OSCI Exercises

Debugging with GDB & GEF

Winter term 2026/27

Manuel Vögele & Benedict Herzog

Bochum Operating Systems and System Software Group (BOSS)

RUHR UNIVERSITÄT BOCHUM

RUB

BOCHUM OPERATING SYSTEMS & SYSTEM SOFTWARE



Your PC ran into a problem that it couldn't handle, and now it needs to restart.

You can search for the error online: HAL_INITIALIZATION_FAILED

:(

Your PC ran into a problem that it couldn't handle, and now it needs to restart.

You can search for the error online: HAL_INITIALIZATION_FAILED



:(



Your PC ran into a prob[illegible] it couldn't handle, and now it needs to restart.

You can search for the error online: HAL_INITIALIZATION_FAILED

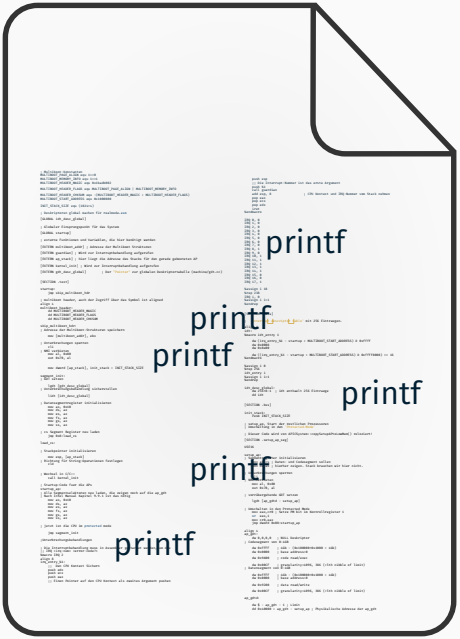


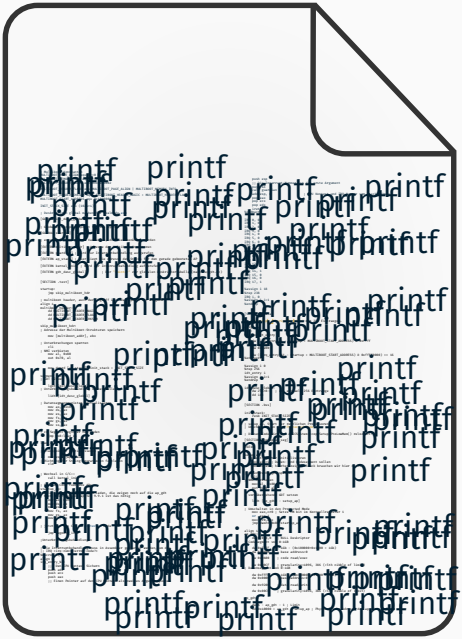
```
├── common.mk
├── kernel
│   ├── boot
│   │   ├── sections.ld
│   │   ├── startup.asm
│   │   └── startup.cc
│   ├── compiler.h
│   ├── config.h
│   ├── debug
│   │   ├── assert.cc
│   │   ├── assert.h
│   │   ├── gdb
│   │   │   ├── handler.asm
│   │   │   ├── handler.cc
│   │   │   ├── init.cc
│   │   │   ├── protocol.cc
│   │   │   └── stub.h
│   │   ├── kernelpanic.h
│   │   ├── null_stream.cc
│   │   ├── null_stream.h
│   │   └── output.h
│   ├── device
│   │   ├── cgastr.cc
│   │   ├── cgastr.h
│   │   ├── console.cc
│   │   ├── console.h
│   │   ├── keyboard.cc
│   │   ├── keyboard.h
│   │   ├── panic.cc
│   │   ├── panic.h
│   │   ├── watch.cc
│   │   └── watch.h
│   ├── guard
│   │   ├── gate.h
│   │   ├── guard.cc
│   │   ├── guard.h
│   │   ├── guardian.cc
│   │   ├── guardian.h
│   │   └── secure.h
│   ├── machine
│   │   ├── acpi.cc
│   │   ├── acpi.h
│   │   ├── apicsystem.cc
│   │   ├── apicsystem.h
│   │   ├── cgascr.cc
│   │   ├── cgascr.h
│   │   ├── cpu.asm
│   │   ├── cpu.h
│   │   ├── gdt.cc
│   │   ├── gdt.h
│   │   ├── ioapic.cc
│   │   ├── ioapic.h
│   │   ├── ioapic_registers.h
│   │   ├── io_port.h
│   │   ├── keyctrl.cc
│   │   ├── keyctrl.h
│   │   ├── keydecoder.cc
│   │   ├── keydecoder.h
│   │   ├── key.h
│   │   ├── lapic.cc
│   │   ├── lapic.h
│   │   ├── lapic_registers.h
│   │   ├── mp_registers.h
│   │   ├── plugbox.cc
│   │   ├── plugbox.h
│   │   ├── serial.cc
│   │   ├── serial.h
│   │   ├── spinlock.h
│   │   ├── ticketlock.h
│   │   ├── toc.asm
│   │   ├── toc.cc
│   │   ├── toc.h
│   │   └── toc.inc
│   ├── main.cc
│   ├── Makefile
│   ├── meeting
│   │   ├── bell.cc
│   │   ├── bell.h
│   │   ├── bellringer.cc
│   │   ├── bellringer.h
│   │   ├── messageinfo.h
│   │   ├── semaphore.cc
│   │   ├── semaphore.h
│   │   ├── waitingroom.cc
│   │   └── waitingroom.h
│   ├── memory
│   │   ├── allocator.cc
│   │   ├── allocator.h
│   │   ├── copy.cc
│   │   ├── copy.h
│   │   ├── initmem.cc
│   │   ├── initmem.h
│   │   ├── kernelpage.h
│   │   ├── malloc.cc
│   │   ├── malloc.h
│   │   ├── multiboot.h
│   │   ├── pagecont.cc
│   │   ├── pagecont.h
│   │   ├── pagedir.cc
│   │   ├── pagedir.h
│   │   ├── pagefault.cc
│   │   ├── pagefault.h
│   │   ├── page.h
│   │   ├── pageref.cc
│   │   ├── pageref.h
│   │   ├── range.h
│   │   └── user_buffer.h
│   ├── object
│   │   ├── bbuffer.h
│   │   ├── o_stream.cc
│   │   ├── o_stream.h
│   │   ├── queue.h
│   │   ├── queuelink.h
│   │   ├── strbuf.cc
│   │   └── strbuf.h
│   ├── syscall
│   │   ├── guarded_bell.cc
│   │   ├── guarded_bell.h
│   │   ├── guarded_keyboard.cc
│   │   ├── guarded_keyboard.h
│   │   ├── guarded_scheduler.h
│   │   ├── guarded_semaphore.h
│   │   ├── syscall.cc
│   │   └── syscall.h
│   ├── thread
│   │   ├── assassin.cc
│   │   ├── assassin.h
│   │   ├── dispatcher.cc
│   │   ├── dispatcher.h
│   │   ├── idlethread.cc
│   │   ├── idlethread.h
│   │   ├── scheduler.cc
│   │   ├── scheduler.h
│   │   ├── thread.cc
│   │   ├── thread.h
│   │   └── wakeup.h
│   ├── types.h
│   ├── user
│   │   ├── app1
│   │   │   ├── app1.cc
│   │   │   └── app1.h
│   │   └── app2
│   │       ├── kapp1.cc
│   │       └── kapp1.h
│   └── utils
│       ├── elf.cc
│       ├── elf.h
│       ├── libcc.cc
│       ├── math.h
│       ├── memutil.cc
│       ├── memutil.h
│       ├── sort.h
│       ├── string.cc
│       ├── string.h
│       ├── tar.cc
│       └── tar.h
├── libsys
│   ├── iostream.h
│   ├── Makefile
│   ├── o_stream.cc
│   ├── o_stream.h
│   ├── strbuf.cc
│   ├── strbuf.h
│   ├── syscall_stubs.asm
│   ├── syscall_stubs.cc
│   └── types.h
├── Makefile
├── README.md
├── test-stream
│   ├── console_out.cc
│   ├── console_out.h
│   ├── file_out.cc
│   ├── file_out.h
│   ├── Makefile
│   └── test.cc
└── user
    ├── app0
    │   ├── app0.cc
    │   └── app0.h
    ├── app1
    │   ├── app1.cc
    │   └── app1.h
    ├── app2
    │   ├── app2.cc
    │   └── app2.h
    ├── app3
    │   ├── app3.cc
    │   └── app3.h
    ├── app4
    │   ├── app4.cc
    │   └── app4.h
    ├── imgbuilder.cc
    ├── init.cc
    ├── Makefile
    ├── Makefile.app
    └── sections.ld

24 directories, 172 files
```




