# Kernel Crash Analysis and Debugging

Max Bruning
Joyent
max@joyent.com
@mrbruning

# Introduction

- This tutorial will go through a few examples of problems on SmartOS and Linux
- Materials can be downloaded from github.com/max123/lisa2014
- Follow instructions in README.md to set up your machines, if you want to try it

# Types of System Crashes

- Bad Trap, Oops
  - Basically, SEGV in the kernel
- Calls to panic()
- Crashes are a safeguard
  - Prevent system from using (possibly) inconsistent data

# Why the Kernel Crashes

- Memory Leaks/Corruption
- Stack Overflow/Corruption
- Synchronization Problems
  - Missing locks
  - Deadlocks
  - Thundering Herd
  - Hardware Problems

# Debugging Strategies

- Use the kernel core file (if one exists) with a debugger(s)
- Live (in-situ) debugging of the running system
- Trace code paths
- Examine data structures
- There is no substitute for reading source code

# System Setup for Debugging

- SmartOS
  - Available at http://wiki.smartos.org/display/DOC/Getting+Started+with+SmartOS
  - Kernel debugger and DTrace come with the system
  - Can add additional debugging capabilities
    - kmem_flags
    - Build a “DEBUG” kernel

# System Setup

- Linux
  - System needs setup for both crash dumps and using a kernel debugger
    - Download source for linux kernel
  - kgdb/gdb/kdb
    - https://www.kernel.org/pub/linux/kernel/people/jwessel/kdb/index.html
  - Crash dumps
    - https://wiki.ubuntu.com/Kernel/CrashdumpRecipe

# Kernel Core Files

- SmartOS
  - `dumpadm(1M)` - Current settings for kernel crash dumps
  - If kernel panics, kernel memory (default) is written to the dump device and system reboots
  - On reboot, dump is written to /var/crash/*hostname*/vmdump.#
  - Dump file includes symbol table at time of panic

# Kernel Core Files (Continued)

- Linux (Ubuntu)

```
$ sudo apt-get install linux-crashdump
$ sudo tee /etc/apt/sources.list.d/ddebs.list << EOF
deb http://ddebs.ubuntu.com/ $(lsb_release -cs)          main restricted universe
multiverse
deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-security main restricted universe
multiverse
deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-updates  main restricted universe
multiverse
deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-proposed main restricted universe
multiverse
EOF
$
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys ECDCAD72428D7C01
$ sudo apt-get update
$ sudo apt-get install linux-image-$(uname -r)-dbgsym
$
$ sudo crash /usr/lib/debug/boot/vmlinux-3.13.0-32-generic /var/crash/201411041321/dump.
201411041321
```

# Tools for Examining Crash Dumps

- SmartOS
  - `mdb(1) - Modular Debugger`
  - `dtrace(1) - Dynamic/static tracing`
- Linux
  - `gdb`
  - `crash`

# Debugger Requirements

- Show Kernel stack backtraces
- Assembler level code
- Display Registers
- Examine kernel data structures/data
- Breakpoints/single step

# An Example Session - Linux

- Simple example of Oops panic
- Console output at time of crash
- How to get help in the debugger
- Stack backtrace
- Process list (what was running)
- Where Oops occurred in the code

# An Example Session - Linux (Continued)

- Assumes you have set up your system to get crash dumps (as described above)
- To cause crash:
  - `git clone https://github.com/max123/lisa2014.git`
  - Follow instructions here or in Readme

# Example Session (Continued)

```
$ cd lisa2014/linux/crashdriver
$ make crashapp
cc -O crashapp.c -o crashapp
$ make
...
...
$ sudo sh ./crash_load
$ ./crashapp nullpointer  <-- should cause system to crash
```

# Example Session (Continued)

```
$ cd /var/crash/201411091822
$ sudo crash /usr/lib/debug/boot/vmlinux-3.13.0-32-generic dump.201411091822
...
      KERNEL: /usr/lib/debug/boot/vmlinux-3.13.0-32-generic
    DUMPFILE: dump.201411091822  [PARTIAL DUMP]
        CPUS: 2
        DATE: Sun Nov  9 18:22:38 2014
      UPTIME: 1 days, 08:23:53
LOAD AVERAGE: 0.00, 0.01, 0.05
       TASKS: 217
    NODENAME: ubuntu
     RELEASE: 3.13.0-32-generic
     VERSION: #57-Ubuntu SMP Tue Jul 15 03:51:08 UTC 2014
     MACHINE: x86_64  (2394 Mhz)
      MEMORY: 2 GB
       PANIC: "Oops: 0002 [#1] SMP " (check log for details)
         PID: 4653
     COMMAND: "crashapp"
        TASK: ffff8800792517f0  [THREAD_INFO: ffff88003668e000]
         CPU: 1
       STATE: TASK_RUNNING (PANIC)

crash>
```

