Defeating ASLR

LIBC

A shorthand for the “standard C library”, a library of standard functions (usually referring to the GNU implementation)

Sample Code

```c
int main() {
    puts("Some leaks for you");
    printf("PRINTF@LIBC: %p\n", printf);
    printf("PUTS@LIBC: %p\n", puts);
}
```

```
$ ./leak
Some leaks for you
PRINTF@LIBC: 0xf7dc1f30
PUTS@LIBC: 0xf7de4190
$ ./leak
Some leaks for you
PRINTF@LIBC: 0xf7d75f30
PUTS@LIBC: 0xf7d98190
$ ./leak
Some leaks for you
PRINTF@LIBC: 0xf7d7df30
PUTS@LIBC: 0xf7da0190
```

Truly Random?

Notice how the pink section always remains the same?

It's only the green section that ASLR is affecting!

```
$ ./leak
Some leaks for you
PRINTF@LIBC: 0xf7dc1f30
PUTS@LIBC: 0xf7de4190
$ ./leak
Some leaks for you
PRINTF@LIBC: 0xf7d75f30
PUTS@LIBC: 0xf7d98190
$ ./leak
Some leaks for you
PRINTF@LIBC: 0xf7d7df30
PUTS@LIBC: 0xf7da0190
```

Truly Random?

0xf7de4190 - 0xf7dc1f30 = 0x22260

0xf7d98190 - 0xf7d75f30 = 0x22260

0xf7da0190 - 0xf7d7df30 = 0x22260

WHY?

The relative offsets between each symbol in LIBC is constant!

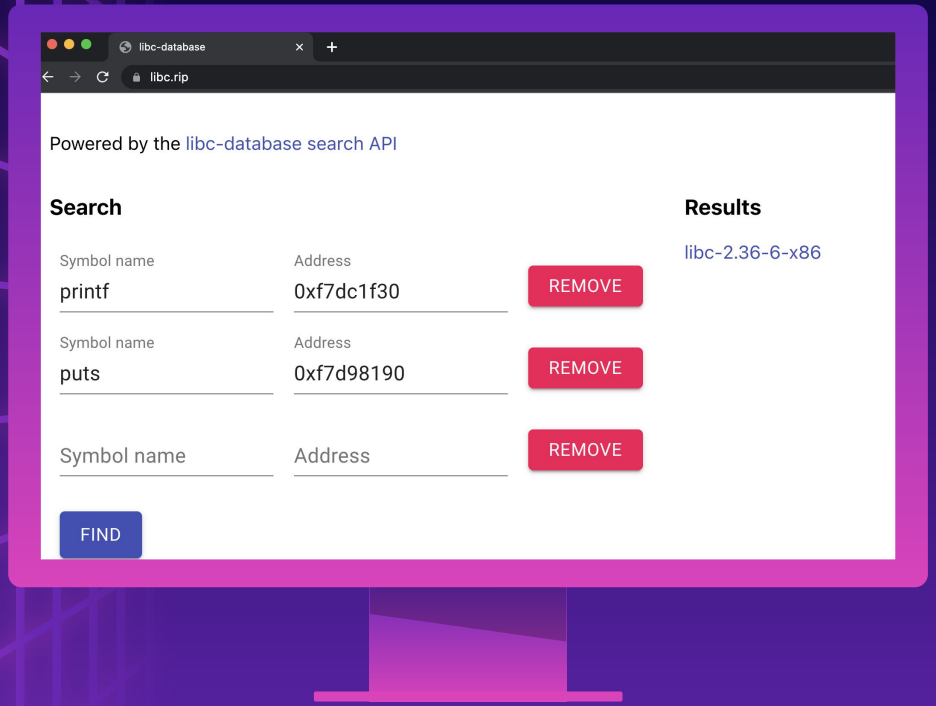
Differs from LIBC to LIBC

NOW WHAT?

If you can leak 1 symbol's address in LIBC, you can calculate the offsets of every other symbol

NOW WHAT?

