Übung zu Betriebssysteme

Entkäfern mit GDB & GEF

KW 48 / 2020

Bernhard Heinloth & Christian Eichler

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

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.

search for the error online: HAL_INITIALIZATION_FAILED

:(

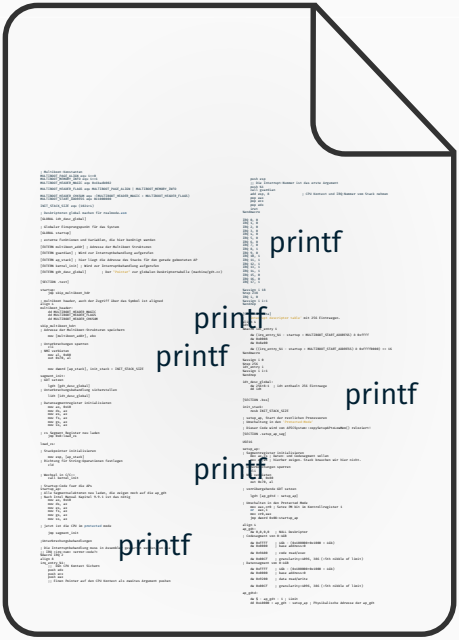
Your PC ran into a problem that it couldn't
ndle, and now it needs to restart.

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
        page
        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

Entkäfern mit GDB & GEF

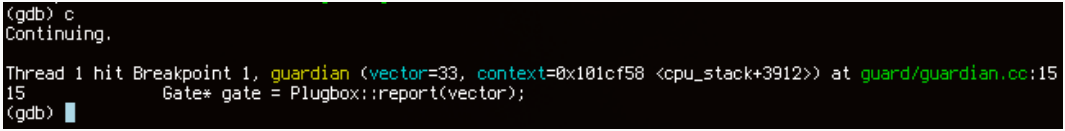
GNU Debugger (GDB)

- \+ Inspizieren des Systemzustands während das System läuft
- \- Nur rudimentäres TUI

Entkäfern mit GDB & GEF

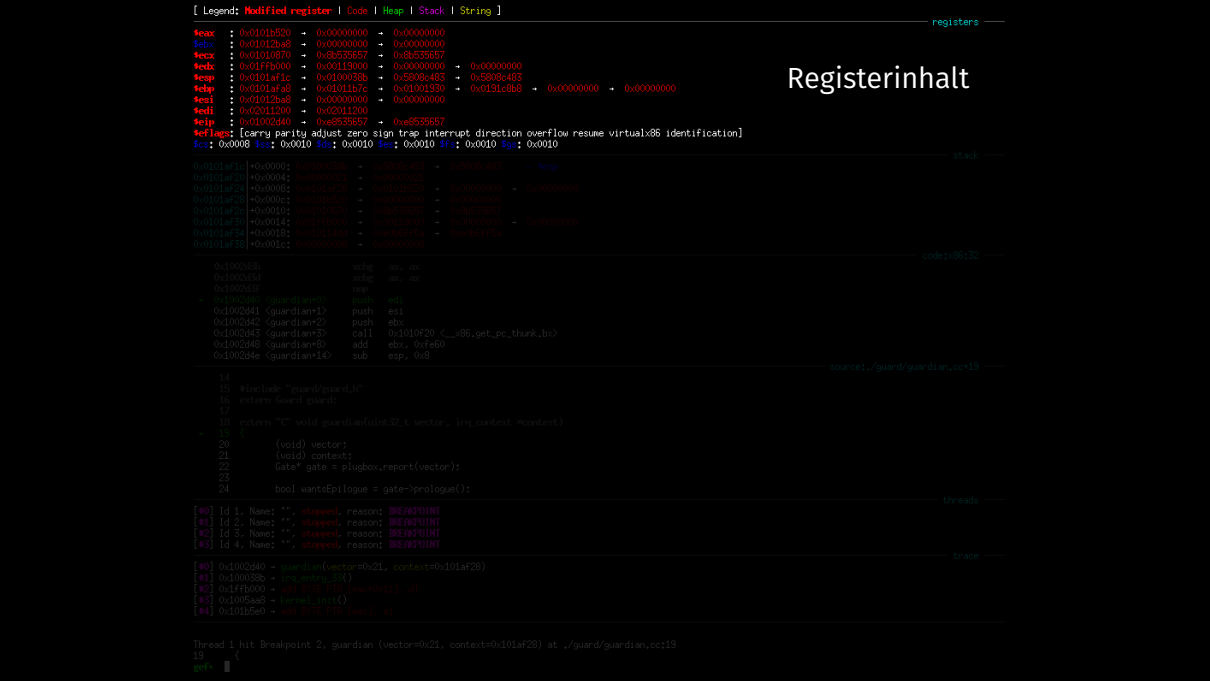
GNU Debugger (GDB)

