시스템 프로그래밍

강의 7 : 3.12 버퍼 오버플로우

http://eslab.cnu.ac.kr

* Some slides are from Original slides of RBE

Linux 메모리 배치

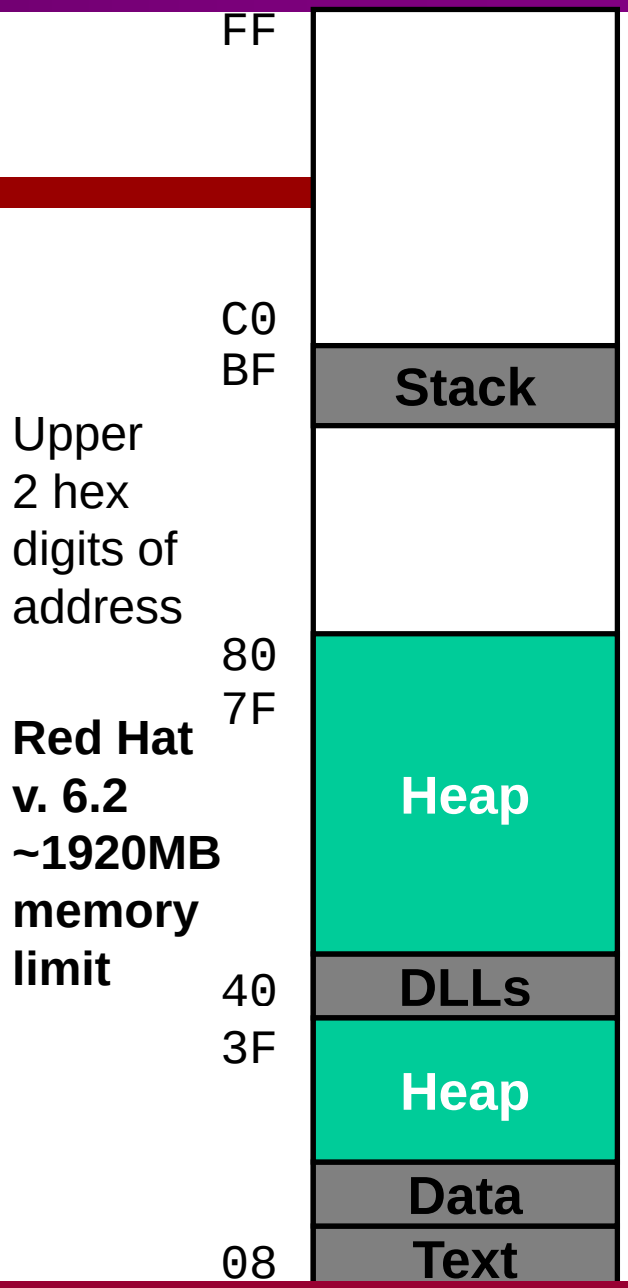
Stack
- 런타임 스택 (8MB 한도)

Heap
- 동적으로 할당되는 공간
- `malloc`, `calloc`, `new` 사용시 할당됨

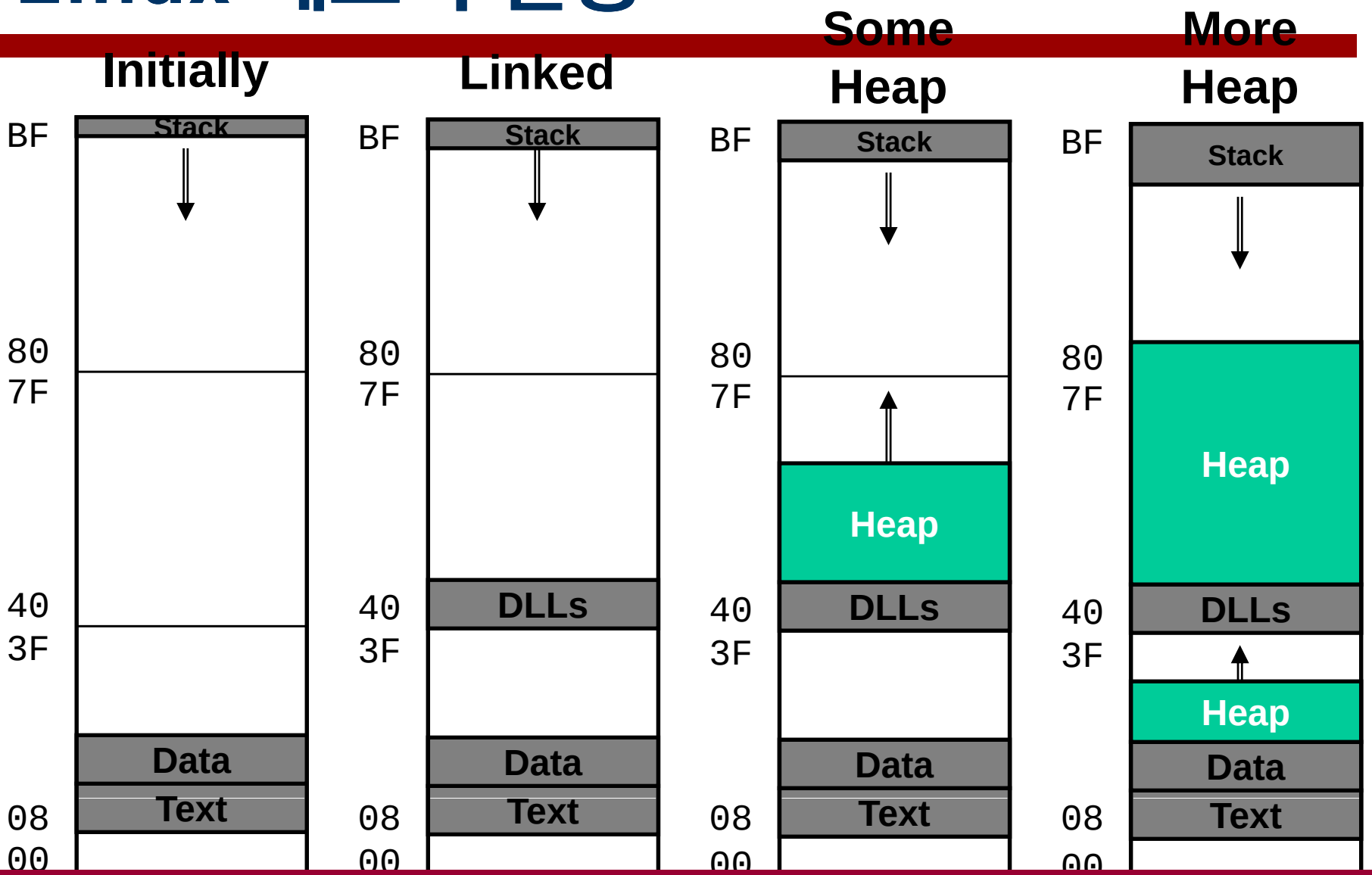
DLLs
- Dynamically Linked Libraries
- Library routines (e.g., `printf`, `malloc`)
- 최초 실행시 링크 됨

