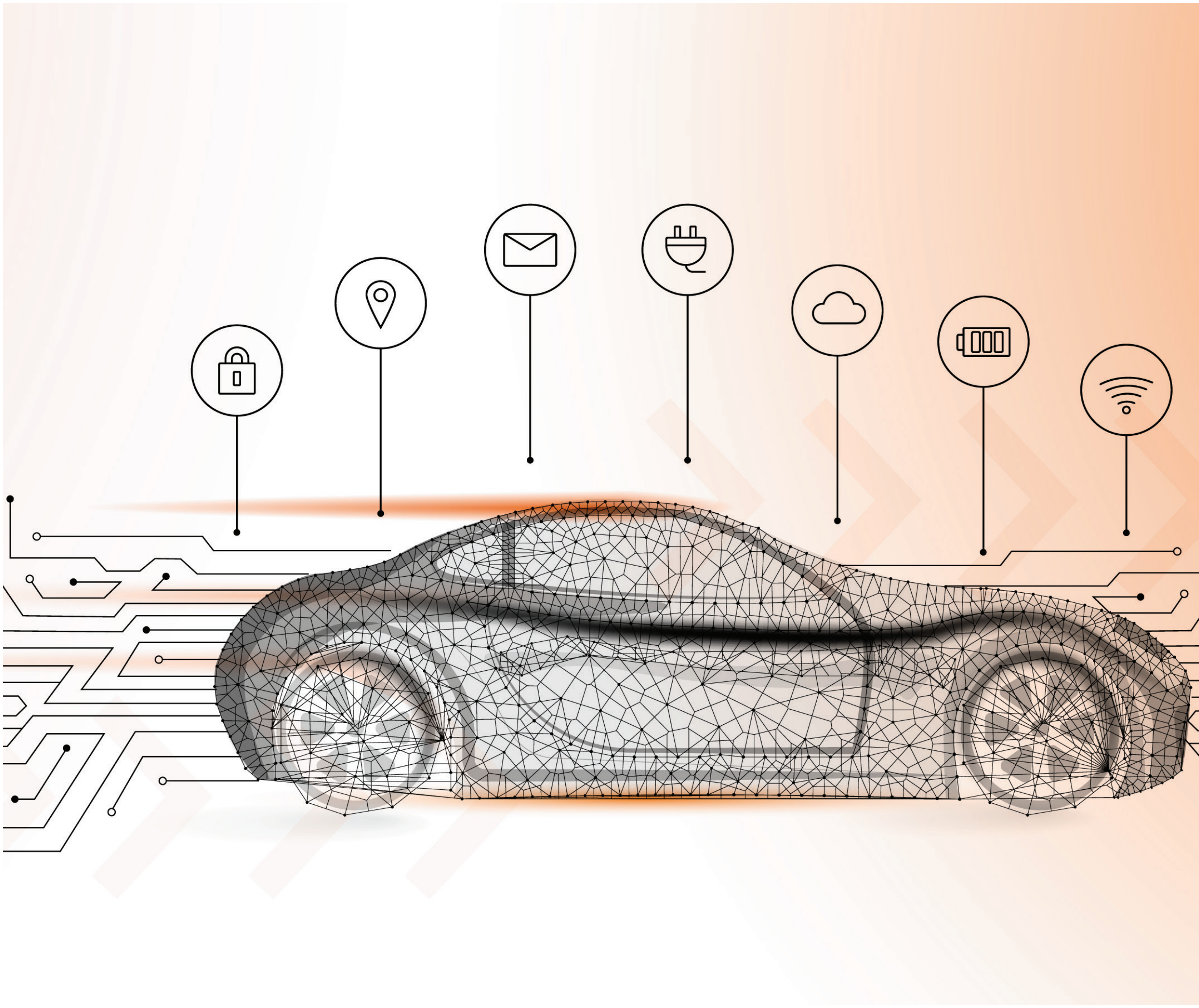
Project 2132 | May 2022

SJSU SAN JOSÉ STATE UNIVERSITY

MTI MINETA TRANSPORTATION INSTITUTE

Protecting Our Community from the Hidden Vulnerabilities of Today's Intelligent Transportation Systems

Shahab Tayeb, PhD



CSUTC California State University Transportation Consortium

CALIFORNIA STATE UNIVERSITY LONG BEACH

Report 22-14

Protecting Our Community from the Hidden Vulnerabilities of Today's Intelligent Transportation Systems

Shahab Tayeb, PhD

May 2022

A publication of the
Mineta Transportation Institute
Created by Congress in 1991

College of Business
San José State University
San José, CA 95192-0219

TECHNICAL REPORT DOCUMENTATION PAGE

1. Report No. 22-14	**2. Government Accession No.**	**3. Recipient's Catalog No.**	
4. Title and Subtitle Protecting Our Community from the Hidden Vulnerabilities of Today's Intelligent Transportation Systems		**5. Report Date** May 2022	
		6. Performing Organization Code	
7. Authors Shahab Tayeb, PhD		**8. Performing Organization Report** CA-MTI-2132	
9. Performing Organization Name and Address Mineta Transportation Institute College of Business San José State University San José, CA 95192-0219		**10. Work Unit No.**	
		11. Contract or Grant No. ZSB12017-SJAUX	
12. Sponsoring Agency Name and Address State of California SB1 2017/2018 Trustees of the California State University Sponsored Programs Administration 401 Golden Shore, 5th Long Beach, CA 90802 U.S. Department of Transportation Office of the Assistant Secretary for Research and Technology University Transportation Centers Program 1200 New Jersey Avenue, SE Washington, DC 20590		**13. Type of Report and Period Covered**	
		14. Sponsoring Agency Code	
15. Supplemental Notes			
17. Key Words Controller Area Network Bus; Internet of Vehicles; Security, Embedded Systems, Cyber Physical Systems	**18. Distribution Statement** No restrictions. This document is available to the public through The National Technical Information Service, Springfield, VA 22161.		
19. Security Classif. (of this report) Unclassified	**20. Security Classif. (of this page)** Unclassified	**21. No. of Pages** 25	**22. Price**

16. Abstract

The ever-evolving technology interwoven into the transportation industry leaves it frequently at risk for cyber-attacks. This study analyzes the security of a common in-vehicle network, the Controller Area Network (CAN), standard in most vehicles being manufactured today. Like many other networks, CAN comes with inherent vulnerabilities that leave CAN implementations at risk of being targeted by cybercriminals. Such vulnerabilities range from eavesdropping, where the attacker can read the raw data traversing the vehicle, to spoofing, where the attacker can place fabricated traffic on the network. The research team initially performed a simulation of CAN traffic generation followed by hardware implementation of the same on a test vehicle. Due to the concealed and untransparent nature of CAN, the team reverse-engineered the missing parameters through a series of passive "sniffing attacks" (attacks using data reading utilities called packet sniffers) on the network and then demonstrated the feasibility of the attack by placing fabricated frames on the CAN.

