CSED702C-01 Walkthrough Presentation

(strncpy)

Byoungho Son

2024.10.08

Challenge

It is well known that `strcpy()` is unsafe. `strncpy()` is a safer alternative, which you should use. But, is it really safe? Carefully read the manual page of `strncpy()`: `$ man strncpy`.

Analysis

```
Dump of assembler code for function copy:
   0x0804923e <+0>:     push   ebp
   0x0804923f <+1>:     mov    ebp,esp
   0x08049241 <+3>:     sub    esp,0x200
   0x08049247 <+9>:     push   DWORD PTR [ebp+0x8]
   0x0804924a <+12>:    call   0x80490a0 <strlen@plt>
   0x0804924f <+17>:    add    esp,0x4
   0x08049252 <+20>:    cmp    eax,0x1ff
   0x08049257 <+25>:    jbe    0x804926d <copy+47>
   0x08049259 <+27>:    push   0x804a008
   0x0804925e <+32>:    call   0x8049070 <puts@plt>
   0x08049263 <+37>:    add    esp,0x4
   0x08049266 <+40>:    push   0xffffffff
   0x08049268 <+42>:    call   0x8049080 <exit@plt>
   0x0804926d <+47>:    mov    eax,DWORD PTR [ebp+0xc]
   0x08049270 <+50>:    push   eax
   0x08049271 <+51>:    push   DWORD PTR [ebp+0x8]
   0x08049274 <+54>:    lea    eax,[ebp-0x200]
   0x0804927a <+60>:    push   eax
   0x0804927b <+61>:    call   0x80490c0 <strncpy@plt>
   0x08049280 <+66>:    add    esp,0xc
   0x08049283 <+69>:    nop
   0x08049284 <+70>:    leave
   0x08049285 <+71>:    ret
```

usage:

$ ./target **str** **n**

seems promising..

strncpy(ebp-0x200, **str**, **n**)

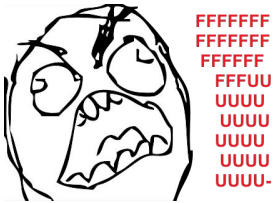
Analysis

```
Dump of assembler code for function copy:
   0x0804923e <+0>:     push   ebp
   0x0804923f <+1>:     mov    ebp,esp
   0x08049241 <+3>:     sub    esp,0x200
   0x08049247 <+9>:     push   DWORD PTR [ebp+0x8]
   0x0804924a <+12>:    call   0x80490a0 <strlen@plt>
   0x0804924f <+17>:    add    esp,0x4
   0x08049252 <+20>:    cmp    eax,0x1ff
   0x08049257 <+25>:    jbe    0x804926d <copy+47>
   0x08049259 <+27>:    push   0x804a008
   0x0804925e <+32>:    call   0x8049070 <puts@plt>
   0x08049263 <+37>:    add    esp,0x4
   0x08049266 <+40>:    push   0xffffffff
   0x08049268 <+42>:    call   0x8049080 <exit@plt>
   0x0804926d <+47>:    mov    eax,DWORD PTR [ebp+0xc]
   0x08049270 <+50>:    push   eax
   0x08049271 <+51>:    push   DWORD PTR [ebp+0x8]
   0x08049274 <+54>:    lea    eax,[ebp-0x200]
   0x0804927a <+60>:    push   eax
   0x0804927b <+61>:    call   0x80490c0 <strncpy@plt>
   0x08049280 <+66>:    add    esp,0xc
   0x08049283 <+69>:    nop
   0x08049284 <+70>:    leave
   0x08049285 <+71>:    ret
```

usage:

$ ./target **str** **n**

asserts strlen(**str**) < 0x1FF



strncpy(ebp-0x200, **str**, **n**)

Seems there’s no way we can overwrite outside of the buffer.

EXCEPT ..

Seems there’s no way we can overwrite outside of the buffer.

