Einführung in

Valgrind

(dynamische Speicheranalyse)

Lucas Beyer

Was ist valgrind?

- Sprich: „Wall“ + G + „Rind“
- Name des (gut bewachten) Haupteingangs zu Valhalla
  - Nordische Mythologie
  - Irrelevant
- Framework für Tools zur **dynamischen Analyse von Programmen**
- Simuliert Prozessor, Cache, RAM
  - Benötigt keine Änderung des Codes
    - Mehr Info wenn mit Debuginfo kompiliert
- Linux/Darwin
- Verfügbare Tools:
  - **Memcheck:** Erkennen von Fehlern im Speicherverhalten
  - **Cachegrind:** Profiler, auch für Cachelines und CPU Branchingprediction
  - **Helgrind, DRD:** Erkennen von Threadfehlern
    - Deadlocks
    - Race conditions
  - Viele weitere Tools
    - Weniger/seltener nützlich

Memcheck

- Unzulässige Speicherzugriffe
- Verwenden von uninitialisiertem Speicher
- Ungültige frees/deletes
- Memoryleaks
- Ungültige memcpy und ähnliche

Unzulässige Speicherzugriffe

- **Lesen und schreiben**
- **Buffer over/underrun**
- **Bereits befreiter Speicher**

```
1  #include <string.h>
2
3  int main()
4  {
5      char *buf = new char[3];
6      strcpy(buf, "Hi!");
7      char c = buf[4];
8      delete[] buf;
9      return c + buf[0] + buf[5];
10 }
```

```
==7578== Memcheck, a memory error detector
==7578== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==7578== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==7578== Command: ./a.out
==7578==
==7578== Invalid write of size 1
==7578==    at 0x4C28E5D: memcpy (mc_replace_strmem.c:497)
==7578==    by 0x4006EF: main (test_buffer.cpp:6)
==7578==  Address 0x5987043 is 0 bytes after a block of size 3 alloc'd
==7578==    at 0x4C27939: operator new[](unsigned long) (vg_replace_malloc.c:305)
==7578==    by 0x4006D5: main (test_buffer.cpp:5)
==7578==
==7578== Invalid read of size 1
==7578==    at 0x4006F8: main (test_buffer.cpp:7)
==7578==  Address 0x5987044 is 1 bytes after a block of size 3 alloc'd
==7578==    at 0x4C27939: operator new[](unsigned long) (vg_replace_malloc.c:305)
==7578==    by 0x4006D5: main (test_buffer.cpp:5)
==7578==
==7578== Invalid read of size 1
==7578==    at 0x400719: main (test_buffer.cpp:9)
==7578==  Address 0x5987040 is 0 bytes inside a block of size 3 free'd
==7578==    at 0x4C26A4B: operator delete[](void*) (vg_replace_malloc.c:409)
==7578==    by 0x400710: main (test_buffer.cpp:8)
==7578==
==7578== Invalid read of size 1
==7578==    at 0x400729: main (test_buffer.cpp:9)
==7578==  Address 0x5987045 is 2 bytes after a block of size 3 free'd
==7578==    at 0x4C26A4B: operator delete[](void*) (vg_replace_malloc.c:409)
==7578==    by 0x400710: main (test_buffer.cpp:8)
==7578==
==7578==
==7578== HEAP SUMMARY:
==7578==     in use at exit: 0 bytes in 0 blocks
==7578==   total heap usage: 1 allocs, 1 frees, 3 bytes allocated
==7578==
==7578== All heap blocks were freed -- no leaks are possible
==7578==
==7578== For counts of detected and suppressed errors, rerun with: -v
==7578== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 4 from 4)
```

Memcheck

- Unzulässige Speicherzugriffe
- Verwenden von uninitialisiertem Speicher
- Ungültige frees/deletes
- Memoryleaks
- Ungültige memcpy und ähnliche

