAULT || $this->tokens[$i+$f][0] === T_ENDSWITCH) ) ))
        {
            if($this->tokens[$i+$f] === '{')
                $newbraceopen++;
            else if($this->tokens[$i+$f] === '}')
```

```
                $newbraceopen--;
            $f++;

            if(!isset($this->tokens[$i+$f]))
            {
                addError('Could not find ending of '.$this->tokens[$i][1].'-statement.', array_slice($this->tokens, $i, $e+5),
                break;
            }
        }
        if($this->tokens[$i+$f][0] === T_BREAK)
        {
            if($this->tokens[$i+$f+1] === ';')
                $this->wrapbraces($i+$e+1, $f-$e+1, $i+$f+2);
            // break 3;
            else
                $this->wrapbraces($i+$e+1, $f-$e+2, $i+$f+3);
        }
        else
        {
            $this->wrapbraces($i+$e+1, $f-$e-1, $i+$f);
        }
        $i++;
    }
}
```

类似的语法结构,使用while定位到冒号,跳过case条件,将case xxx:yyyy分割成case xxx、:yyyy两段.

随后开始处理第二段.

接着的是if语句,进入的条件为:

1. $this->tokens[$i+$e]为: 或 $this->tokens[$i+$e]为;
2. $this->tokens[$i+$f]不为{
3. $this->tokens[$i+$f][0]不为T_CASE或T_DEFAULT

在if语句继续包裹了一个条件要求较多的while语句,对应的条件如下:

```
while(

$newbraceopen
||
    (
    isset($this->tokens[$i+$f])
    &&
    $this->tokens[$i+$f] !== '}'
    &&
        !(
        is_array($this->tokens[$i+$f])
        &&
            (
            $this->tokens[$i+$f][0] === T_BREAK
            ||
            $this->tokens[$i+$f][0] === T_CASE
            ||
            $this->tokens[$i+$f][0] === T_DEFAULT
            ||
            $this->tokens[$i+$f][0] === T_ENDSWITCH)
            )
        )
    )
```

即:

1. $newbraceopen小于等于0
2. $this->tokens[$i+$f][0]处的token不为}或T_BREAK,T_CASE,T_DEFAULT,T_ENDSWITCH

[redacted],```swicth```[redacted],[redacted]:

```php
if($this->tokens[$i+$f][0] === T_BREAK)
```

```
{
   if($this->tokens[$i+$f+1] === ';')
       $this->wrapbraces($i+$e+1, $f-$e+1, $i+$f+2);
   else
       $this->wrapbraces($i+$e+1, $f-$e+2, $i+$f+3);
}
else
{
   $this->wrapbraces($i+$e+1, $f-$e-1, $i+$f);
}
```

这一段主要作用为在break语句处加上{},补全语法结构.

接下来是与上面判断为case同级的else if语句,代码如下:

```
else if( $this->tokens[$i][0] === T_DEFAULT
&& $this->tokens[$i+2] !== '{' )
{
   $f=2;
   $newbraceopen = 0;
   while( $this->tokens[$i+$f] !== ';' && $this->tokens[$i+$f] !== '}' || $newbraceopen )
   {
       if($this->tokens[$i+$f] === '{')
           $newbraceopen++;
       else if($this->tokens[$i+$f] === '}')
           $newbraceopen--;
       $f++;

       if(!isset($this->tokens[$i+$f]))
       {
           addError('Could not find ending of '.$this->tokens[$i][1].'-statement.', array_slice($this->tokens, $i, 5), $this->
           break;
       }
   }
   $this->wrapbraces($i+2, $f-1, $i+$f+1);
}
```

该语句进入的条件为token索引信息对应为T_DEFAULT.

结合上面的分析经验,本段代码作用为将default的条件部分使用花括号包括,补全语法结构.

再往下为:

```
else if( $this->tokens[$i][0] === T_FUNCTION )
{
   $this->tokens[$i+1][1] = strtolower($this->tokens[$i+1][1]);
}
else if( $this->tokens[$i][0] === T_STRING && $this->tokens[$i+1] === '(')
{
   $this->tokens[$i][1] = strtolower($this->tokens[$i][1]);
}
```

这一段是将函数名全部小写,并没有太多要详细说明的内容.接下来是else if语句:

```
else if( $this->tokens[$i][0] === T_DO )
{
   $f=2;
   $otherDOs = 0;
   //■■■■■■while,■■■■while
   while( $this->tokens[$i+$f][0] !== T_WHILE || $otherDOs )
   {
       if($this->tokens[$i+$f][0] === T_DO)
           $otherDOs++;
       else if($this->tokens[$i+$f][0] === T_WHILE)
           $otherDOs--;
       $f++;

       if(!isset($this->tokens[$i+$f]))
       {
           addError('Could not find WHILE of DO-WHILE-statement.', array_slice($this->tokens, $i, 5), $this->tokens[$i][2], $t
           break;
```

```
        }
    }

    // ■■■■■
    if($this->tokens[$i+1] !== '{')
    {
        $this->wrapbraces($i+1, $f-1, $i+$f);
        // by adding braces we added two new tokens
        $f+=2;
    }

    $d=1;
    //$max=count($this->tokens)  ■■■while■■■■■■■■do-while■■■
    while( $this->tokens[$i+$f+$d] !== ';' \&\& $d<$max )
    {
        $d++;
    }

    // ■token■■do-while■■while
    $this->tokens = array_merge(
        array_slice($this->tokens, 0, $i), // before DO
        array_slice($this->tokens, $i+$f, $d), // WHILE condition
        array_slice($this->tokens, $i+1, $f-1), // DO WHILE loop tokens
        array_slice($this->tokens, $i+$f+$d+1, count($this->tokens)) // rest of tokens without while condition
    );
}
```

在前面的基础上,我们再来分析这一段代码便简单许多,简化一下描述便是:该段代码用以整合do-while语句,补齐语法结构并将`do-while`精简为`while`.

最后返回精简过的token信息:

```
$this->tokens = array_values($this->tokens);
```

fix_ternary函数分析

从函数名分析分析,该函数作用为处理三元操作符,使其变为我们常见的语法习惯.大体结构仍然为`for`循环搭配`return`语句.

首先是`if`语句判断是否为?,为真则进入.并在进入后立即删除问号,随后判断在问号之前的符号是否为`)`,为真则进入,随后又删除反括号.并通过while语句将问号之前的使用括号

随后是if语句:

```
if($this->tokens[$i-$f] === '!'
|| (
    is_array($this->tokens[$i-$f])
    && ($this->tokens[$i-$f][0] === T_STRING
        || $this->tokens[$i-$f][0] === T_EMPTY
        || $this->tokens[$i-$f][0] === T_ISSET
        )
    )
){
    unset($this->tokens[$i-$f]);
}
```

该段if语句满足以下条件之一即可进行删除token信息处理:

```
1. $this->tokens[$i-$f] ■ !
2. $this->tokens[$i-$f] ■ ■■■■is_empty()■isset()
```

接着进入与上面if同级的else if语句中:

```
else if(in_array($this->tokens[$i-2][0], Tokens::$T_ASSIGNMENT) || in_array($this->tokens[$i-2][0], Tokens::$T_OPERATOR) )
```

可以看出,仅当`$this->tokens[$i-2][0]`为指定的token信息时,才会进入接下来的操作,而指定的token信息为:

```
1. $T_ASSIGNMENT    // ■■■
2. $T_OPERATOR      // ■■■
```

其中,`$T_ASSIGNMENT`为:

