MWR LABS
Security Advisory

Huawei Frame Buffer Driver Arbitrary Memory Write

18/07/2017

Software	MediaTek Frame Buffer Driver
Affected Versions	Huawei Y6 Pro Dual SIM (TIT-L01C576B115)
Author	Mateusz Fruba
Severity	High
Vendor	Huawei
Vendor Response	Fix Released

Description:

Huawei is a company that provides networking and telecommunications equipment.

The MediaTek frame buffer driver, as shipped with Huawei Y6 Pro, implements an IOCTL interface vulnerable to an arbitrary memory write due to insufficient input validation.

Impact:

Local processes running in the context of a system application, media server, or system server can leverage the frame buffer driver memory corruption to escalate their privileges to root or kernel.

Cause:

The MediaTek frame buffer driver fails to validate user-supplied data.

Solution:

This vulnerability was resolved by Huawei in version TIT-L01C576B119. More information can be found on the Huawei web page: http://www.huawei.com/en/psirt/security-advisories/huawei-sa-20170527-01-smartphone-en

Technical details

The MediaTek frame buffer driver implements the ‘mtkfb_ioctl‘ IOCTL handler which receives data passed from user space to the kernel. This driver is implemented in ‘/drivers/misc/mediatek/videox/mt6735/’.

The ‘layerInfo’ structure is user-controlled and passed to the vulnerable ‘_convert_fb_layer_to_disp_input’ function from ‘mtkfb_ioctl’.

Firstly the IOCTL handler function initializes the ‘layerInfo’ variable by copying data into the ‘fb_overlay_layer’ structure from user space using ‘copy_from_user’ as shown below:

```
static int mtkfb_ioctl(struct fb_info *info, unsigned int cmd, unsigned long arg)
{
…
        switch (cmd)
        {

        case MTKFB_SET_OVERLAY_LAYER:
        {
                struct fb_overlay_layer layerInfo;

                if (copy_from_user(&layerInfo, (void __user *)arg, sizeof(layerInfo)))
                {
                        MTKFB_LOG("[FB]: copy_from_user failed! line:%d \n", __LINE__);
                        r = -EFAULT;
                }
                else
                {
…
                        input = &session_input.config[session_input.config_layer_num++];

                        _convert_fb_layer_to_disp_input(&layerInfo, input);
                        primary_display_config_input_multiple(&session_input);
```

The ‘_convert_fb_layer_to_disp_input’ function then copies the ‘layer_id’ variable from the ‘fb_overlay_layer’ structure provided by the user into ‘disp_input_config’ without any checks:

```
static int _convert_fb_layer_to_disp_input(struct fb_overlay_layer* src, disp_input_config
*dst)
{
        dst->layer_id = src->layer_id;
```

Back in ‘mtkfb_ioctl’, we can see that previously copied ‘layer_id’ which is then passed to ‘primary_display_config_input_multiple’ function inside of ‘session_input’ array variable:

```
static int mtkfb_ioctl(struct fb_info *info, unsigned int cmd, unsigned long arg)
{
…
        input = &session_input.config[session_input.config_layer_num++];

        _convert_fb_layer_to_disp_input(&layerInfo, input);
        primary_display_config_input_multiple(&session_input);
```

Next we see that ‘session_input’ is passed to ‘_config_ovl_input’ function as follows:

```
int primary_display_config_input_multiple(disp_session_input_config *session_input)
{
…
        if (_should_config_ovl_input()) {
                _config_ovl_input(session_input, disp_handle, cmdq_handle);
        }
        else {
                _config_rdma_input(session_input, disp_handle);
        }

done:
        _primary_path_unlock(__func__);
        return ret;
}
```

After that, within the ‘_config_ovl_input’ function, the ‘layer_id’ variable is extracted from ‘session_input’ and its value is assigned to the ‘layer’ variable. Next the user-controlled ‘layer’ variable is used as index into the ‘ovl_config’ array without any validation or range checking. This allows an attacker to write an arbitrary address into the ‘ovl_cfg’ which is in the end passed into ‘_convert_disp_input_to_ovl’ function:

```
static int _config_ovl_input(disp_session_input_config *session_input,
        disp_path_handle disp_handle,
        cmdqRecHandle cmdq_handle)
{
…
        for (i = 0; i<session_input->config_layer_num; i++) {
                disp_input_config *input_cfg = &session_input->config[i];
                OVL_CONFIG_STRUCT *ovl_cfg;

                layer = input_cfg->layer_id;
                ovl_cfg = &(data_config->ovl_config[layer]);
…
                _convert_disp_input_to_ovl(ovl_cfg, input_cfg);
```

The '_convert_disp_input_to_ovl' function then uses the attacker-controlled 'dst' variable for six consecutive write operations:

```
static int _convert_disp_input_to_ovl(OVL_CONFIG_STRUCT *dst, primary_disp_input_config* src)
{
       if (src && dst)
       {
              dst->layer = src->layer;
              dst->layer_en = src->layer_en;
              dst->source = src->buff_source;
              dst->fmt = src->fmt;
              dst->addr = src->addr;
              dst->vaddr = src->vaddr;
```

