What's in a Kernel Oops?

Vlastimil Babka, LinuxDays 2015

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Department of Distributed and Dependable Systems

D3S

Kernel oops/panic output

- Printed in console typically on fatal CPU exceptions
  - Lots of architecture-specific information
  - May be enough to figure out the bug without a crash dump
- Oops leaves the system running
  - Kills just the current process (including kernel threads!)
  - System can still be left inconsistent (locks remain locked …)
- Panic kills the system completely
  - Oops in interrupt, with `panic_on_oops` enabled, manual `panic()` calls
    - HW failure, critical memory allocation fail, init/idle task killed, int. handler killed
  - May trigger crash dump if configured, or reboot after delay

Example kernel oops (x86_64)

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff8800916fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff88008f6b3ba0 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8800914b3c98 RSI: 0000000000000001 RDI: ffff8800914b3c98
[  266.492376] RBP: ffff8800916fbba8 R08: 0000000000000002 R09: 0000000000000000
[  266.492421] R10: 0000000000000008 R11: 0000000000000001 R12: ffff88008f686068
[  266.492465] R13: ffff8800914b3c98 R14: ffff88008f6b3b90 R15: ffff88008f686000
[  266.492513] FS:  00007fb8966f6700(0000) GS:ffff88011ed80000(0000)
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 0000000001b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Example kernel oops (x86_64)

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111ba5d>] free_pgtables+0x8e/0xcc
[  266.492873]  [<ffffffff8112253e>] exit_mmap+0x84/0x116
[  266.492907]  [<ffffffff8103f789>] mmput+0x52/0xe9
[  266.492940]  [<ffffffff81043918>] do_exit+0x3cd/0x9c9
[  266.492975]  [<ffffffff8170c1ec>] ? _raw_spin_unlock_irq+0x2d/0x32
[  266.493016]  [<ffffffff81044d7f>] do_group_exit+0x4c/0xc9
[  266.493051]  [<ffffffff8104eb87>] get_signal+0x58f/0x5bc
[  266.493090]  [<ffffffff810022c4>] do_signal+0x28/0x5b1
[  266.493123]  [<ffffffff8170ca0c>] ? sysret_signal+0x5/0x43
[  266.493162]  [<ffffffff81002882>] do_notify_resume+0x35/0x68
[  266.493200]  [<ffffffff8170cc7f>] int_signal+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48 8b 7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

(source: https://lkml.org/lkml/2015/1/11/14)

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdk  d amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm:  Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLi  allina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 t  f8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff8112  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff880091 fbb  00010286
[  266.492285] RAX  3b90 RCX: ffff8800a3b3cf30
[  266.492331] RD  01 RDI: ffff8800914b3c98
[  266.492376] RE  02 R09: 0000000000000000
[  266.492421] R1  01 R12: ffff88008f686068
[  266.492465] R1  90 R15: ffff88008f686000
[  266.492513] FS  d80000(0000)
knlGS:0000000000
[  266.492566] CS  80050033
[  266.492601] CR  00 CR4: 00000000000407e0
[  266.492652] St
[  266.492665]  0  00916fbba8 ffff88008f686000
[  266.492714]  f  0000000000 ffff88008f686000
[  266.492764]  f  b885918000 ffff88008edf3000
...
```

File + line translation enabled by CONFIG_DEBUG_BUGVERBOSE (implemented by `__bug_table` section on x86 - ~70-100kB)

The line in question contains:
`BUG_ON(anon_vma->degree);`

This is essentially a hard assertion:
`if (<condition>) BUG()`

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware na      AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8      b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[      fff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x1
[  266.492249] RSP: 0018:f      16fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff88      0 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[                               0000000000000001 RDI: ffff8800914b3c98
[                               0000000000000002 R09: 0000000000000000
[                               0000000000000001 R12: ffff88008f686068
[                               ffff88008f6b3b90 R15: ffff88008f686000
[                                GS:ffff88011ed80000(0000)
kn
[                               CR0: 0000000080050033
[                               0000000001b31000 CR4: 00000000000407e0
[
[                               8f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]                  0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

On x86, BUG() emits a standardized invalid opcode UD2 (0F 0B) triggering an exception.
The exception handler checks for UD2 opcode and searches __bug_table for details.

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm f e
[  266.492043] CPU: 3 PID: 5155 mm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: A BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a 40 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ff 81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:fff 5fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff880  RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.4  I: 0000000000000001 RDI: ffff8800914b3c98
[  266  0000000000000002 R09: 0000000000000000
[  26  0000000000000001 R12: ffff88008f686068
[  26  ffff88008f6b3b90 R15: ffff88008f686000
[  26  GS:ffff88011ed80000(0000)
knlGS
[  26  CR0: 0000000080050033
[  26  0000000001b31000 CR4: 00000000000407e0
[  26
[  26  f686078 ffff8800916fbba8 ffff88008f686000
[  26  0000000 0000000000000000 ffff88008f686000
[  26  111ba5d 00007fb885918000 ffff88008edf3000
...
```

x86- and exception-specific error code (32-bit hex number). Typically useful for page fault exceptions where it's a mask:

Bit 0 – Present
Bit 1 – Write
Bit 2 – User
Bit 3 – Reserved write
Bit 4 – Instruction fetch

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD F          INA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b          ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffff          126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff          bb68  EFLAGS: 00010286
[  266.492285] RAX: ffff880          RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266          0000000000000001 RDI: ffff8800914b3c98
[  2             000000000000002 R09: 0000000000000000
[  1             000000000000001 R12: ffff88008f686068
[  1             fff88008f6b3b90 R15: ffff88008f686000
[  2             S:ffff88011ed80000(0000)
knlG
[  1             CR0: 0000000080050033
[  2             000000001b31000 CR4: 00000000000407e0
[  1
[  1             586078 ffff8800916fbba8 ffff88008f686000
[  1             000000 0000000000000000 ffff88008f686000
[  1             11ba5d 00007fb885918000 ffff88008edf3000
...
```

Oops counter, followed by state of selected important kernel config options:

PREEMPT
SMP
DEBUG_PAGEALLOC
KASAN

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm:  ava Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD B   NA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b7     i: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffff      6630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492240] RSP: 0018:fff      68  EFLAGS: 00010286
[  2      ff88008f6b3b90 RCX: ffff8800a3b3cf30
[         000000000001 RDI: ffff8800914b3c98
[         000000000002 R09: 0000000000000000
[         000000000001 R12: ffff88008f686068
[         88008f6b3b90 R15: ffff88008f686000
[         ffff88011ed80000(0000)
kn
[                    : 0000000080050033
[         000001b31000 CR4: 00000000000407e0
[
[         078 ffff8800916fbba8 ffff88008f686000
[         000 0000000000000000 ffff88008f686000
[         a5d 00007fb885918000 ffff88008edf3000
..
```

Mostly useful when it is known which drivers are built as modules (e.g. with standard distro kernel configs).

