ARL-SR-0458 • MAY 2022

Hands-on Cybersecurity Studies: Learning about the Vehicular Controller Area Network (CAN) Bus

by Jennifer Sims, Nicholas Sims, and Jaime C Acosta

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-SR-0458 ● MAY 2022

Hands-on Cybersecurity Studies: Learning about the Vehicular Controller Area Network (CAN) Bus

Jennifer Sims and Nicholas Sims
University of Texas at El Paso

Jaime C Acosta
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
May 2022	Special Report	1 March–1 April 2022

4. TITLE AND SUBTITLE
Hands-on Cybersecurity Studies: Learning about the Vehicular Controller Area Network (CAN) Bus

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jennifer Sims, Nicholas Sims, and Jaime C Acosta

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLC-ND
Adelphi, MD 20783

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-SR-0458

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID: Jaime C Acosta, 0000-0003-2555-9989

14. ABSTRACT
Vehicular networks consist of several components that communicate across the controller area network (CAN) bus. Several open-source tools exist today to understand the functions of these components and how to better secure them. This report is a learning module or exercise that describes several of these tools to make participants more aware of security best practices and mitigations. The exercise begins with a short description of the can-utils tool, which is used to generate and set up a custom interface to speak to the CAN, as well as the Instrument Cluster Simulator tool, which aids with vehicular visualization and data generation. Afterward, these tools are used together to create a simulated environment for experimentation and testing that participants will use to complete a sequence of tasks related to CAN bus security.

15. SUBJECT TERMS
security awareness, security testing, vehicular systems, CAN bus, hands-on cybersecurity, CyberRIG, Network and Computational Sciences

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jaime C Acosta
Unclassified	Unclassified	Unclassified	UU	30	19b. TELEPHONE NUMBER (Include area code) (575) 993-2375

Contents

List of Figures

1. Introduction

Vehicles are well-integrated machines that rely on internal computing and communication across various components. These machines are becoming smarter (in many cases self-driving) and, for this reason, it is critical that they are secured against unauthorized tampering. In the domain of vehicular security, it is difficult to test various functions due to the inherent cross between the digital world and physical manipulation of a vehicle's hardware. As a result, many researchers and analysts in the field of security have developed tools to help understand the complexities of these machines as well as potential dangers and mitigations associated with unauthorized access and modification. This report provides a learning module (or exercise) aimed at introducing some of these tools and concepts to participants to help them understand security best practices and mitigations.

1.1 Exercise Overview

The exercise provided in this report focuses on publicly available tools that can be used to test and experiment with the mechanics and communications representative of a vehicular system. Unlike related security tools, these are generally new to the field and work by mimicking many of the internal systems that are present in real physical systems. However, in many circumstances, the tools are created in such a way that they can also be connected using a hardware-in-the-loop capability.

The participant will walk through the mock scenario and learn the way components interact—down to the binary level—as well as how this information may be manipulated and, as a result, change the behavior of the vehicular system. The learning module is loosely based on several publicly available tutorials.[1,2]

1.2 CAN Bus Communication

Controller area network (CAN) bus differs from many modern-day networking systems in that it does not rely on a host computer. Instead, each device on the network sends messages that are received by all the other communicators on the bus. In a modern vehicle it is not unlikely to see up to 70 such devices (which are called electronic control units [ECUs]). These are used to control various components, such as air conditioning, automatic braking systems, seat adjustments, and the engine. Consequently, data integrity is critical to maintaining the secure operation of the vehicular system. To test and secure these systems, several software and hardware tools have arisen from the community in recent years. Among these is the suite of utilities called can-utils[3] or SocketCAN, which is capable of mimicking various behaviors of the CAN bus as well as displaying,

recording, generating, and replaying CAN traffic. Additionally, these utilities provide a way to access the CAN using networking components, specifically Internet Protocol (IP) sockets among many other features.

1.3 Communication Integrity

Of specific importance in the CAN bus network is data integrity, that is, data must come from a trusted source, and it should not be susceptible to modification before reaching its intended target. Given that all devices receive all data messages, one major drawback is that if an entity is connected to the network, it may read messages sent across the bus. Therefore, data modification and replay are potential issues that may arise and it is important to understand ramifications and potential mitigations against such scenarios.

The Instrument Cluster Simulator (ICSim) tool,[4] developed by the OpenGarages organization and made available publicly, enables analysts to test and experiment with such phenomenon. It enables researchers to understand and quantify the impacts and to develop novel techniques to detect and protect against them. ICSim provides a basic interface that simulates ECUs on a vehicle and the generation of notional data. For example, a user may mimic a left blinker signal (and consequently generating notional ECU data on the CAN bus) by pressing the left keyboard arrow.

1.4 Gaining a Deeper Understanding for Improvement

During the exercise, participants will learn about both can-utils and ICSim. They will learn how to install the tools and their basic functions through a series of steps. Afterward, participants capture data by interacting with the simulation. Through a deeper analysis of the data, using trial and error, the packets responsible for several ECU functions are identified and replayed. The message structure of the data is identified and manually modified using a text editor to perform actions in the simulation through nonconventional means. Finally, the participants are asked to identify some potential mitigations that may strengthen the security of the simulated components—and how they may be automated.