printf
printf

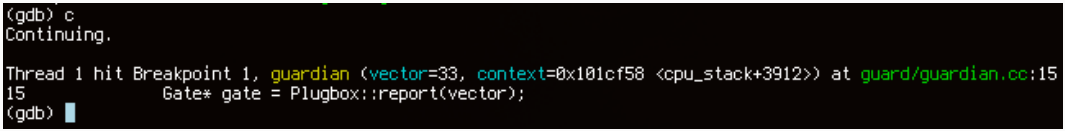

printf
printf
printf
printf
printf
printf


printf

Debugging with GDB & GEF

GNU Debugger (GDB)

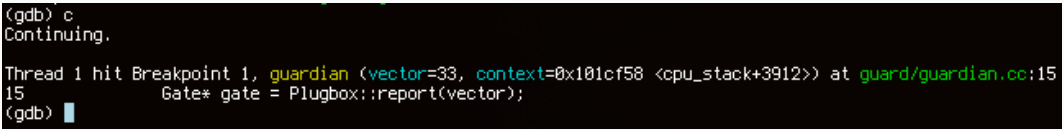
- \+ Inspecting the system state while the system is running
- \- Only rudimentary TUI



Debugging with GDB & GEF

GNU Debugger (GDB)

- \+ Inspecting the system state while the system is running
- \- Only rudimentary TUI



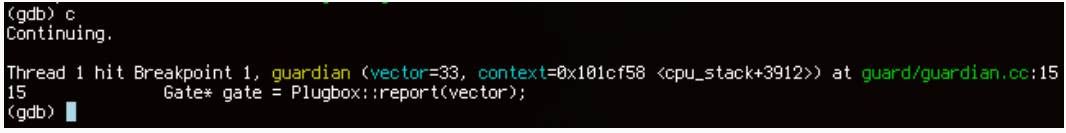
GDB Enhanced Features (GEF)

- \+ Extends GDB with a better interface
- \- in part visually overloaded

Debugging with GDB & GEF

GNU Debugger (GDB)

- \+ Inspecting the system state while the system is running
- \- Only rudimentary TUI

