CHUNGNAM NATIONAL UNIVERSITY

시스템 프로그래밍

강의 7 : 버퍼 오버플로우

http://eslab.cnu.ac.kr

* Some slides are from Original slides of RBE

메모리 참조 버그 예제

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

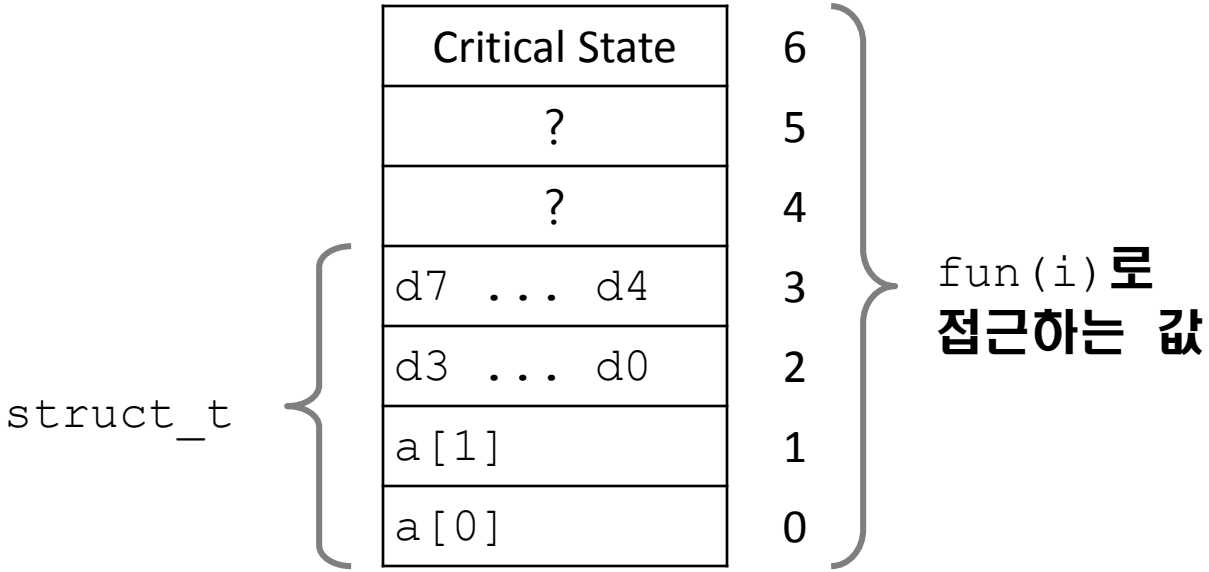
```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.00000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

메모리 참조 버그 예제

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.00000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

설명:



메모리 참조버그의 심각성

일반적으로 "버퍼 오버플로우"라고 부른다
- 배열에 할당된 크기 이상의 메모리를 접근할 때

왜 심각한가?
- 가장 빈번히 발생하는 보안 취약성의 원인

가장 일반적인 형태
- 스트링 입력의 길이를 체크하지 않는 경우
- 스택에 생성되는 제한된 길이의 문자배열

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

```
unix>./bufdemo-nsp
Type a string:0123456789012345678901 23
0123456789012345678901 23
```

