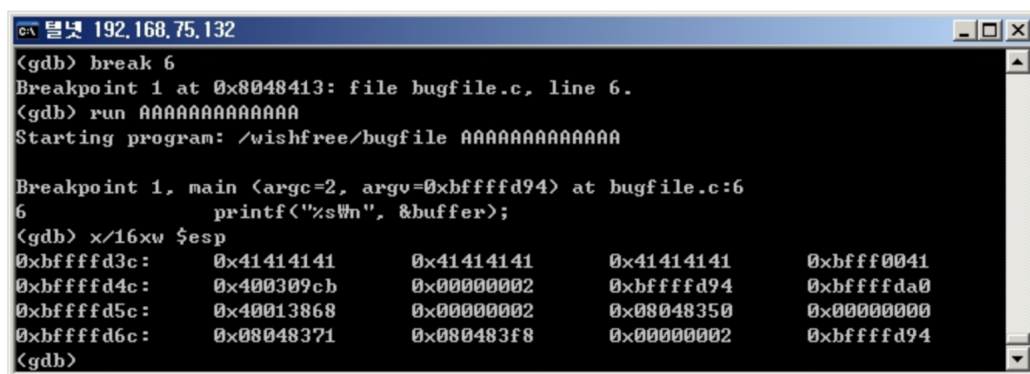


bugfile.c 에서 취약점이 발생하도록 해보자.

1. line 6 에 중단점을 걸고 파라미터를 A 13개 로 전달한다 그리고 esp 값을 확인하면서 스택에 있는 값을 확인한다 13개가 들어갔으니까 41414141 이 들어가 있을텐데 왼쪽 마지막을 보면 41 이 오른쪽에 박혀있다. little endian 이라 그렇다.

```
break 6
run AAAAAAAAAAAAAA
x/16xw $esp
```



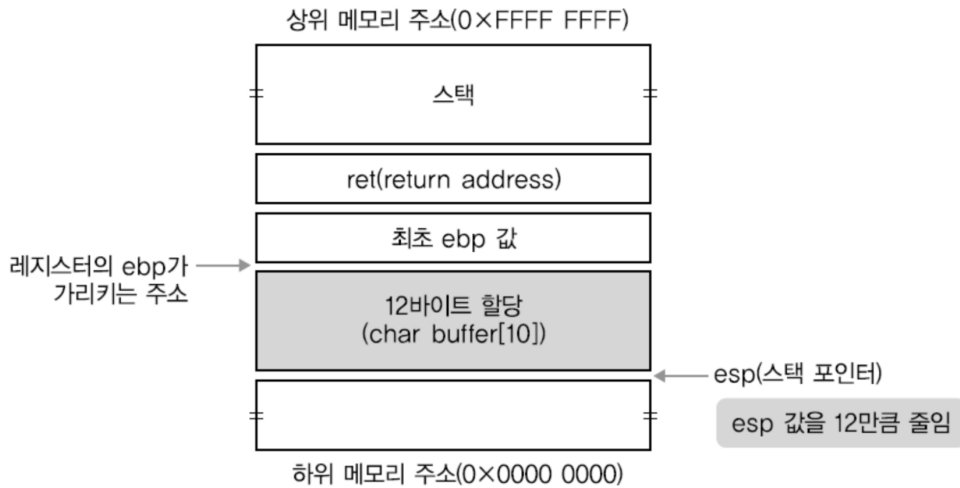
```
G> 텔넷 192.168.75.132
(gdb) break 6
Breakpoint 1 at 0x8048413: file bugfile.c, line 6.
(gdb) run AAAAAAAAAAAAAA
Starting program: /wishfree/bugfile AAAAAAAAAAAAAA

Breakpoint 1, main (argc=2, argv=0xbffffd94) at bugfile.c:6
6      printf("%s\n", &buffer);
(gdb) x/16xw $esp
0xbffffd3c:  0x41414141      0x41414141      0x41414141      0xbfff0041
0xbffffd4c:  0x400309cb      0x00000002      0xbffffd94      0xbffffda0
0xbffffd5c:  0x40013868      0x00000002      0x08048350      0x00000000
0xbffffd6c:  0x08048371      0x080483f8      0x00000002      0xbffffd94
(gdb)
```

