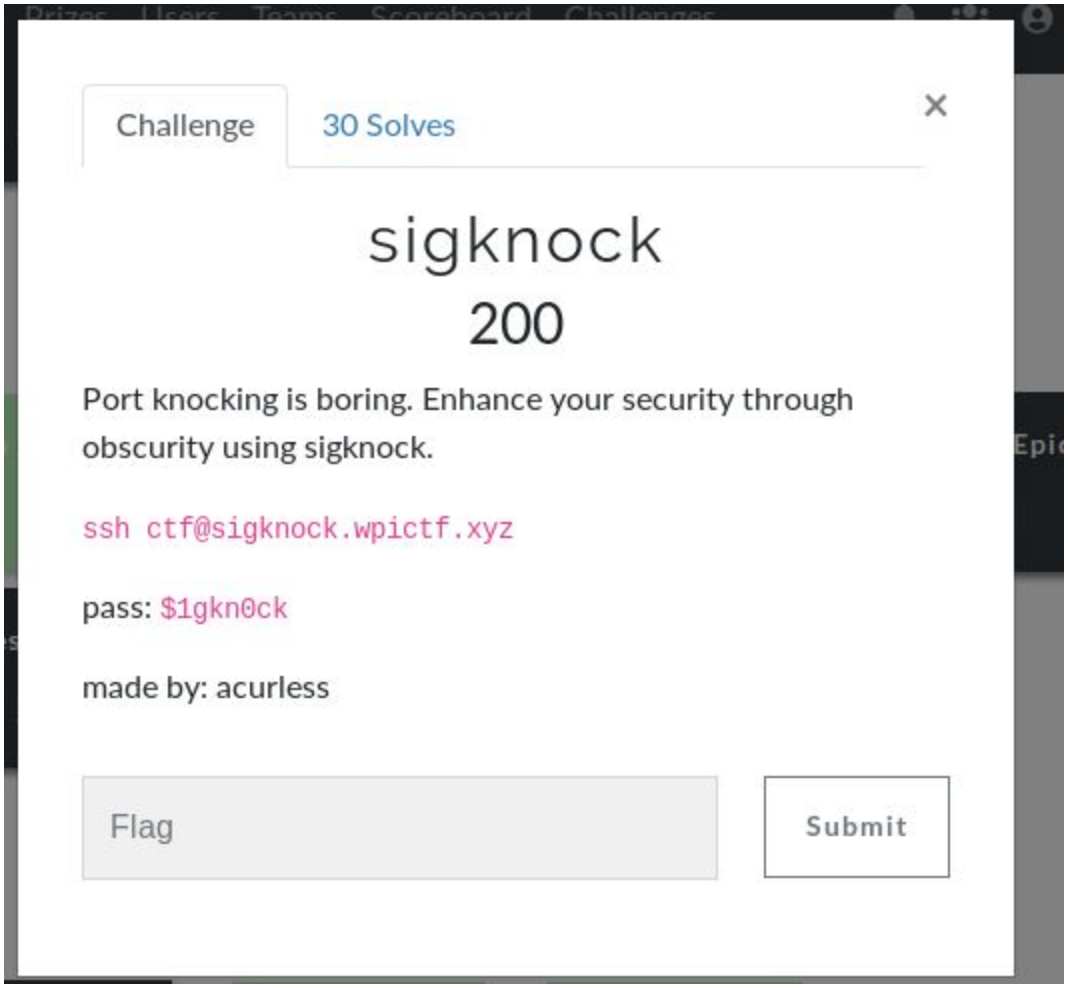
This challenge was presented at WPICTF 2020.

For the challenge, we were given login credentials for a user through SSH, and the following description image below:



Since the description talks about port knocking, but then has a name **sigknock**, my first thoughts were that this may be a case where a program is using **signals**. So the first thing I did was run **ls**, but found nothing in the directory. So the next command I ran was **ps**, to see if there were any interesting running processes.

```
root@kali:~/ctf/wpi/2020/sigknock# ssh ctf@sigknock.wpictf.xyz
Password:
~ $ ls
~ $ ps
PID   USER     TIME  COMMAND
    1 wpictf    0:00 {init_d} /bin/sh /bin/init_d
    6 wpictf    0:00 /usr/bin/irqknock
    7 wpictf    0:00 /bin/sh
    9 wpictf    0:00 ps
~ $
```

I noticed the program **irqknock**, which seemed interesting since I've never heard of it before and it simply had the word "knock" in its name. I also noticed that the program was in /usr/bin, so I ran it myself, and noticed when I tried to close it, it caught the SIGINT (ctrl-C) and advanced its state to 1.

```
~ $ irqknock
^CGot signal 2
State advanced to 1
```

So this confirmed what I thought. This challenge was going to be sending signals to the program, in the correct sequence, to get it to return the flag. The reason I thought of this, was because of the description.

So to solve this challenge the steps are as follows:

1. Determine which signals the program is catching
2. Determine the order in which the signals must be caught

The first step can be completed by viewing the SigCgt bitmask in **/proc/<PID>/status**, where <PID> is the process id of the program. In this case, the process ID is 6, shown in the above screenshot with the output of the **ps** command. So when I **cat** /proc/6/status I saw:

```
SigQ:   0/15686
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000000000
SigCgt: 0000000000011406
CapInh: 00000000a80425fb
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
NoNewPrivs:     0
```

I marked the line that matters here, that is a hexadecimal bitmask of the signals the program catches. Converting 0x11406 to binary results in: **10001010000000110**. This says that the program catches the following signals: 2, 3, 11, 13, 17.

The next step is to determine the order. They could have been in any order, but in this case they just happened to be in sequential order. So when you send the signals in order to the program, at the end it will give you the flag.