The android vulnerability discovery in SoC

Yu Pan and Yang Dai

About us

- Security researcher of Vulpecker Team@360
- Android Vulnerabilities research
- Focus on kernel & driver

Numerous vulnerabilities,including Google, Qualcomm, MediaTek,NVIDIA,Samsung,Huawei,XiaoMi,etc.

Agenda

Dynamic fuzzing tool

- Fuzzing tool’s map
- Anatomy

Static analysis

- Driver model analysis
- Pattern matching

Conclusion

Agenda

Dynamic fuzzing tool

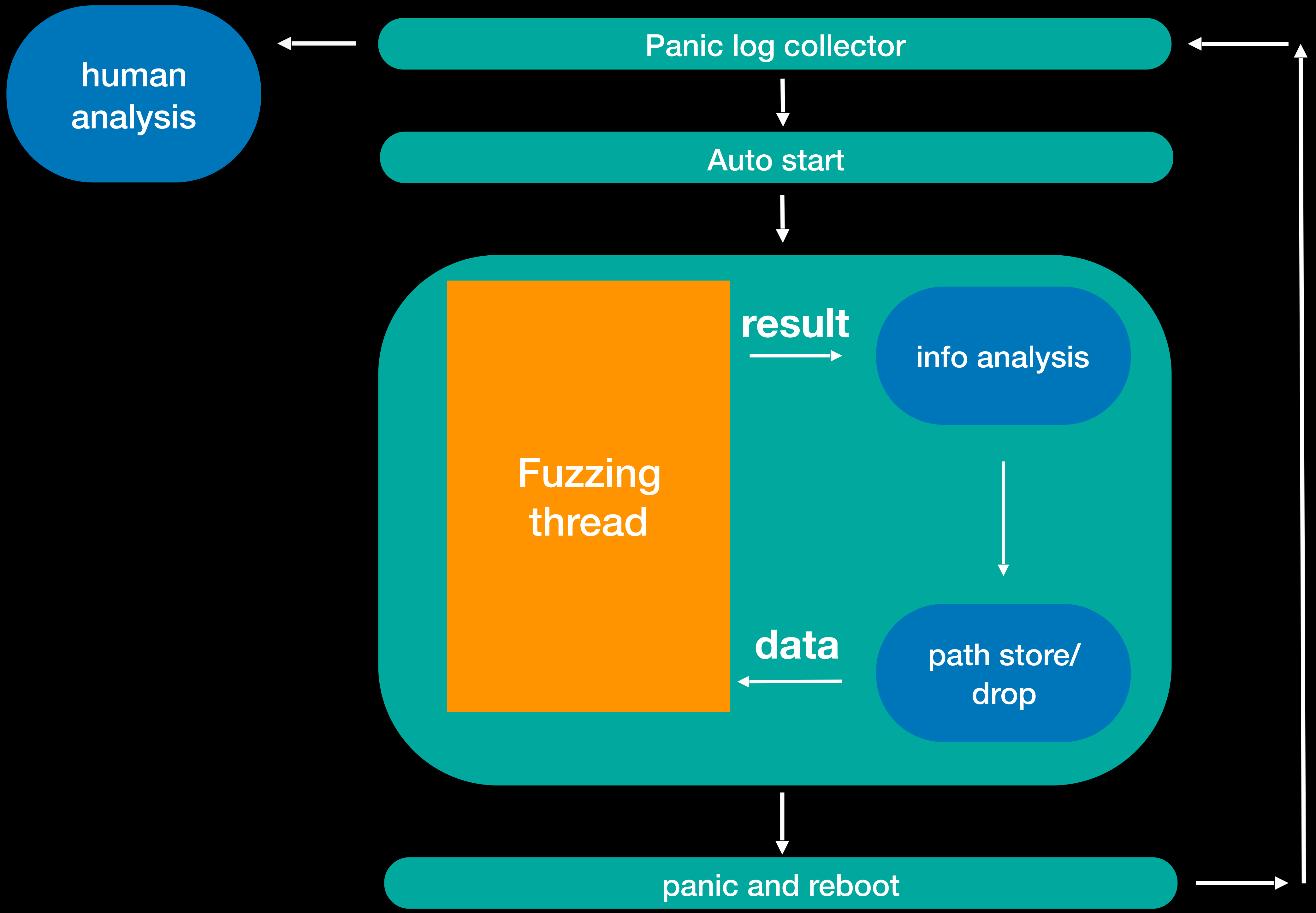
- Fuzzing tool’s map
- Anatomy

Static analysis

- Driver model analysis
- Pattern matching

Conclusion

Fuzzing tool's map



Agenda

Dynamic fuzzing tool

- Fuzzing tool’s map
- Anatomy

Static analysis

- Driver model analysis
- Pattern matching

Conclusion

Anatomy

▶The genetic algorithm:

The genetic algorithm is performed according to the execution results,and is used to determine whether the parameters of test cases should be preserved or transformed.

▶Kernel patch assistant:

We add the analysis and record for the test path by modifying the source code and increased the judgment of the fuzz return information to guide the next test.

Agenda

Dynamic fuzzing tool

- Fuzzing tool’s map
- Anatomy

Static analysis

- Driver model analysis
- Pattern matching

Conclusion

Driver model analysis (ASoC)

- Introduction
- Architecture
- AttackSurface
- Case

Driver model analysis (ASoC)

- Introduction
- Architecture
- AttackSurface
- Case

▶ALSA(Advanced Linux Sound Architecture)