해당 코드 스택 구조

- ❶ 0x80483f8 <main>: push %ebp
- ❷ 0x80483f9 <main+1>: mov %esp,%ebp
- ❸ 0x80483fb <main+3>: sub \$0xc,%esp

Subtract 12(0xc) from esp. **Allocate 12 bytes in the stack.** Allocate buffer[10] in the stack

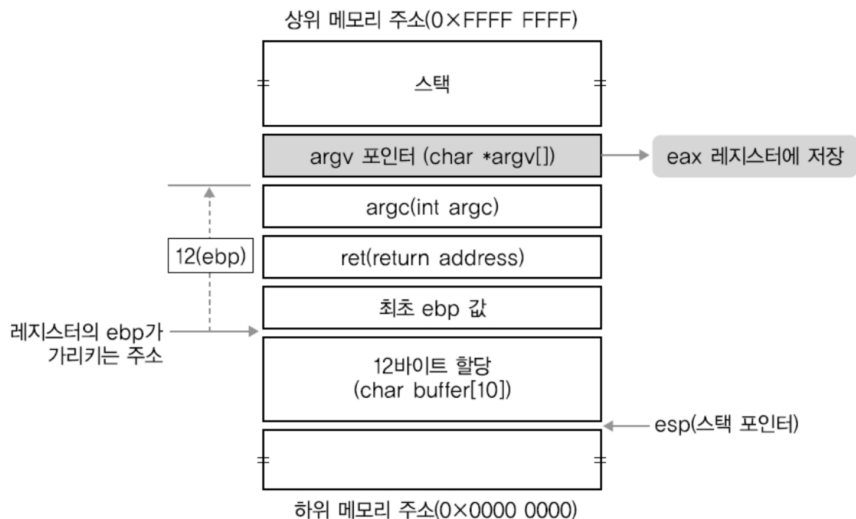


이거 인자를 왜 맨 위에다 가져다 두냐 ?? → 질문

- ❹ 0x80483fe <main+6>: mov 0xc(%ebp),%eax

Stores the content of upper 12 bytes(0xC) from EBP to EAX

This is the parameters (int argc, **char *argv[]**) before calling main



[그림 7-14] main+6까지 실행 시 스택의 구조

⑤ 0x8048401 <main+9>: **add \$0x4, %eax**

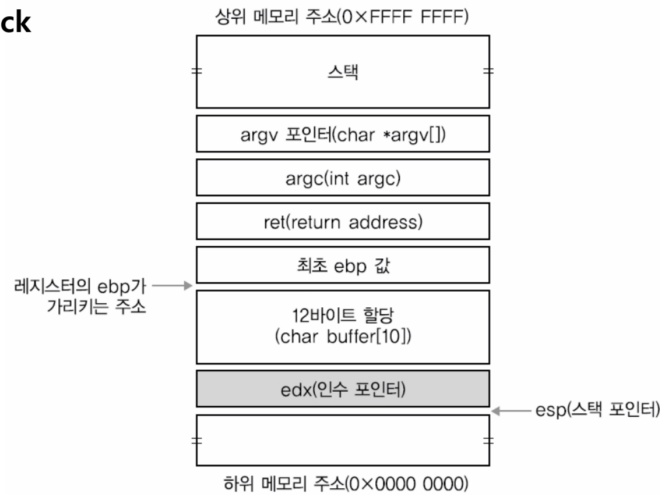
Because EAX is a pointer to argv[0], now **it points to argv[1]**

⑥ 0x8048404 <main+12>: **mov (%eax), %edx**

Move EAX value to EDX register

⑦ 0x8048406 <main+14>: **push %edx**

Stores pointer to the parameters in the stack. If the parameter is empty, then stores 0x0 in the stack



