Majd Hamdan

CSCI 1005

libc and ROP Chain

January 18, 2021

1 Hack the Metadata

- Compile the ”hi” program from last week without the ”execstack” flag. Modify the compiled program to have (when run) an executable stack.

  Confirm that you have done so both by running the program and examining /proc/<pid>/maps, and by figuring out the stack executability is reported in the output of readelf.

Intuition is to to compile to programs, one with the -Wl,-z,execstack option and one without, then diff the compiled two programs. Here is the make file:

```
hi: hi.c
  gcc -g -fno-stack-protector -o hi hi.c


hi-ex: hi.c
  gcc -g -fno-stack-protector -Wl,-z,execstack -o hi_2 hi.c
```

Then I rean diff on hi and hi_2, the output is:

```
[@majd 5]$ diff hi hi_2
Binary files hi and hi_2 differ
```

diff didn’t give me any output. Why? I used readelf to get both hi and hi_2 into a readable form:

```
[@majd 5]$ readelf -a hi > elf_info
[@majd 5]$ readelf -a hi_2 > elf_info_ex
```

I opened the files in ATOM. My first intuition was to find if the word stack appear in there. It does, and in fact, the two files differ with the info there:

```
elf_info (hi):
...
GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
...

elf_info_ex(hi_2):
GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RWE     0x10
```

Just adding the letter E to the GNU_STACK section of hi should make its stack executable! The files obviously differ but why didn’t diff show this different? I looked it up on google and found this article: https://www.geeksforgeeks.org/diff-command-linux-examples/. It seems diff doesn’t work well with binary data. Solution? Use xxd which is a program to produce hex representation of binary data:

```
[@majd 5]$ xxd hi > hi_hex_dump
[]@majd 5]$ xxd hi_2 > hi_2_hex_dump
```

Then I can run the diff command on them:

```
[majd@majd 5]$ diff hi_hex_dump hi_2_hex_dump
43c43
< 000002a0: 0400 0000 0000 0000 51e5 7464 0600 0000  ........Q.td....
---
> 000002a0: 0400 0000 0000 0000 51e5 7464 0700 0000  ........Q.td....
57,58c57,58
< 00000380: 0300 0000 474e 5500 fba1 d3c8 104d 55b2  ....GNU......MU.
< 00000390: 568a 5947 bf8f 296c 95e1 2297 0400 0000  V.YG..)l..".....
---
> 00000380: 0300 0000 474e 5500 9f85 243d c796 4002  ....GNU...$=..@.
> 00000390: 4ff1 23b5 321a c4c0 cc0a 736e 0400 0000  O.#.2.....sn....
```

Great! There are only two lines that differ in the two programs, line 2a0 and lines 380-390. Which of these is where the E permission is added. Easy, we do diff on the read elf output of the two programs:

```
[@majd 5]$ diff elf_info elf_info_ex > elf_info_diff
```

and the output is:

```
132c132
<                    0x0000000000000000 0x0000000000000000  RW     0x10
---
>                    0x0000000000000000 0x0000000000000000  RWE    0x10
288c288
<       Build ID: fba1d3c8104d55b2568a5947bf8f296c95e12297
---
>       Build ID: 9f85243dc79640024ff123b5321ac4c0cc0a736e
```

we also see two differences between the two files, which is expected. It seems that the first difference is the RW, RWE difference. Great!! All we have to do now is change this line in hi:

```
000002a0: 0400 0000 0000 0000 51e5 7464 0600 0000  ........Q.td....
```

to this line:

```
000002a0: 0400 0000 0000 0000 51e5 7464 0700 0000  ........Q.td....
```

so just change 6 to 7 and use xxd to get the binary back of this hex dump (this command is applied after changing 6 to 7 in hi_hex_dump):

```
[@majd 5]$ xxd -r hi_hex_dump hi
```