Verwenden von uninit. Speicher

- **Achtung!**

```
1  int main()
2  {
3      int i;
4      int j = i + 2;
5      if(j > 3)
6          return 1;
7      else
8          return 0;
9  }
```

g++ -O2 -g

valgrind

```
==8207== Memcheck, a memory error detector
==8207== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==8207== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==8207== Command: ./a.out
==8207==
==8207==
==8207== HEAP SUMMARY:
==8207==     in use at exit: 0 bytes in 0 blocks
==8207==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==8207==
==8207== All heap blocks were freed -- no leaks are possible
==8207==
==8207== For counts of detected and suppressed errors, rerun with: -v
==8207== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```

Generiert

- **WTF!? Valgrind funktionniert nicht!!**

```
13              main:
14              .LFB0:
15                  .file 1 "test_uninit.cpp"
 1:test_uninit.cpp **** int main()
 2:test_uninit.cpp **** {
16                  .loc 1 2 0
17                  .cfi_startproc
18                  .cfi_personality 0x3,__gxx_personality_v0
 3:test_uninit.cpp ****     int i;
 4:test_uninit.cpp ****     int j = i + 2;
 5:test_uninit.cpp ****     if(j > 3)
 6:test_uninit.cpp ****         return 1;
 7:test_uninit.cpp ****     else
 8:test_uninit.cpp ****         return 0;
 9:test_uninit.cpp **** }
19                  .loc 1 9 0
20 0000 31C0        xorl    %eax, %eax
21 0002 C3          ret
22                  .cfi_endproc
```

Wo ist mein if?

- **C++ undefined behaviour**

Verwenden von uninit. Speicher

- **Moral der Geschicht'**
- **Ignoriere Compilerwarnungen nicht!**

- **G++ mit -Wall**

```
test_uninit2.cpp: In function ‘int main()’:
test_uninit2.cpp:5: warning: ‘i’ is used uninitialized in this function
```

Verwenden von uninit. Speicher

- **Nur wenn direkt oder indirekt verwendet in**
  - **Bedingung**
  - **Ausgabe**
- **Alles andere erlaubt**

```
1  int main()
2  {
3      int i;
4      int j = i + 2;
5      if(j > 3)
6          return 1;
7      else
8          return 0;
9  }
```

```
==8576== Memcheck, a memory error detector
==8576== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==8576== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==8576== Command: ./a.out
==8576==
==8576== Conditional jump or move depends on uninitialised value(s)
==8576==    at 0x4005C5: main (test_uninit.cpp:5)
==8576==
==8576==
==8576== HEAP SUMMARY:
==8576==     in use at exit: 0 bytes in 0 blocks
==8576==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==8576==
==8576== All heap blocks were freed -- no leaks are possible
==8576==
==8576== For counts of detected and suppressed errors, rerun with: -v
==8576== Use --track-origins=yes to see where uninitialised values come from
==8576== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)
```

```
==8624== Memcheck, a memory error detector
==8624== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==8624== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==8624== Command: ./a.out
==8624==
==8624== Conditional jump or move depends on uninitialised value(s)
==8624==    at 0x4005C5: main (test_uninit.cpp:5)
==8624==  Uninitialised value was created by a stack allocation
==8624==    at 0x4005B4: main (test_uninit.cpp:2)
==8624==
==8624==
==8624== HEAP SUMMARY:
==8624==     in use at exit: 0 bytes in 0 blocks
==8624==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==8624==
==8624== All heap blocks were freed -- no leaks are possible
==8624==
==8624== For counts of detected and suppressed errors, rerun with: -v
==8624== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)
```

Memcheck

- Unzulässige Speicherzugriffe
- Verwenden von uninitialisiertem Speicher
- Ungültige frees/deletes
- Memoryleaks
- Ungültige memcpy und ähnliche

Ungültige frees/deletes

- Double-frees
- Unalloced frees
- new/delete mismatch

```
1  #include <malloc.h>
2  int main()
3  {
4      char* s = new char[5];
5      delete s;
6      delete s;
7      return 0;
8  }
```