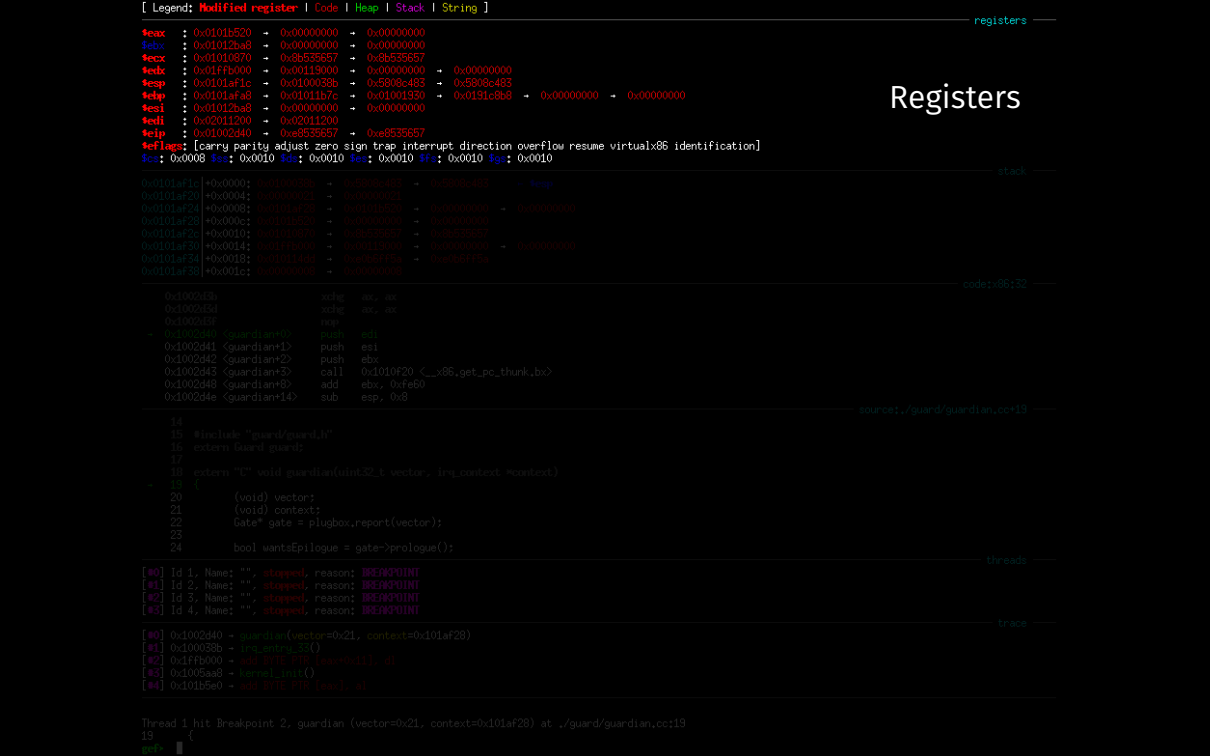


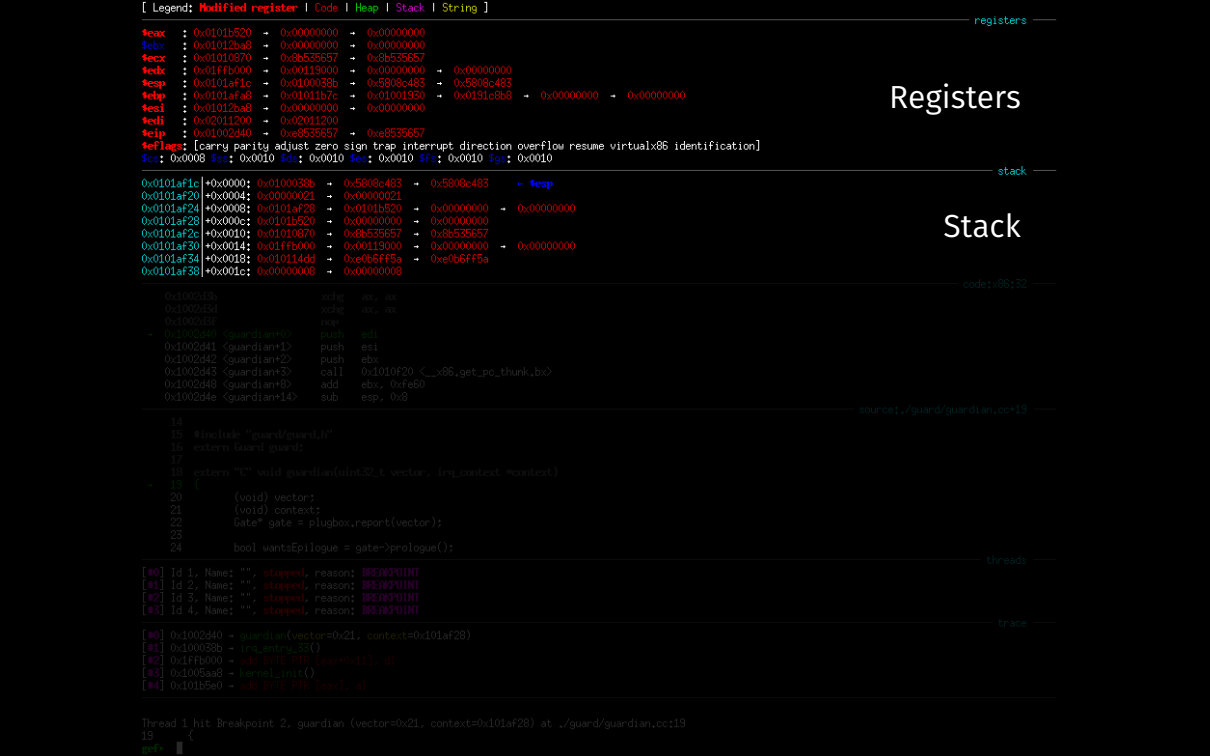
GDB Enhanced Features (GEF)