Data
- 정적으로 할당되는 데이터 영역
- E.g., arrays & strings 로 선언된 데이터들

Text
- 실행가능한 기계어 명령어 영역
- Read-only

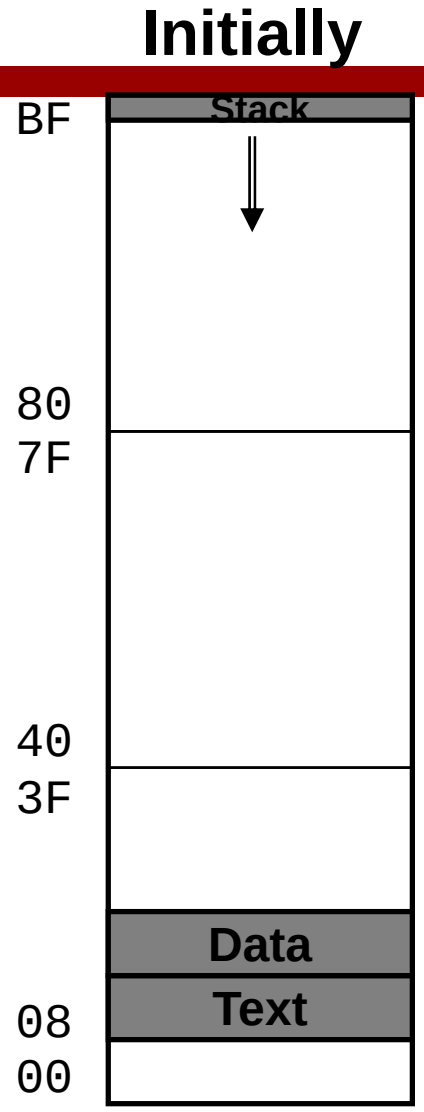


Linux 메모리 할당


Initially
Linked
Some Heap
More Heap
BF
80
7F
40
3F
08
00
Stack
Heap
DLLs
Data
Text

Text와 Stack 예제

```
(gdb) break main
(gdb) run
  Breakpoint 1, 0x804856f in main ()
(gdb) print $esp
  $3 = (void *) 0xbffffc78
```



- **Main**
  - Address `0x804856f` 는 `0x0804856f` 로 읽어야 정확하다
- **Stack**
  - Address `0xbffffc78`

동적 링크 예제

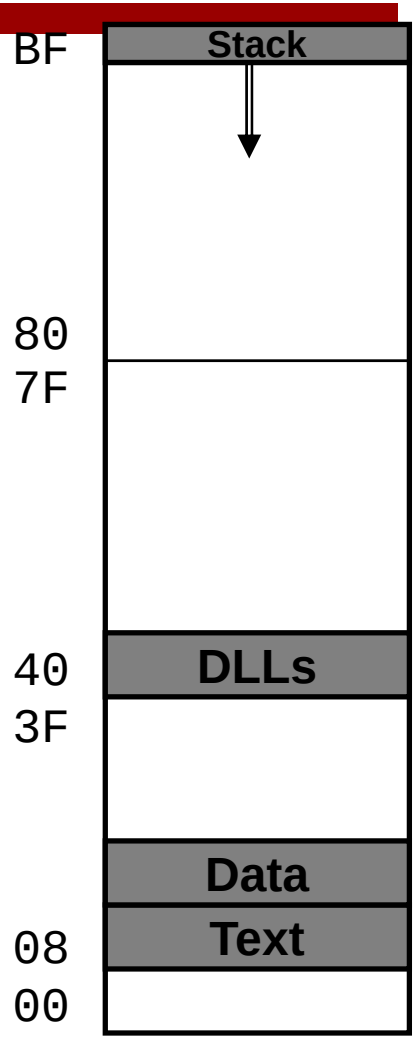
```
(gdb) print malloc
  $1 = {<text variable, no debug info>}
    0x8048454 <malloc>
(gdb) run
  Program exited normally.
(gdb) print malloc
  $2 = {void *(unsigned int)}
    0x40006240 <malloc>
```