2. Setup and Configuration

The setup of the exercise includes one virtual machine and several open-source software tools:

- Kali 2022.1 64-bit VM[5]
- VirtualBox 6.1.30 64-bit[6]
- SocketCAN (aka can-utils) 2021.08.0[3]
- ICSim (aka can-utils)[4]
- Nano (aka can-utils) 6.2[7]

The US Army Combat Capabilities Development Command Army Research Laboratory South Cyber Rapid Innovation Group (CyberRIG) Collaborative Innovation Testbed (CIT) is used to host the virtual machine (VM). The Kali Linux VM is hosted in a vanilla (unmodified from install) state. This machine has a network interface configured to access the Internet, which is how the participants will download and install the specific versions of SocketCAN and ICSim. The reason for locking the Kali operating system and tool versions is for compatibility—specifically, as related to library dependencies. In an alternate form, when Internet access is not enabled, this exercise is prepackaged with the downloaded install files for the tools and the dependencies are preinstalled.

In either case, participants are tasked with compiling ICSim from the source code and then subsequently setting up a testing environment. All artifacts of this learning module are packaged using the repeatable experimentation system.[8]

3. Learning Objectives

The purpose of the exercise is to gain a basic understanding of communication within vehicular components and how it differs from traditional IP networks. Therefore, traditional network troubleshooting and testing tools no longer work in this domain. Additionally, testing real vehicular systems is a nontrivial task, given the need for a relatively expensive machine, which can also be physically very dangerous. Therefore, participants gain an understanding of the reasoning and benefits of using software simulators as an alternative, yet limited, testbed for testing cybersecurity aspects of vehicular technologies. The following are the general cybersecurity topics emphasized in the exercise.

- Access controls are critical in modern systems, especially those that manage critical systems and components.

- While universal bus systems are a simpler and sometimes more efficient approach, they are also embedded with several disadvantages with respect to their security.
- Transitioning a traditionally segregated system of components into a remotely accessible scenario requires special attention due to unintended side effects.

The learning objectives associated with the tool usage during the exercise are as follows:

- **Compilation and installation of publicly available testing tools.** Participants are tasked with installing these tools and setting up an environment to conduct analysis. Subsequently, participants also learn how the inner workings of a primarily physically dependent system such as a vehicle may be simulated using the Linux Operating System, and specifically the kernel networking stack, to model realistic non-IP communications—CAN bus in this case.
- **Data collection.** Participants learn the importance of offline analysis. They use the can-utils utilities to capture several data packets while the simulation is running. Instead of having to repeat this process several times, since the data is in a static state, they can conduct a deep analysis on a nonvolatile set of data.
- **Analysis through trial and error.** Participants are tasked with isolating certain packets of interest from a collected data capture. Specifically, they must find the packets responsible for a left turn signal, a right signal, and an acceleration to 20 mph. Since the participants are not given a specification of the message structures, they learn how to logically narrow down a large capture to a single packet.

4. Exercise

The following exercise is presented to participants in a step-by-step fashion. Participants will complete several tasks through the CIT system. All the data and systems in the exercise are fictional and simulated.

This mission briefing is provided to participants:

Your team has been informed there is an increase in automobile hacking. In order to learn how to protect the vehicles in the General's Auto Company fleet, you must set up a virtual CAN network and a CAN simulation to create/send/receive data using can-utils and ICSim.

4.1 Activity 1: Set Up a Virtual CAN Network

You must create a virtual CAN network to simulate data and learn how the data is transmitted.

****

Kali Linux is a Debian-based Linux distribution loaded with numerous cybersecurity tools.

****

1) Open your Linux distribution. If you are using a default Kali instance, the username and password should both be *"kali"*.

2) Now go to your Kali VM and open a terminal. This can be done by selecting the terminal icon on the taskbar near the dragon as shown in Fig. 1.

Fig. 1 Terminal location in Kali Linux

3) We need to have root privileges for the following commands. As shown in Fig. 2, an easy way to not have to repeat using ***sudo*** is by using the following command to maintain superuser:

sudo su

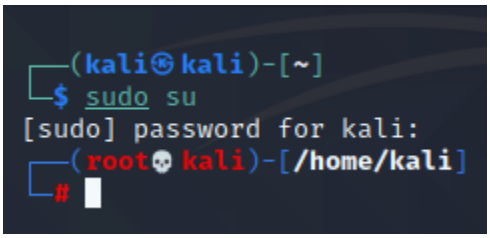

Fig. 2 sudo command

4) Load the virtual CAN (vcan) module by inserting the following command as shown in Fig. 3:

modprobe vcan

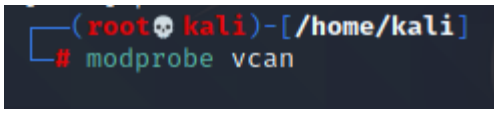

Fig. 3 Load the vcan module

5) Next, set up the virtual interface you will be using with the following commands (also shown in Fig. 4):

ip link add dev vcan0 type vcan

Press Enter

ip link set up vcan0

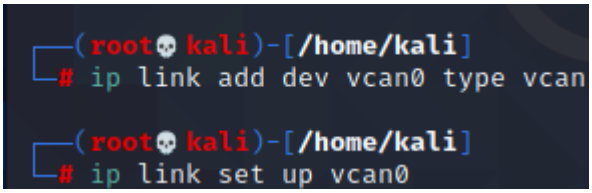

Fig. 4 Configure the vcan interface

6) To ensure the vcan0 is up and running, use the following command (also shown in Fig. 5):