```
public static $T_ASSIGNMENT = array(
    T_AND_EQUAL,
```

```
   T_CONCAT_EQUAL,
   T_DIV_EQUAL,
   T_MINUS_EQUAL,
   T_MOD_EQUAL,
   T_MUL_EQUAL,
   T_OR_EQUAL,
   T_PLUS_EQUAL,
   T_SL_EQUAL,
   T_SR_EQUAL,
   T_XOR_EQUAL
);
```

而$T_OPERATOR为:

```
public static $T_OPERATOR = array(
   T_IS_EQUAL,
   T_IS_GREATER_OR_EQUAL,
   T_IS_IDENTICAL,
   T_IS_NOT_EQUAL,
   T_IS_NOT_IDENTICAL,
   T_IS_SMALLER_OR_EQUAL
);
```

而在接下来的操作中,rips删除了$this->tokens[$i-1]以及$this->tokens[$i-2]的token信息,这里删除-1与-2位置的token是因为上面的操作符通常都是成对出现的

而接下来的while语句则与前面的作用相同,都是用以删除在目标位置前,包裹在括号内的内容以及某几个特定的token信息.

随后进行最后的一次if判断,判断是否条件部分为单独的一个变量,如是,则删除.

最终返回token信息,至此,rips的token分析过程结束

Scanner效果展示

我们自定义待扫描文件内容为:

```
<?php


$a = $_GET['a'];
$b = $_POST['b'];
$c = array("c"=>"c","d"=>"d");
$d = ['1','2'];
// xxxxxxx
//
`ls`;

if($a=="1") $b="2";

$a=isset($c)?"aa":"bb";
```

分别在prepare_token,array_reconstruct_tokens,fix_tokens,fix_ternary函数尾处添加var_dump函数,并在tokenize函数尾处写入die()

首先输出的token为:

```
0|1|/Applications/MAMP/htdocs/aaa.php|0| 0|1|/Applications/MAMP/htdocs/aaa.php (tokenizing)|0|
/Applications/MAMP/htdocs/rips/lib/tokenizer.php:92:
array (size=60)
 0 =>
   array (size=3)
     0 => int 320
     1 => string '$a' (length=2)
     2 => int 3
 1 => string '=' (length=1)
 2 =>
   array (size=3)
     0 => int 320
     1 => string '$_GET' (length=5)
     2 => int 3
 3 => string '[' (length=1)
 4 =>
```

```
  array (size=3)
    0 => int 323
    1 => string ''a'' (length=3)
    2 => int 3
5 => string ']' (length=1)
6 => string ';' (length=1)
7 =>
  array (size=3)
    0 => int 320
    1 => string '$b' (length=2)
    2 => int 4
8 => string '=' (length=1)
9 =>
  array (size=3)
    0 => int 320
    1 => string '$_POST' (length=6)
    2 => int 4
10 => string '[' (length=1)
11 =>
  array (size=3)
    0 => int 323
    1 => string ''b'' (length=3)
    2 => int 4
12 => string ']' (length=1)
13 => string ';' (length=1)
14 =>
  array (size=3)
    0 => int 320
    1 => string '$c' (length=2)
    2 => int 5
15 => string '=' (length=1)
16 =>
  array (size=3)
    0 => int 368
    1 => string 'array' (length=5)
    2 => int 5
17 => string '(' (length=1)
18 =>
  array (size=3)
    0 => int 323
    1 => string '"c"' (length=3)
    2 => int 5
19 =>
  array (size=3)
    0 => int 268
    1 => string '=>' (length=2)
    2 => int 5
20 =>
  array (size=3)
    0 => int 323
    1 => string '"c"' (length=3)
    2 => int 5
21 => string ',' (length=1)
22 =>
  array (size=3)
    0 => int 323
    1 => string '"d"' (length=3)
    2 => int 5
23 =>
  array (size=3)
    0 => int 268
    1 => string '=>' (length=2)
    2 => int 5
24 =>
  array (size=3)
    0 => int 323
    1 => string '"d"' (length=3)
    2 => int 5
25 => string ')' (length=1)
```