Did that make hi stack executable? We need to check the content of /proc/<hi process id>/maps. The stack should be marked executable:

```
[@majd proc]$ ps aux|grep hi
.
.
.
```

```
majd         5548  0.0  0.0   2364   688 pts/0    S+   14:11   0:00 ./hi
majd         5626  0.0  0.1   6804  2480 pts/1    S+   14:13   0:00 grep hi
```

We see that hi process id is 5548:

```
[@majd proc]$ cd "/proc/5548"
[@majd 5548]$ cat maps
...
...
...

7fff5293a000-7fff5295b000 rwxp 00000000 00:00 0                          [stack]
7fff52977000-7fff5297b000 r--p 00000000 00:00 0                          [vvar]
...
...
...
```

The stack has an x in its permissions (rwxp), so it is executable!!

2 Hack the Machine Code

- Write a simple C program that runs a loop 1000 times. Compile it. Using readelf and a hex editor, modify the compiled program to execute this loop 2000 times instead.

```
I made 2 c programs:
/*
 * loop_1000.c
 */

#include <stdio.h>
#include <unistd.h>


int main(int argc, char *argv[])
{
    for (int i = 0; i < 1000; i++){
      printf("%d\n", "i" );

    }
}
```

and:

```
/*
 * loop_2000.c
 */

```

```
#include <stdio.h>
#include <unistd.h>


int main(int argc, char *argv[])
{
    for (int i = 0; i < 2000; i++){
      printf("%d\n", "i" );

    }
}
```

The intuition is to produce the readelf of each programs and diff them:

```
[@majd 5]$ readelf -a loop_1000 > loop_1000_elf_info
[@majd 5]$ readelf -a loop_2000 > loop_2000_elf_info
[@majd 5]$ diff loop_1000_elf_info loop_2000_elf_info
```

the output is:

```
[majd@majd 5]$ diff loop_1000_elf_info loop_2000_elf_info
222c222
<      12: 0000000000000000      0 FILE    LOCAL   DEFAULT   ABS loop_1000.c
---
>      12: 0000000000000000      0 FILE    LOCAL   DEFAULT   ABS loop_2000.c
272c272
<      Build ID: 6c9e1227b7ad3fdd3946d609ba60d47b729aebd8
---
>      Build ID: ce65d225c2ea0d7d9adf93d89450a202a5ef1261
```

Its only displaying that the difference between the two readelf files is the name of the two programs and their build number. This is not very useful and does not reflect the actual differences between the tow program. Readelf is only putting the elf info in human readable form, but not necessarily all the info. So, we produced a hex dump of the two compiled programs using xxd then used diff on them:

```
[@majd 5]$ xxd loop_1000 > loop_1000_hex_dump
[@majd 5]$ xxd loop_2000 > loop_2000_hex_dump
[@majd 5]$ diff loop_1000_hex_dump loop_2000_hex_dump
57,58c57,58
< 00000380: 0300 0000 474e 5500 6c9e 1227 b7ad 3fdd  ....GNU.l..'..?.
< 00000390: 3946 d609 ba60 d47b 729a ebd8 0400 0000  9F...‘.{r.......
---
> 00000380: 0300 0000 474e 5500 ce65 d225 c2ea 0d7d  ....GNU..e.%...}
> 00000390: 9adf 93d8 9450 a202 a5ef 1261 0400 0000  .....P.....a....
279c279
< 00001160: 8345 fc01 817d fce7 0300 007e e4b8 0000  .E...}.....~....
---
> 00001160: 8345 fc01 817d fccf 0700 007e e4b8 0000  .E...}.....~....
777c777
< 00003080: 0000 001d 0000 0000 0c00 0000 3911 0000  ............9...
```

```
---
> 00003080: 0000 001d 1a00 0000 0000 0000 3911 0000  ............9...
799c799
< 000031e0: 0c00 0000 0201 1f02 0f02 0000 0000 0000  ................
---
> 000031e0: 0000 0000 0201 1f02 0f02 1a00 0000 001a  ................
813,815c813,815
< 000032c0: 6c6f 6f70 5f31 3030 302e 6300 2f68 6f6d  loop_1000.c./hom
< 000032d0: 652f 6d61 6a64 2f44 6573 6b74 6f70 2f4c  e/majd/Desktop/L
< 000032e0: 6162 732f 3500 0000 0000 0000 0000 0000  abs/5...........
---
> 000032c0: 2f68 6f6d 652f 6d61 6a64 2f44 6573 6b74  /home/majd/Deskt
> 000032d0: 6f70 2f4c 6162 732f 3500 6c6f 6f70 5f32  op/Labs/5.loop_2
> 000032e0: 3030 302e 6300 0000 0000 0000 0000 0000  000.c...........
888c888
< 00003770: 006c 6f6f 705f 3130 3030 2e63 005f 5f46  .loop_1000.c.__F
---
> 00003770: 006c 6f6f 705f 3230 3030 2e63 005f 5f46  .loop_2000.c.__F
```