ifconfig vcan0

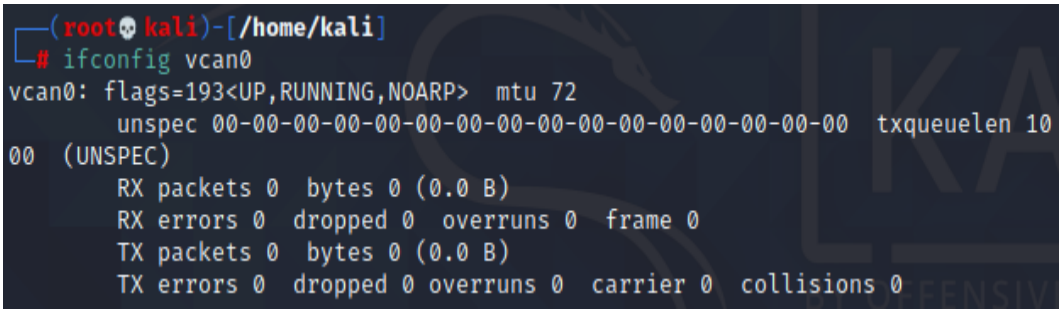

Fig. 5 Display vcan interface configuration

7) We have now verified that our virtual CAN network is up and running.

8) Exit superuser mode with ***Ctrl+D.***

4.2 Activity 2: Set Up Can-utils

We will be using the can-utils library to assist in collecting, sending, and analyzing CAN packets.

****

Can-utils is an open-source library available on GitHub. The repository contains userspace utilities for SocketCAN.

****

9) Before we install can-utils, ensure that the packages are up to date with the following command:

sudo apt update

10) To install can-utils, insert the following command; and when asked if you want to continue, insert the letter ***y*** and hit enter (also shown in Fig. 6).

sudo apt install can-utils

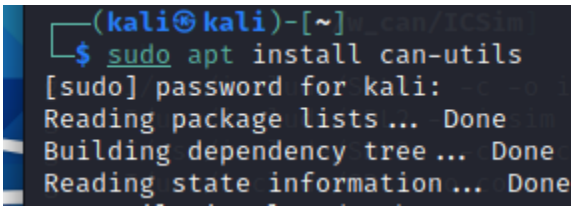

Fig. 6 Install can-utils

4.3 Activity 3: Set Up ICSim

Create virtual CAN traffic using ICSims to reverse engineer the CAN packets and learn which utilities use which packets.

****

ICSim is an open-source instrument simulator for SocketCAN and is available on GitHub.

****

11) To keep everything organized, we will create a directory called w_can by typing the following command (also shown in Fig. 7).

mkdir w_can

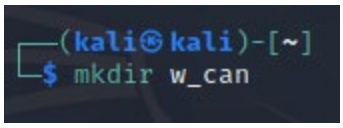

Fig. 7 Create w_can directory

12) Now change your directory to w_can (also shown in Fig. 8).

cd w_can

Fig. 8 Change into the w_can directory

13) We will need to install the library dependencies for ICSim (command and output shown in Fig. 9).

sudo apt install libsdl2-dev libsdl2-image-dev -y

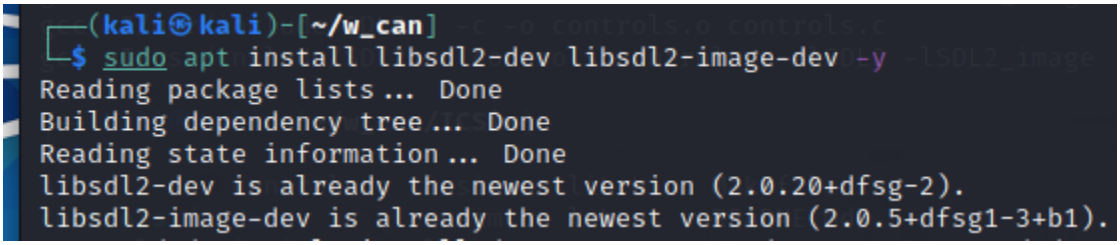

Fig. 9 Install ICSim dependencies

14) We need to clone ICSim into the w_can directory (also shown in Fig. 10).

git clone https://github.com/zombieCraig/ICSim.git

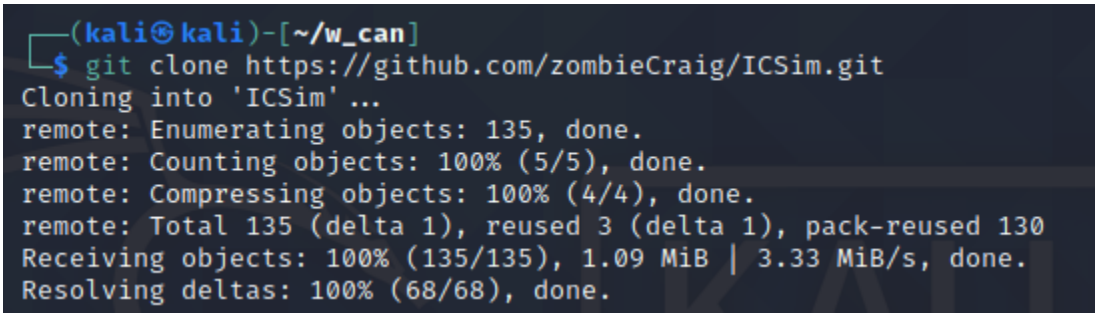

Fig. 10 Download ICSim source code

15) Verify everything was cloned by using the *ls* command to check if the ICSim directory was installed. We can see the ICSim directory is in the w_can directory (also shown in Fig. 11).

