
black hat
ASIA 2018
MARCH 20-23, 2018
MARINA BAY SANDS / SINGAPORE
New Compat Vulnerabilities in Linux Device Drivers
Pengfei Ding
Chenfu Bao
Baidu X-Lab
#BHASIA / @BlackHatEvents

About Us

- Pengfei Ding & Chenfu Bao
- Security Researcher & Developer @ Baidu X-Lab
- Focused on Mobile, IoT and Linux kernel security

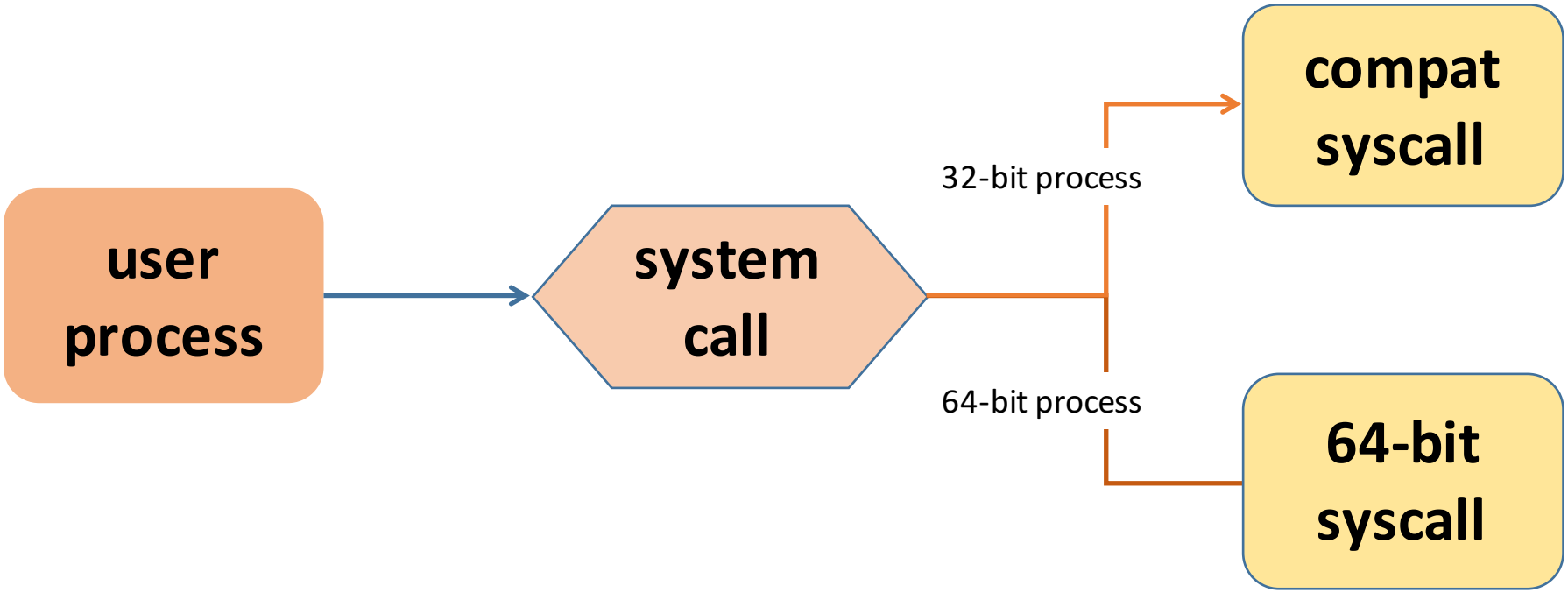
Baidu X-Lab

Agenda

- Introduction
- Past Compat Vulnerabilities
- Newly Identified Compat Vulnerabilities
- Advices & Mitigations
- Conclusions

Introduction of Compat

- 32-bit compatibility mode in 64-bit Linux kernels



Introduction of Compat

- Mainly used to handle the differences in the data sizes
- Many system calls have parameters with different sizes in 32-bit and 64-bit system
  - long, pointer, …
- Convert the values of these parameters to corresponding 64-bit values