This looks promising! But what are we looking for? obviously where 1000 and 2000 show up. This output is in hex, so we have to look for their hex representation 3E8, 7D0. After some careful searching we find them in line 0x1160:

```
< 00001160: 8345 fc01 817d fce7 0300 007e e4b8 0000  .E...}.....~....
---
> 00001160: 8345 fc01 817d fccf 0700 007e e4b8 0000  .E...}.....~....
```

they are written in hex in reverse and, in fact, substracted by 1. Instead 3E8, we see fce7 0300 and instead of 7D0, we find fccf 0700. 07 fc = 7D0 - 1 and 03 e7 = 3e8 -1. Great! Now all we have to do to change that line from fce7 0300 to fccf 0700 in loop_1000_hex_dump, then use xxd to revert it from hex dump to binary (this code is applied after the changes are made):

```
[@majd 5]$ xxd -r loop_1000_hex_dump loop_1000
[@majd 5]$ ./loop_1000
...
...
...
1998
1999
```

success!

3 Explore libc #1

- Recall that within libc, everything stays in the same (relative) locations. Assume that you have a way to find the address of the sleep function within a running process. Write a program (a shell script will likely be easiest) that takes as input this address

> and calculates the address of the system function within the same process. It should automate the process of looking up the addresses of the functions within libc; it should not canonicalize the particular relative addresses of the two functions within the current installed instance.

The idea here is that we can find any libc function if we know the address of one function in libc since all the functions stay relativly at the same location. So, it the system function is stored at 0x16 spaces away from the sleep function at my computer, it will be stored at 0x16 spaces at any computer. If I know where the sleep function is stored at someone else's computer, I can also know where their system function-or any libc function- is stored. All we need to do now is just find the difference between the sleep address and system address. There are two ways of going about this: we can either write a c program with sleep and system and print their addresses in gdb then substract them or just go to a program using libc (mostly all processes running) and objdump the libc file to see where these two functions are stored.