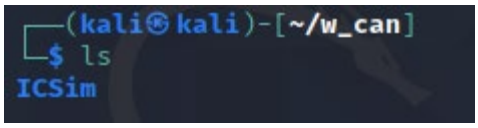

Fig. 11 List the ICSim data directory

16) Change into the ICSim directory (also shown in Fig. 12).

cd ICSim

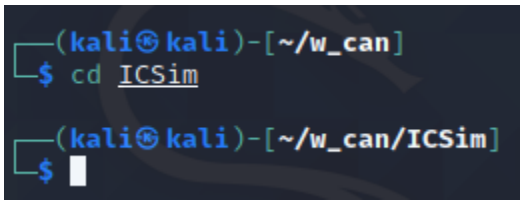

Fig. 12 Navigate to the ICSim data directory

17) Now we need to compile the ICSim source tree (command and results shown in Fig. 13).

make

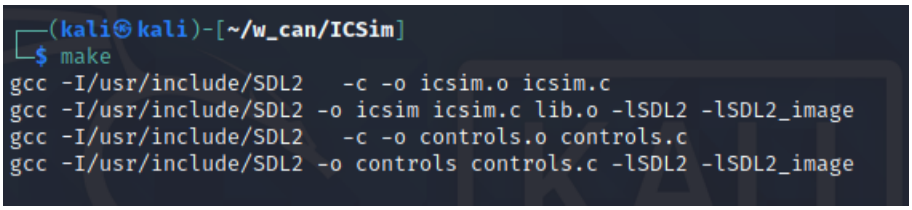

Fig. 13 Compile the ICSim source code

4.4 Activity 4: Create CAN Traffic with ICSim

We will use ICSim to create the CAN traffic, which we will monitor with can-utils. The ICSim simulator controls are as follows.

****

The following chart shows how to simulate the desired behavior with the CAN bus control panel.

ICSim Actions	Keys
Accelerate	Up Arrow (↑)
Left/Right Turn Signal	Left/Right Arrow (←/→)
Unlock Front L/R Doors	Right-Shift+A, Right-Shift+B
Unlock Back L/R Doors	Right-Shift+X, Right-Shift+Y
Lock All Doors	Hold Right Shift Key, Tap Left Shift
Unlock All Doors	Hold Left Shift Key, Tap Right Shift

****

18) Start by running the ICSim within the w_can/ICSim directory (command and result shown in Fig. 14).

./icsim vcan0

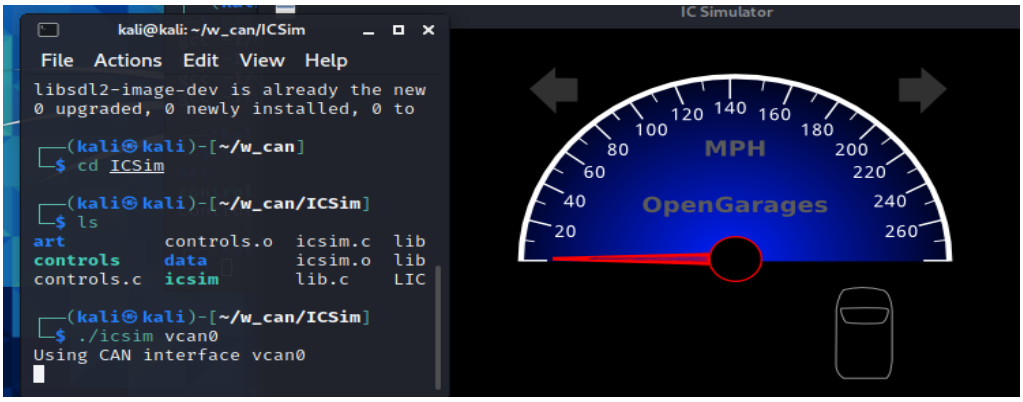

Fig. 14 ICSim speed display

19) Now open another terminal and go into the ICSim directory (also shown in Fig. 15).

cd w_can/ICSim

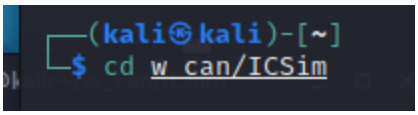

Fig. 15 Navigate to the ICSim data directory

20) We will now start the CAN bus control panel (command and result shown in Fig. 16).