May also contain module taint flags:
- P – proprietary
- O – out-of-tree
- F – force-loaded
- C – staging
- E – unsigned (suse)
- X – external support (suse) / unsigned
- N – no support (suse)
- +/- – being loaded/unloaded

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff88[illegible]3b3c840 ti: [illegible]16f8000
[  266.492191] RIP: 0010:[<ff[illegible]fff81126[illegible]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff88[illegible]
[  266.492285] RAX: ffff88008f6b[illegible]
[  266.492331] RDX: ffff8800914b3c[illegible]
[  266.492376] RBP: ffff8800916fbba[illegible]
[  266.492421] R10: 0000000000000008
[  266.492465] R13: ffff8800914b3c98 R[illegible]
[  266.492513] FS:  00007fb8966f6700(00[illegible]
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0[illegible]
[  266.492601] CR2: 00007f50fa190770 CR[illegible]
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88[illegible]000
[  266.492714]  ffff8800916fbc08 000000[illegible]000
[  266.492764]  ffff8800916fbbf8 ffffff[illegible]000
...
```

Information about CPU, process, kernel version, hardware.

Taint flags:
POFCEXN – same as per-module
R – module was force-unloaded
M – system has reported a MCE
B – bad page was encountered
U – userspace-defined
D – there was an oops before
W – there was a warning before
A – ACPI table was overriden
I – firmware bug workaround
L – soft-lockup has occurred before
K – kernel has been live patched
S – SMP kernel on UP machine

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff8800916fbb68  EFLA       10286
[  266.492285] RAX: ffff88008f6b3ba0 RBX: ffff88       RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8800914b3c98 RSI:
[  266.492376] RBP: ffff8800916fbba8 R08
[  266.492421] R10: 0000000000000008 R11
[  266.492465] R13: ffff8800914b3c98 R14
[  266.492513] FS:  00007fb8966f6700(000
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 0000000001b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Information about task that's supposed to be currently running, and whose stack we are actually running on.

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff      0916fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff8800       0 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8800       RSI: 0000000000000001 RDI: ffff8800914b3c98
[  266.492376] RBP: ffff                                     0000000000000000
[  266.492421] R10:                                          ff88008f686068
[  266.492465] R13:                                          ff88008f686000
[  266.492513] FS:                                           0)
knlGS:0000000000000
[  266.492566] CS:
[  266.492601] CR2:                                          000000000407e0
[  266.492652] Stac
[  266.492665]  0000                                         ffff88008f686000
[  266.492714]  ffff                                         ffff88008f686000
[  266.492764]  ffff8800                                     ffff88008edf3000
...
```

Which instruction was executing, translated to function name + offset.

This may be different from where position where BUG_ON( ) was reported, if the Function containing BUG_ON was inlined.

Example kernel oop

Values for the rest of general registers at the trapping instruction. Some are clearly kernel addresses. Some may hold the bad value of anon_vma->degree. Maybe RSI, R08, R10 or R11?

```
[  266.491864] ------------[ cut here ]--
[  266.491904] kernel BUG at mm/rmap.c:39
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu                cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tain                .0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina,                 L3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff880091                000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffff               fff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff8800916fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff88008f6b3ba0 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8800914b3c98 RSI: 0000000000000001 RDI: ffff8800914b3c98
[  266.492376] RBP: ffff8800916fbba8 R08: 0000000000000002 R09: 0000000000000000
[  266.492421] R10: 0000000000000008 R11: 0000000000000001 R12: ffff88008f686068
[  266.492465] R13: ffff8800914b3c98 R14: ffff88008f6b3b90 R15: ffff88008f686000
[  266.492513] FS:  00007fb8966f6700(0000) GS:ffff88011ed80000(0000)
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 0000000001b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Mod                                    fbfillrect cfbimgblt
cfbcopyarea drm_
[  266.492043] C                                      -kfd+ #24
[  266.492087] H                                      Weekly_13_11_2
11/20/2013
[  266.492141] t                                      i: ffff8800916f8000
[  266.492191] R                                      >]
unlink_anon_vmas
[  266.492249] RSP
[  266.492285] RAX: ffff                  RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8                 8 RSI: 0000000000000001 RDI: ffff8800914b3c98
[  266.492376] RBP: ffff8                 ba8 R08: 0000000000000002 R09: 0000000000000000
[  266.492421] R10: 000000                008 R11: 0000000000000001 R12: ffff88008f686068
[  266.492465] R13: ffff88                3c98 R14: ffff88008f6b3b90 R15: ffff88008f686000
[  266.492513] FS:  00007fb               f6700(0000) GS:ffff88011ed80000(0000)
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 0000000001b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Selected control registers:

CR2: the faulting virtual address
CR3: phys. addr of top level page table
CR4: a mask for enabling various extensions

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff8800916fb
[  266.492285] RAX: ffff88008f6b3ba0 RB                                  cf30
[  266.492331] RDX: ffff8800914b3c98 RS                                  3c98
[  266.492376] RBP: ffff8800916fbba8 R08: 000                0002 R09: 0000000000000000
[  266.492421] R10: 0000000000000008 R11: 0000               0001 R12: ffff88008f686068
[  266.492465] R13: ffff8800914b3c98 R14: ffff8              b3b90 R15: ffff88008f686000
[  266.492513] FS:  00007fb8966f6700(0000) GS:ff             11ed80000(0000)
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0            000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 00000000           b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Raw contents of top of the stack starting at RSP

Example kernel oops

Backtrace reconstructed by unwinding the stack, showing the return addresses from individual call frames

"?" means a pointer to function is on stack but doesn't fit in the stack frame; could be leftover from previous execution, or heuristics failure

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111ba5d>] free_pgtables+0x8e/0xcc
[  266.492873]  [<ffffffff8112253e>] exit_mmap+0x84/0x116
[  266.492907]  [<ffffffff8103f789>] mmput+0x52/0xe9
[  266.492940]  [<ffffffff81043918>] do_exit+0x3cd/0x9c9
[  266.492975]  [<ffffffff8170c1ec>] ? _raw_spin_unlock_irq+0x2d/0x32
[  266.493016]  [<ffffffff81044d7f>] do_group_exit+0x4c/0xc9
[  266.493051]  [<ffffffff8104eb87>] get_signal+0x58f/0x5bc
[  266.493090]  [<ffffffff810022c4>] do_signal+0x28/0x5b1
[  266.493123]  [<ffffffff8170ca0c>] ? sysret_signal+0x5/0x43
[  266.493162]  [<ffffffff81002882>] do_notify_resume+0x35/0x68
[  266.493200]  [<ffffffff8170cc7f>] int_signal+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48 8b 7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

Example kernel oops

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111
[  266.492873]  [<ffffffff8112
[  266.492907]  [<ffffffff810
[  266.492940]  [<ffffffff81
[  266.492975]  [<ffffffff
[  266.493016]  [<fffffff
[  266.493051]  [<ffffff
[  266.493090]  [<ffff
[  266.493123]  [<ff
[  266.493162]  [<f
[  266.493200]  [<ffffffff8170cc7f>] int_signal+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48 8b 7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

A bunch of instructions around the RIP.
RIP position denoted by < >

Recall that 0F 0B is opcode for UD2

We can disassemble the code listing by piping the oops into `./scripts/decodecode` in the kernel source tree.

Example decodecode output