If you can leak 2 or more symbols' addresses in LIBC, you can calculate which LIBC it uses



LIBC.RIP

The **leak** binary was using libc-2.36-6-x86

Process Linkage Table (PLT)

Used to call external functions whose address isn’t known at the time of linking

Global Offset Table (GOT)

Maps symbols in programming code to their corresponding **absolute** memory address
It’s loaded each time the program starts

~ Wikipedia-Kun

LIBC Function Call

function → call 0x8048380 <system@plt> → system@plt

system@plt → jmp DWORD PTR ds:0x804a00c → 0xf7d7b840

Location	Function Name	Address in LIBC
0x804a004	printf	0xf7d82f30
0x804a008	puts	0xf7da5190
0x804a00c	system	0xf7d7b840

PLT and GOT

```
pwndbg> disassemble vuln
...
0x080484d9 <+12>:     call   0x8048380 <gets@plt>
pwndbg> disassemble 0x8048380
Dump of assembler code for function gets@plt:
0x08048380 <+0>:      jmp    DWORD PTR ds:0x804a00c
...
pwndbg> disassemble 0x804a00c
0x804a00c <gets@got.plt>:    xchg   BYTE PTR
[ebx-0x7c69f7fc],al
```

Returning To LIBC

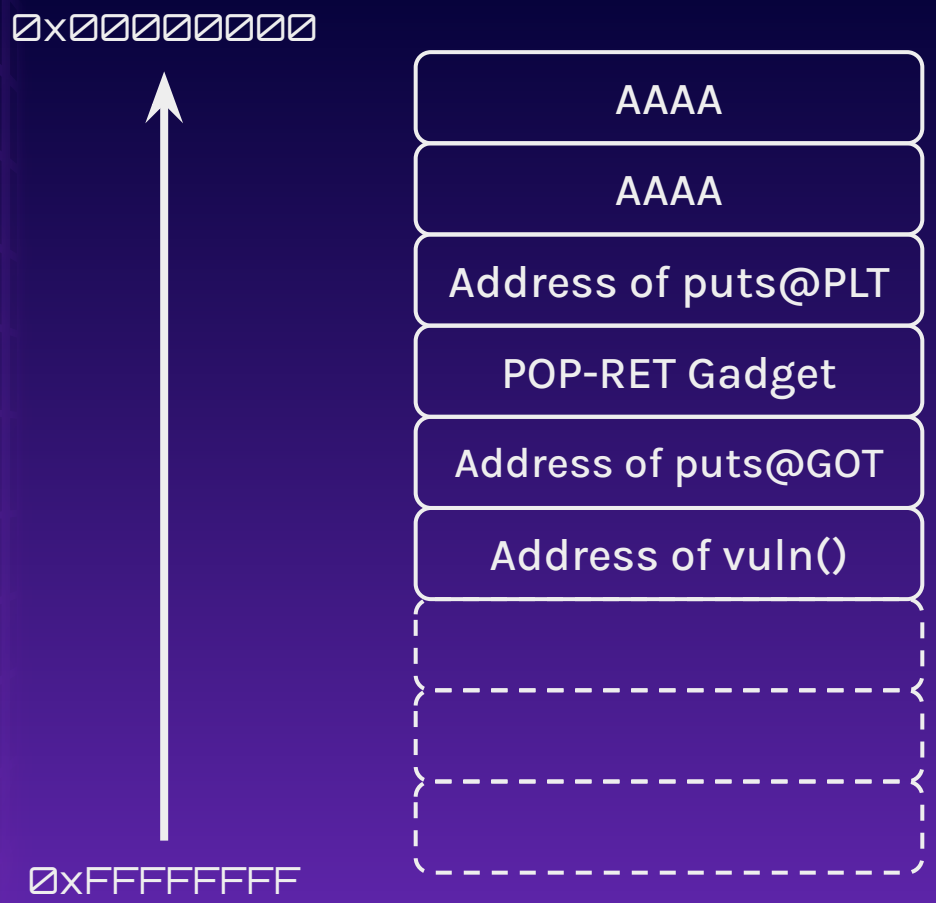
HOW?

