linaro
Connect
2025

Finding Locking Bugs With Static Analysis Using Smatch

Dan Carpenter
dan.carpenter@linaro.org

Introduction: Smatch Static Analysis

- Open Source (GPL)
- https://github.com/error27/smatch
- Good flow analysis
- Cross Function analysis
- Mostly used on the Linux kernel

Introduction 1/5: Flow analysis

```
a = 1;
if (x & 12)
        a = 2;
__smatch_implied(a);
if (!(x & 12))
        return;
__smatch_implied(a);
```

Introduction 2/5: Cross Function Analysis

```
$ smdb functions rpmsg_endpoint_ops send
drivers/rpmsg/qcom_smd.c | (struct rpmsg_endpoint_ops)->send | qcom_smd_send
drivers/rpmsg/qcom_glink_native.c | (struct rpmsg_endpoint_ops)->send | qcom_glink_send
drivers/rpmsg/mtk_rpmsg.c | (struct rpmsg_endpoint_ops)->send | mtk_rpmsg_send
drivers/rpmsg/virtio_rpmsg_bus.c | (struct rpmsg_endpoint_ops)->send | virtio_rpmsg_send
```

Introduction 3/5: Cross Function Analysis

```
$ smdb where gpio_v2_line_attribute id

drivers/gpio/gpiolib-cdev.c  | gpio_desc_to_lineinfo       | (struct gpio_v2_line_attribute)->id | 3
```

Introduction 4/5: Caller Information

```
$ smdb radeon_vm_bo_set_addr | grep LOCK
drivers/gpu/drm/radeon/radeon_gem.c |  radeon_gem_va_ioctl   | LOCK | -2 | rbo->tbo.base.resv
drivers/gpu/drm/radeon/radeon_gem.c |  radeon_gem_va_ioctl   | LOCK | -2 | rbo->tbo.base.resv
drivers/gpu/drm/radeon/radeon_kms.c | radeon_driver_open_kms | LOCK |  0 |
rdev->ring_tmp_bo.bo->tbo.base.resv
```

Introduction 5/5: Returned States

```
$ smdb return_states radeon_vm_bo_set_addr | grep LOCK
radeon_vm_bo_set_addr | 180 | s32min-(-1),1-s32max| UNLOCK |  1 | $->bo->tbo.base.resv
radeon_vm_bo_set_addr | 181 | (-512),(-114),(-35) | UNLOCK |  1 | $->bo->tbo.base.resv
radeon_vm_bo_set_addr | 182 |                   0| UNLOCK |  1 | $->bo->tbo.base.resv
radeon_vm_bo_set_addr | 183 |                   0| UNLOCK |  1 | $->bo->tbo.base.resv
radeon_vm_bo_set_addr | 185 |       (-22),(-12)| UNLOCK |  1 | $->bo->tbo.base.resv
```

Basic Locking Check

Kernel: spin_lock(&e->lock);

Smatch: set_state_expr(my_id, expr, &lock);

(expr is "&e->lock")

Basic Locking Check

- Same for unlock
- The states are tracked automatically
- The state can also be &merged (both locked and unlocked)

Basic Locking Check (Summary)

1. Match the lock
2. Match the unlock
3. At the end of the function check the error paths to see if we forgot to drop the lock

Cross function Support

```
{"spin_lock",        LOCK,   spin_lock, 0, "$"},
{"spin_trylock",     LOCK,   spin_lock, 0, "$", &int_one, &int_one},
```

Cross function Support

Hundred of other functions:

```
{"dma_resv_lock",              LOCK,   mutex, 0, "$", &int_zero, &int_zero},
{"dma_resv_trylock",           LOCK,   mutex, 0, "$", &int_one, &int_one},
{"dma_resv_lock_interruptible", LOCK,   mutex, 0, "$", &int_zero, &int_zero},
{"dma_resv_unlock",            UNLOCK, mutex, 0, "$"},
```

Cross function Support

Rebuild the Cross Function Database over and over.

one() calls two() with the $->lock held

two() calls three() with the $->lock held

…

Problem #1 Recording locks/unlocks

Tracking the start state is tricky. It can mostly be inferred.

Problem #2 One lock with two names

lock(&foo->bar)

unlock(&something->foo->bar);

If both end in “foo->bar” then mark it as unlocked?

Problem #3 The code was a mess

Problem #4 Not extensible

New Modular Rewrite

- Module to handle transitions
- Module to record caller states
- Module to record return states
- Check to print “forgot to unlock” warnings
- Check to print “double lock” warnings
- Check to print “double unlock” warnings

Transitions

```
void add_lock/unlock_hook(locking_hook *hook)
{
    add_ptr_list(&lock_hooks, hook);
}
```

Uses information from the big lock table

```
select_return_states_hook(LOCK, &db_param_locked);
select_return_states_hook(UNLOCK, &db_param_unlocked);
```

Caller info table

Not actually used.

Handling Return States

Keep it simple!

- &locked -> &ignore
- &unlocked -> &ignore

We started &unlocked, we did exactly one lock, insert a lock in the database

One lock with two names

Create a new module which instead of tracking the variables it

tracks "(struct foo)->bar" locked/unlocked.

Rule: If we have two locks, one is locked and one is unlocked, and the lock_type says that nothing changed the let's say nothing changed.

Guard locks

scoped_cond_guard(mutex_intr, return -ERESTARTSYS, &profile_lock) {

{"class_device_destructor", UNLOCK, mutex, 0, "%s.mutex", NULL, NULL, &match_class_generic_unlock},

{"class_mutex_destructor", UNLOCK, mutex, 0, "%s", NULL, NULL, &match_class_generic_unlock},

Guard locks (Conditional locking)

db/kernel.return_fixes

class_write_lock_irqsave_lock_ptr 0-u64max[$0->lock] 4096-ptr_max[$0->lock]

Guard locks (Conditional locking)

Guard locks avoid a lot of locking bugs so we can mostly just ignore them.

#Lazy

#Confession

Improvements

- Extensible
- Cleaner code
- Much fewer states to deal with
- Easier to debug
- One lock with two names handled correctly
- Originally the code was biased towards unlocking
- Easy to create new small checks

Forgot to unlock on error path

```
spin_lock(&foo);

ret = frob();
if (ret)
        return ret;

spin_unlock(&foo);
```

Double Lock Bugs (Copy and Paste)

```
spin_lock(&foo);

....

spin_lock(&foo); // This was supposed to unlock
```

Double Unlock Bugs (goto out)

```
spin_unlock(&foo);
…
        goto out;
…
out:
        spin_unlock(&foo);
```

Lessons for Cross Function Analysis

- Module to handle transitions
- Module to record return states
- Module to record caller states
- Checks built on top of that

Uses:

- Reference counting
- Tracking user data
- Tracking places which disable IRQs

Further work

- Warning about double locks across function boundaries
- Tracking the relationship between the lock and the data it protects
- Lock ordering bugs


linaro
Connect
2025
Thank You!