Agenda – Debugging Scenarios

• Debugging production kernels
  – Post-mortem analysis: interpreting kernel oops/panic output, creating and analyzing kernel crash dumps
  – Kernel observability – dynamic debug, tracing, alt-sysrq dumps, live crash session
• Debugging during individual kernel development
  – Debug prints – printk() facility
  – Debugger (gdb) support
• Finding (latent) bugs during collaborative development
  – Optional runtime checks configurable during build
  – Testing and fuzzing
  – Static analysis
Enterprise Linux Distro and Bugs (incl. Kernel)

• The software installation (e.g. ISO) itself is free (and open source, obviously)
• Customers pay for **support subscription**
  – Delivery of (tested by QA) package updates – fixing known bugs, CVE’s… but not more!
  – **Getting reported bugs fixed**
    • Bugs specific to customer’s workload, hardware, “luck”, large number of machines…
    • Upstream will also fix reported bugs, but only with latest kernel and no effort guarantees
• Dealing with kernel bugs, e.g. crashes
  – Find out the root cause (buggy code) with limited possibilities (compared to local development)
    • Typically no direct access to customer’s system or workload
    • Long turnarounds for providing a modified debug kernel and reproducing the bug
  – Write and deliver a fix (upstream first!) or workaround; fix goes to next update
    • Possibly a livepatch in some cases
  – Is a lot of fun ;-)
Kernel Oops - The Real World Example

• In September 2017, a customer reported a kernel **Oops**
  – Kernel detects an unexpected situation and reports it on the console(s)
    • Usually triggered by a CPU exception
    • The exception might be also triggered by an assertion in kernel code
  – Lots of (architecture-specific) information which may or may not be enough to find the root cause

• Kernel might survive an Oops and keep running
  – Kill just a single process, but that includes kernel threads
  – Possibly inconsistent state (locks that were locked are not unlocked…)

• Less serious kind of Oops: kernel **warning**, doesn’t kill, just taints the kernel with **W** flag
  – Usually an assert-like condition for “this should not happen but should be able recover” situations

• Fatal oops (kernel **panic**) kills the system completely
  – Oops in interrupt context, panic_on_oops, panic_on_taint enabled, specific panic() calls
  – HW failure, critical memory allocation failure, init or idle task killed
  – May trigger crash dump if configured, or reboot automatically after set delay
Kernel Oops - The Real World Example

------------[ cut here ]------------
kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nls_utf8 cifs(X) nfs fs cache nf sd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vs ock(EX)
[...]
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>] [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffffffffffffe RBX: ffffffffffffffff RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: fff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffffffff80008ceca8c R14: ffffffff80423c3740 R15: 0000000000000001
FS: 00007f2945893760(0000) GS:fffffff8fd0000(0000) k nlgS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000000
CR2: 00007f6941216960 CR3: 00000003f3bea00 CR4: 0000000000000000
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffffffff80211336000, task ffffffff801b49c8040)
Kernel Oops - The Real World Example

Stack:
ffff88016d2319e0  ffff880158f0a140  00007ffce0bf0000  fffffff81127fa4
ffff88008cecac80  ffff880211337dd8  ffff88016d2319e0  fffffff811280f8
0000000000000001  ffff88014c826088  ffff88008cecac80  00000000100000028

Call Trace:
[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP  <fff880211337d88>
---[ end trace 3dad41c41965c82c ]---
Kernel Oops in Detail

--- cut here ---

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!

invalid opcode: 0000 [#1] SMP

CPU 1

Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb

md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsoc(EX)

[last unloaded: ppa]

Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual

Platform/440BX Desktop Reference Platform

RIP: 0010:[<ffffffff811e466e>]  [<ffffffff811e466e>] shm_close+0x3e/0xb0

RSP: 0018:ffff880211337d88  EFLAGS: 00010202

RAX: ffffffff81a46920 RBX: ffffffff81a46920 RCX: 0000000000000006

RDX: 0000000000000000 RSI: 0000000000000002 RDI: 0000000000000006

RBP: ffffffff81215a80 R10: ffffffff81215a80 R12: ffffffff81a469c0

R13: ffffffff810192cecc00 R14: ffffffff810192cecc00 R15: 0000000000000001

FS: 0000f2945893760(0000) GS:fffffff8043fd0000(0000) knlGS:0000000000000000

CS: 0010 DS: 0000 ES: 0000 CR0: 000000000005003b

CR2: 00007f6941216960 CR3: 0000000000000000 CR4: 0000000000000000

DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000

DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000

Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)
Kernel Oops in Detail – What Happened

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vssock(EX)
<...>
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>] [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffff811e466e RBX: ffffffff811e466e RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: fffffff880192cecc00 R11: ffffffff81215a80 R12: fffffff81a469c0
R13: ffff88008ce cac80 R14: ffffffff81a469c0 R15: 0000000000000001
FS: 00007f2945893760(0000) GS: ffffffff81a469c0 R10: 0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000008005003b
CR2: 00007f6941216960 CR3: 00000003f37e0a00 CR4: 000000000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000040
Process <redacted> (pid: 26341, threadinfo ffffffff8801b49c8040, task ffffffff8801b49c8040)
Result of a BUG() macro.

File + line translation enabled by CONFIG_DEBUG_BUGVERBOSE (implemented by __bug_table section on x86 - ~70-100kB)
Kernel Oops in Detail – What Happened

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1

Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G           E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform

RIP: 0010:[<ffffffff811e466e>]  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88  EFLAGS: 00010202
RAX: ffffffffffffffea RBX: ffffffffffffffea RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008cecac80 R14: ffff880423c33740 R15: 0000000000000001
FS:  00007f2945893760(0000) GS:ffff88043fd00000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 00007f6941216960 CR3: 00000000000001407e0
DR0: 000000000000000000000000000000 DR1: 000000000000000000000000000000 DR2: 0000000000000000000000000000000000000000
DR3: 00000000000000000000000000000000 DR6: 0000000000000000000000000000000000000000 DR7: 00000000000000000000000000000000
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)

-----[ cut here ]-----

Result of a BUG() macro.

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

The indicated line contains:
struct shmid_kernel *shp;
...
shp = shm_lock(ns, sfd->id);
BUG_ON(IS_ERR(shp));

This is essentially a hard assertion:
if (<condition>) BUG()
Kernel Oops in Detail – CPU Exception

------------[ cut here ]------------

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1

Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsock(EX)

[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>] [ffffffff811e466e] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffffffffffff RBX: ffffffffffffffff RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffffffff81215a80 R11: ffffffff81a469c0 R12: ffffffff81a469c0
R13: ffff88008ceca08 R14: ffff880423c3740 R15: 0000000000000001
FS: 00007f2945893760(0000) GS: ffffffff3fd00000(0000) kncGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000503b
CR2: 00007f6941216960 CR3: 000000003f3bea00 CR4: 00000000000000107e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000000000400000000000000000
Process <redacted> (pid: 26341, threadinfo ffffffff880211336000, task ffffffff8801b49c8040)
Kernel Oops in Detail – CPU Exception

invalid opcode: 0000 [#1] SMP

CPU 1

Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls nls_unicode(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autofs4 binfmt_misc mperf vsock(EX)

On x86_64, BUG( ) emits a standardized invalid opcode UD2 (0F 0B) triggering a CPU exception.

The exception handler checks for UD2 opcode and searches the __bug_table for details.

This reduces instruction cache footprint compared to BUG( ) being a call. Also prevents speculation into BUG( ) path.
On x86_64, BUG() emits a standardized invalid opcode UD2 (0F 0B) triggering a CPU exception.

The exception handler checks for UD2 opcode and searches the __bug_table for details.

This reduces instruction cache footprint compared to BUG() being a call. Also prevents speculation into BUG() path.

Since 4.11, the same trick is used for WARN(), WARN_ON() etc.

The UD0 opcode (0F FF) was used because some emulators terminate when they encounter UD2.

However turns out UD0 is not that well standardized (AMD vs Intel).
Kernel Oops in Detail – Error Code

---------[ cut here ]---------
kernell BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsock(EX)
<...>
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>] [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffff811e466ea RBX: ffffffff811e466ea RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffffffff81215a80 R11: ffffffff81a469c0
R12: ffffffff81a469c0 R13: ffffffff81a469c0 R14: ffffffff81215a80 R15: 0000000000000001
FS: 00007f2945893760(0000) GS:fffffff8193fd00000(0000) knlGS:0000000000000000
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)
Kernel Oops in Detail – Error Code

 kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
 invalid opcode: 0000 [#1] SMP
 CPU 1
 Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
 md4 nls_utf8 e2fsprogs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autofs4 binfmt_misc mperf vsock(EX)
 <...>
 [last unloaded: ppa]
 Supported: Yes, External
 Pid: 26341, comm: <redacted> Tainted: G           E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
 Plat
 RIP: 0010:[<ffffffff811e466e>]  [<ffffffff811e466e>] shm_close+0x3e/0xb0
 RSP: 0018:ffff880211337d88  EFLAGS: 00010202
 RAX: ffffffffffffffea RBX: ffffffffffffffea RCX: 0000000000000006
 RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
 RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
 R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
 R13: ffff88008cecac80 R14: ffff880423c33740 R15: 0000000000000001
 FS:  00007f2945893760(0000) GS:ffff88043fd00000(0000) knlGS:0000000000000000
 CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
 CR2: 00007f6941216960 CR3: 00000003f3bea000 CR4: 00000000001407e0
 DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
 DR3: 0000000000000000

Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)

---[ cut here ]---

x86- and exception-specific error code (32-bit hex number). Not applicable to invalid opcode. 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

(newer kernels decode this)
Kernel Oops in Detail – Counter and Config

------------[ cut here ]------------
kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsch(0)
<...
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G X 3.0.101-84-default #1 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff81e466e>] [<ffffffff81e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffff81e466e RBX: ffffffff81e466e RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81e466e R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffffffff81e466e R11: ffffffff81e466e R12: ffffffff81e466e
R13: ffffffff81e466e R14: ffffffff81e466e R15: ffffffff81e466e
FS: 00007f2945893760(0000) GS: ffffffff81e466e(0000) knlGS: 0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000003b
CR2: 0000007f2945893760 CR3: 00000000003f3bea00 CR4: 0000000000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffffffff81e466e, task ffffffff81e466e)
Kernel Oops in Detail – Counter and Config