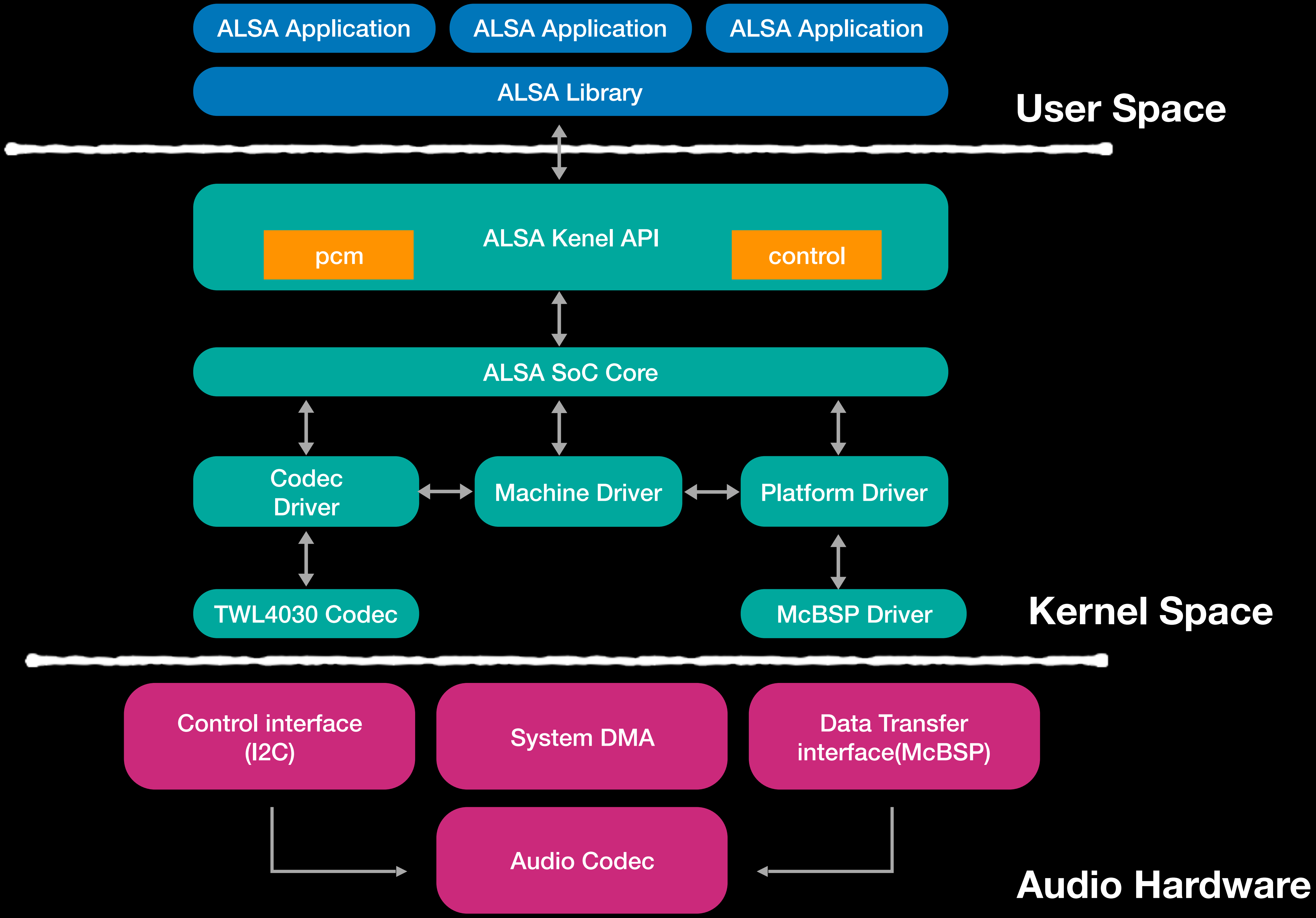
ALSA provides audio-related support to the Linux kernel.

▶ASoC(ALSA System on Chip)

ASoC is a Linux kernel subsystem created to provide better ALSA support for system-on-chip and portable audio codecs.

Driver model analysis (ASoC)

- Introduction
- Architecture
- AttackSurface
- Case


ALSA Application
ALSA Application
ALSA Application
ALSA Library
User Space
ALSA Kenel API
pcm
control
ALSA SoC Core
Codec Driver
Machine Driver
Platform Driver
TWL4030 Codec
McBSP Driver
Kernel Space
Control interface (I2C)
System DMA
Data Transfer interface(McBSP)
Audio Codec
Audio Hardware

Documentation/sound/alsa/soc/overview.txt

▶Machine driver:

The machine driver handles any machine specific controls and audio events (e.g. turning on an amp at start of playback).

▶Codec Driver :

The codec driver is platform independent and contains audio controls, audio interface capabilities, codec DAPM definition and codec I/O functions.

▶Platform driver:

The platform driver contains the audio DMA engine and audio interface drivers (e.g. I2S, AC97, PCM) for that platform.

Driver model analysis (ASoC)

- Introduction
- Architecture
- AttackSurface
- Case

▶Control:

Control is the interface that alsa provides for controlling the sound card for user space programs.

```
struct snd_kcontrol_new {
	snd_ctl_elem_iface_t iface;  /* interface identifier */
	unsigned int device;         /* device/client number */
	unsigned int subdevice;              /* subdevice (substream) number */
	const unsigned char *name;/* ASCII name of item */
	unsigned int index;          /* index of item */
	unsigned int access;         /* access rights */
	unsigned int count;          /* count of same elements */
	snd_kcontrol_info_t *info;
	snd_kcontrol_get_t *get;
	snd_kcontrol_put_t *put;
	union {
		snd_kcontrol_tlv_rw_t *c;
		const unsigned int *p;
	} tlv;
	unsigned long private_value;
};
```

Driver model analysis (ASoC)

- Introduction
- Architecture
- AttackSurface
- Case

Qualcomm SoC

```
static long snd_ctl_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    struct snd_ctl_file *ctl;
    struct snd_card *card;
    struct snd_kctl_ioctl *p;
    void __user *argp = (void __user *)arg;
    int __user *ip = argp;

    switch (cmd) {
    case SNDRV_CTL_IOCTL_ELEM_INFO:
            return snd_ctl_elem_info_user(ctl, argp);
    case SNDRV_CTL_IOCTL_ELEM_READ:
            return snd_ctl_elem_read_user(card, argp);
    case SNDRV_CTL_IOCTL_ELEM_WRITE:
            return snd_ctl_elem_write_user(ctl, argp);
    }
```

Qualcomm SoC

```
static int snd_ctl_elem_write(struct snd_card *card, struct snd_ctl_file *file,
                              struct snd_ctl_elem_value *control)
{
        kctl = snd_ctl_find_id(card, &control->id);
        if (kctl == NULL) {
                result = -ENOENT;
        } else {

                } else {
                        snd_ctl_build_ioff(&control->id, kctl, index_offset);
                        result = kctl->put(kctl, control);
                }

        return result;
}
```

Qualcomm SoC

```
static int msm_routing_lsm_mux_put(struct snd_kcontrol *kcontrol,
                                   struct snd_ctl_elem_value *ucontrol)
{

        int mux = ucontrol->value.enumerated.item[0];

        if (ucontrol->value.integer.value[0]) {
                lsm_mux_slim_port = ucontrol->value.integer.value[0];
                snd_soc_dapm_mux_update_power(widget, kcontrol, mux, e);
        } else {
                snd_soc_dapm_mux_update_power(widget, kcontrol, mux, e);
                lsm_mux_slim_port = ucontrol->value.integer.value[0];
        }

        return 0;
}
```