# Example Session - Console Output - Linux

```
crash> dmesg
...
[116728.719876] BUG: unable to handle kernel NULL pointer dereference at           (null)
[116728.727800] IP: [<ffffffffa028117f>] crash_ioctl+0x16f/0x197 [crash]
[116728.728616] PGD 7a246067 PUD 7bdfd067 PMD 0
[116728.729220] Oops: 0002 [#1] SMP
...
[116728.734944] RIP: 0010:[<ffffffffa028117f>]  [<ffffffffa028117f>] crash_ioctl+0x16f/
0x197 [crash]
[116728.735594] RSP: 0018:ffff88003668fec8  EFLAGS: 00010246
[116728.736114] RAX: 0000000000000000 RBX: ffff88003674ed00 RCX: 000000000000006b
[116728.736685] RDX: 0000000000000000 RSI: 0000000000006b02 RDI: ffff88003674ed00
[116728.737535] RBP: ffff88003668ff30 R08: 00007f2a5c678e80 R09: 00007f2a5c68e560
[116728.738395] R10: 00007fff6deb0340 R11: 0000000000000206 R12: ffff880079cbcc78
[116728.739242] R13: 0000000000000000 R14: 0000000000000000 R15: 0000000000000000
[116728.740078] FS:  00007f2a5c894740(0000) GS:ffff88007fc20000(0000) knlGS:
0000000000000000
[116728.740968] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[116728.741847] CR2: 0000000000000000 CR3: 00000000369a7000 CR4: 00000000001407e0
[116728.757017] RIP  [<ffffffffa028117f>] crash_ioctl+0x16f/0x197 [crash]
[116728.757952]  RSP <ffff88003668fec8>
[116728.758862] CR2: 0000000000000000
```

# Console Output - Linux

```
[116728.742832] Stack:
[116728.743716]  ffffffff811cfd10 0000000000000000 ffff88003668fef8 ffffffff8109ddf4
[116728.744627]  0000000000000246 00007fff6deb0678 ffff88003668ff18 0000000000000246
[116728.745522]  ffff88003668ff58 ffff88003674ed00 0000000000000003 0000000000006b02
[116728.746430] Call Trace:
[116728.751231]  [<ffffffff811cfd10>] ? do_vfs_ioctl+0x2e0/0x4c0
[116728.752312]  [<ffffffff8109ddf4>] ? vtime_account_user+0x54/0x60
[116728.752957]  [<ffffffff811cff71>] SyS_ioctl+0x81/0xa0
[116728.754518]  [<ffffffff8172c87f>] tracesys+0xe1/0xe6
[116728.755084] Code: 3f 3f 3f 3f 48 83 c7 04 f6 c2 02 74 09 66 c7 07 3f 3f 48 83 c7 02 f6
c2 01 74 03 c6 07 3f b8 00 00 00 00 c3 48 8b 05 e1 21 00 00 <48> c7 00 3f 00 00 00 b8 00 00
00 00 c3 48 c7 c0 e7 ff ff ff c3
[116728.757017] RIP  [<ffffffffa028117f>] crash_ioctl+0x16f/0x197 [crash]
[116728.757952]  RSP <ffff88003668fec8>
[116728.758862] CR2: 0000000000000000
crash>
```

# Stack Backtrace - Linux

```
crash> bt
PID: 4653   TASK: ffff88007925l7f0  CPU: 1   COMMAND: "crashapp"
 #0 [ffff88003668faf0] machine_kexec at ffffffff8104a742
 #1 [ffff88003668fb40] crash_kexec at ffffffff810e6cf3
 #2 [ffff88003668fc08] oops_end at ffffffff817251a8
 #3 [ffff88003668fc30] no_context at ffffffff81714974
 #4 [ffff88003668fc78] __bad_area_nosemaphore at ffffffff817149f4
 #5 [ffff88003668fcc0] bad_area at ffffffff81714d6d
 #6 [ffff88003668fce8] __do_page_fault at ffffffff81727ed9
 #7 [ffff88003668fde8] do_page_fault at ffffffff81727fda
 #8 [ffff88003668fe10] page_fault at ffffffff81724448

 ...
```

# Stack Backtrace - Linux

```
    [exception RIP: crash_ioctl+367]
    RIP: ffffffffa028117f  RSP: ffff88003668fec8  RFLAGS: 00010246
    RAX: 0000000000000000  RBX: ffff88003674ed00  RCX:
00000000000006b
    RDX: 0000000000000000  RSI: 0000000000006b02  RDI:
ffff88003674ed00
    RBP: ffff88003668ff30   R8: 00007f2a5c678e80   R9:
00007f2a5c68e560
    R10: 00007fff6deb0340  R11: 0000000000000206  R12:
ffff880079cbcc78
    R13: 0000000000000000  R14: 0000000000000000  R15:
0000000000000000
    ORIG_RAX: ffffffffffffffff  CS: 0010  SS: 0018
 #9 [ffff88003668fec8] do_vfs_ioctl at ffffffff811cfd10
#10 [ffff88003668ff38] sys_ioctl at ffffffff811cff71
#11 [ffff88003668ff80] tracesys at ffffffff8172c87f (via
system_call)
    RIP: 00007f2a5c3a9e77  RSP: 00007fff6deb0578  RFLAGS: 00000206
    RAX: ffffffffffffffda  RBX: ffffffff8172c87f  RCX:
ffffffffffffffff
...
```

