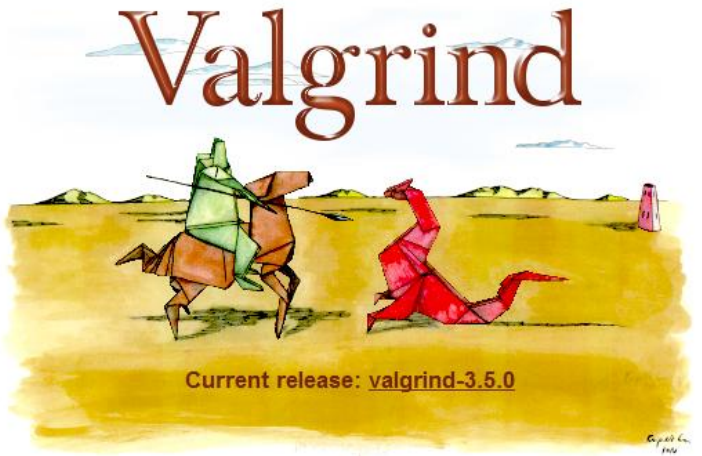
UCLA

Valgrind
Debugger and Profiler

NanoCAD LAB
Santiago Mok
smok@ucla.edu



Outline

- Symptoms and Common Memory Errors
- Valgrind Overview
  - Memcheck
- Examples:
  - Invalid Read/Write
  - Invalid Free
  - Memory Leaks

Symptoms (Disasters)

- Compilation Errors

```
[smok@dfm ~/Presentation]$ gcc -Wall a.c
a.c: In function `main':
a.c:6: error: syntax error before "int"
a.c:8: error: `y' undeclared (first use in this function)
a.c:8: error: (Each undeclared identifier is reported only once
a.c:8: error: for each function it appears in.)
```

- Run-Time Errors

```
[smok@dfm ~/Presentation]$ gcc -Wall a.c
[smok@dfm ~/Presentation]$ ./a.out
Segmentation fault
```

Memory Errors (BUGS)

- Heap
  - Free memory already freed
  - Using freed memory
  - Using uninitialized variables
  - Using undeclared memory
  - Read/Write to invalid bound
  - Memory leaks
- Stacks
  - Array index out of bound
  - Function pointer corruption

Valgrind Overview: Tool Suite

- **Memcheck**: memory error detector
- **Cachegrind**: cache and branch-prediction profiler
- **Callgrind**: call-graph generating cache profiler
- **Massif**: heap profiler
- **Helgrind**: thread error detector

```
valgrind [valgrind-options] your-prog [your-prog-options]
```

Memcheck

- Memcheck detects memory-management problems, and is aimed primarily at C and C++ programs.
- All reads and writes of memory are checked, and calls to malloc/new/free/delete are intercepted.

valgrind --tool=memcheck ls -l

```
[smok@dfm ~/Presentation]$ valgrind ls -l
==7711== Memcheck, a memory error detector
==7711== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==7711== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==7711== Command: ls -l
==7711==
total 12
-rw-r--r--  1 smok puneet  158 Apr  6 10:57 a.c
-rwxr-xr-x  1 smok puneet 6676 Apr  6 10:58 a.out
==7711==
==7711== HEAP SUMMARY:
==7711==     in use at exit: 19,634 bytes in 20 blocks
==7711==   total heap usage: 254 allocs, 234 frees, 214,307 bytes allocated
==7711==
==7711== LEAK SUMMARY:
==7711==    definitely lost: 52 bytes in 1 blocks
==7711==    indirectly lost: 240 bytes in 10 blocks
==7711==      possibly lost: 0 bytes in 0 blocks
==7711==    still reachable: 19,342 bytes in 9 blocks
==7711==         suppressed: 0 bytes in 0 blocks
==7711== Rerun with --leak-check=full to see details of leaked memory
==7711==
==7711== For counts of detected and suppressed errors, rerun with: -v
==7711== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```

Memcheck machinery

- Uses “V bit” for every bit of data to check validity

```
int i, j;
int a[10], b[10];
for ( i = 0; i < 10; i++ ) {
  j = a[i];
  b[i] = j;
}
```

```
for ( i = 0; i < 10; i++ ) {
  j += a[i];
}
if ( j == 77 )
  printf("hello there\n");
```

```
struct S { int x; char c; };
struct S s1, s2;
s1.x = 42;
s1.c = 'z';
s2 = s1;
```

- Uses “A bit” for every byte of data in memory to detect accessibility
- Set on malloc/new and reset on free/delete

Example

```c
#include <stdlib.h>

void func(){
    int *x = malloc(10 * sizeof(int));
    int *y = malloc(10 * sizeof(int));
    //int *x;
    x[10] = 0;
    y[2] = x[10];

}
int main(void){
    func();
    return 0;
}
```

```
$./a.out
$
```

- Successful compilation
- Successful execution
- When ran with valgrind, this returned errors.
- Can you guess why?

Running Example: invalid write/read and free

```c
#include <stdlib.h>

void func(){
    int *x = malloc(10 * sizeof(int));
    int *y = malloc(10 * sizeof(int));
    //int *x;
    x[10] = 0;
    y[2] = x[10];