[그림 7-15] main+14까지 실행 시 스택의 구조

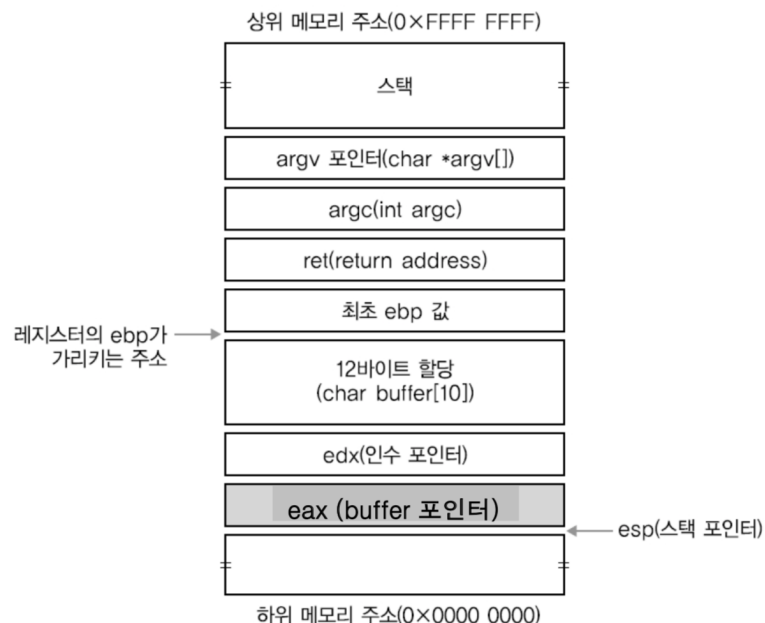
⑧ 0x8048407 <main+15>: **lea 0xffffffff4(%ebp),%eax**

Stores -12(%ebp) to EAX register

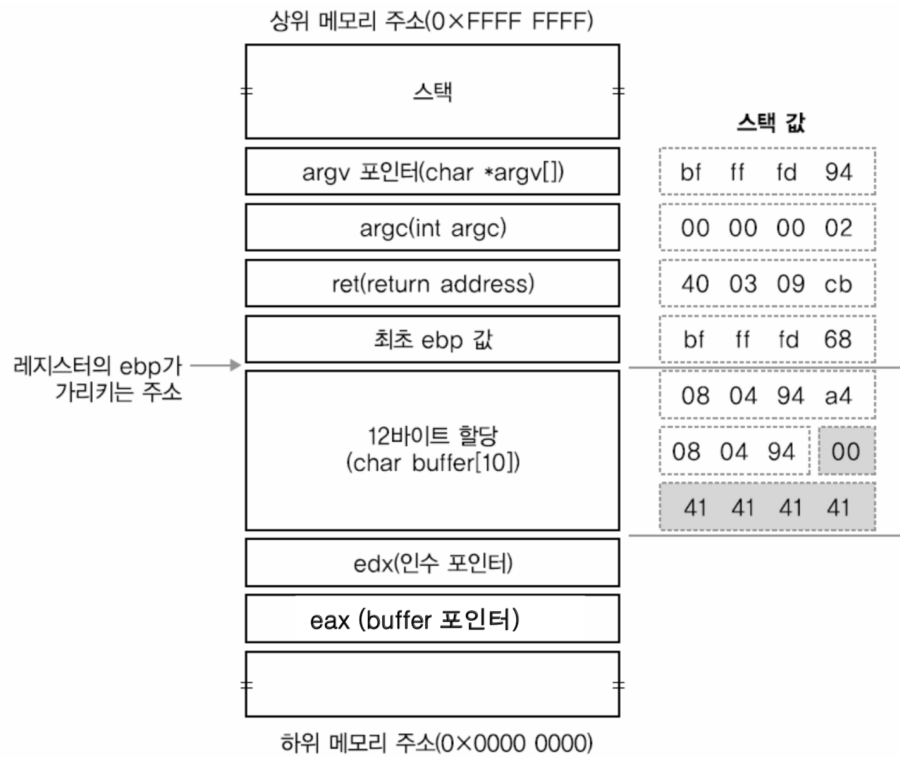
Address to buffer[0]