```
~/linux.git> ./scripts/decodecode < oops-example.txt

[ 266.493235] Code: e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83 ec 10 48 8d 43 10 48 39 45 c8 74 55
48 8b 7b 08 83 bf 8c 00 00 00 00 74 02 <0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
All code
========
   0:   e8 03 b7 f4 ff          callq  0xfffffffffff4b708
   5:   49 8b 47 78             mov    0x78(%r15),%rax
   9:   4c 8b 20                mov    (%rax),%r12
   c:   48 8d 58 f0             lea    -0x10(%rax),%rbx
  10:   49 83 ec 10             sub    $0x10,%r12
  14:   48 8d 43 10             lea    0x10(%rbx),%rax
  18:   48 39 45 c8             cmp    %rax,-0x38(%rbp)
  1c:   74 55                   je     0x73
  1e:   48 8b 7b 08             mov    0x8(%rbx),%rdi
  22:   83 bf 8c 00 00 00 00    cmpl   $0x0,0x8c(%rdi)
  29:   74 02                   je     0x2d
  2b:*  0f 0b                   ud2              <-- trapping instruction
  2d:   e8 a4 fd ff ff          callq  0xffffffffffffdd6
  32:   48 8b 43 18             mov    0x18(%rbx),%rax
  36:   48 8b 53 10             mov    0x10(%rbx),%rdx
  3a:   48 89 df                mov    %rbx,%rdi
  3d:   48                      rex.W
  3e:   89                      .byte 0x89
  3f:   42                      rex.X

Code starting with the faulting instruction
===========================================
   0:   0f 0b                   ud2
   2:   e8 a4 fd ff ff          callq  0xffffffffffffdab
   7:   48 8b 43 18             mov    0x18(%rbx),%rax
   b:   48 8b 53 10             mov    0x10(%rbx),%rdx
   f:   48 89 df                mov    %rbx,%rdi
  12:   48                      rex.W
  13:   89                      .byte 0x89
  14:   42                      rex.X
```

Example decodecode output

```
All code
========
   0:   e8 03 b7 f4 ff          callq  0xfffffffffff4b708
   5:   49 8b 47 78             mov    0x78(%r15),%rax
   9:   4c 8b 20                mov    (%rax),%r12
   c:   48 8d 58 f0             lea    -0x10(%rax),%rbx
  10:   49 83 ec 10             sub    $0x10,%r12
  14:   48 8d 43 10             lea    0x10(%rbx),%rax
  18:   48 39 45 c8             cmp    %rax,-0x38(%rbp)
  1c:   74 55                   je     0x73
  1e:   48 8b 7b 08             mov    0x8(%rbx),%rdi
  22:   83 bf 8c 00 00 00 00    cmpl   $0x0,0x8c(%rdi)
  29:   74 02                   je     0x2d
  2b:*  0f 0b                   ud2              <-- trapping instruction
  2d:   e8 a4 fd ff ff          callq  0xfffffffffffffdd6
  32:   48 8b 43 18             mov    0x18(%rbx),%rax
  36:   48 8b 53 10             mov    0x10(%rbx),%rdx
  3a:   48 89 df                mov    %rbx,%rdi
  3d:   48                      rex.W
  3e:   89                      .byte 0x89
  3f:   42                      rex.X
```

Examp…ut

```
All code
========
   0:   e…                                  …f4b708
   5:   4…                                  …rax
   9:   4…
   c:   4…                                  …%rbx
  10:   4…
  14:   48 …                                …,%rax
  18:   48 39 45 c8                         cm…,-0x38(%rbp)
  1c:   74 55                               je…    0x73
  1e:   48 8b 7b 08                         mo…    0x8(%rbx),%rdi
  22:   83 bf 8c 00 00 00 00                cmpl   $0x0,0x8c(%rdi)
  29:   74 02                               je     0x2d
  2b:*  0f 0b                               ud2              <-- trapping instruction
  2d:   e8 a4 fd ff ff                      callq  0xfffffffffffffdd6
  32:   48 8b 43 18                         mov    0x18(%rbx),%rax
  36:   48 8b 53 10                         mov    0x10(%rbx),%rdx
  3a:   48 89 df                            mov    %rbx,%rdi
  3d:   48                                  rex.W
  3e:   89                                  .byte 0x89
  3f:   42                                  rex.X
```

BUG_ON(anon_vma->degree);

We skip the UD2 instruction if `0x8c(%rdi)` equals zero → we trap if the value is non-zero

This suggests that RDI holds the `struct anon_vma` pointer and degree is at offset `0x8c`

However, we can't determine the value that had been compared to zero in this case...

Example decodecode output

```
All code
========
   0:   e8 03 b7 f4 ff
   5:   49 8b 47 78
   9:   4c 8b 20
   c:   48 8d 58 f0
  10:   49 83 ec 10
  14:   48 8d 43 10
  18:   48 39 45 c8
  1c:   74 55
  1e:   48 8b 7b 08
  22:   83 bf 8c 00 00 00 00
  29:   74 02
  2b:*  0f 0b
  2d:   e8 a4 fd ff ff
  32:   48 8b 43 18
  36:   48 8b 53 10
  3a:   48 89 df
  3d:   48
  3e:   89
  3f:   42
```

```
su                12
lea               rbx),%rax
cmp               ,-0x38(%rbp)
je                73
mov     0x8(%rbx),%rdi
cmpl    $0x0,0x8c(%rdi)
je      0x2d
ud2                     <-- trapping instruction
callq   0xffffffffffffdd6
mov     0x18(%rbx),%rax
mov     0x10(%rbx),%rdx
mov     %rbx,%rdi
rex.W
.byte 0x89
rex.X
```

```
struct anon_vma_chain *avc;
list_for_each_entry_safe(avc, ...)
    struct anon_vma *anon_vma = avc->anon_vma;
    BUG_ON(anon_vma->degree);
```

Suggests that RBX holds the `struct anon_vma_chain` pointer `avc` and `anon_vma` member is at offset `0x8`

Verifying structure offsets

- We can use pahole from dwarves package
  - May depend on GCC version, .config options
    - rwsem size depends on CONFIG_DEBUG_SPINLOCK, CONFIG_DEBUG_LOCK_ALLOC

