



RedHat Linux Kernel Internals Laboratory Exercises

Lab 17a: Debugging

Objective: In this lab you will debug a kernel module using gdb post mortem (not live since this would require kgdb and multiple machines).

1. Reboot the computer and choose the debug kernel.

```
Red Hat Enterprise Linux Server (3.10.0-123.6.3.el7.x86_64.debug) 7.0 (M→
Red Hat Enterprise Linux Server, with Linux 3.10.0-123.el7.x86_64
Red Hat Enterprise Linux Server, with Linux 0-rescue-0537ebc56bf2400e823→

Use the ↑ and ↓ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
```

2. Open a terminal and verify that the kdump service is operational using the command “sudo service kdump status”.
3. Navigate to the “~/RHI/labs/Lab17” folder and view the debugme.c file.

```

/* ANRC RHKI */
/* Lab17: Debugging Lab */
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

#define DRIVER_AUTHOR "ANRC"
#define DRIVER_DESC  "Lab17"

MODULE_LICENSE("GPL");      // Get rid of taint message by declaring code as GPL.

/* Or with defines, like this: */
MODULE_AUTHOR(DRIVER_AUTHOR); // Who wrote this module?
MODULE_DESCRIPTION(DRIVER_DESC); // What does this module do?

int anrc_1;
int anrc_2 = 20;
int anrc_3 = 30;

EXPORT_SYMBOL(anrc_3);

int init(void)
{
    printk(KERN_INFO "init_module() called\n");

    panic("something bad happened!");
    dump_stack();

    return 0;
}

void cleanup(void)
{
    printk(KERN_ALERT "Unloading debugme ...\n");
}

module_init(init);
module_exit(cleanup);

```

4. The program debugme simply causes a kernel panic during LKM initialization.
5. Load the module via “sudo /sbin/insmod debugme.ko”. Observe what happens. (Make sure you do not interfere with the system until it boots for the second time and choose the debug kernel again).
6. When the system resumes open a terminal and view the crash dump file using the “crash” command.

```

user@localhost lab17a1$ sudo crash /usr/lib/debug/lib/modules/3.10.0-123.6.3.el7.x86_64/debug/vmlinux \
/var/crash/127.0.0.1-2014.09.11-23:49:44/vmcore

```

7. The crash program gives us a wealth of information about what happened.

```

crash 7.0.2-6.el7
Copyright (C) 2002-2013 Red Hat, Inc.
Copyright (C) 2004, 2005, 2006, 2010 IBM Corporation
Copyright (C) 1999-2006 Hewlett-Packard Co
Copyright (C) 2005, 2006, 2011, 2012 Fujitsu Limited
Copyright (C) 2006, 2007 VA Linux Systems Japan K.K.
Copyright (C) 2005, 2011 NEC Corporation
Copyright (C) 1999, 2002, 2007 Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002 Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions. Enter "help copying" to see the conditions.
This program has absolutely no warranty. Enter "help warranty" for details.

GNU gdb (GDB) 7.6
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu"...

        KERNEL: /usr/lib/debug/lib/modules/3.10.0-123.6.3.el7.x86_64.debug/vmlinux
        DUMPFILE: /var/crash/127.0.0.1-2014.09.11-23:49:44/vmcore [PARTIAL DUMP]
        CPUS: 1
        DATE: Thu Sep 11 23:49:39 2014
        UPTIME: 00:23:18
LOAD AVERAGE: 0.56, 0.76, 0.78
        TASKS: 513
        NODENAME: localhost.localdomain
        RELEASE: 3.10.0-123.6.3.el7.x86_64.debug
        VERSION: #1 SMP Wed Jul 16 15:18:28 EDT 2014
        MACHINE: x86_64 (3073 Mhz)
        MEMORY: 2 GB
        PANIC: "kernel panic - not syncing: something bad happened!"
        PID: 25740
        COMMAND: "insmod"
        TASK: ffff88004a832790 [THREAD_INFO: ffff880007d5a000]
        CPU: 0
        STATE: TASK_RUNNING (PANIC)

crash>