Qualcomm SoC

```
static int soc_dapm_mux_update_power(struct snd_soc_dapm_widget *widget,
                                     struct snd_kcontrol *kcontrol, int mux, struct soc_enum *e)
{
        if (!(strcmp(path->name, e->texts[mux]))) {

                path->connect = 1;
                dapm_mark_dirty(path->source, "mux connection");
        } else {
                if (path->connect)
                        dapm_mark_dirty(path->source,
                                        "mux disconnection");
                path->connect = 0;
        }
    }

    return found;
}
```

Qualcomm SoC

```
static int soc_dapm_mux_update_power(struct snd_soc_dapm_widget *widget,
                struct snd_kcontrol *kcontrol, int mux, struct soc_enum *e)
{
        if (!(strcmp(path->name, e->texts[mux]))) {
                path->connect = 1;
                dapm_mark_dirty(path->source, "mux connection");
        } else {
                if (path->connect)
                        dapm_mark_dirty(path->source,
                                "mux disconnection");
                path->connect = 0;
        }
    }

    return found;
}
```

Out Of Bound Access

OOB & Overflow vulnerability in ASoC

common :

Samsung	Qualcomm	Xiaomi
5	1	1

Driver model analysis (Thermal)

- Introduction
- Architecture
- AttackSurface
- Case

Driver model analysis (Thermal)

- Introduction
- Architecture
- AttackSurface
- Case

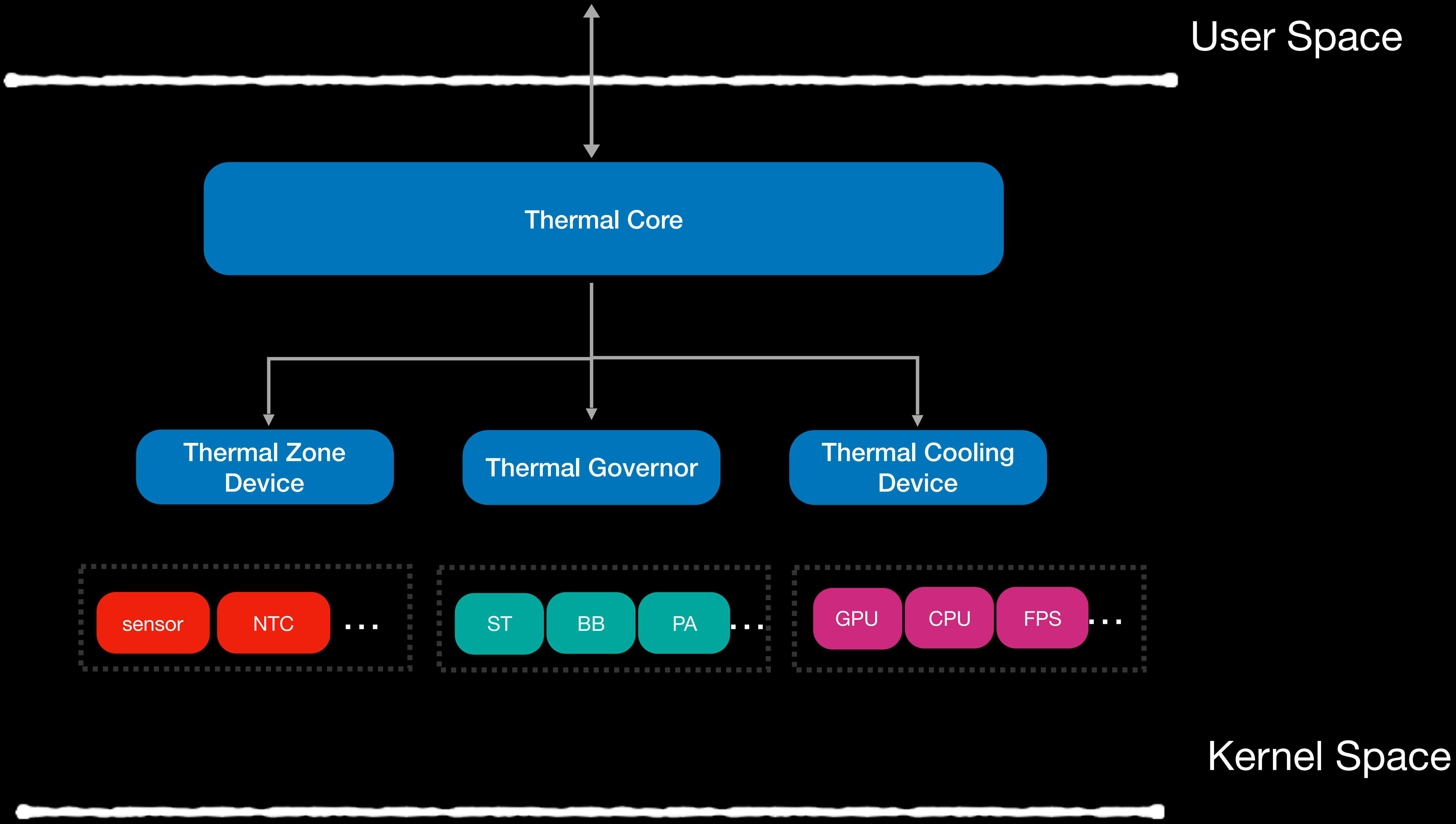
▶Thermal Framework

Linux manages the system’s temperature through its thermal framework. The framework includes thermal zones, thermal sensors,cooling devices, governors, trip points and thermal instances.

The thermal framework also exposes information to user-space applications through sysfs, a virtual filesystem for device and driver information provided by Linux.

Driver model analysis (Thermal)

- Introduction
- Architecture
- AttackSurface
- Case


User Space
Thermal Core
Thermal Zone Device
Thermal Governor
Thermal Cooling Device
sensor
NTC
...
ST
BB
PA
...
GPU
CPU
FPS
...
Kernel Space

▶Thermal Zone Device:

The thermal zone device includes a thermal sensor and multiple cooling devices.It represents a region managed by thermal framework.

▶Thermal Governor:

The thermal governor determines cooling policy.

▶Thermal Cooling Device:

The thermal cooling device is actual functional units for cooling down the thermal zone.