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000  [#1] SMP
CPU 1

Modules linked in: lp lpport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls_utf8 cifs(X) umacache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsock(EX)
<...>
[last unloaded: ppa]
Supported: Yes, External
Pid: 26341, comm: <redacted> Tainted: G           E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform

RIP: 0010:[<ffffffff811e466e>]  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88  EFLAGS: 00010202
RAX: ffffffffffffffea RBX: ffffffffffffffea RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008cecac80 R14: ffff880423c33740 R15: 0000000000000001
FS:  00007f2945893760(0000) GS:ffff88043fd00000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000008005003b
CR2: 00007f6941216960 CR3: 00000003f3bea000 CR4: 00000000000000147e0
DR0: 0000000000000000000000000000000000 DR1: 0000000000000000000000000000000000
DR3: 0000000000000000000000000000000000 DR6: 000000000000000000000000000000400
DR7: 000000000000000000000000000000400
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)

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

PREEMPT
SMP
DEBUG_PAGEALLOC
KASAN
PTI/NOPTI
Kernel Oops in Detail – Kernel Modules

--------[ cut here ]--------
kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nlsUtf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsock(EX)

[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform
RIP: 0010: [<ffffffff811e466e>][<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018: fffffff880211337d88 EFLAGS: 00010202
RAX: ffffffffffffffff RBX: ffffffffffffffff RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: fffffffff811eaf20 R08: 0000000000000002 R09: ffff8804256a4d0
R10: ffffffff81215a80 R11: ffffffff81a46920 R12: ffffffff81a469c0
R13: ffff88008cecaec80 R14: ffffffff81215a80 R15: 0000000000000001
FS: 000007f2945893760(0000) GS: ffff88043fd00000(0000) knlGS: 0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 00009b0000000000
CR2: 00007f6941216960 CR3: 00000003f3bea000 CR4: 00000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)
Kernel Oops in Detail – Kernel Modules

invalid opcode: 0000 [#1] SMP
CPU 1

Modules linked in: lp parport ppp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls_utf8 cifs(X) nfs fs nfsd lockd nfs_acl auth_rpcgss sunrpc autofs4 binfmt_misc mperf vsock(EX)
<...>

[Last unloaded: ppa]

Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G           E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual

RIP: 0010:[<ffffffff811e466e>]  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88  EFLAGS: 00010202
RAX: ffffffffffffffea RBX: ffffffffffffffea RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008cecac80 R14: ffff88008cece9a0 R15: 0000000000000001
FS:  00007f2945893760(0000) GS:ffff88043fd00000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000000000000
CR2: 00007f6941216960 CR3: 000000003f3bea00 CR4: 0000000000000000
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000000400 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)

Modules (~drivers) and their taint flags:

E – unsigned
X – externally supported (SUSE)
P – proprietary
O – out-of-tree
F – force-loaded
C – staging/ tree module
N – no support (SUSE)
+/- – being loaded/unloaded

Last unloaded – insufficient cleanup?
Kernel Oops in Detail – Basic Info

-----------[ cut here ]-----------
kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nls_utf8 cifs(X) nfscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsock(EX)
...>
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010: [xffffffff811e466e] [fffffffff811e466e] shm_close+0x3e/0xb0
RSP: 0018: ffffffff811e466e EFLAGS: 00010202
RAX: ffffffff811e466e RBX: ffffffff811e466e RCX: 0000000000000006
RDX: 0000000000000000 RSI: 0000000000000000 RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffffffff81a46920
R10: ffffffff81a46920c R11: ffffffff81a46920 R12: ffffffff81a46920
R13: ffffffff81a46920 R14: ffffffff81a46920 R15: ffffffff81a46920
FS: 0000000000000000 GS: ffffffff81a469200000 kntLS: 0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000000 CR2: 0000000000000000
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffffffff880211336000, task ffffffff8802149c8040)
Kernel Oops in Detail – Basic Info

-----------[ cut here ]-----------
kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nls_utf8 cifs(X) nfs fscache lockd nfs_acl auth_rpcgss sunrpc autofs4 binfmt_misc mperf vsoc(Ex)
<...>
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform

Information about CPU, process in whose context the bug happened, exact kernel version, HW (or virtual host) platform.

RIP: 0010:0[fffffffff811e466e]  [0xfffffffff811e466e] shm_close+0x3e/0xb0
RSP: 0018:0ffff880211337d88  EFLAGS: 00010202
RAX: ffffffff81a46920 RBX: ffffffff81a46920 RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008cecac80 R14: ffff880423c33740 R15: 0000000000000001
FS: 00007f2945893760(0000) GS:ffff88043fd0000(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 00007f6941216960 CR3: 00000003f3bea000 CR4: 00000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400

Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)
Kernel Oops in Detail – Basic Info

---[ cut here ]---

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP

CPU 1

Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autosf4 binfmt_misc mperf vssock(EX)

[last unloaded: ppa]

Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G  E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform

RIP: 0010:[<ffffffff811e466e>]  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88  EFLAGS: 00010202
RAX: ffffffff811e466e RBX: ffffffff811e466e RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffffffff81215a80 R11: ffffffff81a469c0 R12: ffffffff81a469c0
R13: ffff88008cecac80 R14: ffffffff81215a80 R15: 0000000000000001
CR0: 000000008005003b CR2: 00007f6941216960 CR3: 00000003f3bea000
CR4: 00000000001407e0 CR8: 0000000000000000
CR7: 0000000000000000

Information about CPU, process in whose context the bug happened, exact kernel version, HW (or virtual host) platform.

More process (task) details (addresses of related structures) Not printed in newer kernels.
Kernel Oops in Detail – Kernel taint flags

------------[ cut here ]------------

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1

Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nls_utf8 cifs(X) nfs fscache nbsd lockd nfs_acl auth_rpcgss sunrpc autosfs4 binfmt_misc mperf vsock(EX)
[..]
[last unloaded: ppa]

Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>] [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffff8000000000000000 RDX: 0000000000000000
RBX: ffffffff81a46920 RSI: 000000000000005c RDI: 0000000000000006
RCX: 0000000000000000 RDI: ffff880423c33740 R12: ffffffff81a469c0
RBP: ffffffff81a469c0 R11: ffffffff81215a80 R10: ffffffff81215a80
R13: ffffffff80008ceca0 R14: ffffffff81e469c0 R15: 0000000000000000
FS: 00007f2945893760(0000) GS: ffffffff800000000000001
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000000 CR3: 0000000000000000
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffffffff880211336000, task ffffffff8801b49c8040)
Kernel Oops in Detail – Kernel taint flags

------------[ cut here ]------------
kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod md4 nls_utf8 cifs(X) nfs fscache nfSD lockd nfS_aop...
[...]
[last unloaded: ppa]
Supported: Yes, External

RIP: 0010:[<ffffffff811e466e>] [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffffffffffea RBX: ffffffffffffffea RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008cecac80 R14: ffff880423c33740 R15: 0000000000000001
FS: 00007f2945893760(0000) GS:ffff88043fd0000(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 00007f6941216960 CR3: 00000003f3bea000 CR4: 00000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000400
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)

Kernel taint flags:

POFCEX – same as per-module
G – no proprietary module (not P)
R – module was force-unloaded
D – there was an oops before
W – there was a warning before
L – soft-lockup has occurred before
B – bad page was encountered
K – kernel has been live patched
T – kernel structures randomized
M – system has reported a MCE
A – ACPI table was overriden
I – firmware bug workaround
S – “CPU out of spec”
X – distro-defined (auxiliary)
U – userspace-defined
Kernel Oops in Detail – instruction pointer

------------[ cut here ]------------

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1

Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nlsUtf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vssock(EX)
<...>
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G           E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform

RIP: 0010:<ffffffff811e466e> [ffffffff811e466e] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88  EFLAGS: 00010202
RAX: ffffffffffffffff RBX: ffffffffffffffff RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R10: ffffffff81215a80 R12: ffffffff81a469c0
R10: ffffffff818012cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffffffff8008eceac80 R14: ffffffff80423c3740 R15: 0000000000000001
FS: 00007f2945893760(0000) GS: ffffffff8043fd0000(0000) knlGS: 0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 000000000005003b
CR2: 00006f941216960 CR3: 00000003f3bea000 CR4: 0000000000000000
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffffffff80211336000, task ffffffff801b49c8040)
Which instruction was executing, translated to function name + offset / size.

May appear as different function from where BUG_ON() was reported, if the function containing BUG_ON() was inlined.

In newer kernel the raw value (address) was removed for security reasons (KASLR).
Kernel Oops in Detail – General Registers

Kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1

Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsock(EX)
...
[last unloaded: ppa]

Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>] [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffff811e466e RBX: ffffffff811e466e RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81215a80 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008ce cac0 R14: ffff880423c37340 R15: 0000000000000001
FS: 0000000000000000 GS: ffffffff8043fd00000(0000) knlGS: 0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000008005003b
CR2: 00007f6941216960 CR3: 00000003f3bea000 CR4: 0000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)
Kernel Oops in Detail

--- [cut here] ---

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nls_utf8 cifs(X) nfs fscache nfsd lockd mt aha ide_virt_recv loop
<...>
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G           E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>]<ffffffff811e466e> shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88  EFLAGS: 00010202
RAX: ffffffffffffffff RBX: ffffffffffffffff RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008ce cac 00 R14: ffff880423c33740 R15: 0000000000000001
FS: 00007f2945893760(0000) GS: ffff88043fd00000(0000) knlGS: 0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000006
CR2: 00007f6941216960 CR3: 00000003f3bea00 CR4: 0000000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)

Values of the general registers at the trapping instruction. We can recognize kernel addresses:

- FFFFFFFF8xxxxxxx – kernel code + data
- FFFFFFFFAxxxxxxx – kernel modules code + data
- FFF88xxxxxxxxxxxx – direct mapped phys. mem.
- FFFFFFFExxxxxxxxxxx – array of struct pages

RAX – small negative value, probably error code

```
shp = shm_lock(ns, sfd->id);
BUG_ON(IS_ERR(shp));
```

RAX probably result of `shm_lock()`
### Crash Reference Card (64-bit)

