Pengfei Ding & Chenfu Bao

Security Researcher & Developer @ Baidu X-Lab

Focused on Mobile, IoT and Linux kernel security
Introduction

Past Compat Vulnerabilities

Newly Identified Compat Vulnerabilities

Advices & Mitigations

Conclusions
32-bit compatibility mode in 64-bit Linux kernels
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 */
• 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
- 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
Inconsistency of data structure definition

```c
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];
};
```
memcpy leads to stack overflow!
Inconsistency of user input validation logic

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

patch url: https://source.codeaurora.org/quic/la/kernel/msm-3.18/commit/drivers/media/platform/msm/camera_v2/sensor/eeeprom/msm_eeprom_c
- 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/drivers/media/platform/msm/camera_v2/sensor/msm_sensor_driver.c
index 16cae6c..1f76654 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_set_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.r&i=17c31f3f3438c9f3e05b0d92c70b2b65d430d6cd
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?
ioctl

syscall entry

file_operations

unlocked_ioctl ( fd, cmd64, pstruct64 )

64-bit process

compat_ioctl ( fd, cmd32, pstruct32 )

32-bit process
compat_ioctl (fd, cmd32, pstruct32)

check user input

check user input & convert to cmd64 pstruct64

compat_ioctl (fd, cmd32, pstruct32)

common function

unlocked_ioctl (fd, cmd64, pstruct64)

goto kernel

file_operations

return to user

convert to pstruct32

64-bit process

32-bit process

cmd32

common cmd

cmd64

common cmd

64-bit process

32-bit process
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)
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)
compat_ioctl(fd, cmd64, pstruct)
compat_ioctl(fd, cmd32, pstruct32)

Goto kernel

file_operations

check user input & convert to cmd64 & pstruct64

check user input

cmd64

common cmd

compat_ioctl(fd, cmd32, pstruct32)

common function

unlock_ioctl(fd, cmd64, pstruct64)

convert to pstruct32

return to user

64-bit process

32-bit process

bypass

cmd32

common cmd

cmd64

common cmd

cmd64

cmd32
Existing Linux syscall fuzzing tools do not support compat
  - Trinity
  - Syzkaller

We extended Trinity and syzkaller and discovered more vulnerabilities
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.
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
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`.  

```c
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_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.
• 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
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.
● 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
Thanks!