```
==8791== Memcheck, a memory error detector
==8791== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==8791== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==8791== Command: ./a.out
==8791==
==8791== Mismatched free() / delete / delete []
==8791==    at 0x4C26DCF: operator delete(void*) (vg_replace_malloc.c:387)
==8791==    by 0x400695: main (test_free.cpp:5)
==8791==  Address 0x5987040 is 0 bytes inside a block of size 5 alloc'd
==8791==    at 0x4C27939: operator new[](unsigned long) (vg_replace_malloc.c:305)
==8791==    by 0x400685: main (test_free.cpp:4)
==8791==
==8791== Invalid free() / delete / delete[]
==8791==    at 0x4C26DCF: operator delete(void*) (vg_replace_malloc.c:387)
==8791==    by 0x4006A1: main (test_free.cpp:6)
==8791==  Address 0x5987040 is 0 bytes inside a block of size 5 free'd
==8791==    at 0x4C26DCF: operator delete(void*) (vg_replace_malloc.c:387)
==8791==    by 0x400695: main (test_free.cpp:5)
==8791==
==8791==
==8791== HEAP SUMMARY:
==8791==     in use at exit: 0 bytes in 0 blocks
==8791==   total heap usage: 1 allocs, 2 frees, 5 bytes allocated
==8791==
==8791== All heap blocks were freed -- no leaks are possible
==8791==
==8791== For counts of detected and suppressed errors, rerun with: -v
==8791== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 4 from 4)
```

Memcheck

- Unzulässige Speicherzugriffe
- Verwenden von uninitialisiertem Speicher
- Ungültige frees/deletes
- Memoryleaks
- Ungültige memcpy und ähnliche

Memoryleaks

- Es ist die Rede von memory Blöcken
- „Still reachable“
  - Es gibt noch einen gültigen Pointer auf den Anfang des blocks
- „Definitely lost“
  - Es gibt keinen gültigen Pointer mehr
  - Typischer memory-leak
- „Indirectly lost“
  - Es gibt noch gültige Pointer
    - Jedoch befinden sich diese auch nur in verlorenem Speicher
  - Typischer Folgefehler
    - z.B. Baumstruktur, wenn die Wurzel verloren ist
- „Possibly lost“
  - Es gibt noch gültige Pointer, aber nur mitten in den Block
    - Zufall
    - Fehler
    - C++ new[] „magic cookie“

Memoryleaks

- „Still reachable“ und „Indirectly lost“
  - nur angezeigt, wenn –show-reachable=yes
  - Meist unwichtig

- „Possibly lost“
  - „Potential memory leak“-Kategorie in Cdash
  - Nicht zu unterschätzen, wegen C++ new[]

Memoryleaks

```
1  int* foo()
2  {
3    return new int[5];
4  }
5
6  int main()
7  {
8    int* j = foo();
9    j++;
10   return 0;
11 }
```

```
==9632== Memcheck, a memory error detector
==9632== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==9632== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==9632== Command: ./a.out
==9632==
==9632==
==9632== HEAP SUMMARY:
==9632==     in use at exit: 20 bytes in 1 blocks
==9632==   total heap usage: 1 allocs, 0 frees, 20 bytes allocated
==9632==
==9632== 20 bytes in 1 blocks are definitely lost in loss record 1 of 1
==9632==    at 0x4C27939: operator new[](unsigned long) (vg_replace_malloc.c:305)
==9632==    by 0x400631: foo() (test_leak.cpp:3)
==9632==    by 0x400640: main (test_leak.cpp:8)
==9632==
==9632== LEAK SUMMARY:
==9632==    definitely lost: 20 bytes in 1 blocks
==9632==    indirectly lost: 0 bytes in 0 blocks
==9632==      possibly lost: 0 bytes in 0 blocks
==9632==    still reachable: 0 bytes in 0 blocks
==9632==         suppressed: 0 bytes in 0 blocks
==9632==
==9632== For counts of detected and suppressed errors, rerun with: -v
==9632== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)
```

Memcheck

- Unzulässige Speicherzugriffe
- Verwenden von uninitialisiertem Speicher
- Ungültige frees/deletes
- Memoryleaks
- Ungültige memcpy und ähnliche
  - Input ptr und output ptr überlappen
  - Auch bei strcpy und strcat
    - Werden diese heutzutage überhaupt noch verwendet?

Bonusquiz 1