```
unix>./bufdemo-nsp
Type a string:012345678901234567890123
Segmentation Fault
```

버퍼 오버플로우 역어셈블리

echo:

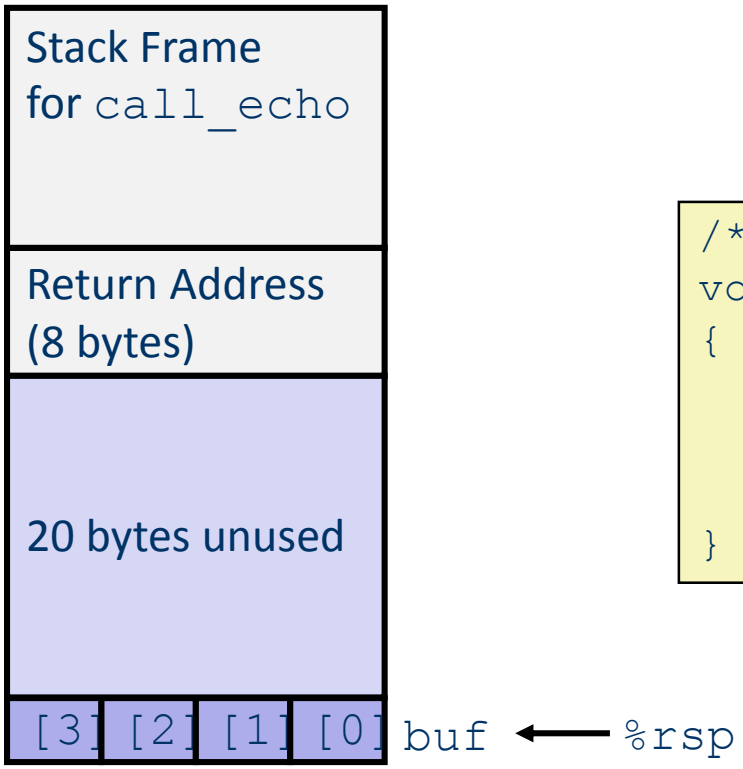
```
0000000004006cf <echo>:
 4006cf:  48 83 ec 18           sub    $0x18,%rsp
 4006d3:  48 89 e7              mov    %rsp,%rdi
 4006d6:  e8 a5 ff ff ff        callq  400680 <gets>
 4006db:  48 89 e7              mov    %rsp,%rdi
 4006de:  e8 3d fe ff ff        callq  400520 <puts@plt>
 4006e3:  48 83 c4 18           add    $0x18,%rsp
 4006e7:  c3                    retq
```

call_echo:

```
 4006e8:   48 83 ec 08          sub    $0x8,%rsp
  4006ec:  b8 00 00 00 00       mov    $0x0,%eax
  4006f1:  e8 d9 ff ff ff       callq  4006cf <echo>
  4006f6:  48 83 c4 08          add    $0x8,%rsp
  4006fa:  c3                   retq
```

버퍼 오버플로우 스택

Before call to gets

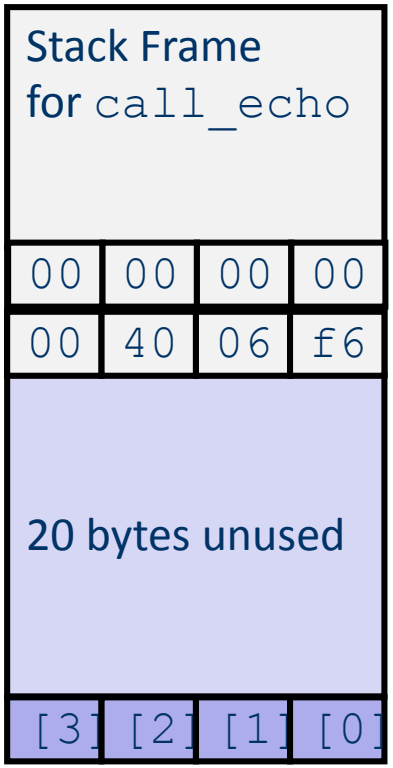



```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
echo:
   subq   $24, %rsp
   movq   %rsp, %rdi
   call   gets
   . . .
```

버퍼 오버플로우 스택 예제

Before call to gets




```
void echo()
{
    char buf[4];
    gets(buf);
    . . .
}
```

```
echo:
   subq   $24, %rsp
   movq   %rsp, %rdi
   call   gets
   . . .
```

call_echo:

```
   . . .
   4006f1:  callq  4006cf <echo>
   4006f6:  add    $0x8,%rsp
   . . .
```

버퍼 오버플로우 스택 예제 #1

After call to gets

Stack Frame for `call_echo`

00	00	00	00
00	40	06	f6
00	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

buf ← %rsp

```
void echo()
{
    char buf[4];
    gets(buf);
    . . .
}
```

```
echo:
   subq   $24, %rsp
   movq   %rsp, %rdi
   call   gets
   . . .
```

call_echo:

```
   . . .
   4006f1:  callq  4006cf <echo>
   4006f6:  add    $0x8,%rsp
   . . .
```