Form DOT F 1700.7 (8-72)

DOI: 10.31979/mti.2022.2132

Mineta Transportation Institute
College of Business
San José State University
San José, CA 95192-0219

Tel: (408) 924-7560
Fax: (408) 924-7565
Email: mineta-institute@sjsu.edu

transweb.sjsu.edu/research/2132

Acknowledgments

The author acknowledges the contributions of faculty collaborators Drs. Youngwook Kim, Matin Pirouz, Reza Raeisi, Aaron Stillmaker, and Aly Tawfik.

CONTENTS

LIST OF FIGURES

Executive Summary

This project highlights the security vulnerabilities and discusses the attack surface of the Controller Area Network (CAN) bus. As the de facto technology for In-Vehicle Network (IVN) backbones, the CAN bus can be found in all automobiles nowadays. Therefore, any vulnerabilities in such a system have widespread implications for the safety and security of society. Our simulation findings are supplemented by hardware implementation of particular event triggers on an actual vehicle. We demonstrate the lack of security by eavesdropping, reverse engineering, and spoofing CAN frames. These measures shine a light on the need for securing CAN bus by redesign.

1. Introduction

The Controller Area Network (CAN) bus was originally developed in the 1980s but became standard in most vehicles by 1993. The CAN bus provides a solution to vehicles' increasingly complex wiring and communication needs as vehicles adapt more and more electrical subsystems. Each electric subsystem in a vehicle is controlled by an Electronic Control Module (ECM). Most subsystems require some degree of communication with other subsystems to coordinate activity or trigger actuators. The original vehicle network design used point-to-point wiring to connect each ECM to other ECMs, as shown in the left panel of Figure 4. As the number of ECMs in vehicles continues to grow, this method requires the addition of more complex wiring. The CAN bus offered a much more efficient solution wherein all ECMs are connected through a single two-wire bus as shown in the right panel of Figure 1. All communications on the bus are broadcast to all the other ECMs, which are also referred to as nodes on the CAN bus. The CAN bus is a multi-master broadcast network, meaning that all messages broadcast from one node traverse the network and are received by every other node. Each node is responsible for determining whether the messages received need action based on the Arbitration ID information in the transmitted message, also known as the CAN ID.

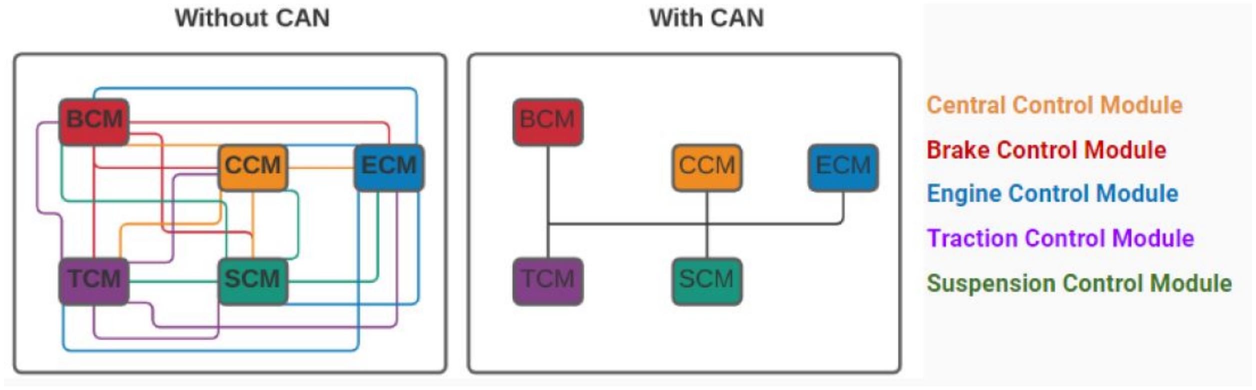


Figure 1. Backbone of the Controller Area Network Bus

The structure of a typical CAN frame can be broken up into several specific segments, as shown in Figure 2. The first component of the CAN frame is the start of frame (SOF) bit, which only indicates the beginning of the incoming frame. The next 11 bits specify the CAN ID, and each node will analyze this ID to determine whether the message is intended for it. On the CAN bus, a lower CAN ID has a higher priority. The IDs assigned to specific nodes are up to the system designer's discretion. The general practice is to assign higher-priority IDs to the more critical nodes such as powertrain, anti-lock braking, or emergency systems. It is also common practice to assign IDs that are close in range to nodes with similar functionality. The next bit is the remote transmission request bit (RTR), which indicates whether the message is being sent to or requested from the node indicated by the CAN ID. The next two bits control whether the frame is a standard or extended frame. The data length code consists of four bits that indicate to the receiving node how many bytes to expect in the payload of the frame. The next 16 bits are used for the calculation of a cyclic redundancy check for transmission accuracy, followed by a two-bit acknowledgment, and the final seven bits are reserved to show the end of the frame.