Lets starts with the libc method. The steps are: 1- find a process, go to its maps file and find the file it is using to get libc, then objdump that file:

```
majd@majd 5]$ cd /proc
[majd@majd proc]$ ls
1     13    186   249  29   344  404  517   74   85    966
   diskstats      keys           net            thread-self
10    1361  2     25   291  345  408  518   75   87    993         dma
          key-users      pagetypeinfo   timer_list
1003  139   20    250  292  346  413  521   750  88    acpi        driver
          kmsg           partitions     tty
1013  14    200   251  293  349  435  522   76   89    asound
   dynamic_debug  kpagecgroup    pressure       uptime
1020  140   21    252  294  356  436  595   77   90    bootconfig
   execdomains    kpagecount     schedstat      version
103   141   2151  253  3    359  442  6     78   91    buddyinfo   fb
          kpageflags     scsi           vmallocinfo
104   143   22    254  30   365  443  620   79   915   bus
   filesystems    latency_stats  self           vmstat
105   15    221   255  301  369  444  669   8    93    cgroups     fs
          loadavg        slabinfo       zoneinfo
1055  153   222   256  312  375  453  6854  80   94    cmdline
   interrupts     locks          softirqs
106   154   226   259  313  376  486  6947  81   940   config.gz   iomem
          meminfo        stat
107   16    23    26   334  392  492  7128  810  958   consoles    ioports
          misc           swaps
108   1672  24    262  335  394  496  7139  813  960   cpuinfo     irq
          modules        sys
11    17    246   27   337  4    502  7217  82   963   crypto      kallsyms
          mounts         sysrq-trigger
12    18    248   28   338  401  503  7248  84   964   devices     kcore
          mtrr           sysvipc
```

```
[majd@majd proc]$ cd 248
[majd@majd 248]$ grep libc maps
[majd@majd 248]$ cd ..
[majd@majd proc]$ cd 338
[majd@majd 338]$ grep libc maps
grep: maps: Permission denied
[majd@majd 338]$ cd ..
[majd@majd proc]$ cd 401
[majd@majd 401]$ grep libc maps
7f3838814000-7f3838817000 r--p 00000000 08:01 813679                     /usr/lib/libcanberra.so.0.2.5
7f3838817000-7f3838821000 r-xp 00003000 08:01 813679                     /usr/lib/libcanberra.so.0.2.5
7f3838821000-7f3838825000 r--p 0000d000 08:01 813679                     /usr/lib/libcanberra.so.0.2.5
7f3838825000-7f3838826000 ---p 00011000 08:01 813679                     /usr/lib/libcanberra.so.0.2.5
7f3838826000-7f3838827000 r--p 00011000 08:01 813679                     /usr/lib/libcanberra.so.0.2.5
7f3838827000-7f3838828000 rw-p 00012000 08:01 813679                     /usr/lib/libcanberra.so.0.2.5
7f3838828000-7f383882a000 r--p 00000000 08:01 813676                     /usr/lib/libcanberra-gtk3.so.0.1.9
7f383882a000-7f383882c000 r-xp 00002000 08:01 813676                     /usr/lib/libcanberra-gtk3.so.0.1.9
7f383882c000-7f383882d000 r--p 00004000 08:01 813676                     /usr/lib/libcanberra-gtk3.so.0.1.9
7f383882d000-7f383882e000 r--p 00004000 08:01 813676                     /usr/lib/libcanberra-gtk3.so.0.1.9
7f383882e000-7f383882f000 rw-p 00005000 08:01 813676                     /usr/lib/libcanberra-gtk3.so.0.1.9
7f3838ce2000-7f3838ce5000 r--p 00000000 08:01 799306                     /usr/lib/libcap.so.2.62
7f3838ce5000-7f3838cea000 r-xp 00003000 08:01 799306                     /usr/lib/libcap.so.2.62
