Lab 08 Nailgun Defense

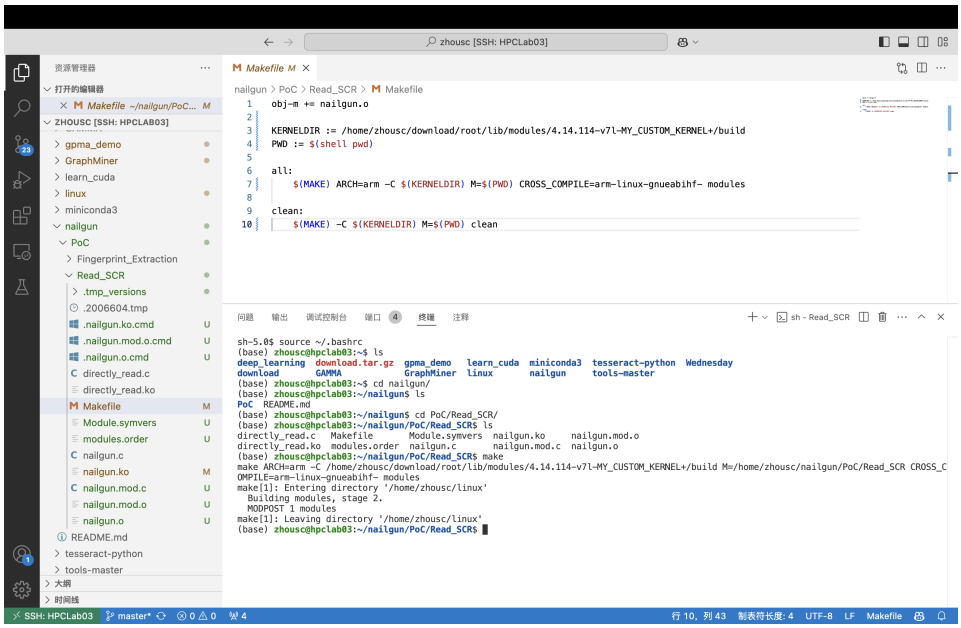
SID: 12110644

Name: Sicheng Zhou

Question 1:(20%) Can you prove that (1) you have replaced the kernel (with ”uname -r” or other approaches), and (2) you have built the nailgun module with new headers? Please provide a figure.

```
claudiacumberbatch — pi@raspberrypi: ~ — ssh pi@172.20.10.3 — 80×24
ssh: connect to host 192.168.54.2 port 22: Connection refused
(base) claudiacumberbatch@ClaudiadeMacBook-Air ~ % ssh pi@172.20.10.3
pi@172.20.10.3's password:
Permission denied, please try again.
pi@172.20.10.3's password:
Linux raspberrypi 4.14.114-v7l-MY_CUSTOM_KERNEL+ #1 SMP Fri Nov 15 10:26:34 CST
2024 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Nov 15 11:16:46 2024

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.14.114-v7l-MY_CUSTOM_KERNEL+ #1 SMP Fri Nov 15 10:26:34 CST
2024 armv7l GNU/Linux
pi@raspberrypi:~ $
```

the kernel has been replaced



new nailgun module

Question 2:(20%) Can you run the Nailgun Attack on your new kernel? Please provide a figure. You can use ”dmesg” to show the execution result of Nailgun Attack.

```
[   14.781969] Bluetooth: RFCOMM TTY layer initialized
[   14.782001] Bluetooth: RFCOMM socket layer initialized
[   14.782019] Bluetooth: RFCOMM ver 1.11
[   15.525752] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[   17.332397] ICMPv6: process `dhcpcd' is using deprecated sysctl (syscall) net.ipv6.neigh.wlan0.retrans_time - use net.ipv6.neigh.wlan0.retrans_time_ms instead
[  522.141063] nailgun: loading out-of-tree module taints kernel.
[  522.141694] Step 1: Unlock debug and cross trigger registers
[  522.141700] Step 2: Enable halting debug
[  522.141705] Step 3: Halt the target processor
[  522.141710] Step 4: Wait the target processor to halt
[  522.141714] Step 5: Save context
[  522.141719] Step 6: Switch to EL3
[  522.141723] Step 7: Read SCR
[  522.141728] Step 8: Restore context
[  522.141733] Step 9: Send restart request to the target processor
[  522.141738] Step 10: Wait the target processor to restart
[  522.141744] All done! The value of SCR is 0x00000131
pi@raspberrypi:~ $
```

nailgun attack success

Question 3

0. Page table assumption. The provided source code shows that the base address of level 1 page table is `0x3200 0000`. We assume that the base address of level 2 page table is `0x3201 0000` (`0x4` - `0x8`) and `0x3203 0000` (`0x0` - `0x4`), level 3 is `0x3202 0000`.

VTCR = `0x8000 0040` so SL0 = `0b01` which means the translation table walk starts at level 1, T0SZ = `0b0000` so n = 5.

My SID is 12110644 so IPA = `0x4003 0644`.

1. Initial lookup. According to figure G4-16, descriptor address =`0x3200 0000` + (`0b01` << 3) = `0x3200 0008`.
2. Level 1 lookup. Assume the value in `0x3200 0008` is `0x3201 xxx3`. The next descriptor address = (`0x3201 xxx3` >> 12 << 12) | (`0b0000 0000 0000`) = `0x3201 x000`.
3. Level 2 lookup. Assume the value in `0x3201 x000` is `0x3202 xxx3`. The next descriptor address = (`0x3202 xxx3` >> 12 << 12) | (`0b0001 1000 0000`) = `0x3202 x180`.
4. Level 3 lookup. In protection, the last bit of this lookup should be set to 0 to indicate that this is invalid. But for now, we want to output the PA with the same value of IPA. We let the last two bits to be `0b01` so the translation finished here, and bits[39:12] should be `0x40030`. So we store `0x4003 0001` here.

Note that all the `x` value in the calculation process can simply be 0.

Question 4

IPA = `0x4211 0644` = `0b 0100 0010 0001 0001 0000 0110 0100 0100`.

1. Initial lookup. The same as Question 3, descriptor address =`0x3200 0008`.
2. Level 1 lookup. The value in `0x3200 0008` is `0x3201 xxx3`. The next descriptor address = (`0x3201 xxx3` >> 12 << 12) | (`0b0000 1000 0000`) = `0x3201 x080`.
3. Level 2 lookup. Assume the value in `0x3201 x080` is `0x3202 xxx3`. The next

descriptor address = (`0x3202 xxx3` >> 12 << 12) | (`0b1000 1000 0000`) = `0x3202 x880`.

4. Level 3 lookup. We let the last two bits to be`0b01` so the translation finished here, and bits[39:12] should be `0x42110`. So we store `0x4211 0001` here.