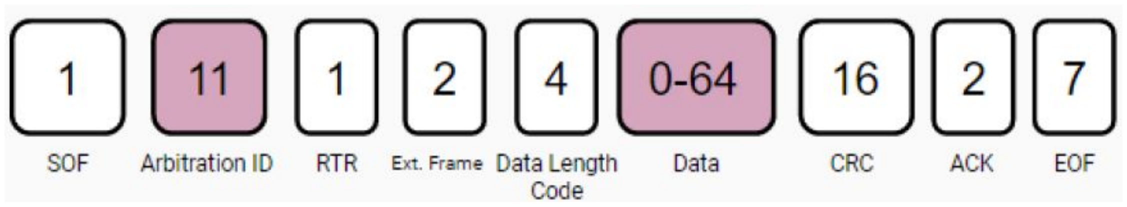


Figure 2. CAN Frame Message Format

2. Methodology

There are several options for connecting to the CAN bus in a modern vehicle. These range from physical access connections to short- and long-range wireless options. The system design requires a physical presence within the automobile, so we configured a physical connection to the CAN bus. The standard physical access point to the CAN bus in a vehicle is the OBDII port, interchangeably referred to as the DLC (diagnostic link connector) by many automobile OEMs (original equipment manufacturers). The OBDII port is typically located on the lower left or right side near the steering column of the vehicle. It has been standard on all automobiles since 1996 and has a sixteen-pin connection. Although it is an industry-standard connection, car manufacturers typically utilize only about half of the pins as part of the standard, and several pins are available for use at the manufacturer's discretion. The standard pinout is shown in Figure 3 and Figure 4.

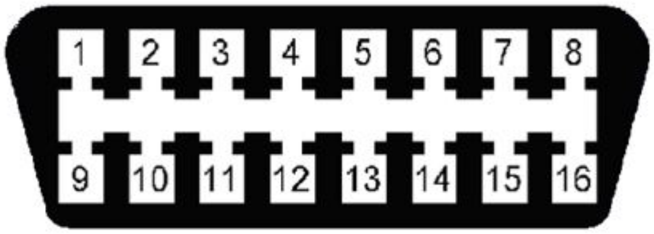
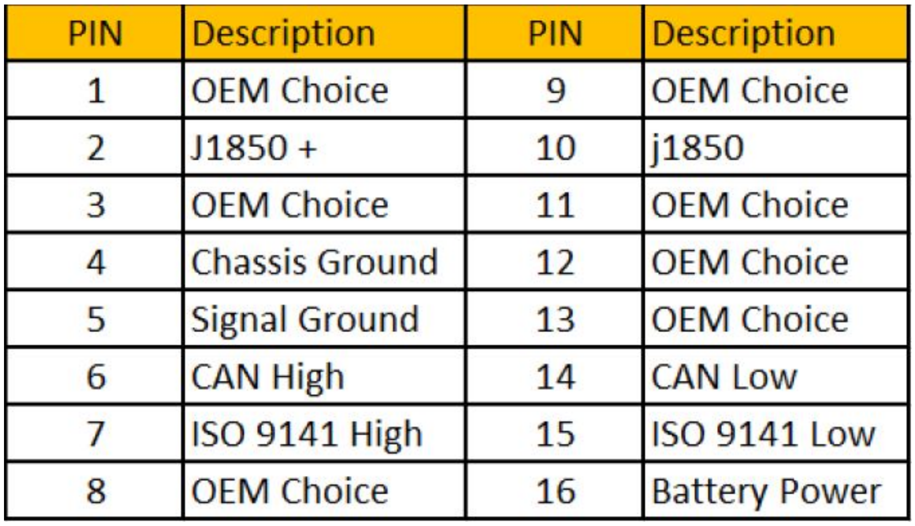


Figure 3. OBDII Pinout



PIN	Description	PIN	Description
1	OEM Choice	9	OEM Choice
2	J1850 +	10	j1850
3	OEM Choice	11	OEM Choice
4	Chassis Ground	12	OEM Choice
5	Signal Ground	13	OEM Choice
6	CAN High	14	CAN Low
7	ISO 9141 High	15	ISO 9141 Low
8	OEM Choice	16	Battery Power

Figure 4. OBDII Pin Functions

Miller previously found that Ford engineers employ a secondary middle-speed CAN bus operating at 125 kbps for lower-priority ECUs. This secondary CAN bus is accessible through the OBD II port on pins 3 and 11 for CAN high and low signals, respectively.

When the CAN bus was originally designed, security was not a top priority. The CAN bus fails to satisfy even the most critical security pillars of confidentiality, integrity, and availability. Data on the CAN bus are not encrypted in any fashion. Anyone or any device with access to the network can freely eavesdrop and sniff traffic. Furthermore, the CAN bus does not support any kind of authorization or authentication features. Additionally, there is no way to verify that any specific message was sent or received by a specific device. Finally, CAN bus frames are delivered using arbitration ID, which handles priority on the bus. Lower IDs have higher priority which ensures that critical high-priority messages are delivered first. However, there is no congestion or traffic management on the bus to protect against starvation of the lower-priority messages, which can generate vulnerabilities for availability.

With these limitations, it is possible to physically access the CAN bus network as a new node and read all traffic occurring within it. A typical node on the CAN bus network consists of a microcontroller, a CAN controller, and a CAN transceiver. Most easily explained in terms of the OSI model, the microcontroller handles the functions at the application layer including user input and drivers for specific actuators. The CAN controller communicates between the application layer and the data link layer transmitting and receiving the necessary information from the CAN transceiver, which translates the data into the physical layer for transfer over the bus. This typical node structure is shown in Figure 5.

