Common Memory Errors

C++ Object Oriented Programming

Pei-yih Ting

NTOU CS

Main Categories of Errors

- ✧ Memory leakage
  - allocate, allocate, allocate …. without free
- ✧ Unallocated memory
  - use memory without preparation
- ✧ Memory corruption
  - underrun, overrun your buffer, runaway pointer
- ✧ Illegal access
  - use memory after you free it, runaway (wild) pointer, null pointer access

Early Versions of Microsoft Windows System/ Tools are good examples, you blame the M$ company for it, but you are following suit unconsciously

Your First Memory Trap in C

- ✧ Passing an arbitrary integer as the address
- ✧ Example:

```
int x=0;
….
scanf("%d", x);
```

- ✶ Often cause illegal memory access, fortunately, abort the program execution on the spot
- ✶ Sometimes, unfortunately, this error does not halt the program right at this line ….
- ✶ Should be scanf("%d", &x);

Where is the address?

- ✧ Case 1: address got lost

```
{
   char *leakage1;
   leakage1 = (char *) malloc(5*sizeof(char));
}
// There is no way to access that 5-byte memory any more.
```

- ✧ Case 2: address got overwritten

```
char *leakage2;
leakage2 = (char *) malloc(5*sizeof(char));
  ...
leakage2 = "hello";
```

Cause memory leakages, some of your virtual memory will not be used by your process anymore? Your program is going to crash someday for insufficient resources. Don't blame the system for it!

Use Memory W/O Allocation

- Oh! Make sure the chair is in place before you sit down!!
- Case 1: reading something out of the air

```
char *msg;
printf("%s\n", msg); // printing something, but WHAT is it?
```

- Case 1':

```
int *ptr;
somefun(*ptr);
```

- Case 2: writing something into the air

```
char *buffer;
strcpy(buffer, "some data"); // where do you think you copy to
scanf("%s", buffer); // where do you think you read into
```

- Case 2':

```
int *ptr;
*ptr = 10;
```

Use Memory W/O Allocation

- Sometimes CAUSE
  - Illegal memory access
    - If the memory address is 0 or pointed to somewhere you have no right to read/write in the memory
    - Turbo C/ Borland C famous error: null pointer assignment
  - Unexpected (but legal) memory content changes
    - Wild pointers: your code might overwrite some useful data in the program (maintained by yourself or by your teammate)
- They are all RUN TIME errors. Most troublesome, they are not necessarily hanging on each execution or on a specific line of codes

Overrun The Buffer

- The notorious BUFFER OVERFLOW attacks:
  - created daily, casually by numerous naïve, benign programmers
  - Do NOT think that you ruin at most your program only!!

    If your program is privileged, you open your system up!!
- Case 1:

```
char *buf;
buf = (char *) malloc(5*sizeof(char));
strcpy(buf,"abcde");
```

- Case 2:

```
int data[1000], i;
for (i=0; i<=1000; i++)
    data[i] = i;
```

although still not harmful in these two cases.

You must have destroy something useful in the memory!!

CERT Advisories

- http://www.cert.org/advisories
- Starting from 1988, **Buffer Overflow** vulnerabilities are the most common break-in courses.
- 2003 Jan-Mar: 7/13 advisories are about Buffer Overflow
  - CA-2003-12 :Buffer Overflow in Sendmail Mar 29 2003
  - CA-2003-10 :Integer overflow in Sun RPC XDR library routines Mar 19 2003
  - CA-2003-09 :Buffer Overflow in Core Microsoft Windows DLL Updated Mar 19 2003
  - CA-2003-07 :Remote Buffer Overflow in Sendmail Mar. 3, 2003
  - CA-2003-04 :MS-SQL Server Worm(SQL Slammer) Jan 25 2003
  - CA-2003-03 :Buffer Overflow in Windows Locator Service Jan 23 2003
  - CA-2003-01 :Buffer Overflows in ISC DHCPD Minires Library Jan 15 2003

Example: Changing the control flow

- What is the output of the following program?

