ROP lab task

Vulnerable program is same as bof lab task, but enable the NX protection.
Note the system-wide ASLR remains disabled for our first task.

Vulnerable program:

```
#include <stdio.h>

void win() {  // at 0x08048456
        puts("Excellent, now let's go hack the world");
}

void vuln() {
        char buf[16];
        scanf("%s", buf);
}

int main() {
        puts("Welcome back to 2023 CS315, let's have some fun!");
        vuln();
        puts("Have a good day, Bye~");
        return 0;
}
```

0 WarmUp

Find the address of system, exit and "/bin/sh" in libc, you may choose one of the following methods:

- Using gdb:
  - adter run, use `p /x &system` or just `p system` to get the address of system and exit
  - use `find` command to find the address of exit and "/bin/sh", remeber we can use `info proc mappings` or `vmmap` to find the address of libc.

```
pwndbg> help find
Search memory for a sequence of bytes.
Usage:
find [/SIZE-CHAR] [/MAX-COUNT] START-ADDRESS, END-ADDRESS, EXPR1 [, EXPR2 ...]
find [/SIZE-CHAR] [/MAX-COUNT] START-ADDRESS, +LENGTH, EXPR1 [, EXPR2 ...]
SIZE-CHAR is one of b,h,w,g for 8,16,32,64 bit values respectively,
and if not specified the size is taken from the type of the expression
in the current language.
Note that this means for example that in the case of C-like languages
a search for an untyped 0x42 will search for "(int) 0x42"
which is typically four bytes, and a search for a string "hello" will
include the trailing '\0'.  The null terminator can be removed from
searching by using casts, e.g.: {char[5]}"hello".

The address of the last match is stored as the value of "$_".
Convenience variable "$numfound" is set to the number of matches.
```

    we need to specify the range of address to search, for example:

```
pwndbg> find 0x[redacted], 0x[redacted], "/bin/sh"
0xf7[redacted]
1 pattern found.
pwndbg> x /s 0xf7[redacted]
0xf7[redacted]:     "/bin/sh"
```

- Using pwntools:
  - use `ldd` to find which libc is used, for example: `ldd ret2libc`, ldd can also tell you base address of libc.
  - use pwntools to read exported symbols from libc and search the address of "/bin/sh", for example:

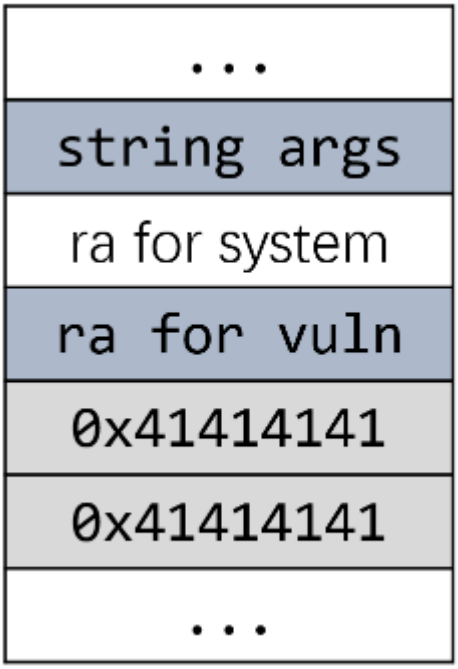
    ```
    from pwn import *

    libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
    print(f"system: {hex(libc.symbols['system'])}")
    print(f"/bin/sh: {hex(next(libc.search(b'/bin/sh')))}")
    ```

    note that this will find load offset of system, exit and "/bin/sh", you need to **add the base address** of libc to get the real address.

1. Lab Task

1.1 Use ROP to get the shell access.

construct payload to make stack like this:



here is a reference to generete payload file in C:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```c
int main(int argc, char **argv){
  char buf[48];
  FILE *badfile;
  badfile = fopen("./payload", "w");
  /* You need to decide the addresses and
  the values for X, Y, Z. The order of the following
  three statements does not imply the order of X, Y, Z.
  Actually, we intentionally scrambled the order. */
  *(long *) &buf[X] = some address ; // "/bin/sh"
  *(long *) &buf[Y] = some address ; // system()
  *(long *) &buf[Z] = some address ; // exit()
  fwrite(buf, sizeof(buf), 1, badfile);
  fclose(badfile);
}
```

in python, we can use below code to generate payload **byte string**

```python
from pwn import *

payload = b'a' * ?? + p32(??) + p32(??) + p32(??)
with open('payload', 'wb') as f:
    f.write(payload)
```

2. Submit (Assume in total 100 points)

- 1.The screenshot and payload of: 35 pts)
  - find address for system and exit (10 pts)
  - find address for system, exit and "/bin/sh" (17 pts)
  - lunch shell (25 pts)
  - lunch shell and exit normally (full 35 pts)
- 2. Compile the program yourself and disable protection mechanism you think it can prevent ROP attack, then try to get the shell access, **briefly describe** the process. (20 pts)
- 3. Without changing code, choose three of the following protection mechanism can prevent or mitigate ROP attack? (30 pts, 10 pts each, **briefly explain** why)
  - Canary, ASLR, NX, SafeStack(https://clang.llvm.org/docs/SafeStack.html), Pointer-Authentication(https://llvm.org/docs/PointerAuth.html), RELRO, Other you know...
- 4. Enable system-wide ASLR, can you still get the shell access in `ret2libc`? (10 pts)
  - hint: leak address of libc https://tc.gts3.org/cs6265/2021/tut/tut06-02-advrop.html
    - we can call `puts(&puts)` to leak the address of libc, use `leak_address - offset_of_puts_in_libc` to get the base address of libc
    - If you choose this way, you may need to call a function in libc, then return to main function and overflow again
  - hint: return to dl-resolve exploit
    - This may spend more time to implement, and requires "stack pivoit" skill used in bonus part.
  - full points for exploit explaination

- 5. Challenge: there is a statically linked binary which has the same source code with `ret2libc` as `ret2libc_static`. Can you still get the shell access? (5 pts)
    - you may design your own payload or use tools like https://github.com/JonathanSalwan/ROPgadget (**ROPgadget is already istalled**), or another possible approch: SROP
    - in this time, you may need to call `read(0, some_address, 0x12345678)` first to avoid chopping the payload, then use `leave; ret` topivot the stack to your unconstrained payload chain.
        - Usually we can write the payload to `.bss` section, then return to `read` again to read the payload to `.bss` section, then pivot stack to `.bss` section.
        - read 0x999 or more does not mean you need to read full 0x999 bytes, you can read less bytes(end with \n is better) and sleep(wait) for a while, then read will reach a timeout and return.

Happy hacking!