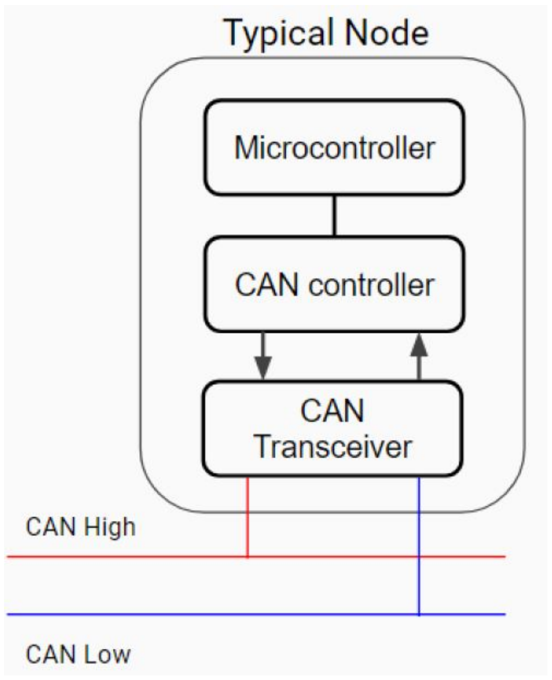


Figure 5. CAN Bus Node

The CAN controller and CAN transceiver for the system are the MCP2515 and MCP2562 integrated circuits, respectively. These are packaged and assembled along with an integrated Switched Mode Power Supply (SMPS) on the PiCan3 Raspberry Pi Hat from SK Pang electronics. This hardware is attached to the Raspberry Pi 40 pin GPIO connector and sits atop the Raspberry Pi board. A cable with a standard OBD II female connector on one side and exposed open wires on the other side is then connected to the input screw terminals on the PiCAN3. The wire corresponding to pin 16 is connected to the 12V+ terminal, and the wire from pin 4 (chassis ground) is connected to the ground terminal. To interface with the medium-speed CAN bus in the test vehicle, the wire for pin 3 is connected to the CAN high terminal, and the wire for pin 11 is connected to CAN low terminal, as shown in Figure 6.

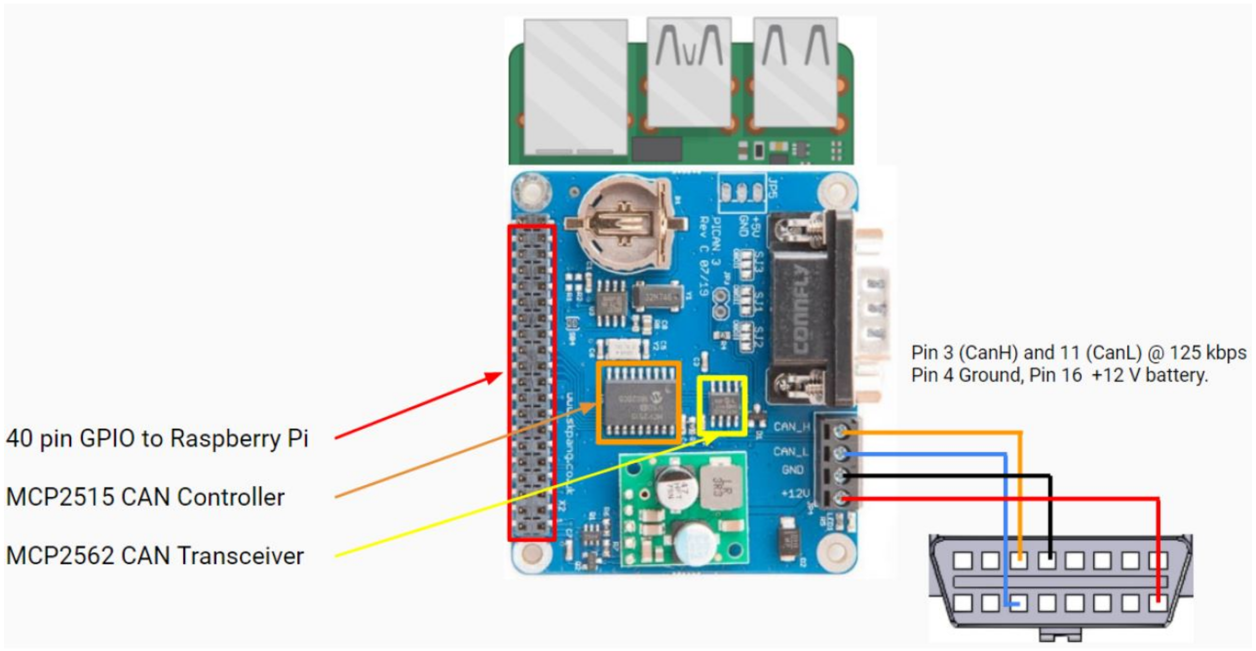
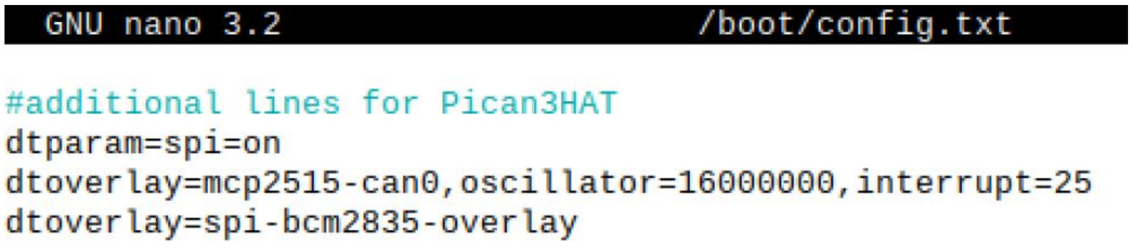

Pin 3 (CanH) and 11 (CanL) @ 125 kbps
Pin 4 Ground, Pin 16 +12 V battery.
40 pin GPIO to Raspberry Pi
MCP2515 CAN Controller
MCP2562 CAN Transceiver
CAN_H
CAN_L
GND
+12V

Figure 6. PiCAN3 Wiring

To enable the PiCan in the Raspbian OS, the default boot configuration file of the Raspberry Pi was modified as shown in Figure 7.



