How to Read an Oops Message and Some Debugging Tips

Eddie Lai
Cisco Systems

This is an oops that had me trying to debug for hours. The issue I was having was that I was printing out pointer addresses within a struct. I had a struct with several fields and I was printing out their pointer addresses to make sure they were not null. The oops message presented is the oops from printing out an address within a null struct. A null struct's address is 0x0000, and then trying to access a structure address within it will just offset the null pointer by some fixed amount. I did not understand why this was happening until I basically started over. So, here is a tutorial on how to read an oops message and how to figure out what is causing it.

```
BUG: unable to handle kernel NULL pointer dereference at 00000374
IP: [<c01478d2>] sys_mailbox_send+0x1c/0x2f1
*pdpt = 0000000030a54001 *pde = 0000000000000000
Oops: 0000 [#1] SMP
last sysfs file: /sys/devices/LNXSYSTM:00/device:00/ACPI0003:00/power_supply/ACAD/online
Modules linked in: ip6t_LOG xt_tcpudp xt_pkttype ipt_LOG xt_limit fuse af_packet edd ip6t_REJECT nf_conntrack_ipv6 nf_defrag_ipv6 ip6table_raw xt_NOTRACK ipt_REJECT iptable_raw ipt
able_filter ip6table_mangle nf_conntrack_netbios_ns nf_conntrack_ipv4 nf_defrag_ipv4 ip_tables xt_conntrack nf_conntrack ip6table_filter ip6_tables x_tables ipv6 mperf dm_mod pcnet
32 sr_mod floppy i2c_piix4 ac button intel_agp intel_gtt agpgart power_supply container cdrom mii sg mptctl uhci_hcd ehci_hcd usbcore sd_mod crc_t10dif fan processor thermal therma
l_sys hwmon ata_generic ata_piix libata mptspi mptscsih mptbase scsi_transport_spi vmw_pvscsi scsi_mod [last unloaded: speedstep_lib]

Pid: 5271, comm: testmailbox1 Not tainted 2.6.37.6-CS502_Cisco #4 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform
EIP: 0060:[<c01478d2>] EFLAGS: 00210292 CPU: 1
EIP is at sys_mailbox_send+0x1c/0x2f1
EAX: 00000100 EBX: 00001498 ECX: bfd5fc9f EDX: 000000d8
ESI: 000003ed EDI: 00000011 EBP: 00001498 ESP: f0815f8c
 DS: 007b ES: 007b FS: 00d8 GS: 0033 SS: 0068
Process testmailbox1 (pid: 5271, ti=f0814000 task=f06c4db0 task.ti=f0814000)
Stack:
 b7719000 f0814000 c01afe47 f0815f9c 00000000 00001498 00000000 b76d24c0
 f0814000 c0102810 00001498 bfd5fc9f 00000011 00000000 b76d24c0 b75c8e36
 00000155 0000007b 0000007b 00000000 00000000 00000155 ffffe430 00000073
Call Trace:
 [<c0102810>] sysenter_do_call+0x12/0x22
 [<ffffe430>] 0xffffe430
Code: 49 c0 89 44 24 04 e8 8c 6c 27 00 83 c4 0c c3 55 57 56 be ed 03 00 00 53 83 ec 14 8a 44 24 34 8b 7c 24 30 8b 6c 24 28 88 44 24 13 <a1> 74 03 00 00 c7 04 24 41 95 49 c0 89 44 2
4 04 e8 57 6c 27 00
EIP: [<c01478d2>] sys_mailbox_send+0x1c/0x2f1 SS:ESP 0068:f0815f8c
CR2: 0000000000000374
---[ end trace f9cfecbe5f272a92 ]---
```

Figure 1 – A screenshot of the oops message from the terminal

I have also copied and pasted it here for reference.

BUG: unable to handle kernel NULL pointer dereference at 00000374
IP: [<c01478d2>] sys_mailbox_send+0x1c/0x2f1
*pdpt = 000000002c91e001 *pde = 0000000000000000
Oops: 0000 [#3] SMP
last sysfs file: /sys/devices/pci0000:00/0000:00:18.7/modalias
Modules linked in: ip6t_LOG xt_tcpudp xt_pkttype ipt_LOG xt_limit af_packet edd ip6t_REJECT nf_conntrack_ipv6 nf_defrag_ipv6 ip6table_raw xt_NOTRACK ipt_REJECT iptable_raw iptable_filter ip6table_mangle nf_conntrack_netbios_ns nf_conntrack_ipv4 nf_defrag_ipv4 ip_tables xt_conntrack nf_conntrack ip6table_filter ip6_tables x_tables ipv6 mperf dm_mod floppy intel_agp intel_gtt agpgart ac i2c_piix4 power_supply sg sr_mod pcnet32 container button mii cdrom mptctl uhci_hcd ehci_hcd sd_mod usbcore crc_t10dif fan processor thermal thermal_sys hwmon ata_generic ata_piix libata mptspi mptscsih mptbase scsi_transport_spi vmw_pvscsi scsi_mod [last unloaded: speedstep_lib]