```
26 => string ';' (length=1)
27 =>
  array (size=3)
    0 => int 320
    1 => string '$d' (length=2)
    2 => int 6
28 => string '=' (length=1)
29 => string '[' (length=1)
30 =>
  array (size=3)
    0 => int 323
    1 => string ''1'' (length=3)
    2 => int 6
31 => string ',' (length=1)
32 =>
  array (size=3)
    0 => int 323
    1 => string ''2'' (length=3)
    2 => int 6
33 => string ']' (length=1)
34 => string ';' (length=1)
35 => string '`' (length=1)
36 =>
  array (size=3)
    0 => int 322
    1 => string 'ls' (length=2)
    2 => int 9
37 => string '`' (length=1)
38 => string ';' (length=1)
39 =>
  array (size=3)
    0 => int 327
    1 => string 'if' (length=2)
    2 => int 11
40 => string '(' (length=1)
41 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 11
42 =>
  array (size=3)
    0 => int 285
    1 => string '==' (length=2)
    2 => int 11
43 =>
  array (size=3)
    0 => int 323
    1 => string '"1"' (length=3)
    2 => int 11
44 => string ')' (length=1)
45 =>
  array (size=3)
    0 => int 320
    1 => string '$b' (length=2)
    2 => int 11
46 => string '=' (length=1)
47 =>
  array (size=3)
    0 => int 323
    1 => string '"2"' (length=3)
    2 => int 11
48 => string ';' (length=1)
49 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 13
50 => string '=' (length=1)
```

```
 51 =>
   array (size=3)
     0 => int 358
     1 => string 'isset' (length=5)
     2 => int 13
 52 => string '(' (length=1)
 53 =>
   array (size=3)
     0 => int 320
     1 => string '$c' (length=2)
     2 => int 13
 54 => string ')' (length=1)
 55 => string '?' (length=1)
 56 =>
   array (size=3)
     0 => int 323
     1 => string '"aa"' (length=4)
     2 => int 13
 57 => string ':' (length=1)
 58 =>
   array (size=3)
     0 => int 323
     1 => string '"bb"' (length=4)
     2 => int 13
 59 => string ';'
```

随后经过array_reconstruct_tokens函数处理,重写了数组相关的token信息:

```
/Applications/MAMP/htdocs/rips/lib/tokenizer.php:454:
array (size=54)
 0 =>
   array (size=3)
     0 => int 320
     1 => string '$a' (length=2)
     2 => int 3
 1 => string '=' (length=1)
 2 =>
   array (size=4)
     0 => int 320
     1 => string '$_GET' (length=5)
     2 => int 3
     3 =>
       array (size=1)
         0 => string 'a' (length=1)
 3 => string ';' (length=1)
 4 =>
   array (size=3)
     0 => int 320
     1 => string '$b' (length=2)
     2 => int 4
 5 => string '=' (length=1)
 6 =>
   array (size=4)
     0 => int 320
     1 => string '$_POST' (length=6)
     2 => int 4
     3 =>
       array (size=1)
         0 => string 'b' (length=1)
 7 => string ';' (length=1)
 8 =>
   array (size=3)
     0 => int 320
     1 => string '$c' (length=2)
     2 => int 5
 9 => string '=' (length=1)
 10 =>
   array (size=3)
     0 => int 368
```