최초 상태

- 동적 링커를 호출하는 코드는 text 영역에 존재
- 0x08048454 에 위치

최종 상태

- Code in DLL region

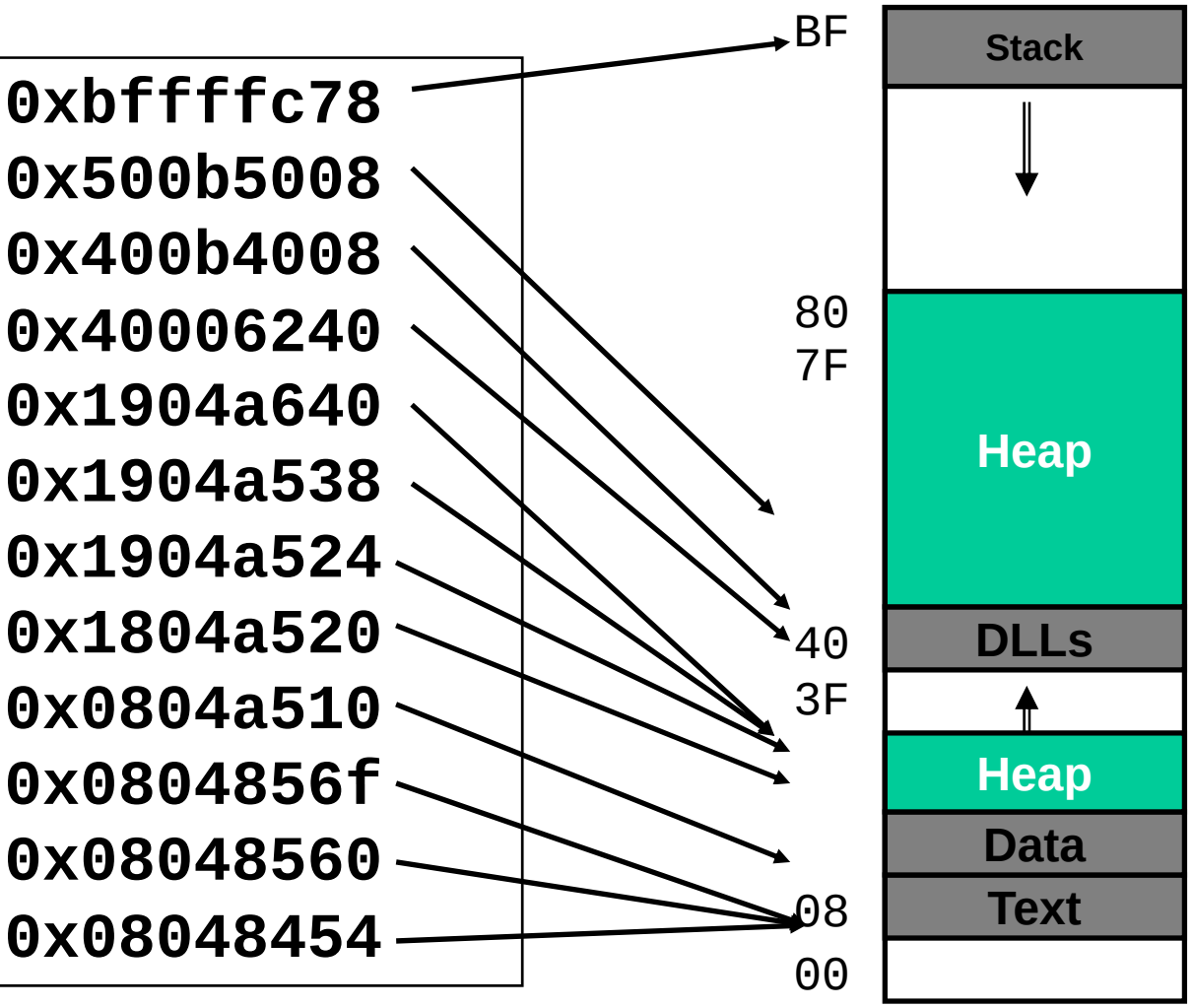


메모리 할당 예제

```
char big_array[1<<24];  /*  16 MB */
char huge_array[1<<28]; /* 256 MB */

int beyond;
char *p1, *p2, *p3, *p4;

int useless() {  return 0; }

int main()
{
 p1 = malloc(1 <<28);  /* 256 MB */
 p2 = malloc(1 << 8);  /* 256 B  */
 p3 = malloc(1 <<28);  /* 256 MB */
 p4 = malloc(1 << 8);  /* 256 B  */
 /* Some print statements ... */
}
```

할당 결과

$esp	0xbffffc78
p3	0x500b5008
p1	0x400b4008
Final malloc	0x40006240
p4	0x1904a640
p2	0x1904a538
beyond	0x1904a524
big_array	0x1804a520
huge_array	0x0804a510
main()	0x0804856f
useless()	0x08048560
Initial malloc	0x08048454