EXCEPT ..

zeroing out!

man strncpy

```
DESCRIPTION
       The  strcpy()  function  copies  the  string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest.  The
       strings may not overlap, and the destination string dest must be large enough to receive the copy.  Beware of buffer overruns!  (See BUGS.)

       The strncpy() function is similar, except that at most n bytes of src are copied.  Warning: If there is no null byte among the first  n  bytes  of
       src, the string placed in dest will not be null-terminated.

       If the length of src is less than n, strncpy() writes additional null bytes to dest to ensure that a total of n bytes are written.

       A simple implementation of strncpy() might be:

           char *
           strncpy(char *dest, const char *src, size_t n)
           {
               size_t i;

               for (i = 0; i < n && src[i] != '\0'; i++)
                   dest[i] = src[i];
               for ( ; i < n; i++)
                   dest[i] = '\0';

               return dest;
           }
```

Observation: strncpy(buffer, “hello”, 100) gives

h	e	l	l	0	\0	\0	\0	\0	\0	\0	\0	…	\0

Potential Exploit

usage:

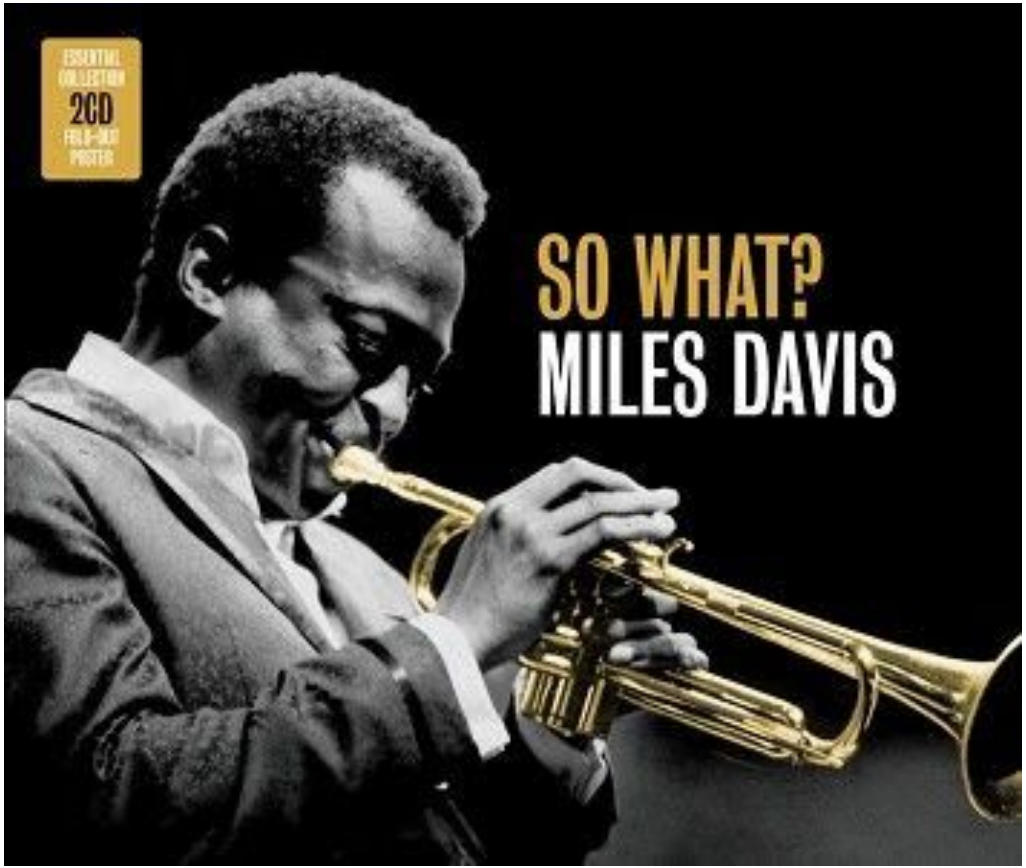
$ ./target **str** **n**

exploit:

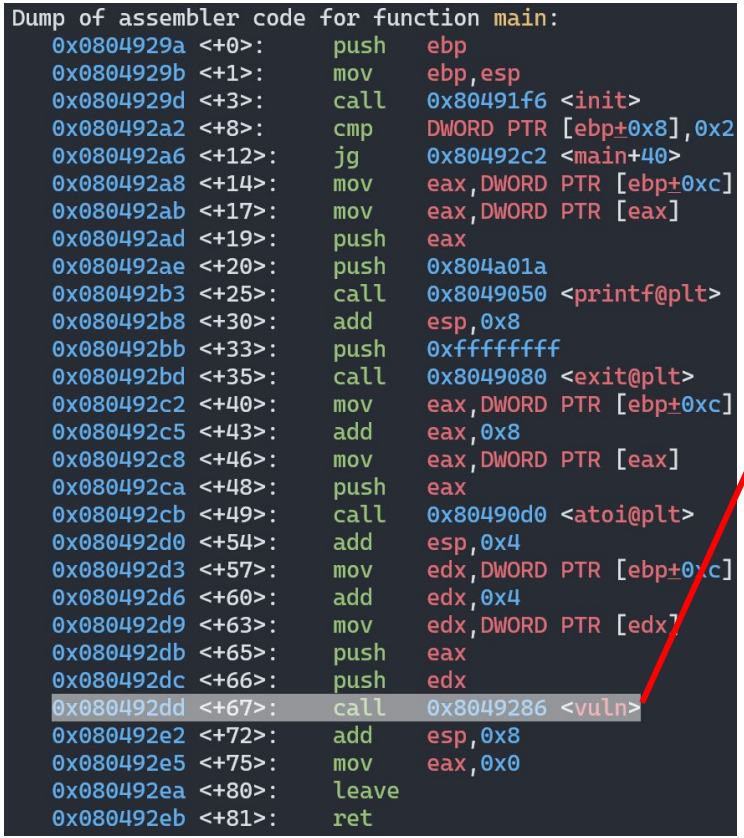
$ ./target **payload** **n**

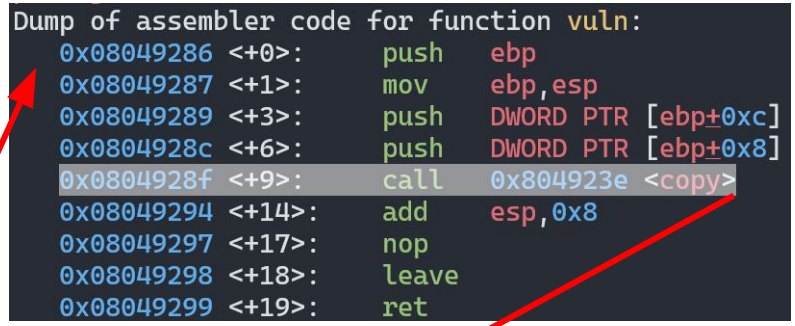
, where len(**payload**) < 0x1FF and **n** > 0x200

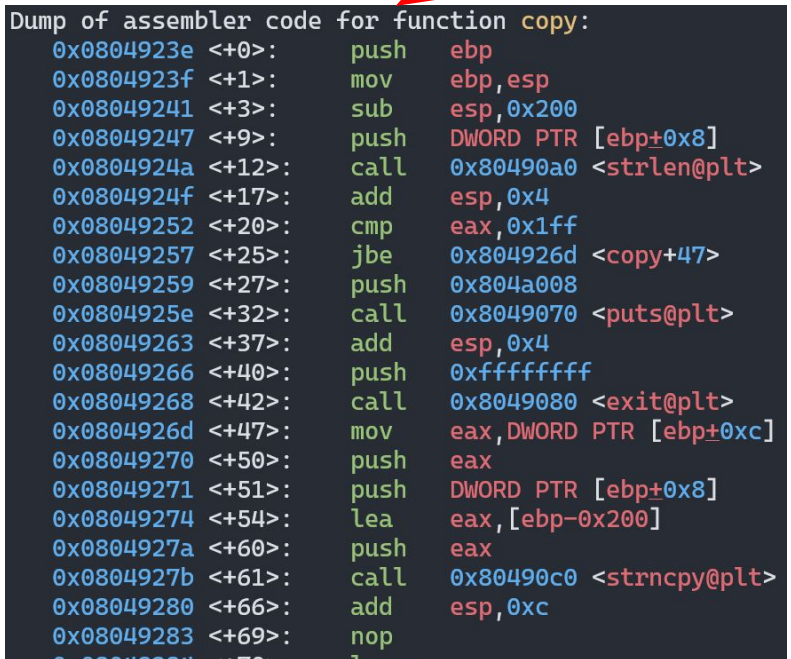
.. and it will zero out memory below ebp!