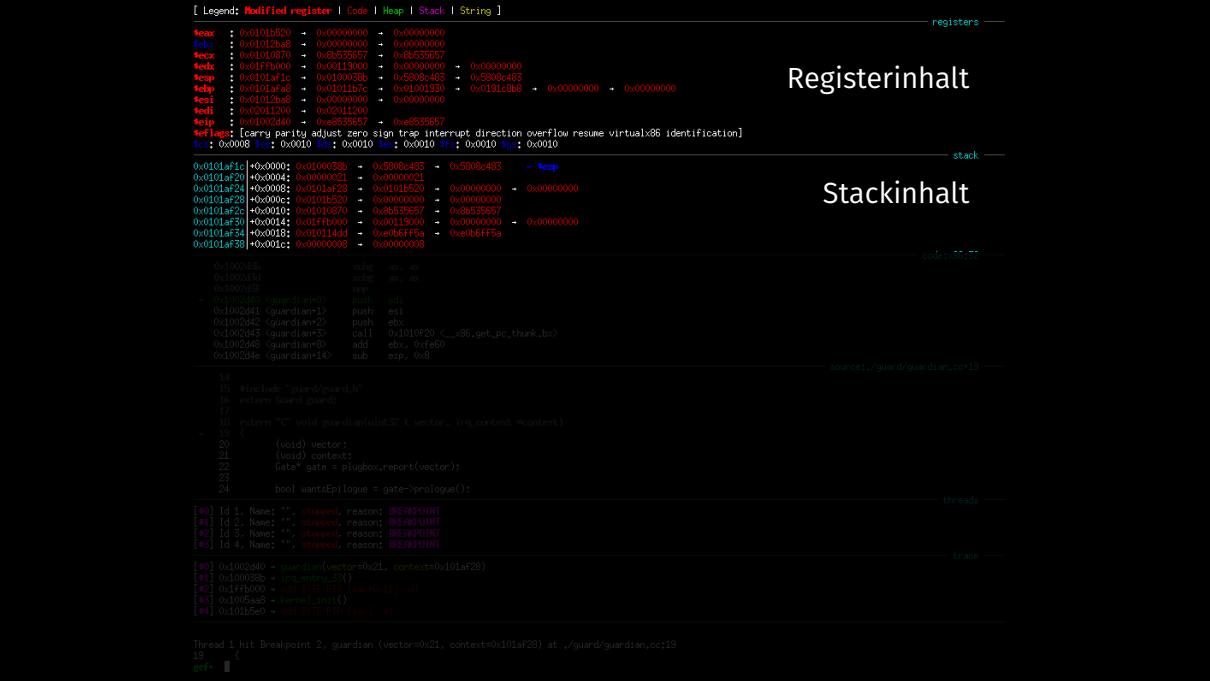
+ Inspizieren des Systemzustands während das System läuft
- Nur rudimentäres TUI