<table>
<thead>
<tr>
<th>Reg</th>
<th>Usage</th>
<th>Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAX</td>
<td>Return value</td>
<td>no</td>
</tr>
<tr>
<td>RBX</td>
<td>Local variable</td>
<td>yes</td>
</tr>
<tr>
<td>RCX</td>
<td>Argument #4</td>
<td>no</td>
</tr>
<tr>
<td>RDX</td>
<td>Argument #3</td>
<td>no</td>
</tr>
<tr>
<td>RSI</td>
<td>Argument #2</td>
<td>no</td>
</tr>
<tr>
<td>RDI</td>
<td>Argument #1</td>
<td>no</td>
</tr>
<tr>
<td>RBP</td>
<td>(Stack base pointer)</td>
<td>yes</td>
</tr>
<tr>
<td>RSP</td>
<td>Stack pointer</td>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reg</th>
<th>Usage</th>
<th>Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>R8</td>
<td>Argument #5</td>
<td>no</td>
</tr>
<tr>
<td>R9</td>
<td>Argument #6</td>
<td>no</td>
</tr>
<tr>
<td>R10</td>
<td>Scratch registers</td>
<td>no</td>
</tr>
<tr>
<td>R11</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>R12</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>R13</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>R14</td>
<td>Local variables</td>
<td>yes</td>
</tr>
<tr>
<td>R15</td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

### Virtual Memory Layout

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000-00007FFFFFFFFFFF</td>
<td>user space</td>
</tr>
<tr>
<td>DEAD0000xxxxxxxxxx</td>
<td>pointer poisons</td>
</tr>
<tr>
<td>FFFF800000000000-FFFF87FFFFFFFFFFF</td>
<td>hypervisor area</td>
</tr>
<tr>
<td>FFFF880000000000-FFFFC7FFFFFFFFFFF</td>
<td>direct mapping</td>
</tr>
<tr>
<td>FFFFC900000000000-FFFFE8FFFFFFFFFFF</td>
<td>vmalloc/ioremap</td>
</tr>
<tr>
<td>FFFFFFFA00000000-FFFFFFEAAAAAAAAA</td>
<td>vmemmap</td>
</tr>
<tr>
<td>FFFFFFFF80000000-FFFFFFFF9FFFFFFF</td>
<td>kernel text+data</td>
</tr>
<tr>
<td>FFFFFFFF80000000-FFFFFFFF9FFFFFFF</td>
<td>kernel modules</td>
</tr>
<tr>
<td>FFFFFFFFxxxxxxx-FFFFFFFFF5FFFFFFF</td>
<td>permanent fixmaps</td>
</tr>
<tr>
<td>FFFFFFFF6000000-FFFFFFF6DFFFFFFF</td>
<td>vsyscalls (deprecated)</td>
</tr>
</tbody>
</table>

### Calling Conventions

<table>
<thead>
<tr>
<th>Function</th>
<th>Syscall</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDI</td>
<td>RDI</td>
</tr>
<tr>
<td>RSI</td>
<td>RSI</td>
</tr>
<tr>
<td>RDX</td>
<td>RDX</td>
</tr>
<tr>
<td>RCX</td>
<td>R10</td>
</tr>
<tr>
<td>R8</td>
<td>R8</td>
</tr>
<tr>
<td>R9</td>
<td>R9</td>
</tr>
<tr>
<td>RAX</td>
<td>RAX</td>
</tr>
<tr>
<td>RDX</td>
<td>–</td>
</tr>
</tbody>
</table>

Syscall number: RAX
Kernel Oops in Detail – Other Registers

------------[ cut here ]------------

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vssock(EX)
<...
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffff811e466e>] [<ffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffffffffffff RBX: ffffffffffffffff RCX: 0000000000000006
RDX: 0000000000000000 RSI: 0000000000000000 RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008ceecac80 R14: ffff880423c33740 R15: 0000000000000001
FS:  00007f2945893760(0000) GS:fffe88043fd0000(0000) kllGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 000007f6941216960 CR3: 00000003f3bea000 CR4: 0000000000000000
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000f0000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)
Kernel Oops in Detail – Other Registers

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nls_utf8 cifs(X) nfs fscache nfsd lockd nfs_acl auth_rpcgss sunrpc autofs4 binfmt_misc mperf vsock(EX)
<...>
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G           E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>]  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88  EFLAGS: 00010202
RAX: ffffffffffffffea RBX: ffffffffffffffea RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffff880192cecc00 R08: 0000000000000002 R09: ffff880192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffff88008ce cac80 R14: ffff880423c33740 R15: 0000000000000001
FS:  00007f2945893760(0000) GS: ffff88043fd00000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 000000000005003b
CR2: 00007f6941216960 CR3: 00000003f3bea000 CR4: 0000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffff880211336000, task ffff8801b49c8040)

Segment registers, and selected control registers:
FS – userspace thread-local storage
GS – kernel percpu base

CR0: enables protected mode, paging...
CR2: the faulting virtual address
CR3: physical address of top-level page table
CR4: a mask for enabling various extensions

DRx: x86 debug registers (only printed if not in default state)
Kernel Oops in Detail – Raw Stack Contents

Stack:
ffff88016d2319e0 ffff880158fea140 00007ffce0bf0000 ffffffff81127fa4
ffff88008cecac80 ffff880211337dd8 ffff88016d2319e0 ffffffff811280f8
0000000000000001 ffff88014c826088 ffff88008cecac80 0000000100000028

Call Trace:
[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP <fff880211337d88>

---[ end trace 3dad41c41965c82c ]---
Kernel Oops in Detail – Raw Stack Contents

Stack:

ffff88016d2319e0  ffff880158fea140  00007ffce0f0000  fffffffff81127fa4
ffff88008cecac80  ffff880211337dd8  ffff88016d2319e0  fffffffff811280f8
0000000000000000  ffff88014c826088  ffff88008cecac80  0000000100000028

Call Trace:

[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP  [<ffffffff811e466e>]  shm_close+0x3e/0xb0
RSP  <fff880211337d88>

---[ end trace 3dad41c41965c82c ]---
Kernel Oops in Detail – Backtrace

Stack:

```
ffff88016d2319e0  ffff880158fea140  00007ffce0bf0000  fffffffff81127fa4
ffff88008cecac80  ffff880211337dd8  ffff88016d2319e0  fffffffff811280f8
0000000000000001  ffff88014c826088  ffff88008cecac80  0000000010000028
```

Call Trace:

```
[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997
```

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP  <ffff880211337d88>

---[ end trace 3dad41c41965c82c ]---
Kernel Oops in Detail – Backtrace

Stack:

ffff88016d2319e0  ffff880158fea140  00007ffc
ffff88008ceac80  ffff880211337dd8  ffff88022d200000000000001  ffff88014c826088  ffff88008ce

Call Trace:

[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba ff ff 48 3d 00 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00 00 48 8b

RIP  [<ffffffff811e466e>]  shm_close+0x3e/0xb0
RSP  <fff880211337d88>
---[ end trace 3dad41c41965c82c ]---
How is stack unwinding implemented?

• “Guess”: All code lies in a designated range of addresses
  – There is a symbol table to convert addresses to individual function names
  – Every value on stack that looks like a pointer to this range can be a return address
  – Simple, but relatively slow and with many false positives (everything is marked “?”)

• Use RBP register when CONFIG_FRAME_POINTER is enabled
  – RBP will always point to the previous frame’s stored RBP value, and return address lies next to it
  – Simple pointer chasing plus collecting the return addresses
  – Fast, reliable, but maintaining RBP has performance impact on the kernel (5-10%)

• Using debuginfo to locate the stack frames from current RIP value
  – DWARF Call Frame Info (CFI) – DWARF unwinder was in mainline for a while, but then removed
    (slow, sometimes unreliable, requires assembler annotations)
  – ORC – uses custom unwinder data generated by objtool during build – since 4.14, also for
    reliable stack traces needed by some of the live patching consistency models
  – Relatively fast, reliable, no performance impact on the kernel (2-4 MB memory overhead)
Kernel Oops in Detail – Backtrace

Stack:

ffff88016d2319e0  ffff880158fea140  00007ffce0bf0000  fffffffff81127fa4
ffff88008cecac80  ffff880211337dd8  ffff88016d2319e0  fffffffff811280f8
0000000000000001  ffff88014c826088  ffff88008cecac80  0000000100000002

Call Trace:

[<ffffffff81127fa4>]  remove_vma+0x24/0x80
[<ffffffff811280f8>]  exit_mmap+0xf8/0x120
[<ffffffff810602d9>]  mmput+0x49/0x100
[<ffffffff81065192>]  exit_mm+0x122/0x160
[<ffffffff81066f39>]  do_exit+0x189/0x470
[<ffffffff8106725d>]  do_group_exit+0x3d/0xb0
[<ffffffff810672e2>]  sys_exit_group+0x12/0x20
[<ffffffff81471df2>]  system_call_fastpath+0x16/0x1b
[<00007f29408be99b>]  0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP  [<ffffffff811e466e>]  shm_close+0x3e/0xb0
RSP  <fff880211337d88>

---[ end trace 3dad41c41965c82c ]---
Kernel Oops in Detail – Backtrace

Stack:

ffff88016d2319e0 ffff880158fea140 00007ffce0bf0000 ffffffff81127fa4
ffff88008ceccac80 ffff880211337dd8 ffff88016d2e000000000000000001 ffff88014c826088 ffff88008ce

Call Trace:

[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP <ffff880211337d88>

---[ end trace 3dad41c41965c82c ]---

Context matters! `shm_close()` is unlikely to be buggy by itself.