Driver model analysis (Thermal)

- Introduction
- Architecture
- AttackSurface
- Case

```
static struct thermal_cooling_device *
__thermal_cooling_device_register(struct device_node *np,
                char *type, void *devdata,
                const struct thermal_cooling_device_ops *ops)
{
        INIT_LIST_HEAD(&cdev->thermal_instances);
        cdev->np = np;
        cdev->ops = ops;
        cdev->class = &thermal_class;
        cdev->groups = cooling_device_attr_groups;
        cdev->devdata = devdata;
        dev_set_name(&cdev->device,”cooling_device%d”,cdev->id);

        result = device_create_file(&cdev->device, &dev_attr_cdev_type);

        result = device_create_file(&cdev->device, &dev_attr_max_state);

        result = device_create_file(&cdev->device, &dev_attr_cur_state);
}
```

```
static struct device_attribute dev_attr_cdev_type =
__ATTR(type, 0444, thermal_cooling_device_type_show, NULL);
static DEVICE_ATTR(max_state, 0444,
                   thermal_cooling_device_max_state_show, NULL);
static DEVICE_ATTR(cur_state, 0644,
                   thermal_cooling_device_cur_state_show,
                   thermal_cooling_device_cur_state_store);


#define DEVICE_ATTR(_name, _mode, _show, _store) \
    struct device_attribute dev_attr_##_name = __ATTR(_name, _mode, _show, _store)


#define __ATTR(_name, _mode, _show, _store) {           \
  .attr = {.name = __stringify(_name),           \
     .mode = VERIFY_OCTAL_PERMISSIONS(_mode) },    \
  .show   = _show,                   \
  .store  = _store,                  \
}
```

```c
static ssize_t
thermal_cooling_device_cur_state_store(struct device *dev,
                struct device_attribute *attr,
                const char *buf, size_t count)
{
    struct thermal_cooling_device *cdev = to_cooling_device(dev);
    unsigned long state;
    int result;

    if (!sscanf(buf, "%ld\n", &state))
        return -EINVAL;

    if ((long)state < 0)
        return -EINVAL;

    result = cdev->ops->set_cur_state(cdev, state);
    if (result)
        return result;
    return count;
}
```

```
#define DEVICE_ATTR(_name, _mode, _show, _store) \
	struct device_attribute dev_attr_##_name = __ATTR(_name, _mode, _show, _store)

struct device_attribute {
    struct attribute    attr;
    ssize_t (*show)(struct device *dev, struct device_attribute *attr,char *buf);
    ssize_t (*store)(struct device *dev, struct device_attribute *attr,const char *buf,
    size_t count);
};

static DEVICE_ATTR(max_state, 0444,
                thermal_cooling_device_max_state_show,  NULL);
static DEVICE_ATTR(cur_state, 0644,
                thermal_cooling_device_cur_state_show,
                thermal_cooling_device_cur_state_store);
```

Driver model analysis (Thermal)

- Introduction
- Architecture
- AttackSurface
- Case

Samsung S8 Thermal

```
static ssize_t mfc_show_data(struct device *dev,
                             struct device_attribute *attr,
                             char *buf)
{
      for (i = 0; i < charger->size; i++) {
            if (mfc_reg_read(charger->client, charger->addr+i, &data) < 0) {
                  dev_info(charger->dev,
                              "%s: read fail\n", __func__);
                  count += sprintf(buf+count, "addr: 0x%x read fail\n", charger->addr+i);
                  continue;
            }
            count += sprintf(buf+count, "addr: 0x%x, data: 0x%x\n", charger->addr+i,data);
      }
      return count;
}
```

Samsung S8 Thermal

```
static ssize_t mfc_show_data(struct device *dev,
                             struct device_attribute *attr,
                             char *buf)
{
      for (i = 0; i < charger->size ; i++) {
            if (mfc_reg_read(charger->client, charger->addr+i, &data) < 0) {
                  dev_info(charger->dev,
                              "%s: read fail\n", __func__);
                  count += sprintf(buf+count, "addr: 0x%x read fail\n", charger->addr+i);
                  continue;
            }
            count += sprintf(buf+count, "addr: 0x%x, data: 0x%x\n", charger->addr+i,data);
      }
      return count;
}
```

Samsung S8 Thermal

```
static ssize_t mfc_store_size(struct device *dev,
                              struct device_attribute *attr,
                              const char *buf, size_t count)
{
        struct power_supply *psy = dev_get_drvdata(dev);
        struct mfc_charger_data *charger = power_supply_get_drvdata(psy);
        int x;
        if (sscanf(buf, "%10d\n", &x) == 1) {
                charger->size = x;
        }
        return count;
}
```

Samsung S8 Thermal

```
static ssize_t mfc_show_data(struct device *dev,
                        struct device_attribute *attr,
                        char *buf)
{
    for (i = 0; i < charger->size ; i++) {
        if (mfc_reg_read(charger->client, charger->addr+i, &data) < 0) {
            dev_info(charger->dev,
                    "%s: read fail\n", __func__);
            count += sprintf(buf+count, "addr: 0x%x read fail\n", charger->addr+i);
            continue;
        }
        count += sprintf(buf+count, "addr: 0x%x, data: 0x%x\n", charger->addr+i,data);
    }
    return count;
}
```

buf Overflow

Tegra Thermal

```
static struct thermal_cooling_device *balanced_throttle_register(
                        struct tegra_balanced_throttle_ins *bthrot_ins,
                        char *type)
{
      bthrot_ins->cdev = thermal_of_cooling_device_register(
                                    bthrot_ins->np,
                                    type,
                                    bthrot_ins,
                                    &tegra_throttle_cooling_ops);

      sprintf(name, "%s_throttle_table", type);
      bthrot_ins->d_tab = debugfs_create_file(name, 0644,
                                    throttle_debugfs_root,
                                    bthrot_ins, &table_fops);
      return bthrot_ins->cdev;
}
```

```
static struct thermal_cooling_device_ops tegra_throttle_cooling_ops = {
 .get_max_state = tegra_throttle_get_max_state,
 .get_cur_state = tegra_throttle_get_cur_state,
 .set_cur_state = tegra_throttle_set_cur_state,
};

struct thermal_cooling_device_ops {
   int (*get_max_state) (struct thermal_cooling_device *, unsigned long *);
   int (*get_cur_state) (struct thermal_cooling_device *, unsigned long *);
   int (*set_cur_state) (struct thermal_cooling_device *, unsigned long);
   int (*get_requested_power)(struct thermal_cooling_device *,
                struct thermal_zone_device *, u32 *);
   int (*state2power)(struct thermal_cooling_device *,
             struct thermal_zone_device *, unsigned long, u32 *);
   int (*power2state)(struct thermal_cooling_device *,
             struct thermal_zone_device *, u32, unsigned long *);
};
```