# Stack Backtrace - Linux

```
    RDX: 0000000000000000  RSI: 0000000000006b02  RDI:
000000000000003
    RBP: 0000000000000003   R8: 00007f2a5c678e80   R9:
00007f2a5c68e560
    R10: 00007fff6deb0340  R11: 0000000000000206  R12:
0000000000000000
    R13: 0000000000000000  R14: 00007fff6deb0670  R15:
00007fff6deb0678
    ORIG_RAX: 0000000000000010  CS: 0033  SS: 002b
crash>
```

# Where Oops Occurred

```
crash> dis -l crash_ioctl+367
0xffffffffa028117f <crash_ioctl+367>:   movq   $0x3f,(%rax)
crash>
```

- From stack backtrace output, %rax is 0
- Where is %rax set?

```
crash> dis crash_ioctl
...
0xffffffffa028116f <crash_ioctl+351>:   movb   $0x3f,(%rdi)
0xffffffffa0281172 <crash_ioctl+354>:   mov    $0x0,%eax
0xffffffffa0281177 <crash_ioctl+359>:   retq
0xffffffffa0281178 <crash_ioctl+360>:   mov    0x21e1(%rip),%rax
# 0xffffffffa0283360
0xffffffffa028117f <crash_ioctl+367>:   movq   $0x3f,(%rax)
0xffffffffa0281186 <crash_ioctl+374>:   mov    $0x0,%eax
0xffffffffa028118b <crash_ioctl+379>:   retq
...
```

# Source of the Bug

```
caddr_t *crash_addr;

long crash_ioctl(struct file *filp,
                 unsigned int cmd, unsigned long arg)
{
...
    case CRASH_NULLPOINTER:
        *crash_addr = (char *)'?';
        break;
...
```

# What is Running?

```
crash> ps
WARNING: terminal is not fully functional
-  (press RETURN)
   PID    PPID  CPU       TASK        ST  %MEM     VSZ    RSS  COMM
>     0      0   0  ffffffff81c15480  RU   0.0       0      0
[swapper/0]
      0      0   1  ffff88007c06c7d0  RU   0.0       0      0
[swapper/1]
      1      0   1  ffff88007c5c0000  IN   0.1   33488   2744  init
      2      0   1  ffff88007c5c17f0  IN   0.0       0      0
[kthreadd]
      3      2   0  ffff88007c5c2fe0  IN   0.0       0      0
[ksoftirqd/0]
      5      2   0  ffff88007c5c5fc0  IN   0.0       0      0
[kworker/0:0H]
      7      2   1  ffff88007c5f17f0  RU   0.0       0      0
[rcu_sched]
      8      2   0  ffff88007c5f2fe0  IN   0.0       0      0
[rcuos/0]
...
   4652   2913   0  ffff880079250000  IN   0.1   64952   2124  sudo
>  4653   4652   1  ffff880079251f0  RU   0.0    4192    360
crashapp
crash>
```

# Other Stack Traces

```
crash> bt ffff88007c5c0000
PID: 1      TASK: ffff88007c5c0000  CPU: 1   COMMAND: "init"
 #0 [ffff88007c54f8b8] __schedule at ffffffff8171fc19
 #1 [ffff88007c54f920] schedule at ffffffff817200d9
 #2 [ffff88007c54f930] schedule_hrtimeout_range_clock at
ffffffff8171f73d
 #3 [ffff88007c54f9c8] schedule_hrtimeout_range at ffffffff8171f773
 #4 [ffff88007c54f9d8] poll_schedule_timeout at ffffffff811d07c9
 #5 [ffff88007c54f9f8] do_select at ffffffff811d11b6
 #6 [ffff88007c54fd90] core_sys_select at ffffffff811d154c
 #7 [ffff88007c54ff20] sys_select at ffffffff811d170b
 #8 [ffff88007c54ff80] tracesys at ffffffff8172c87f (via
system_call)
    RIP: 00007f5c656cd8c3  RSP: 00007fff0357e068  RFLAGS: 00000246
    RAX: ffffffffffffffda  RBX: ffffffff8172c87f  RCX:
ffffffffffffffff
    RDX: 00007fff0357e130  RSI: 00007fff0357e0b0  RDI:
0000000000000013
    RBP: 00007f5c6663d110   R8: 0000000000000000   R9:
00007f5c67e3d3d0
...
crash>
```