Here, task was exiting and cleaning up its memory layout – a list of vma’s.
(see `/proc/pid/maps` for an example)
A vma was backed by shared memory segment, unregistering its usage led to BUG.
Kernel Oops in Detail – Code Listing

Stack:

```
ffff88016d2319e0 ffff880158fea140 00007ffce0bf0000 ffffffff81127fa4
ffff88008ce cac80 ffff880211337dd8 ffff88016d2319e0 ffffffff811280f8
00000000000000001 ffff88014c826088 ffff88008ce cac80 0000000100000028
```

Call Trace:

```
[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff810666f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997
```

Code:
```
8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b
RIP  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP  <ffff880211337d88>
```

---[ end trace 3dad41c41965c82c ]---
Kernel Oops in Detail – Code Listing

Stack:

ffff88016d2319e0  ffff880158fea140  00007ffce0bf0000  ffffffff81127fa4
ffff88008cecac80  ffff880211337dd8  ffff88016d2319e0  0000000000000001  ffff88014c826088  ffff88008cecac80

Call Trace:

[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP <fff880211337d88>

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.
Kernel Oops - ./scripts/decode/decode output

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00 00 48 8b

All code

========

0:   8b 6b 08                mov    0x8(%rbx),%ebp
 3:   4c 8d a5 a0 00 00 00    lea    0xa0(%rbp),%r12
  a:   4c 89 e7                mov    %r12,%rdi
  d:   e8 0b 49 28 00          callq  0x28491d
 12:   8b 33                   mov    (%rbx),%esi
 14:   48 8d bd 98 00 00 00    lea    0x98(%rbp),%rdi
 1b:   e8 7d ba ff ff          callq  0xffffffffffffba9d
 20:   48 3d 00 f0 ff ff       cmp    $0xfffffffffffff000,%rax
 26:   48 89 c3                mov    %rax,%rbx
 29:   76 0a                   jbe    0x35
 2b:*  0f 0b                   ud2             <-- trapping instruction
 2d:   eb fe                   jmp    0x3d
 2f:   66 0f 1f 44 00 00       nopw   0(%rax,%rax,1)
 35:   65 48 8b 04 25 00 a6    mov    %gs:0xa600,%rax
 3c:   00 00                   re...
 3e:   48                      .byte 0x8b
 3f:   8b
Kernel Oops - ./scripts/decodecode output

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8f 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44
All code
========
0:  8b 6b 08                mov    0x8(%rbx),%ebp
3:  4c 8d a5 a0 00 00 00    lea    0xa0(%rbp),%r12
a:  4c 89 e7                mov    %r12,%rdi
d:  e8 0b 49 28 00          callq  0x28491d
12: 8b 33                   mov    (%rbx),%esi
14: 48 8d bd 98 00 00 00    lea    0x98(%rbp),%rdi
1b: e8 7d ba ff ff          callq  0xffffffffffffba9d
20: 48 89 c3                mov    %rdi,%rbx
23: 76 0a                   jbe    0x35
2b:* 0f 0b                   ud2             <-- trapping instruction
2d: eb fe
2f: 66 0f 1f 44 00 00       jmp    0x2d
35: 65 48 8b 04 25 00 a6    nopw   0x0(%rax,%rax,1)
3c: 00 00                   mov    %gs:0xa600,%rax
3e: 48                      rex.W
3f: 8b                      .byte 0x8b

struct shmid_kernel *shp;
...
shp = shm_lock(ns, sfd->id);
BUG_ON(IS_ERR(shp));

rdi contained value of ns
rsi contained value of sfd->id
both might be lost now
struct shmid_kernel *shp;
...
shp = shm_lock(ns, sfd->id);
BUG_ON(IS_ERR(shp));

rax contains value of shp current, not lost
### Kernel Oops - ./scripts/decodecode output

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8f 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>8b 6b 08</td>
</tr>
<tr>
<td>3:</td>
<td>4c 8d a5 a0 00 00 00</td>
</tr>
<tr>
<td>a:</td>
<td>4c 89 e7</td>
</tr>
<tr>
<td>d:</td>
<td>e8 0b 49 28 00</td>
</tr>
<tr>
<td>12:</td>
<td>8b 33</td>
</tr>
<tr>
<td>14:</td>
<td>48 8d bd 98 00 00 00</td>
</tr>
<tr>
<td>1b:</td>
<td>e8 7d ba ff ff</td>
</tr>
<tr>
<td>20:</td>
<td>48 3d 00 f0 ff ff</td>
</tr>
<tr>
<td>26:</td>
<td>48 89 c3</td>
</tr>
<tr>
<td>29:</td>
<td>76 0a</td>
</tr>
<tr>
<td>2b:*</td>
<td>0f 0b</td>
</tr>
<tr>
<td>2d:</td>
<td>eb fe</td>
</tr>
<tr>
<td>2f:</td>
<td>66 0f 1f 44 00 00</td>
</tr>
<tr>
<td>35:</td>
<td>65 48 8b 04 25 00 a6</td>
</tr>
<tr>
<td>3c:</td>
<td>00 00</td>
</tr>
<tr>
<td>3e:</td>
<td>48</td>
</tr>
<tr>
<td>3f:</td>
<td>8b</td>
</tr>
</tbody>
</table>

```c
struct shmid_kernel *shp;
...
shp = shm_lock(ns, sfd->id);
BUG_ON(IS_ERR(shp));
```
Kernel Oops – Revisiting Register Values

--------------[ cut here ]--------------
kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet st ide_cd_mod ide_core bridge stp llc joydev ext2 des_generic ecb
md4 nls_utf8 cifs(X) nfs fscache nfds lockd nfs_acl auth_rpcgss sunrpc autos4 binfmt_misc mperf vsock(EX)
<...>
[last unloaded: ppa]
Supported: Yes, External

Pid: 26341, comm: <redacted> Tainted: G E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>] [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88 EFLAGS: 00010202
RAX: ffffffff811e466e RBX: ffffffff811e466e RCX: 0000000000000006
RDX: 0000000000000000 RSI: 000000000000005c RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffffffff81a46920
R10: ffffffff81a46920 R11: ffffffff81a46920 R12: ffffffff81a46920
R13: ffffffff81a46920 R14: ffffffff81a46920 R15: 0000000000000001
FS: 00007f2945893760 GS: ffffffff8043fd0000(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000003b
CR2: 0000000000000000 CR3: 0000000000000000000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000000000000 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffffffff80211336000, task ffffffff801b49c8040)
Kernel Oops – Revisiting Register Values

kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/ipc/shm.c:205!
invalid opcode: 0000 [#1] SMP
CPU 1
Modules linked in: lp parport_pc af_packet md4 nls_utf8 cifs(X) nfs fscache nbsd lockd<...> supported: Yes, External
Pid: 26341, comm: <redacted> Tainted: G           E X 3.0.101-84-default #1 VMware, Inc. VMware Virtual
Platform/440BX Desktop Reference Platform
RIP: 0010:[<ffffffff811e466e>]  [<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP: 0018:ffff880211337d88  EFLAGS: 00010202
RAX: fffffffffffffffea RBX: fffffffffffffffea RCX: 0000000000000006
RDX: 0000000000000000 RSI: 0000000000000000 RDI: 0000000000000006
RBP: ffffffff81a46920 R08: 0000000000000002 R09: ffff8804256a84d0
R10: ffffffff8180192cecc00 R11: ffffffff81215a80 R12: ffffffff81a469c0
R13: ffffffff880808ceacac80 R14: ffffffff880423c33740 R15: 0000000000000001
FS: 00007f2945893760(0000) GS:fffffffff000000000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000000503b
CR2: 000007f6941216960 CR3: 00000003f3bea000 CR4: 0000000001407e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 0000000ff0fff0 DR7: 0000000000000000
Process <redacted> (pid: 26341, threadinfo ffffffff880211336000, task ffffffff8801b49c840)

RAX (and RBX) is the error value -22 == -EINVAL
RDI is a ns pointer? definitely not
RSI is shm->id? could be?
RBP, R11, R12 – kernel code/static data
R09, R10, R13, R14 – kernel data
Kernel Oops in Detail – End of the Oops

Stack:
```
ffff88016d2319e0 ffff880158fe0a140 00007ffce0bf0000 ffffffffff81127fa4
ffff88008cecac80 ffff880211337dd8 ffff88016d2319e0 ffffffffff811280f8
0000000000000000 1 ffff88014c826088 ffff88008cecac80 00000001000000028
```

Call Trace:
```
[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997
```

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 8e 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP  [ffffffff811e466e>]  shm_close+0x3e/0xb0
RSP  <fff880211337d88>
---[ end trace 3dad41c41965c82c ]---
Kernel Oops in Detail – End of theOops

Stack:
ffff88016d2319e0 ffff880158fea140 00007ffce0bf0000 ffff880881127fa4
ffff88008cecac80 ffff880211337dd8 ffff88016d2319e0 ffff8808811280f8
0000000000000001 ffff88014c826088 ffff88008cecac80 0000000100000028

Call Trace:
[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810606d9>] mmput+0x49/0x100
[<ffffffff81065192>] mmput+0x49/0x100
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff81471df2>] system_call_fastpath+0x16/0x1b
[00007f29408be998] 0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 52 48 8d bd 98 00 00 00 e8 7d ba
ff ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 4b 89
RIP <ffffffff811e466e] shm_close+0x3e/0xb0
RSP <fff880211337d88>

---[ end trace 3dad41c41965c82c ]---

RIP + RSP again in case the first ones scrolled away.
Kernel Oops in Detail – End of the Oops

Stack:
ffff88016d2319e0 ffff880158fea140 00007ffce0bf0000 fffffff81127fa4
ffff88008cecac80 ffff880211337dd8 ffff88016d2319e0 fffffff811280f8
0000000000000001 ffff88014c826088 ffff88008cecac80 0000000100000028

Call Trace:
[<ffffffff81127fa4>] remove_vma+0x24/0x80
[<ffffffff811280f8>] exit_mmap+0xf8/0x120
[<ffffffff810602d9>] mmput+0x49/0x100
[<ffffffff81065192>] exit_mm+0x122/0x160
[<ffffffff81066f39>] do_exit+0x189/0x470
[<ffffffff8106725d>] do_group_exit+0x3d/0xb0
[<ffffffff810672e2>] sys_exit_group+0x12/0x20
[<ffffffff471df2>] system_call_fastpath+0x16/0x1b
[<00007f29408be998>] 0x7f29408be997

Code: 8b 6b 08 4c 8d a5 a0 00 00 00 4c 89 e7 e8 0b 49 28 00 8b 33 48 8d bd 98 00 00 00 e8 7d ba
ff ff 48 3d 00 f0 ff ff 48 89 c3 76 0a <0f> 0b eb fe 66 0f 1f 44 00 00 65 48 8b 04 25 00 a6 00
00 48 8b

RIP  
[<ffffffff811e466e>] shm_close+0x3e/0xb0
RSP <fff880211337d88>

---[ end trace 3dad41c41965c82c ]---

RIP + RSP again in case the first ones scrolled away.

First oops_id during uptime is random, then increased monotonically.

The intention is to recognize duplicate reports by sites such as www.kerneloops.org
What else can produce oops/panic?

• BUG_ON() as seen in the example – hard assertion
  – WARN_ON_ONCE() - soft assertion, unless panic_on_warn is enabled
• Memory paging related faults – check CR2 register!
  – BUG: unable to handle kernel paging request
  – ... handle NULL pointer dereference (when bad_addr < PAGE_SIZE) – a structure’s field might be accessed with non-zero offset
  – Corrupted page table (reserved bits set, etc.)
  – Kernel trying to execute NX-protected page
  – Kernel trying to execute/access userspace page (Intel SMEP/SMAP feature)
  – Failed bounds check in kernel mode (Intel MPX feature)
  – Kernel stack overflow
  – General protection fault, unhandled double fault
• FPU, SIMD exceptions from kernel mode
What else can produce oops/panic?