```

8. Where is the result of our panic() call?
9. Finally you can issue normal gdb commands to get to the bottom of what may have gone wrong. Example "bt" or backtrace.

```

crash> bt
PID: 25740 TASK: ffff88004a832790 CPU: 0 COMMAND: "insmod"
#0 [ffff880007d5bbb0] machine_kexec at ffffffff8104c2b6
#1 [ffff880007d5bc08] crash_kexec at ffffffff811001d2
#2 [ffff880007d5bcd8] panic at ffffffff81697fa0
#3 [ffff880007d5bd58] init at ffffffff8a0662025 [debugme]
#4 [ffff880007d5bd68] do_one_initcall at ffffffff810020e2
#5 [ffff880007d5bd98] load_module at ffffffff810fb382
#6 [ffff880007d5bee8] sys_finit_module at ffffffff810fbc86
#7 [ffff880007d5bf80] system_call_fastpath at ffffffff816b3859
RIP: 00007fcee0e0fbb9 RSP: 00007fff02e00ad0 RFLAGS: 00010206
RAX: 0000000000000139 RBX: ffffffff816b3859 RCX: 0000000000000000
RDX: 0000000000000000 RSI: 00007fcee10f0d89 RDI: 0000000000000003
RBP: 0000000000000000 R8: 0000000000000000 R9: 00000000010a0010
R10: 0000000000000003 R11: 0000000000000202 R12: 00000000010a12f0
R13: 00000000010a13c0 R14: 00000000010a0090 R15: 00007fcee10f0d89
ORIG_RAX: 0000000000000139 CS: 0033 SS: 002b

```

10. If you are familiar with gdb, disassemble (dis) the init function and locate the call to panic.

```
crash> dis init
0xfffffffffa0662000 <init_module>:      data32 data32 data32 xchg %ax,%ax
0xfffffffffa0662005 <init+5>:      push    %rbp
0xfffffffffa0662006 <init+6>:      mov     $0xfffffffffa0663048,%rdi
0xfffffffffa066200d <init+13>:     xor     %eax,%eax
0xfffffffffa066200f <init+15>:     mov     %rsp,%rbp
0xfffffffffa0662012 <init+18>:     callq  0xfffffffff8169812f <printf@plt>
0xfffffffffa0662017 <init+23>:     mov     $0xfffffffffa0663060,%rdi
0xfffffffffa066201e <init+30>:     xor     %eax,%eax
0xfffffffffa0662020 <init+32>:     callq  0xfffffffff81697ebc <panic@plt>
0xfffffffffa0662025 <init+37>:     data32 nopw %cs:0x0(%rax,%rax,1)
crash>
```

11. Other helpful commands include: **set** (provides information on current process context, can be changed to other process, ex. *set 1*, change to init), **dmesg** (system log, page down to see any important information), **ps** (view process information, note insmod was last active process), **task** (dump active task_struct, likely the culprit which brought down the system, this works because the crash utility operates in the *current context*), **kmem -i** (was crash memory consumption related?), **files** (provide information on active file handles), **sig** (signal information), **net** (network device info), **vm** (memory map status of active task), use **struct** (to dereference memory as structures, ex. struct task_struct 0xdeadC0de)

```
crash> set
PID: 25740
COMMAND: "insmod"
TASK: ffff88004a832790 [THREAD_INFO: ffff880007d5a000]
CPU: 0
STATE: TASK_RUNNING (PANIC)
crash> struct task_struct 0xffff88004a832790
struct task_struct {
  state = 0,
  stack = 0xffff880007d5a000,
  usage = {
    counter = 2
  },
  flags = 4202752,
  ptrace = 0,
  wake_entry = {
    next = 0x0
  },
  on_cpu = 1,
  last_wakee = 0xffff88006e650000,
  wakee_flips = 12,
  wakee_flip_decay_ts = 4296065489,
  wake_cpu = 0,
  on_rq = 1,
  prio = 120,
  static_prio = 120,
}
```

12. Type quit to exit crash. Continue to Lab 17b.