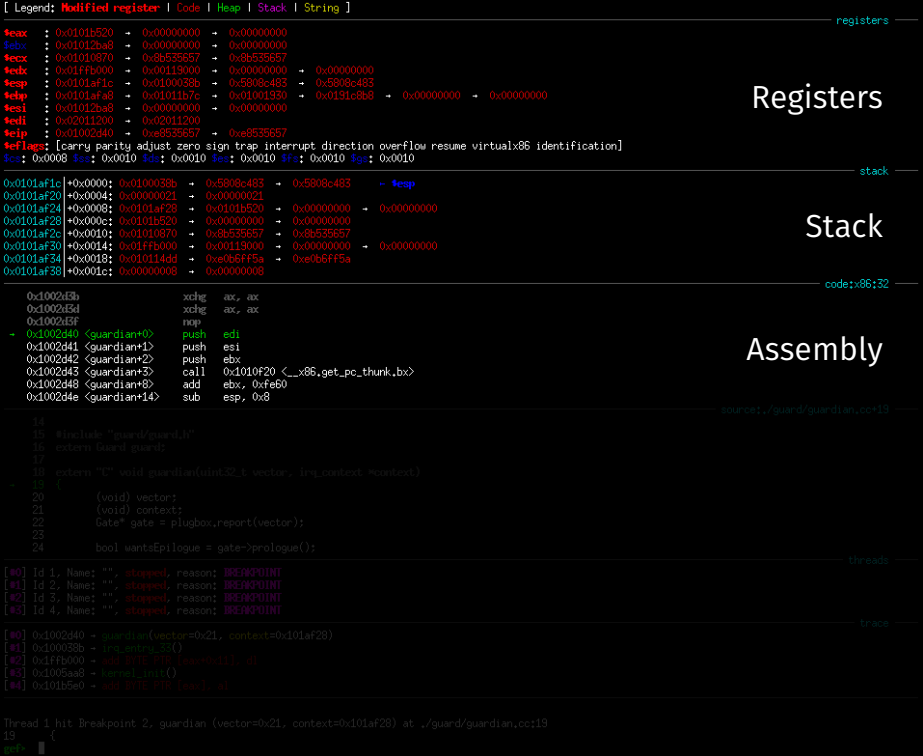
- \+ Extends GDB with a better interface
- \- in part visually overloaded