CVE-2017-6274

```
static int
tegra_throttle_set_cur_state(struct thermal_cooling_device *cdev,
                    unsigned long cur_state)
{
    bthrot_ins->cur_state = cur_state;

    list_for_each_entry(bthrot_ins, &bthrot_list, node) {
        if (!bthrot_ins->cur_state)
            continue;

        throt_entry = &bthrot_ins->throt_tab[bthrot_ins->cur_state-1];
        for (i = 0; i < max_cap_clock; i++) {
            cur_throt_freq.cap_freqs[i] = min(
                    cur_throt_freq.cap_freqs[i],
                    throt_entry->cap_freqs[i]);
        }
    }
}
```

CVE-2017-6274

```
static int
tegra_throttle_set_cur_state(struct thermal_cooling_device *cdev,
                             unsigned long cur_state )
{
    bthrot_ins->cur_state = cur_state;

    list_for_each_entry(bthrot_ins, &bthrot_list, node)
        if (!bthrot_ins->cur_state)
            continue;

        throt_entry = &bthrot_ins->throt_tab[bthrot_ins->cur_state-1];
        for (i = 0; i < max_cap_clock; i++) {
            cur_throt_freq.cap_freqs[i] = min(
                    cur_throt_freq.cap_freqs[i],
                    throt_entry->cap_freqs[i]);
        }
    }
}
```

Arbitrary read

OOB & Overflow vulnerability in Thermal

common :

Samsung	Mtk	Nvidia
3	10	5

Agenda

Dynamic fuzzing tool

- Fuzzing tool’s map
- Anatomy

Static analysis

- Driver model analysis
- Pattern matching

Conclusion

Pattern matching

- Race condition in list
- Reference count overflow

Pattern matching

- Race condition in list
- Reference count overflow

race condition in list

- no lock protection on list operation

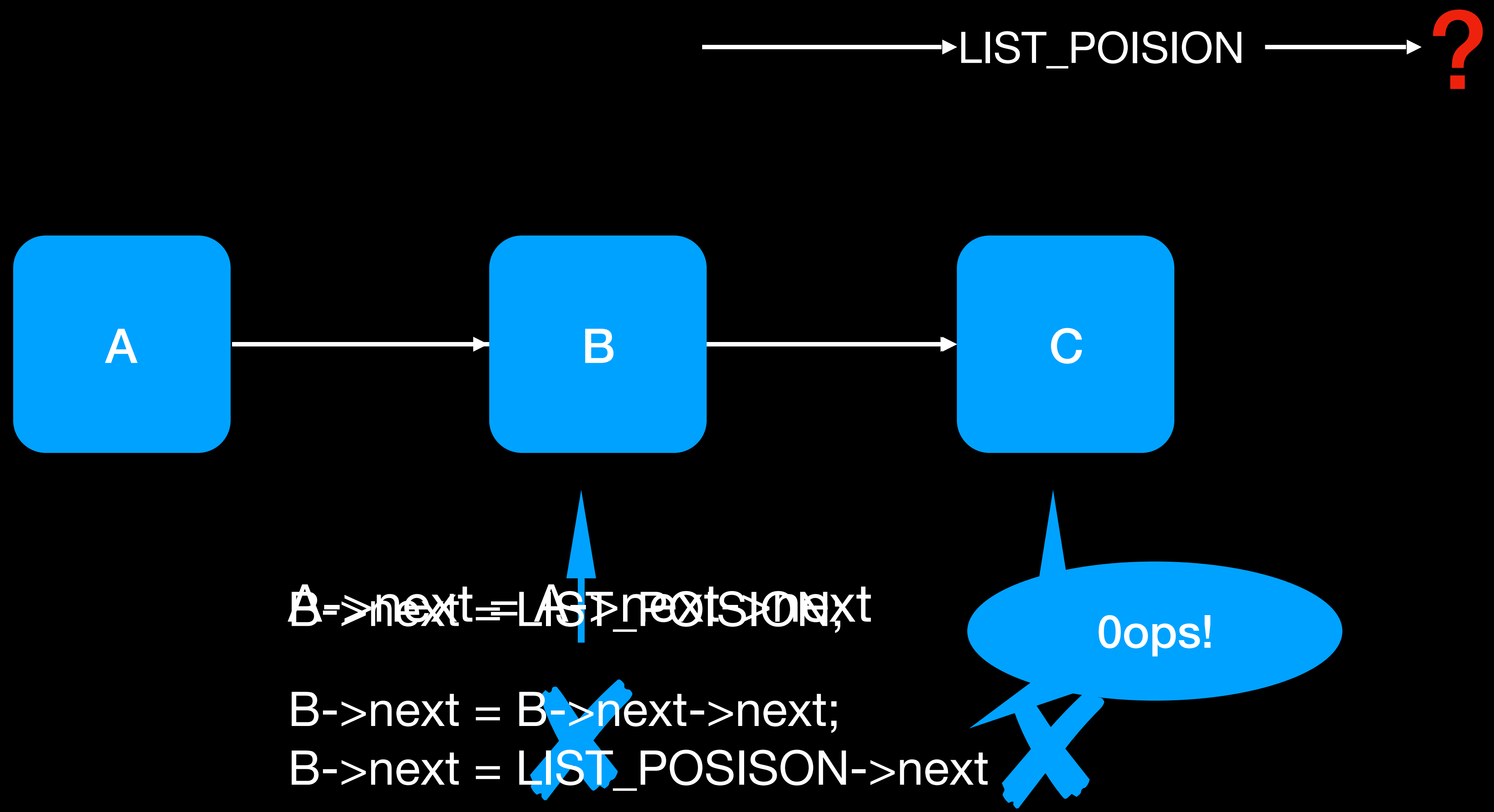
  simple but rare

- no lock protection on list node

  complex but common

race condition in list

simple :