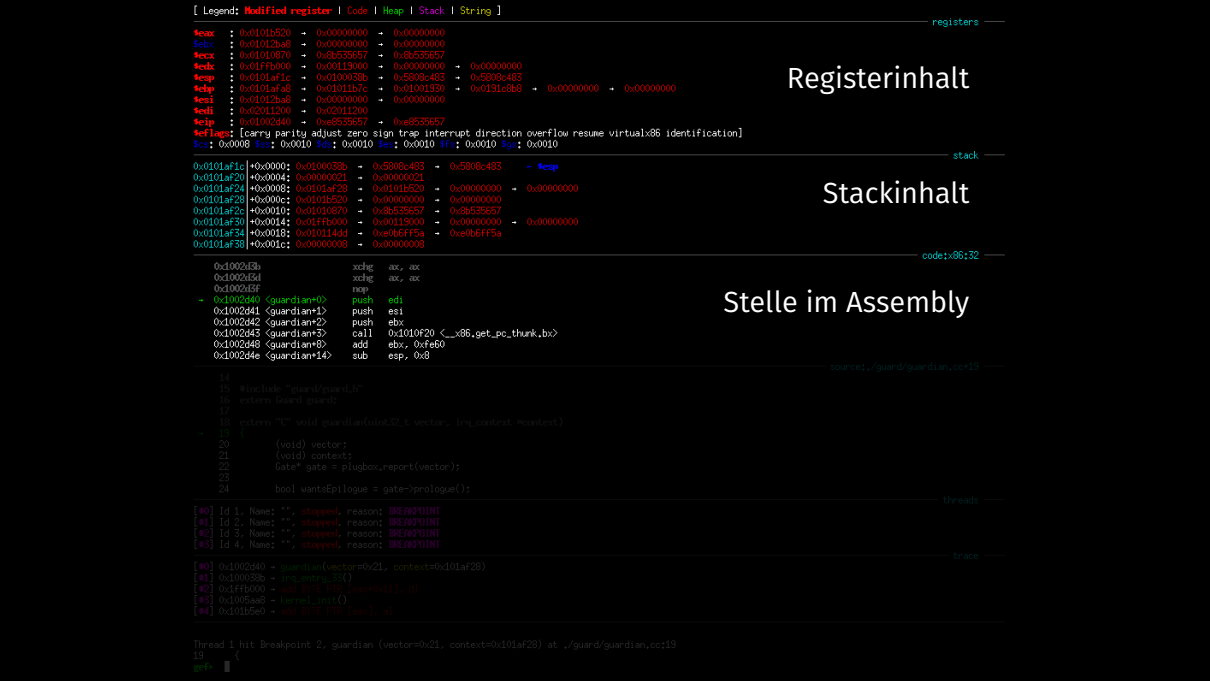


GDB Enhanced Features (GEF)