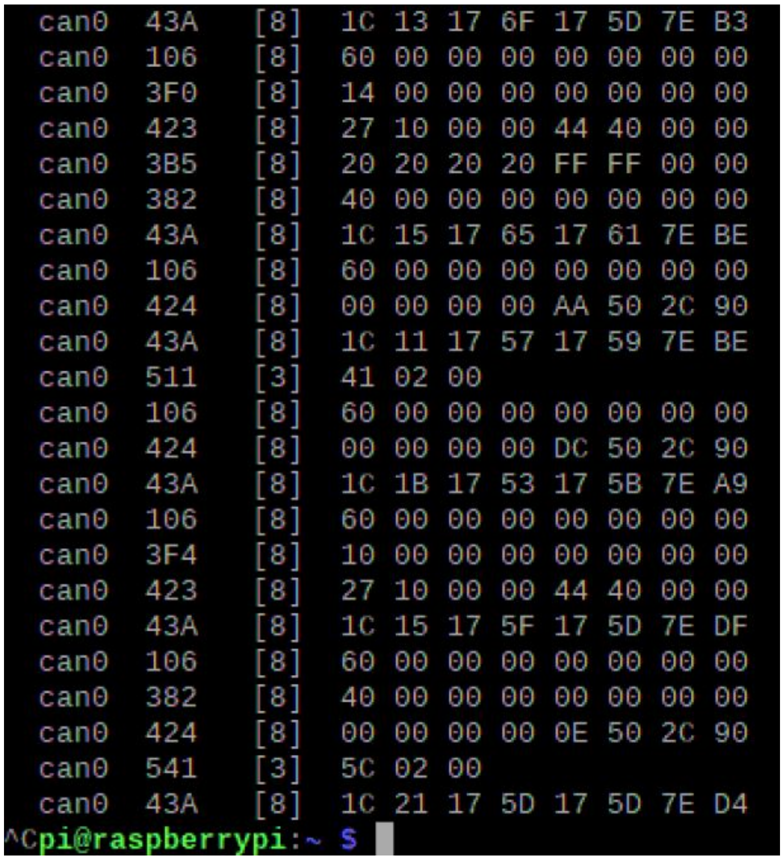
```
  GNU nano 3.2                    /boot/config.txt

#additional lines for Pican3HAT
dtparam=spi=on
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
dtoverlay=spi-bcm2835-overlay
```

Figure 7. Boot Configuration

Once in the vehicle settings, the Raspberry Pi will not have a direct connection to any video output display or input devices. The sole connection is from the PiCan3 to the vehicle ODBII port. Rather than directly connecting a monitor and keyboard to the device, a VNC server on the Raspberry Pi can enable access to the Pi over the WiFi network using the VNC software. To initialize and establish the socket interface on the Raspberry Pi, the command syntax is “/sbin/ip link set can0 up type can bitrate 1250000." In the preceding command, "can0" is the network label that is used moving forward to refer to the vehicle CAN bus network. The baud rate is provided in bits and was set to 1250000 corresponding to the expected speed of the medium-speed CAN bus (125 kbps).

For interacting with a CAN bus protocol, the Linux operating system (OS) has a very capable set of utilities available through the open-source SocketCAN drivers. The drivers and related CAN-Utils provide support for the CAN bus at the network level and are installed from the repository using the command "sudo apt-get install can-utils." The drivers enable access to a CAN network by initializing a socket in a very similar fashion to a TCP/IP connection and binding the socket to an interface. Once bound, the socket can be used for reading or writing data.

CAN-Utils provide several different tools for interacting with a CAN bus network. A couple of those tools are used to complete this project and will be discussed in the order in which they are required. To read data from the CAN bus, the candump tool is used to dump all traffic on the specified network to the connected display. This tool also supports filtering and logging to an output file, but only the live display is necessary to verify an active connection to the CAN bus. Once the vehicle ignition is set to the accessory state to make sure there is traffic on the CAN bus, the "candump can0" command is used to produce the output in Figure 8.



```
  can0  43A   [8]  1C 13 17 6F 17 5D 7E B3
  can0  106   [8]  60 00 00 00 00 00 00 00
  can0  3F0   [8]  14 00 00 00 00 00 00 00
  can0  423   [8]  27 10 00 00 44 40 00 00
  can0  3B5   [8]  20 20 20 20 FF FF 00 00
  can0  382   [8]  40 00 00 00 00 00 00 00
  can0  43A   [8]  1C 15 17 65 17 61 7E BE
  can0  106   [8]  60 00 00 00 00 00 00 00
  can0  424   [8]  00 00 00 00 AA 50 2C 90
  can0  43A   [8]  1C 11 17 57 17 59 7E BE
  can0  511   [3]  41 02 00
  can0  106   [8]  60 00 00 00 00 00 00 00
  can0  424   [8]  00 00 00 00 DC 50 2C 90
  can0  43A   [8]  1C 1B 17 53 17 5B 7E A9
  can0  106   [8]  60 00 00 00 00 00 00 00
  can0  3F4   [8]  10 00 00 00 00 00 00 00
  can0  423   [8]  27 10 00 00 44 40 00 00
  can0  43A   [8]  1C 15 17 5F 17 5D 7E DF
  can0  106   [8]  60 00 00 00 00 00 00 00
  can0  382   [8]  40 00 00 00 00 00 00 00
  can0  424   [8]  00 00 00 00 0E 50 2C 90
  can0  541   [3]  5C 02 00
  can0  43A   [8]  1C 21 17 5D 17 5D 7E D4
^Cpi@raspberrypi:~ $
```

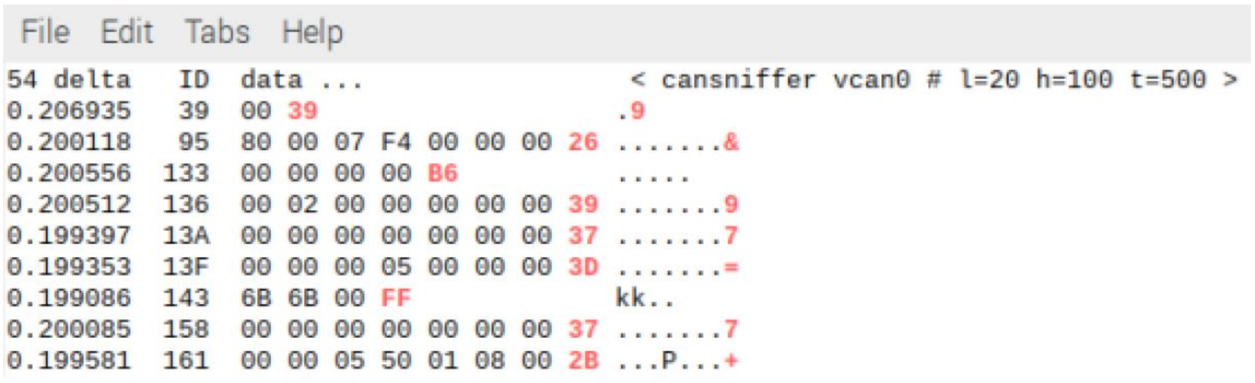
Figure 8. Captured CAN Traffic

The candump command is useful for verifying that data are being captured from the CAN bus. However, the typical frame length is just 108 bits. Even at 125 kbps, a relatively low speed by today's standards, there can be up to 1,150 frames transmitted in just a single second. The rate of change is far too fast to observe in real time. One solution to this is to use the candump logging feature of "-l" to log all the CAN data being received. Even with a log, there is typically so much traffic on the bus that trying to find any specific information would be impractical. Instead, it is useful to first identify specific CAN IDs of interest and log the traffic with a filter to exclude all other CAN IDs.