# Getting Help

```
crash> help bt
NAME
  bt - backtrace

SYNOPSIS
  bt [-a|-g|-r|-t|-T|-l|-e|-E|-f|-F|-o|-O] [-R ref] [-s [-x|d]]
[-I ip] [-S sp]
     [pid | task]

DESCRIPTION
  Display a kernel stack backtrace.  If no arguments are given,
the stack
  trace of the current context will be displayed.

       -a  displays the stack traces of the active task on each
CPU.
           (only applicable to crash dumps)
...
```

# Printing Data Structures

```
crash> task ffff88007c5c0000
PID: 1      TASK: ffff88007c5c0000  CPU: 1   COMMAND: "init"
struct task_struct {
  state = 1,
  stack = 0xffff88007c54e000,
  usage = {
    counter = 2
  },
  flags = 4219136,
  ptrace = 0,
  wake_entry = {
    next = 0x0
...
```

# Bad Trap Example - SmartOS

- No need to run debug kernel or install any additional software to get dump

```
# dumpadm
      Dump content: kernel pages
       Dump device: /dev/zvol/dsk/zones/dump (dedicated)
Savecore directory: /var/crash/volatile
  Savecore enabled: yes
   Save compressed: on
```

# Crashing on SmartOS

- You'll need to either create a zone on SmartOS or `scp` the files from github.com/max123/lisa2014/SmartOS to the global zone. (no git in global zone)
- See http://wiki.smartos.org/display/DOC/How+to+create+a+zone+%28+OS+virtualized+machine+%29+in+SmartOS for instructions on creating a zone.

# Crashing on SmartOS Example

```
# cp bdtrp /kernel/drv/amd64
# cp bdtrp.conf /kernel/drv
# add_drv bdtrp
# cat /devices/pseudo/bdtrp@0:bdtrp
```

- This should panic the system

- Dump will be in `/var/crash/volatile/`

```
# cd /var/crash/volatile
# savecore -f vmdump.0
savecore: System dump time: Mon Nov 10 22:18:14 2014
savecore: saving system crash dump in /var/crash/volatile/
{unix,vmcore}.0
Constructing namelist /var/crash/volatile/unix.0
Constructing corefile /var/crash/volatile/vmcore.0
 0:02 100% done: 98761 of 98761 pages saved
#
```

# Console Output on SmartOS

```
# mdb 0
Loading modules: [ unix genunix specfs dtrace mac cpu.generic
uppc pcplusmp scsi_vhci ufs ip hook neti sockfs arp usba
stmf_sbd stmf zfs lofs mpt idm sd crypto random cpc logindmux
ptm kvm sppp nsmb smbsrv nfs ]
>
> msgbuf
MESSAGE
sd1 is /pci@0,0/pci15ad,1976@10/sd@1,0
/pci@0,0/pci15ad,1976@10/sd@1,0 (sd1) online
...
panic[cpu0]/thread=ffffff00c70dc3a0:
BAD TRAP: type=e (#pf Page fault) rp=ffffff00025b7a80 addr=8
occurred in module
"bdtrp" due to a NULL pointer dereference
...
```

# Console Output Continued

```
...
cat:
#pf Page fault
Bad kernel fault at addr=0x8
pid=13048, pc=0xffffffff8161f49, sp=0xffffff00025b7b70,
eflags=0x10286
cr0: 8005003b<pg,wp,ne,et,ts,mp,pe> cr4:
6b8<xmme,fxsr,pge,pae,pse,de>
cr2: 8
cr3: 121ab000
cr8: c
...
```

# Console Output Continued

```
rdi:          c300000000 rsi: ffffff00025b7de0 rdx: ffffff00ccb0ca30
rcx:                  c3  r8:                2  r9: ffffff00c5a6f080
rax:                   0 rbx: ffffff00025b7de0 rbp: ffffff00025b7bb0
r10: ffffff00d111fe40 r11:                   0 r12: ffffff00d111fe40
r13:          c300000000 r14: ffffff00d8135e60 r15: ffffff00ccb0ca30
fsb:                   0 gsb: fffffffffbc32600  ds:                 4b
 es:                  4b  fs:                   0  gs:                1c3
trp:                   e err:                   0 rip: ffffffffff8161f49
 cs:                  30 rfl:               10286 rsp: ffffff00025b7b70
 ss:                  38

ffffff00025b7960 unix:die+df ()
ffffff00025b7a70 unix:trap+db3 ()
ffffff00025b7a80 unix:cmntrap+e6 ()
ffffff00025b7bb0 bdtrp:bdtrp_read+2f ()
ffffff00025b7be0 genunix:cdev_read+2d ()
ffffff00025b7c80 specfs:spec_read+2b9 ()
ffffff00025b7d20 genunix:fop_read+8b ()
ffffff00025b7e80 genunix:read+2a7 ()
ffffff00025b7eb0 genunix:read32+1e ()
ffffff00025b7f10 unix:brand_sys_sysenter+1d3 ()

syncing file systems...
 1
 done
```