7f3838cea000-7f3838cec000 r--p 00008000 08:01 799306                     /usr/lib/libcap.so.2.62
7f3838cec000-7f3838ced000 r--p 00009000 08:01 799306                     /usr/lib/libcap.so.2.62
7f3838ced000-7f3838cee000 rw-p 0000a000 08:01 799306                     /usr/lib/libcap.so.2.62
7f383baac000-7f383bab3000 r--p 00000000 08:01 814488                     /usr/lib/libcloudproviders.so.0.3.1
7f383bab3000-7f383babc000 r-xp 00007000 08:01 814488                     /usr/lib/libcloudproviders.so.0.3.1
7f383babc000-7f383bac1000 r--p 00010000 08:01 814488                     /usr/lib/libcloudproviders.so.0.3.1
7f383bac1000-7f383bac2000 ---p 00015000 08:01 814488                     /usr/lib/libcloudproviders.so.0.3.1
7f383bac2000-7f383bac3000 r--p 00015000 08:01 814488                     /
```

```
    usr/lib/libcloudproviders.so.0.3.1
7f383bac3000-7f383bac4000 rw-p 00016000 08:01 814488                          /
    usr/lib/libcloudproviders.so.0.3.1
7f383bfac000-7f383bfbe000 r--p 00000000 08:01 811326                          /
    usr/lib/libcairo.so.2.11704.0
7f383bfbe000-7f383c090000 r-xp 00012000 08:01 811326                          /
    usr/lib/libcairo.so.2.11704.0
7f383c090000-7f383c0c5000 r--p 000e4000 08:01 811326                          /
    usr/lib/libcairo.so.2.11704.0
7f383c0c5000-7f383c0c9000 r--p 00118000 08:01 811326                          /
    usr/lib/libcairo.so.2.11704.0
7f383c0c9000-7f383c0ca000 rw-p 0011c000 08:01 811326                          /
    usr/lib/libcairo.so.2.11704.0
7f383c0cd000-7f383c0d1000 r--p 00000000 08:01 811320                          /
    usr/lib/libcairo-gobject.so.2.11704.0
7f383c0d1000-7f383c0d3000 r-xp 00004000 08:01 811320                          /
    usr/lib/libcairo-gobject.so.2.11704.0
7f383c0d3000-7f383c0d5000 r--p 00006000 08:01 811320                          /
    usr/lib/libcairo-gobject.so.2.11704.0
7f383c0d5000-7f383c0d6000 ---p 00008000 08:01 811320                          /
    usr/lib/libcairo-gobject.so.2.11704.0
7f383c0d6000-7f383c0d8000 r--p 00008000 08:01 811320                          /
    usr/lib/libcairo-gobject.so.2.11704.0
7f383c0d8000-7f383c0d9000 rw-p 0000a000 08:01 811320                          /
    usr/lib/libcairo-gobject.so.2.11704.0
7f383c26e000-7f383c294000 r--p 00000000 08:01 789845                          /
    usr/lib/libc-2.33.so
7f383c294000-7f383c3df000 r-xp 00026000 08:01 789845                          /
    usr/lib/libc-2.33.so
7f383c3df000-7f383c42b000 r--p 00171000 08:01 789845                          /
    usr/lib/libc-2.33.so
7f383c42b000-7f383c42e000 r--p 001bc000 08:01 789845                          /
    usr/lib/libc-2.33.so
7f383c42e000-7f383c431000 rw-p 001bf000 08:01 789845                          /
    usr/lib/libc-2.33.so
7f383d3d9000-7f383d3db000 r--p 00000000 08:01 813661                          /
    usr/lib/gtk-3.0/modules/libcanberra-gtk3-module.so
7f383d3db000-7f383d3de000 r-xp 00002000 08:01 813661                          /
    usr/lib/gtk-3.0/modules/libcanberra-gtk3-module.so
7f383d3de000-7f383d3df000 r--p 00005000 08:01 813661                          /
    usr/lib/gtk-3.0/modules/libcanberra-gtk3-module.so
7f383d3df000-7f383d3e0000 r--p 00005000 08:01 813661                          /
    usr/lib/gtk-3.0/modules/libcanberra-gtk3-module.so
7f383d3e0000-7f383d3e1000 rw-p 00006000 08:01 813661                          /
    usr/lib/gtk-3.0/modules/libcanberra-gtk3-module.so
```