3. Findings

Before attempting to reverse engineer the CAN bus signals on a physical vehicle, we modelled the procedure in a CAN bus simulator. The simulator selected is the open-source ICSim (Instrument Cluster Simulator) which was developed by Craig Smith of OpenGarges.org. When using the simulator, the CAN network used is a virtual network. SocketCAN for Linux includes native support for virtual CAN networks. Initially, a virtual CAN network in the OS is established, which is then available for use by the ICSim software. It is necessary to re-enable the virtual network following any restart of the OS. The commands were added to a setup file "setup_vcan.sh" so they could be run easily after each restart. Note that the ICSim software requires simple direct media layer development files to display the graphic output. Due to variations in these libraries between Linux OS versions, ICSim is available in source code only. The dependencies are installed and then each of the ICSim files needs to be compiled using the MAKE command before it can be executed on the OS.

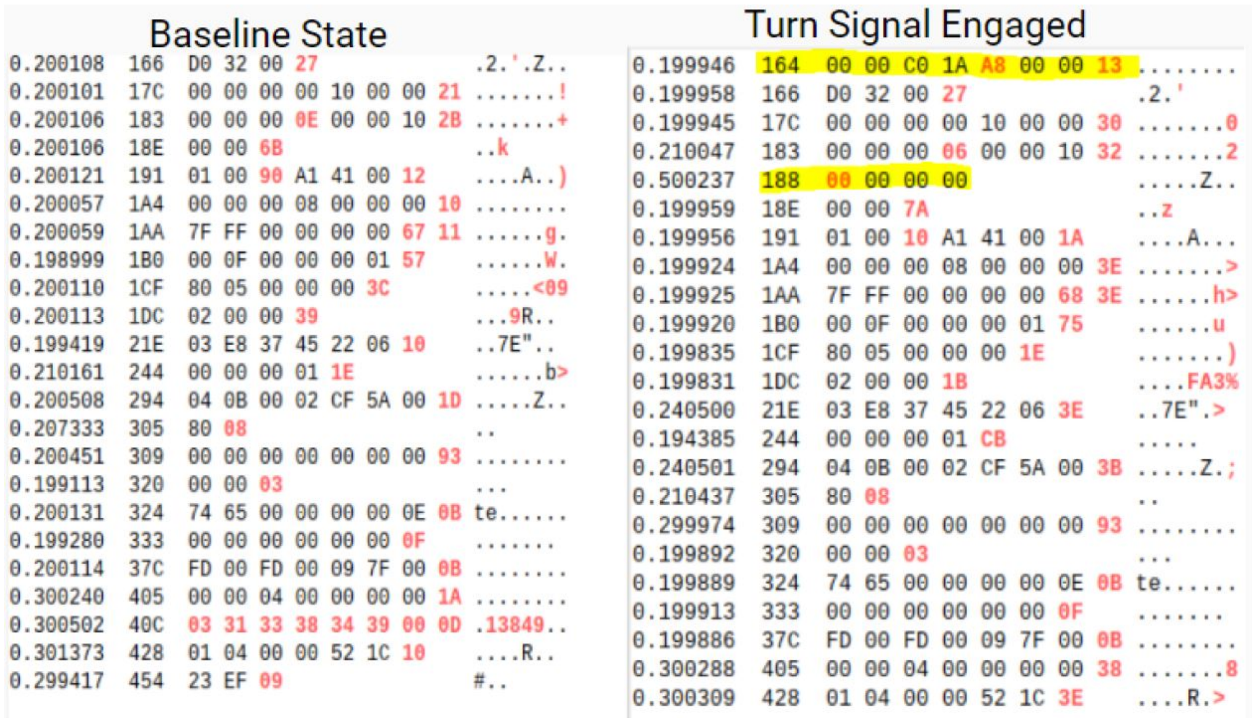
To establish communication with the CAN bus network by observing the traffic on the CAN bus, we utilized the cansniffer tool of CAN-Utils. Cansniffer has a major advantage over candump for this task because it will automatically filter out and remove CAN IDs that have static signals. This will leave a view of only CAN IDs which have recently changed. The output displayed by cansniffer left to right is as follows. First is the delta field: this field is a time delta that can be used to express the time each can frame was received in relation to the others on the bus. The next field to the right is the ID field. This is the CAN ID (arbitration ID) and is listed in the hexadecimal format. The next field consists of eight columns, each displaying a hexadecimal digit representing one nibble (4 bits) of the data payload. Directly to the right of the data payload columns is the data information represented in ASCII character output. The cansniffer tool allows an option "-c" at runtime which will add a color indication to any of the signals which have changed since the last time a frame was received from that CAN ID. This is shown below in Figure 9, where the hexadecimal and ASCII values colored in red have a changed value from the previous frame that was received from the same CAN ID.



```
54 delta   ID  data ...                 < cansniffer vcan0 # l=20 h=100 t=500 >
0.206935   39  00 39                    .9
0.200118   95  80 00 07 F4 00 00 00 26 .......&
0.200556  133  00 00 00 00 B6           .....
0.200512  136  00 02 00 00 00 00 00 39 .......9
0.199397  13A  00 00 00 00 00 00 00 37 .......7
0.199353  13F  00 00 00 05 00 00 00 3D .......=
0.199086  143  6B 6B 00 FF              kk..
0.200085  158  00 00 00 00 00 00 00 37 .......7
0.199581  161  00 00 05 50 01 08 00 2B ...P...+
```

Figure 9. CAN Sniffer Output with Highlighted Changes

To establish a baseline of traffic on the network, traffic is observed for a few seconds while taking no actions. Once the baseline is established, the second step is to continue observing the traffic while triggering an event (in this example, turn on the left turn signal). While the event is triggered, the CAN bus traffic is observed to see if any new CAN IDs appear or if there are corresponding changes to the signals of the existing CAN IDs that were not present before the event. Figure 10 shows the baseline compared to the new IDs (highlighted in yellow) observed when the turn signal was activated.