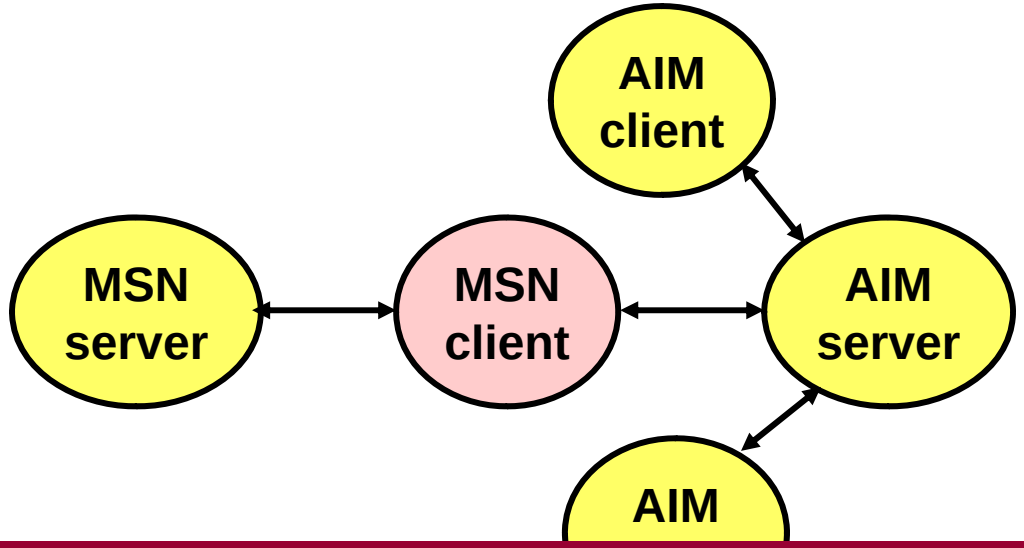
인터넷 웜(Worm)과 IM War

November, 1988, The Morris Worm

- 인터넷 웜은 수천개의 인터넷 호스트를 공격
- 웜 : 스스로 시행되어 다른 기계에 자신을 전파하는 프로그램
- 바이러스 : 다른 프로그램에 자신을 추가하는 프로그램. 비독립적
- How did it happen?

July, 1999

- 마이크로 소프트사는 인터넷 메신저를 출시
- 메신저는 AOL사의 Instant Messaging Service (AIM) 서버를 액세스 할 수 있도록 설계



인터넷 웜(Worm)과 IM War

August 1999

- 갑자기 메신저가 AOL서버를 사용 못하게 됨
- Microsoft and AOL begin the IM war:
  - AOL 은 서버를 교체하여 메신저의 접근을 차단함
  - 마이크로소프트는 바뀐 서버에 접속할 수 있도록 메신저를 수정
  - 13차례 반복
- 어떻게 이런 일이 발생했는가?

Internet Worm 와 AOL/Microsoft War 는 모두 *stack buffer overflow* 현상을 이용한 것이다!

- 많은 Unix functions 는 매개변수의 크기를 체크하지 않는다
- 버퍼 overflow를 허용한다

스트링 라이브러리 함수

- Unix function `gets` 의 구현
  - 읽어 들일 수 있는 문자의 갯수를 한정할 수 없는 구조이다

```
/* Get string from stdin */
char *gets(char *dest)
{
    int c = getc();
    char *p = dest;
    while (c != EOF && c != '\n') {
        *p++ = c;
        c = getc();
    }
    *p = '\0';
    return dest;
}
```

- 유사한 다른 Unix 들에서도 같은 문제가 있다
  - `strcpy`: 임의의 길이의 스트링을 복사
  - `scanf`, `fscanf`, `sscanf` 함수를 `%s` 와 함께 사용하는 경우

위험한 버퍼 코드

```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
int main()
{
  printf("Type a string:");
  echo();
  return 0;
}
```

버퍼 오버플로우의 실행

```
unix>./bufdemo
Type a string:123
123
```

```
unix>./bufdemo
Type a string:12345678
Segmentation Fault
```

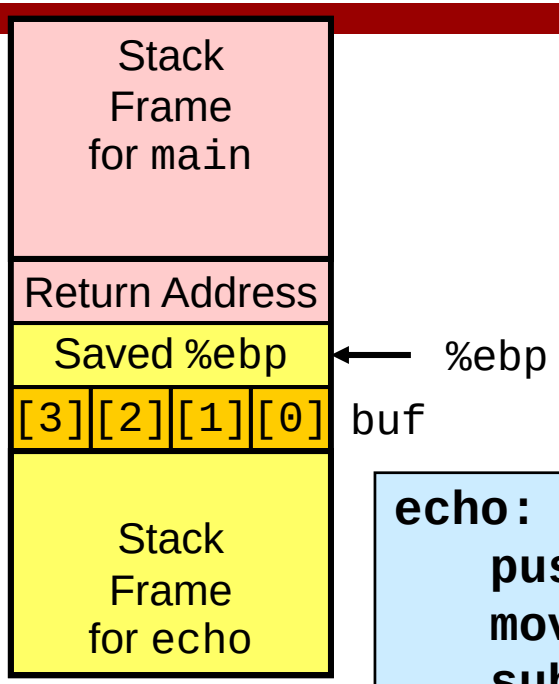
```
unix>./bufdemo
Type a string:12345678ABC
Segmentation Fault
```