# Stack Backtrace

```
> ::stack
bdtrp_read+0x2f()
cdev_read+0x2d(c300000000, ffffff00025b7de0, ffffff00ccb0ca30)
spec_read+0x2b9(ffffff00d111fe40, ffffff00025b7de0, 0,
ffffff00ccb0ca30, 0)
fop_read+0x8b(ffffff00d111fe40, ffffff00025b7de0, 0,
ffffff00ccb0ca30, 0)
read+0x2a7(3, 80632c0, 2000)
read32+0x1e(3, 80632c0, 2000)
_sys_sysenter_post_swapgs+0x153()
>
> <rip/i
bdtrp_read+0x2f:movq    0x8(%rax),%rax
>
```

# Disassemble Function

```
> bdtrp_read::dis
bdtrp_read:                 pushq   %rbp
bdtrp_read+1:               movq    %rsp,%rbp
bdtrp_read+4:               subq    $0x40,%rsp
bdtrp_read+8:               movq    %rdi,-0x28(%rbp)
bdtrp_read+0xc:             movq    %rsi,-0x30(%rbp)
bdtrp_read+0x10:            movq    %rdx,-0x38(%rbp)
bdtrp_read+0x14:            movq    -0x28(%rbp),%rax
bdtrp_read+0x18:            movq    %rax,%rdi
bdtrp_read+0x1b:            call    +0x38627e6          <getminor>
bdtrp_read+0x20:            movl    %eax,-0x4(%rbp)
bdtrp_read+0x23:            movq    $0x0,-0x10(%rbp)
bdtrp_read+0x2b:            movq    -0x10(%rbp),%rax
bdtrp_read+0x2f:            movq    0x8(%rax),%rax
...
bdtrp_read+0x4f:            movl    $0x0,%eax
bdtrp_read+0x54:            leave
bdtrp_read+0x55:            ret
> <rbp-10/K
0xffffff00025b7ba0:         0
```

# Source of the Bug

```
/*ARGSUSED*/
static int
bdtrp_read(dev_t dev, struct uio *uiop, cred_t *credp)
{
    int instance = getminor(dev);
    volatile dev_info_t devinfop;
    bdtrp_devstate_t *rsp = NULL;

    devinfop = rsp->dip;
    rsp = ddi_get_soft_state(bdtrp_state, instance);
    return(0);

}
```

# What is Running

```
> ::ps
S    PID   PPID   PGID    SID    UID      FLAGS             ADDR
NAME
R      0      0      0      0      0 0x00000001 ffffffffbc30440
sched
R     88      0      0      0      0 0x00020001 ffffff00c7136068
zpool-zones
R      3      0      0      0      0 0x00020001 ffffff00c6612020
fsflush
R      2      0      0      0      0 0x00020001 ffffff00c6615018
pageout
R      1      0      0      0      0 0x4a004000 ffffff00c6619010
init
...
R   2389   2386   2389   2389      0 0x4a014000 ffffff00cc87e060
bash
R  13048   2389  13048   2389      0 0x4a004000 ffffff00d3f23088
cat
R   2270      1   2267   2267      0 0x42000000 ffffff00cc8e6080
auditd
...
```

# Other Stack Traces

```
> ::pgrep login | ::walk thread | ::findstack <-- show all threads
for login process
stack pointer for thread ffffff00c6ddc180: ffffff00045eac30
[ ffffff00045eac30 _resume_from_idle+0xf4() ]
  ffffff00045eac60 swtch+0x141()
  ffffff00045eacf0 cv_wait_sig_swap_core+0x1b9()
  ffffff00045ead10 cv_wait_sig_swap+0x17()
  ffffff00045eada0 waitid+0x2b3()
  ffffff00045eaeb0 waitsys32+0x36()
  ffffff00045eaf10 _sys_sysenter_post_swapgs+0x153()
> ::threadlist -v  <-- show kernel stack traces for all threads
             ADDR                 PROC                 LWP CLS  PRI
WCHAN
fffffffffbc313a0 fffffffffbc30440 fffffffffbc32ec0   0   96
0
  PC: _resume_from_idle+0xf4    CMD: sched
  stack pointer for thread fffffffffbc313a0: fffffffffbc73af0
  [ fffffffffbc73af0 _resume_from_idle+0xf4() ]
    swtch+0x141()
    sched+0x835()
    main+0x46c()
    _locore_start+0x90()
...
```

# Getting Help

- Show list of commands (dcmds)

```
> ::dcmds
...
$?                          - print status and registers
$C                          - print stack backtrace
$G                          - enable/disable C++ demangling support
$M                          - list macro aliases
$P                          - set debugger prompt string
$Q                          - quit debugger
$V                          - get/set disassembly mode
$W                          - re-open target in write mode
$X                          - print floating point registers
$Y                          - print floating point registers
...
/                           - format data from virtual as
\                           - format data from physical as
abuf_find                   - find arc_buf_hdr_t of a specified DVA
acl                         - given an inode, display its in core acl's
...
```