```
    1 => string 'array' (length=5)
    2 => int 5
11 => string '(' (length=1)
12 =>
  array (size=3)
    0 => int 323
    1 => string '"c"' (length=3)
    2 => int 5
13 =>
  array (size=3)
    0 => int 268
    1 => string '=>' (length=2)
    2 => int 5
14 =>
  array (size=3)
    0 => int 323
    1 => string '"c"' (length=3)
    2 => int 5
15 => string ',' (length=1)
16 =>
  array (size=3)
    0 => int 323
    1 => string '"d"' (length=3)
    2 => int 5
17 =>
  array (size=3)
    0 => int 268
    1 => string '=>' (length=2)
    2 => int 5
18 =>
  array (size=3)
    0 => int 323
    1 => string '"d"' (length=3)
    2 => int 5
19 => string ')' (length=1)
20 => string ';' (length=1)
21 =>
  array (size=3)
    0 => int 320
    1 => string '$d' (length=2)
    2 => int 6
22 => string '=' (length=1)
23 => string '[' (length=1)
24 =>
  array (size=3)
    0 => int 323
    1 => string ''1'' (length=3)
    2 => int 6
25 => string ',' (length=1)
26 =>
  array (size=3)
    0 => int 323
    1 => string ''2'' (length=3)
    2 => int 6
27 => string ']' (length=1)
28 => string ';' (length=1)
29 => string '`' (length=1)
30 =>
  array (size=3)
    0 => int 322
    1 => string 'ls' (length=2)
    2 => int 9
31 => string '`' (length=1)
32 => string ';' (length=1)
33 =>
  array (size=3)
    0 => int 327
    1 => string 'if' (length=2)
    2 => int 11
```

```
 34 => string '(' (length=1)
 35 =>
   array (size=3)
     0 => int 320
     1 => string '$a' (length=2)
     2 => int 11
 36 =>
   array (size=3)
     0 => int 285
     1 => string '==' (length=2)
     2 => int 11
 37 =>
   array (size=3)
     0 => int 323
     1 => string '"1"' (length=3)
     2 => int 11
 38 => string ')' (length=1)
 39 =>
   array (size=3)
     0 => int 320
     1 => string '$b' (length=2)
     2 => int 11
 40 => string '=' (length=1)
 41 =>
   array (size=3)
     0 => int 323
     1 => string '"2"' (length=3)
     2 => int 11
 42 => string ';' (length=1)
 43 =>
   array (size=3)
     0 => int 320
     1 => string '$a' (length=2)
     2 => int 13
 44 => string '=' (length=1)
 45 =>
   array (size=3)
     0 => int 358
     1 => string 'isset' (length=5)
     2 => int 13
 46 => string '(' (length=1)
 47 =>
   array (size=3)
     0 => int 320
     1 => string '$c' (length=2)
     2 => int 13
 48 => string ')' (length=1)
 49 => string '?' (length=1)
 50 =>
   array (size=3)
     0 => int 323
     1 => string '"aa"' (length=4)
     2 => int 13
 51 => string ':' (length=1)
 52 =>
   array (size=3)
     0 => int 323
     1 => string '"bb"' (length=4)
     2 => int 13
 53 => string ';' (length=1)
```

再经过fix_tokens处理,统一了部分token信息的写法(如对if语句统一使用花括号的标示形式)

```
/Applications/MAMP/htdocs/rips/lib/tokenizer.php:379:
array (size=57)
 0 =>
   array (size=3)
     0 => int 320
     1 => string '$a' (length=2)
```

```
    2 => int 3
1 => string '=' (length=1)
2 =>
  array (size=4)
    0 => int 320
    1 => string '$_GET' (length=5)
    2 => int 3
    3 =>
      array (size=1)
        0 => string 'a' (length=1)
3 => string ';' (length=1)
4 =>
  array (size=3)
    0 => int 320
    1 => string '$b' (length=2)
    2 => int 4
5 => string '=' (length=1)
6 =>
  array (size=4)
    0 => int 320
    1 => string '$_POST' (length=6)
    2 => int 4
    3 =>
      array (size=1)
        0 => string 'b' (length=1)
7 => string ';' (length=1)
8 =>
  array (size=3)
    0 => int 320
    1 => string '$c' (length=2)
    2 => int 5
9 => string '=' (length=1)
10 =>
  array (size=3)
    0 => int 368
    1 => string 'array' (length=5)
    2 => int 5
11 => string '(' (length=1)
12 =>
  array (size=3)
    0 => int 323
    1 => string '"c"' (length=3)
    2 => int 5
13 =>
  array (size=3)
    0 => int 268
    1 => string '=>' (length=2)
    2 => int 5
14 =>
  array (size=3)
    0 => int 323
    1 => string '"c"' (length=3)
    2 => int 5
15 => string ',' (length=1)
16 =>
  array (size=3)
    0 => int 323
    1 => string '"d"' (length=3)
    2 => int 5
17 =>
  array (size=3)
    0 => int 268
    1 => string '=>' (length=2)
    2 => int 5
18 =>
  array (size=3)
    0 => int 323
    1 => string '"d"' (length=3)
    2 => int 5
```

