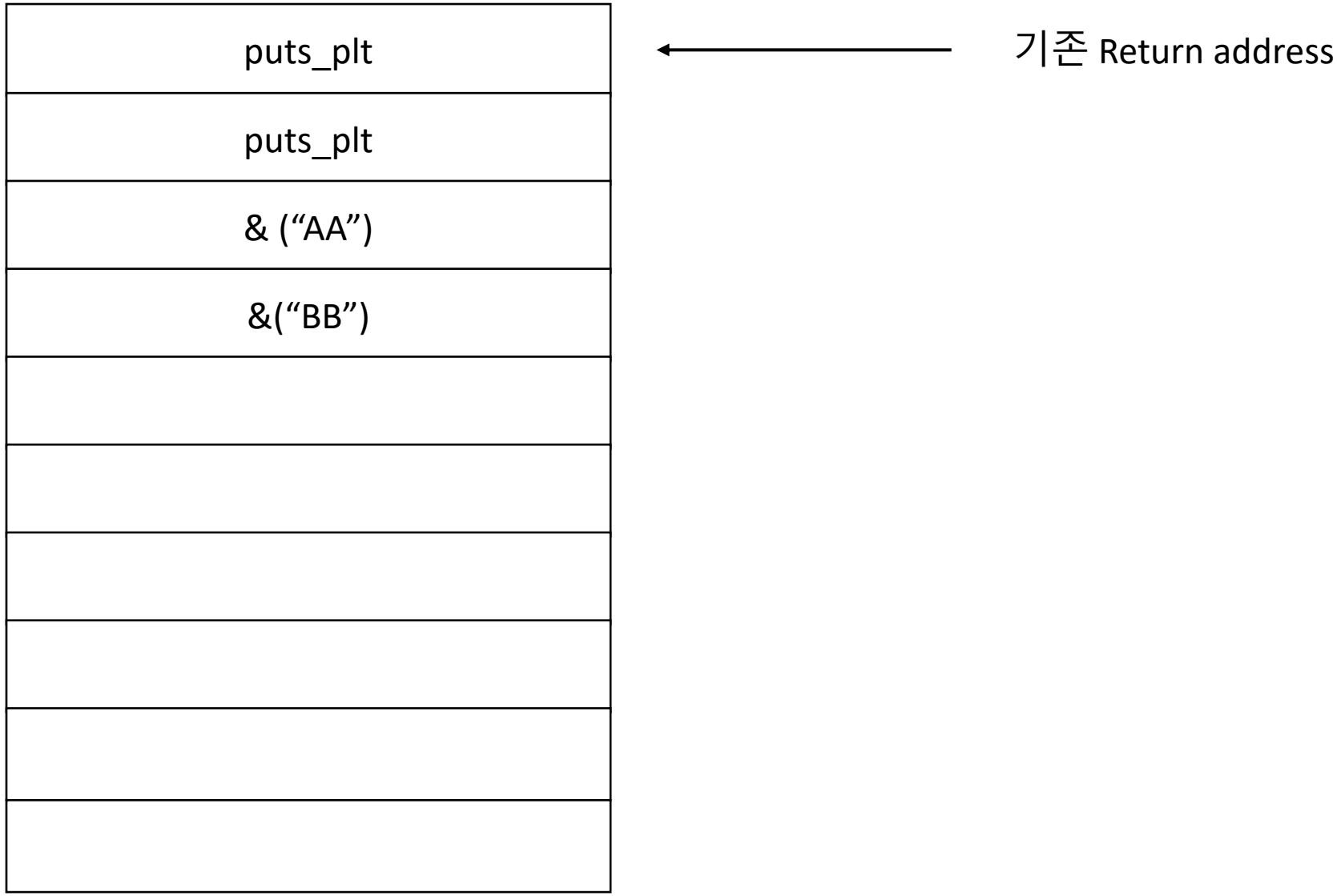
ROP

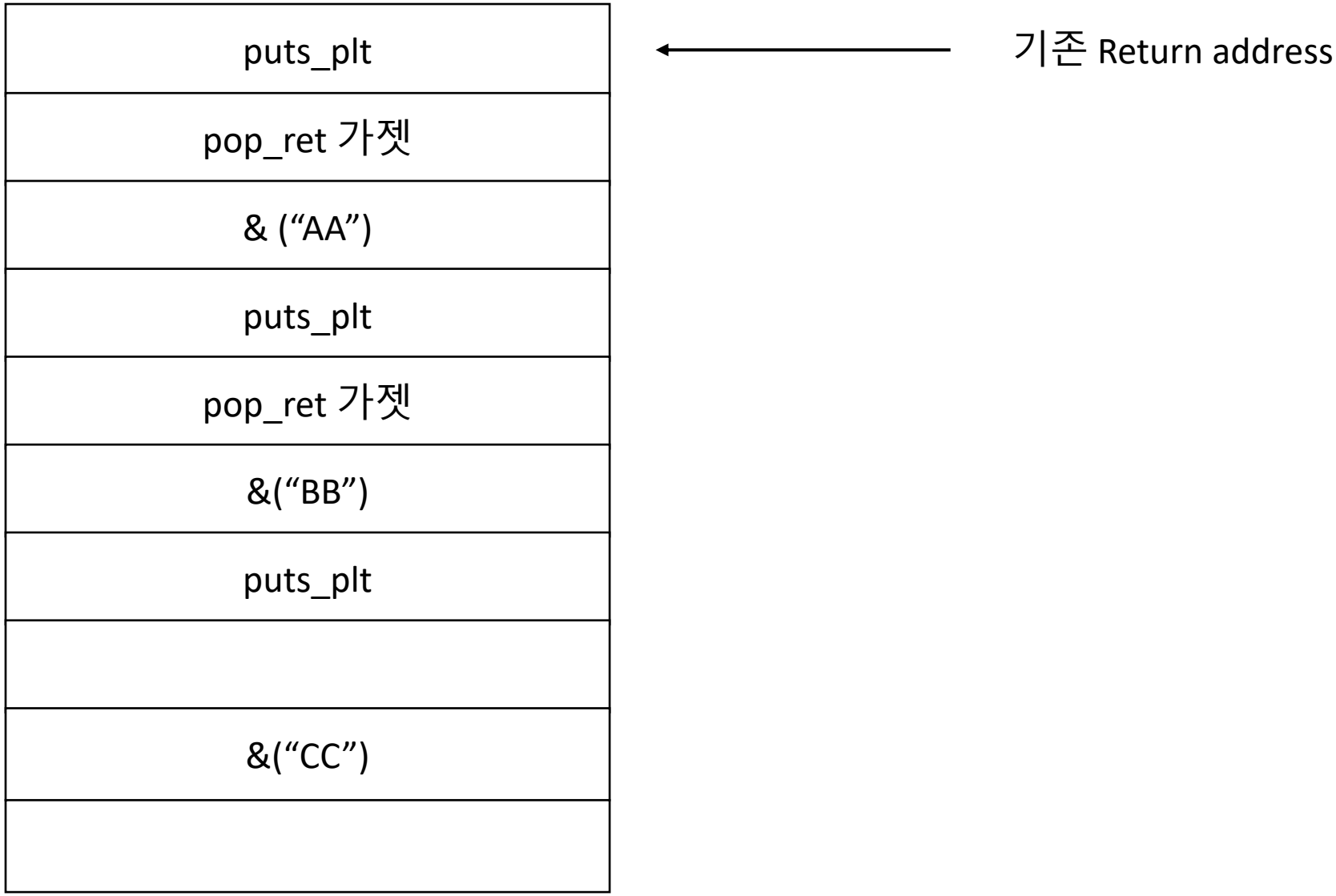
복습…

지금 하려는 것.
puts(“AA”);
puts(“BB”);



복습…

지금 하려는 것.
puts(“AA”);
puts(“BB”);
puts(“CC”);



ROP

- 지난 시간에 배운 것을 이용한 공격 기법
- 지난 시간에 배운 것 + libc_leak + Got_overwrite

- 우리의 목표 => get _ shell
- 코드 내에서 사용된 함수만 plt / got 가 존재.
- 만약 system 함수가 없으면 어떡하지??

- 우리의 목표 => get _ shell
- 코드 내에서 사용된 함수만 plt / got 가 존재.
- 만약 system 함수가 없으면 어떡하지??
- 실제 주소를 찾자!!!