./controls vcan0

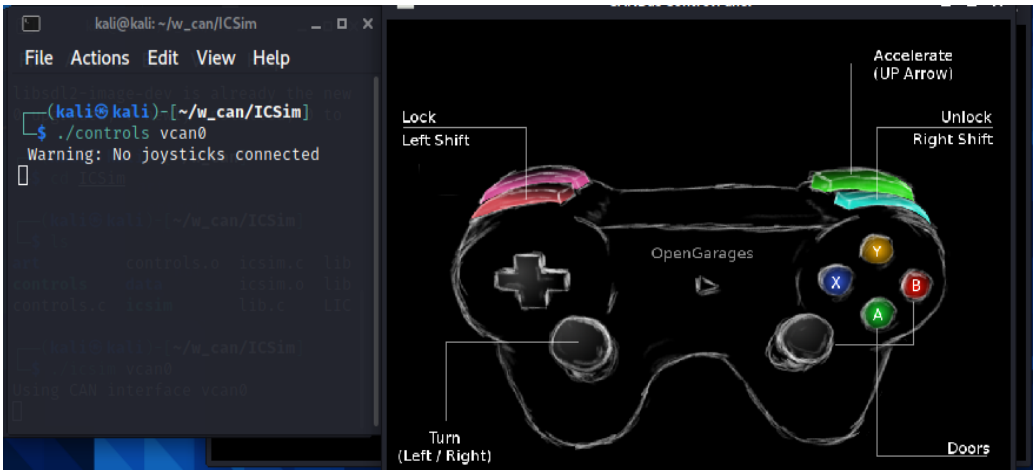

Fig. 16 ICSim control display

21) Next, open another terminal and change directory to ICSim with the same command from step no. 16.

22) Start collecting the traffic and send the output to a file called ***turn***:

candump -L vcan0 > turn

23) While ***candump*** is running, go to your CAN bus control panel and hold down the left arrow key on your keyboard until you see the left turn signal becomes green on the ICSim (shown in Fig. 17).

Fig. 17 ICSim left signal activated

24) Next, stop the ***candump*** by selecting the terminal running ***candump*** and pressing ***Ctrl+C***.

25) Check if data was sent to the ***turn*** file we created in step no. 22. You should see the dumped data on the terminal (command and results are shown in Figs. 18 and 19).

cat turn

Fig. 18 cat command to display turn signal data

```
(1648586480.791249) vcan0 294#040B0002CF5A002C
(1648586480.791256) vcan0 21E#03E8374522062F
(1648586480.792595) vcan0 183#0000000600001032
(1648586480.792638) vcan0 244#0000000148
(1648586480.794280) vcan0 143#6B6B00FF
(1648586480.795351) vcan0 095#800007F400000026
(1648586480.797424) vcan0 166#D0320036
(1648586480.798597) vcan0 158#0000000000000037
(1648586480.799673) vcan0 161#000005500108003A
(1648586480.799693) vcan0 191#010010A1410029
```

Fig. 19 Turn signal data snippet

26) We are going to move the ***turn*** file into a directory called ***can_testing***. Create the directory and then move the ***turn*** file (also shown in Fig. 20).

mkdir can_testing

mv turn can_testing

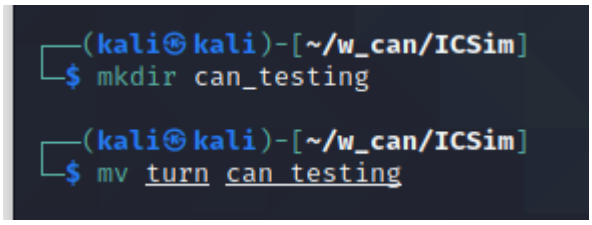

Fig. 20 Create folder and move data

27) Change into your ***can_testing*** directory with ***cd*** and ensure the ***turn*** file is in that directory with ***ls*** (also shown in Fig. 21).

cd can_testing

ls

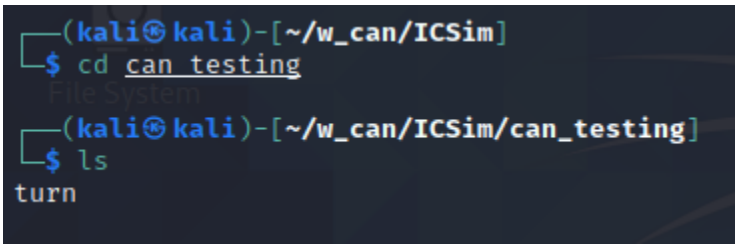

Fig. 21 Navigate to the test data folder and list contents

****

We want to see what CAN ID is used for the turn signal functionality on the CAN bus. Here we are going to use the CAN bus simulation to identify individual packets. To do this we are going to manually find the packet that causes the left turn signal to light up. For this we will be using ***canplayer*** and then analyzing the ***turn*** file.

****

28) Make sure the dumped file has the command to turn on the left signal by using ***canplayer***. Run the command and watch the ICSim to make sure the left turn signal turns on (also shown in Fig. 22).

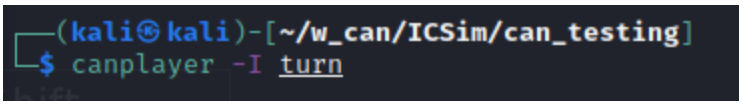

Fig. 22 Play data for turn signal

29) Find out how many lines are in the ***turn*** file. The number of lines in your ***turn*** file might be different, but the same method can be applied to get the correct CAN ID (command and results shown in Fig. 23).

wc -l turn

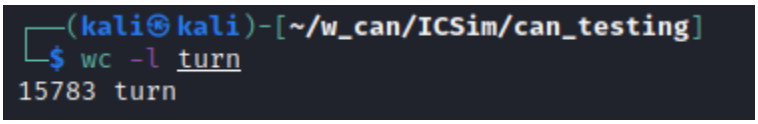

Fig. 23 Count the number of lines in turn signal data file

30) The idea is to break up the file into manageable pieces. Here, the file is approximately 15,000 lines. For this example, the file is ***split*** at 5000 lines. Use discretion based on the size of the file you are splitting.

split -l5000 turn

31) Get the names of the split files by seeing the files in the directory with ***ls.*** We can see four new files resulting from ***split***. These are named automatically by the ***split*** tool (results shown in Fig. 24).