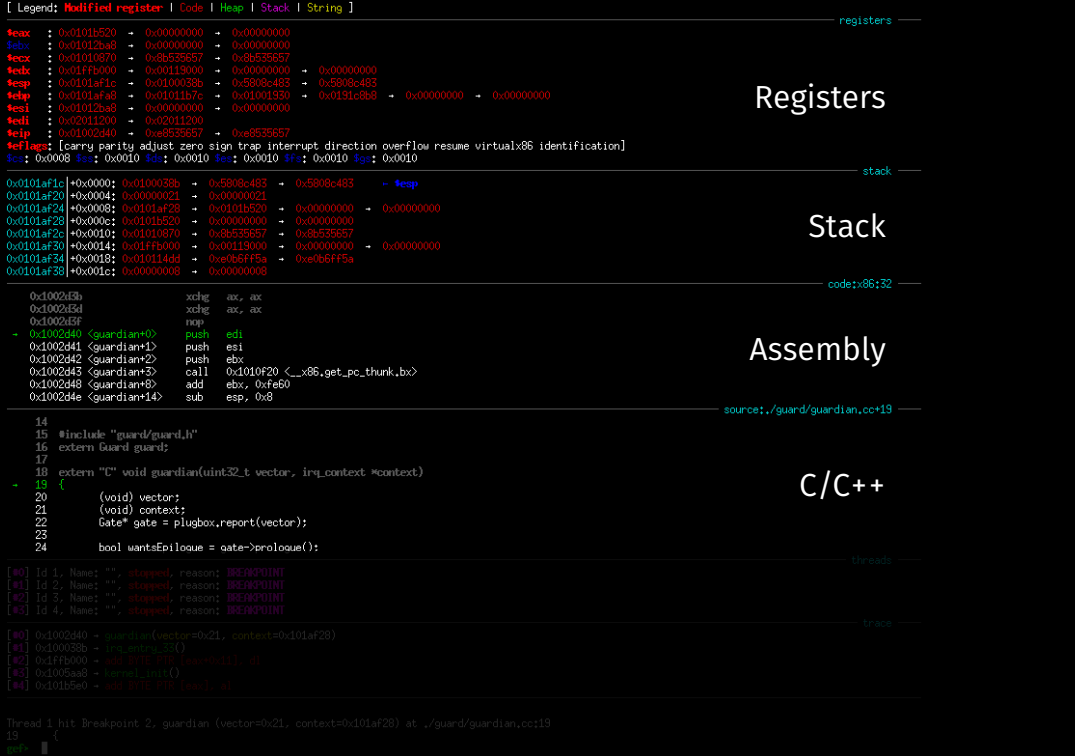
Editor/IDE integration

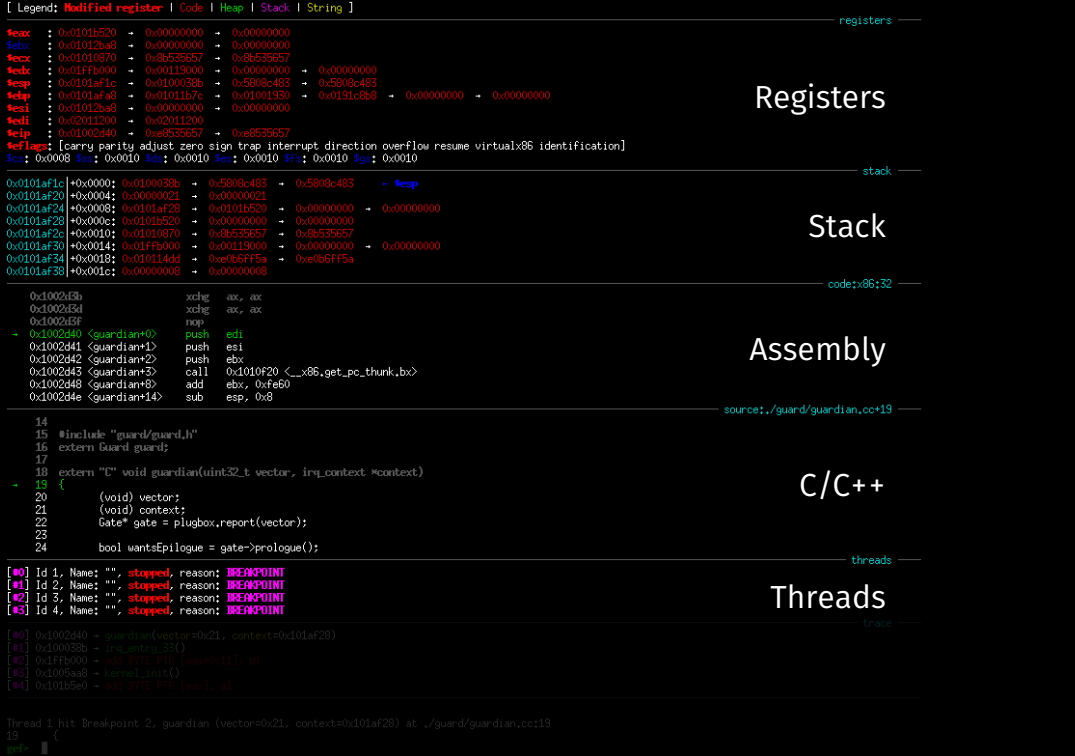
- \+ familiar environment
- \- setup overhead


[Legend: Modified register | Code | Heap | Stack | String]
registers
$eax : 0x0101b520 → 0x00000000 → 0x00000000
$ebx : 0x01012ba8 → 0x00000000 → 0x00000000
$ecx : 0x01010870 → 0x8b535657 → 0x8b535657
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x100038b → 0x5808e483 → 0x5808e483
$ebp : 0x0101af28 → 0x01011b7c → 0x01001930 → 0x0191c8b8 → 0x00000000 → 0x00000000
$esi : 0x01012ba8 → 0x00000000 → 0x00000000
$edi : 0x02011200 → 0x02011200
$eip : 0x01002d40 → 0xe8535657 → 0xe8535657
$eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
$cs: 0x0008 $ss: 0x0010 $ds: 0x0010 $es: 0x0010 $fs: 0x0010 $gs: 0x0010
Registers
stack
0x0101af1c|+0x0000: 0x100038b → 0x5808e483 → 0x5808e483 ← $esp
0x0101af20|+0x0004: 0x01ffb000 → 0x00119000 → 0x00000000
0x0101af24|+0x0008: 0x01012ba8 → 0x00000000 → 0x00000000 → 0x00000000
0x0101af28|+0x000c: 0x01011b7c → 0x01001930 → 0x0191c8b8 → 0x00000000
0x0101af2c|+0x0010: 0x01010870 → 0x8b535657 → 0x8b535657
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x01011f80 → 0x01ffb770 → 0x01ffb770
0x0101af38|+0x001c: 0x00000000 → 0x00000000
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d xchg ax, ax
0x1002d3f nop
→ 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk.bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8
source:./guard/guardian.cc+19
14
15 #include "guard/guard.h"
16 extern board guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
→ 19 {
20 (void) vector;
21 (void) context;
22 Gate* gate = plugbox.report(vector);
23
24 bool wantsEpilogue = gate->prologue();
threads
[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT
trace
[#0] 0x1002d40 → guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b → irq_entry_33()
[#2] 0x1ffb000 → add BYTE PTR [eax+0x1], dl
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [eax], al
Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef➤


[Legend: Modified register | Code | Heap | Stack | String]
registers
$eax : 0x0101b520 → 0x00000000 → 0x00000000
$ebx : 0x01012ba8 → 0x00000000 → 0x00000000
$ecx : 0x01010870 → 0x8b535657 → 0x8b535657
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x01011b7c → 0x01001930 → 0x0191c8b8 → 0x00000000 → 0x00000000
$esi : 0x01012ba8 → 0x00000000 → 0x00000000
$edi : 0x02011200 → 0x02011200
$eip : 0x01002d40 → 0xe8535657 → 0xe8535657
$eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
$cs: 0x0008 $ss: 0x0010 $ds: 0x0010 $es: 0x0010 $fs: 0x0010 $gs: 0x0010
Registers
stack
0x0101af1c|+0x0000: 0x0100038b → 0x5808c483 → 0x5808c483 ← $esp
0x0101af20|+0x0004: 0x00000021 → 0x00000021
0x0101af24|+0x0008: 0x0101af28 → 0x0101b520 → 0x00000000 → 0x00000000
0x0101af28|+0x000c: 0x0101b520 → 0x00000000 → 0x00000000
0x0101af2c|+0x0010: 0x01010870 → 0x8b535657 → 0x8b535657
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x010114dd → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stack
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d xchg ax, ax
0x1002d3f nop
0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk.bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8
source:./guard/guardian.cc+19
14
15 #include "guard/guard.h"
16 extern board guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
19 {
20 (void) vector;
21 (void) context;
22 Gate* gate = plugbox.report(vector);
23
24 bool wantsEpilogue = gate->prologue();
threads
[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT
trace
[#0] 0x1002d40 → guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b → irq_entry.33()
[#2] 0x1ffb000 → add BYTE PTR [eax+0x11], dl
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [eax], al
Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef➤