- Dynamic linking

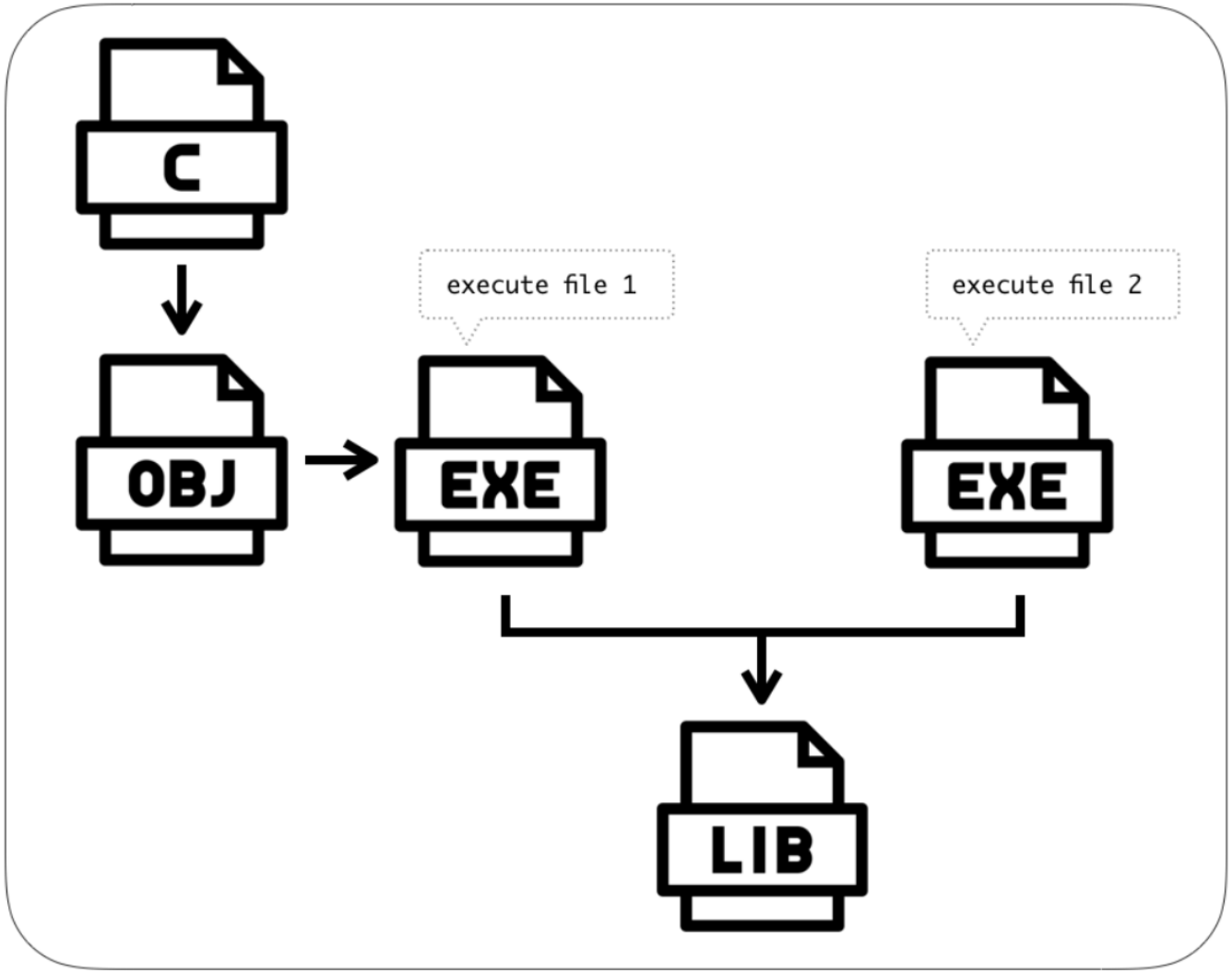


그림 출처 : https://bpsecblog.wordpress.com/2016/03/07/about_got_plt_1/

- 1. libc_leak

    목표 : libc_base를 구해야하는 것

- 1. libc_leak

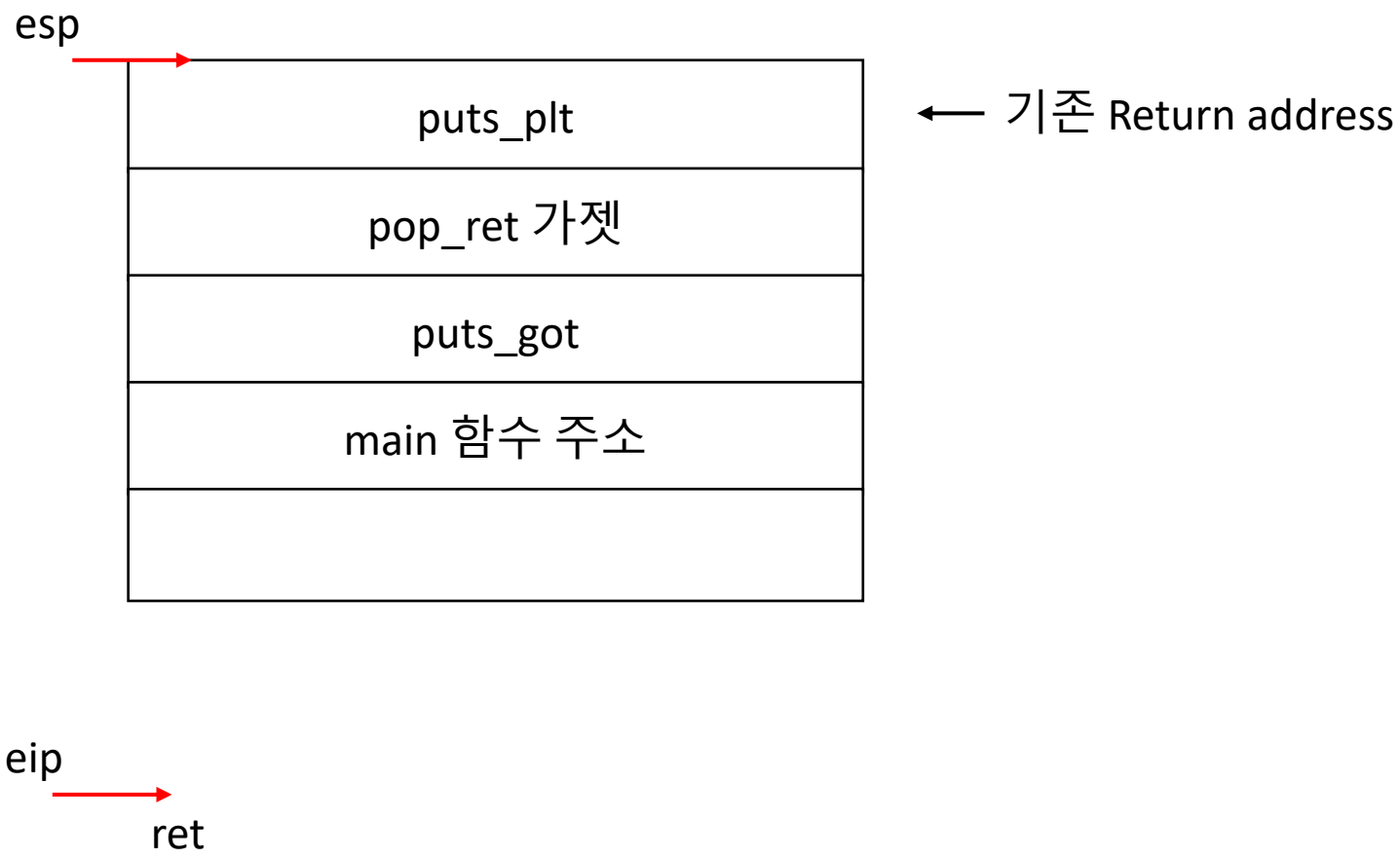
    목표 : libc_base를 구해야하는 것