• Soft lockup
  – CPU spent over 20s in kernel without reaching a schedule point (in non-preemptive kernels)
  – A warning, unless related config or bootparam softlockup_panic enabled
    • Soft lockup can often recover, so not good idea to enable that in production, especially in a guest VM

• Hard lockup
  – CPU spent over 10s with disabled interrupts
  – Panic when hardlockup_panic is enabled

• Detection of both combines several generic mechanisms (for each CPU)
  – High priority kernel watchdog thread updates the soft lockup timestamp
  – High resolution timer (hrtimer) is configured to deliver periodic interrupts, the handler resets the hard lockup flag and wakes up the watchdog thread
  – It also reports soft lockup when the watchdog thread did not touch the soft lockup timestamp
  – Non-maskable interrupt (NMI) perf event reports hard lockup if hrtimer interrupts were not processed and hard lockup flag remains set
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 debugging config options (later)
Kernel Debugging – General Approach

• First, understand the immediate cause
  – Typically some unexpected/wrong value somewhere in memory
    • NULL pointer access, because certain structure's field was NULL/bogus
    • Page table corruption, SLAB corruption, strange lock value...
  – Here we know shm_lock() returned -EINVAL, but we don't know yet why

• Second, try to determine what could cause the value
  – Single bit flip? RAM error (yes, they do happen without ECC)
    • Often manifests as multiple different bugs from the same machine
  – Wrong use by upper layers? For example, SLAB corruption is almost never a bug in SLAB code, but e.g. result of double-free of a kernel object allocated from SLAB
  – Logical error in code? Race condition? Stray write?

• Note: no general and complete recipe
  – Mostly from own experience, or learn from others’ analyses
  – Knowing the subsystem helps, still lots of staring into source code of the exact version
Why does `shm_lock()` return `-EINVAL`?

/*
 * shm_lock_(check_) routines are called in the paths where the rw_mutex
 * is not necessarily held.
 */

static inline struct shmid_kernel *shm_lock(struct ipc_namespace *ns, int id)
{
    struct kern_ipc_perm *ipcp = ipc_lock(&shm_ids(ns), id);

    if (IS_ERR(ipcp))
        return (struct shmid_kernel *)ipcp;

    return container_of(ipcp, struct shmid_kernel, shm_perm);
}
Why does ipc_lock() return -EINVAL?

/**
 * ipc_lock - Lock an ipc structure without rw_mutex held
 * @ids: IPC identifier set
 * @id: ipc id to look for
 *
 * Look for an id in the ipc ids idr and lock the associated ipc object.
 *
 * The ipc object is locked on exit.
 */

struct kern_ipc_perm *ipc_lock(struct ipc_ids *ids, int id)
{
    struct kern_ipc_perm *out;
    int lid = ipcid_to_idx(id);

    rcu_read_lock();
    out = idr_find(&ids->ipcs_idr, lid);
    if (out == NULL) {
        rcu_read_unlock();
        return ERR_PTR(-EINVAL);
    }

    return out;
}
Why does `ipc_lock()` return `-EINVAL`?

```c
spin_lock(&out->lock);

/* `ipc_rmid()` may have already freed the ID while `ipc_lock` was spinning: here verify that the structure is still valid */
if (out->deleted) {
    spin_unlock(&out->lock);
    rcu_read_unlock();
    return ERR_PTR(-EINVAL);
}

return out;
```

- Either `idr_find()` didn’t find the `shmid`, or it was already deleted
- The oops report can’t help anymore, need to inspect memory – crash dump
Obtaining crash dumps

- Several historical methods
  - diskdump, netdump, LKCD project...
  - Not very reliable (some parts of crashed kernel must still work) nor universal, needs dedicated server on same network etc.
  - Out of tree patches, included in old enterprise distros

- Current solution: kexec-based kdump
  - Crash kernel loaded into a boot-reserved memory area
    - Size specified as boot parameter, no universally good value, depends on hardware
  - On panic, kexec switches to the crash kernel without reboot
  - Memory of crashed kernel available as /proc/vmcore
  - Kdump utility can save to disk, network, filter pages...
    - kexec (8), kdump (5), makedumpfile (8)
  - In VM guest environment, hypervisor dumps also possible
Analyzing kernel crash dumps

- **gdb** can be used to open ELF based dumps
  - But those are not easily compressed and filtered
- **gdb** has no understanding of kernel internals or virtual/physical mapping
  - There are some Python scripts under `scripts/gdb` in the Linux source
  - Can obtain per-cpu variables, dmesg, modules, tasks
- A better tool for Linux kernel crash dumps – **crash**
  - Created by David Anderson from Red Hat
  - Understands all dump formats – kdump (compressed), netdump, diskdump, xendump, KVM dump, s390, LKCD, ...
  - Understands some kernel internals: memory mapping, tasks, SLAB/SLUB objects, ...
  - Can e.g. walk linked lists, pipe the output for further postprocessing
  - Extensible with Eppic – a C interpreter tailored to work with C structures stored in a dump, or Python (pykdump)
crash – disadvantages

- Uses gdb internally in a suboptimal way
  - Old version, embedded in the tool itself
  - Mostly by invoking some gdb command and postprocessing its output
- Backtraces are not using full potential of gdb
  - Not using debuginfo to print values of function parameters, local variables...
- Machine running crash must be of the same architecture as the machine that created the kernel dump
- pykdump works by executing crash commands and parsing their output
Invoking crash

- On core dump
  - `crash vmlinux.gz vmlinux.debug vmcore`
- On live system
  - `crash vmlinux.gz vmlinux.debug`

- Options
  - `-s` silent, output not paged to less
  - `-i file` execute commands from file
  - `--mod dir` search for module debuginfo in dir
  - `--minimal` only basic commands (for too broken dumps)
Invoking crash – welcome screen

KERNEL: vmlinux.gz
DEBUGINFO: vmlinux.debug
DUMPFILE: vmcore
CPUS: 8
DATE: Thu Apr 10 16:07:34 2014
UPTIME: 7 days, 03:17:51
LOAD AVERAGE: 0.01, 0.02, 0.05
TASKS: 161
NODENAME: lpapp114
RELEASE: 3.0.101-0.7.17-default
VERSION: #1 SMP Tue Feb 4 13:24:49 UTC 2014 (90aac76)
MACHINE: x86_64 (2399 Mhz)
MEMORY: 64 GB
PANIC: "[615702.371868] kernel BUG at /usr/src/packages/BUILD/kernel-default-3.0.101/linux-3.0/mm/slab.c:539!"
PID: 58
COMMAND: "kworker/6:1"
TASK: ffff88080e03e680 [THREAD_INFO: ffff88080e040000]
CPU: 6
STATE: TASK_RUNNING (PANIC)
## Invoking crash – help screen

```
invoking crash – help screen

crash> help

*              extend         log            rd             task
alias          files          mach           repeat         timer
ascii          foreach        mod            runq           tree
bpf            fuser          mount          search         union
tb             gdb            net            set            vm
btop           help           p              sig            vtop
dev            ipcs           ps             struct         waitq
dis            irq            pte            swap           whatis
eval           kmem           ptob           sym            wr
exit           list           ptov           sys            q
```
Basic crash commands

- `dmesg` (log) – same as the shell command
- `mod -t [mod]` – module taint flags
- `ps` – list processes (kernel/user), count by state, sort by last scheduled time...
- `dis [-l] [-r] [addr|sym]` – disassemble code
- `bt [task|pid] [-a]` – show backtrace(s)
  - `-l` – include file/line transition
  - `-FF` – translate addresses to symbols/slab objects

- Full input/output redirection/piping support
Basic crash commands

- `struct [-o] <name> [addr]` – print structure layout, offsets, values at address
- `rd [addr|symbol] [count]` – read/format raw memory contents
  - `wr` – write memory (for live systems)
- `search [-m mask] [value|expr|sym|string]`
  - search memory for given value (with optional mask)
- `kmem [-s] addr` – show info about the address
  - Is it a symbol? Slab object? Free page? Stack of a task?
- `vtop/ptov, pte` – address translation commands
crash - More complex inspection

- **list** `<addr>` – traverse objects via embedded `list_head`, print them out (as `struct command` does)
- **tree** `<root>` – traverse red-black or radix tree
- **foreach** `<command>` – apply one of a subset of commands on each task
- **dev, files, mount, ipcs, irq, net, swap, timer, runq, waitq**...
- **fuser** `[path|inode]` – who has a file open?
Using crash to solve our bug

- Recall, with the Oops and sources, we determined:
  - `shm_lock()` returned -EINVAL unexpectedly
  - Because `ipc_lock()` returned -EINVAL unexpectedly
  - Because `idr_find(shmid)` didn’t find the `shmid`, or it was already deleted.

- With crash, we want to find:
  - What was the `shmid` value?
    - Where was it obtained from?
  - Was it not found at all or was the object deleted?
    - Why?
crash – shm_close() disassembly