[Legend: Modified register | Code | Heap | Stack | String]
registers
$eax : 0x0101b520 → 0x00000000 → 0x00000000
$ebx : 0x01012ba8 → 0x00000000 → 0x00000000
$ecx : 0x01010870 → 0x8b535657 → 0x8b535657
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x01011b7c → 0x01001930 → 0x0191e8b8 → 0x00000000 → 0x00000000
$esi : 0x01012ba8 → 0x00000000 → 0x00000000
$edi : 0x02011200 → 0x02011200
$eip : 0x01002d40 → 0xe8535657 → 0xe8535657
$eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
$cs: 0x0008 $ss: 0x0010 $ds: 0x0010 $es: 0x0010 $fs: 0x0010 $gs: 0x0010
Registers
stack
0x0101af1c|+0x0000: 0x0100038b → 0x5808c483 → 0x5808c483 ← $esp
0x0101af20|+0x0004: 0x00000021 → 0x00000021
0x0101af24|+0x0008: 0x0101af28 → 0x01011b520 → 0x00000000 → 0x00000000
0x0101af28|+0x000c: 0x01011b520 → 0x00000000 → 0x00000000
0x0101af2c|+0x0010: 0x01010870 → 0x8b535657 → 0x8b535657
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x010114dd → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stack
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d xchg ax, ax
0x1002d3f nop
→ 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk.bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8
Assembly
source:./guard/guardian.cc+19
14
15 #include "guard/guard.h"
16 extern board guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
→ 19 {
20 (void) vector;
21 (void) context;
22 Gate* gate = plugbox.report(vector);
23
24 bool wantsEpilogue = gate->prologue();
threads
[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT
trace
[#0] 0x1002d40 → guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b → irq_entry_33()
[#2] 0x1ffb000 → add BYTE PTR [eax+0x31], dl
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [eax], al
Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef➤


[Legend: Modified register | Code | Heap | Stack | String]
registers
$eax : 0x0101b520 → 0x00000000 → 0x00000000
$ebx : 0x01012ba8 → 0x00000000 → 0x00000000
$ecx : 0x01010870 → 0x8b535657 → 0x8b535657
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x01011b7c → 0x01001930 → 0x0191e8b8 → 0x00000000 → 0x00000000
$esi : 0x01012ba8 → 0x00000000 → 0x00000000
$edi : 0x02011200 → 0x02011200
$eip : 0x01002d40 → 0xe8535657 → 0xe8535657
$eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
$cs: 0x0008 $ss: 0x0010 $ds: 0x0010 $es: 0x0010 $fs: 0x0010 $gs: 0x0010
Registers
stack
0x0101af1c|+0x0000: 0x0100038b → 0x5808c483 → 0x5808c483 ← $esp
0x0101af20|+0x0004: 0x00000021 → 0x00000021
0x0101af24|+0x0008: 0x0101af28 → 0x0101b520 → 0x00000000 → 0x00000000
0x0101af28|+0x000c: 0x0101b520 → 0x00000000 → 0x00000000
0x0101af2c|+0x0010: 0x01010870 → 0x8b535657 → 0x8b535657
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x010114dd → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stack
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d xchg ax, ax
0x1002d3f nop
→ 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk.bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8
Assembly
source:./guard/guardian.cc+19
14
15 #include "guard/guard.h"
16 extern Guard guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
→ 19 {
20 (void) vector;
21 (void) context;
22 Gate* gate = plugbox.report(vector);
23
24 bool wantsEpiloque = gate->prologue();
C/C++
threads
[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT
trace
[#0] 0x1002d40 → guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b → irq_entry33()
[#2] 0x1ffb000 → add BYTE PTR [eax+0x31], dl
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [eax], al
Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef➤


[Legend: Modified register | Code | Heap | Stack | String]
registers
$eax : 0x0101b520 → 0x00000000 → 0x00000000
$ebx : 0x01012ba8 → 0x00000000 → 0x00000000
$ecx : 0x01010870 → 0x8b535657 → 0x8b535657
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x01011b7c → 0x01001930 → 0x0191e8b8 → 0x00000000 → 0x00000000
$esi : 0x01012ba8 → 0x00000000 → 0x00000000
$edi : 0x02011200 → 0x02011200
$eip : 0x1002d40 → 0xe8535657 → 0xe8535657
$eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
$cs: 0x0008 $ss: 0x0010 $ds: 0x0010 $es: 0x0010 $fs: 0x0010 $gs: 0x0010
Registers
stack
0x0101af1c|+0x0000: 0x0100038b → 0x5808c483 → 0x5808c483 ← $esp
0x0101af20|+0x0004: 0x00000021 → 0x00000021
0x0101af24|+0x0008: 0x0101af28 → 0x0101b520 → 0x00000000 → 0x00000000
0x0101af28|+0x000c: 0x0101b520 → 0x00000000 → 0x00000000
0x0101af2c|+0x0010: 0x01010870 → 0x8b535657 → 0x8b535657
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x010114dd → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stack
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d xchg ax, ax
0x1002d3f nop
→ 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk.bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8
Assembly
source:./guard/guardian.cc+19
14
15 #include "guard/guard.h"
16 extern Guard guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
→ 19 {
20 (void) vector;
21 (void) context;
22 Gate* gate = plugbox.report(vector);
23
24 bool wantsEpilogue = gate->prologue();
C/C++
threads
[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT
Threads
trace
[#0] 0x1002d40 → guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b → irq_entry33()
[#2] 0x1ffb000 → add BYTE PTR [eax+0x0], al
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [eax], al
Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef➤