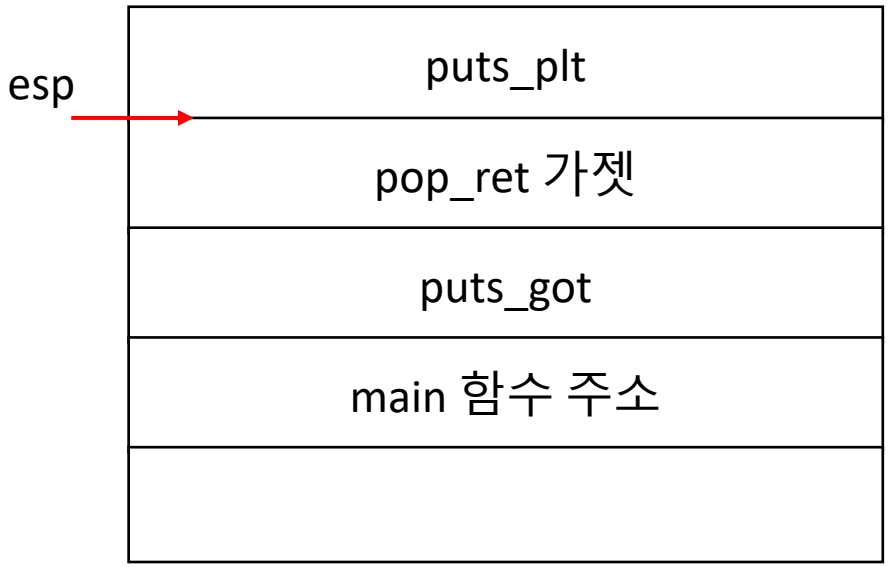
    방법 : put / write / printf / … 등등 출력 함수를 이용!!

- puts(puts_got)??

- puts(puts_got)??
- puts_got -> 실제 함수 주소(in libc)

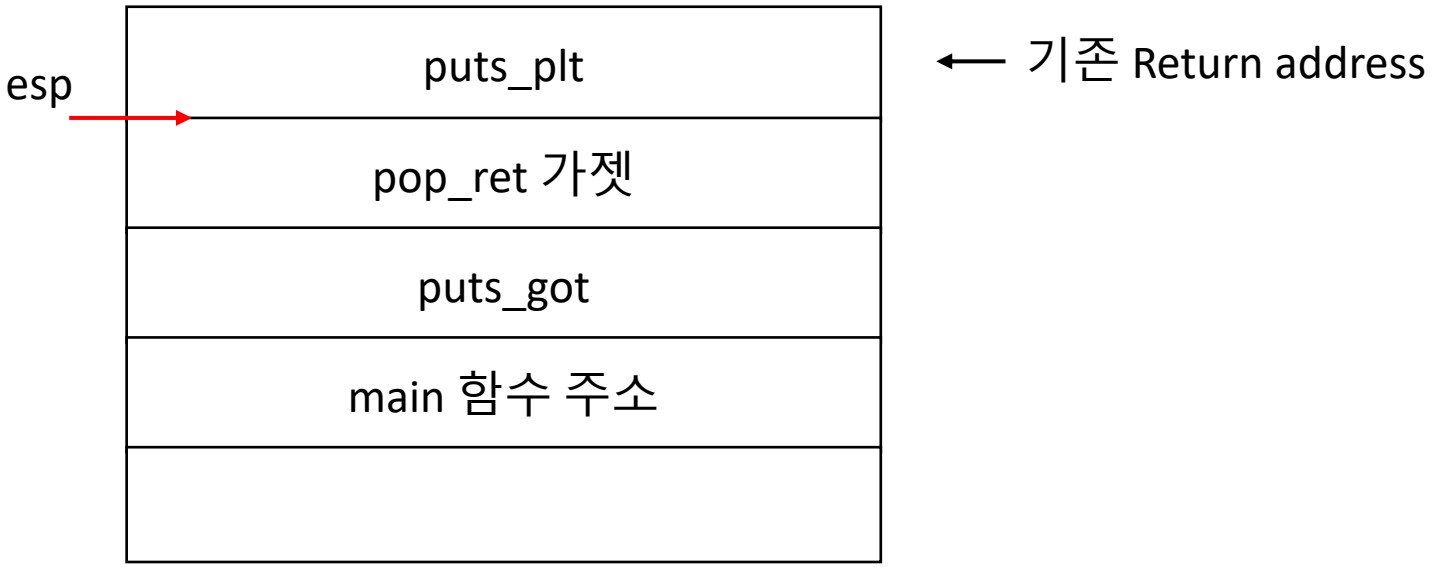
- puts(puts_got)??
- puts_got -> 실제 함수 주소(in libc)
- => 실제 함수 주소 출력!!!! (libc_base leak)