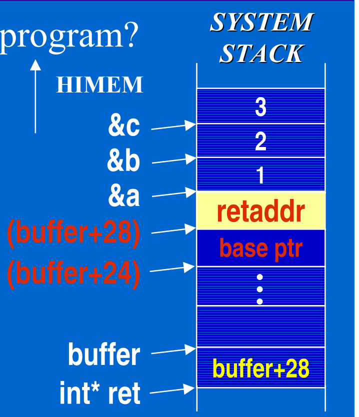
```
void function(int a, int b, int c) {
    char buffer[5];
    int *ret;
    ret = buffer + 28;
    (*ret) += 10;
}
int main() {
   int x;
   x = 0;
   function(1,2,3);
   x = 1;
   printf("x = %d\n",x); // unmodified by x=1;!!
   return 0;
}
```

tampering statement

retaddr

retaddr+10

Output: x = 0



Example: modified function pointer

```
void fun1() {                    void fun2() {
     …                                …
}                                }
typedef void (*FP)();
void main() {
   FP fp;
   char buffer[8];
   fp = fun1;
      …
   (FP)(buffer+8) = fun2;
      …
   (*fp)();
}
```

tampering statement

Which function does it call?

Buffer Overflow Attack

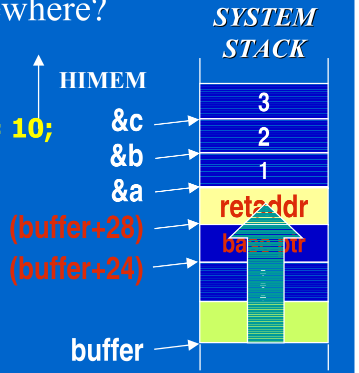
- Cause the program to jump to somewhere?

```
void function(int a, int b, int c) {
    char buffer[5];
    gets(buffer);
    int *ret = buffer + 28; (*ret) += 10;
}
int main() {
   int x;
   x = 0;
   function(1,2,3);
   x = 1;
   printf("x = %d\n",x); // unmodified by x=1;!!
   return 0;
}
```

Problematic statement

retaddr

retaddr+10



- What happened if the destination has a segment of malicious code!!!

Unsafe functions in C library

- strcpy(char *dest, const char *src) ;
- strcat(char *dest, const char *src) ;
- getwd(char *buf) ;
- gets(char *s) ;
- fscanf(FILE *stream, const char *format, ...) ;
- scanf(const char *format, ...) ;
- sscanf(char *str, const char *format, …);
- realpath(char *path, char resolved_path[]) ;
- sprintf(char *str, const char *format) ;
- syslog
- getopt
- getpass

String Operations Without '\0'

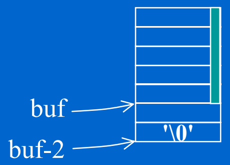
- Cause buffer overflow

```
char buf1[5], buf2[5];
buf1[0] = 'a';
buf1[1] = 'b';
strcpy(buf2, buf1); // don’t know what would happen,
                    // buf2 most probably overwritten
  ...
printf("%s\n",buf1); // don’t know what would happen,
                     // the printf statement does not just print
                     // out “ab” but “ab(*&%^^$$%&*^…”
```

Underrun The Buffer

- Case 1:

```
char *buf;
buf = (char *) malloc(5*sizeof(char));
  ... buf-- … buf-- …
*buf = '\0';
```



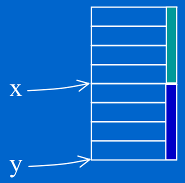
- Case 2:

```
char buf[5];
  …
*(buf - 2) = 'a'
```

Extraneous pointer access is evil.