# Getting Help for a dcmd

```
> ::dcmds !wc
     527    3558   31033 <-- 527 different dcmds
> ::help print

NAME
  print - print the contents of a data structure

SYNOPSIS
  [ addr ] ::print [-aCdhiLptx] [-c lim] [-l lim] [type]
[member|offset ...]

DESCRIPTION

  -a          show address of object
  -C          unlimit the length of character arrays
  -c limit    limit the length of character arrays
  -d          output values in decimal
  -x          output values in hexadecimal

 ...
```

# Help Finding a dcmd

```
> ::dcmds !grep process
:A                              - attach to process or core file
:R                              - release the previously attached process
:r                              - run a new target process
attach                          - attach to process or core file
class                           - print process scheduler classes
gcore                           - generate a user core for the given
process
pfiles                          - print process file information
pgrep                           - pattern match against all processes
pmap                            - print process memory map
ps                              - list processes (and associated thr,lwp)
ptree                           - print process tree
release                         - release the previously attached process
run                             - run a new target process
> ::pgrep cat | ::ptree
fffffffffbc30440  sched
     ffffff00c6619010  init
          ffffff00cc9d70b0  sshd
               ffffff00cca3b000  sshd
                    ffffff00cc85d040  sshd
                         ffffff00cc87e060  bash
                              ffffff00d3f23088  cat
```

# Walkers

- Structure iterators

```
> ::walk proc | ::print proc_t p_user.u_psargs
p_user.u_psargs = [ "sched" ]
p_user.u_psargs = [ "zpool-zones" ]
p_user.u_psargs = [ "fsflush" ]
p_user.u_psargs = [ "pageout" ]
p_user.u_psargs = [ "/sbin/init" ]
p_user.u_psargs = [ "atd" ]
p_user.u_psargs = [ "cron" ]
p_user.u_psargs = [ "zsched" ]
p_user.u_psargs = [ "/sbin/init" ]
...
> ::walkers !wc
     696    4734   42157
> ::walkers !grep process
pid2state                   - walk a processes dtrace_state
structures
process_cache               - walk the process_cache cache
task                        - given a task pointer, walk its
processes
thread                      - global or per-process kthread_t
structures
>
```

# Memory Corruption - Causes

- Use after free or before initialization
- Buffer overflow
- Incorrect or missing synchronization
- Stack corruption
- Hardware error
- Symptoms can appear anywhere, often unrelated to cause of problem

# Memory Corruption Debugging

- Linux
  - Set `CONFIG_DEBUG_SLAB`
    - Buffer "poisoning" (pattern is 0x5a)
    - Buffer over/underflow checking
- Other memory allocators also have debug flags
- Set `CONFIG_DEBUG_STACKOVERFLOW`

# Memory Corruption Debugging

- SmartOS
  - From kmdb boot (choice in grub menu during boot):
    - `kmem_flags/W 0xf` *<- turn on debug flags*
    - `:c` *<- continue (boot)*
  - Turns on “use after free”, buffer overflow, and auditing of allocations and frees

# Memory Corruption Example - Linux

```
$ cd lisa2014/linux/crashdriver
$ make crashapp
cc -O crashapp.c -o crashapp
$ make
...
$ sudo sh ./crash_load
$ sudo ./crashapp deadbeef
```

*wait ~5 minutes.  You may want to run some stuff in the background*

# Memory Corruption Example - Linux

```
$ cd /var/crash/201411112037
$ sudo crash /usr/lib/debug/boot/vmlinux-3.13.0-32-generic dump.
201411112037
...
      KERNEL: /usr/lib/debug/boot/vmlinux-3.13.0-32-generic
    DUMPFILE: dump.201411112037  [PARTIAL DUMP]
        CPUS: 2
        DATE: Tue Nov 11 20:37:07 2014
      UPTIME: 01:40:41
LOAD AVERAGE: 1.17, 0.81, 0.39
       TASKS: 222
    NODENAME: ubuntu
     RELEASE: 3.13.0-32-generic
     VERSION: #57-Ubuntu SMP Tue Jul 15 03:51:08 UTC 2014
     MACHINE: x86_64  (2393 Mhz)
      MEMORY: 2 GB
       PANIC: ""
         PID: 2217
     COMMAND: "grep"
        TASK: ffff8800790bc7d0  [THREAD_INFO: ffff880036a5e000]
         CPU: 1
       STATE: TASK_RUNNING (PANIC)
```