esp
puts_plt
pop_ret 가젯
puts_got
main 함수 주소
← 기존 Return address
eip
ret



← 기존 Return address

puts(puts_got); 실행과 동일!!



puts() 함수

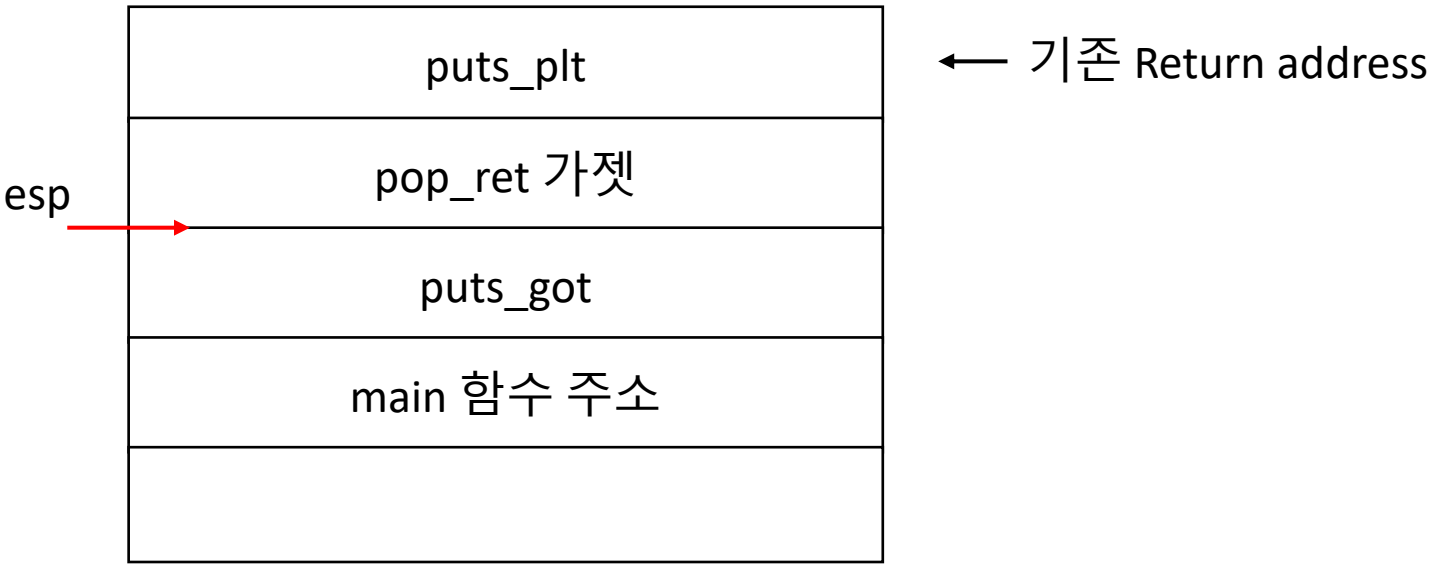
.