```
19 => string ')' (length=1)
20 => string ';' (length=1)
21 =>
  array (size=3)
    0 => int 320
    1 => string '$d' (length=2)
    2 => int 6
22 => string '=' (length=1)
23 => string '[' (length=1)
24 =>
  array (size=3)
    0 => int 323
    1 => string ''1'' (length=3)
    2 => int 6
25 => string ',' (length=1)
26 =>
  array (size=3)
    0 => int 323
    1 => string ''2'' (length=3)
    2 => int 6
27 => string ']' (length=1)
28 => string ';' (length=1)
29 =>
  array (size=3)
    0 => int 319
    1 => string 'backticks' (length=9)
    2 => int 9
30 => string '(' (length=1)
31 =>
  array (size=3)
    0 => int 322
    1 => string 'ls' (length=2)
    2 => int 9
32 => string ')' (length=1)
33 => string ';' (length=1)
34 =>
  array (size=3)
    0 => int 327
    1 => string 'if' (length=2)
    2 => int 11
35 => string '(' (length=1)
36 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 11
37 =>
  array (size=3)
    0 => int 285
    1 => string '==' (length=2)
    2 => int 11
38 =>
  array (size=3)
    0 => int 323
    1 => string '"1"' (length=3)
    2 => int 11
39 => string ')' (length=1)
40 => string '{' (length=1)
41 =>
  array (size=3)
    0 => int 320
    1 => string '$b' (length=2)
    2 => int 11
42 => string '=' (length=1)
43 =>
  array (size=3)
    0 => int 323
    1 => string '"2"' (length=3)
    2 => int 11
```

```
 44 => string ';' (length=1)
 45 => string '}' (length=1)
 46 =>
   array (size=3)
     0 => int 320
     1 => string '$a' (length=2)
     2 => int 13
 47 => string '=' (length=1)
 48 =>
   array (size=3)
     0 => int 358
     1 => string 'isset' (length=5)
     2 => int 13
 49 => string '(' (length=1)
 50 =>
   array (size=3)
     0 => int 320
     1 => string '$c' (length=2)
     2 => int 13
 51 => string ')' (length=1)
 52 => string '?' (length=1)
 53 =>
   array (size=3)
     0 => int 323
     1 => string '"aa"' (length=4)
     2 => int 13
 54 => string ':' (length=1)
 55 =>
   array (size=3)
     0 => int 323
     1 => string '"bb"' (length=4)
     2 => int 13
 56 => string ';' (length=1)
```

最终经过fix_ternary函数处理,是三元运算符的表达形式得到重写($a=isset($c)?"aa":"bb"; => $a="aa":"bb"):

```
/Applications/MAMP/htdocs/rips/lib/tokenizer.php:558:
array (size=52)
 0 =>
   array (size=3)
     0 => int 320
     1 => string '$a' (length=2)
     2 => int 3
 1 => string '=' (length=1)
 2 =>
   array (size=4)
     0 => int 320
     1 => string '$_GET' (length=5)
     2 => int 3
     3 =>
       array (size=1)
         0 => string 'a' (length=1)
 3 => string ';' (length=1)
 4 =>
   array (size=3)
     0 => int 320
     1 => string '$b' (length=2)
     2 => int 4
 5 => string '=' (length=1)
 6 =>
   array (size=4)
     0 => int 320
     1 => string '$_POST' (length=6)
     2 => int 4
     3 =>
       array (size=1)
         0 => string 'b' (length=1)
 7 => string ';' (length=1)
 8 =>
```