Example: settimeofday syscall


```
COMPAT_SYSCALL_DEFINE2(settimeofday, struct compat_timeval __user *, tv,
                struct timezone __user *, tz)
{
    struct timeval user_tv;
    struct timespec new_ts;
    struct timezone new_tz;

    if (tv) {
        if (compat_get_timeval(&user_tv, tv))
            return -EFAULT;
        new_ts.tv_sec = user_tv.tv_sec;
        new_ts.tv_nsec = user_tv.tv_usec * NSEC_PER_USEC;
    }
    if (tz) {
        if (copy_from_user(&new_tz, tz, sizeof(*tz)))
            return -EFAULT;
    }

    return do_sys_settimeofday(tv ? &new_ts : NULL, tz ? &new_tz : NULL);
} ? end COMPAT_SYSCALL_DEFINE2 ?
```

```
SYSCALL_DEFINE2(settimeofday, struct timeval __user *, tv,
        struct timezone __user *, tz)
{
    struct timeval user_tv;
    struct timespec new_ts;
    struct timezone new_tz;

    if (tv) {
        if (copy_from_user(&user_tv, tv, sizeof(*tv)))
            return -EFAULT;

        if (!timeval_valid(&user_tv))
            return -EINVAL;

        new_ts.tv_sec = user_tv.tv_sec;
        new_ts.tv_nsec = user_tv.tv_usec * NSEC_PER_USEC;
    }
    if (tz) {
        if (copy_from_user(&new_tz, tz, sizeof(*tz)))
            return -EFAULT;
    }

    return do_sys_settimeofday(tv ? &new_ts : NULL, tz ? &new_tz : NULL);
} ? end SYSCALL_DEFINE2 ?
```

Security Risks of Compat

- Code redundancy requires more maintenance efforts, thus introducing more security risks
- Additional definition of data structures, type conversion and data processing logic expose new attack surfaces

Past Compat Vulnerabilities

- Occasionally discovered
- Mostly in device drivers
- Mostly caused by inconsistency between compat and non-compat mode
  - Inconsistency of data structure definition
  - Inconsistency of user input validation logic

Past Compat Vulnerabilities

- Inconsistency of data structure definition

```
struct mdp_layer_commit_v1_32 {
    uint32_t          flags;
    int               release_fence;
    struct mdp_rect   left_roi;
    struct mdp_rect   right_roi;
    compat_caddr_t    input_layers;
    uint32_t          input_layer_cnt;
    compat_caddr_t    output_layer;
    int               retire_fence;
    uint32_t          reserved[6];
};
```

```
#define MDP_LAYER_COMMIT_V1_PAD 3

struct mdp_layer_commit_v1 {
    uint32_t          flags;
    int               release_fence;
    struct mdp_rect   left_roi;
    struct mdp_rect   right_roi;
    struct mdp_input_layer __user *input_layers;
    uint32_t          input_layer_cnt;
    struct mdp_output_layer __user *output_layer;
    int               retire_fence;
    void __user       *dest_scaler;
    uint32_t          dest_scaler_cnt;
    uint32_t          reserved[MDP_LAYER_COMMIT_V1_PAD];
};
```

Past Compat Vulnerabilities


```
static void __copy_atomic_commit_struct(struct mdp_layer_commit  *commit,
    struct mdp_layer_commit32 *commit32)
{
    commit->version = commit32->version;
    commit->commit_v1.flags = commit32->commit_v1.flags;
    commit->commit_v1.input_layer_cnt =
        commit32->commit_v1.input_layer_cnt;
    commit->commit_v1.left_roi = commit32->commit_v1.left_roi;
    commit->commit_v1.right_roi = commit32->commit_v1.right_roi;
    memcpy(&commit->commit_v1.reserved, &commit32->commit_v1.reserved,
        sizeof(commit32->commit_v1.reserved));
}
```

memcpy leads to stack overflow!