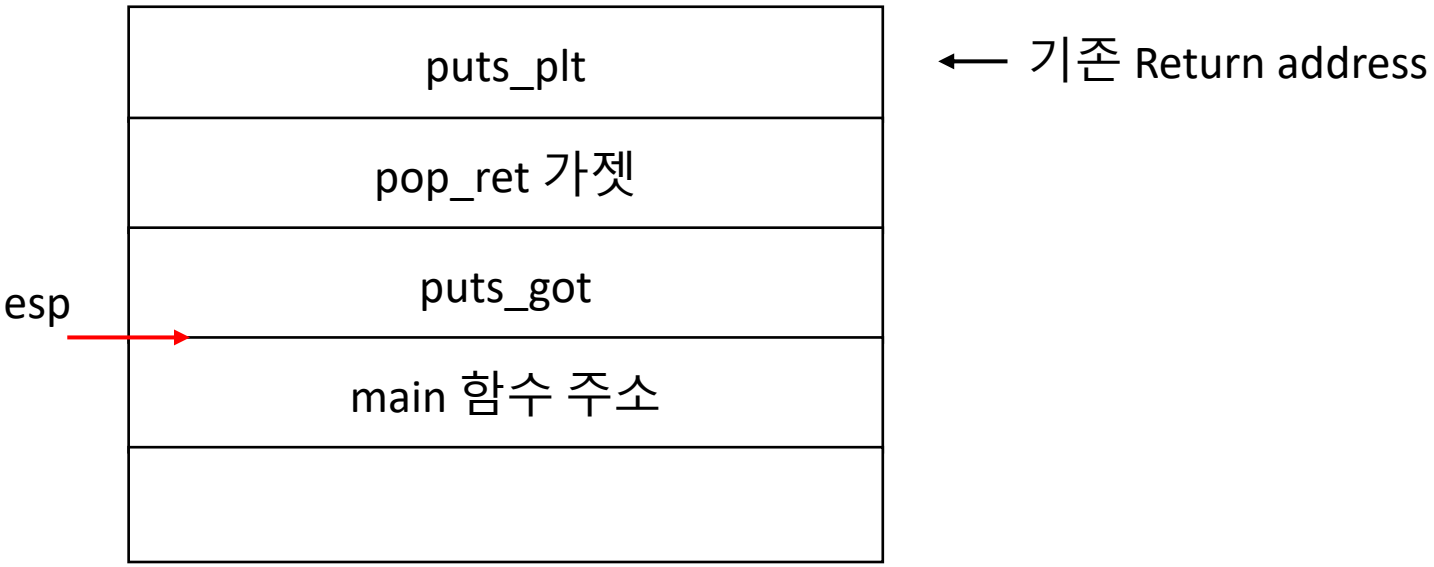
.

.