+ Erweitert GDB um ein brauchbar(er)es Interface


[Legend: Modified register | Code | Heap | Stack | String]
registers
$eax : 0x0101b520 → 0x00000000 → 0x00000000
$ebx : 0x01012ba8 → 0x00000000 → 0x00000000
$ecx : 0x01010870 → 0x8b539b57 → 0x8b539b57
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101afa8 → 0x01011b7c → 0x01001930 → 0x0191e8b8 → 0x00000000 → 0x00000000
$esi : 0x01012ba8 → 0x00000000 → 0x00000000
$edi : 0x02011200 → 0x02011200
$eip : 0x01002d40 → 0xe8539b57 → 0xe8539b57
$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
Registerinhalt
stack
0x0101af1c|+0x0000: 0x0100038b → 0x5808c483 → 0x5808c483 ← $esp
0x0101af20|+0x0004: 0x00000021
0x0101af24|+0x0008: 0x0101afc8 → 0x00000000 → 0x00000000
0x0101af28|+0x000c: 0x01010870 → 0x8b539b57
0x0101af2c|+0x0010: 0x01012ba8 → 0x00000000 → 0x00000000
0x0101af30|+0x0014: 0x01011200 → 0x02011200
0x0101af34|+0x0018: 0x0101afa8 → 0x01011b7c → 0x01001930 → 0x0191e8b8
0x0101af38|+0x001c: 0x00000000 → 0x00000000
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d nop 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 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();
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 [rax], al
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [rax], 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 → 0x8b535857 → 0x8b535857
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101afa8 → 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
Registerinhalt
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 → 0x8b535857 → 0x8b535857
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x01011dd0 → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stackinhalt
code:x86:32
0x1002d3b xchg ax, ax
0x1002d3d nop 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+15
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();
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 [rax], al
[#3] 0x1005aa8 → kernel_init()
[#4] 0x101b5e0 → add BYTE PTR [rax], 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 → 0x0101b7c0 → 0x01001930 → 0x191e8b8 → 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
Registerinhalt
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: 0x01011dd0 → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stackinhalt
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
Stelle im 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();
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], 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 → 0x8b535857 → 0x8b535857
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x0101b7c → 0x01001930 → 0x191c8b8 → 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
Registerinhalt
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 → 0x8b535857 → 0x8b535857
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x01011dd0 → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stackinhalt
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
Stelle im 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();
Stelle in 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_entry35()
[#2] 0x1ffb000 → add BYTE PTR [eax], 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 → 0x8b535857 → 0x8b535857
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x0101b7c → 0x01001930 → 0x191e8b8 → 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
Registerinhalt
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 → 0x8b535857 → 0x8b535857
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x01011dd0 → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stackinhalt
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
Stelle im 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();
Stelle in 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_entry35()
[#2] 0x1ffb000 → add BYTE PTR [eax], 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 → 0x8b535857 → 0x8b535857
$edx : 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x0101b7c → 0x01001930 → 0x191e8b8 → 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
Registerinhalt
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 → 0x8b535857 → 0x8b535857
0x0101af30|+0x0014: 0x01ffb000 → 0x00119000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x01011dd0 → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stackinhalt
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
Stelle im 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();
Stelle in 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 → 0x8b535857 → 0x8b535857
$edx : 0x01ffb000 → 0x00113000 → 0x00000000 → 0x00000000
$esp : 0x0101af1c → 0x0100038b → 0x5808c483 → 0x5808c483
$ebp : 0x0101af28 → 0x01011b7c → 0x01001930 → 0x191e8b8 → 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
Registerinhalt
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 → 0x8b535857 → 0x8b535857
0x0101af30|+0x0014: 0x01ffb000 → 0x00113000 → 0x00000000 → 0x00000000
0x0101af34|+0x0018: 0x01011ddd → 0xe0b6ff5a → 0xe0b6ff5a
0x0101af38|+0x001c: 0x00000008 → 0x00000008
Stackinhalt
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
Stelle im 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();
Stelle in 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➤
Eingabezeile

Breakpoints: (gdb) break <location>

Breakpoints

Unterbrechen der Ausführung, sobald eine bestimmte **Codestelle** erreicht wird.

- Funktionsname
- absolute/relative Codezeile
- *Adresse

} optionaler Prefix: Quelldatei

Unterbricht vor dem Ausführen von...

`(gdb) b main`	Funktion `main`
`(gdb) b main.cc:main`	... aus main.cc
`(gdb) b 63`	Zeile 63 in aktueller Datei
`(gdb) b main.cc:63`	Zeile 63 in main.cc
`(gdb) b +3`	In 3 Zeilen
`(gdb) b *0x100a9ca`	An Adresse 0x100a9ca

Temporäre & Bedingte Breakpoints

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

Werden nach dem 1. Auslösen entfernt, sonst wie „normale“ Breakpoints.

Temporäre & Bedingte Breakpoints

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

Werden nach dem 1. Auslösen entfernt, sonst wie „normale“ Breakpoints.

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

Unterbrechung nur falls Bedingung erfüllt ist, z.B:

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

Unterbricht nur, falls die Funktion interrupt_handler aufgrund von Tastatureingabe (Vektor 33) betreten wurde.

Temporäre & Bedingte Breakpoints

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

Werden nach dem 1. Auslösen entfernt, sonst wie „normale“ Breakpoints.

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

Unterbrechung nur falls Bedingung erfüllt ist, z.B:

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

Unterbricht nur, falls die Funktion interrupt_handler aufgrund von Tastatureingabe (Vektor 33) betreten wurde.