버퍼 오버플로우 disassembly

```
080484f0 <echo>:
 80484f0:  55                   push   %ebp
 80484f1:  89 e5                mov    %esp,%ebp
 80484f3:  53                   push   %ebx
 80484f4:  8d 5d f8             lea    0xfffffff8(%ebp),%ebx
 80484f7:  83 ec 14             sub    $0x14,%esp
 80484fa:  89 1c 24             mov    %ebx,(%esp)
 80484fd:  e8 ae ff ff ff       call   80484b0 <gets>
 8048502:  89 1c 24             mov    %ebx,(%esp)
 8048505:  e8 8a fe ff ff       call   8048394 <puts@plt>
 804850a:  83 c4 14             add    $0x14,%esp
 804850d:  5b                   pop    %ebx
 804850e:  c9                   leave
 804850f:  c3                   ret
```

```
 80485f2:  e8 f9 fe ff ff       call   80484f0 <echo>
 80485f7:  8b 5d fc             mov 0xfffffffc(%ebp),%ebx
 80485fa:  c9                   leave
 80485fb:  31 c0                xor    %eax,%eax
 80485fd:  c3                   ret
```

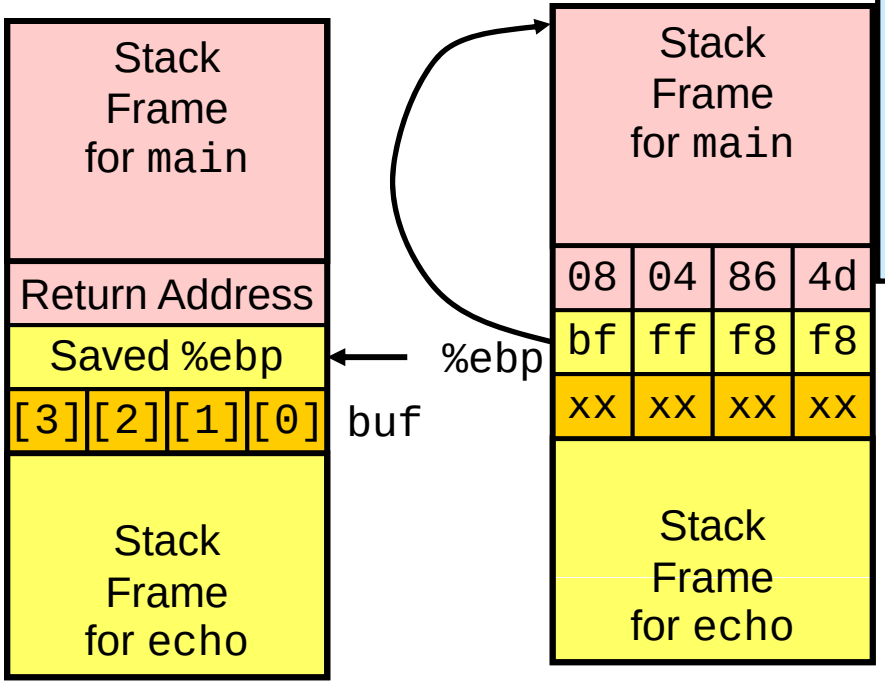
버퍼 오버플로우의 스택




```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
echo:
    pushl %ebp              # Save %ebp on stack
    movl %esp,%ebp
    subl $20,%esp           # Allocate space on stack
    pushl %ebx              # Save %ebx
    addl $-12,%esp          # Allocate space on stack
    leal -4(%ebp),%ebx      # Compute buf as %ebp-4
    pushl %ebx              # Push buf on stack
    call gets               # Call gets
    . . .
```

버퍼 오버플로우 스택 예제




```
unix> gdb bufdemo
(gdb) break echo
Breakpoint 1 at 0x8048583
(gdb) run
Breakpoint 1, 0x8048583 in echo ()
(gdb) print /x *(unsigned *)$ebp
$1 = 0xbffff8f8
(gdb) print /x *((unsigned *)$ebp + 1)
$3 = 0x804864d
```