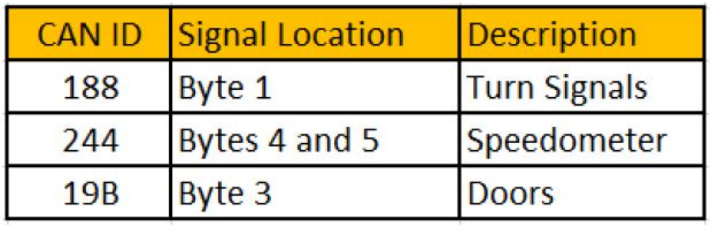


```
0.200108  166  D0 32 00 27              .2.'.Z..
0.200101  17C  00 00 00 00 10 00 00 21  .......!
0.200106  183  00 00 00 0E 00 00 10 2B  .......+
0.200106  18E  00 00 6B                 ..k
0.200121  191  01 00 90 A1 41 00 12     ....A..)
0.200057  1A4  00 00 00 08 00 00 00 10  ........
0.200059  1AA  7F FF 00 00 00 00 67 11  ......g.
0.198999  1B0  00 0F 00 00 00 01 57     ......W.
0.200110  1CF  80 05 00 00 00 3C        .....<09
0.200113  1DC  02 00 00 39              ...9R..
0.199419  21E  03 E8 37 45 22 06 10     ..7E"..
0.210161  244  00 00 00 01 1E           ......b>
0.200508  294  04 0B 00 02 CF 5A 00 1D  .....Z..
0.207333  305  80 08                    ..
0.200451  309  00 00 00 00 00 00 00 93  ........
0.199113  320  00 00 03                 ...
0.200131  324  74 65 00 00 00 00 0E 0B  te......
0.199280  333  00 00 00 00 00 00 0F     .......
0.200114  37C  FD 00 FD 00 09 7F 00 0B  ........
0.300240  405  00 00 04 00 00 00 00 1A  ........
0.300502  40C  03 31 33 38 34 39 00 0D  .13849..
0.301373  428  01 04 00 00 52 1C 10     ....R..
0.299417  454  23 EF 09                 #..
```

```
0.199946  164  00 00 C0 1A A8 00 00 13  ........
0.199958  166  D0 32 00 27              .2.'
0.199945  17C  00 00 00 00 10 00 00 30  .......0
0.210047  183  00 00 00 06 00 00 10 32  .......2
0.500237  188  00 00 00 00              .....Z..
0.199959  18E  00 00 7A                 ..z
0.199956  191  01 00 10 A1 41 00 1A     ....A...
0.199924  1A4  00 00 00 08 00 00 00 3E  .......>
0.199925  1AA  7F FF 00 00 00 00 68 3E  ......h>
0.199920  1B0  00 0F 00 00 00 01 75     ......u
0.199835  1CF  80 05 00 00 00 1E        .......)
0.199831  1DC  02 00 00 1B              ....FA3%
0.240500  21E  03 E8 37 45 22 06 3E     ..7E".>
0.194385  244  00 00 00 01 CB           .....
0.240501  294  04 0B 00 02 CF 5A 00 3B  .....Z.;
0.210437  305  80 08                    ..
0.299974  309  00 00 00 00 00 00 00 93  ........
0.199892  320  00 00 03                 ...
0.199889  324  74 65 00 00 00 00 0E 0B  te......
0.199913  333  00 00 00 00 00 00 0F     .......
0.199886  37C  FD 00 FD 00 09 7F 00 0B  ........
0.300288  405  00 00 04 00 00 00 00 38  .......8
0.300309  428  01 04 00 00 52 1C 3E     ....R.>
```

Figure 10. CAN Traffic Generated by an Event Trigger

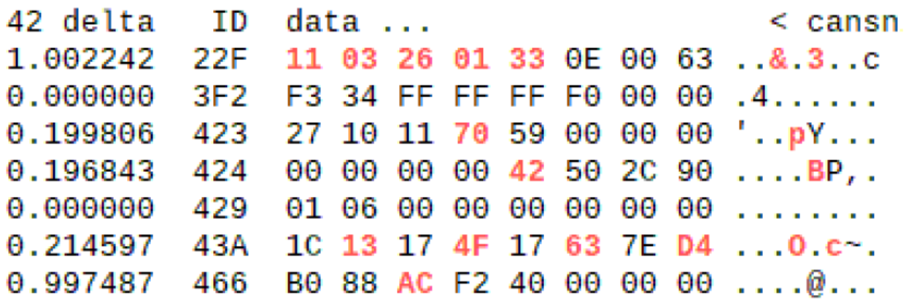
Cansniffer provides real-time filtering options that are used to further narrow down the output. First, all CAN IDs are removed from the output. Then, the IDs to be observed are added back to the output by entering +188 and +164. With the output filtered to only two IDs, ID 164 is not related to the event as it is continuously transmitting signals and was therefore a false positive identification. That CAN ID is removed, leaving only ID 188. Observing the changing data packet when the turn signal is engaged, it can be seen that ID 188 is directly related to the illumination of the turn signal indicators on the IC display. It is observed that the most significant byte (MSB) of the data payload controls this output. A data value of 00 turns the indicators off and a data value of 02 turns the right turn signal indicator on. It is further observed that when triggering the left turn signal, an event value of 01 is present in the MSB of the frame for CAN ID 188. It is therefore concluded that CAN ID 188 is responsible for the display of the turn signal indicators on the instrument cluster display. This process was repeated to identify the other CAN IDs available on the ICSim simulator software. The reverse-engineered IDs are presented in Figure 11, where the signal location refers to the bit position and length in the CAN frame responsible for the indicated event. These could constitute a spoofing or a relay attack.