Achtung: Nichttriviale Break- oder Watchpoints werden ohne Hardwareunterstützung umgesetzt → Langsam!

Watchpoints (Data Breakpoints)

Unterbricht wenn Speicherbereich geschrieben (oder gelesen) wird:

watch <location>	Schreibzugriff
rwatch <location>	Lesezugriff
awatch <location>	Schreib- oder Lesezugriff

```
(gdb) watch guard
```

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

Verwalten von Break-/Watchpoints

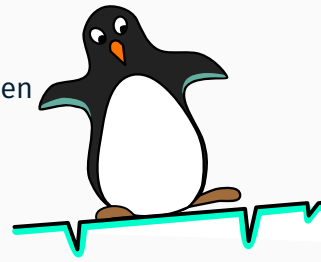
ignore	<id> <N>	Breakpoint N mal ignorieren
enable	<id> <id> ..	Breakpoints aktivieren
disable	<id> <id> ..	Breakpoints deaktivieren
delete	<id> <id> ..	Breakpoints löschen

Schrittweise Ausführung

step *count* – Nächste Zeile
step**i** *count* – Nächste Instruktion
next *count* – Nächste Zeile (ohne Funktionen zu betreten)
next**i** *count* – Nächste Instruktion (ohne Funktionen zu betreten)
until *count* – Wiederhole **n**ext bis zur **textuell** nächsten Zeile

Optional: Anzahl Wiederholungen

finish – Bis zum return des aktuellen Stackframes
advance <location> – Bis zu <location>
continue – Ausführung (bis zum nächsten Breakpoint) fortsetzen



Der skip Befehl

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

Der skip Befehl

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

Übergehe eine einzelne Funktion:

```
(gdb) skip Guard::enter
```

Übergehe alle Funktionen aus einer Datei:

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

Der `info` Befehl

info **lo**cals	Auflistung aller lokalen Variablen
info **r**egisters	Auflistung der Registerwerte
info **b**reakpoints	Auflistung der Breakpoints
info **t**hreads	Auflistung der Threads
info **sk**ip	Auflistung der übersprungenen Funktionen
...	siehe `(gdb) info`

Ausgabe von Werten in Speicher / Register

```
(gdb) print/<Format> <Ausdruck>
(gdb) x/<Anzahl><Format> <Ausdruck>
```

Werte für <Format>

x	Ganzzahl (hex)	**a**	Adresse (hex) + Offset zum Startsymbol
d	Ganzzahl (mit VZ, dezimal)	**f**	Float
u	Ganzzahl (ohne VZ, dezimal)	**i**	Instruktion
t	Ganzzahl (binär) (**t**wo)		

Bestimmen des Typs eines Symbols

ptype <Symbol>

Verändern des Zielsystems: (gdb) set

Verändern eines Registers

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

Verändern einer Variable / Speicherbereichs

```
(gdb) set numCPUs = 2
```

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

GDB vs. Optimierungen

Generell: Optimierungen sind doof fürs Debuggen:

- Inlining von Funktionen
- Elimination von Variablen
- ...

Relevante Compileroptionen

-g	Generiere Debuginformationen
-Oo	Optimierungen aus
-Og	Nur Optimierungen, die das Debuggen nicht stören

GDB vs. Optimierungen

Generell: Optimierungen sind doof fürs Debuggen:

- Inlining von Funktionen
- Elimination von Variablen
- ...

Relevante Compileroptionen

-g	Generiere Debuginformationen
-O0	Optimierungen aus
-Og	Nur Optimierungen, die das Debuggen nicht stören
-O2	Fast alle Optimierungen

GDB & GEF in BS

Die bittere Wahrheit...

Ihr werdet entkäfern müssen

GDB & GEF in BS

Die bittere Wahrheit...

Ihr werdet entkäfern müssen

- Anlegen einer eigenen .gdbinit:
  `cp /proj/i4stubs/tools/gdbinit ~/.gdbinit`
- `make qemu-gdb-noopt`

→ Profit?

- `make qemu-gdb`

Fragen?