I had to try with multiple processes ids until I found a process that uses libc and I have permission to see its maps file. The 248 process worked. In the maps file, we find that this process is using libc from:

```
...
```

```
...
...
7f383c294000-7f383c3df000 r-xp 00026000 08:01 789845                     /usr/lib/libc-2.33.so
7f383c3df000-7f383c42b000 r--p 00171000 08:01 789845                     /usr/lib/libc-2.33.so
7f383c42b000-7f383c42e000 r--p 001bc000 08:01 789845                     /usr/lib/libc-2.33.so
7f383c42e000-7f383c431000 rw-p 001bf000 08:01 789845                     /usr/lib/libc-2.33.so
...
...
...
```

so the libc file is just "/usr/lib/libc-2.33.so". Now, we just have to objdump it. The output is huge, so we just objdump it and then grep sleep and system:

```
[@majd 401]$ objdump -j .text -d "/usr/lib/libc-2.33.so" | grep system
0000000000049840 <do_system>:
   49898: 0f 85 5a 03 00 00     jne    49bf8 <do_system+0x3b8>
   498bb: 0f 84 5f 02 00 00     je     49b20 <do_system+0x2e0>
   498cb: 0f 85 4f 03 00 00     jne    49c20 <do_system+0x3e0>
   4991b: 74 0c                 je     49929 <do_system+0xe9>
   499b8: 0f 85 aa 00 00 00     jne    49a68 <do_system+0x228>
   499e8: 0f 85 02 01 00 00     jne    49af0 <do_system+0x2b0>
   49a09: eb 16                 jmp    49a21 <do_system+0x1e1>
   49a1f: 75 0f                 jne    49a30 <do_system+0x1f0>
   49a2e: 74 e0                 je     49a10 <do_system+0x1d0>
   49a34: 74 08                 je     49a3e <do_system+0x1fe>
   49a41: 74 2d                 je     49a70 <do_system+0x230>
   49a60: eb 0e                 jmp    49a70 <do_system+0x230>
   49a7a: 0f 85 18 01 00 00     jne    49b98 <do_system+0x358>
   49a93: 0f 84 bf 00 00 00     je     49b58 <do_system+0x318>
   49aa3: 0f 85 17 01 00 00     jne    49bc0 <do_system+0x380>
   49ab2: 74 0a                 je     49abe <do_system+0x27e>
   49ad3: 0f 85 78 01 00 00     jne    49c51 <do_system+0x411>
   49b1b: e9 e4 fe ff ff        jmp    49a04 <do_system+0x1c4>
   49b50: e9 6c fd ff ff        jmp    498c1 <do_system+0x81>
   49b8d: e9 07 ff ff ff        jmp    49a99 <do_system+0x259>
   49ba7: 0f 84 df fe ff ff     je     49a8c <do_system+0x24c>
   49bb9: e9 ce fe ff ff        jmp    49a8c <do_system+0x24c>
   49bcb: 0f 8e df fe ff ff     jle    49ab0 <do_system+0x270>
   49bec: e9 bf fe ff ff        jmp    49ab0 <do_system+0x270>
   49c07: 0f 84 9d fc ff ff     je     498aa <do_system+0x6a>
   49c19: e9 8c fc ff ff        jmp    498aa <do_system+0x6a>
   49c2b: 0f 8e a7 fc ff ff     jle    498d8 <do_system+0x98>
   49c4c: e9 87 fc ff ff        jmp    498d8 <do_system+0x98>
0000000000049de0 <__libc_system>:
   49de7: 74 07                 je     49df0 <__libc_system+0x10>
   49de9: e9 52 fa ff ff        jmp    49840 <do_system>
   49dfb: e8 40 fa ff ff        call   49840 <do_system>
```

```
000000000012e2e0 <svcerr_systemerr@GLIBC_2.2.5>:
  12e33b: 75 05                jne    12e342 <svcerr_systemerr@GLIBC_2
   .2.5+0x62>
[majd@majd 401]$ objdump -j .text -d "/usr/lib/libc-2.33.so" | grep sleep
00000000000864d0 <thrd_sleep>:
   864e2: e8 39 05 04 00       call   c6a20 <clock_nanosleep@@GLIBC_2.17>
   864e9: 74 0c                je     864f7 <thrd_sleep+0x27>
00000000000c6a20 <clock_nanosleep@@GLIBC_2.17>:
   c6a27: 74 27                je     c6a50 <clock_nanosleep@@GLIBC_2
   .17+0x30>
   c6a41: 75 1d                jne    c6a60 <clock_nanosleep@@GLIBC_2
   .17+0x40>
00000000000cbb70 <sleep>:
   cbba9: e8 b2 00 00 00       call   cbc60 <__nanosleep>
   cbbb0: 78 1e                js     cbbd0 <sleep+0x60>
   cbbc5: 75 0e                jne    cbbd5 <sleep+0x65>
   cbbd3: eb e2                jmp    cbbb7 <sleep+0x47>
00000000000cbc60 <__nanosleep>:
   cbc72: e8 a9 ad ff ff       call   c6a20 <clock_nanosleep@@GLIBC_2.17>
   cbc79: 75 05                jne    cbc80 <__nanosleep+0x20>
   cbc8f: eb ea                jmp    cbc7b <__nanosleep+0x1b>
00000000000f6a50 <usleep>:
   f6a94: e8 c7 51 fd ff       call   cbc60 <__nanosleep>
   f6aa7: 75 05                jne    f6aae <usleep+0x5e>
  109de5: e8 86 1d fc ff       call   cbb70 <sleep>
  114212: e8 59 79 fb ff       call   cbb70 <sleep>
  114e6e: e8 fd 6c fb ff       call   cbb70 <sleep>
  12e9ce: e8 8d d2 f9 ff       call   cbc60 <__nanosleep>
  12ea46: e8 15 d2 f9 ff       call   cbc60 <__nanosleep>
```