[Legend: Modified register | Code | Heap | Stack | String]
registers
$eax : 0x0101b520 → 0x00000000 → 0x00000000
$ebx : 0x01012ba8 → 0x00000000 → 0x00000000
$ecx : 0x01010870 → 0x8b535657 → 0x8b535657
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x01011b7c → 0x01001930 → 0x0191e8b8 → 0x00000000 → 0x00000000
$esi : 0x01012ba8 → 0x00000000 → 0x00000000
$edi : 0x02011200 → 0x02011200
$eip : 0x1002d40 → 0xe8535657 → 0xe8535657
$eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
$cs: 0x0008 $ss: 0x0010 $ds: 0x0010 $es: 0x0010 $fs: 0x0010 $gs: 0x0010
Registers
stack
0x0101af1c|+0x0000: 0x0100038b → 0x5808c483 → 0x5808c483 ← $esp
0x0101af20|+0x0004: 0x00000021 → 0x00000021
0x0101af24|+0x0008: 0x0101af28 → 0x0101b520 → 0x00000000 → 0x00000000
0x0101af28|+0x000c: 0x0101b520 → 0x00000000 → 0x00000000
0x0101af2c|+0x0010: 0x01010870 → 0x8b535657 → 0x8b535657
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x010114dd → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stack
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d xchg ax, ax
0x1002d3f nop
→ 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk.bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8
Assembly
source:./guard/guardian.cc+19
14
15 #include "guard/guard.h"
16 extern Guard guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
→ 19 {
20 (void) vector;
21 (void) context;
22 Gate* gate = plugbox.report(vector);
23
24 bool wantsEpilogue = gate->prologue();
C/C++
threads
[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT
Threads
trace
[#0] 0x1002d40 → guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b → irq_entry_33()
[#2] 0x1ffb000 → add BYTE PTR [eax+0x11], dl
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [eax], al
Backtrace
Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef➤


[Legend: Modified register | Code | Heap | Stack | String]
registers
$eax : 0x0101b520 → 0x00000000 → 0x00000000
$ebx : 0x01012ba8 → 0x00000000 → 0x00000000
$ecx : 0x01010870 → 0x8b535657 → 0x8b535657
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x01011b7c → 0x01001930 → 0x0191e8b8 → 0x00000000 → 0x00000000
$esi : 0x01012ba8 → 0x00000000 → 0x00000000
$edi : 0x02011200 → 0x02011200
$eip : 0x1002d40 → 0xe8535657 → 0xe8535657
$eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
$cs: 0x0008 $ss: 0x0010 $ds: 0x0010 $es: 0x0010 $fs: 0x0010 $gs: 0x0010
Registers
stack
0x0101af1c|+0x0000: 0x0100038b → 0x5808c483 → 0x5808c483 ← $esp
0x0101af20|+0x0004: 0x00000021 → 0x00000021
0x0101af24|+0x0008: 0x0101af28 → 0x0101b520 → 0x00000000 → 0x00000000
0x0101af28|+0x000c: 0x0101b520 → 0x00000000 → 0x00000000
0x0101af2c|+0x0010: 0x01010870 → 0x8b535657 → 0x8b535657
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x010114dd → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stack
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d xchg ax, ax
0x1002d3f nop
→ 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk.bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8
Assembly
source:./guard/guardian.cc+19
14
15 #include "guard/guard.h"
16 extern Guard guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
→ 19 {
20 (void) vector;
21 (void) context;
22 Gate* gate = plugbox.report(vector);
23
24 bool wantsEpilogue = gate->prologue();
C/C++
threads
[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT
Threads
trace
[#0] 0x1002d40 → guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b → irq_entry_33()
[#2] 0x1ffb000 → add BYTE PTR [eax+0x11], dl
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [eax], al
Backtrace
Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef➤
Command Prompt

Breakpoints: (gdb) break <location>

Breakpoints

Interrupt the execution when it arrives at a defined **code location**.

- fuction name
- abolute/relative line of code
- *address

} optional prefix: source file

Interrupts before executing...

```
(gdb) b main
(gdb) b main.cc:main
(gdb) b 63
(gdb) b main.cc:63
(gdb) b +3
(gdb) b *0x100a9ca
```

function `main`
... in main.cc
line 63 in the current file
line 63 in main.cc
in 3 lines
instruction at address 0x100a9ca

Temporary & Conditional Breakpoints

Temporary Breakpoints: `(gdb) tbreak <location>`

Will be removed after they have been hit once, otherwise identical to „normal“ breakpoints.

Temporary & Conditional Breakpoints

Temporary Breakpoints: `(gdb) tbreak <location>`

Will be removed after they have been hit once, otherwise identical to „normal“ breakpoints.