⑨ 0x804840a <main+18>: **push %eax**

Stores EAX in the stack



⑩ 0x804840b <main+19>: **call 0x8048340 <strcpy>** Call strcpy function

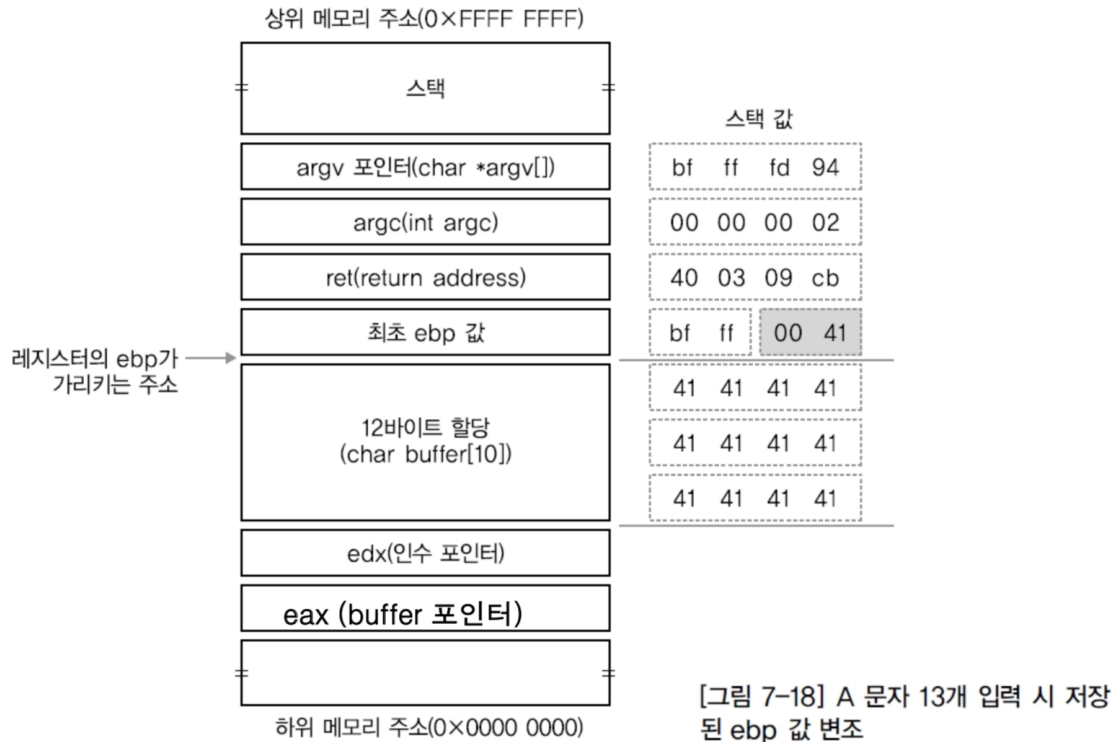


[그림 7-17] main+18까지 실행 시 스택의 구조

이건 아까 A 가 4개 들어갔을 때 strcpy 호출 구조.