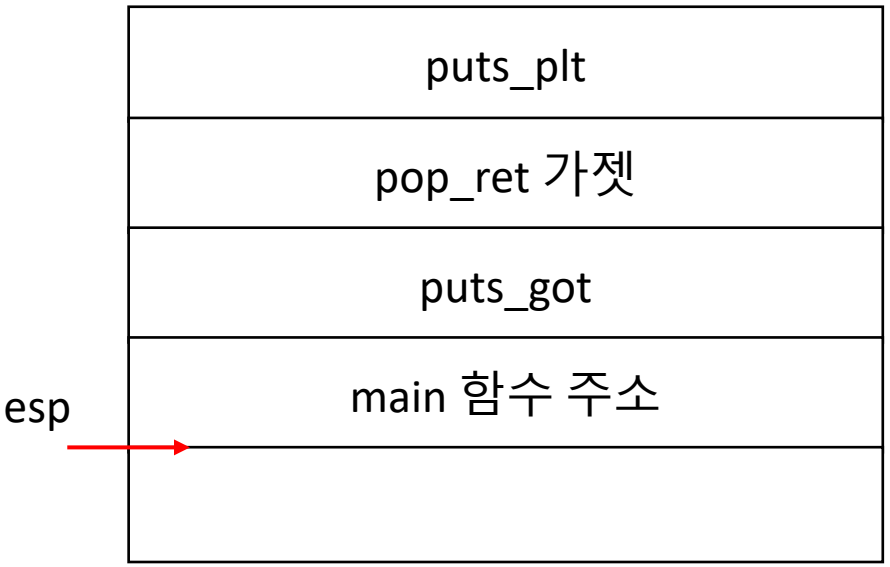
eip

ret


puts_plt
pop_ret 가젯
puts_got
main 함수 주소
esp
← 기존 Return address
eip
pop
ret



eip → pop
ret

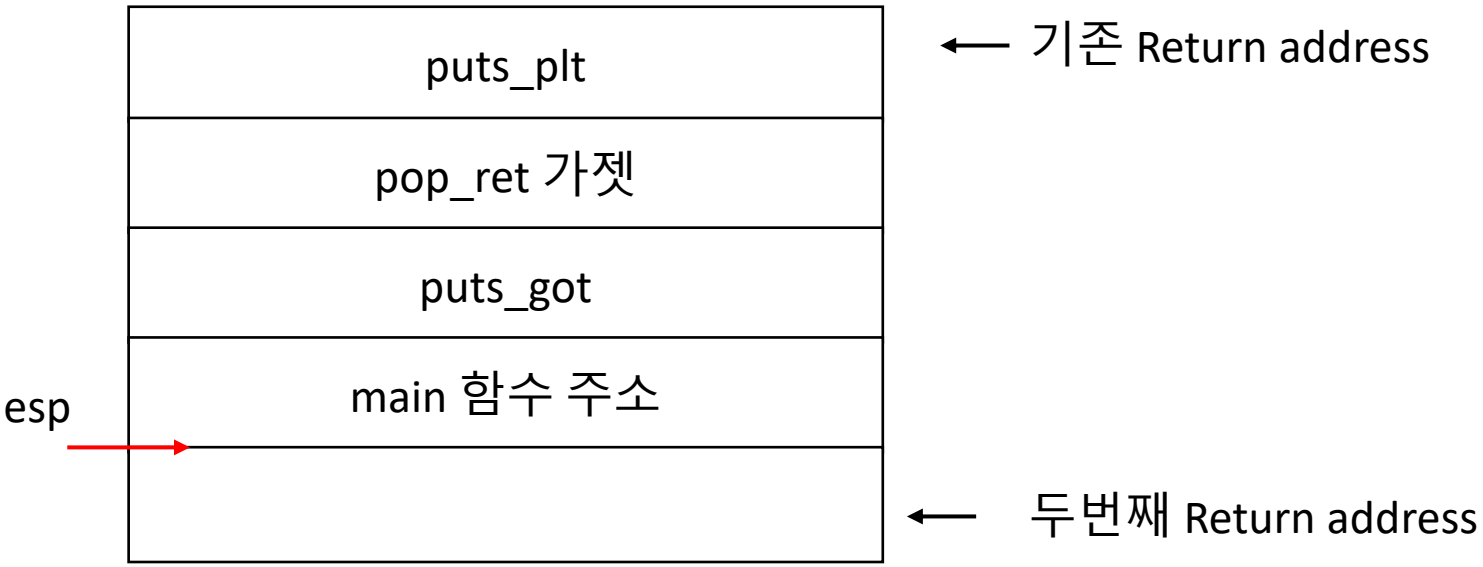


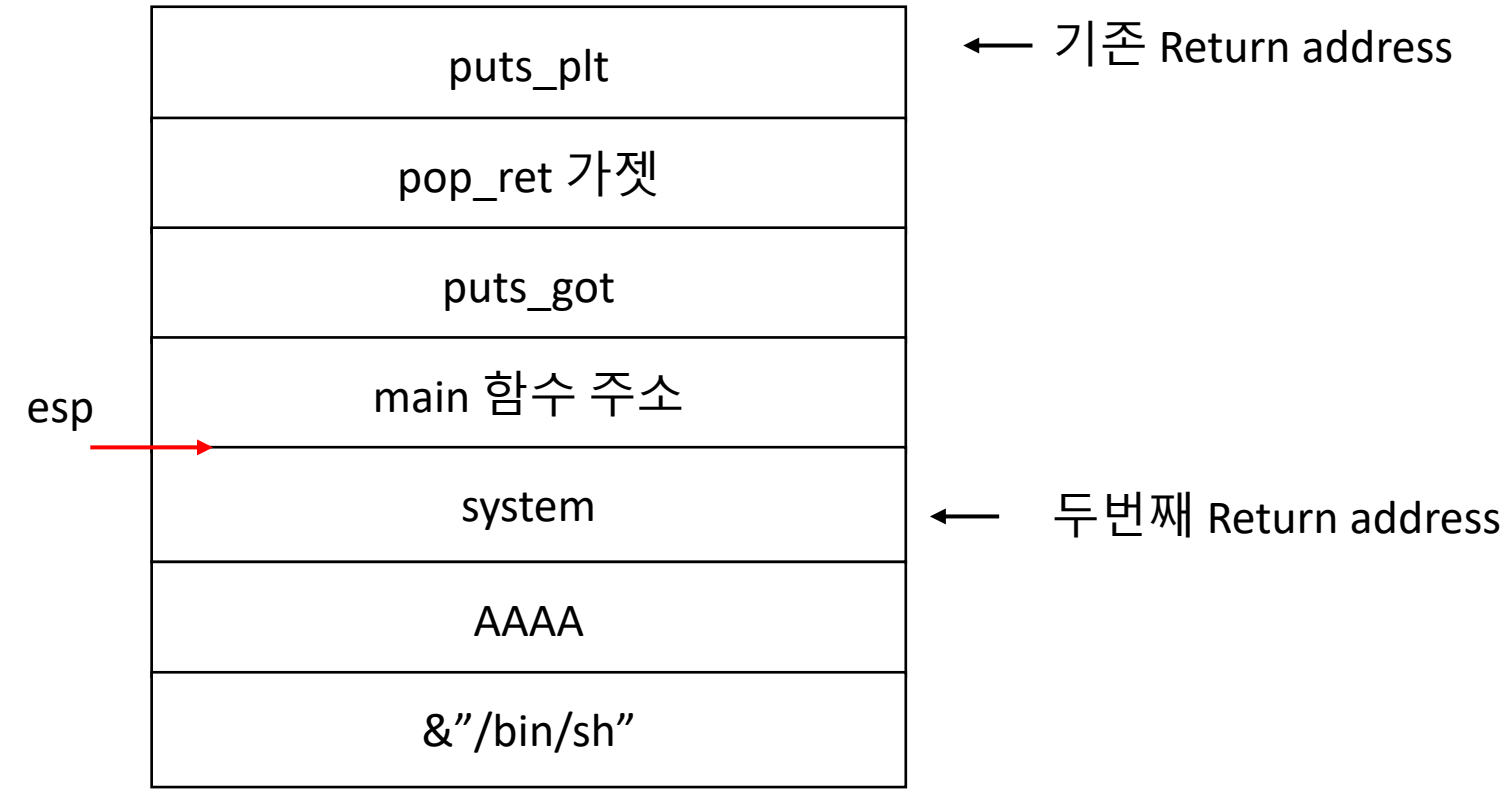
← 기존 Return address

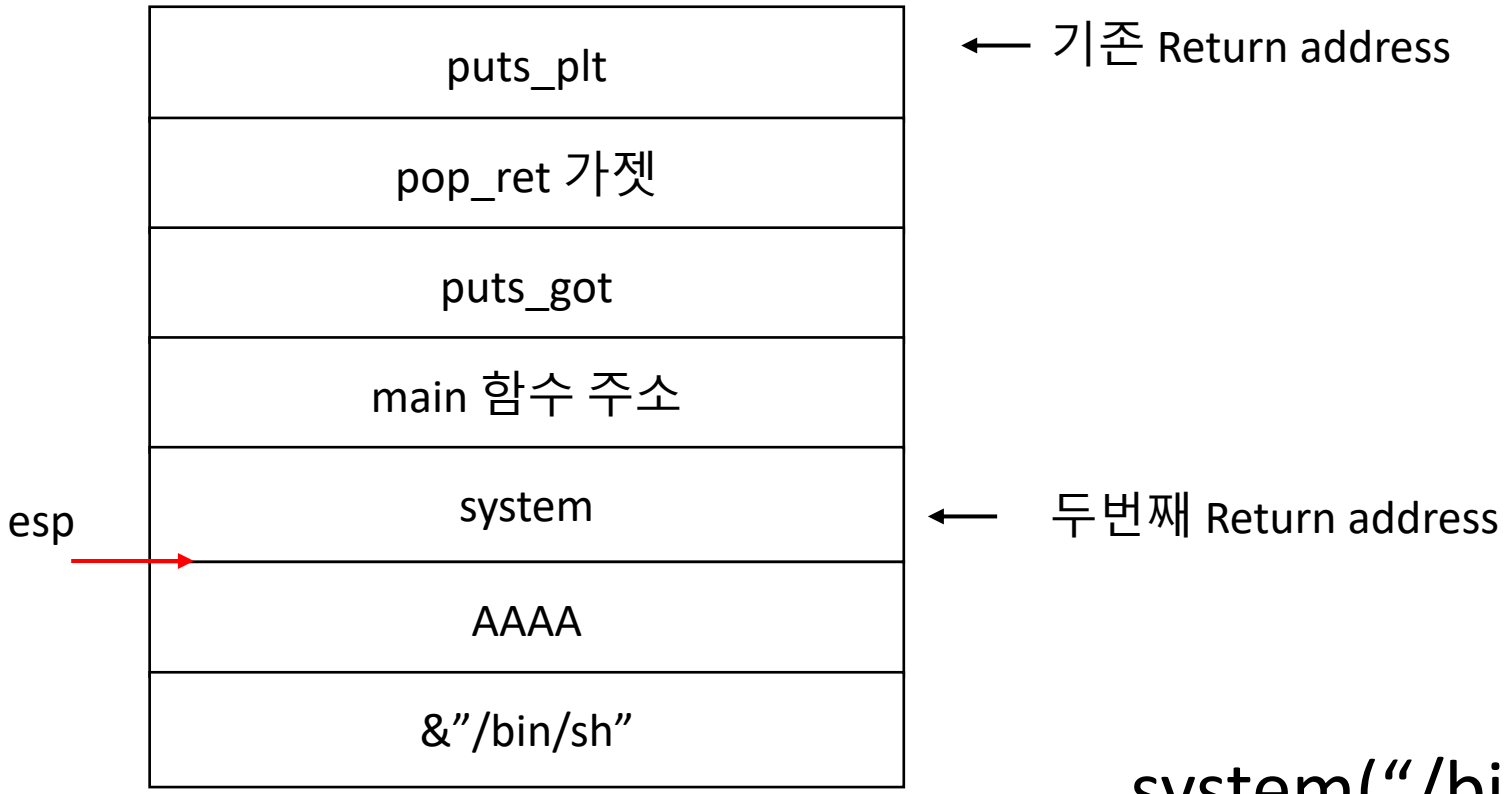
main 함수 실행과 동일!!

eip →

main 함수 시작


puts_plt
pop_ret 가젯
puts_got
main 함수 주소
esp
기존 Return address
두번째 Return address
eip
main 함수 시작


puts_plt
pop_ret 가젯
puts_got
main 함수 주소
system
AAAA
&"/bin/sh"
esp
기존 Return address
두번째 Return address
eip
ret



system("/bin/sh") 실행과 동일!!

eip

system

32bit vs 64bit (x86 vs x64)

- 32bit : 함수가 매개변수를 스택에서 참조함
- 64bit: 함수가 매개변수를 “레지스터”에서 참조함

64bit rop

- 레지스터에 값들을 저장해 놓아야함..
  어떤 레지스터??

rdi -> rsi -> rdx -> r10 -> r8 -> r9 … 순서로 레지스터

64bit rop

rdi -> rsi -> rdx -> r10 -> r8 -> r9 … 순서로 레지스터

ex)

read(0,buf,0x10);

rdi : 0

rsi : buf

rdx : 0x10

read(0,buf,0x10);