```
  array (size=3)
    0 => int 320
    1 => string '$c' (length=2)
    2 => int 5
9 => string '=' (length=1)
10 =>
  array (size=3)
    0 => int 368
    1 => string 'array' (length=5)
    2 => int 5
11 => string '(' (length=1)
12 =>
  array (size=3)
    0 => int 323
    1 => string '"c"' (length=3)
    2 => int 5
13 =>
  array (size=3)
    0 => int 268
    1 => string '=>' (length=2)
    2 => int 5
14 =>
  array (size=3)
    0 => int 323
    1 => string '"c"' (length=3)
    2 => int 5
15 => string ',' (length=1)
16 =>
  array (size=3)
    0 => int 323
    1 => string '"d"' (length=3)
    2 => int 5
17 =>
  array (size=3)
    0 => int 268
    1 => string '=>' (length=2)
    2 => int 5
18 =>
  array (size=3)
    0 => int 323
    1 => string '"d"' (length=3)
    2 => int 5
19 => string ')' (length=1)
20 => string ';' (length=1)
21 =>
  array (size=3)
    0 => int 320
    1 => string '$d' (length=2)
    2 => int 6
22 => string '=' (length=1)
23 => string '[' (length=1)
24 =>
  array (size=3)
    0 => int 323
    1 => string ''1'' (length=3)
    2 => int 6
25 => string ',' (length=1)
26 =>
  array (size=3)
    0 => int 323
    1 => string ''2'' (length=3)
    2 => int 6
27 => string ']' (length=1)
28 => string ';' (length=1)
29 =>
  array (size=3)
    0 => int 319
    1 => string 'backticks' (length=9)
    2 => int 9
```

```
30 => string '(' (length=1)
31 =>
  array (size=3)
    0 => int 322
    1 => string 'ls' (length=2)
    2 => int 9
32 => string ')' (length=1)
33 => string ';' (length=1)
34 =>
  array (size=3)
    0 => int 327
    1 => string 'if' (length=2)
    2 => int 11
35 => string '(' (length=1)
36 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 11
37 =>
  array (size=3)
    0 => int 285
    1 => string '==' (length=2)
    2 => int 11
38 =>
  array (size=3)
    0 => int 323
    1 => string '"1"' (length=3)
    2 => int 11
39 => string ')' (length=1)
40 => string '{' (length=1)
41 =>
  array (size=3)
    0 => int 320
    1 => string '$b' (length=2)
    2 => int 11
42 => string '=' (length=1)
43 =>
  array (size=3)
    0 => int 323
    1 => string '"2"' (length=3)
    2 => int 11
44 => string ';' (length=1)
45 => string '}' (length=1)
46 =>
  array (size=3)
    0 => int 320
    1 => string '$a' (length=2)
    2 => int 13
47 => string '=' (length=1)
48 =>
  array (size=3)
    0 => int 323
    1 => string '"aa"' (length=4)
    2 => int 13
49 => string ':' (length=1)
50 =>
  array (size=3)
    0 => int 323
    1 => string '"bb"' (length=4)
    2 => int 13
51 => string ';' (length=1)
```

流程总结

1. 通过prepare_token生成初始token信息
2. 通过array_reconstruct_tokens函数重写数组相关token信息
3. 通过fix_tokens修复大量写法不统一的语句

4. 用过`fix_ternary`统一三元运算符的表达形式

通过以上四步,我们可以得到大致处理好的token信息,而对于漏洞的扫描也是建立在上面这四步生成的token信息基础上的.

上一篇：ColdFusion再爆远程代码执... 下一篇：通过Unquoted servic...

1. 0 条回复
   - 动动手指，沙发就是你的了！