- Case 3:

```
int x;
char y[4];
scanf("%d", &x); scanf("%d", &y[2]);
```

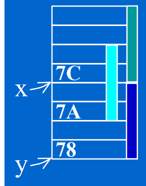


Probe into the Memory

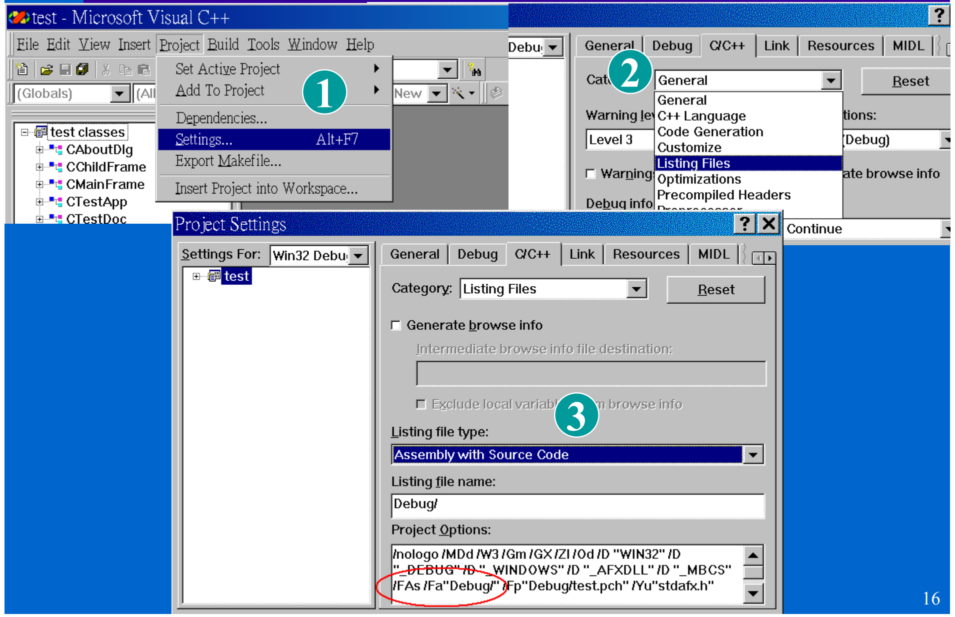
- Using compiler listing to see the memory layout

```
// cl /FAs /FatestBuf.asm testBuf.c
#include <stdio.h>
void main()
{
  int x;
  char y[4];
  scanf("%d", &x);
  printf("x=%d\n", x);
  printf("&x=%p &y=%p &y[2]=%p\n", &x, y, &y[2]);
  printf("%02x %02x %02x %02x %02x %02x %02x %02x\n",
         y[0],y[1],y[2],y[3],y[4],y[5],y[6],y[7]);
  scanf("%d", &y[2]);
  printf("%02x %02x %02x %02x %02x %02x %02x %02x\n",
         y[0],y[1],y[2],y[3],y[4],y[5],y[6],y[7]);
  printf("x=%d  %d\n", x, *((int *)&y[2]));
}
```

```
10
x=10
&x=0012FF7C &y=0012FF78 &y[2]=0012FF7A
00 00 00 00 0a 00 00 00
20
00 00 14 00 00 00 00 00
x=0  20
```



Visual Studio Environment



Settings For: Win32 Debu

Category: Listing Files

☐ Generate browse info

Intermediate browse info file destination:

☐ Exclude local variabl m browse info

Listing file type: Assembly with Source Code

Listing file name: Debug/

Project Options:

```
/nologo /MDd /W3 /Gm /GX /ZI /Od /D "WIN32" /D
"_DEBUG" /D "_WINDOWS" /D "_AFXDLL" /D "_MBCS"
/FAs /Fa"Debug/" /Fp"Debug/test.pch" /Yu"stdafx.h"
```

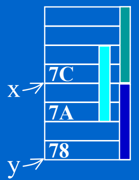
Compiler Assembly Listing

```
$SG772      DB     '%d', 00H
$SG776      DB     '%d', 00H
_x$ = -4
_y$ = -8
…
lea     eax, DWORD PTR _x$[ebp]
push    eax
push    OFFSET FLAT:$SG772
call    _scanf
```

scanf("%d", &x);

```
…
lea     ecx, DWORD PTR _y$[ebp+2]
push    ecx
push    OFFSET FLAT:$SG776
call    _scanf
…
```

scanf("%d", &y[2]);