Pid: 5112, comm: testmailbox1 Tainted: G D 2.6.37.6-CS502_Cisco #4 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform
EIP: 0060:[<c01478d2>] EFLAGS: 00210292 CPU: 0
EIP is at sys_mailbox_send+0x1c/0x2f1
EAX: 00000100 EBX: 000013f9 ECX: bfbea81f EDX: 000000d8
ESI: 000003ed EDI: 00000011 EBP: 000013f9 ESP: ec8e3f8c
 DS: 007b ES: 007b FS: 00d8 GS: 0033 SS: 0068
Process testmailbox1 (pid: 5112, ti=ec8e2000 task=ec9e9130 task.ti=ec8e2000)
Stack:
 b7861000 ec8e2000 c01afe47 ec8e3f9c 00000000 000013f9 00000000 b781a4c0
 ec8e2000 c0102810 000013f9 bfbea81f 00000011 00000000 b781a4c0 b7710e36

```
 00000155 0000007b 0000007b 00000000 00000000 00000155 ffffe430 00000073
Call Trace:
 [<c0102810>] sysenter_do_call+0x12/0x22
 [<ffffe430>] 0xffffe430
Code: 49 c0 89 44 24 04 e8 8c 6c 27 00 83 c4 0c c3 55 57 56 be ed 03 00 00 53 83 ec 14 8a 44 24 34 8b 7c 24 30 8b 6c 24 28 88 44 24 13 <a1> 74 03
00 00 c7 04 24 41 95 49 c0 89 44 24 04 e8 57 6c 27 00
EIP: [<c01478d2>] sys_mailbox_send+0x1c/0x2f1 SS:ESP 0068:ec8e3f8c
CR2: 0000000000000374
---[ end trace dfbaf6dadda3703f ]---
```

Oops: 0000 – This is the counter that shows you what level error it is. Usually an oops can cause multiple oops, but in this case, there is only one and this is it. If there are multiple oops, you can only trust the first one.
BUG: unable to handle kernel NULL pointer dereference at 00000374 – This error occurs because you are trying to read from a null pointer. The low address means that you are trying to access a structure member from a null struct.
IP: [<c01478d2>] sys_mailbox_send+0x1c/0x2f1 – This gives the instruction pointer and the name of the function. Sometimes the function name is not presented and you have to find the function name. I will explain how to find the function name later.

Finding the function name

From the oops message:

IP: [<c01478d2>] sys_mailbox_send+0x1c/0x2f1

If the function name is not presented, then it will give you an address (in this case c01478d2) and an offset (in this case 0x1c). If you subtract the offset from the instruction pointer (C01478d2 – 1c), you will get C01478B6. This is the base address of the function. Navigate to /boot/ on your machine and open up the System.map-xxx of your kernel (mine is System.map-2.6.37.6-Project4). Open this with your favorite text editor and search for the address that you got from the subtraction. As figure 2 shows, this is the function name.