```
> pahole --hex -C anon_vma mm/vmscan.o
struct anon_vma {
        struct anon_vma *          root;                 /*     0   0x8 */
        struct rw_semaphore        rwsem;                /*   0x8  0x80 */
        /* --- cacheline 2 boundary (128 bytes) was 8 bytes ago --- */
        atomic_t                   refcount;             /*  0x88   0x4 */
        unsigned int               degree;               /*  0x8c   0x4 */
        struct anon_vma *          parent;               /*  0x90   0x8 */
        struct rb_root             rb_root;              /*  0x98   0x8 */

        /* size: 160, cachelines: 3, members: 6 */
        /* last cacheline: 32 bytes */
};
```

Example kernel oops

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111ba5d>] free_pgtables+0x8e/0xcc
[  266.492873]  [<ffffffff8112253e>] exit_mmap+0x84/0x116
[  266.492907]  [<ffffffff8103f789>] mmput+0x52/0xe9
[  266.492940]  [<ffffffff81043918>] do_exit+0x3cd/0x9c9
[  266.492975]  [<ffffffff8170c1
[  266.493016]  [<ffffffff81044
[  266.493051]  [<ffffffff8104e
[  266.493090]  [<ffffffff81002
[  266.493123]  [<ffffffff8170c
[  266.493162]  [<ffffffff8100288
[  266.493200]  [<ffffffff8170cc7f>] i                1+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49                78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48                7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8                53 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

The most important registers again, with higher `printk` level, or in case the details had scrolled away.

Example kernel oops

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111ba5d>] free_pgtables+0x8e/0xcc
[  266.492873]  [<ffffffff8112253e>] exit_mmap+0x84/0x116
[  266.492907]  [<ffffffff8103f789>] mmput+0x52/0xe9
[  266.492940]  [<ffffffff81043918>] do_exit+0x3cd/0x9c9
[  266.492975]  [<ffffffff817
[  266.493016]  [<ffffffff81
[  266.493051]  [<ffffffff81
[  266.493090]  [<ffffffff810
[  266.493123]  [<ffffffff8170ca0c>]                    gnal+0x5/0x43
[  266.493162]  [<ffffffff81002882>]                    resume+0x35/0x68
[  266.493200]  [<ffffffff8170cc7f>]                    1+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49                  78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48                  08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8                 3 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>]                unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

Randomization to distinguish reports of same bug instance from separate instances.

Example kernel oops

The task was already exiting when it oopsed. In this case it's clearly graceful exit (from the backtrace), but it could be exiting due to previous oops. It's safer to leave task as zombie than to risk infinite oopses in the exit path.

```
...
[
[                                                                    e/0xcc
[                                                                    x116
[  266.492
[  266.49294                 ff81043918>] do_exit+0x3cd/0x9c9
[  266.49297                 fff8170c1ec>] ? _raw_spin_unlock_irq+0x2d/0x32
[  266.493016                fff81044d7f>] do_group_exit+0x4c/0xc9
[  266.493051                ffff8104eb87>] get_signal+0x58f/0x5bc
[  266.493090                fffff810022c4>] do_signal+0x28/0x5b1
[  266.493123]               fffff8170ca0c>] ? sysret_signal+0x5/0x43
[  266.493162]               ffffff81002882>] do_notify_resume+0x35/0x68
[  266.493200]               ffffff8170cc7f>] int_signal+0x12/0x17
[  266.493235]          : e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43    48 39 45 c8 74 55 48 8b 7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fc    f ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
[  266.493404] R     [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

How is stack unwinding implemented?

- Start at value of RSP and increment in a loop
  - Check if stack contains kernel text address
  - Print with translation to function name+offset
  - When RSP matches RBP `+ sizeof(long)`, consider address *reliable* (i.e. without "?") and update RBP from the address it points to
- Not fully reliable, even with frame pointers
  - Cannot be relied upon functionally (live patching?)
  - Assembler functions now audited for missing frame pointers
    - Planned: runtime checks + DWARF validations

Stack Trace Example

```
int c(int i) { return i; }

int b(int i) { return c(i); }

int a(int i) { return b(i); }

int main(int argc, char *argv[]) {
    return a(argc);
}
```

- Compile using `gcc -O1 -m64` for AMD64
- Disassemble and single-step `main()` and `a()`
- Observe the stack

AMD64 Stack and Code Example

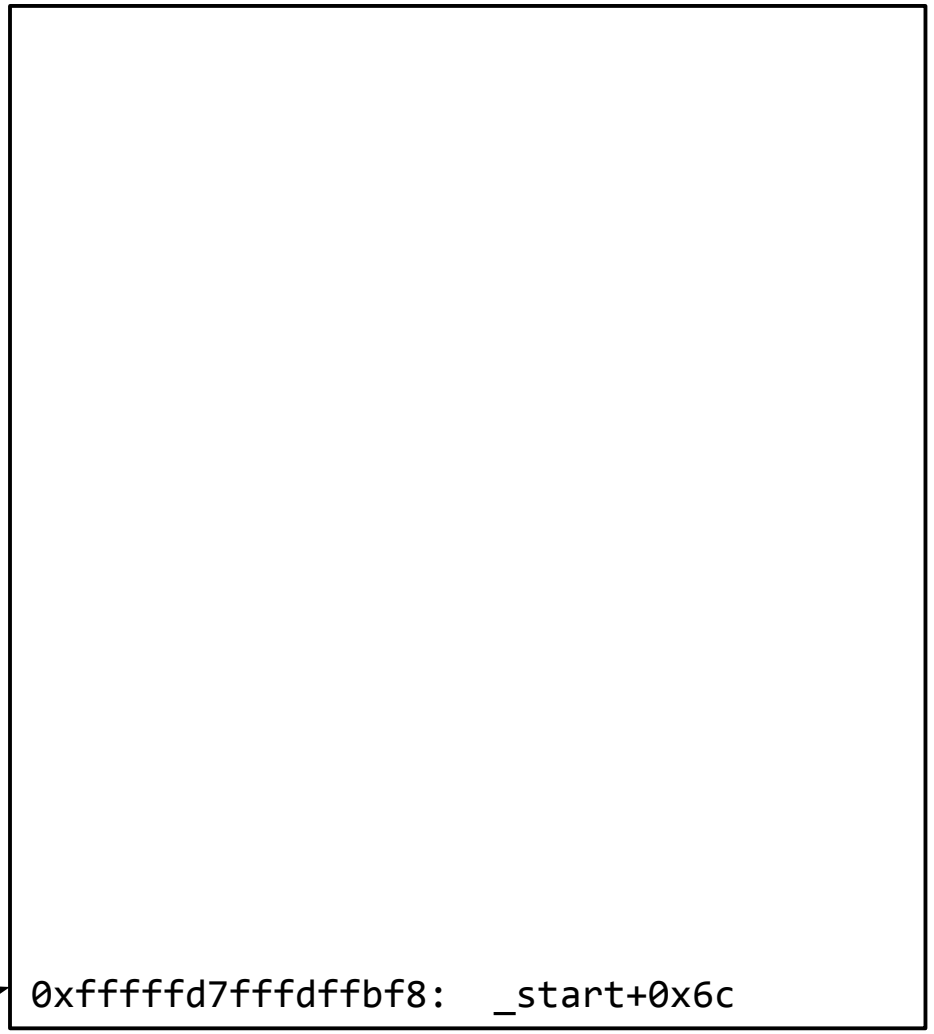
```
a:          pushq  %rbp
a+1:        movq   %rsp,%rbp
a+4:        movl   $0x0,%eax
a+9:        call   +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