Free Buffer Twice

- ✧ Cause runtime memory management internal error

```
char *buf;
buf = (char *) malloc(5*sizeof(char));
free(buf);
    …
free(buf);
```

```
char *buf;
buf = new char[200];
delete[] buf;
    …
delete[] buf;
```

Illegal Free

- ✧ Free an address not previously allocated:

```
char *buf, *ptr;
buf = (char *) malloc(5*sizeof(char));
ptr = buf; … ptr++; … ptr--; … ptr++; …
free(ptr);
```

- ✧ Free an automatic variable, a static variable, or a global variable:

```
char *ptr, array[100];
 …
ptr = array;
free(ptr);
```

Illegal Free (cont’d)

- ✧ Free null pointer:

```
char *buf=0;
free(buf)
```

- ✧ Free a character string constant

```
char *buf;
buf = (char *) malloc(6*sizeof(char));
…
buf = "hello";
…
free(buf);  // buf now contains the address of the string constant
```

Assess Freed Memory

- Case 1:

```
char *buf;
buf = (char *) malloc(5*sizeof(char));
   ...
free(buf);
strcpy(buf, "memory bomb");
```

- Case 2:

```
char *fun() {
   char *ptr, buf[10];
   ...
   ptr = buf;
   return ptr;
}
```

```
char *dataPtr, buf[20];
dataPtr = func();
   …
strcpy(buf, dataPtr);
   …
strcpy(dataPtr, buf);
```

- it is a common practice to forget any freed pointer values

```
free(ptr);
ptr = 0;
```

Dangling Pointers

- You might think that you would never commit the stupid errors in the previous slide.
- Modified case 1:

```
char *buf, *buf2;
buf = (char *) malloc(5*sizeof(char));
buf2 = buf;  // save the pointer somewhere else
   ...
free(buf);
   …
strcpy(buf2, "memory bomb through the dangling pointer");
```

Pointer Arithmetic Error

```
int (*ptr)[10], buf[20][10];

ptr = buf;
*(int *)(ptr + 199*sizeof(int)) = 20;  // Is it buf[19][9]?

// should be ptr[19][9] = 20;
//       or *((int *)(ptr + 19) + 9) = 20;
//       or *((int *)ptr + 199) = 20;
```

Careless pointer arithmetic produces **wild pointer**

Stack Overrun

- Case 1: large auto memory blocks

```
void func()
{
   double image[2000][2000];
      ...
}
```

  - Compiler would generate the code and hope that your system have this number of virtual memory allocated as the runtime stack

    2000*2000*8 = 32 M bytes

  - Visual C++ uses 1 M bytes stack as default, you can use

    /F2000000   to set the stack size as 2000000 bytes

Stack Overrun

- Case 2: deep recursive function call

```
void bizarrePrint(int n, int buf[]){
   int localBuf[1000];
   int i, pivot;
   if (n == 1){
      printDigit(n, buf);
      return;
   }
   else {
      for (i=0; i<5; i++) {
         pivot = n*i/5;
         copyDigit(localbuf, n/5, &buf[pivot]);
         bizarrePrint(n-1, localbuf);
      }
   }
}
```

```
int i;
int buf[2000];
for (i=0; i<2000; i++)
   buf[i] = i;
bizarrePrint(2000, buf);
```

2000 * 1000 * 4 = 8 M bytes

Unchecked Memory Allocation

- Case: malloc() might fail

```
int i, *ptr;
int n = 25000;
ptr = (int *) malloc(n*sizeof(int));
for (i=0; i<n; i++)
   ptr[i] = i;
```