Conditional Breakpoints: `(gdb) break <location> if <cond>`

Only interrupts if condition is fulfilled, e.g:

```
(gdb) break interrupt_handler if vector == 33
```

Only interrupts if the function interrupt_handler has been interrupted by keyboard input (vector 33).

Temporary & Conditional Breakpoints

Temporary Breakpoints: `(gdb) tbreak <location>`

Will be removed after they have been hit once, otherwise identical to „normal“ breakpoints.

Conditional Breakpoints: `(gdb) break <location> if <cond>`

Only interrupts if condition is fulfilled, e.g:

```
(gdb) break interrupt_handler if vector == 33
```

Only interrupts if the function interrupt_handler has been interrupted by keyboard input (vector 33).



Attention: Non-trivial break- or watchpoints are implemented in software $\rightarrow$ slow!

Watchpoints (Data Breakpoints)

Interrupts if a memory location is being written (or read):

watch <location>	write access
rwatch <location>	read access
awatch <location>	read & write access

```
(gdb) watch guard
```

```
(gdb) watch guard if guard.locked == 1
```

Managing Break-/Watchpoints

ignore	<id> <N>	Ignore breakpoint N times
enable	<id> <id> ..	Enable breakpoint
disable	<id> <id> ..	Disable breakpoint
delete	<id> <id> ..	Delete breakpoint

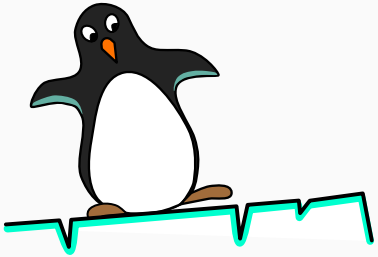


Step-by-Step Exection

step *[count]* – Next line
step**i** *[count]* – Next instruction
next *[count]* – Next line (don't enter functions)
next**i** *[count]* – Next instruction (don't enter functions)
until *[count]* – Repeat **n**ext until reaching the next **souce code line**

Optional: repetition count

finish – Run to the return of the current stack frame
advance <location> – Run to <location>
continue – Execute to the next break-/watchpoint



skip

```
unsigned int numCPUs = System::getNumberOfCPUs();
kout << "numCPUs: " << numCPUs << endl;
ApplicationProcessor::boot();
```

skip

```
unsigned int numCPUs = System::getNumberOfCPUs();
kout << "numCPUs: " << numCPUs << endl;
ApplicationProcessor::boot();
```

Skip the current function:

```
(gdb) skip
```

Skip a specific function:

```
(gdb) skip function OutputStream::operator<<
```

Skip all functions in a file:

```
(gdb) skip file object/outputstream.cc
```

info

info **a**rgs	List all function parameters
info **lo**cals	List all local variables
info **r**egisters	List all registers
info **b**reakpoints	List all breakpoints
info **t**hreads	List all threads/CPUs
info **sk**ip	List all skipped functions
...	see `(gdb) help info`

Additionally:

back**t**race	print the backtrace
frame <id>	Switch to stack frame
thread <id>	Switch to thread/CPU

Emit values from Memory or Registers

```
(gdb) print/<format> <expression>
(gdb) examine/<count><format><unit> <expression>
```

Values for <format>

x	Integer (hex)
d	Integer (signed, decimal)
u	Integer (unsigned, decimal)
t	Integer (binary) (**t**wo)
a	Address (hex) + Offset to the entry symbol
f	Float
i	Instruction

Values for <unit>

b	Byte	8-bit
h	Halfword	16-bit
w	Word	32-bit
g	Giant word	64-bit

Determine the type of a symbol

```
ptype <Symbol>
```

Modfiy the system under test: (gdb) set

Change a register

```
(gdb) set $esp = 0xdeadbeef
```

Change a variable / memory location

```
(gdb) set numCPUs = 2
(gdb) set *((int *) 0x1013fdc) = 42
```

GDB vs. Optimizations

In general: Optimizations make debugging hard:

- Elimination of variables
- Reordering of code
- ...

Relevant Compiler Options

-g	Generate debug information
-Oo	Disable optimizations
-Og	Only optimizaions that don't interfere with debugging

GDB vs. Optimizations

In general: Optimizations make debugging hard:

- Elimination of variables
- Reordering of code
- ...

Relevant Compiler Options

`-g`	Generate debug information
`-O0`	Disable optimizations
`-Og`	Only optimizaions that don't interfere with debugging
`-O2`	Almost all optimizations

.gdbinit

GDB automatically executes commands in `.gdbinit`

- Predefined breakpoints
- Custom scripts

~/.config/gdb/gdbinit or ~/.gdbinit

auto-load local-gdbinit	load local `.gdbinit` file
add-auto-load-safe-path ~/stubs	Enable Path for loading
set disassembly-flavor intel	Use intel syntax instead of AT&T syntax
set print asm-demangle on	Demangle names

Example: project local `.gdbinit`

```
b assertion_failed
b Core::die
```

GDB & GEF in BS

The bitter truth...

You will have to debug



GDB & GEF in BS

The bitter truth...

You will have to debug



- `make qemu-gdb-noopt`

→ profit?

- `make qemu-gdb`
- `make VERBOSE= QEMUCPUS=1 all`
- `make qemu-wait` + `make gdb` or external debugger

Debug with GEF

`make <debug target> GDB=gef`

Questions



https://sourceware.org/gdb/current/onlinedocs/gdb.pdf
https://www.qemu.org/docs/master/system/monitor.html