Past Compat Vulnerabilities

- Inconsistency of user input validation logic

eeprom_init_config validates user input, while its compat version *eeprom_init_config32* does not validate user input

```
diff --git a/drivers/media/platform/msm/camera_v2/sensor/eeprom/msm_eeprom.c b/drivers/m
index 1f891ac..32d2e07 100644
--- a/drivers/media/platform/msm/camera_v2/sensor/eeprom/msm_eeprom.c
+++ b/drivers/media/platform/msm/camera_v2/sensor/eeprom/msm_eeprom.c
@@ -1402,6 +1402,16 @@ static int eeprom_init_config32(struct msm_eeprom_ctrl_t *e_ctrl,

        power_info = &(e_ctrl->eboard_info->power_info);

+       if ((power_setting_array32->size > MAX_POWER_CONFIG) ||
+               (power_setting_array32->size_down > MAX_POWER_CONFIG) ||
+               (!power_setting_array32->size) ||
+               (!power_setting_array32->size_down)) {
+               pr_err("%s:%d invalid power setting size=%d size_down=%d\n",
+                       __func__, __LINE__, power_setting_array32->size,
+                       power_setting_array32->size_down);
+               rc = -EINVAL;
+               goto free_mem;
+       }
        msm_eeprom_copy_power_settings_compat(
                power_setting_array,
                power_setting_array32);
```

patch url : https://source.codeaurora.org/quic/la/kernel/msm-3.18/commit/drivers/media/platform/msm/camera_v2/sensor/eeprom?h=rel/msm-3.18.r5&id=5393 99acbaecba7af7c411f6f28ac0189e459bb5