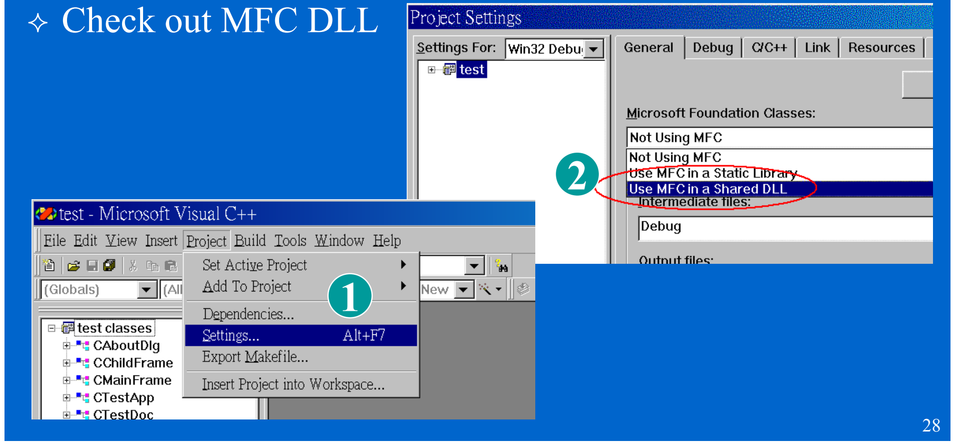
  - Cause illegal memory access if the allocation failed

Detecting Memory Errors

- MFC DLL
- VC++ Runtime Support
- Electric Fence
- wpr
- stack guard
- gcc (a version of it)
- object counts
- Memory checking API

Using MFC DLL

- #include <afx.h> in all your source files (at least the file that contains main())
- Using new/delete instead of malloc/free
- Check out MFC DLL



Microsoft Foundation Classes: Use MFC in a Shared DLL

Intermediate files: Debug

Using MFC DLL

- Source

```
#include <afx.h>
void main() {
    int *ptr;
    ptr = new int[100];
    ptr[0] = 1;
}
```

- Sample error messages

```
Detected memory leaks!
Dumping objects ->
{45} normal block at 0x003426C0, 400 bytes long.
 Data: <            > 01 00 00 00 CD CD CD CD CD CD CD CD CD
Object dump complete.
```

VC Runtime Leakage Detection (1/5)

- memory_leak.h

Step1: Initially set to zero, such that the memory manager would not break at any allocation.

Step2: set to a desired leakage object number so that the program breaks at the allocation of that object (you can identify which object is leaked in this way)

```
#ifndef MEMORY_LEAK_H
#define MEMORY_LEAK_H

    /* 1 to test for memory leaks */
    #define TEST_MEM_LEAKS      1
    #ifdef TEST_MEM_LEAKS

        /* allocation # at which to break */
        #define TEST_MEM_LEAKS_BREAK_NUM  0

        /* 1 to break at an allocation*/
        #define TEST_MEM_LEAKS_BREAK 1

        void set_initial_leak_test();

    #endif

#endif
```

VC Runtime Leakage Detection (2/5)

- memory_leak.cpp

```
#include "memory_leak.h"

#include <stdio.h>
#include <crtdbg.h>

void set_initial_leak_test(){

    int tmpFlag;

   /* set flag to automatically report memory leaks at image exit */
   printf("\n[Leak test being performed]\n");

   tmpFlag =  _CrtSetDbgFlag( _CRTDBG_REPORT_FLAG );
   …
```

VC Runtime Leakage Detection (3/5)

- In your program:
  - Step 1: #include "memory_leak.h"
  - Step 2: call set_initial_leak_test() at the start of main()
  - Step 3: **#define TEST_MEM_LEAKS_BREAK_NUM 0**
  - Step 4: compile your program, run your program
  - Step 5: observe the leakage report, ex. **cl /MLd /Zi …**