ESSENTIAL COLLECTION
2CD
FOLD-OUT POSTER
SO WHAT?
MILES DAVIS

Yet another CRUCIAL observation ..


Dump of assembler code for function main:
0x0804929a <+0>: push ebp
0x0804929b <+1>: mov ebp,esp
0x0804929d <+3>: call 0x80491f6 <init>
0x080492a2 <+8>: cmp DWORD PTR [ebp+0x8],0x2
0x080492a6 <+12>: jg 0x80492c2 <main+40>
0x080492a8 <+14>: mov eax,DWORD PTR [ebp+0xc]
0x080492ab <+17>: mov eax,DWORD PTR [eax]
0x080492ad <+19>: push eax
0x080492ae <+20>: push 0x804a01a
0x080492b3 <+25>: call 0x8049050 <printf@plt>
0x080492b8 <+30>: add esp,0x8
0x080492bb <+33>: push 0xffffffff
0x080492bd <+35>: call 0x8049080 <exit@plt>
0x080492c2 <+40>: mov eax,DWORD PTR [ebp+0xc]
0x080492c5 <+43>: add eax,0x8
0x080492c8 <+46>: mov eax,DWORD PTR [eax]
0x080492ca <+48>: push eax
0x080492cb <+49>: call 0x80490d0 <atoi@plt>
0x080492d0 <+54>: add esp,0x4
0x080492d3 <+57>: mov edx,DWORD PTR [ebp+0xc]
0x080492d6 <+60>: add edx,0x4
0x080492d9 <+63>: mov edx,DWORD PTR [edx]
0x080492db <+65>: push eax
0x080492dc <+66>: push edx
0x080492dd <+67>: call 0x8049286 <vuln>
0x080492e2 <+72>: add esp,0x8
0x080492e5 <+75>: mov eax,0x0
0x080492ea <+80>: leave
0x080492eb <+81>: ret


Dump of assembler code for function vuln:
0x08049286 <+0>: push ebp
0x08049287 <+1>: mov ebp,esp
0x08049289 <+3>: push DWORD PTR [ebp+0xc]
0x0804928c <+6>: push DWORD PTR [ebp+0x8]
0x0804928f <+9>: call 0x804923e <copy>
0x08049294 <+14>: add esp,0x8
0x08049297 <+17>: nop
0x08049298 <+18>: leave
0x08049299 <+19>: ret


Dump of assembler code for function copy:
0x0804923e <+0>: push ebp
0x0804923f <+1>: mov ebp,esp
0x08049241 <+3>: sub esp,0x200
0x08049247 <+9>: push DWORD PTR [ebp+0x8]
0x0804924a <+12>: call 0x80490a0 <strlen@plt>
0x0804924f <+17>: add esp,0x4
0x08049252 <+20>: cmp eax,0x1ff
0x08049257 <+25>: jbe 0x804926d <copy+47>
0x08049259 <+27>: push 0x804a008
0x0804925e <+32>: call 0x8049070 <puts@plt>
0x08049263 <+37>: add esp,0x4
0x08049266 <+40>: push 0xffffffff
0x08049268 <+42>: call 0x8049080 <exit@plt>
0x0804926d <+47>: mov eax,DWORD PTR [ebp+0xc]
0x08049270 <+50>: push eax
0x08049271 <+51>: push DWORD PTR [ebp+0x8]
0x08049274 <+54>: lea eax,[ebp-0x200]
0x0804927a <+60>: push eax
0x0804927b <+61>: call 0x80490c0 <strncpy@plt>
0x08049280 <+66>: add esp,0xc
0x08049283 <+69>: nop

Observation: nested functions again!