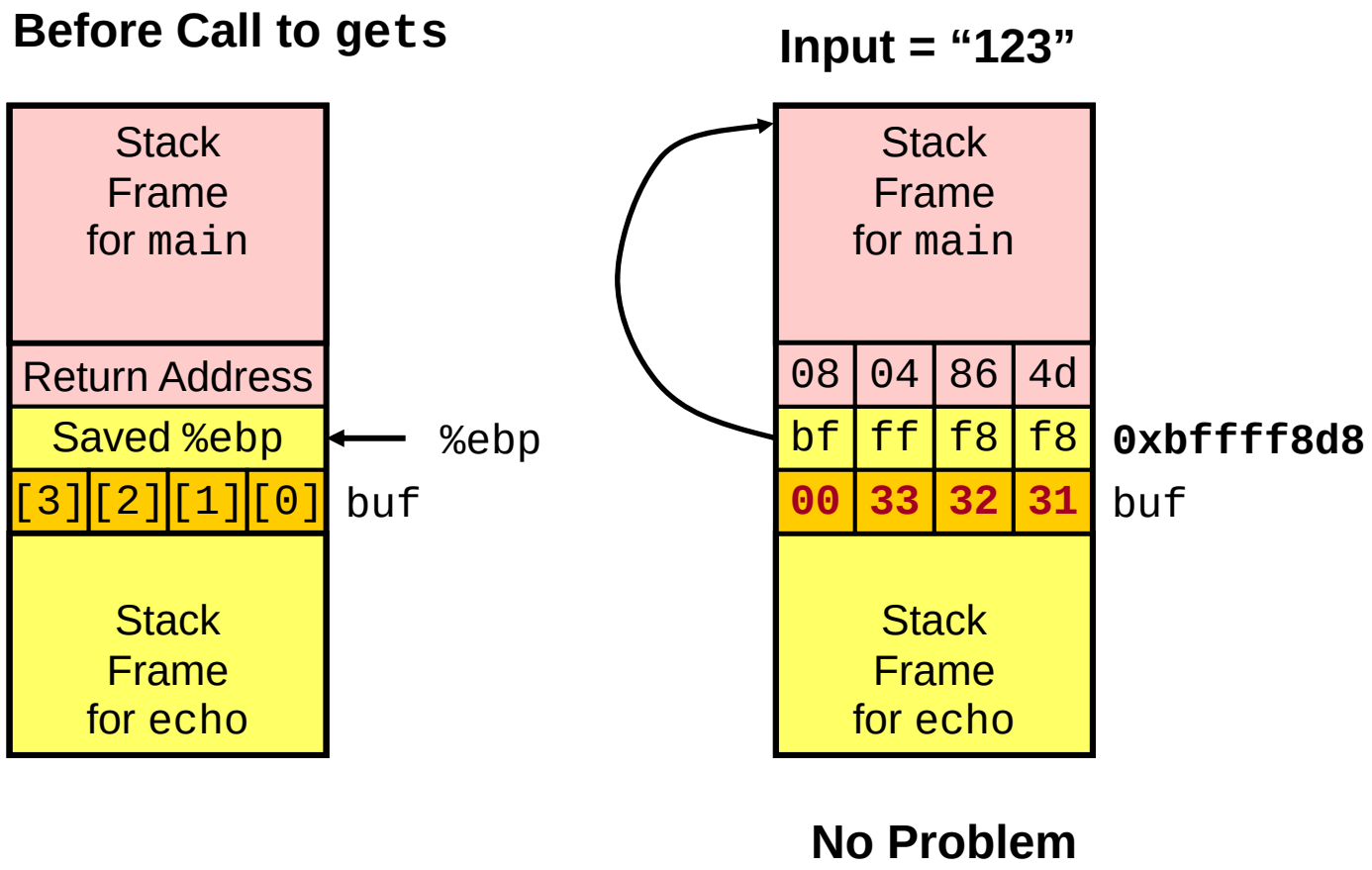
0xbffff8d8

buf

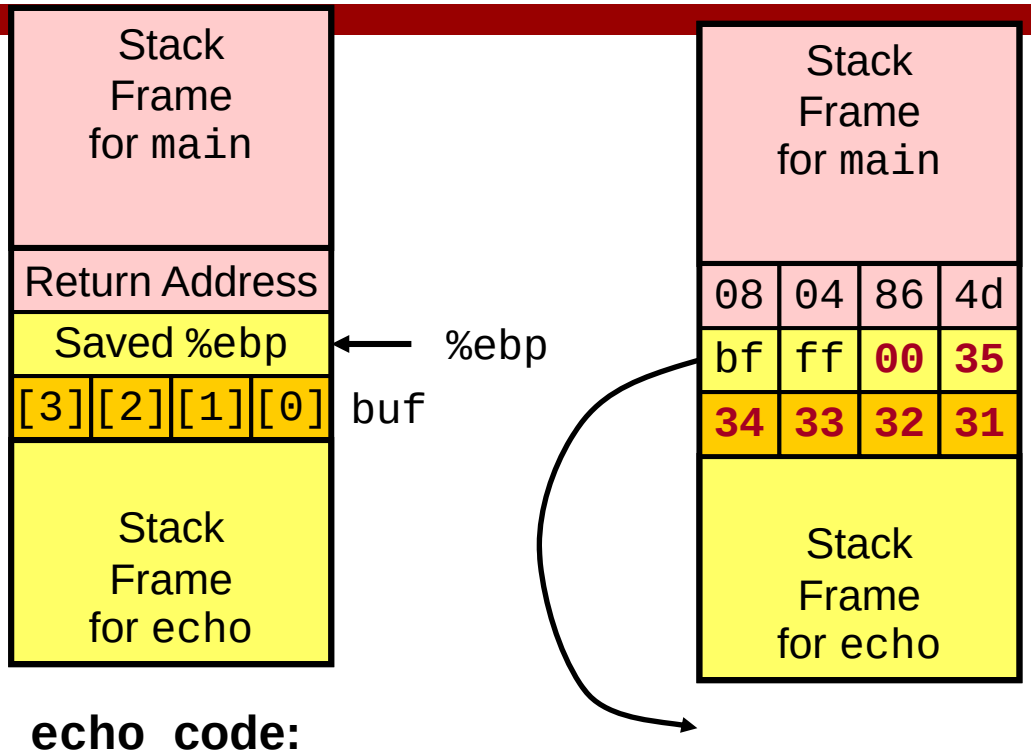
Before call to gets

```
8048648:  call 804857c <echo>
804864d:  mov  0xffffffe8(%ebp),%ebx # Return Point
```

버퍼 오버플로우 예제 1


Before Call to gets
Stack Frame for main
Return Address
Saved %ebp
%ebp
[3] [2] [1] [0]
buf
Stack Frame for echo
Input = "123"
Stack Frame for main
08 04 86 4d
bf ff f8 f8
0xbffff8d8
00 33 32 31
buf
Stack Frame for echo
No Problem