Past Compat Vulnerabilities

- Inconsistency of user input validation logic

is_compat_task can reduce code redundancy, but inconsistency still exists

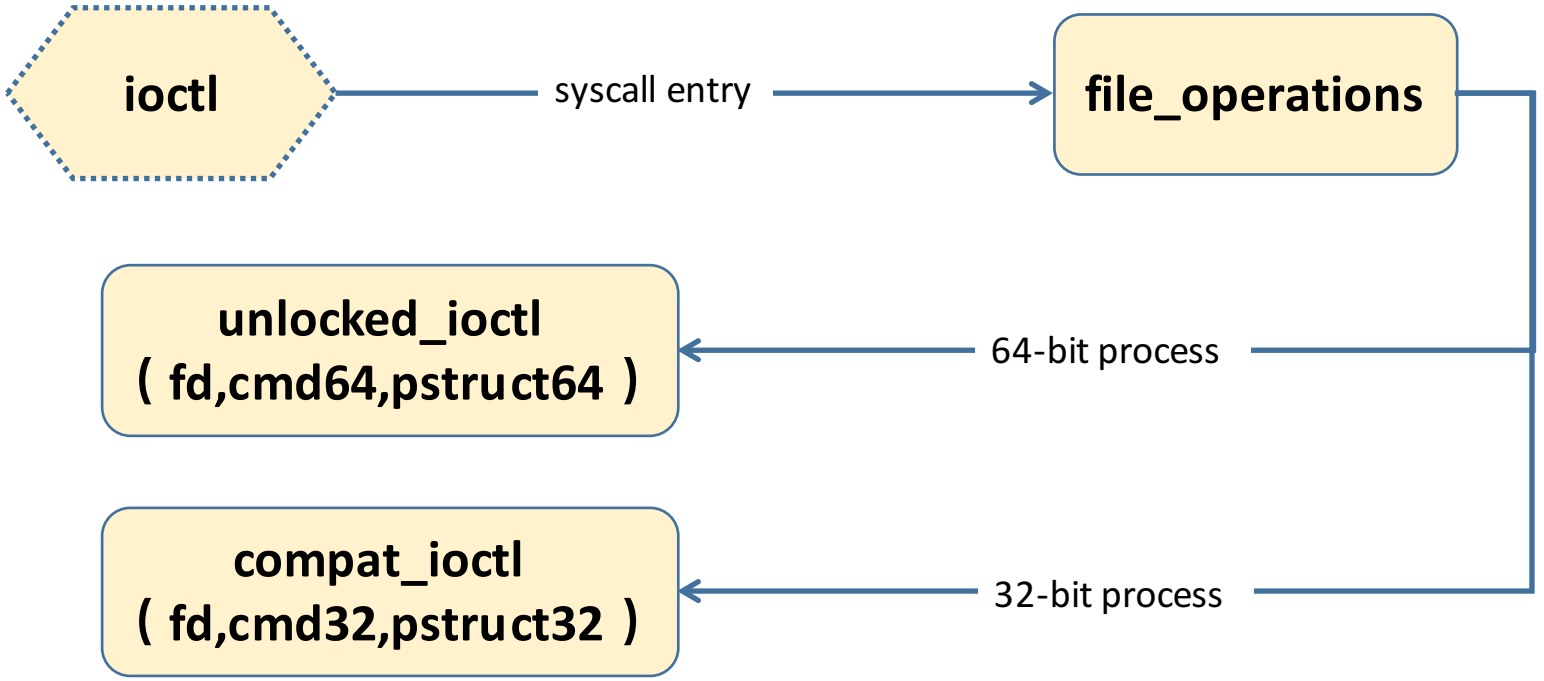
```
diff --git a/drivers/media/platform/msm/camera_v2/sensor/msm_sensor_driver.c b/driver
index 16cacec..1f76654e 100644..100755
--- a/drivers/media/platform/msm/camera_v2/sensor/msm_sensor_driver.c
+++ b/drivers/media/platform/msm/camera_v2/sensor/msm_sensor_driver.c
@@ -420,17 +420,11 @@ static int32_t msm_sensor_create_pd_settings(void *setting,

 #ifdef CONFIG_COMPAT
        if (is_compat_task()) {
-               int i = 0;
-               struct msm_sensor_power_setting32 *power_setting_iter =
-               (struct msm_sensor_power_setting32 *)compat_ptr((
-               (struct msm_camera_sensor_slave_info32 *)setting)->
-               power_setting_array.power_setting);
-
-               for (i = 0; i < size_down; i++) {
-                       pd[i].config_val = power_setting_iter[i].config_val;
-                       pd[i].delay = power_setting_iter[i].delay;
-                       pd[i].seq_type = power_setting_iter[i].seq_type;
-                       pd[i].seq_val = power_setting_iter[i].seq_val;
+               rc = msm_sensor_get_pw_settings_compat(
+                       pd, pu, size_down);
+               if (rc < 0) {
+                       pr_err("failed");
+                       return -EFAULT;
```