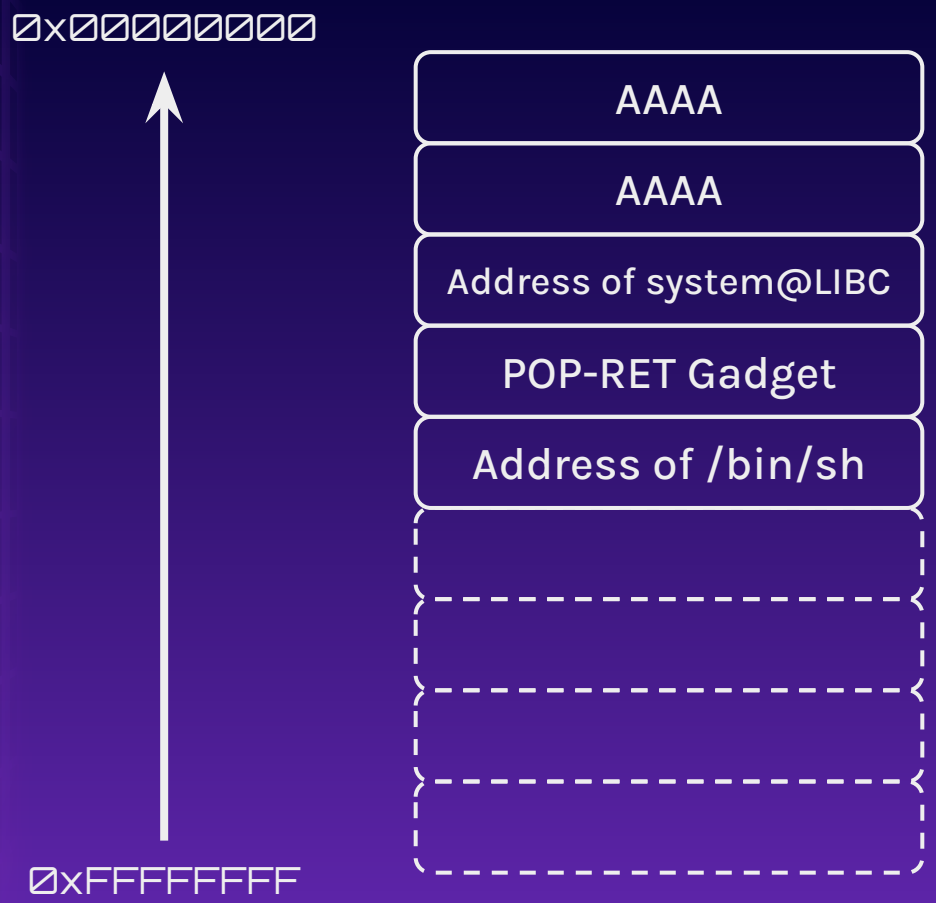
Leak the contents of the global offset table

Generate a ROP chain to call puts() and set the GOT as the parameter

HOW?

Chain the vuln() function again so you can insert another ROP chain


0x00000000
AAAA
AAAA
Address of puts@PLT
POP-RET Gadget
Address of puts@GOT
Address of vuln()
0xFFFFFFFF


0x00000000
AAAA
AAAA
Address of system@LIBC
POP-RET Gadget
Address of /bin/sh
0xFFFFFFFF

Use u32() which inverts p32()

```
>>> from pwn import *
>>> hex(u32(b"\x44\x43\x42\x41"))
'0x41424344'
```

Calculate LIBC Base Address

```
>>> from pwn import *
>>> libc = ELF("/usr/lib32/libc.so.6")
>>> libc.address = PUTS_LIBC -
libc.sym["puts"]
```

Generate ROP with multiple ELFs

```
>>> from pwn import *
>>> elf = ELF("./binaryfile")
>>> libc = elf.libc
>>> rop = ROP([elf, libc])
```

ret2libc.c

35 mins to pwn ret2libc32

Download files at:
http://ctfd.platypew.social

nc pwn.platypew.social 30006

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

void vuln() {
    char buffer[64];
    gets(buffer);
}

int main() {
    puts("Guess my name");
    vuln();
    puts("Wrong!");

    return 0;
}
```