버퍼 오버플로우 예제 2



Input = "12345"

0xbffff8d8

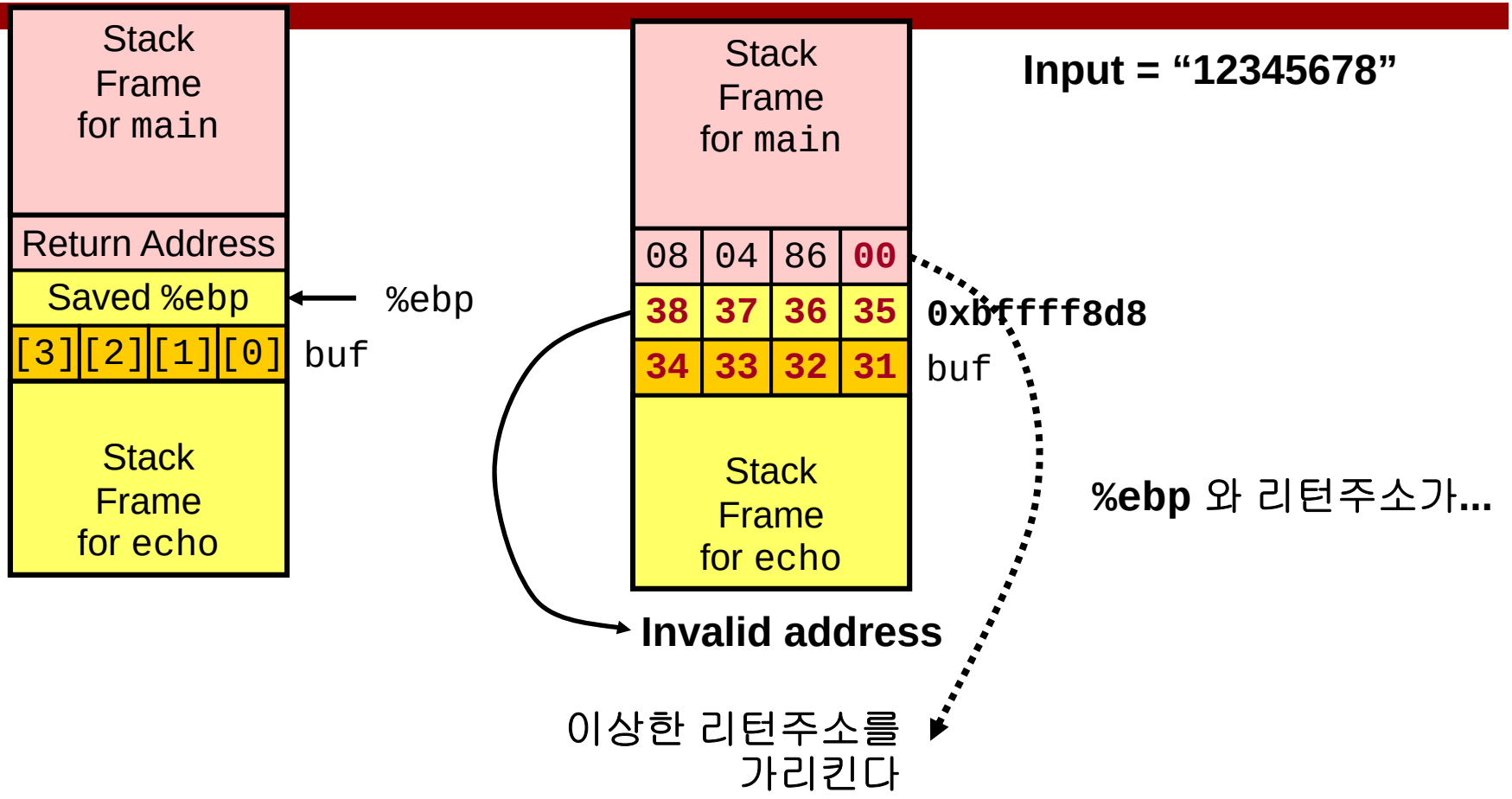
저장한 **%ebp** 값이
0xbfff0035 으로 바뀐다

나중에 **%ebp**를
복구시킬때 문제가 된다

echo code:

```
8048592:  push   %ebx
8048593:  call   80483e4 <_init+0x50>  # gets
8048598:  mov    0xffffffe8(%ebp),%ebx
804859b:  mov    %ebp,%esp
804859d:  pop    %ebp     # %ebp gets set to invalid value
804859e:  ret
```