The functions we are looking at are libc_system which is stored at 0x49de0 and the sleep function which is stored at 0xcbb70. The difference between them is 0x81D90, or system address = sleep address - 0x81D90.

Can we check this? yes, using the second method. Just write a c program with both functions, gdb it and print sleep and system addresses. The c program is:

```c
/*
 * sleep.c
 */

#include <stdio.h>
#include <unistd.h>


int main(int argc, char *argv[])
{
    sleep(1);
    system("ls");
    printf("%s\n", "done" );
}
```

The make file is:

```
sleep: sleep.c
  gcc -g -fno-stack-protector -o sleep sleep.c



.PHONY: clean
clean:
  rm -f sleep
```

Now we gdb it:

```
[@majd /]$ cd home/majd/Desktop/Labs/5
[@majd 5]$ make
make: ’sleep’ is up to date.
[@majd 5]$ gdb sleep
GNU gdb (GDB) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sleep...
(gdb) b sleep
Breakpoint 1 at 0x1050
(gdb) run
Starting program: /home/majd/Desktop/Labs/5/sleep

Breakpoint 1, 0x00007ffff7eb7b70 in sleep () from /usr/lib/libc.so.6
(gdb) p sleep
$1 = {<text variable, no debug info>} 0x7ffff7eb7b70 <sleep>
(gdb) p system
$2 = {<text variable, no debug info>} 0x7ffff7e35de0 <system>
(gdb)
```

sleep is stored at 0x7ffff7eb7b70 and system is stored at 0x7ffff7e35de0 and they are 0x81D90 apart with sleep being at the higher address!

Great! Now we write the shell program:

```
#!/bin/bash
#! find_system.bash
```

```

read sleep_address
printf "system address is 0x%X\n" $(( $sleep_address - 0x81D90))
```

if we run the program and provide sleep address from gdb: 0x7ffff7eb7b70, we get the correct system address:

```
[@majd 5]$ bash find_system.bash
0x7ffff7eb7b70
system address is 0x7FFFF7E35DE0
```

success!