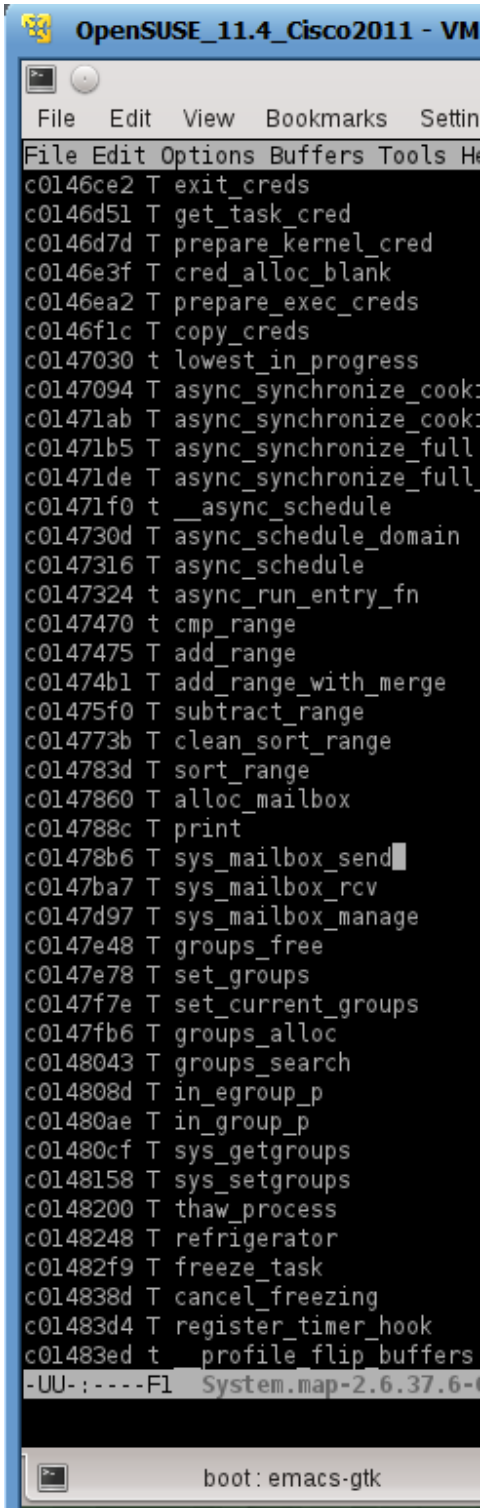

Figure 2 – The function name matched up to the address after subtracting the offset from the instruction pointer

Tracing your oops and viewing the assembly code

There is a very useful program already built into your kernel called ksymoops. To load this program, simply type in ksymoops into your terminal. It should then look something like this.

```
student@linux-zudt:/proc> ksymoops
ksymoops 2.4.11 on i686 2.6.37.6-CS502_Cisco.  Options used
     -V (default)
     -k /proc/kallsyms (default)
     -l /proc/modules (default)
     -o /lib/modules/2.6.37.6-CS502_Cisco/ (default)
     -m /boot/System.map-2.6.37.6-CS502_Cisco (default)

Warning: You did not tell me where to find symbol information.  I will
assume that the log matches the kernel and modules that are running
right now and I'll use the default options above for symbol resolution.
If the current kernel and/or modules do not match the log, you can get
more accurate output by telling me the kernel version and where to find
map, modules, ksyms etc.  ksymoops -h explains the options.

Warning (read_ksyms): no kernel symbols in ksyms, is /proc/kallsyms a valid ksyms file?
No modules in ksyms, skipping objects
No ksyms, skipping lsmod
Reading Oops report from the terminal
```

Figure 3 – The terminal output after starting the ksymoops program

What you want to do now is to copy the entire oops message from running "dmesg" and paste it here. Once pasted, you will have something like Figure 4.

```
>>EIP; c01478d2 <sys_mailbox_send+1c/2f1>   <=====

>>ECX; bfd5fc9f <phys_startup_32+bfc5fc9f/c0000000>

Trace; c0102810 <sysenter_do_call+12/22>
Trace; ffffe430 <END_OF_CODE+3f58e430/????>

Code;  c01478a7 <print+1b/2a>
00000000 <_EIP>:
Code;  c01478a7 <print+1b/2a>
   0:   49                      dec    %ecx
Code;  c01478a8 <print+1c/2a>
   1:   c0 89 44 24 04 e8 8c    rorb   $0x8c,-0x17fbdbbc(%ecx)
Code;  c01478af <print+23/2a>
   8:   6c                      insb   (%dx),%es:(%edi)
Code;  c01478b0 <print+24/2a>
   9:   27                      daa
Code;  c01478b1 <print+25/2a>
   a:   00 83 c4 0c c3 55       add    %al,0x55c30cc4(%ebx)
Code;  c01478b7 <sys_mailbox_send+1/2f1>
  10:   57                      push   %edi
Code;  c01478b8 <sys_mailbox_send+2/2f1>
  11:   56                      push   %esi
Code;  c01478b9 <sys_mailbox_send+3/2f1>
  12:   be ed 03 00 00          mov    $0x3ed,%esi
Code;  c01478be <sys_mailbox_send+8/2f1>
  17:   53                      push   %ebx
Code;  c01478bf <sys_mailbox_send+9/2f1>
  18:   83 ec 14                sub    $0x14,%esp
Code;  c01478c2 <sys_mailbox_send+c/2f1>
  1b:   8a 44 24 34             mov    0x34(%esp),%al
Code;  c01478c6 <sys_mailbox_send+10/2f1>
  1f:   8b 7c 24 30             mov    0x30(%esp),%edi
Code;  c01478ca <sys_mailbox_send+14/2f1>
  23:   8b 6c 24 28             mov    0x28(%esp),%ebp
Code;  c01478ce <sys_mailbox_send+18/2f1>
  27:   88 44 24 13             mov    %al,0x13(%esp)
Code;  c01478d2 <sys_mailbox_send+1c/2f1>   <=====
  2b:   a1 74 03 00 00          mov    0x374,%eax   <=====
Code;  c01478d7 <sys_mailbox_send+21/2f1>
  30:   c7 04 24 41 95 49 c0    movl   $0xc0499541,(%esp)
Code;  c01478de <sys_mailbox_send+28/2f1>
  37:   89 44 24 04             mov    %eax,0x4(%esp)
Code;  c01478e2 <sys_mailbox_send+2c/2f1>
  3b:   e8 57 6c 27 00          call   276c97 <_EIP+0x276c97>

EIP: [<c01478d2>] sys_mailbox_send+0x1c/0x2f1 SS:ESP 0068:f0815f8c
CR2: 0000000000000374
```