```
1  struct Base {
2      char* bla;
3
4      Base() : bla(new char[5]) {}
5      ~Base() {delete[] bla;}
6
7      virtual void f() {};
8  };
9
10 struct Deriv : public Base {
11     int* bli;
12
13     Deriv() : bli(new int) {}
14     ~Deriv() {delete bli;}
15
16     void f() {*bli = 3;};
17 };
18
19 int main()
20 {
21     Base* b = new Deriv();
22     delete b;
23 }
```

```
==9787== Memcheck, a memory error detector
==9787== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==9787== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==9787== Command: ./a.out
==9787==
==9787==
==9787== HEAP SUMMARY:
==9787==     in use at exit: 4 bytes in 1 blocks
==9787==   total heap usage: 3 allocs, 2 frees, 33 bytes allocated
==9787==
==9787== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==9787==    at 0x4C27CC1: operator new(unsigned long) (vg_replace_malloc.c:261)
==9787==    by 0x40098B: Deriv::Deriv() (quizz_leak.cpp:13)
==9787==    by 0x400868: main (quizz_leak.cpp:21)
==9787==
==9787== LEAK SUMMARY:
==9787==    definitely lost: 4 bytes in 1 blocks
==9787==    indirectly lost: 0 bytes in 0 blocks
==9787==      possibly lost: 0 bytes in 0 blocks
==9787==    still reachable: 0 bytes in 0 blocks
==9787==         suppressed: 0 bytes in 0 blocks
==9787==
==9787== For counts of detected and suppressed errors, rerun with: -v
==9787== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)
```

- **G++ option -Wnon-virtual-dtor ist einfacher**

```
quizz_leak.cpp:1: warning: 'struct Base' has virtual functions and accessible non-virtual destructor
quizz_leak.cpp:10: warning: 'struct Deriv' has virtual functions and accessible non-virtual destructor
```

Bonusquiz 2

```
1  #include <fstream>
2
3  int main()
4  {
5      int* i = new int;
6      int j = *i;
7      j = 2;
8      if(*i == 2) {
9          std::ofstream of("bla");
10         of << j;
11     }
12     return j;
13 }
```

```
==9868== Memcheck, a memory error detector
==9868== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==9868== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==9868== Command: ./a.out
==9868==
==9868== Conditional jump or move depends on uninitialised value(s)
==9868==    at 0x4008A7: main (quizz2.cpp:8)
==9868==  Uninitialised value was created by a heap allocation
==9868==    at 0x4C27CC1: operator new(unsigned long) (vg_replace_malloc.c:261)
==9868==    by 0x40087A: main (quizz2.cpp:5)
==9868==
==9868==
==9868== HEAP SUMMARY:
==9868==     in use at exit: 4 bytes in 1 blocks
==9868==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==9868==
==9868== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==9868==    at 0x4C27CC1: operator new(unsigned long) (vg_replace_malloc.c:261)
==9868==    by 0x40087A: main (quizz2.cpp:5)
==9868==
==9868== LEAK SUMMARY:
==9868==    definitely lost: 4 bytes in 1 blocks
==9868==    indirectly lost: 0 bytes in 0 blocks
==9868==      possibly lost: 0 bytes in 0 blocks
==9868==    still reachable: 0 bytes in 0 blocks
==9868==         suppressed: 0 bytes in 0 blocks
==9868==
==9868== For counts of detected and suppressed errors, rerun with: -v
==9868== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 4 from 4)
```