```
unix>./bufdemo-nsp
Type a string:012345678901234567890123
012345678901234567890123
```

버퍼오버플로우 발생. 그러나, 상태는 깨뜨리지 않았다

버퍼 오버플로우 스택 예제 #2

After call to gets

Stack Frame for `call_echo`

00	00	00	00
00	40	00	34
33	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

buf ← %rsp

```
void echo()
{
    char buf[4];
    gets(buf);
    . . .
}
```

```
echo:
   subq   $24, %rsp
   movq   %rsp, %rdi
   call   gets
   . . .
```

call_echo:

```
   . . .
   4006f1:  callq  4006cf <echo>
   4006f6:  add    $0x8,%rsp
   . . .
```

```
unix>./bufdemo-nsp
Type a string:0123456789012345678901234
Segmentation Fault
```

버퍼 오버플로우 발생. 리턴 포인터가 깨졌다.

버퍼 오버플로우 스택 예제 #3

After call to gets

Stack Frame for `call_echo`

00	00	00	00
00	40	06	00
33	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

buf ← %rsp

```
void echo()
{
    char buf[4];
    gets(buf);
    . . .
}
```

```
echo:
   subq   $24, %rsp
   movq   %rsp, %rdi
   call   gets
   . . .
```

call_echo:

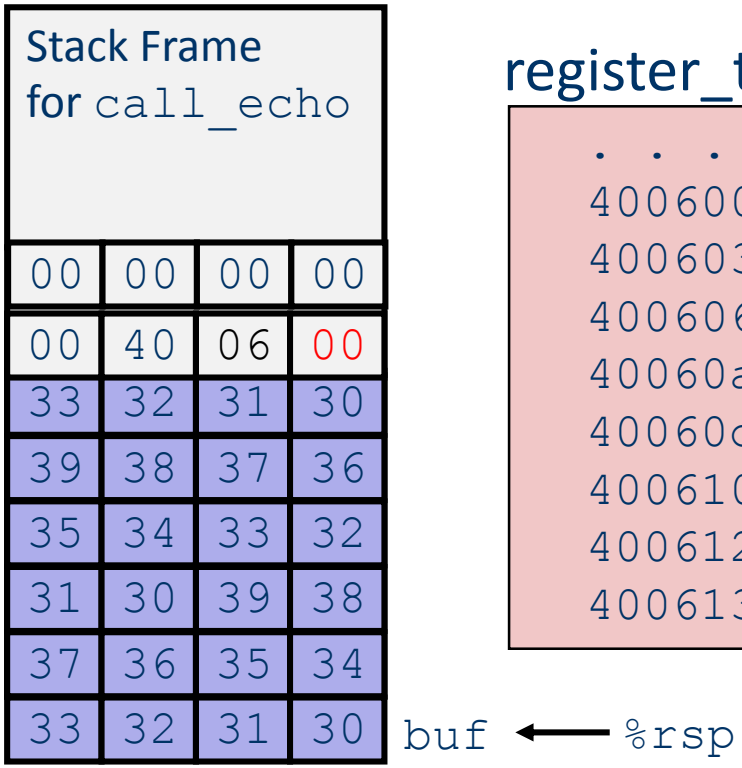
```
   . . .
   4006f1:  callq  4006cf <echo>
   4006f6:  add    $0x8,%rsp
   . . .
```