crash> dis -lr ffffffff811e466e
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 196
0xffffffff811e4630 <shm_close>:         push   %r12
0xffffffff811e4632 <shm_close+2>:       push   %rbp
0xffffffff811e4633 <shm_close+3>:       push   %rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 198
0xffffffff811e4634 <shm_close+4>:       mov    0x98(%rdi),%rax
0xffffffff811e463b <shm_close+11>:      mov    0xa0(%rax),%rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 200
0xffffffff811e4642 <shm_close+18>:      mov    0x8(%rbx),%rbp
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 202
0xffffffff811e4646 <shm_close+22>:      lea    0xa0(%rbp),%r12
0xffffffff811e464d <shm_close+29>:      mov    %r12,%rdi
0xffffffff811e4650 <shm_close+32>:      callq  0xffffffff81468f60 <down_write>
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 125
0xffffffff811e4655 <shm_close+37>:      mov    (%rbx),%esi
0xffffffff811e4657 <shm_close+39>:      lea    0x98(%rbp),%r12
0xffffffff811e465e <shm_close+46>:      callq  0xffffffff811e00e0 <ipc_lock>
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 127
0xffffffff811e4663 <shm_close+51>:      cmp    $0xffffffffffffff000,%rax
0xffffffff811e4669 <shm_close+57>:      mov    %rax,%rbx
0xffffffff811e4669 <shm_close+57>:      mov    (%rbx),%esi
0xffffffff811e4669 <shm_close+57>:      lea    0x98(%rbp),%r12
0xffffffff811e466e <shm_close+62>:      ud2
Our goal here is to find whether some register contained shmid or a pointer to some structure that contains it (or pointer to a structure that has a pointer to another structure...)

```plaintext
Our goal here is to find whether some register contained shmid or a pointer to some structure that contains it (or pointer to a structure that has a pointer to another structure...)
```
crash – shm_close() disassembly

```
crash> dis -lr ffffffff811e466e
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 196
0xffffffff811e4630 <shm_close>:         push   %r12
0xffffffff811e4632 <shm_close+2>:       push   %rbp
0xffffffff811e4633 <shm_close+3>:       push   %rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 198
0xffffffff811e4634 <shm_close+4>:       mov    0x98(%rdi),%rax
0xffffffff811e463b <shm_close+11>:      mov    0xa0(%rax),%rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 200
0xffffffff811e4642 <shm_close+18>:      mov    0x8(%rbx),%rbp
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 202
0xffffffff811e4646 <shm_close+22>:      lea    0xa0(%rbp),%r12
0xffffffff811e464d <shm_close+29>:      mov    %r12,%rdi
0xffffffff811e4650 <shm_close+32>:      callq  0xffffffff811e00e0 <ipc_lock>
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 125
0xffffffff811e4655 <shm_close+37>:      mov    (%rbx),%esi
0xffffffff811e4657 <shm_close+39>:      lea    0x98(%rbp),%r12
0xffffffff811e465e <shm_close+46>:      callq  0xffffffff811e466e <shm_close+62> <ud2>
```
crash – shm_close() disassembly

```
crash> dis -lr ffffffff811e466e
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 196
0xffffffff811e4630 <shm_close>:         push   %r12
0xffffffff811e4632 <shm_close+2>:       push   %rbp
0xffffffff811e4633 <shm_close+3>:       push   %rbx

/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 198
0xffffffff811e4634 <shm_close+4>:       mov    0x98(%rdi),%rax
0xffffffff811e463b < shm_close+11>:      mov    0xa0(%rax),%rbx

/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 200
0xffffffff811e4642 <shm_close+18>:      mov    0x8(%rbx),%rbp
0xffffffff811e4646 <shm_close+22>:      lea    0xa0(%rbp),%r12
0xffffffff811e464d <shm_close+29>:      mov    %r12,%rdi
0xffffffff811e4650 <shm_close+32>:      callq  0xffffffff81468f60 <down_write>

/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 125
0xffffffff811e4663 <shm_close+51>:      cmp    $0xfffffffffffff000,%rax

```

shm_close(struct vm_area_struct *vma)

vma pointer is in rdi

```
struct file * file = vma->vm_file;
```

file pointer is in rax
crash – shm_close() disassembly

```c
void shm_close(struct vm_area_struct *vma) {
  struct file * file = vma->vm_file;
  struct shm_file_data *sfd = shm_file_data(file);
}
```
crash – shm_close() disassembly

crash> dis -lr ffffffff811e466e
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 196
0xffffffff811e4630 <shm_close>:         push   %r12
0xffffffff811e4632 <shm_close+2>:       push   %rbp
0xffffffff811e4633 <shm_close+3>:       push   %rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 198
0xffffffff811e4634 <shm_close+4>:       mov    0x98(%rdi),%rax
0xffffffff811e463b <shm_close+11>:      mov    0xa0(%rax),%rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 200
0xffffffff811e4642 <shm_close+18>:      mov    0x8(%rbx),%rbp
0xffffffff811e4646 <shm_close+22>:      lea    0xa0(%rbp),%r12
0xffffffff811e464d <shm_close+29>:      mov    %r12,%rdi
0xffffffff811e4650 <shm_close+32>:      callq  0xffffffff81468f60 <down_write>
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 125
0xffffffff811e4663 <shm_close+51>:       cmp    $0xfffffffffffff000,%rax
0xffffffff811e466e <shm_close+62>:      ud2

shm_close(struct vm_area_struct *vma)
vma pointer is in rdi

struct file * file = vma->vm_file;
file pointer is in rax (?)

struct shm_file_data *sfd = shm_file_data(file);
sfd pointer is in rbx

struct ipc_namespace *ns = sfd->ns;
down_write(&shm_ids(ns).rw_mutex);
we just lost vma that was in rdi
we might have lost *file in rax
crash – shm_close() disassembly

```plaintext
shm_close(struct vm_area_struct *vma)

vma-pointer is in rdi

struct file * file = vma->vm_file;
file-pointer is in rax

struct shm_file_data *sfd = shm_file_data(file);
sfd-pointer is in rbx

shp = shm_lock(ns, sfd->id);
shm_lock(ns, id) is inlined and does:
struct kern_ipc_perm *ipcp = ipc_lock(&shm_ids(ns), id);
rsi has sfd->id which we want to know
rdi has &shm_ids(ns) we definitely lost rax now
```
crash – shm_close() disassembly

```
crash> dis -lr ffffffff811e466e
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 196
0xffffffff811e4630 <shm_close>:         push   %r12
0xffffffff811e4632 <shm_close+2>:       push   %rbp
0xffffffff811e4633 <shm_close+3>:       push   %rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 198
0xffffffff811e4634 <shm_close+4>:       mov    0x98(%rdi),%rax
0xffffffff811e463b <shm_close+11>:      mov    0xa0(%rax),%rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 200
0xffffffff811e4642 <shm_close+18>:      mov    0x8(%rbx),%rbp
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 202
0xffffffff811e4646 <shm_close+22>:      lea    0xa0(%rbp),%r12
0xffffffff811e464d <shm_close+29>:      mov    %r12,%rdi
0xffffffff811e4650 <shm_close+32>:      callq  0xffffffff81468f60 <down_write>
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 125
0xffffffff811e4655 <shm_close+37>:      mov    (%rbx),%esi
0xffffffff811e4657 <shm_close+39>:      lea    0x98(%rbp),%rdi
0xffffffff811e465e <shm_close+46>:      callq  0xffffffff811e00e0 <ipc_lock>
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 127
0xffffffff811e4663 <shm_close+51>:      cmp    $0xfffffffffffff000,%rax
0xffffffff811e4669 <shm_close+57>:      mov    %rax,%rbx
/usr/src/debug/kernel-default-3.0.101/linux-3.0/ipc/shm.c: 127
0xffffffff811e466c <shm_close+60>:      jbe    0xffffffff811e4678 <shm_close+72>
0xffffffff811e466e <shm_close+62>:      ud2
```

shm_close(struct vm_area_struct *vma)

vma-pointer is in rdi

struct file * file = vma->vm_file;

file-pointer is in rax

struct shm_file_data *sfd = shm_file_data(file);

sfd-pointer is in rbx

we lost sfd in rbx :(

```
Getting the vma pointer

- We lost the vma pointer in shm_close() and all intermediate pointers leading to shmid
  - Checking ipc_lock() asm also showed that shmid gets quickly lost inside it
- There's a high chance some function up the call stack saved the vma, but how to find it without following more assembler?
  - vma's (struct vm_area_struct objects) have a dedicated SLAB cache, and crash can mark these caches automatically in the backtrace
crash – rich backtrace with local data

```
crash> bt -FF
...
#4 [ffff880211337cd0] invalid_op at ffffffff81472ddb
   [exception RIP: shm_close+62]
   RIP: ffffffff811e466e  RSP: ffffffff81337d88  RFLAGS: 00010202
   RAX: ffffffffeffffffffa  RBX: ffffffffeffffffffa  RCX: 0000000000000006
   RDX: 0000000000000000  RSI: 0000000000000005c  RDI: 0000000000000006
   RBP: ffffffff81a46920  R8: 0000000000000002  R9: ffffffff811e466e
   R10: ffffffff81337f80  R11: ffffffff81337f80  R12: ffffffff81337f80
   R13: ffffffff81337f80  R14: ffffffff81337f80  R15: 0000000000000001
   ORIG_RAX: ffffffff81337f80  CS: 0010  SS: 0018
   ffffffff81337f80: 0000000000000001 [ffff880423c33740:signal_cache]
   ffffffff81337f80: [ffff88008ceca80:mm_struct] init_ipc_ns+160
   ffffffff81337f80: init_ipc_ns [ffff88008ceca80:mm_struct] ffffffff81337f80
   ffffffff81337f80: apparmor_file_free_security [ffff880192cecc0:size-32]
   ffffffff81337f80: [ffff8804256a84d0:idr_layer_cache] 0000000000000002
   ffffffff81337f80: ffffffff81337f80 0000000000000006
   ffffffff81337f80: 0000000000000000 0000000000000005c
   ffffffff81337f80: 0000000000000006 ffffffff81337f80
   ffffffff81337f80: shm_close+62 0000000000010202
   ffffffff81337f80: 0000000000000018 shm_close+51
   ffffffff81337f80: shm_close+51 0000000000010202
   ffffffff81337f80: shm_close+51 0000000000000018 shm_close+51
   ffffffff81337f80: [ffff88016d2319e:vm_area_struct] [ffff880158fe140:vm_area_struct]
   ffffffff81337f80: 00007ffce0bf0000 remove_vma+36
#5 [ffff880211337d90] remove_vma at ffffffff81127fa4
   ffffffff81337d90: [ffff88008ceca80:mm_struct] ffffffff81337d90
   ffffffff81337d90: [ffff88008ceca80:mm_struct] exit_mmap+248
#6 [ffff880211337d90] exit_mmap at ffffffff811280f8
```
remove_vma(vma) - vma in rdi
<remove_vma>: push %rbp
<remove_vma+1>: push %rbx
<remove_vma+2>: mov %rdi,%rbx
<remove_vma+5>: sub $0x8,%rsp
<remove_vma+9>: mov 0x88(%rdi),%rax
<remove_vma+16>: mov 0x18(%rdi),%rbp
<remove_vma+20>: test %rax,%rax
<remove_vma+23>: je <remove_vma+36>
<remove_vma+25>: mov 0x8(%rax),%rax
<remove_vma+29>: test %rax,%rax
<remove_vma+32>: je <remove_vma+36>
<remove_vma+34>: callq *%rax
<remove_vma+36>: mov 0x98(%rbx),%rdi

We enter shm_close() with vma in rbx, return address to remove_vma+36 on stack:

<shm_close>: push %r12
<shm_close+2>: push %rbp
<shm_close+3>: push %rbx

Our vma is ffff88016d2319e0, hooray!
crash – finally getting the shmid

```plaintext
crash> struct vm_area_struct.vm_file ffff88016d2319e0
    vm_file = 0xffff8800148fcb80

crash> struct file.private_data 0xffff8800148fcb80
    private_data = 0xffff8803f382cc60

crash> struct shm_file_data 0xffff8803f382cc60
struct shm_file_data {
    id = 13008988,
    ns = 0xfffffffff81a46920 <init_ipc_ns>,
    file = 0xffff88037a645680,
    vm_ops = 0xfffffffff816268a0 <shmem_vm_ops>
}
```
crash – checking existing shmid’s with `ipcs`

- This checks the same structure that `ipc_lock()` was searching by `idr_find()`
  - No need to inspect it manually (a colleague did that, anyway)
- Our id 13008988 is indeed not there – was it freed too early, or was there a corruption?
  - The id 13008943 looks pretty close...