4 Explore libc #2 & Make a ROP chain & Do something useful with ROP

We mixed the above three section in one excersise after reading https://hovav.net/ucsd/dist/geometry.pdf. For the purpose of these exercises, we choose to add to integer variables in a c code and to change the value of a variable that controls the security of a program. Take the following c program for example:

```c
/*
 * vul.c
 */

#include <stdio.h>
#include <unistd.h>

void over_flow(void) {
    char name[8];
    int isAdmin = 0;
    int x = 1;
    int y = 1;

    printf("Enter your exploit: ");
    fflush(stdout);
    read(0, name, 100);

    printf("Exploit loaded!\n");
}

int main(int argc, char *argv[])
{

  over_flow();

}
```

We want to change the value of isAdmin to 1. To store to memory, we need the gadget movl %eax, 24(%edx); ret where we store the content of eax into the memory address stored in edx (the address we actually want to store to is edx + 24 because of the offset of movl). To load from memory, we need the gadget movl 64(%eax), %eax; where we load the content of the memory address at eax+64 to eax.

How do we find these gadgets in memory? Well, they must be found in libc (or any other library the program is using) so we must find where libc is first. We can do that in gdb using "info proc map":

```
[majd@majd 5]$ gdb vul
GNU gdb (GDB) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vul...
(gdb) b main
Breakpoint 1 at 0x11e0: file vul.c, line 24.
(gdb) run
Starting program: /home/majd/Desktop/Labs/5/vul

Breakpoint 1, main (argc=1, argv=0x7fffffffe9a8) at vul.c:24
24      over_flow();
(gdb) info proc map
process 12641
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
      0x555555554000     0x555555555000     0x1000        0x0 /home/majd/Desktop/Labs/5/vul
      0x555555555000     0x555555556000     0x1000     0x1000 /home/majd/Desktop/Labs/5/vul
      0x555555556000     0x555555557000     0x1000     0x2000 /home/majd/Desktop/Labs/5/vul
      0x555555557000     0x555555558000     0x1000     0x2000 /home/majd/Desktop/Labs/5/vul
      0x555555558000     0x555555559000     0x1000     0x3000 /home/majd/Desktop/Labs/5/vul
      0x7ffff7dea000      0x7ffff7dec000     0x2000        0x0
      0x7ffff7dec000      0x7ffff7e12000    0x26000        0x0 /usr/lib/
   libc-2.33.so
      0x7ffff7e12000      0x7ffff7f5d000   0x14b000    0x26000 /usr/lib/
   libc-2.33.so
      0x7ffff7f5d000      0x7ffff7fa9000    0x4c000   0x171000 /usr/lib/
   libc-2.33.so
      0x7ffff7fa9000      0x7ffff7fac000     0x3000   0x1bc000 /usr/lib/
   libc-2.33.so
      0x7ffff7fac000      0x7ffff7faf000     0x3000   0x1bf000 /usr/lib/
   libc-2.33.so
      0x7ffff7faf000      0x7ffff7fba000     0xb000        0x0
      0x7ffff7fc7000      0x7ffff7fcb000     0x4000        0x0 [vvar]
      0x7ffff7fcb000      0x7ffff7fcd000     0x2000        0x0 [vdso]
      0x7ffff7fcd000      0x7ffff7fce000     0x1000        0x0 /usr/lib/ld
   -2.33.so
      0x7ffff7fce000      0x7ffff7ff2000    0x24000     0x1000 /usr/lib/ld
   -2.33.so
      0x7ffff7ff2000      0x7ffff7ffb000     0x9000    0x25000 /usr/lib/ld
   -2.33.so
      0x7ffff7ffb000      0x7ffff7ffd000     0x2000    0x2d000 /usr/lib/ld
   -2.33.so
      0x7ffff7ffd000      0x7ffff7fff000     0x2000    0x2f000 /usr/lib/ld
   -2.33.so
--Type <RET> for more, q to quit, c to continue without paging--Quit
```

we find libc starting at 0x7ffff7e12000 and ending at 0x7ffff7f5d000(the largest libc size):

```
0x7ffff7e12000      0x7ffff7f5d000   0x14b000    0x26000 /usr/lib/libc
   -2.33.so
```