버퍼 오버플로우 예제 3



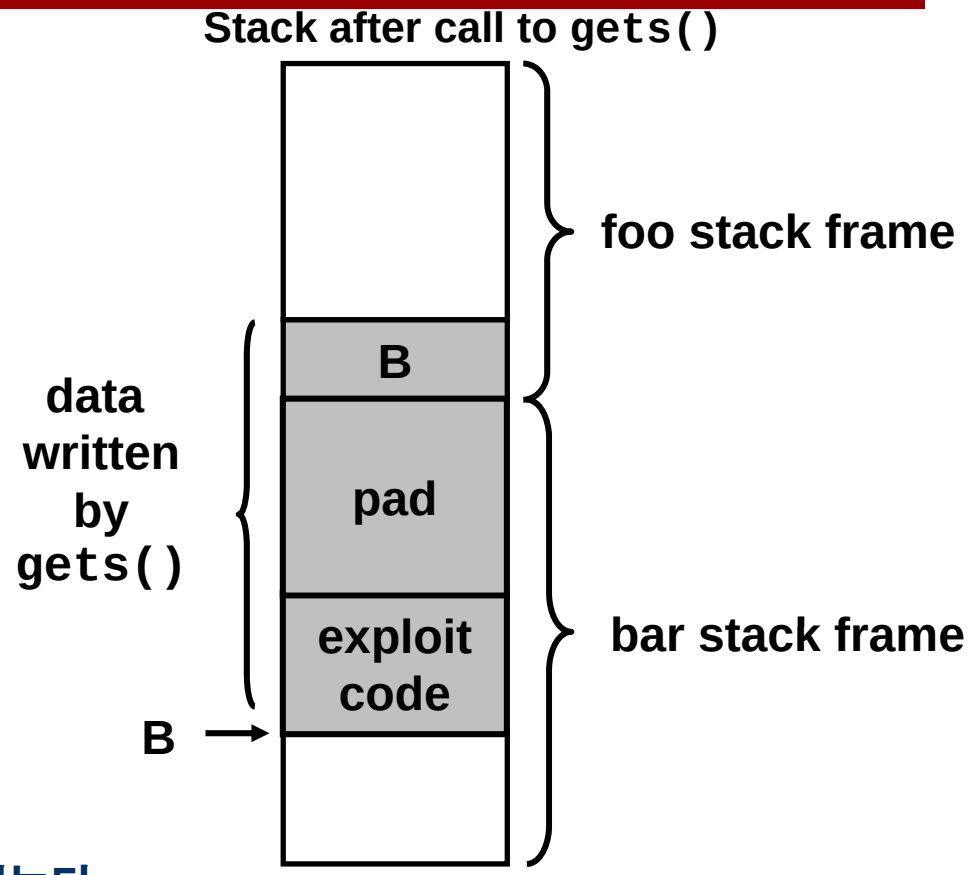

```
8048648:  call 804857c <echo>
804864d:  mov  0xfffffe8(%ebp),%ebx # Return Point
```

버퍼 오버플로우의 사악한 이용

return address A

```
void foo(){
  bar();
  ...
}
```

```
void bar() {
  char buf[64];
  gets(buf);
  ...
}
```



- 입력 스트링에 프로그램 코드를 넣는다
- 리턴 주소 부분에 버퍼의 주소를 써 넣는다
- bar()가 ret을 실행하면, exploit code 부분이 실행된다

버퍼 오버플로우를 이용한 Exploits

버퍼 오퍼플로우 버그는 원격지 컴퓨터들이 타겟 컴퓨터에서 원하는 프로그램을 실행 시킬 수 있도록 해준다.

Internet worm

- 초기 finger server (fingerd)는 `gets()` 클라이언트가 보내준 매개변수를 읽어 들이기 위해서 `gets()` 를 사용하였다

  `finger hyungshin@cnu.ac.kr`

- Worm 은 다음과 같은 스트링을 fingerd server에 전송하여 공격한다:
  - *`finger "exploit-code padding new-return-address"`*
  - exploit code: 해커와 TCP 직접 연결을 설정하는 root shell 을 타켓 컴퓨터에 실행시킨다.

버퍼 오버플로우를 이용한 Exploits

버퍼 오퍼플로우 버그는 원격지 컴퓨터들이 타겟 컴퓨터에서 원하는 프로그램을 실행 시킬 수 있도록 해준다.

IM War

- AOL 은 AIM 클라이언트에 존재하는 buffer overflow bug 를 exploit 방식으로 이용하였다.
- exploit code: 4 바이트의 시그너춰를 서버로 리턴하도록 함 (AIM 클라이언트의 특정 부분에서 4 바이트를 선택).
- Microsoft사가 시그너춰 부분을 찾아내서 메신저를 수정하면, AOL사는 시그너춰의 위치를 변경하였음.

Letter from .....

```
Date: Wed, 11 Aug 1999 11:30:57 -0700 (PDT)
From: Phil Bucking <philbucking@yahoo.com>
Subject: AOL exploiting buffer overrun bug in their own software!
To: rms@pharlap.com

Mr. Smith,

I am writing you because I have discovered something that I think you
might find interesting because you are an Internet security expert with
experience in this area. I have also tried to contact AOL but received
no response.

I am a developer who has been working on a revolutionary new instant
messaging client that should be released later this year.
...
It appears that the AIM client has a buffer overrun bug. By itself
this might not be the end of the world, as MS surely has had its share.
But AOL is now *exploiting their own buffer overrun bug* to help in
its efforts to block MS Instant Messenger.
....
Since you have significant credibility with the press I hope that you
can use this information to help inform people that behind AOL's
friendly exterior they are nefariously compromising peoples' security.

Sincerely,
Phil Bucking
Founder, Bucking Consulting
philbucking@yahoo.com
```

나중에 이 편지가 마이크로소프트사에서 보내진 것으로 밝혀졌다!!!

오버플로우 약점을 피하는 방법

```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    fgets(buf, 4, stdin);
    puts(buf);
}
```

스트링의 길이를 제한하는 라이브러리를 사용한다

- `fgets` **instead of** `gets`
- **strncpy instead of** `strcpy`
- `scanf` **를** `%s` **와 함께 사용하지 않는다**
  - `fgets`**를 사용한다**

요약

3장을 마치며…
앞으로의 일정