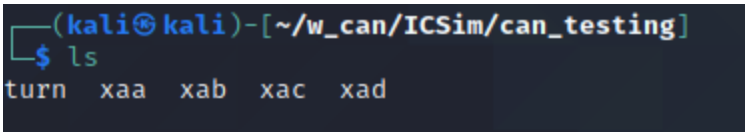

Fig. 24 List split turn signal data

32) Next, we are going to play back the packets in each new file and see which of them triggers the left turn signal.

canplayer -I <filename>

33) In this example, the turn signal was in file ***xab***. For whichever file triggers the signal (and thus has the packet we want), rename said file as ***turn***. Afterward, remove the remaining split files (also shown in Fig. 25).

mv xab turn

ls

rm xa*

ls

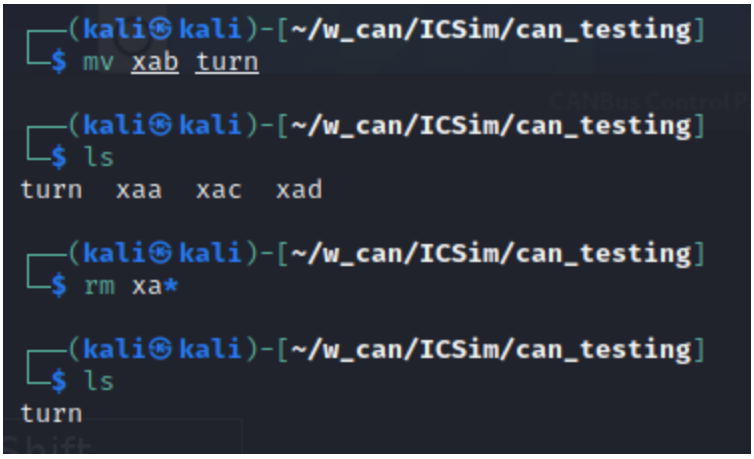

Fig. 25 Isolate turn signal message

34) Now we will repeat Steps 28–33 while changing the number of ***split*** lines (depending on the size of each new ***turn*** we create). We repeat this until the file is small enough to split at a single line. This will leave us with a file containing the packet that activates the left turn signal. Looking at the final single-packet file, we can see the CAN ID for the turn signal. We can use ***cat*** to see the file contents (also shown in Fig. 26).

cat turn

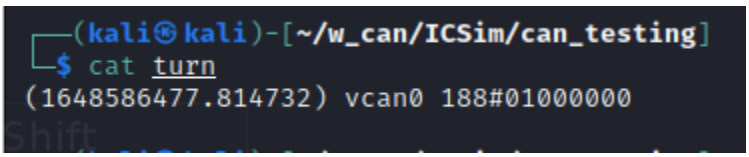

Fig. 26 Display contents of turn signal message

a. What is the ID no. associated with the turn signal? _____

35) Now that we know the ID no., we can filter incoming traffic and see what data changes when the turn signal activates. Use ***cansniffer*** to filter only packets with the ID no. we care about. Run the ***cansniffer*** command, then press the ***n <ENTER>*** (this will tell ***cansniffer*** to show none of the packets). You should see the column information, but nothing else (results are shown in Fig. 27).

cansniffer vcan0

n + enter

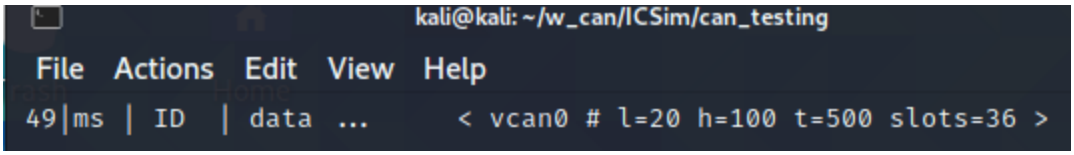

Fig. 27 Sniff message associated with turn signal data

36) Now, enter ***+<CAN ID><ENTER>***. Then click on the CAN bus control panel and hold down the left arrow. We see the packet with the first visible byte of message ID +188 value set to 01 (as shown in Fig. 28).

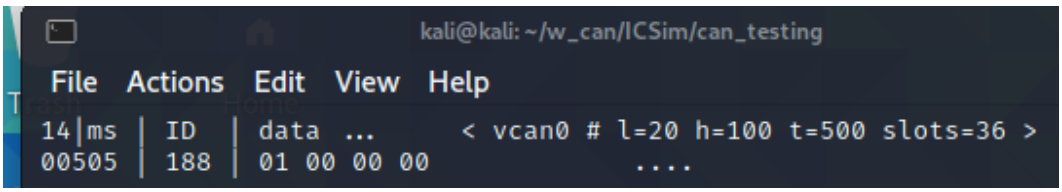

Fig. 28 Display of left turn signal data within message

37) Now press the right arrow key on the keyboard and we can see that the first visible data byte value is now 02 (as shown in Fig. 29).