Observation: nested functions again!

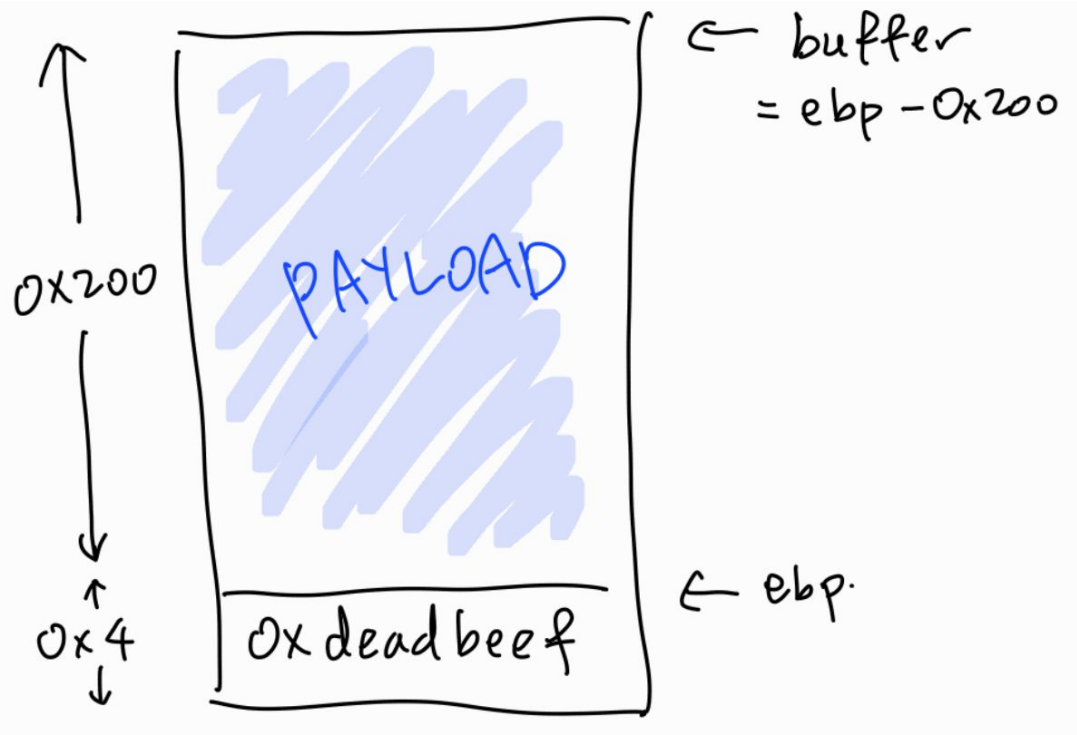
Hint: steal the ideal from the previous challenge!

Observation: nested functions again!