strcpy

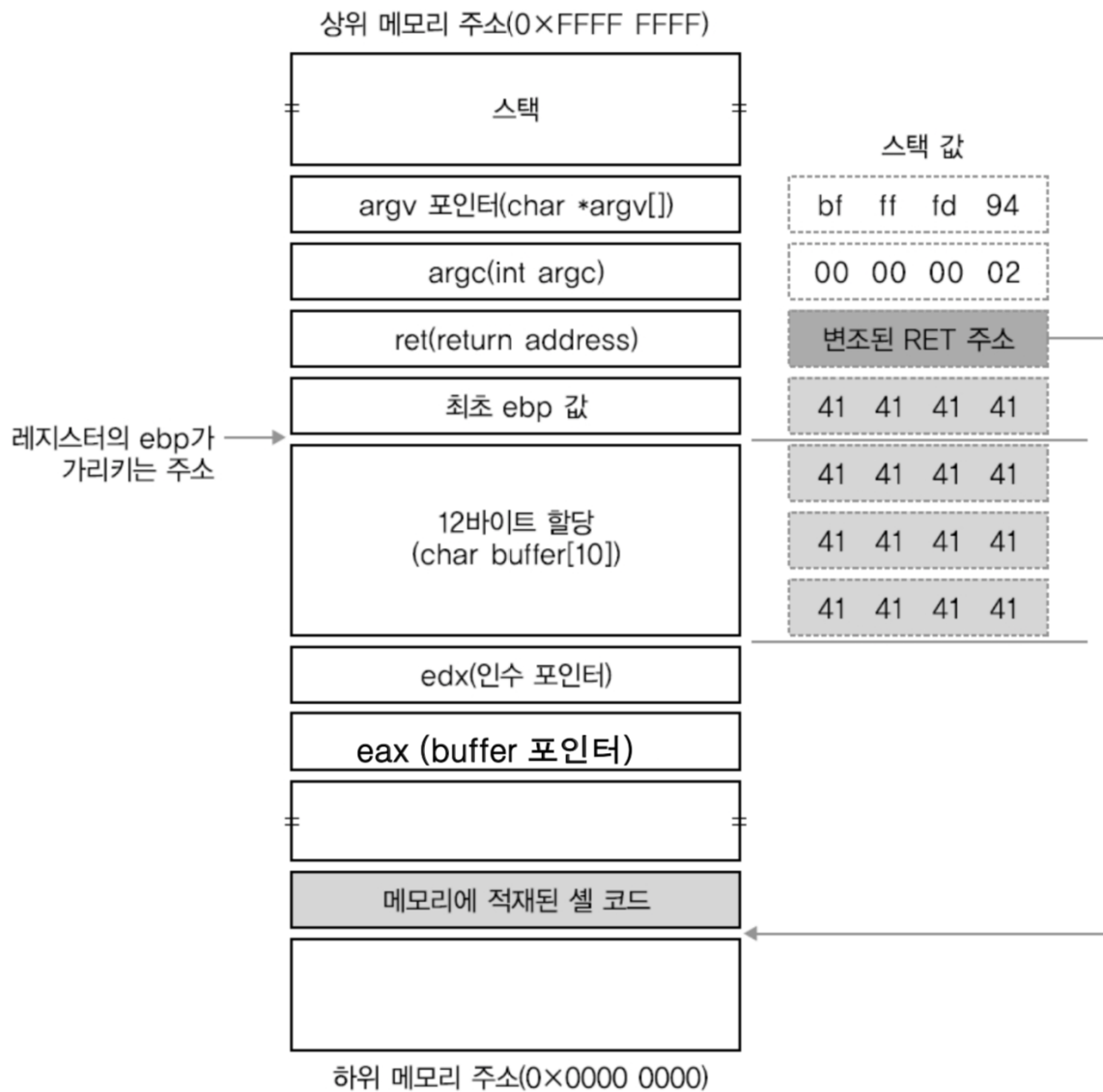
- **Doesn't check the boundary** of input parameter
- The parameter is `buffer[10]`, so the length can't exceed 10 bytes
- However, it write bigger parameter than this in the stack
- If the parameter is **13 As** like Step 8, it is as following



근데 취약점 발생하는 인자를 넣었을 때는 이 처럼 할당 된 buffer 의 크기를 넘어서 값을 저장하게 된다. 00 은 문자열 끝을 나타내는 '\0'

boundary check 를 하지 않기 때문에 넘쳐서(buffer overflow) 저장함.. 그렇게 되면서 ebp 값에 덮여쓰지면서 변조 되었다.

스택에서 자신에게 할당 된 영역이외에 다른 영역을 침범하면서 값을 조작할 수 있게 되는 것이 buffer overflow 이다. 이 경우 Ret 값을 조정하면서 eip 를 조작할 수 있게 된다. 가령 shell code 를 삽입하여 해당 코드를 실행하도록 할 수도 있겠지 !



[그림 7-19] ebp 값을 지나 ret 값 변조