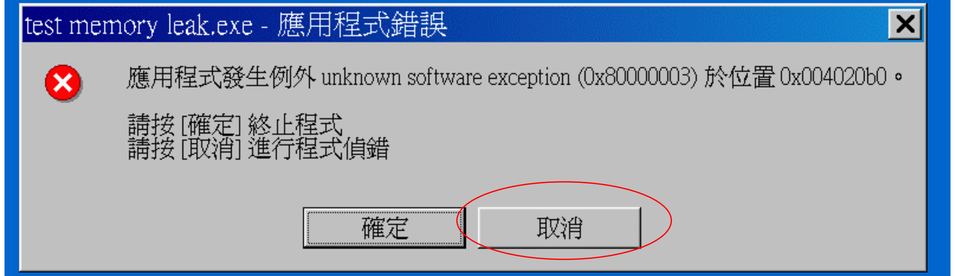
```
[Leak test being performed]
Detected memory leaks!
Dumping objects ->
{103} normal block at 0x009C6108, 10 bytes long.
 Data: <          > CD CD CD CD CD CD CD CD CD CD
Object dump complete.
```

  - Step 6: **#define TEST_MEM_LEAKS_BREAK_NUM 103**

VC Runtime Leakage Detection (4/5)

Step 7: compile your program, run your program again

Step 8: your program should now break at the allocation of that specified object. If you start the debugger



test memory leak.exe - 應用程式錯誤

應用程式發生例外 unknown software exception (0x80000003) 於位置 0x004020b0。

請按 [確定] 終止程式
請按 [取消] 進行程式偵錯

you can use call stack to see where your program allocates the leaked storage.

VC Runtime Leakage Detection (5/5)

Step 9: If you don’t start the debugger, you will observe the leakage report

```
[Leak test being performed]
Detected memory leaks!
Dumping objects ->
{102} normal block at 0x009C60D0, 10 bytes long.
 Data: <          > CD CD CD CD CD CD CD CD CD CD
…
{64} normal block at 0x009C2C80, 10 bytes long.
 Data: <          > CD CD CD CD CD CD CD CD CD CD
{63} normal block at 0x009C2C48, 10 bytes long.
 Data: <          > CD CD CD CD CD CD CD CD CD CD
Object dump complete.
Press any key to continue
```

Memory Checking Win 32 API

```
#include <windows.h>  // or #include <afx.h>
void mem() {
   MEMORYSTATUS stat;
   GlobalMemoryStatus(&stat);
   printf ("%ld percent of memory is in use.\n",
               stat.dwMemoryLoad);
   printf("TotalPhys=%d AvailPhys=%d\n",
               stat.dwTotalPhys, stat.dwAvailPhys);
   printf("TotalVirtual=%d AvailVirtual=%d\n",
               stat.dwTotalVirtual, stat.dwAvailVirtual);
}
```

DO NOT BE A NUISANCE!!

- Naturally you don’t want to be a TROUBLE in a group
- If everybody knows that you are a trouble, everybody can get used to it through some kinds of accommodation.
- Sometime, it is even worse that you are a trouble but you don’t know it.
- Having a programmer in a software team that ABUSE the memory in any of the previously listed ways is dangerous
- The biggest problem is that he is completely unaware of his blunder because the errors most likely do not show up immediately and he keeps generating bugs bugs and bugs.

Some C++ Memory Errors

- Unmatched new/new[] and delete/delete[]
- Pointer type coercion might change the values of it
- Incorrect down cast

Implementing Object Counts

- Sometimes, without the help of tools, you would like to monitor at run time whether your program has any unreleased objects and avoid memory leakages from the ground up.
- Implement with class variable

```
class MyClass {
…
public:
   MyClass();
   ~MyClass();
   static void printCounts();
private:
   static int objectCounts;
…
};
…
int MyClass::objectCounts=0;
```

```
MyClass::MyClass() {
   objectCounts++;
}
MyClass::~MyClass() {
   objectCounts--;
}
void MyClass::printCounts() {
   cout << "Class MyClass "
             "active objects: "
         << objectCounts << endl;
}
```