```
    crash> ipcs -m
    SHMID_KERNEL      KEY      SHMID      UID   PERMS   BYTES      NATTCH
    STATUS
    ffff880424c74b90  00004dc4  98304      50016  760   40141728   1
    ...  
    ffff8803792ed0d0  000027b4 12976174   28900  740   4194480    24
    ffff8803790f5790  000027c5 13008943   28900  740   20672      12
    ffff880365da6b90  0000277a 13795376   28900  740   512000000  8519
    ffff880365da6c90  000027ac 13893683   28900  740   535000000  11
```
crash – checking suspicious shmid

.crash> struct shmid_kernel ffff8803790f5790
struct shmid_kernel {
    shm_perm = {
        ...  
        deleted = 0,
        id = 13008943,
        key = 10181,
    }
    shm_file = 0xffff88037a645680,
    shm_nattch = 12,
}

• Let's compare it with our shm_file_data – same file pointer 0xffff88037a645680!
• One of the two objects here has most likely corrupted id field, but which one?

.crash> struct shm_file_data 0xffff8803f382cc60
struct shm_file_data {
    id = 13008988,
    file = 0xffff88037a645680,
}
crash – which shmид is corrupted?

```
crash> search 0xfffff88037a645680 # the shm_file pointer
```

The search returns a number of addresses that we can inspect:

- the `shm_id_kernel` ffff8803790f5790 with `shm_file` field
- two self-references (empty `list_head`)
- 12 objects from the size-32 kmalloc cache
  - one is our `shm_file_data` 0xfffff8803f382cc60 with id 13008988
  - the other 11 appear to be correct `shm_file_data` structures with id 13008943
  - `shm_nattch` == 12 – there should be 12 instances with id 13008943
- Conclusion: the `shm_id_kernel` in the registry with id 13008943 is correct, the single `shm_file_data` with id 13008988 (that triggered BUG) is wrong

- Note: it’s often possible to cross-check data in kernel like this, as there are redundancies for functionality or performance reasons
What next?

- We found what exactly was wrong (first phase), but not why it was wrong
  - Doesn’t look like a RAM error. And catching stray writes done by kernel itself is very hard, especially from a crash dump.
  - The corrupted value was in object from size-32 slab cache, shared by everyone calling kmalloc() with size between 17 and 32 bytes – overflow, use-after-free?
  - It only happened once. Further occurrences would show if this happens at the same place or randomly, and if there’s a pattern in the corruption itself.
- Indeed, other bug reports appeared later from same set of customer’s systems – crashes while reading /proc/slabinfo and other operation in SLAB due to broken lists of slab structures
  - crash will report some errors in those while running the kmem commands
  - Not an extensive integrity check of SLAB itself though
  - Manual checking would be tedious
crash-python

• An alternative to crash to overcome some of its limitations
  • Especially lack of rich stack traces, architecture dependency and complicated scripting
• Extend gdb Python API so that the whole gdb target can be provided by Python code
  • Originally patches from Linaro, adopted by SUSE colleagues
  • Use libkdumpfile and libaddrxlat libraries via their Python API to read vmcore files and translate virtual addresses to the dumped memory contents
  • Write gdb target on top (Linux tasks translate to gdb’s threads, etc.)
• All kernel-specific knowledge written in Python on top of gdb API for symbols, types and values
  • Implement equivalents to crash commands
  • Implement new commands to inspect kernel state – **SLAB/SLUB integrity checking**
  • Building blocks reusable for further ad-hoc scripting to help with a particular bug
crash-python applied on the bug
crash-python applied on the bug

- Several instances of a crash while reading /proc/slabinfo and other operation in SLAB due to broken lists of slab structures
- In several dumps, the minimal corruption of a doubly-linked list allowed to detect a corrupted pointer and compare it with an expected value
- Others were corrupted much more because they didn’t panic soon enough and the it was hard to point to the initial corruption
crash-python applied on the bug

- Several instances of a crash while reading /proc/slabinfo and other operation in SLAB due to broken lists of slab structures.
- In several dumps, the minimal corruption of a doubly-linked list allowed to detect a corrupted pointer and compare it with an expected value.
- Others were corrupted much more because they didn’t panic soon enough and the it was hard to point to the initial corruption.
The Pattern Emerges...

- Turns out all the “simple” corruptions changed a byte 2f to 5c, and affected a size-32 or size-64 slab object
- For example, back to the shmid case:

```bash
crash> eval -b 13008943 # the expected value
hexadecimal: c6802f
decimal: 13008943
octal: 61500057
   binary:0000000000000000000000000000000011000110100000000101111
   bits set: 23 22 18 17 15 5 3 2 1 0

crash> eval -b 13008988 # the wrong value
hexadecimal: c6805c
decimal: 13008988
octal: 61500134
   binary:0000000000000000000000000000000011000110100000000101100
   bits set: 23 22 18 17 15 6 4 3 2
```
If In Doubt, Try ASCII (kudos to Michal Hocko)

- 2f is ‘/’, 5c is ‘\’
- Was somebody rewriting Linux filesystem paths to Windows?
  - The CIFS module (Samba client) has a function for that – `convert_delimiter()`, called by e.g. `cifs_build_path_to_root()

```c
static inline void convert_delimiter(char *path, char delim)
{
    int i;
    char old_delim;

    if (path == NULL)
        return;

    if (delim == '/')
        old_delim = '\';
    else
        old_delim = '/';

    for (i = 0; path[i] != '\0'; i++) {
        if (path[i] == old_delim)
            path[i] = delim;
    }
}
```
cifs_build_path_to_root() (v3.0.101)

char *cifs_build_path_to_root(struct smb_vol *vol, struct cifs_sb_info *cifs_sb, 
struct cifs_tcon *tcon)
{
    int pplen = vol->prepath ? strlen(vol->prepath) : 0;
    int dfsplen;
    char *full_path = NULL;
...

dfsplen = strnlen(tcon->treeName, MAX_TREE_SIZE + 1);
...

full_path = kmalloc(dfsplen + pplen + 1, GFP_KERNEL);
if (full_path == NULL)
    return full_path;

if (dfsplen)
    strncpy(full_path, tcon->treeName, dfsplen);
strncpy(full_path + dfsplen, vol->prepath, pplen);
convert_delimiter(full_path, CIFS_DIR_SEP(cifs_sb));
full_path[dfsplen + pplen] = 0; /* add trailing null */
return full_path;
}
cifs_build_path_to_root() (v3.0.101)

```c
char *cifs_build_path_to_root(struct smb_vol *vol, struct cifs_sb_info *cifs_sb,
                                struct cifs_tcon *tcon)
{
    int pplen = vol->prepath ? strlen(vol->prepath) : 0;
    int dfsplen;
    char *full_path = NULL;
    ...
    dfsplen = strnlen(tcon->treeName, MAX_TREE_SIZE + 1);
    ...
    full_path = kmalloc(dfsplen + pplen + 1, GFP_KERNEL);
    if (full_path == NULL)
        return full_path;
    if (dfsplen)
        strncpy(full_path, tcon->treeName, dfsplen);
    strncpy(full_path + dfsplen, vol->prepath, pplen);
    convert_delimiter(full_path, CIFS_DIR_SEP(cifs_sb));
    full_path[dfsplen + pplen] = 0; /* add trailing null */
    return full_path;
}
```

`strlen()` doesn't count trailing null
cifs_build_path_to_root() (v3.0.101)

```c
char *cifs_build_path_to_root(struct smb_vol *vol, struct cifs_sb_info *cifs_sb,
                               struct cifs_tcon *tcon)
{
    int pplen = vol->prepath ? strlen(vol->prepath) : 0;
    int dfsplen;
    char *full_path = NULL;
    ...
    dfsplen = strnlen(tcon->treeName, MAX_TREE_SIZE + 1);
    ...
    full_path = kmalloc(dfsplen + pplen + 1, GFP_KERNEL);
    if (full_path == NULL)
        return full_path;
    if (dfsplen)
        strncpy(full_path, tcon->treeName, dfsplen);
    strncpy(full_path + dfsplen, vol->prepath, pplen);
    convert_delimiter(full_path, CIFS_DIR_SEP(cifs_sb));
    full_path[dfsplen + pplen] = 0; /* add trailing null */
    return full_path;
}
```

`strlen()` doesn’t count trailing null

Neither this one
cifs_build_path_to_root() (v3.0.101)

```c
char *cifs_build_path_to_root(struct smb_vol *vol, struct cifs_sb_info *cifs_sb,
                              struct cifs_tcon *tcon)
{
    int pplen = vol->prepath ? strlen(vol->prepath) : 0;
    int dfsplen;
    char *full_path = NULL;
    ...
    dfsplen = strnlen(tcon->treeName, MAX_TREE_SIZE + 1);
    ...
    full_path = kmalloc(dfsplen + pplen + 1, GFP_KERNEL);
    if (full_path == NULL)
        return full_path;
    if (dfsplen)
        strncpy(full_path, tcon->treeName, dfsplen);
    strncpy(full_path + dfsplen, vol->prepath, pplen);
    convert_delimiter(full_path, CIFS_DIR_SEP(cifs_sb));
    full_path[dfsplen + pplen] = 0; /* add trailing null */
    return full_path;
}
```

`strlen()` doesn’t count trailing null

Neither this one

+ 1 makes space for it, OK
```c
char *cifs_build_path_to_root(struct smb_vol *vol, struct cifs_sb_info *cifs_sb,
                              struct cifs_tcon *tcon)
{
    int pplen = vol->prepath ? strlen(vol->prepath) : 0;
    int dfsplen;
    char *full_path = NULL;
    ...
    dfsplen = strlen(tcon->treeName, MAX_TREE_SIZE + 1);
    ...
    full_path = kmalloc(dfsplen + pplen + 1, GFP_KERNEL);
    if (full_path == NULL)
        return full_path;
    if (dfsplen)
        strncpy(full_path, tcon->treeName, dfsplen);
    strncpy(full_path + dfsplen, vol->prepath, pplen);
    convert_delimiter(full_path, CIFS_DIR_SEP(cifs_sb));
    full_path[dfsplen + pplen] = 0; /* add trailing null */
    return full_path;
}
```

- `strlen()` doesn’t count trailing null
- Neither this one
- + 1 makes space for it, OK
- `strncpy()` copies trailing null but the given length excludes the byte where null is
cifs_build_path_to_root() (v3.0.101)