The following crash is observed when this vulnerability is triggered:

```
[  183.190844]<0> (0)[4740:MTKFB_SET_OVERL]Unable to handle kernel paging request at
virtual address ffffffc86fe94214
[  183.190861]<0> (0)[4740:MTKFB_SET_OVERL]pgd = ffffffc04cc43000
[  183.190869][ffffffc86fe94214] *pgd=0000000000000000
[  183.190880]<0> (0)[4740:MTKFB_SET_OVERL][KERN Warning] ERROR/WARN forces debug_lock off!
[  183.190889]<0> (0)[4740:MTKFB_SET_OVERL][KERN Warning] check backtrace:
…
[  183.191305]<0> (0)[4740:MTKFB_SET_OVERL][<ffffffc000083c58>] el1_da+0x1c/0x88
[  183.191319]<0> (0)[4740:MTKFB_SET_OVERL][<ffffffc000498294>]
primary_display_config_input_multiple+0x14/0x24
[  183.191331]<0> (0)[4740:MTKFB_SET_OVERL][<ffffffc000486598>] mtkfb_ioctl+0x6c0/0xd40
[  183.191345]<0> (0)[4740:MTKFB_SET_OVERL][<ffffffc000315ea8>] do_fb_ioctl+0x4f8/0x630
[  183.191358]<0> (0)[4740:MTKFB_SET_OVERL][<ffffffc0003165c0>] fb_ioctl+0x30/0x44
[  183.191374]<0> (0)[4740:MTKFB_SET_OVERL][<ffffffc00019f48c>] do_vfs_ioctl+0x368/0x588
[  183.191388]<0> (0)[4740:MTKFB_SET_OVERL][<ffffffc00019f72c>] SyS_ioctl+0x80/0x98
[  183.191399]<0>-(0)[4740:MTKFB_SET_OVERL]Internal error: Oops: 96000045 [#1] PREEMPT SMP
[  183.191449]<0>-(0)[4740:MTKFB_SET_OVERL]mrdump: cpu[0] tsk:ffffffc0478f8000
ti:ffffffc0482ec000
[  191.405820]<0>-(0)[4740:MTKFB_SET_OVERL]CPU: 0 PID: 4740 Comm: MTKFB_SET_OVERL Tainted:
G        W    3.10.65 #1
[  191.405836]<0>-(0)[4740:MTKFB_SET_OVERL]task: ffffffc0478f8000 ti: ffffffc0482ec000
task.ti: ffffffc0482ec000
[  191.405853]<0>-(0)[4740:MTKFB_SET_OVERL]PC is at
primary_display_config_input_multiple.part.30+0x194/0x89c
[  191.405864]<0>-(0)[4740:MTKFB_SET_OVERL]LR is at
primary_display_config_input_multiple.part.30+0xc8/0x89c
[  191.405874]<0>-(0)[4740:MTKFB_SET_OVERL]pc : [<ffffffc000497b78>] lr :
[<ffffffc000497aac>] pstate: 80000145
[  191.405883]<0>-(0)[4740:MTKFB_SET_OVERL]sp : ffffffc0482ef9f0
[  191.405892]x29: ffffffc0482ef9f0 x28: ffffffc001072430
[  191.405906]x27: 0000000000000000 x26: 000000000f000000
[  191.405919]x25: 000000000f000000 x24: ffffffc001071ed0
[   191.405932]x23: ffffffc077e1e800 x22: 0000000000000000
[   191.405945]x21: ffffffc001077000 x20: ffffffc86fe94210
[   191.405957]x19: ffffffc077e94208 x18: 000000000000000a
[   191.405970]x17: 0000007f7e18d2ec x16: ffffffc0000c3398
[   191.405983]x15: 000000000000393c x14: 0000000000020000
[   191.405996]x13: 00000000000119d8 x12: ffffffc001072440
[   191.406009]x11: 0000000000000040 x10: ffffffc000bb1bb8
[   191.406022]x9 : ffffffc0482ef780 x8 : ffffffc0478f8550
[   191.406035]x7 : 0000002aa704c339 x6 : ffffffc86fe94208
[   191.406047]x5 : ffffffc86fe94208 x4 : ffffffc001072430
[   191.406059]x3 : 0000000000000001 x2 : 00000007f8000000
[   191.406071]x1 : ffffffc077e90000 x0 : 000000000f000000
```

As shown in the above crash log it was possible to force MediaTek frame buffer to use ‘0xf000000’ value as layer variable what caused kernel panic. As after sum of ‘x0’ and ‘x19’ registers ‘0xffffffc86fe94214’ kernel space address was accessed which was currently not mapped within kernel address space.

Detailed Timeline

Date	Summary
2017-03-28	Issue reported to Huawei.
2017-06-05	Huawei confirmed this issue was fixed in version TIT-L01C576B119.