patch url :https://source.codeaurora.org/quic/la/kernel/msm-3.18/commit/drivers/media/platform/msm/camera_v2/sensor/msm_sensor_driver.c?h=rel/msm-3.18.r5&id=17c31f3f3438c9f3e05b0d92c70b2b65d430d6cd

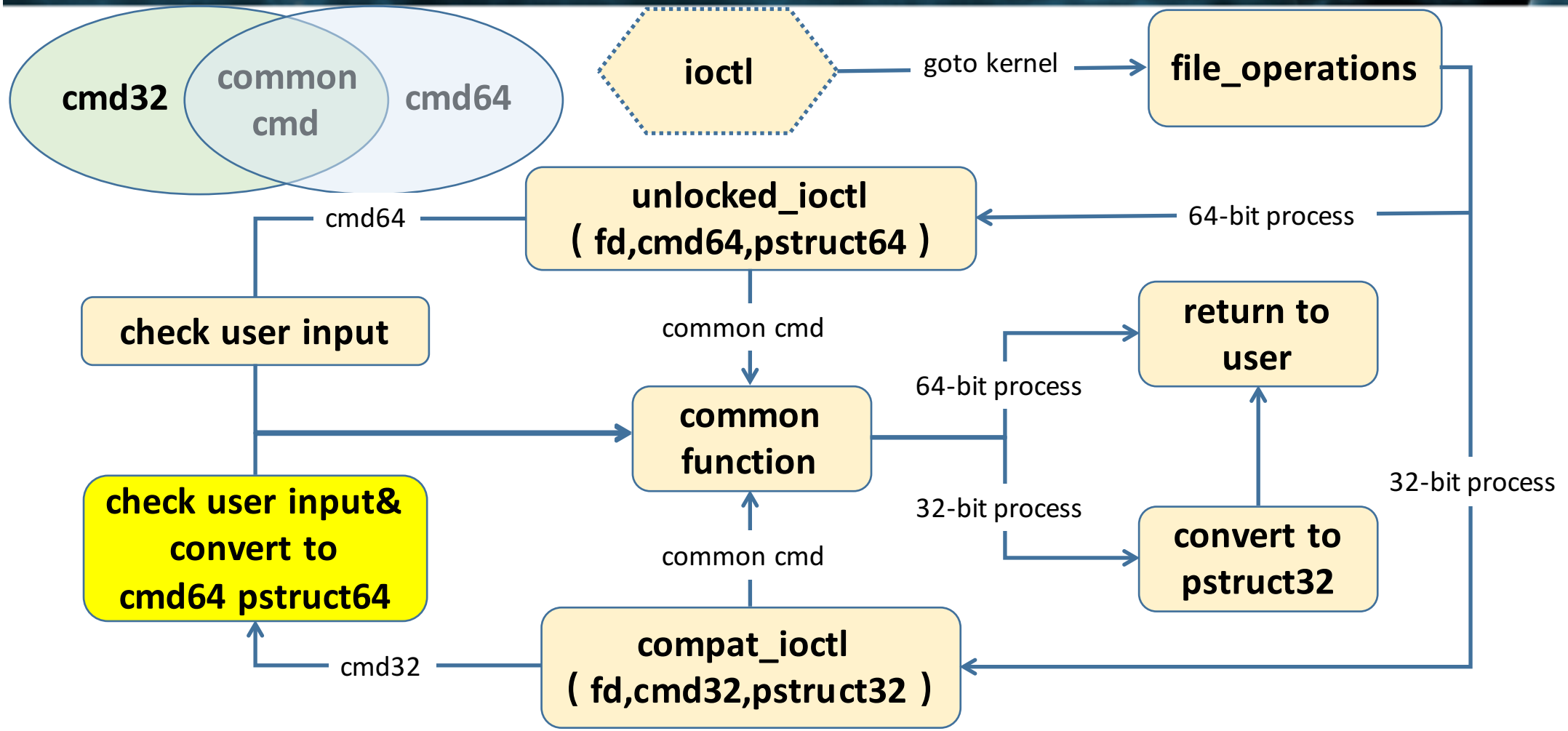
Past Compat Vulnerabilities

- Past research on compat vulnerabilities only focuses on normal program logic
- In device drivers, compat and non-compat codes are often mixed together
- Can mixed codes cause abnormal program logic?

Compat in ioctl



Mixed Codes in ioctl



Idea of New Compat Vulnerabilities

- *compat_ioctl* will make conversion according to the value of cmd32
- What if we **intentionally confuse *compat_ioctl* parameters with *unlocked_ioctl* parameters?**