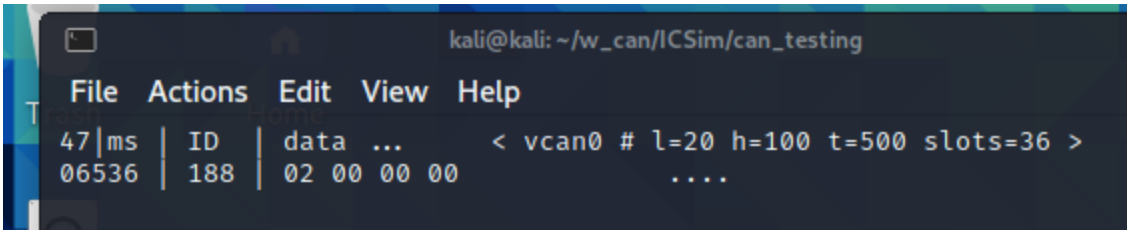

Fig. 29 Display of right turn signal data within message

38) Congratulations! You have found the ID and the specific byte for the left and right turn signals. Press **Ctrl+C** to stop ***cansniffer***.

4.5 Activity 5: Find the Arbitration ID for Acceleration

We are going to run *candump* while accelerating and dump the data into the *accel* file.

39) Start by dumping the data to a file called ***accel***.

candump -L vcan0 > accel

40) Go back to the CAN bus control panel and hold down the "up" arrow key for acceleration. After accelerating to approximately 20 mph (as shown in Fig. 30), stop ***candump*** with ***Ctrl+C.***

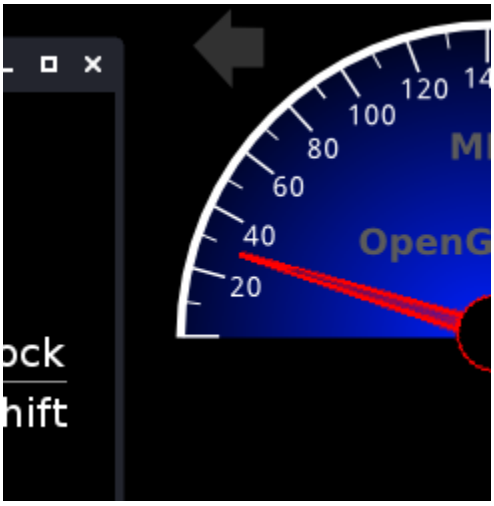

Fig. 30 ICSim speedometer at 25 mph

41) We need a baseline to test where the acceleration is 0 versus 20 mph so get noise data from ***candump*** and store it into a ***control*** file. Press ***Ctrl+C*** to stop ***candump***. You can always check contents of the file with ***cat <filename>*** as shown in Fig. 31.

```
┌──(kali㉿kali)-[~/w_can/ICSim/can_testing]
└─$ candump -L vcan0 > control
^C
```

Fig. 31 Capture speed control data

42) Exit your CAN bus controller so you can test your ***accel*** file with ***canplayer***.

43) Repeat Steps 29–33 with your ***accel*** file and identify a packet that will make the simulation accelerate to 20 mph.

44) When you have your single-line file, run your ***control*** file to reset to 0 mph (also shown in Fig. 32).

canplayer -I control

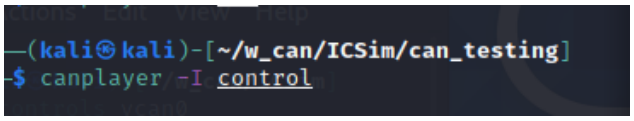

Fig. 32 Play speed control data

45) Now double check that the single-line file sets the speedometer to 20 mph. Your results should resemble that shown in Fig. 33.

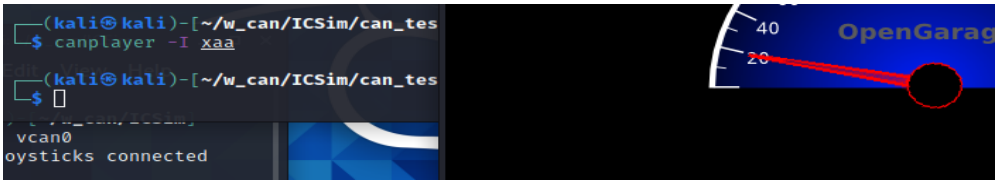

Fig. 33 Play single message for speed control data

46) Move your one-line file into your ***accel*** file.

47) Look into your ***accel*** file to see what the CAN ID is for associated with acceleration.

48) Open your ***accel*** file in ***nano***. You should see something that resembles Fig. 34.

nano accel

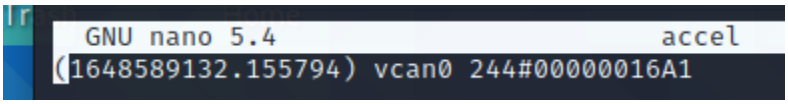

Fig. 34 Display acceleration message

49) Change the last four numbers to 0000, then save the file with the following commands.