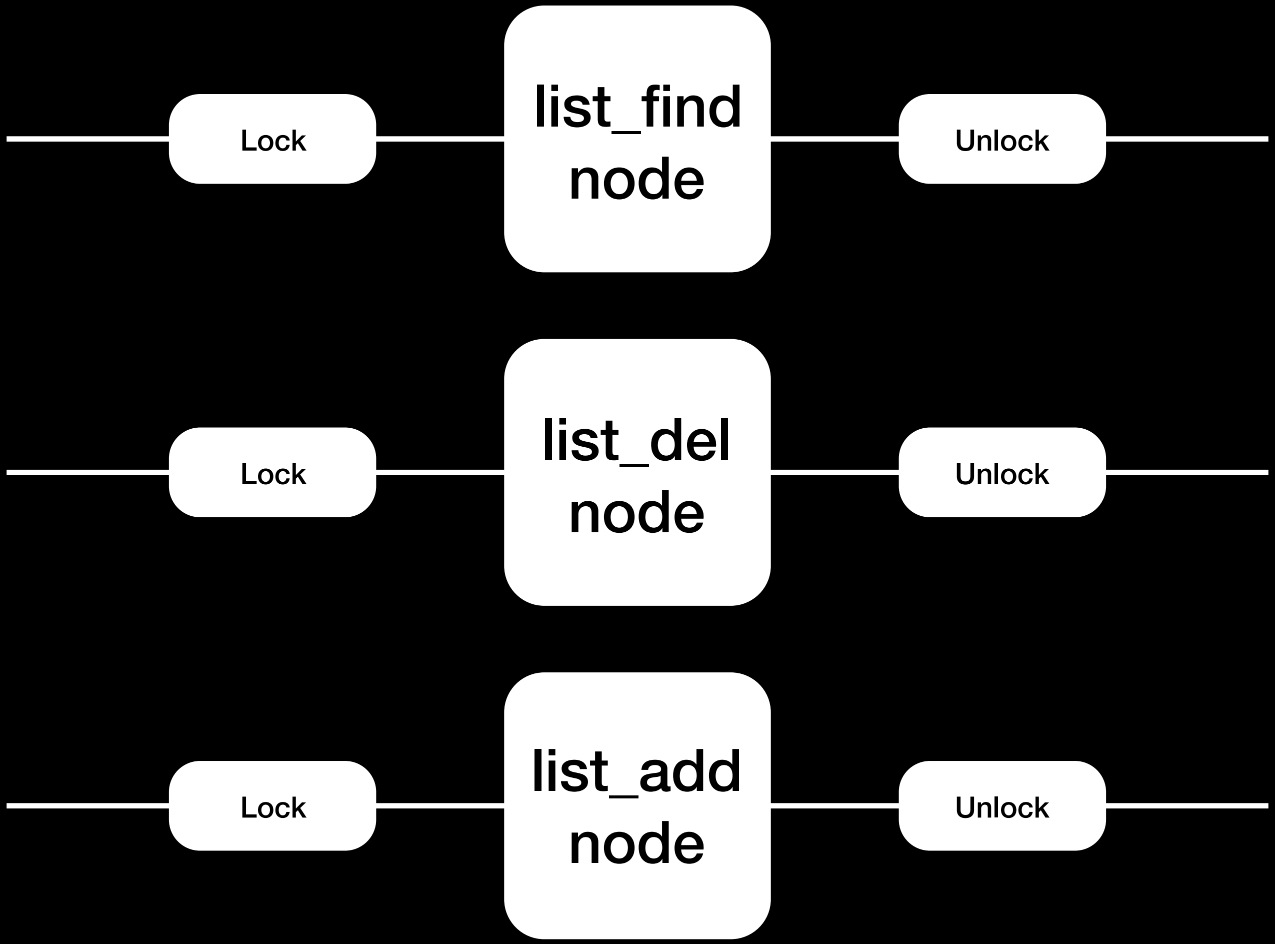
```
A->next = A->next->next
B->next = LIST_POISION;

B->next = B->next->next;
B->next = LIST_POSISON->next
```

```
B = A->next;
A->next = A->next->next;
B->next = LIST_POISION;
```

```
C = B->next;
B->next = B->next->next;
C->next = LIST_POISION;
```

race condition in list

To have lock protection on every list operation:

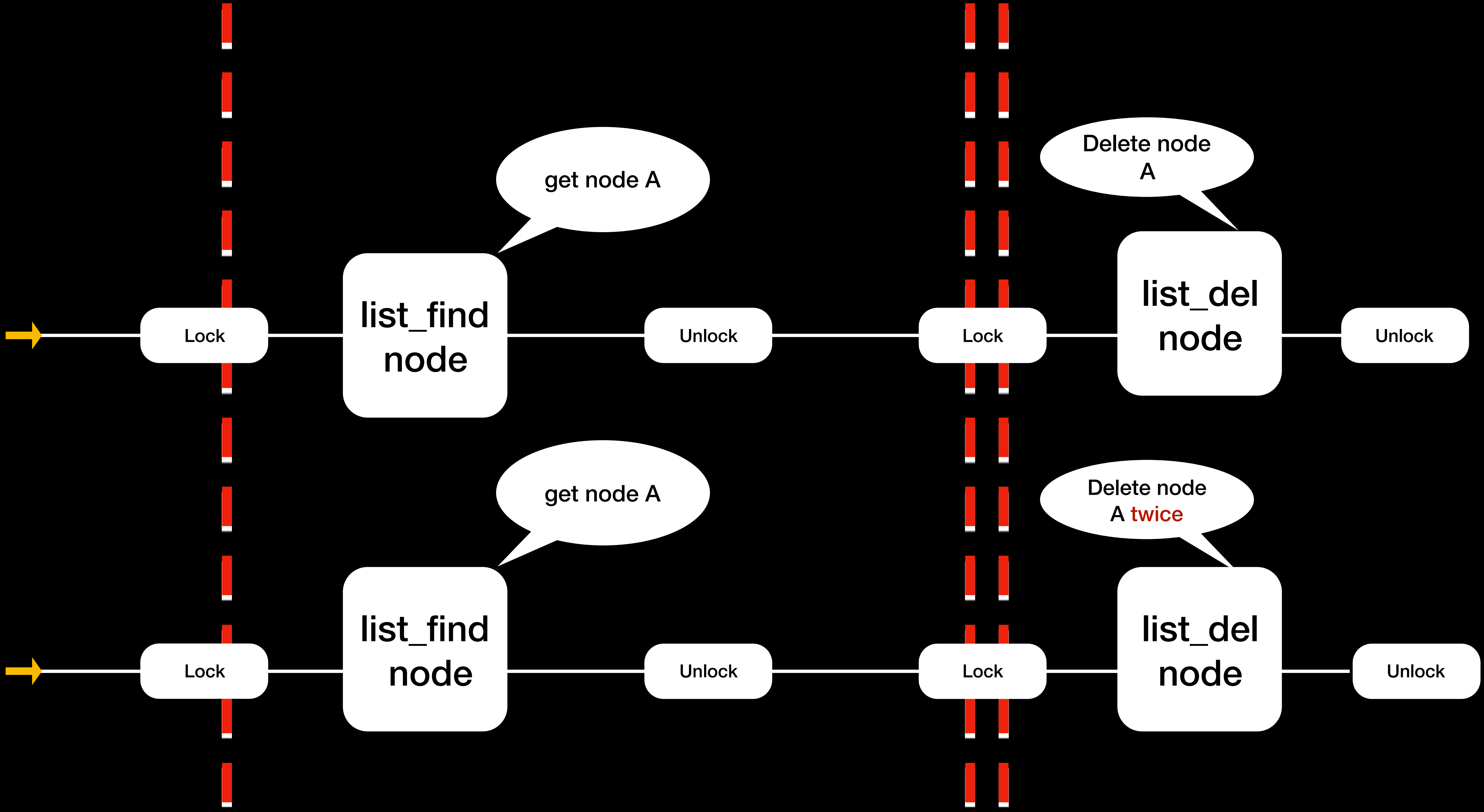


race condition in list

Is it safe now?

race condition in list

complex :



race condition in list

No lock protection on Node A !

Two threads are using the same node

race condition in list

common :

- Samsung s8 HDCP race
- Samsung s8 del_kek race
- …

race condition in list

Samsung s8 HDCP race

Lock for list operation :
list_find

```
struct hdcp_session_node *hdcp_session_list_find(…)
{…
    mutex_lock(&ss_list->ss_mutex);
    ss_head = &ss_list->hdcp_session_head;
    for (pos = ss_head->next; pos != ss_head && pos != NULL; pos = pos->next) {
        if (pos->ss_data->id == id) {
            mutex_unlock(&ss_list->ss_mutex);
            return pos;
        }
    }
    mutex_unlock(&ss_list->ss_mutex);

    return NULL;
}
```

race condition in list

Samsung s8 HDCP race

Lock for list operation :
list_del

```
void hdcp_session_list_del(...)
{
        if (!del_ent || !ss_list)
                return;

        mutex_lock(&ss_list->ss_mutex);
        del_ent->prev->next = del_ent->next;
        del_ent->next->prev = del_ent->prev;
        mutex_unlock(&ss_list->ss_mutex);
}
```

race condition in list

Samsung s8 HDCP race

No lock protection for ss_node

```
enum hdcp_result hdcp_session_close(...)
{
    ...

    ss_node = hdcp_session_list_find(ss_handle, &g_hdcp_session_list);

        Two threads hold the same ‘ss_node’ at here
        Thread 1 gets a node ‘ss_node’
        Thread 2 gets the same ‘ss_node’

    hdcp_session_list_del(ss_node, &g_hdcp_session_list);
    hdcp_session_data_destroy(&(ss_node->ss_data));

                    delete the node and free the data
                    cause double delete and double free
}
```

race condition in list

Samsung s8 del_kek race

```
int del_kek(int engine_id, int kek_type)
{
	kek_pack_t *pack;
	kek_item_t *item;
	...
	item = find_kek_item(pack, kek_type);
	if(item == NULL) return -ENOENT;

	spin_lock(&pack->kek_list_lock);
	del_kek_item(item);
	spin_unlock(&pack->kek_list_lock);

	return 0;
}
```

Almost the same vulnerability

race condition in list

common :

Samsung	Qualcomm	Nvidia	Mtk
6	2	4	5

Pattern matching

- Race condition in list
- Reference count overflow

reference count overflow

kref_get(struct kref *ref)

increase the reference count

kref_put(struct kref *ref, void (*release)(struct kref *kref))

decrease the reference count

call “release” if count == 0

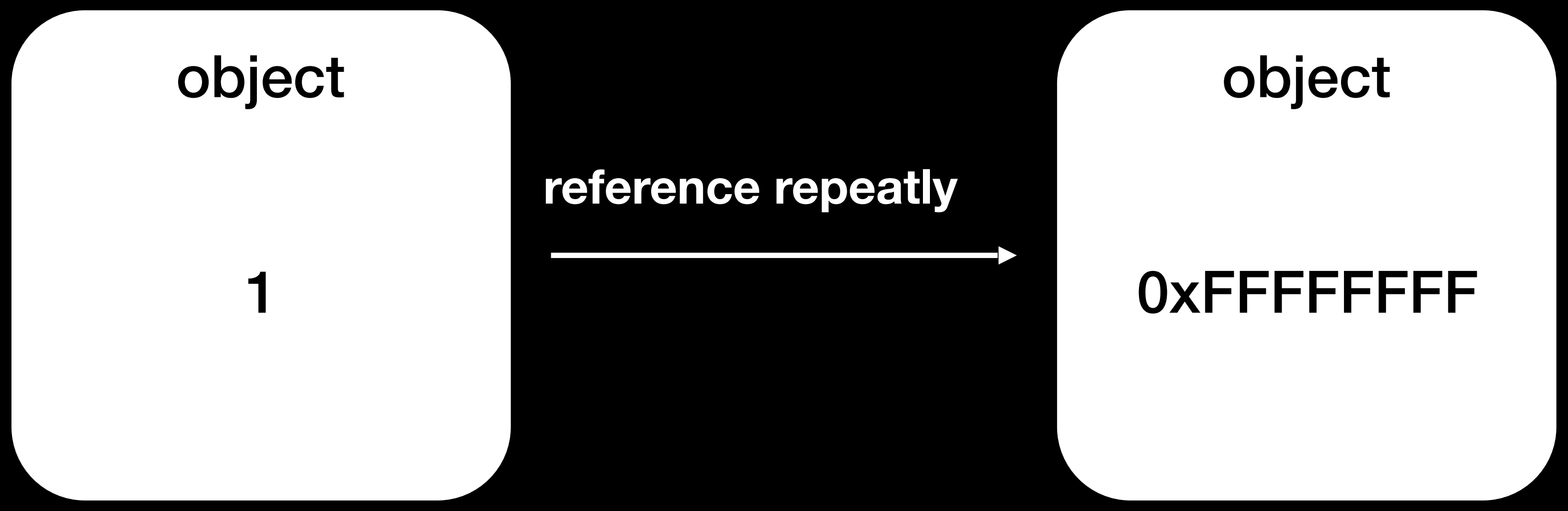
reference count overflow

```
typedef struct {
    int counter;
} atomit_t;
```