```
main:       pushq  %rbp
main+1:     movq   %rsp,%rbp
main+4:     call   -0x2c    <a>
main+9:     leave
main+0xa:   ret
```

AMD64 Stack and Code Example

- **Initial state**
  - No instructions executed
  - Inherited stack pointer from `main()`'s caller

```
main:       pushq  %rbp
main+1:     movq   %rsp,%rbp
main+4:     call   -0x2c     <a>
main+9:     leave
main+0xa:   ret
```

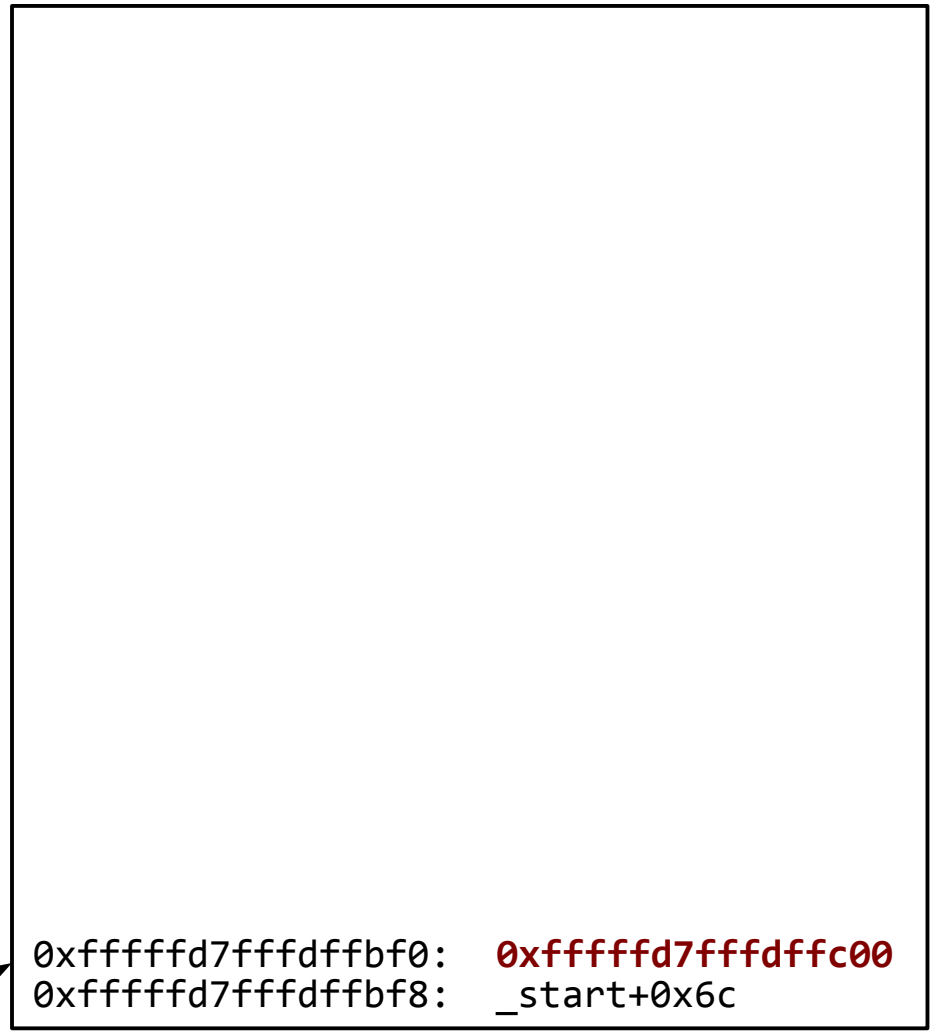


rsp=0xffffffd7fffdffbf8
rbp=0xffffffd7fffdffc00

AMD64 Stack and Code Example

- **Save previous frame pointer on the stack**

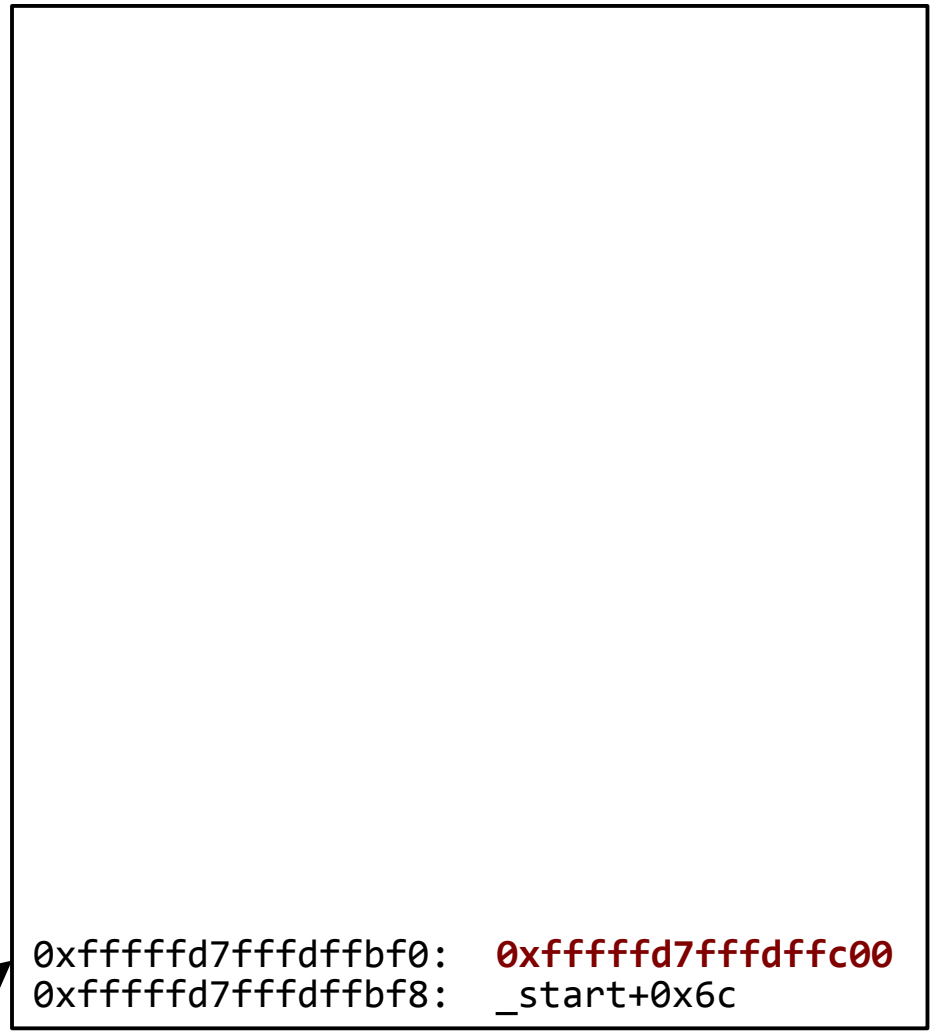
```
main:      pushq  %rbp
main+1:    movq   %rsp,%rbp
main+4:    call   -0x2c     <a>
main+9:    leave
main+0xa:  ret
```



AMD64 Stack and Code Example

- **Establish new fixed frame pointer in RBP**
  - It points to where we saved the previous one