```
unix>./bufdemo-nsp
Type a string:0123456789012345678901 23
0123456789012345678901 23
```

버퍼오버플로우 발생, 리턴포인터가 깨졌으나, 프로그램은 동작하는것 처럼 보인다

버퍼 오버플로우 스택 예제 #3에 대한 설명

After call to gets

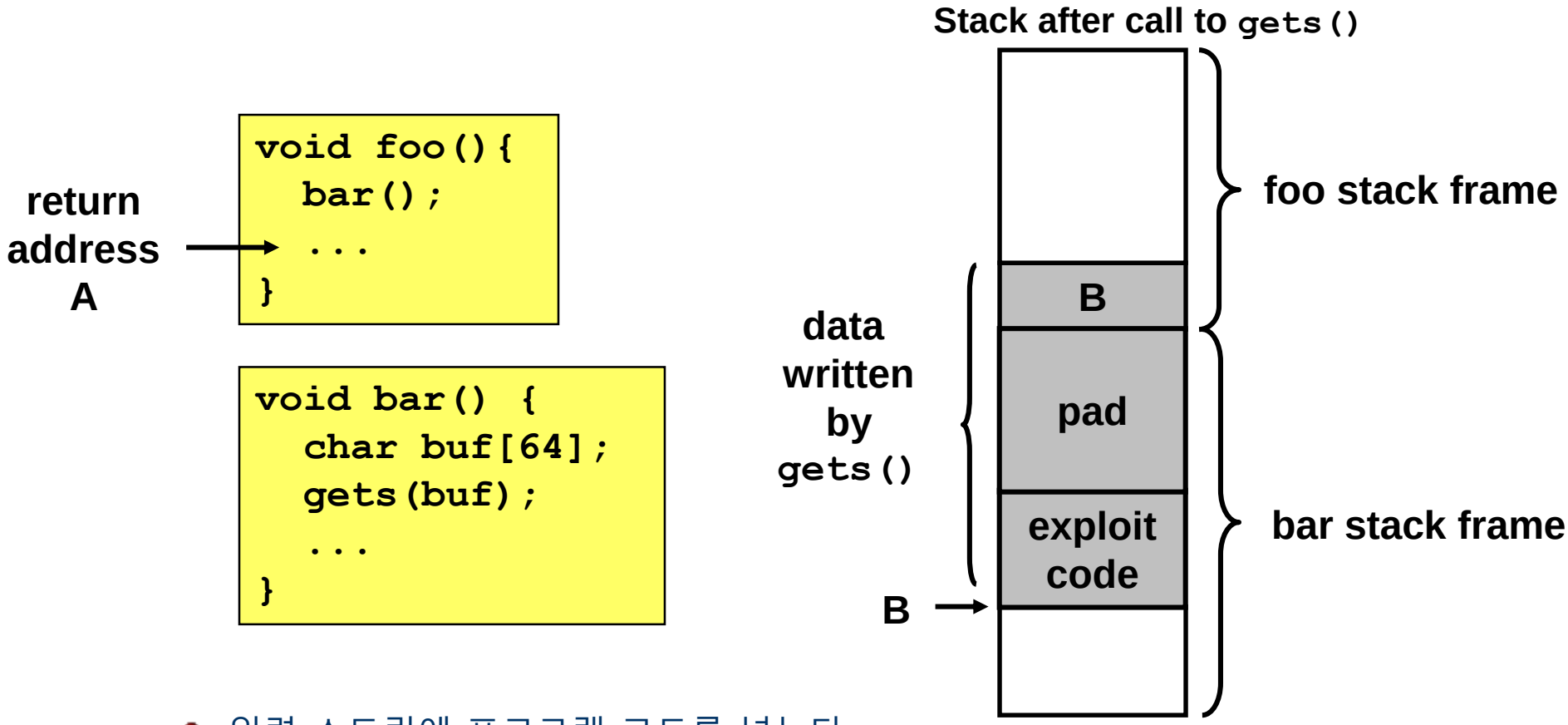


register_tm_clones:

```
. . .
400600:  mov    %rsp,%rbp
400603:  mov    %rax,%rdx
400606:  shr    $0x3f,%rdx
40060a:  add    %rdx,%rax
40060d:  sar    %rax
400610:  jne    400614
400612:  pop    %rbp
400613:  retq
```

관련없는 코드로 리턴한다
여러 오동작을 하지만, 그래도 중요한 상태는 변경되지 않았다
최종적으로 retq를 만나 main으로 리턴한다

버퍼 오버플로우의 사악한 이용



- 입력 스트링에 프로그램 코드를 넣는다
- 리턴 주소 부분에 버퍼의 주소를 써 넣는다
- `bar()`가 `ret`을 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 @1988

- 초기 finger server (fingerd)는 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.
- 한시간 동안 6000대의 컴퓨터를 공격 (인터넷 전체의 10%)
- Worm 설치한 사람은 잡혀서 구속3년, 400시간 노동, 벌금 1천만원

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