Ctrl + X

Y

Enter

50) Feed ***canplayer*** your modified accel file (also shown in Fig. 35).

canplayer -I accel

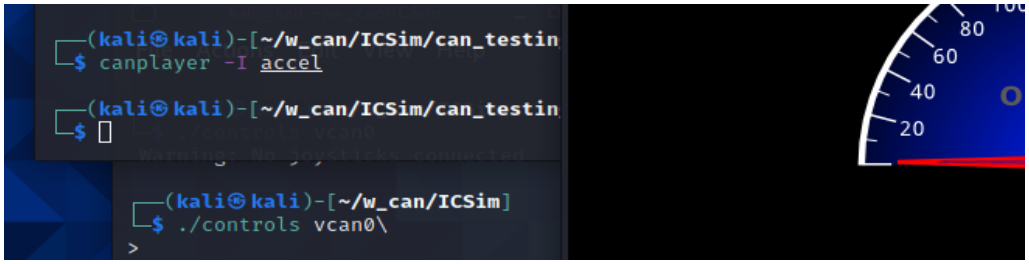

Fig. 35 Play acceleration data; speedometer shows 0 mph

51) We can see that the speedometer went back to 0 mph. Now reopen the ***accel*** file and edit the last four digits back to 16A1 (your *mileage* may vary 😊). After saving and inserting the ***accel*** file into ***canplayer*** again, we can see the speedometer went straight to 20 mph (also shown in Fig. 36).

canplayer -I accel

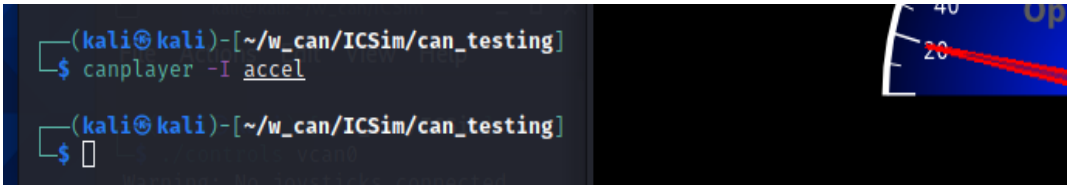

Fig. 36 Speedometer at 20 mph after message insertion

52) Bonus: What does the speedometer do when the last four digits are 0?

4.6 Activity 6: Hardening

So far, you have experimented with a simulation of vehicular components and the way they communicate. Now it is your job to identify ways to further strengthen the security of such systems.

53) How could the CAN bus protect against unauthorized modification and injection of messages on the network?

54) Come up with ideas for strengthening the CAN bus.

55) How could your ideas from the previous question be implemented in the software and then into a physical vehicular system?

You have successfully completed the exercise.

5. Conclusion

In this report, we provided a basic learning module that introduces participants to vehicular systems and how they are tested from the security perspective. By providing a hands-on exercise, participants will gain knowledge about the risks associated with unauthenticated access to CAN bus networks. The open-source tools used in the exercise provide a unique perspective that will help in several ways. Researchers can gain an understanding of the issues that need scientific solutions in the near-term and through the long-term. Developers can use these tools to test new systems and to assess their security posture. End-users become aware of the potential dangers of having components they use being accessible through physical or remote means.

This exercise is used for training and awareness, but also to fuel research in cybersecurity defense focused on vehicular and non-IP-based networked systems.

6. References

1. Berg J. CAN communication tutorial, using simulated CAN bus. 2016 [accessed 2022 Apr 26]. https://sgframework.readthedocs.io/en/latest/cantutorial.html.

2. X Sophia. Hax and Furious: an introduction to CAN bus hacking with ICSim, 2020 Sep 11 [accessed 2022 Apr 26]. https://securityqueens.co.uk/hax-and-furious-an-introduction-to-can-bus-hacking-with-icsim/.

3. Linux-CAN / SocketCAN user space applications. 2022 April 12 [accessed 2022 Apr 26]. https://github.com/linux-can/can-utils.

4. Instrument cluster simulator. 2020 June 11 [accessed 2022 Apr 26]. https://github.com/zombieCraig/ICSim.

5. Kali Linux 2022.1 release (visual updates, Kali everything ISOs, legacy SSH). 2022 Feb 14 [accessed 2022 Apr 26]. https://www.kali.org/blog/kali-linux-2022-1-release/.

6. VirtualBox. n.d. [accessed 2022 Apr 26]. https://www.virtualbox.org/.

7. GNU Nano. 2022 Feb 18 [accessed 2022 Apr 26]. https://www.nano-editor.org/.

8. Acosta JC, Clarke L, Medina S, Akbar M, Hossain MS, Free-Nelson F. Repeatable experimentation for cybersecurity moving target defense. In: Garcia-Alfaro J, Li S, Poovendran R, Debar H, Yung M, editors. Security and Privacy in Communication Networks. SecureComm 2021; 2021 Nov 3. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 398. Springer. p. 82–99. doi.org/10.1007/978-3-030-90019-9_5.

List of Symbols, Abbreviations, and Acronyms

aka	also known as
ARL	Army Research Laboratory
CAN	controller area network
CIT	Collaborative Innovation Testbed
CyberRIG	Cyber Rapid Innovation Group
DEVCOM	US Army Combat Capabilities Development Command
ECU	electronic control unit
ICSim	Instrument Cluster Simulator
ID	identification
IP	Internet Protocol
mph	miles per hour
vcan	virtual CAN
VM	virtual machine

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD DCI TECH LIB
2 (PDF)	DEVCOM ARL FCDD RLC ND J CLARKE J ACOSTA