Qemu/KVM

Gabriel Laskar <gabriel@lse.epita.fr>

LSE Security System

Overview

- KVM is the Linux Hypervisor
- Splitted in 2 parts :
  - kvm : kernel module
  - qemu : device emulation, vm setup

KVM

- Leverage Linux APIs & subsystems for Virtualization
- 3 modules : kvm.ko, kvm-intel.ko, kvm-amd.ko
- code size :
  - ~7kloc arch-independant code
  - ~29kloc arch-dependant code

KVM

- Expose virtualization api to the userland
- Use only Hardware virtualization instructions
- small size
- reuse linux apis when possible (scheduling, memory management, events, ...)

LSE Security System

Qemu

- Use also in Xen
- VM creation
- Device emulation

LSE Security System

KVM Api

- VM creation
- Memory assignation
- irq chip
- launch a cpu
- devices

/dev/kvm

- /dev/kvm expose an anonymous virtual filesystem for the hypervisor
- Every resources are managed through a fd :
  - kvm configuration
  - vm management
  - vcpu management

/dev/kvm : system fd

- `ioctl(fd, KVM_CREATE_VM)`
- `ioctl(fd, KVM_GET_MSR_LIST)`
- `ioctl(fd, KVM_CHECK_EXTENSION)`
- `ioctl(fd, KVM_GET_VCPU_MMAP_SIZE)`

Example : vm creation

```
int fd_kvm = open("/dev/kvm", O_RDWR);

int kvm_run_size = ioctl(fd_kvm, KVM_GET_VCPU_MMAP_SIZE,
0);

int fd_vm = ioctl(fd_kvm, KVM_CREATE_VM, 0);

// add space for some strange reason on intel (3 pages)
ioctl(fd_vm, KVM_SET_TSS_ADDR, 0xffffd000);

ioctl(fd_vm, KVM_CREATE_IRQCHIP, 0);
```

LSE Security System

/dev/kvm : vm fd

- KVM_SET_MEMORY_REGION
- KVM_CREATE_VCPU
- KVM_GET_DIRTY_LOG
- KVM_CREATE_IRQCHIP (extension)
- KVM_{GET,SET}_DEBUGREGS

Example : Memory Assignation

```
// set memory region
void *addr = mmap(NULL, 10 * MB, PROT_READ | PROT_WRITE,
                  MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);

struct kvm_userspace_memory_region region = {
        .slot = 0,
        .flags = 0,
        .guest_phys_addr = 0x100000,
        .memory_size = 10 * MB,
        .userspace_addr = (__u64)addr
};

ioctl(fd_vm, KVM_SET_MEMORY_REGION, &region);
```

LSE Security System

/dev/kvm : VCPU fd

- KVM_RUN
- KVM_{GET,SET}_REGS
- KVM_{GET,SET}_SREGS
- KVM_TRANSLATE
- KVM_INTERRUPT (without local apic)
- KVM_{GET,SET}_MSRS
- KVM_SET_CPUID

LSE Security System

Example : VCPU Creation & setup

```
int fd_vcpu = ioctl(fd_vm, KVM_CREATE_VCPU, 0);

struct kvm_sregs sregs;
ioctl(fd_vcpu, KVM_GET_SREGS, &sregs);

#define set_segment(Seg, Base, Limit, G) \
  do {                                   \
    Seg.base = Base;                     \
    Seg.limit = Limit;                   \
    Seg.g = G;                           \
  } while (0)

set_segment(sregs.cs, 0x0, 0xffffffff, 1);
set_segment(sregs.ds, 0x0, 0xffffffff, 1);
set_segment(sregs.ss, 0x0, 0xffffffff, 1);

sregs.cs.db = 1;
sregs.ss.db = 1;
```

```
sregs.cr0 |= 0x01;

ioctl(fd_vcpu, KVM_SET_SREGS, &sregs);

struct kvm_regs regs;
ioctl(fd_vcpu, KVM_GET_REGS, &regs);
regs.rflags = 0x02;
regs.rip = 0x00100f00;
ioctl(fd_vcpu, KVM_SET_REGS, &regs);
```

LSE Security System

Example : Run VM

```
struct kvm_run *run_state =
    mmap(0, kvm_run_size, PROT_READ|PROT_WRITE,
MAP_PRIVATE,
         fd_vcpu, 0);

for (;;) {
        int res = ioctl(fd_vcpu, KVM_RUN, 0);

        switch (run_state->exit_reason) {
                /* … */

        }
}
```

LSE Security System

Exit Reasons

- KVM_EXIT_EXCEPTION
- KVM_EXIT_IO
- KVM_EXIT_MMIO
- KVM_EXIT_SHUTDOWN
- …

LSE Security System

Port IO

```
case KVM_EXIT_IO:
        if (run_state->io.port == CONSOLE_PORT
           && run_state->io.direction == KVM_EXIT_IO_OUT)
{

                __u64 offset = run_state->io.data_offset;
                __u32 size = run_state->io.size;

                write(STDOUT_FILENO,
                        (char*)run_state + offset, size);
        }
        break;
```

MMIO, PIO : How fast ?

- For each mmio access, there is an exit
- We have to assert the read/write, and process the command
- Can’t be asynchronous, ie : we can’t do that with the vcpu guest running
- What solutions do we have ?

LSE Security System

Eventfd

- KVM_IOEVENTFD
  - Attach an ioeventfd to a pio/mmio guest address
  - When guest write into this address, it fire an event instead of an exit

Irqfd

- KVM_IRQFD
  - Allow setting an eventfd that will trigger a guest interrupt

How can we solve the IO Problem ?

- With eventfd and irqfd, we can offload io traffic into another thread, and just listen/fire event through fds.

Example : handling device

```
void handle_device(void *device, int eventfd, int irqfd)
{
     struct pollfd input_queue = {
          .fd = eventfd,
          .events = POLLIN;
     };

     for (;;) {
          int ret = poll(input_queue, 1, timeout);

          if (ret > 0) {
               uint64_t event_value;
               read(eventfd, &event_value, sizeof(event_value));
               uint64_t res = do_something(device, event_value);
               write(irqfd, &res, sizeof(res));
          }
     }
}
```

LSE Security System

What did we not cover

- Vhost
- VFIO
- KSM
- libvirt