```
main:       pushq  %rbp
main+1:     movq   %rsp,%rbp
main+4:     call   -0x2c     <a>
main+9:     leave
main+0xa:   ret
```



rsp=0xffffffd7fffdffbf0
rbp=0xffffffd7fffdffbf0

AMD64 Stack and Code Example

- **Call a()**
  - The argument is passed in RDI

```
main:       pushq  %rbp
main+1:     movq   %rsp,%rbp
main+4:     call   -0x2c    <a>
main+9:     leave
main+0xa:   ret
```

rsp=0xffffffd7fffdffbe8
rbp=0xffffffd7fffdffbf0

AMD64 Stack and Code Example

```
a:        pushq  %rbp
a+1:      movq   %rsp,%rbp
a+4:      movl   $0x0,%eax
a+9:      call   +0x2    <b>
a+0xe:    leave
a+0xf:    ret
```

- **Save the previous frame pointer to the stack**

AMD64 Stack and Code Example

```
a:          pushq   %rbp
a+1:        movq    %rsp,%rbp
a+4:        movl    $0x0,%eax
a+9:        call    +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

- **Establish new frame pointer in RBP**
  - It points to the address where the previous one is stored

AMD64 Stack and Code Example

```
a:          pushq   %rbp
a+1:        movq    %rsp,%rbp
a+4:        movl    $0x0,%eax
a+9:        call    +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

- **Zero EAX**
  - Zero-extend to upper 32bits of RAX
    - Clears the whole RAX
  - Not needed

AMD64 Stack and Code Example

```
a:          pushq   %rbp
a+1:        movq    %rsp,%rbp
a+4:        movl    $0x0,%eax
a+9:        call    +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

- **Call b()**
  - The argument is still in RDI

rsp=0xffffffd7fffdffbd8
rbp=0xffffffd7fffdffbe0

AMD64 Stack and Code Example

```
a:          pushq   %rbp
a+1:        movq    %rsp,%rbp
a+4:        movl    $0x0,%eax
a+9:        call    +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

- **Stack Unwinding**
  - rbp is a head of a **linked list on the stack**
  - each list member has **function pointer** at 8byte offset

AMD64 Stack and Code Example

```
a:          pushq   %rbp
a+1:        movq    %rsp,%rbp
a+4:        movl    $0x0,%eax
a+9:        call    +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

- **Leaving a()**
- **leave** is equivalent to
  - mov rsp, rbp
  - pop rbp

rsp=0xffffffd7fffdffbd8
rbp=0xffffffd7fffdffbe0

AMD64 Stack and Code Example

```
a:          pushq   %rbp
a+1:        movq    %rsp,%rbp
a+4:        movl    $0x0,%eax
a+9:        call    +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

- Leaving a()
  - mov rsp, rbp
  - pop rbp

rsp=0xffffffd7fffdffbe0
rbp=0xffffffd7fffdffbe0

AMD64 Stack and Code Example

```
a:          pushq   %rbp
a+1:        movq    %rsp,%rbp
a+4:        movl    $0x0,%eax
a+9:        call    +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

- Leaving a()
  - mov rsp, rbp
  - pop rbp

AMD64 Stack and Code Example

```
a:          pushq   %rbp
a+1:        movq    %rsp,%rbp
a+4:        movl    $0x0,%eax
a+9:        call    +0x2    <b>
a+0xe:      leave
a+0xf:      ret
```

- **Leaving a()**
  - mov rsp, rbp
  - pop rbp
  - ret

rsp=0xffffffd7fffdffbf0
rbp=0xffffffd7fffdffbf0

How is stack unwinding implemented?

- For perf callgraph sampling, this would be slow
  - Therefore, fully rely on frame pointer walk there
- Alternative approach: use DWARF2 exception handler (EH) frame info
  - Patch in SUSE kernels, rejected upstream
  - Also not fully reliable, and more complex

What to do about the oops anyway?

- Fix it and become a kernel developer :)
  - Or report it, a good report is very welcome
    - You can get “Reported-by:” credit
- Guidelines are in ./REPORTING-BUGS
  - Try googling the contents of the report (the oops title and names of the functions…), chances are you are not the first one to see it
  - Check the kernel version – listed on kernel.org?
    - If not, try reproducing with one of those (i.e. latest)
    - On distro kernel? Reproduce with vanilla, or report to distro only

What to do about the oops anyway?

- Make sure the oops is whole (“cut here...”)
  - And that it's not only a secondary oops
- If possible, avoid tainting modules
- Identify which subsystem is likely responsible
  - Sending to LKML only is possible, but might get lost in the thousands of mails per day
  - Match functions/modules mentioned in backtrace to files, then ./scripts/get_maintainer.pl -f <file>
    - Look for maintainers and specific mailing lists

What else can produce oops/panic?