# Memory Corruption Example - Linux

```
crash> bt
PID: 2217   TASK: ffff8800790bc7d0  CPU: 1   COMMAND: "grep"
 ...
 #5 [ffff880036a5fdd0] general_protection at ffffffff817243e8
    [exception RIP: kmem_cache_alloc+117]
    RIP: ffffffff811a1c25  RSP: ffff880036a5fe80  RFLAGS: 00010286
    RAX: 0000000000000000  RBX: 0000000000008000  RCX:
00000000002a8e7c
    RDX: 00000000002a8e7b  RSI: 00000000000000d0  RDI:
ffff88007d403300
    RBP: ffff880036a5feb0   R8: 00000000000172e0   R9:
ffffffff811cc9af
    R10: 0000000000000000  R11: 0000000000000246  R12:
3f3f3f3f3f3f3f3f
    R13: 00000000000000d0  R14: ffff88007d403300  R15:
ffff88007d403300
    ORIG_RAX: ffffffffffffffff  CS: 0010  SS: 0018
 #6 [ffff880036a5feb8] getname_flags at ffffffff811cc9af
 ...
 #9 [ffff880036a5ff70] sys_openat at ffffffff811bc014
#10 [ffff880036a5ff80] tracesys at ffffffff8172c87f (via
```

# Memory Corruption Example

```
crash> dis -r -l kmem_cache_alloc+117
/build/buildd/linux-3.13.0/mm/slub.c: 2459
0xffffffff811a1bb0 <kmem_cache_alloc>:  nopl   0x0(%rax,%rax,1)
0xffffffff811a1bb5 <kmem_cache_alloc+5>:         push   %rbp
0xffffffff811a1bb6 <kmem_cache_alloc+6>:         mov    %rsp,%rbp
...
/build/buildd/linux-3.13.0/mm/slub.c: 260
0xffffffff811a1c1a <kmem_cache_alloc+106>:       movslq 0x20(%r14),
%rax
/build/buildd/linux-3.13.0/mm/slub.c: 1727
0xffffffff811a1c1e <kmem_cache_alloc+110>:       lea    0x1(%rdx),
%rcx
/build/buildd/linux-3.13.0/mm/slub.c: 2432
0xffffffff811a1c22 <kmem_cache_alloc+114>:       mov    (%r14),%r8
/build/buildd/linux-3.13.0/mm/slub.c: 260
0xffffffff811a1c25 <kmem_cache_alloc+117>:       mov    (%r12,%rax,
1),%rbx
```

# Memory Corruption Example

mm/slub.c

```
258 static inline void *get_freepointer(struct kmem_cache
*s, void *object)
259 {
260         return *(void **)(object + s->offset);
261 }
```

- A check on other line numbers and files in the output show that source and binary are probably not in sync
- Inlining functions can make debugging more difficult

# What's Running

```
crash> ps | grep "> "  <-- show running processes
>     0       0   0  ffffffff81c15480  RU   0.0         0        0
[swapper/0]
>  2217   1981   1  ffff8800790bc7d0  RU   0.0      8996      796
grep
crash> ps  <-- look at all processes for anything unusual
...
   1528   1527   1  ffff880079238000  IN   0.0      4192      360
crashapp
...
> crash> bt 1528
PID: 1528   TASK: ffff880079238000  CPU: 1   COMMAND: "crashapp"
 #0 [ffff880036b35ef0] __schedule at ffffffff8171fc19
 #1 [ffff880036b35f58] schedule at ffffffff817200d9
 #2 [ffff880036b35f68] sys_pause at ffffffff8107be8c
 #3 [ffff880036b35f80] tracesys at ffffffff8172c87f (via
system_call)  <-- it has called pause.  good thing it didn't exit
```

# Memory Corruption Example - SmartOS

- The same problem as the last. But with kmem_flags set to 0xf.

```
Loading kmdb...  <-- from kmdb option at boot

Welcome to kmdb
Loaded Modules: [ unix krtld genunix ]
[0]> kmem_flags/W 0xf
kmem_flags:     0                =     0xf
[0]> :c
... system boots, when system is up:
# path_to_corrupt/corrupt deadbeef
...  the system should panic shortly after
In the debugger,
[0]> :c
and the system should boot with a dump
```

# Memory Corruption - SmartOS

```
# cd /var/crash/volatile
# savecore -f vmdump.3
...
# mdb 3
...
> ::msgbuf
...
kernel memory allocator: buffer modified after being freed
modification occurred at offset 0x0 (0xdeadbeefdeadbeef
replaced by 0x3f3f3f3f3f3f3f3f)
buffer=ffffff00f18794c0  bufctl=ffffff00f18aaec8  cache:
kmem_alloc_256
previous transaction on buffer ffffff00f18794c0:
thread=ffffff0003ee6c40  time=T-30.081796501
slab=ffffff00f1fa5450  cache: kmem_alloc_256
kmem_cache_free_debug+10f
kmem_cache_free+123
kmem_free+4e
...
```

# More Console Output

```
panic[cpu0]/thread=ffffff00dd09f0c0:
kernel heap corruption detected


ffffff0002173ad0 ffffffffba289a4 ()
ffffff0002173b60 genunix:kmem_cache_alloc_debug+1e7 ()
ffffff0002173bc0 genunix:kmem_cache_alloc+d4 ()
ffffff0002173c00 genunix:kmem_zalloc+47 ()
ffffff0002173c40 genunix:anon_create+81 ()
ffffff0002173c90 genunix:anonmap_alloc+61 ()
ffffff0002173d60 genunix:segvn_dup+1bb ()
ffffff0002173de0 genunix:as_dup+11f ()
ffffff0002173e90 genunix:cfork+996 ()
ffffff0002173eb0 genunix:forksys+3c ()
ffffff0002173f10 unix:brand_sys_syscall32+184 ()
>
```