Figure 4 – The output of ksymoops after pasting the oops message into it

This may or may not be helpful to you. Being able to see the assembly code can help in learning how the compiled code works, or you may see the issue right away.

The first two lines give you a trace of the function calls before it oops'ed. The instruction pointer (EIP) was in sys_mailbox_send and then called sysenter_do_call, which then caused the issue and gave an END_OF_CODE. Typically in larger code bases, the trace stack has a lot more call backs.

Finding the exact line of code that caused the kernel oops

Step 1:

- Locate the file that has the function that is causing your oops. You want to navigate to the destination folder that has the .o file in it (in my case eFinalDest/kernel/mailbox.o)

Step 2:

- Generate the following file: objdump -d mailbox.o > mailbox.disassem (replace mailbox.o with the object file corresponding to the source file causing the oops)

Step 3:

- Navigate to the very top level of your kernel destination (like ~/kernelDst). You now want to generate the assembly code of the source file that you are debugging. You will run a make command on the same file as step 1 but with a .s extension. In my case, this is: make kernel/mailbox.s. **NOTE: You must be at the top level of the kernel destination folder**

```
student@linux-zudt:~/WPI/proj4/eFinalDest> pwd
/home/student/WPI/proj4/eFinalDest
student@linux-zudt:~/WPI/proj4/eFinalDest> make kernel/mailbox.s
make -C /home/student/WPI/proj4/finalSrc O=/home/student/WPI/proj4/eFinalDest/. kernel/mailbox.s
  Using /home/student/WPI/proj4/finalSrc as source for kernel
  GEN     /home/student/WPI/proj4/eFinalDest/Makefile
  CHK     include/linux/version.h
  CHK     include/generated/utsrelease.h
  CALL    /home/student/WPI/proj4/finalSrc/scripts/checksyscalls.sh
  CC      kernel/mailbox.s
/home/student/WPI/proj4/finalSrc/kernel/mailbox.c: In function 'sys_mailbox_send':
/home/student/WPI/proj4/finalSrc/kernel/mailbox.c:24:3: warning: ISO C90 forbids mixed declarations and code
student@linux-zudt:~/WPI/proj4/eFinalDest>
```

Step 4: Look through the .disassem file and find the starting address of the function that caused the oops.

```
00000056 <sys_mailbox_send>:
  56:   55                      push   %ebp
  57:   57                      push   %edi
  58:   56                      push   %esi
  59:   be ed 03 00 00          mov    $0x3ed,%esi
```

Step 5:

- From the oops message: IP: [<c01478d2>] sys_mailbox_send+0x1c/0x2f1
- 0x1c is your offset so now you must add 0x1c to the starting address of the function (in my case 0x56). You can either input this to a hexadecimal calculator or use the shell by typing in: printf "%x\n" $((0x56+0x1c))

```
student@linux-zudt:~/WPI/proj4/eFinalDest/kernel> printf "%x\n" $((0x56+0x1c))
72
```

Step 6:

- Look for the address from step 4 inside the .disassem file that you saved.

```
00000056 <sys_mailbox_send>:
  56:   55                      push   %ebp
  57:   57                      push   %edi
  58:   56                      push   %esi
  59:   be ed 03 00 00          mov    $0x3ed,%esi
  5e:   53                      push   %ebx
  5f:   83 ec 14                sub    $0x14,%esp
  62:   8a 44 24 34             mov    0x34(%esp),%al
  66:   8b 7c 24 30             mov    0x30(%esp),%edi
  6a:   8b 6c 24 28             mov    0x28(%esp),%ebp
  6e:   88 44 24 13             mov    %al,0x13(%esp)
  72:   a1 74 03 00 00          mov    0x374,%eax
  77:   c7 04 24 15 00 00 00    movl   $0x15,(%esp)
  7e:   89 44 24 04             mov    %eax,0x4(%esp)
```

- Recognize that 0x374? From the kernel oops message: BUG: unable to handle kernel NULL pointer dereference at 00000**374**

Step 7:

- Find these corresponding lines within the .s file.
    - Find the function name
    - In the .disassem file, there was one sub and then my oops was the 5th move command (884 = 0x374)

```
sys_mailbox_send:
.LFB1575:
        .loc 1 20 0
        .cfi_startproc
.LVL3:
        pushl   %ebp    #
.LCFI4:
        .cfi_def_cfa_offset 8
        pushl   %edi    #
.LCFI5:
        .cfi_def_cfa_offset 12
        pushl   %esi    #
.LCFI6:
        .cfi_def_cfa_offset 16
        .loc 1 26 0
        movl    $1005, %esi     #, D.27736
        .cfi_offset 6, -16
        .cfi_offset 7, -12
        .cfi_offset 5, -8
        .loc 1 20 0
        pushl   %ebx    #
.LCFI7:
        .cfi_def_cfa_offset 20
        subl    $20, %esp       #,
.LCFI8:
        .cfi_def_cfa_offset 40
        .loc 1 20 0
        movb    52(%esp), %al   # block,
        movl    48(%esp), %edi  # len, len
        movl    40(%esp), %ebp  # dest, dest
        movb    %al, 19(%esp)   #, %sfp
        .loc 1 24 0
        movl    884, %eax       # OB->mailbox, OB->mailbox
        movl    $.LC2, (%esp)   #,
        movl    %eax, 4(%esp)   # OB->mailbox,
        .cfi_offset 3, -20
        call    printk  #
```

Step 8:

- If that assembly line is not obvious to you (which most times it isn't, then you can add these lines to your .c source file: asm("#1")
- What this line does is break the assembly code with the tag you placed so that you can follow the assembly.

```
struct task_struct *ts = NULL;
asm("#1");
struct CS502_message *m;
asm("#2");
printk("ts->mailbox=%p\n", ts->mailbox);
asm("#3");
```

- After adding in these lines, you will want to make your kernel again and then regenerate the assembly file (will need to delete the old .s)

```
        subl    $20, %esp       #,
.LCFI8:
        .cfi_def_cfa_offset 40
        .loc 1 20 0
        movb    52(%esp), %al   # block,
        movl    40(%esp), %ebp  # dest, dest
        .cfi_offset 3, -20
        .cfi_offset 6, -16
        .cfi_offset 7, -12
        .cfi_offset 5, -8
        movl    48(%esp), %edi  # len, len
        movb    %al, 19(%esp)   #, %sfp
        .loc 1 23 0
#APP
# 23 "/home/student/WPI/proj4/finalSrc/kernel/mailbox.c" 1
        #1
# 0 "" 2
        .loc 1 25 0
# 25 "/home/student/WPI/proj4/finalSrc/kernel/mailbox.c" 1
        #2
# 0 "" 2
        .loc 1 26 0
#NO_APP
        movl    884, %eax       # OB->mailbox, OB->mailbox
        movl    $.LC2, (%esp)   #,
        movl    %eax, 4(%esp)   # OB->mailbox,
        call    printk  #
        .loc 1 27 0
#APP
# 27 "/home/student/WPI/proj4/finalSrc/kernel/mailbox.c" 1
        #3
# 0 "" 2
        .loc 1 28 0
```

- My cursor is pointing at the instruction that caused my oops. It is between labels #2 and #3. If you look at my source code, the instruction between #2 and #3 are "printk("ts->mailbox=%p\n", ts->mailbox)" and this was the line of code which caused my oops.