- BUG_ON seen in the example – hard assertion
- Memory paging related faults – check CR2!
  - "`BUG: unable to handle kernel paging request`"
    - "`... handle NULL pointer dereference`" (when bad_addr < PAGE_SIZE)
  - Corrupted page table
  - Kernel trying to execute NX-protected page
  - Kernel trying to execute userspace page (Intel SMEP)
  - Failed bounds check in kernel mode (Intel MPX feature)
  - General protection fault, unhandled double fault

Example: NULL pointer dereference

```
[  526.950444] BUG: unable to handle kernel NULL pointer dereference at 0000000000000048
[  526.950450] IP: [<ffffffff81137e81>] dequeue_hwpoisoned_huge_page+0x121/0x1c0
[  526.950457] PGD 135863067 PUD 1249a9067 PMD 0
[  526.950460] Oops: 0000 [#1] SMP
[  526.950464] CPU 0
[  526.950465] Modules linked in: (...)
[  526.950506] Supported: Yes
[  526.950507]
[  526.950509] Pid: 6226, comm: thugetlb_overco Tainted: G    B          3.0.101-63-
default #1 QEMU Standard PC (i440FX + PIIX, 1996)
[  526.950514] RIP: 0010:[<ffffffff81137e81>]  [<ffffffff81137e81>]
dequeue_hwpoisoned_huge_page+0x121/0x1c0
[  526.950518] RSP: 0018:ffff880136965e48  EFLAGS: 00010246
[  526.950520] RAX: 0000000000000040 RBX: 0000000000000000 RCX: 0000000000000009
[  526.950523] RDX: 0000000000000000 RSI: 0000000000000048 RDI: ffffffff81e5fe08
[  526.950525] RBP: ffffea0003f8c000 R08: 0000000000001000 R09: 0000000000013a78
[  526.950527] R10: 0000000000000002 R11: 0000000000000200 R12: 0020000002000000
[  526.950529] R13: 0000000000000000 R14: ffff880136965e68 R15: ffff880136965ef8
[  526.950532] FS:  00007fca3e257700(0000) GS:ffff88013fc00000(0000)
knlGS:0000000000000000
[  526.950535] CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
[  526.950537] CR2: 0000000000000048 CR3: 00000001369a3000 CR4: 00000000000406f0
[  526.950542] DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
[  526.950546] DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400
```

Example: NULL pointer dereference

```
[  526.950548] Process thugetlb_overco (pid: 6226, threadinfo ffff880136964000, task
ffff8801356f2440)
[  526.950551] Stack:
[  526.950552]  ffffea0003f8c000 0000000000122800 ffffea0003f8c000 ffffffff8115a3c2
[  526.950556]  ffff880136965e68 ffff880136965e68 ffffea0003f8c000 ffffea0003f8c000
[  526.950560]  0001600000000000 0000000000000001 0000000000000065 ffffffff8115a5be
[  526.950563] Call Trace:
[  526.950571]  [<ffffffff8115a3c2>] soft_offline_huge_page+0x132/0x240
[  526.950576]  [<ffffffff8115a5be>] soft_offline_page+0xee/0x330
[  526.950580]  [<ffffffff8111e23e>] madvise_hwpoison+0x8e/0x120
[  526.950584]  [<ffffffff8111e794>] sys_madvise+0x44/0x280
[  526.950589]  [<ffffffff8146f3f2>] system_call_fastpath+0x16/0x1b
[  526.950595]  [<00007fca3ddcd307>] 0x7fca3ddcd306
[  526.950597] Code: 48 d3 e0 48 39 f0 74 0e 48 81 c2 68 70 00 00 48 39 fa 72 e6 31 d2 48
8b 45 00 48 c1 e8 36 48 c1 e0 04 48 8d 44 02 40 48 8d 70 08
8>[  526.950611]  8b 40 08 48 8d 50 d8 48 39 c6 48 8b 4a 28 74 27 48 39 d5 74
[  526.950618] RIP  [<ffffffff81137e81>] dequeue_hwpoisoned_huge_page+0x121/0x1c0
[  526.950622]  RSP <ffff880136965e48>
[  526.950623] CR2: 0000000000000048
```

The relevant subsystem will be typically the files containing the executing function (RIP) and the more immediate callers on the stack. There might be exceptions! Think a list manipulating function being passed NULL.

What else can produce oops/panic?

- Soft lockup
  - CPU spent 20s in kernel without reaching a schedule point
  - A warning, unless config/bootparam `softlockup_panic` enabled
    - Soft lockup can be harmless, so not good idea in production
- Hard lockup
  - CPU spent 10s with disabled interrupts
- Detection of both combines several generic mechanisms
  - High priority kernel watchdog thread updates soft lockup timestamp
  - hrtimer set to deliver periodic interrupts, increments hard lockup counter and wakes up the watchdog thread
  - NMI perf event checks the timestamps and thus knows if hrtimers interrupts were processed and if watchdog thread was scheduled

Example: soft lockup (32bit x86)

```
kernel: NMI watchdog: BUG: soft lockup - CPU#1 stuck for 23s! [kworker/u8:2:31788]
kernel: Modules linked in:
kernel: CPU: 1 PID: 31788 Comm: kworker/u8:2 Not tainted 3.18.0-rc3 #7
kernel: Hardware name:                  /DG41MJ, BIOS MJG4110H.86A.0006.2009.1223.1155
12/23/2009
kernel: Workqueue: khelper __call_usermodehelper
kernel: task: e0f35580 ti: c0f12000 task.ti: c0f12000
kernel: EIP: 0060:[<c40b82bd>] EFLAGS: 00000206 CPU: 1
kernel: EIP is at __zone_watermark_ok+0xd/0xa0
kernel: EAX: c4bcb1c0 EBX: 00001aca ECX: 00000472 EDX: 00000001
kernel: ESI: 00000001 EDI: c4bcb1c0 EBP: c0f13c4c ESP: c0f13c40
kernel: DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
kernel: CR0: 8005003b CR2: 0834af00 CR3: 04c6f000 CR4: 00040790
kernel: Stack:
kernel: 00001aca 00000000 c0f13ccc c0f13c68 c40b9511 00000000 00000000 00001aca
kernel: 00037400 c4bcb1c0 c0f13ca8 c40ce958 00000000 00000000 00000000 c4bcb3ec
kernel: 00000000 00000004 00000000 e0f35580 c0f13fec c4bcb1c0 00037400 c4bcb1c0
```

Example: soft lockup (32bit x86)

```
kernel: Call Trace:
kernel: [<c40b9511>] zone_watermark_ok+0x21/0x30
kernel: [<c40ce958>] compact_zone+0x268/0x4d0
kernel: [<c40cef64>] try_to_compact_pages+0xd4/0x2[illegible]
kernel: [<c40bb211>] __alloc_pages_direct_compact+0x41/[illegible]170
kernel: [<c40bb686>] __alloc_pages_nodemask+0x346/0x790
kernel: [<c40bbb83>] alloc_kmem_pages_node+0x13/0x20
kernel: [<c4030982>] copy_process.part.53+0xb2/0x1410
kernel: [<c4042740>] ? ____call_usermodehelper+0x140/0x140
kernel: [<c4042740>] ? ____call_usermodehelper+0x140/0x140
kernel: [<c4031e65>] do_fork+0xb5/0x390
kernel: [<c406a42a>] ? __lock_acquire.isra.24+0x2ea/0xc10
kernel: [<c4042740>] ? ____call_usermodehelper+0x140/0x140
kernel: [<c403215a>] kernel_thread+0x1a/0x20
kernel: [<c40425c2>] __call_usermodehelper+0x22/0x60
kernel: [<c404546f>] process_one_work+0x15f/0x390
kernel: [<c4045419>] ? process_one_work+0x109/0x390
kernel: [<c4045ae9>] worker_thread+0x39/0x440
kernel: [<c4045ab0>] ? cancel_delayed_work+0x70/0x70
kernel: [<c4049f4e>] kthread+0xae/0xd0
kernel: [<c4040000>] ? SyS_getpriority+0xe0/0x270
kernel: [<c48303c1>] ret_from_kernel_thread+0x21/0x30
kernel: [<c4049ea0>] ? kthread_create_on_node+0x170/0x170
kernel: Code: 8d 0c db 8d 50 34 8d 44 88 10 8d 74 26 00 c7 02 01 00 00 00 83 c2 24 39 c2
75 f3 5b 5d c3 55 89 e5 57 56 be 01 00 00 00 53 89 c7 <89> cb 89 f0 89 d1 d3 e0 29 c6 03
75 10 f6 45 0c 20 74 0b 89 d9
```

The problem was an infinite loop in this function.