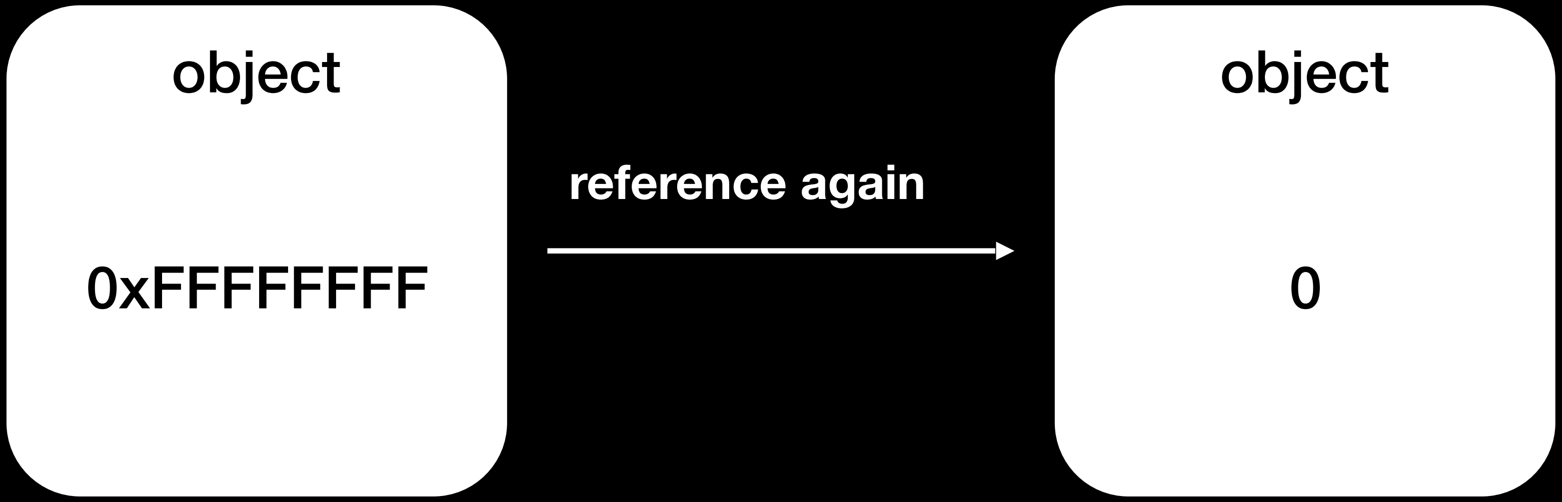
The range of refcount is [0, 0xFFFFFFFF]

```
struct kref {
    atomic_t refcount;
};
```

reference count overflow

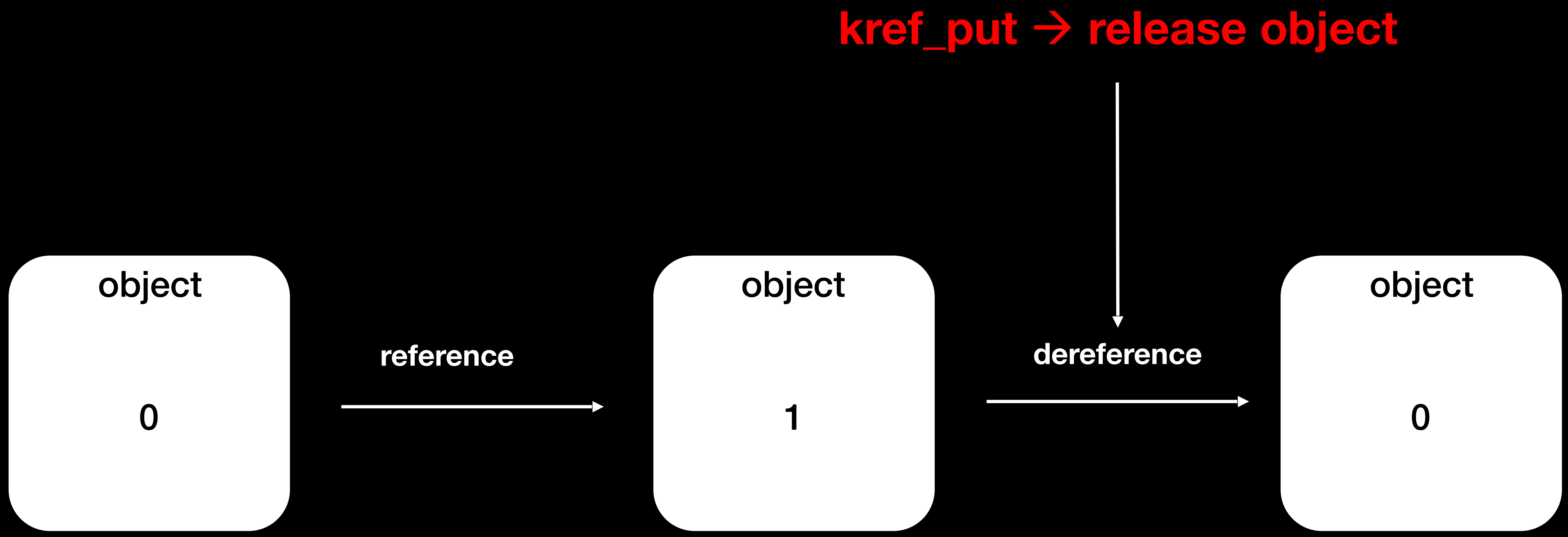


reference count overflow



overflow issue

reference count overflow



reference count overflow

- Keyring refcount overflow (keyring root CVE-2016-0728)
- ION buf refcount overflow (CVE-2017-0507)
- Samsung iva refcount overflow
- …

reference count overflow

- Hard to fuzz (cost much time to do increasing)
- Stable Use-After-Free issue
- Easily ignore

reference count overflow

Samsung	Qualcomm	Nvidia	Mtk
2	1	1	3

Agenda

Dynamic fuzzing tool

- Fuzzing tool’s map
- Anatomy

Static analysis

- Driver model analysis
- Pattern matching

Conclusion

Conclusion

- There are various vulnerabilities lying in Android SoCs.
- New attack surface and new attack model remain to be found.

Q&A