```c
char *cifs_build_path_to_root(struct smb_vol *vol, struct cifs_sb_info *cifs_sb,
                           struct cifs_tcon *tcon)
{
    int pplen = vol->prepath ? strlen(vol->prepath) : 0;
    int dfsplen;
    char *full_path = NULL;

    dfsplen = strnlen(tcon->treeName, MAX_TREE_SIZE + 1);

    full_path = kmalloc(dfsplen + pplen + 1, GFP_KERNEL);
    if (full_path == NULL)
        return full_path;

    if (dfsplen)
        strncpy(full_path, tcon->treeName, dfsplen);
    strncpy(full_path + dfsplen, vol->prepath, pplen);
    convert_delimiter(full_path, CIFS_DIR_SEP(cifs_sb));
    full_path[dfsplen + pplen] = 0; /* add trailing null */
    return full_path;
}
```

- `strlen()` doesn’t count trailing null
- Neither this one
- + 1 makes space for it, OK
- `strncpy()` copies trailing null but the given length excludes the byte where null is
- too late, `convert_delimiter()` has already done the damage
The Bug’s (Upstream) History

• The bug was introduced upstream probably in 2011, kernel 3.1 (the commit was then backported to stable 3.0.x)
• The bug was fixed upstream in 2012, kernel 3.8, unknowingly (the commit intends to fix something else, much less critical)
  • This is a very common experience...
  • Upstream stable 3.2 was still maintained and vulnerable back in 2018, but CIFS maintainers didn’t respond to my report...
• Customer reported it only in 2017 – 6 years old bug
  • Needs a specific CIFS client/server settings, including frequent reconnect
  • Maybe it was reported previously, but happened just once and went unfixed
• Soon after we fixed our kernel, a different customer (with unpatched kernel) suddenly reported the same bug
  • Also a very common experience
  • Actually surprised that there was just one...
Debug prints

- `printk()` - send text to console/dmesg...
  - Including loglevels, from debugging to emergency
    - `printk(KERN_ERR “msg”), pr_err(), dev_err()`
- Correct implementation surprisingly nontrivial
  - Locking of buffers – what about printing from NMI?
  - Flooding slow consoles – printing task stalled
  - Timestamping/ordering from multiple CPUs
  - Prioritizing important info on panic
- Major rewrite addressing the above is ongoing for years now
- Printing very early during boot – earlyprintk setup needed
- `trace_printk()` – simpler, but output has to be captured later from the trace buffers
Dynamic debug prints

- The lowest level messages are actually compiled out when using `pr_debug()` and `dev_dbg()` wrappers
  - Unless `#define DEBUG` is active when compiling the file
  - Or `CONFIG_DYNAMIC_DEBUG` (dyndbg) is enabled
- With dyndbg, debug messages can be switched on/off at runtime via simple query language
  - `/sys/kernel/debug/dynamic_debug/control` or boot/modprobe parameters
  - Module, file, function, line (range), format string granularity
  - Flags to include func/line/module/thread id when printing
- Switching on/off uses live code patching (static keys) to minimize runtime impact (still, around 2% text size impact)
  - Ftrace uses the same mechanism for tracepoints
Live kernel debugging - /proc/kcore

- /proc/kcore enabled by CONFIG_PROC_KCORE
  - Provides virtual ELF “core dump” file
  - Usable by gdb, crash, drgn for read-only inspection
  - Printing values of global variables
  - Inspecting structures the same way as in a crash dump
- /dev/kmem – gone in 5.13 – could have bee configured read/write
  - crash can set variables and modify structures
- For full live debugging, we would need also to control execution, which is much trickier
  - Provide a server for gdb client that doesn't rely on the rest of the kernel functionality
Live kernel debugging - kgdb

• kgdb was merged in 2.6.26 (2008)
• Provides a server for remote gdb client
  • Over serial port – CONFIG_KGDB_SERIAL_CONSOLE
  • Over network using NETPOLL – not mainline (KDBoE)
• Enable on server
  • Boot with kgdboc=ttyS0,115200
  • echo g > /proc/sysrq-trigger or kgdbwait boot param
• Use from a client
  • % gdb ./vmlinux
  • (gdb) set remotebaud 115200
  • (gdb) target remote /dev/ttyS0
  • Allows limited gdb debugging similar to a userspace program
Live kernel debugging - kdb

- kdb is a frontend for kgdb that runs in the debugged kernel (no need for other client) – since 2.6.35 (2010)
- Provides a shell accessed via serial terminal, with optional PS/2 keyboard support
  - Enabled same way as the kgdb server
  - Switch between kdb/kgdb by $3#33 and kgdb
- Provides some kernel-specific commands not available in pure gdb
  - lsmod, ps, ps A, summary, bt, dmesg, go, help
    - Some can be executed from gdb – monitor help
    - Out of tree discontinued version seemed to be more capable
- KMS console support was proposed, but dropped
Live debugging - User-Mode Linux (UML)

- Special pseudo-hardware architecture
  - Otherwise compatible with the target architecture
- Running Linux kernel as a user space process
  - Originally a virtualization effort
- Useful for debugging and kernel development
  - A plain standard gdb can be used to attach to the running kernel
  - Guest threads are threads of the UML process
    - Slightly more complicated to follow processes
- Recommended environment for running KUnit tests
Magic SysRq hot keys

• Operator's intervention to the running system
  • For dealing with hangs or security issues
• Can be enabled/disabled by /proc/sys/kernel/sysrq
  • Alt+SysRq+H – show help
  • Invoke crash, reboot, shutdown, kill processes, OOM killer
  • Reset nice level of all real-time processes
  • Sync, remount read-only, freeze filesystems
  • Dump registers, tasks, stacks, memory stats, locks taken, armed timers, sleeping tasks, ftrace buffer
  • **Raising Elephants Is So Utterly Boring** or **Reboot Even If System Utterly Broken**
    • Raw keyboard, Send SIGTERM to all processes, Send SIGKILL to all processes, Sync data to disk, Remount all filesystems read-only, Reboot
• Can be activated also from console (/proc/sysrq-trigger) or via network
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
- Useful when hunting a particular bug, but mainly for regression testing
- Some now intended also for production kernels, can be compiled in but inactive unless enabled with a boot parameter
  - Again, using static keys to minimize overhead when not enabled
Kernel debugging config options

- **DEBUG_LIST** – catch some list misuses, poisoning
- **DEBUG_VM** – enable VM_BUG_ON() checks
- **PAGE_OWNER** – track who allocated and freed which pages in order to find a memory leak or double free
- **DEBUG_PAGEALLOC** – unmap (or poison) pages after they are freed
- **DEBUG_SLAB** – detect some cases of double free, or use-after-free (by poisoning), buffer overflow (red-zoning)
- **SLUB_DEBUG** – the SLUB variant can enable/disable debugging at boot for individual caches
  - Extra sanity checks, poisoning, red zoning, alloc/free tracking, tracing
- **DEBUG_KMEMLEAK** – detect leaks with a conservative garbage collection based algorithm
- **KFENCE** – low-overhead sampling based detection of overflow, use-after-free, invalid-free for slab objects
Kernel debugging config options

- **KASAN** – Find out of bounds accesses and use-after-free bugs using shadow memory (~valgrind) or sw/hw tags
  - GENERIC - Instrument each access to check shadow memory
    - Cost is 1/8 memory and 3x slower performance, needs new enough GCC or Clang
  - SW_TAGS – embeds tags to pointers, checks by instrumentation
    - Only slab and page allocations, arm64 with Top Byte Ignore, Clang, 1/16 memory
  - HW_TAGS – arm64 with Memory Tagging Extension, checks by hardware
    - Also slab and page allocations, only reports first bug, then disables itself

- **UBSAN** – Find out presence of undefined behavior (per C standard)

- **KCSAN** – dynamic race detector, based on compile-time instrumentation
  - Detect situations with two plain memory accesses to same place, one write
  - Needs GCC or Clang 11+, inserts soft watchpoints and stalls
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 – a.k.a. “lockdep” - mechanism for online proving that deadlocks cannot happen and report that deadlock can occur before it actually does
- Various subsystem specific options that enable both KERN_DEBUG printk()'s and extra checks
Kernel Fuzzing

• Try to trigger bugs by exposing the program to various inputs (i.e. chains of syscalls in the case of kernel)
• trinity – mostly random syscalls and parameters, only avoids known invalid input (flags) to not waste time on it
• syzkaller – unsupervised coverage-guided fuzzer from Google
  • For Akaros, FreeBSD, Fuchsia, gVisor, Linux, NetBSD, OpenBSD, Windows.
  • More efficient in finding corner-cases, but needs instrumentation
  • Often can generate a short reproducer with the report
• syzbot - https://syzkaller.appspot.com/
  • CI for automated fuzzing, reporting and tracking of found bugs
  • Linux: 3078 fixed, 978 open, 5529 invalid
  • Often used with debug options enabled, such as KASAN, UBSAN, lockdep...
Kernel testing (CI) initiatives

- Developers can’t possibly test their code in all possible architectures and configurations
- Automated testing and reporting very useful for development (linux-next) and stabilization (rc versions)
- LKP (Linux Kernel Performance) a.k.a. 0-day bot by Intel – tests linux-next, developer git trees, patches on mailing lists, replies with bug reports, sometimes proposed fixes
- kernelci.org by Linaro – for various ARM SoCs
- “Hulk Robot” used in Huawei
Linux Kernel Static Analysis

- Sparse – semantic checker for types and locks relying on attributes
  - Types – bitwise, kernel, user, iomem
  - Locks – acquire, release, must_hold
- Smatch – built upon sparse, can report e.g. missing NULL checks, array overflow
- Coccinelle – allows finding code matching a pattern as well as changing it
- Coverity – proprietary static analysis tool, scans Linux for free, but limited access to results
Thank you.