Example: NULL pointer on SPARC

```
 Unable to handle kernel NULL pointer dereference
 tsk->{mm,active_mm}->context = 0000000000000000
 tsk->{mm,active_mm}->pgd = ffffff80000402000
                \|/ ____ \|/
                "@'/ .. \`@"
                /_| \__/ |_\
                   \__U_/
 swapper(0): Oops [#1]
 CPU: 0 PID: 0 Comm: swapper Not tainted 3.16.0-4-sparc64 #1 Debian 3.16.7-2
 task: 0000000000a14470 ti: 00000000009fc000 task.ti: 00000000009fc000
 TSTATE: 0000004480e01601 TPC: 00000000004a36fc TNPC: 00000000004a3700 Y: 00000000    Not
tainted
 TPC: <kstat_incr_irq_this_cpu+0x1c/0x60>
 g0: 0000000000a161a0 g1: 0000000000a2eaf8 g2: 0000000000000001 g3: 0000000000000000
 g4: 0000000000a14470 g5: 0000000000000000 g6: 00000000009fc000 g7: 0000000000000001
 o0: 0000000000000000 o1: 0000000000000000 o2: 0000000000000000 o3: 0000000000000020
 o4: 000000000000000e o5: 0000000000a8f000 sp: 00000000009fef31 ret_pc: 00000000004a36f4
 RPC: <kstat_incr_irq_this_cpu+0x14/0x60>
 l0: 0000000000b30320 l1: 0000000000b0f650 l2: 0000000000000001 l3: 0000000000b30448
 l4: 0000000000000000 l5: 0000000000b10b18 l6: 0000000000b30300 l7: 0000000000000000
 i0: 0000000000000000 i1: 0000000000b0f648 i2: 0000000000b30310 i3: 000000000000000e
 i4: 0000000000b30000 i5: 0000000000b30000 i6: 00000000009fefe1 i7: 00000000008aaf58
 I7: <timer_interrupt+0x38/0xa0>
```

Example: NULL pointer on SPARC

```
 Call Trace:
  [00000000008aaf58] timer_interrupt+0x38/0xa0
  [00000000004209d4] tl0_irq14+0x14/0x20
  [00000000004db6c0] touch_softlockup_watchdog+0x0/0x20
  [0000000000ab8568] memblock_virt_alloc_try_nid+0x84/0x94
  [0000000000ab9754] sparse_init+0x1c/0x224
  [0000000000aabcb8] paging_init+0xd80/0xeb0
  [0000000000aa7068] setup_arch+0x2d4/0x718
  [0000000000aa4668] start_kernel+0x78/0x420
  [0000000000aa6d84] start_early_boot+0x1bc/0x1cc
  [0000000000897410] tlb_fixup_done+0x4c/0x5c
  [0000000000000000]           (null)
 Disabling lock debugging due to kernel taint
 Caller[00000000008aaf58]: timer_interrupt+0x38/0xa0
 Caller[00000000004209d4]: tl0_irq14+0x14/0x20
 Caller[00000000008984e4]: panic+0x1c8/0x208
 Caller[0000000000ab8568]: memblock_virt_alloc_try_nid+0x84/0x94
 Caller[0000000000ab9754]: sparse_init+0x1c/0x224
 Caller[0000000000aabcb8]: paging_init+0xd80/0xeb0
 Caller[0000000000aa7068]: setup_arch+0x2d4/0x718
 Caller[0000000000aa4668]: start_kernel+0x78/0x420
 Caller[0000000000aa6d84]: start_early_boot+0x1bc/0x1cc
 Caller[0000000000897410]: tlb_fixup_done+0x4c/0x5c
 Caller[0000000000000000]:           (null)
 Instruction DUMP: 92100018  40076593  901222f8 <c45a2048> 030028b1  c6008000  8600e001
c6208000  c4586160
 ---[ end Kernel panic - not syncing: Aiee, killing interrupt handler!
```

What else can produce oops/panic?

- Hung task check
  - "`INFO: task … blocked for more than 120 seconds`"
  - `khungtaskd` periodically processes tasks in uninterruptible sleep and checks if their switch count changed
- RCU stall detector
  - Detects when RCU grace period is too long (21s)
    - CPU looping in RCU critical section or disabled interrupts, preemption or bottom halves, no scheduling points in non-preempt kernels
    - RT task preempting non-RT task in RCU critical section
- Several other related to debugging config options

Kernel debugging config options

- Kernel can be built with additional debugging options enabled
  - Extra checks that can catch errors sooner, or provide extra information, at the cost of CPU and/or memory overhead
  - Can also hide errors such as race conditions...
- Many of them under “Kernel hacking” in make menuconfig
  - Others placed in the given subsystem/driver

Kernel debugging config options (VM)

- DEBUG_VM – enable VM_BUG_ON(cond) checks
- PAGE_OWNER – track who allocated which pages in order to find a memory leak
- DEBUG_PAGEALLOC – unmap (or poison) pages after they are freed
- DEBUG_SLAB – detect some cases of double free, or use-after-free (by poisoning)
  - SLUB_DEBUG variant can enable/disable debugging in runtime
- DEBUG_KMEMLEAK – detect leaks with a conservative garbage collection based algorithm
- KASAN – Find out of bounds accesses and use-after-free bugs at the cost of 1/8 memory and 3x slower performance

Kernel debugging config options

- DEBUG_STACKOVERFLOW – check if random corruption involving struct thread_info is caused by too deep call chains
- DEBUG_SPINLOCK and others for different locks – catch missing init, freeing of live locks, some deadlocks
- LOCK_STAT – for lock contention, perf lock
- PROVE_LOCKING - “lockdep” mechanism for online proving that deadlocks cannot happen and report that deadlock can occur before it actually does

Thank you!

- Crash dump analysis course
  - http://d3s.mff.cuni.cz/cda
- We are hiring!
  - https://www.suse.com/careers/