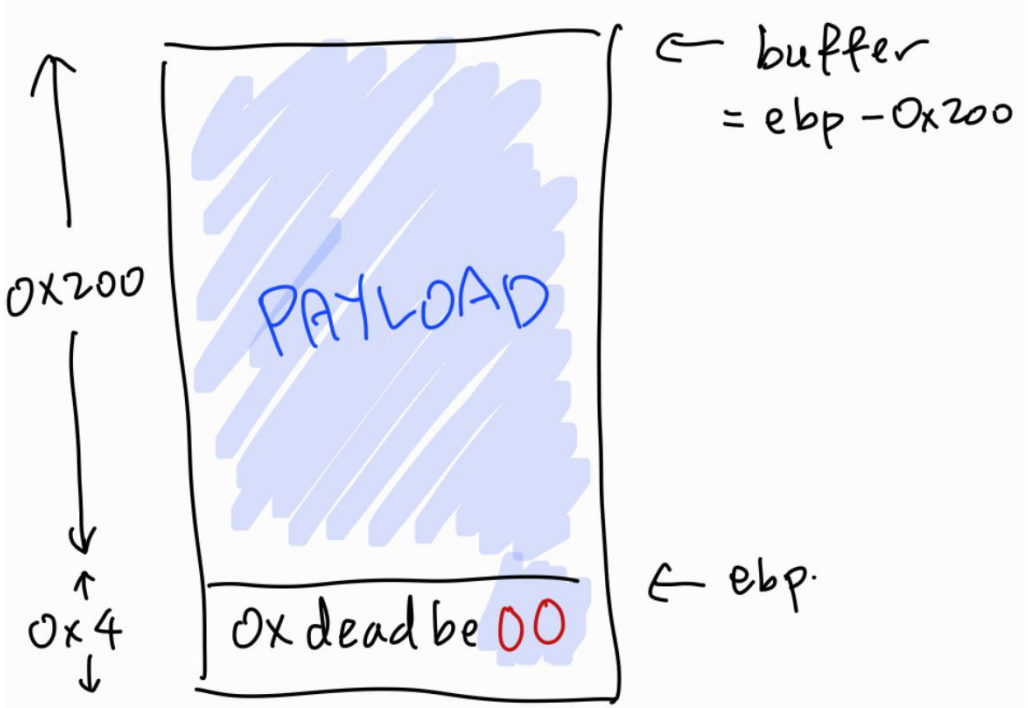
Hint: steal the ideal from the previous challenge!