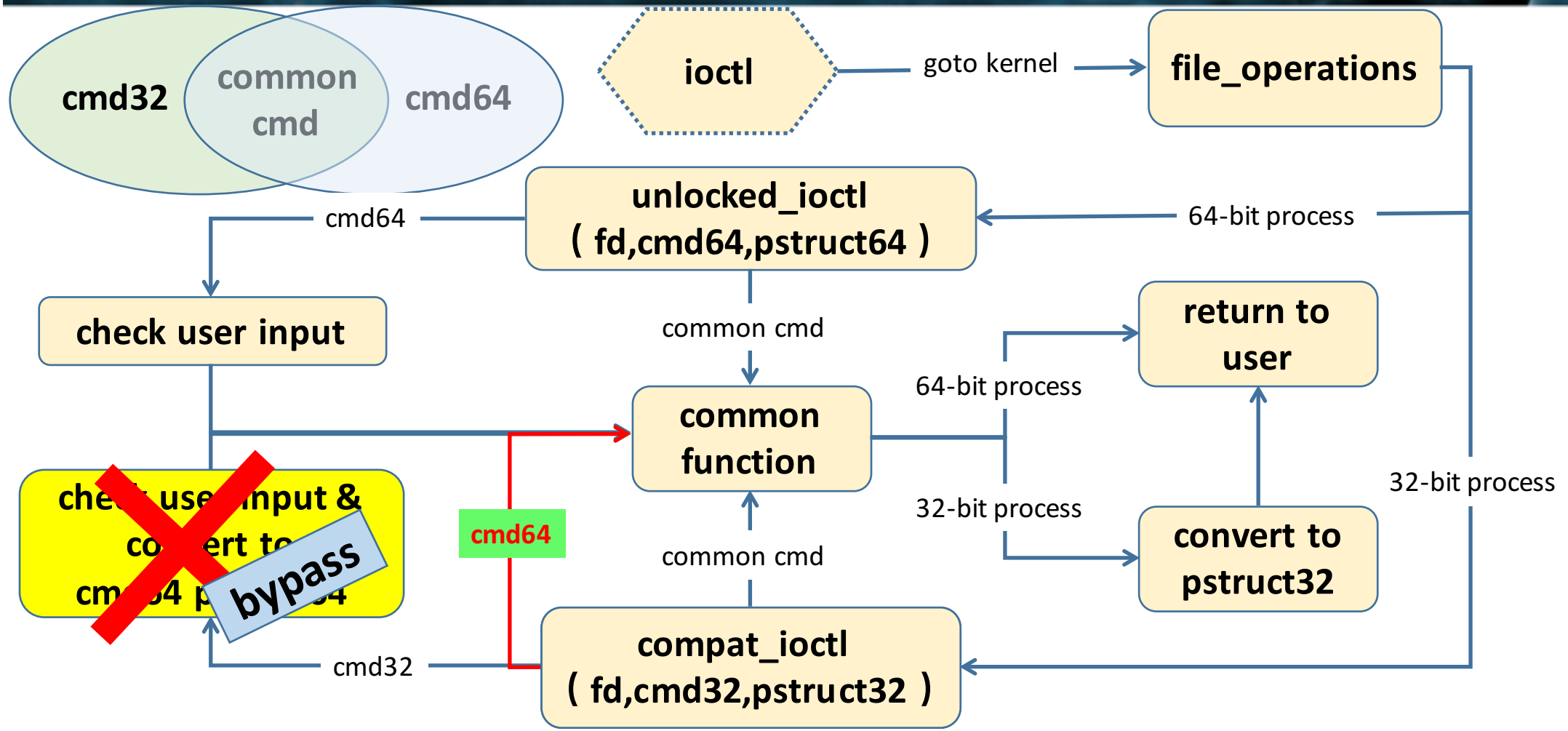
1.compat_ioctl(fd, cmd64, pstruct32)
2.compat_ioctl(fd, cmd64, pstruct64)
3.compat_ioctl(fd, cmd32, pstruct64)
4.unlocked_ioctl(fd, cmd32, pstruct64)
5.unlocked_ioctl(fd, cmd32, pstruct32)
6.unlocked_ioctl(fd, cmd64, pstruct32)

Test & Verification

1.compat_ioctl(fd, cmd64, pstruct32)
2.compat_ioctl(fd, cmd64, pstruct64)
3.compat_ioctl(fd, cmd32, pstruct64)
4.unlocked_ioctl(fd, cmd32, pstruct64)
5.unlocked_ioctl(fd, cmd32, pstruct32)
6.unlocked_ioctl(fd, cmd64, pstruct32)

- *unlocked_ioctl* does not have conversion behavior, cmd32 parameters will be filtered, thus will not cause security problems.
  - 4 and 5 are ruled out
- Processing logic of pstruct parameter in ioctl depends on the value of cmd, so we ignore pstruct(pstruct32， pstruct64) parameter, only focusing on how changes of cmd parameter will affect ioctl
- **compat_ioctl(fd, cmd64, pstruct)**

Bypassed Check & Conversion



Fuzzing Compat

- Existing Linux syscall fuzzing tools do not support compat
  - Trinity
  - Syzkaller
- We extended Trinity and syzkaller and discovered more vulnerabilities

