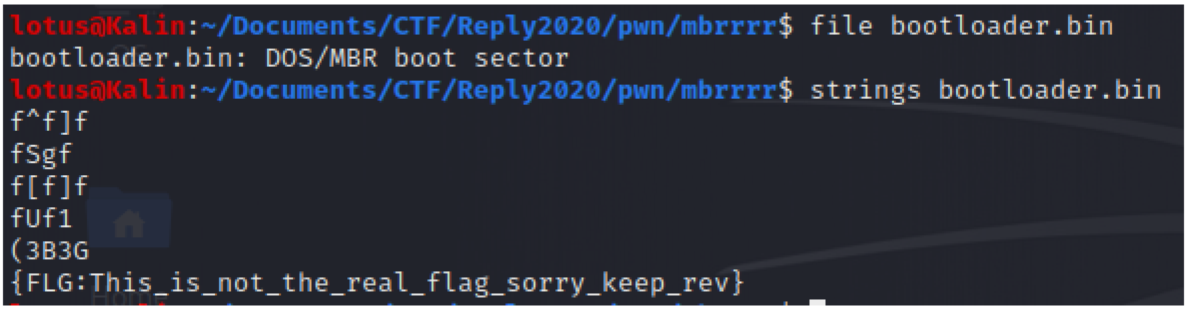
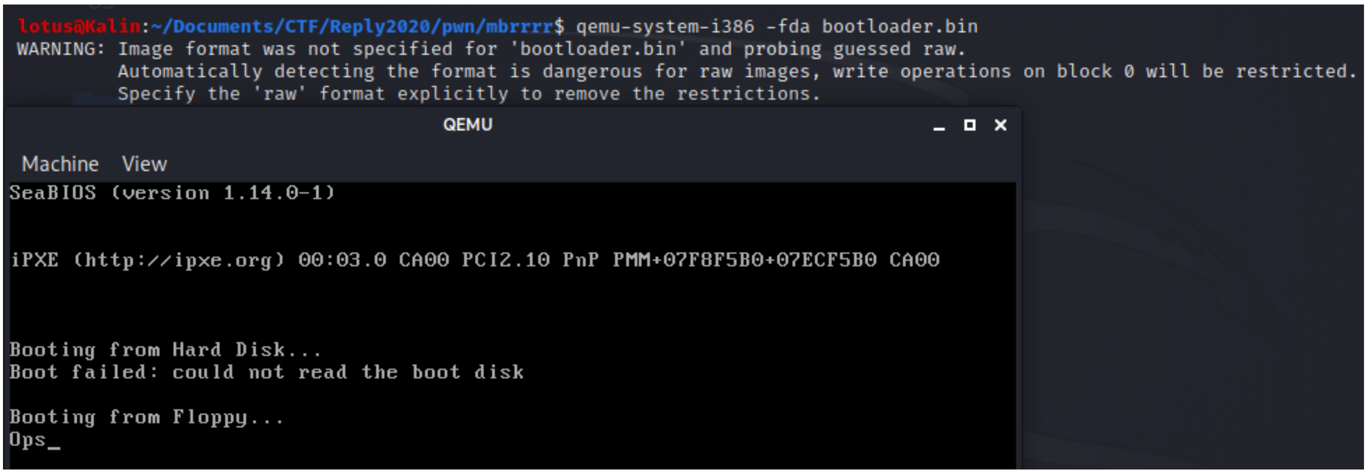
[Binary] mBRrrr Writeup

We are given a file named bootloader.bin. Lets run some basic command:



So we have a boot sector file, and it has a fake flag inside it as well. Let's try to execute it with qemu and see what happens:

- `qemu-system-i386 -fda bootloader.bin`



It's printing "Ops", let's disassemble it and look at the code:

- `ndisasm -b 16 bootloader.bin`

Let's also see the structure of the file with xxd:

- `xxd bootloader.bin`

Executing these commands we can see that this boot sector is divided in two parts, the first is basically loading the second at address `0x8000` in memory and then jumping to instructions at address `0x802a`.

We know the second part of our boot sector starts at address `0x200` in our disassemble so effective execution will start at `0x22a`, let's see what instructions are there:

```
00000200  6655              push ebp
00000202  6689E5            mov ebp,esp
00000205  6653              push ebx
00000207  67668B5508        mov edx,[ebp+0x8]
0000020C  66BB07000000      mov ebx,0x7
00000212  67660FBE02        movsx eax,byte [edx]
00000217  84C0              test al,al
00000219  7409              jz 0x224
0000021B  80CC0E            or ah,0xe
0000021E  CD10              int 0x10
00000220  6642              inc edx
00000222  EBEE              jmp short 0x212
00000224  665B              pop ebx
00000226  665D              pop ebp
00000228  66C3              retd
0000022A  6655              push ebp
0000022C  6631D2            xor edx,edx
0000022F  6689E5            mov ebp,esp
00000232  6683EC08          sub esp,byte +0x8
00000236  678A8240810000    mov al,[edx+0x8140]
0000023D  67328200810000    xor al,[edx+0x8100]
00000244  00D0              add al,dl
00000246  3413              xor al,0x13
00000248  00D0              add al,dl
0000024A  3419              xor al,0x19
0000024C  00D0              add al,dl
0000024E  673A8280810000    cmp al,[edx+0x8180]
00000255  740C              jz 0x263
00000257  6683EC0C          sub esp,byte +0xc
0000025B  6668AE810000      push dword 0x81ae
00000261  EB12              jmp short 0x275
00000263  6642              inc edx
00000265  6683FA2E          cmp edx,byte +0x2e
00000269  75CB              jnz 0x236
0000026B  6683EC0C          sub esp,byte +0xc
0000026F  666840810000      push dword 0x8140
00000275  66E885FFFFFF      call dword 0x200
```

This is the interesting part of this challenge, here we can see that a byte from an address (`edx+0x8140`) is loaded in `al` register and then it's xored with some other bytes, one taken from another address (`edx+0x8100`) and the others are some constants `0x13` and `0x19`, then, it's going to be compared to the value stored at `edx+0x8180`. What is going on here? This is simply a loop that will execute for 0x2e times and after that, if all the compares are successful, we are pushing a value which is the same used before `0x8140` and then calling code at offset `0x200`. If compares will fail the value pushed will be `0x81ae` but still the call will happen. Looking at these addresses we see that `0x8140` is where the fake flag is stored at while `0x81ae` contains the "Ops" string. Now we know what to do, we know that the execution will be successful if we can pass all the compares, so I took the bytes strings from addresses `0x8100` and `0x8140` which are used for xor and cmp and reverse the for loop in a simple python script:

```
#!/bin/python3

flag = b""
string2=bytes.fromhex("c7bb877c20f7f36583b12370ed0283a9c31fd98fe402faf939feddbd0de943489bee622c89102833423347c59555")
test = bytearray(bytes.fromhex("b6f8fb2c0cc9d752b6c37a852a9ffed040312b08288c11126e2cdcfd973986d208e1c06c0e8778a4e8b8ca4c1b97"))
```

```
for i in range(46):
    test[i] = (test[i]-i) % 256
    test[i] = test[i]^0x19
    test[i] = (test[i]-i) % 256
    test[i] = test[i]^0x13
    test[i] = (test[i]-i) % 256
    test[i] = test[i]^string2[i]

print(test.decode())
```

We run the script and get the flag `{FLG:18471a01b9b9528273857ee47a19d6710848f568}`

Tips

To debug this dynamically we can use qemu with gdb in the following way:

- `qemu-system-i386 -s -S -fda bootloader.bin`

Inside gdb:

- `target remote localhost:1234`