CAN ID	Signal Location	Description
188	Byte 1	Turn Signals
244	Bytes 4 and 5	Speedometer
19B	Byte 3	Doors

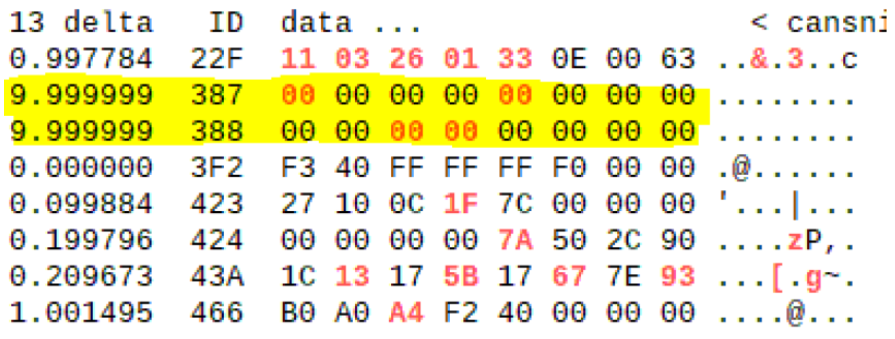
Figure 11. Reverse Engineering CAN IDs

Having exhausted the events possible on the simulator, the above process is now applied to the test vehicle. A baseline is established (see Figure 12). Events are then triggered by pressing the buttons on the control panel to turn on the fan blowers, turn on the AC, turn on the MAX AC, and adjust the air output through the dash and floor vents. Traffic is observed on the CAN network during these changes, and through observation, two CAN IDs appear in the traffic whenever changes are made. Those IDs are 387 and 388, as highlighted in Figure 13. The IDs of 387 and 388 with all-zero values are the signals for the event, both of which should be simultaneously transmitted as shown in Figures 12 and 13.



```
42 delta   ID   data ...                  < cansn
1.002242  22F  11 03 26 01 33 0E 00 63 ..&.3..c
0.000000  3F2  F3 34 FF FF FF F0 00 00 .4......
0.199806  423  27 10 11 70 59 00 00 00 '..pY...
0.196843  424  00 00 00 00 42 50 2C 90 ....BP,.
0.000000  429  01 06 00 00 00 00 00 00 ........
0.214597  43A  1C 13 17 4F 17 63 7E D4 ...O.c~.
0.997487  466  B0 88 AC F2 40 00 00 00 ....@...
```

Figure 12. Establishing Baseline CAN on the Test Vehicle



```
13 delta   ID   data ...                  < cansni
0.997784  22F  11 03 26 01 33 0E 00 63 ..&.3..c
9.999999  387  00 00 00 00 00 00 00 00 ........
9.999999  388  00 00 00 00 00 00 00 00 ........
0.000000  3F2  F3 40 FF FF FF F0 00 00 .@......
0.099884  423  27 10 0C 1F 7C 00 00 00 '...|...
0.199796  424  00 00 00 00 7A 50 2C 90 ....zP,.
0.209673  43A  1C 13 17 5B 17 67 7E 93 ...[.g~.
1.001495  466  B0 A0 A4 F2 40 00 00 00 ....@...
```

Figure 13. Event Trigger on the Test Vehicle

With this knowledge, we can proceed with generating the desired CAN frame, and transmitting it to the IVN is a trivial task. For this, we use the cansend tool from CAN-Utils. Cansend transmits a specified CAN frame to the bus. The syntax for the command is "cansend <device> <can_frame>." The canframe itself is specified by the canID followed by the symbol and finally the data payload in hexadecimal. For the system to turn on the blower, the following command is sent: "cansend can0 387#80 00 00 00 50." This sends a canframe with the arbitration ID of 387, a data payload MSB of 128 and 80 in the leading byte and the fifth byte, respectively. The same process is used to send the second required CAN frame to trigger the event. To incorporate these into the system, which was coded entirely in Python, the Python-Can library module was imported. This library provides an application programming interface for using all the CAN-Utils tools in Python. Due to the potential problems that may have been caused by introduced fabricated CAN packets on a physical vehicle, the completed system was tested while sending the CAN frames to a virtual CAN network rather than a physical vehicle network.

4. Summary & Conclusions

This research analyzed the common vulnerabilities of the CAN protocol, including passive and active attacks. For passive attacks, we eavesdropped on the ongoing traffic on a simulated and an actual CAN bus. For active attacks, we injected spoofed traffic on the CAN and observed the impact. Experimentation was performed using various software and hardware tools to conduct simulation, emulation, and real-time analysis. The findings outlined in Figures 11-13 suggest the necessity of a layered defense mechanism on all pillars of systems security, from authentication to confidentiality and integrity checks.

Bibliography

Basavaraj, Dheeraj, and Shahab Tayeb. "Towards a Lightweight Intrusion Detection Framework for In-Vehicle Networks." *Journal of Sensor and Actuator Networks* 11, no. 1 (2022): 6.

Bozdal, Mehmet, Mohammad Samie, Sohaib Aslam, and Ian Jennions. "Evaluation of CAN Bus security challenges." *Sensors* 20, no. 8 (2020): 2364.

Jacuinde-Alvarez, Daniel, James Dols, and Shahab Tayeb. "A Real-Time Intelligent Intra-vehicular Temperature Control Framework." In *Proceedings of SAI Intelligent Systems Conference*, pp. 593–612. Springer, Cham, 2021.

Miller, Charlie, and Chris Valasek. "Adventures in Automotive Networks and Control Units." *Def Con* 21, nos. 260–264 (2013): 15–31.

About the Author

Shahab Tayeb, PhD

Dr. Shahab Tayeb is a faculty member with the Department of Electrical and Computer Engineering in the Lyles College of Engineering at California State University, Fresno. Dr. Tayeb's research expertise and interests include network security and privacy, particularly in the context of the Internet of Vehicles. His research incorporates machine learning techniques and data analytics approaches to tackle the detection of zero-day attacks. Through funding from the Fresno State Transportation Institute, his research team has been working on the security of the network backbone for Connected and Autonomous Vehicles over the past two years. He has also been the recipient of several scholarships and national awards, including a US Congressional Commendation for STEM mentorship.

MTI MINETA TRANSPORTATION INSTITUTE