}
int main(void){
    func();
    return 0;
}
```

valgrind ./a.out

```
==8120== Invalid write of size 4
==8120==    at 0x4004D4: func (a.c:7)
==8120==    by 0x4004FD: main (a.c:12)
==8120==  Address 0x4a21068 is 0 bytes after a block of size 40 alloc'd
==8120==    at 0x490514E: malloc (vg_replace_malloc.c:195)
==8120==    by 0x4004B9: func (a.c:4)
==8120==    by 0x4004FD: main (a.c:12)
==8120==
==8120== Invalid read of size 4
==8120==    at 0x4004EA: func (a.c:8)
==8120==    by 0x4004FD: main (a.c:12)
==8120==  Address 0x4a21068 is 0 bytes after a block of size 40 alloc'd
==8120==    at 0x490514E: malloc (vg_replace_malloc.c:195)
==8120==    by 0x4004B9: func (a.c:4)
==8120==    by 0x4004FD: main (a.c:12)
==8120==
==8120==
==8120== HEAP SUMMARY:
==8120==     in use at exit: 80 bytes in 2 blocks
==8120==   total heap usage: 2 allocs, 0 frees, 80 bytes allocated
==8120==
==8120== LEAK SUMMARY:
==8120==    definitely lost: 80 bytes in 2 blocks
==8120==    indirectly lost: 0 bytes in 0 blocks
==8120==      possibly lost: 0 bytes in 0 blocks
==8120==    still reachable: 0 bytes in 0 blocks
==8120==         suppressed: 0 bytes in 0 blocks
==8120== Rerun with --leak-check=full to see details of leaked memory
```

More Examples: uninitialized values

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

int main(void){
    int x;
    printf("x= %d\n",x);
    return 0;
}
```

```
==2697==
==2697== Use of uninitialised value of size 8
==2697==    at 0x37FA03EA7A: _itoa_word (in /lib64/tls/libc-2.3.4.so)
==2697==    by 0x37FA041E1C: vfprintf (in /lib64/tls/libc-2.3.4.so)
==2697==    by 0x37FA048607: printf (in /lib64/tls/libc-2.3.4.so)
==2697==    by 0x4004C1: main (b.c:6)
==2697==
==2697== Conditional jump or move depends on uninitialised value(s)
==2697==    at 0x37FA03EA85: _itoa_word (in /lib64/tls/libc-2.3.4.so)
==2697==    by 0x37FA041E1C: vfprintf (in /lib64/tls/libc-2.3.4.so)
==2697==    by 0x37FA048607: printf (in /lib64/tls/libc-2.3.4.so)
==2697==    by 0x4004C1: main (b.c:6)
```

--track-origins=<yes|no> [default: no]

- Invalid De-allocation

```
Mismatched free() / delete / delete []
   at 0x40043249: free (vg_clientfuncs.c:171)
   by 0x4102BB4E: QGArray::~QGArray(void) (tools/qgarray.cpp:149)
   by 0x4C261C41: PptDoc::~PptDoc(void) (include/qmemarray.h:60)
   by 0x4C261F0E: PptXml::~PptXml(void) (pptxml.cc:44)
 Address 0x4BB292A8 is 0 bytes inside a block of size 64 alloc'd
   at 0x4004318C: operator new[](unsigned int) (vg_clientfuncs.c:152)
   by 0x4C21BC15: KLaola::readSBStream(int) const (klaola.cc:314)
   by 0x4C21C155: KLaola::stream(KLaola::OLENode const *) (klaola.cc:416)
   by 0x4C21788F: OLEFilter::convert(QCString const &) (olefilter.cc:272)
```

Memory Leaks

- --leak-check=<no|summary|yes|full> [default: summary]

```
8 bytes in 1 blocks are definitely lost in loss record 1 of 14
   at 0x........: malloc (vg_replace_malloc.c:...)
   by 0x........: mk (leak-tree.c:11)
   by 0x........: main (leak-tree.c:39)

88 (8 direct, 80 indirect) bytes in 1 blocks are definitely lost in loss record 13 of 14
   at 0x........: malloc (vg_replace_malloc.c:...)
   by 0x........: mk (leak-tree.c:11)
   by 0x........: main (leak-tree.c:25)
```

```
LEAK SUMMARY:
   definitely lost: 48 bytes in 3 blocks.
   indirectly lost: 32 bytes in 2 blocks.
     possibly lost: 96 bytes in 6 blocks.
   still reachable: 64 bytes in 4 blocks.
        suppressed: 0 bytes in 0 blocks.
```

```
64 bytes in 4 blocks are still reachable in loss record 2 of 4
   at 0x........: malloc (vg_replace_malloc.c:177)
   by 0x........: mk (leak-cases.c:52)
   by 0x........: main (leak-cases.c:74)

32 bytes in 2 blocks are indirectly lost in loss record 1 of 4
   at 0x........: malloc (vg_replace_malloc.c:177)
   by 0x........: mk (leak-cases.c:52)
   by 0x........: main (leak-cases.c:80)
```

--show-reachable=<yes|no> [default: no]

Conclusion

- Common Memory Bugs:
  - Invalid Read/Write
  - Invalid Free
  - Leaks
- Memory problems lead to:
  - Performance
  - Security (buffer-overflow hack)

Valgrind 3.5.0
/w/apps.icsl/puneet/tools/valgrind-3.5/bin

References

- Valgrind Manual
  - http://www.valgrind.org/docs/manual/manual.html