Security Issues

- Bypassing verification on user input array length can lead to out-of-bounds R/W to this array, thus causing privilege escalation

- Bypassing verification on user input pointer value can lead to arbitrary memory read, thus causing information leakage

Security Issues

- Operate kernel memory instead of user memory in check & conversion, which increases the security risk when check & conversion is bypassed
  - *Kmalloc* vs *compat_alloc_user_space*
- When *is_compat_task* is used in *common function*, it is easy to cause logic confusion, and it is more likely to cause security problems when check & conversion is bypassed

Identified CVEs

Example 1: CVE-2017-11029


```
static long msm_cpp_subdev_fops_compat_ioctl(struct file *file,
        unsigned int cmd, unsigned long arg)
```

```
case VIDIOC_MSM_CPP_POP_STREAM_BUFFER32:
{
...
        if (copy_from_user(&k32_frame_info,
                (void __user *)kp_ioctl.ioctl_ptr,
                sizeof(k32_frame_info))) {
 ...
        cmd = VIDIOC_MSM_CPP_POP_STREAM_BUFFER;
        break;
```

```
static int msm_cpp_copy_from_ioctl_ptr(void *dst_ptr,
        struct msm_camera_v4l2_ioctl_t *ioctl_ptr)
...
        /* For compat task, source ptr is in kernel space */
        if (is_compat_task()) {
                memcpy(dst_ptr, ioctl_ptr->ioctl_ptr, ioctl_ptr->len);
```

The processing flow of qualcomm driver function msm_cpp_subdev_fops_compat_ioctl to cmd32:

VIDIOC_MSM_CPP_POP_STREAM_BUFFER32 is shown in the left diagram.

If we pass directly to its corresponding cmd64:

VIDIOC_MSM_CPP_POP_STREAM_BUFFER, the validation of user space pointer **ioctl_ptr** will be bypassed, so it can be assigned to any value by the user, resulting in arbitrary address access when using memcpy.

Identified CVEs

Example 2: CVE-2017-15814


```
#ifdef CONFIG_COMPAT
static long msm_flash_subdev_do_ioctl(
        struct file *file, unsigned int cmd, void *arg)
```

```
switch (cmd) {
case VIDIOC_MSM_FLASH_CFG32:
    cmd = VIDIOC_MSM_FLASH_CFG;
...
        flash_data.cfg.flash_init_info = &flash_init_info;
        if (copy_from_user(&flash_init_info32,
            (void *)compat_ptr(u32->cfg.flash_init_info),
            sizeof(struct msm_flash_init_info_t32))) {
```

```
static int32_t msm_flash_init(
    struct msm_flash_ctrl_t *flash_ctrl,
    struct msm_flash_cfg_data_t *flash_data)
{
...
    if (flash_data->cfg.flash_init_info->flash_driver_type ==
        FLASH_DRIVER_DEFAULT) {
```

The processing flow of qualcomm driver function **msm_flash_subdev_do_ioctl** to cmd32: **VIDIOC_MSM_FLASH_CFG32** is shown in the left diagram. **copy_from_user** checks user space pointer **cfg.flash_init_info** . If we pass directly to its corresponding cmd64:**VIDIOC_MSM_FLASH_CFG**, the validation will be bypassed, so **cfg.flash_init_info** can be assigned to any value , resulting in arbitrary address access when it's dereferenced.

Mitigations

- Try to use *compat_alloc_user_space* instead of *kmalloc* during entire user input check & conversion
- Try to avoid using is_compat_task in *common function*
- Try to use structs instead of pointers in user input to minimize validation of user input

Advices

- Development and test engineers should strengthen the testing and auditing of compat codes
- Fuzz tools and code auditing tools should give more attention to compat codes
- Security researchers can continue to explore compat attack on more platforms

Black Hat Sound Bytes

- Concept and security risks of compat, as well as some compat vulnerabilities in the past
- New type of compat vulnerabilities in Linux device drivers
- How to discover this kind of vulnerabilities and how to avoid them in development

Ending

Thanks!

百度安全

有 AI 更 安 全

Baidu X-Lab