Key Takeaway from shortage:
“**ebp is all you need**” for control hijacking

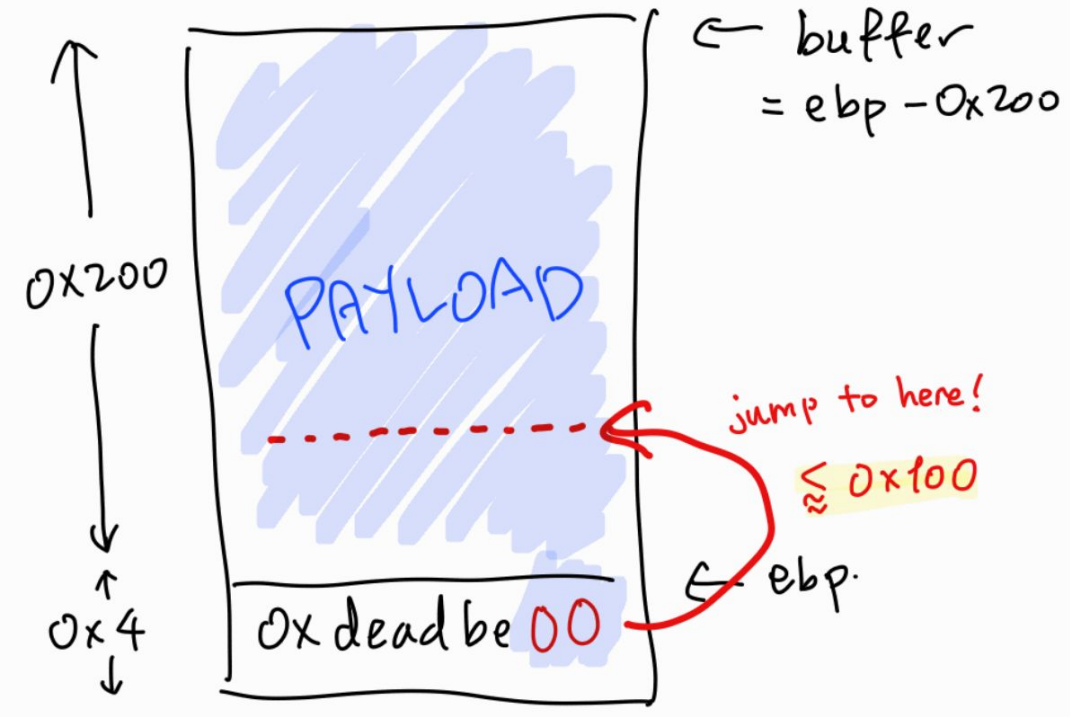
Memory Layout



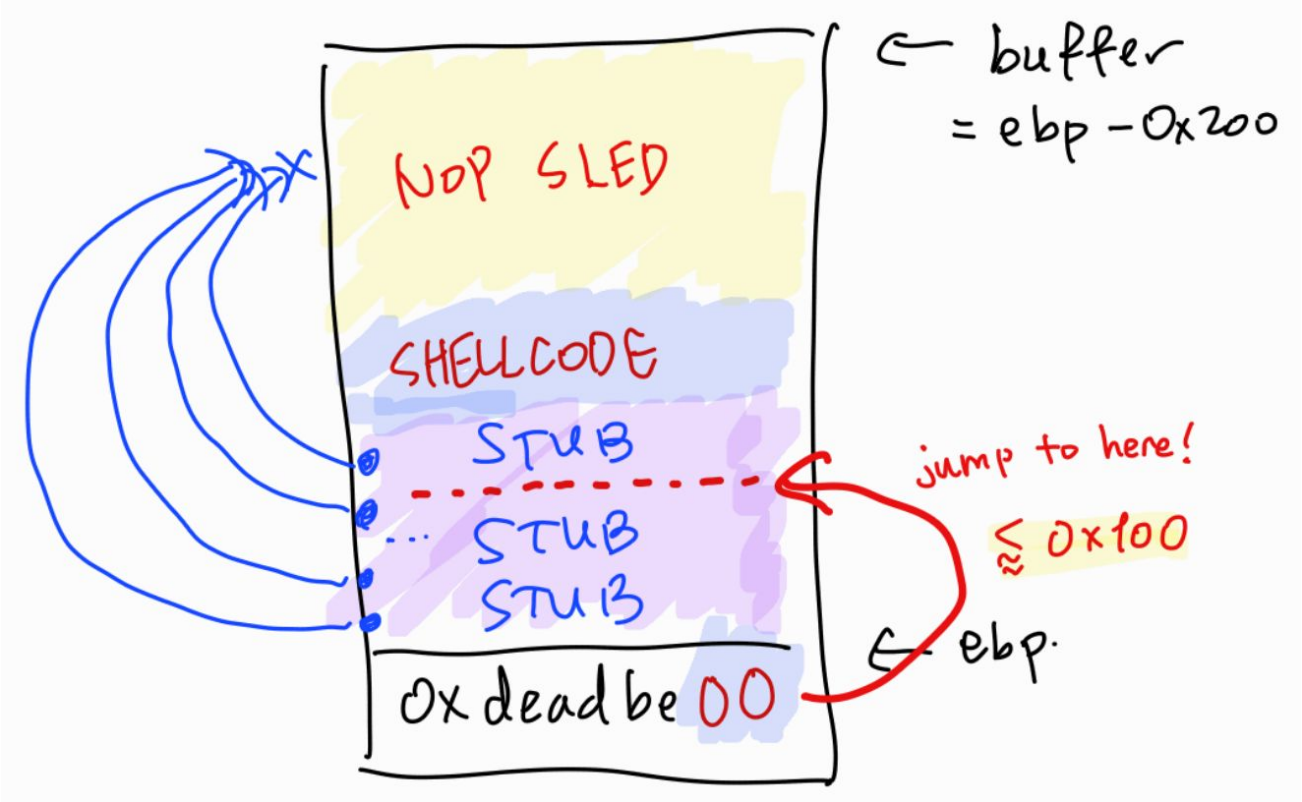
./target payload 0x201



leave = mov esi, ebp; pop ebp



Redirection along ebp -> stub -> nop sled



Q. but why ebp? why not overwrite ret directly in the first place, if we can overwrite ret?

A. because the address in ebp is close to the buffer!

Thank you!