Janar Saks
Overview of CVE-2013-2094 vulnerability

Supervisor: Meelis Roos

Tartu 2017
Description of the vulnerability

CVE-2013-2094 is a vulnerability that affects Linux kernel versions 2.6.37 to 3.8.9. Vulnerability comes from an internal kernel function `perf_swevent_init`, where kernel before the version 3.8.9 uses an incorrect integer data type, which allows local users to gain privileges via a crafted `perf_event_open` system call.

Function `perf_swevent_init` takes an argument that is `struct perf_event_attr`, which has field `config` that is defined as a 64-bit unsigned integer.

```c
struct perf_event_attr {
    /*
     * Major type: hardware/software/tracepoint/etc.
     */
    __u32 type;
    /*
     * Size of the attr structure, for fwd/bwd compat.
     */
    __u32 size;
    /*
     * Type specific configuration information.
     */
    __u64 config;
};
```

But the function `perf_swevent_init` casts the unsigned 64-bit config value to a signed 32-bit integer. The problem is the fact that the casted value is only checked for its upper bound. Therefore, when any config values that have the 31st bit set will result in a negative index, which does not get caught by the if-statement. Also, in function `perf_swevent_init` we will end up incrementing kernel mode addresses.

```c
static int perf_swevent_init(struct perf_event *event) {
    int event_id = event->attr.config;
    /* ... */
    if (event_id >= PERF_COUNT_SW_MAX)
        return -ENOMEM;
    /* ... */
    atomic_inc(&perf_swevent_enabled[event_id]);
    /* ... */
}
```

The problem that an unsigned 64-bit value passes the if-statement does not end there. After the function `perf_swevent_init` another function is called: `sw_perf_event_destroy`. Where the codes assumes that the code looked above will reject a config value that is too large, but that is not always the case as shown before. Therefore, it is possible to allow a user program to cause the kernel to decrement an address to a memory area that the user can control.
Exploiting the vulnerability

Example code segments to show, how the vulnerability can be exploited (see full exploit code: https://github.com/realtalk/cve-2013-2094/blob/master/rewritten_semtext.c#L56-L201).

First, the exploit will set up an initial memory region with mmap at the address 0x38000000 for a length of 0x01000000 bytes, which is close to the end of the user controllable address space. Then the exploit will fill the region with zeros and call twice a method that use system call to run `perf_event_open`:

```c
break_perf_event_open(-1);
broke_perf_event_open(-2);
```

These arguments are chosen, because then there will be a result of two decrement addresses that which are inside the initial memory region.

After iterating over the initial memory region the exploit will find the exact place where the decrements landed, which is important, because then we find out the relative position of the overflowed array `perf_swevent_enabled` to us.

Next the exploit will use SIDT\(^2\) to get the interrupted descriptor table pointer, which is an array of 16-bit descriptors. Furthermore, the exploit does a clever calculation with the address, which allows the exploit to allocate memory in user space. Also, this calculations make it so that the adjusted address is same as some interrupt vector pointers.

```c
asm volatile("sidt %0": "=m" (idt)); kbase = idt.addr & 0xfff00000;
```

The newly allocated region is filled with NOP instructions that move the CPU’s instruction execution flow to the desired destination,\(^3\) because the address is unknown that the kernel calls. Also the malicious function `ix_idt_and_overwrite_cred` code is copied to the last 1 kB of the memory and a shellcode stub is copied into the memory region just before the malicious code.

---

1 This is a rewritten and commented version of the original and it was rewritten mostly to help people understand how it works.

2 Store Interrupt Descriptor Table Register

Next the exploit locates and sets all uids and gids to the current ones, which will help the malicious code to find the right place to begin overwriting data. Then the exploit is going to overwrite an interrupt handler. To do that the exploit executes:

```
break_perf_event_open(-i + ((idt.addr0xffffffff)-0x80000000)/4) + 16);
```

This calculation uses the found address of the array `perf_swevent_enabled` to calculate an offset for an interrupt vector. This call will look for the IDT\(^4\) entry of the [int 0x4] interrupt. The exploit will then increment the upper 32-bits of the found address, which currently is 0xffffffff816ee00. But the increase will cause an overflow in the upper 32-bits and the interrupt pointer address 0xffffffff816ee00 will change to 0x00000000816ee00. Furthermore, the new adjusted interrupt vector to point directly to the region with the malicious code.

Finally, the exploit needs to call the interrupt, which will call the malicious code in kernel context and you can execute a shell as root:

```
asm volatile("int $0x4");
```

### Solution

To change the event_id type in the function `perf_swevent_init` to 64-bit integer.

```c
static int perf_swevent_init(struct perf_event *event)
{
    int event_id = event->attr.config;
    u64 event_id = event->attr.config;

    if (event->attr.type != PERF_TYPE_SOFTWARE)
        return -EINVAL;
```

---

\(^4\) The Interrupt Descriptor Table (IDT) is a data structure used by the x86 architecture to implement an interrupt vector table. Wikipedia https://en.wikipedia.org/wiki/Interrupt_descriptor_table Visited: 30.04.17
References

2. https://www.slideshare.net/kerneltlv/semtex-cve20132094-a-linux-privilege-escalation
4. https://bugzilla.redhat.com/show_bug.cgi?id=962792#c16
5. https://github.com/torvalds/linux/commit/8176cced706b5e5d15887584150764894e94e02f