# What size is being Allocated?

```
> $c
vpanic()
0xffffffffba289a4()
kmem_cache_alloc_debug+0x1e7(ffffff00c4829c88, ffffff00f18794c0,
0, 0,
ffffffffba2b147)
kmem_cache_alloc+0xd4(ffffff00c4829c88, 0)
kmem_zalloc+0x47(100, 0)  <-- 256 bytes
anon_create+0x81(20, 0)
anonmap_alloc+0x61(20000, 0, 0)
segvn_dup+0x1bb(ffffff00f7e6dad8, ffffff00f66be2d8)
as_dup+0x11f(ffffff00f66d6400, ffffff01180c90b0)
cfork+0x996(0, 1, 0)
forksys+0x3c(0, 0)
sys_syscall32+0x109()
>
```

# All the Allocations

```
> ::kmalog
T-0.000000000  addr=ffffff00f4ffce00  kmem_alloc_32
          kmem_cache_alloc_debug+0x2e0
          kmem_cache_alloc+0xd4
          kmem_zalloc+0x47
          anon_create+0x3a
          anonmap_alloc+0x61
...
T-30.081795702  addr=ffffff00f18794c0  kmem_alloc_256
          kmem_cache_free_debug+0x10f
          kmem_cache_free+0x123
          kmem_free+0x4e
          zil_itxg_clean+0xab
...
T-31.866416494  addr=ffffff00f18794c0  kmem_alloc_256
          kmem_cache_free_debug+0x10f
          kmem_cache_free+0x123
          kmem_free+0x4e
          corrupt_ioctl+0x4e
          cdev_ioctl+0x39
          spec_ioctl+0x60
          fop_ioctl+0x55
          ioctl+0x9b
```

# The Bug

```
int
corrupt_ioctl(dev_t dev, int cmd, intptr_t arg, int
mode,
          cred_t *cred_p, int *rval_p)
{
    int retval = 0;

    switch(cmd) {

    case CRASH_ALLOC:
        crash_addr = kmem_alloc(256, KM_SLEEP);
        kmem_free(crash_addr, 256);
        break;

    case CRASH_USEAFTERFREE:
        memset(crash_addr, '?', 256);
        break;
```

# Something else

- Joyent is working on “lx branded zones”
- Run native linux apps on top of SmartOS kernel in a SmartOS zone
- `mdb` **and** `dtrace` **are available on Linux!**

# lx branded zones setup

```
# imgadm avail | grep lx  <-- in global zone on SmartOS
b7493690-f019-4612-958b-bab5f844283e  lx-ubuntu
14.04.002  other    2014-07-23T12:00:59Z
# imgadm import b7493690-f019-4612-958b-bab5f844283e
...
# vmadm create -f lx.json
...
# vmadm list
UUID                                  TYPE  RAM      STATE
ALIAS
20cc3ca2-b5a3-4c7f-bfe9-1278019db7cf  LX    512      running
lxtest
# zlogin 20cc3ca2-b5a3-4c7f-bfe9-1278019db7cf
[Connected to zone '20cc3ca2-b5a3-4c7f-bfe9-1278019db7cf'
pts/3]
Last login: Mon Nov 10 15:21:43 MST 2014 from zone:global on
pts/2
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0 i686)
...
root@ubuntu14:~#
```

# DTrace and mdb on Linux

```
# LD_LIBRARY_PATH=/native/lib/amd64:/native/usr/lib/
amd64:$LD_LIBRARY_PATH dtrace -n 'pid
$target::strcmp:entry{printf("strcmp entered\n");}' -c
"find / -name '*foo*'" 2>/dev/null
CPU     ID                    FUNCTION:NAME
  0  65955                     strcmp:entry strcmp entered

  0  65955                     strcmp:entry strcmp entered
...
# mdb -p $$
> $c
libc_hwcap1.so.1`__waitid+0x15(7, 0, 8047b50, f, 0, 0)
libc_hwcap1.so.1`waitpid+0x75(ffffffff, 8047c2c, c, 0,
3, fed52bcc)
waitchld+0x7d(15b2, 1, 0, 8, 80f4676, 81f12c8)
wait_for+0x1ed(15b2, 0, 15af, 15af, 0, 0)
execute_command_internal+0x1610(81ed6a8, 0, ffffffff,
ffffffff, 81f1368, 81f1368
)
execute_command+0x45(81ed6a8, 0, 8047d98, 8071202)
reader_loop+0x7e(fef80530, fef80530, 3, d7acdc4, 0,
8047e40)
main+0xd16(8047dec, fef23668, 8047e2c, 806ebcf, 1,
```