CMPE 283 - Assignment 1

Sarthak Jain
014508013

Question 1: *For each member in your team, provide 1 paragraph detailing what parts of the lab that member implemented/researched.*

Answer: For the lab, I queried the **Procbased** (IA32_VMX_PROCBASED_CTLS) and the **Secondary Procbased** (IA32_VMX_PROCBASED_CTLS2) MSRs to find the features present in my machine. As a processor is given a new mode of operation called VMX Operation to provide for hardware capability, the VMM operates in VMX root and Guest OS operates in VMX non-root. Certain events/instructions/interrupts can make a VM exits to the VMM i.e. a transition from VMX root to VMX non-root and thus the VMM can handle them the way it wants and these transitions replace the ordinary behaviors of those events/instructions/interrupts.

Question 2: *Describe in detail the steps you used to complete the assignment. Consider your reader to be someone skilled in software development but otherwise unfamiliar with the assignment. Good answers to this question will be recipes that someone can follow to reproduce your development steps.*

Answer: Please find below the list of steps I followed to complete the assignment.

1. My teammate (*Mrugesh*) forked the Github repository, *torvalds/linux,* in his account and added me as a collaborator in the forked repository.

2. I git cloned the repository using the web URL present in the Clone or download option in the repository.

3. After cloning the repository, I installed certain required modules to build a Linux kernel. The command that was executed was ***sudo apt-get install libncurses5-dev gcc make git exuberant-ctags bc libssl-dev fakeroot build-essential ncurses-dev xz-utils flex libelf-dev bison***

4. Then I copied the config file of my current Linux kernel and pasted in the root directory of the repository I cloned with the command **cp /boot/config-$(uname -r) .config**

5. I then executed make menuconfig and accepted the default values.

6. Next, I checked for number of cores I had in my machine by **nproc** (I had 4) followed by which I executed the make command by **sudo time make -j4.**

7. This builds my new Linux kernel that I just cloned from Github and provided my current config file and built it. The next step was to install this kernel and update my grub too which I skipped for now and started with Assignment 1.

8. I created a folder titled **cmpe283** using the mkdir command.

9. Inside cmpe283, I created (using mkdir) another folder titled **assignment1** to put files related to assignment 1 in this folder.

10. Inside the assignment1 folder, I created a folder with my name and SJSU id, i.e. **sarthakjain_014508013** to put files related to my part in my dedicated separate folder.

11. Inside **sarthakjain_014508013,** I put the starter code files provided to us, Makefile and cmpe283-1.c

12. Then I opened cmpe283-1.c using **vim cmpe283-1.c** command.

13. I had to read the *IA32_VMX_PROCBASED_CTLS* and *IA32_VMX_PROCBASED_CTLS2* MSRs and find the features supported by my hardware for those MSRs.

14. Firstly, I referred the **Volume 3, section 24.6.2 and table 24-6 and table 24-7** of the *Intel 64 and IA-32 Architectures Software Developer's Manual* to understand the meaning of bit positions in the above MSRs and how to interpret those results.

15. I **wrote the bit position and the control name** for both Primary Processor-based (*IA32_VMX_PROCBASED_CTLS*) and Secondary Processor-based (*IA32_VMX_PROCBASED_CTLS2*) in two separate arrays of

structs in my c file.

16. Then in my **detect_vmx_features(void)** function, I read the *IA32_VMX_PROCBASED_CTLS* MSR using **rdmsr** command and supplied lo, hi parameter which is of type uint32_t.

17. After getting the high and low 32 bits of the msr in hi and lo parameters respectively, I displayed a message using **pr_info** to say I am displaying features of *IA32_VMX_PROCBASED_CTLS* MSR.

18. I then **called** the **report_capability** function passing the **procbased array of structs**, the **number of elements in this array** which is equal to number of bit positions we will be reading from the *IA32_VMX_PROCBASED_CTLS* MSR and the **lo** and **hi** parameters in which the low and high bits of the above MSR is present.

19. The report_capabilty function loops through the array of struct and for each bit position mentioned in the array of structs, reads and interprets the bit value in lo and hi parameters at that bit position.

20. For **hi parameter if** the **value** is **1** at given bit position then **we can set** that control, otherwise, we cannot.

21. For **lo parameter**, it's **reverse** of how we interpreted the **hi** bit. For lo, if the **value** of the bit at the given position **is 0**, then **we can clear** the value at that bit position, otherwise, we cannot.

22. The above steps complete the reading and writing of capabilities provided by my machine for the *IA32_VMX_PROCBASED_CTLS* MSR. Now we have to see whether we have **Secondary Processor-based controls present or not** by reading the **31st bit (assuming 0 indexing)** and checking if the value is 1 or not. If the value is 1 then secondary processor-based controls are present, otherwise, not. To check the 31st bit (assuming 0 indexing) of hi or **63 bit (assuming 0 indexing)** of IA32_VMX_PROCBASED_CTLS, I referred **SDM**, **Volume 3, Appendix A, section A.3.3**

23. Upon checking, my machine did support secondary processor-based controls and the steps to read the supported capabilities are the same as for procbased. Firstly, we read MSR *IA32_VMX_PROCBASED_CTLS2* using **rdmsr** into lo and

hi parameter which is of type unint32_t.

24. Then we display a message with **pr_info** stating we are displaying features for Secondary Processor-Based Controls.

25. Then we call the **report_capability()** passing the secondary_procbased array of structs, the given number of elements in it, and lo and hi parameter to read and interpret the values in the MSR. The interpretation is the same as mentioned in steps 15 and 16.

26. In the case of **Step 17**, if the value would have been **0**, then I was displaying a message using pr_info that *secondary processor-based controls are not available*.

27. This concludes the writing of the kernel module. We save it by pressing esc and then :wq to save and exit the vim.

28. I then execute make command in the directory where both my **cmpe283-1.c** and **Makefile** are present. It builds the file and outputs a kernel object (.ko) which needs to be then inserted into the kernel.

29. I executed **sudo insmod ./cmpe283-1.ko** to insert the kernel object into the kernel and then executed **dmesg** to check the output from my inserted kernel module.

30. **On a side note**, executing the above command was giving me Operation not Permitted error because of **UEFI Secure Boot** being enabled. I had to enter the BIOS setup, by restarting the laptop and pressing esc, going to boot options, then boot manager and **disabling Secure Boot.**

31. I then executed **lsmod | grep cmpe** to find the name of my module and remove it using the command **sudo rmmod cmpe283_1**. Again doing a dmesg shows the message printed when **cleanup_module()** is called on doing the rmmod.

32. I then executed **dmesg > output.md** and deleted unwanted message lines and just kept the logs from inserting, execution and removal of my kernel module followed by adding some syntactic sugar in terms of headings and formatting.

33. I then had to push my changes to Github for which I executed the following commands:
    a. **git add cmpe283-1.c**
    b. **git add cmpe283-1.ko**
    c. **git add Makefile**
    d. **git add output.md**

34. I then executed **git commit -m “”** and gave a commit message.
35. I then executed **git pull origin master** followed by **git push origin master** to get the latest changes and push my changes. This was executed from the directory where .git was present which is the root of the cloned GitHub repository folder.