Static Analysis for TOCTTU bugs

I. Case Study

A. *Case 1 (Suspicious)*

- Global Variable: acpi_gbl_next_history_index
- File: drivers/acpi/acpica/dbhistry.c
- Function: acpi_db_add_to_history
- Use Location 1: Line 90 Use as index
- Use Location 2: Line 94 Use as index
- Use Location 3: Line 99 Use as index
- Use Location 4: Line 100 Use as index
- Use Location 5: Line 104 Use as index
- Use Location 6: Line 108 Use as index
- Use Location 7: Line 117 Use as check
- Use Location 8: Line 125 Use as check
- Define Location 1: Line 124 Reset
- Define Location 2: Line 126 Increment
- Analysis;
  - Use/Def Analysis:
    - Use as array index and serve condition check at line 90; May cause undefined behavior when corrupted
    - Use as array index and serve string length calculation at line 94; May pollute the string length and further memory overflow
    - Use as array index and serve memory copy at line 108; May lead to arbitrary memory write
  - Lock Analysis
    - Reverse call sequence: acpi_db_add_to_history (No lock) → acpi_db_command_dispatch (No lock) → acpi_db_user_commands (No lock) → acpi_db_execute_thread (No lock) → acpi_os_execute (Being started as a new thread; No lock)
    -
  - Parallel Access Analysis:
  - Exploitability Analysis

B. *Case 2 (Suspicious)*

- Global Variable: he_devs
- File: drivers/atm/he.c
- Function: he_init_one
- Use Location 1: Line 395 Use as check
- Use Location 2: Line 396 Use as node on linked list
- Define Location 2: Line 397 define as the new head of linked list
- Analysis;
  - Use/Def Analysis:
    - Use as the current head of a linked list at Line 395
    - Update it to the new head the current linked list at Line 396
  - Lock Analysis
    - Reverse call sequence: he_init_one (No lock) → used as the probe handler of the ”he” pci driver.
  - Parallel Access Analysis:
  - Exploitability Analysis: If two threads can reach Line 397 at the same time, then only one thread can update he_devs (aka, the head of the linked list). As a consequence, one newly allocated node is not linked in and can never be freed (memory leakage)?

C. *Case 3 (Unknown; Looks less interesting)*

- Global Variable: iadev_count
- File: drivers/atm/iphase.c

D. *Case 4 (Similar to Case 2)*

- Global Variable: ia_boards
- File: drivers/atm/iphase.c
- Function: ia_init_one

E. *Case 5 (Similar to Case 3)*

- Global Variable: num_cards
- File: drivers/atm/nicstar.c

F. *Case 6 (Not very interesting)*

- Global Variable: fpga_upgrade
- File: drivers/atm/solos-pci.c

G. *Case 7 (Similar to Case 2)*

- Global Variable: eni_boards
- File: drivers/atm/eni.c
- Function: eni_init_one

II. False Positive Filters for A Pair of Uses

A. *Reachability*

The pair of uses are in at least one execution path (reachable in the CFG)

B. *Lock*

The pair of use are not covered by a set of lock/unlock

C. *Define*

At least one define can be affected by external input

D. *Initialization*

The pair of uses are not in an initialization function (Check if a function is placed into the .init.text section)

The define is not in an initialization function (Check if a function is placed into the .init.text section)

III. Counter Examples

A. *Class 1: lock is added by a parent (still a dominator)*

```
int device_attach(struct device *dev)
{
        int ret = 0;

        device_lock(dev);
        if (dev->driver) {
                if (klist_node_attached(&dev->p->knode_driver)) {
                        ret = 1;
                        goto out_unlock;
                }
                ret = device_bind_driver(dev);
                if (ret == 0)
                        ret = 1;
                else {
                        dev->driver = NULL;
                        ret = 0;
                }
        } else {
                ret = bus_for_each_drv(dev->bus, NULL, dev, __device_attach);
                pm_request_idle(dev);
        }
out_unlock:
        device_unlock(dev);
        return ret;
}
```

B. *Class 2: The function is only called by init/exit function (in principle, same as init/exit function)*

```
int xs_init(void)
{
        int err;
        struct task_struct *task;

        INIT_LIST_HEAD(&xs_state.reply_list);
        spin_lock_init(&xs_state.reply_lock);
        init_waitqueue_head(&xs_state.reply_waitq);

        mutex_init(&xs_state.request_mutex);
        mutex_init(&xs_state.response_mutex);
        mutex_init(&xs_state.transaction_mutex);
        init_rwsem(&xs_state.watch_mutex);
        atomic_set(&xs_state.transaction_count, 0
        init_waitqueue_head(&xs_state.transaction_

        /* Initialize the shared memory rings to
        err = xb_init_comms();
        if (err)
                return err;

        task = kthread_run(xenwatch_thread, NULL,
        if (IS_ERR(task))
                return PTR_ERR(task);
        xenwatch_pid = task->pid;

        task = kthread_run(xenbus_thread, NULL, ”
        if (IS_ERR(task))
                return PTR_ERR(task);

        return 0;
}
```

C. *Class 3: Lock/unlock during the atomicity operation is legit*

```
static void *read_reply(enum xsd_sockmsg_type *ty
{
        struct xs_stored_msg *msg;
        char *body;

        spin_lock(&xs_state.reply_lock);

        while (list_empty(&xs_state.reply_list))
                spin_unlock(&xs_state.reply_lock)
                /* XXX FIXME: Avoid synchronous w
                wait_event(xs_state.reply_waitq,
                           !list_empty(&xs_state.
                spin_lock(&xs_state.reply_lock);
        }

        msg = list_entry(xs_state.reply_list.next
                         struct xs_stored_msg, li
        list_del(&msg->list);

        spin_unlock(&xs_state.reply_lock);

        *type = msg->hdr.type;
        if (len)
                *len = msg->hdr.len;
        body = msg->u.reply.body;

        kfree(msg);

        return body;
}
```

D. *Class 4: I have no idea here...*

```
void osduld_unregister_test(unsigned ioctl)
```

```
{
        if (ioctl == g_test_ioctl) {
                g_test_ioctl = 0;
                g_do_test = NULL;
        }
}
```

IV. NOTE

hfi1_user_sdma_process_request

does not check size after copy from user.