CSIE 5310 Assignment 2 (Due on April 11st 14:10)

Nowadays, virtual machines are widely used in cloud applications, and many different users' VMs run on the same host machine simultaneously. However, a malicious host can access the data in the guest VMs arbitrarily, creating significant security risks. For this assignment, you will implement such an attack.

In Assignment 2, you are required to extend the implementation of KVM/Arm by adding a new ioctl that allows the host to write to the memory in guest VMs. Additionally, you are asked to perform a code injection attack by injecting shellcode into a target process running on the guest VM using the new ioctl.

As you have learned in the class, when you create a VM on KVM, you first launch a process running QEMU. QEMU then sets up the environment to run vCPUs. In this assignment, you are asked to perform a cross-process code injection attack; carrying out the attack against the VM running in the QEMU process, from an external process running `hw2-test`.

0. Late submission policy

- 1 pt deduction for late submissions within a day (before Apr. 12nd 14:10)
- 2 pts deduction for late submissions within 2 days (before Apr. 13rd 14:10)
- zero points for submissions delayed by more than 2 days.

1. Before you start

- This assignment builds on the Ubuntu environment for KVM for Armv8 that you created in Assignment 1.
- Download the attachment `vm_hw2_files.zip` from NTU Cool into your Ubuntu host. There are 2 files used in this assignment:
  - `hw2-test.c`
  - `hw2-sheep.c`

2. Adding new ioctls to KVM (60%)

You are required to modify the KVM/Arm codebase by adding a new system ioctl: `KVM_ARM_WRITE_GPA`. The ioctl takes a pointer to a `struct kvm_arm_write_gpa_args` as argument, which has the following fields:

```
struct kvm_arm_write_gpa_args {
        uint32_t vmid;  // the vmid that you, as the host, want to write to
        uint64_t gpa;   // the gpa of the guest
        uint64_t data;  // address of the payload in host user space
        uint64_t size;  // size of the payload
};
```

The `KVM_ARM_WRITE_GPA` ioctl should copy the data at the `data` buffer to the `gpa` in the VM specified by the `vmid`. The copy size is specified by `size`.

A `struct kvm` instance will be initialized when a new VM is launched by KVM (see `kvm_create_vm()` in `virt/kvm/kvm_main.c`), and a distinct `vmid` will be associated to it (see `kvm->arch.mmu.vmid.vmid`). Also, the `struct kvm` will be added into a global linked-list of all active VMs `vm_list`. Your system ioctl should first find the `struct kvm` corresponding to the provided `vmid` by searching the `vm_list` list.

After this, your ioctl should make a request to one of the vcpus in the target VM (see `kvm_make_request()` in `include/linux/kvm_host.h`). The request should be checked before next time entering the guest by running the vcpu (see `check_vcpu_requests()` in `arch/arm64/kvm/arm.c`). Optionally, you may "kick" the vcpu so that the vcpu returns to the hypervisor immediately, and your request may be checked earlier (see `kvm_vcpu_kick()` in `virt/kvm/kvm_main.c`). When the request is being checked, the data should finally be written to the guest VM memory.

In your ioctl implementation, you are required to properly handle errors (for instance, invalid vmid), and return error codes to userspace. For errors that happen when the request is being handled, you can simply print out some error messages.

HINT 1:

Note that your ioctl should be a **system ioctl**, so that the ioctl can be used by any userspace program. See KVM API Documentation.

HINT 2:

See The Linux Kernel API for dealing with linked-list in Linux kernel.

HINT 3:

The `kvm_vcpu_write_guest()` function may be useful to you.

3. Userspace Test Programs (30%)

3-1. Testing new ioctl (10%)

You are asked to write userspace program(s) to test your ioctl. You should describe how you use your program(s) to test the correctness of your ioctl in your write-up.

3-2. Code injection (20%)

In addition to the test program(s) in 3-1, you should use the new ioctl to carry out a code injection attack on a target process running on a guest VM. You are provided a template test program `hw2-test.c` that initializes an argument containing pointer to the shellcode. You should modify this program to invoke the new ioctl, inject the shellcode into the target process compiled from the provided `hw2-sheep.c` running on the guest VM, and trigger the payload that launches `/bin/sh` in the target process.

To perform the attack, you should first launch a guest VM on your KVM/Arm host, and run the `hw2-sheep` process in the guest VM. After starting the `hw2-sheep` process, you should find out the gpa of the while loop in your `hw2-sheep` process, and inject the shellcode to the gpa using your userspace program in your KVM/Arm host. After a successful injection, the shell running the `hw2-sheep` process will be running `/bin/sh` and you may see the shell prompt `#`. It should be able to process shell command inputs.

HINT 4:

You may use the `page-types` tool provided in the Linux codebase for finding the corresponding physical address of a virtual address in a process. To compile the tool:

```
# make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -C ./tools/vm/
```

The binary `page-types` will be at `tools/vm`. You can see all mappings from virtual page number to physical frame number of a process using

```
# ./page-types -p <pid> -L
```

4. Write-up (10%)

- (8%) You are required to provide a write-up about the assignment. The write-up should include instructions for compiling and running your test programs, an explanation of your source code (including a description of how your code and test program work), how you test your ioctl, and a description of how you carried out the shellcode injection attack.
- Answer the following question:
    - (2%) Your ioctl writes to the VM memory by making a vcpu request, and the VM memory will be written when the request is handled. Why can't your ioctl write to the VM memory directly (for example, using `kvm_vcpu_write_guest()`), without making requests?

5. Homework submission

You should submit the assignment via NTU Cool.

Submission format

You are required to submit the kernel patch for Linux v5.15, the source codes of your userspace programs to test the ioctls and perform the code injection attack, a Makefile to compile the program, and a write-up file. Compress all the files in hw2.zip, and upload the zip file to NTU Cool.

Your Makefile does not have to deal with copying or deploying the test program to your VM.

```
.
├── source codes of your user space test programs
├── Makefile
├── write-up.pdf
├── YOUR_5.15_KERNEL.patch
├── any extra file
```

Grading criteria

You will get zero point if your patch fails to apply or your implementation fails to compile. You are required to properly manage resources (free allocated memory), handle errors, and return error codes to user space.