Reverse Engineering of an OEM specific CANBUS and creating an application to control the instrument cluster.

Billy Irvin
C00251069
Richard Butler
South East Technology University
17 April 2023

Table of Contents

Introduction

Cars now have lane detection, auto parking, self-braking, and even fully automated driving. With all these features and new features arising, it is a leap forward in the technological advancements in the automotive industry. However from my understanding, security is not at the forefront of the manufacturers' priorities. My project demonstrates the vulnerabilities that can be exploited, which shows the lack of security infrastructure in the network of a vehicle that possesses an OBDII port (from 1988 to present).

General Issues

Throughout the development of the application, many issues arose. Listed below are the issues documented in detail:

API

In the development of the application, the API “PCAN BASIC API” was a critical part as this was used for the programming language to communicate with the PCAN adapters. I had never used an API before, and when looking online through forums and the peak-systems website, there was very little documentation about how to implement the API. This was the most significant issue I had when developing the application. Overall, it took me over three months to understand what was required from me.

Python language

When I started the project, I had only had experience using Java and C++ however, upon completing my research, I learnt that Python would be the most suitable programming language to use and going from Java and C++ which were very similar in ways. Python however was different in many ways. Due to this, I learned Python, which had certain aspects I never worked with, like threading and Gui development which brought on issues further in the project.

CAN Bus Network

Starting this project, I knew very little about cars, and nothing about the network the cars ran off. I had to learn a lot of things such as the baud rate (the transmission speed of messages) and the bitrate which decides whether you were reading from a high or low transmission rate. Another problem was determining the priority of the messages (High or Low Transmission rate) and how the message was encapsulated and sent across the network.

Reverse Engineering

This year I stated a new module called Reverse Engineering and Malware Analysis. When looking through the notes to see if I could find anything to help me with the reverse engineering of the car, there was not much that I could relate to, so I had to carry out my own research and learn how I was going to reverse the message that was being sent across the CAN bus network.

PCAN Adapters

First starting out, I had one adapter which I had to swap from high to low, so when attempting to reverse engineer the dashboard. I had to do one speed at a time. This also affected how my application was going to be built, as I would either have to create two separate applications or one application but have to change the adapter back and forward depending on whether the message, I was sending was a high or low transmission rate.

Threading and queue overflow

Just before my presentation, I ran into problems with the threading I had implemented in the application, as I had two threads along with the main thread. The main thread was everything to do with the GUI, and the other two were one for the high-speed messages and the other for low. The application would turn on however, as soon as I interacted with a button to send a message, it would freeze the GUI. Then the adapter was outputting an error to inform me that the queue was full when trying to send messages. The results were that nothing would execute as required.

Accomplishments

All the above issues have been solved. I solved the threading and queue overflow by adding some sleep time in the code after each of the messages was sent. I also had to contact the company that made the adapters to enquire why the queue was being filled up. It turned out I needed to add a clear queue in the code after the messages were sent. To solve the adapter problem, I used two adapters, one for high and one for low. This benefited me significantly as I only needed to create one application to control the dashboard. It took a significant amount of time to solve how to implement the API, as it is not well documented. I solved all the other issues by researching further. The overall outcome was that I developed the application based on what I reverse-engineered, which could run smoothly and have ease of use. I will describe in more detail under the following headings user experience and user interface.

User experience

Functionality	Description
1. Engine Start	Removes the engine and oil light on the dashboard, simulating the startup.
2. Engine Reset	Resets the engine and oil light back on.
3. ABS	Removes the ABS light from the dashboard.
4. ABS Reset	Resets the ABS light back.
5. Radio 1	Brings up “” on the radio display.
6. Radio 2	Brings up “” on the radio display.
7. Radio Reset	Clears the radio display.
8. Sound 1	The dashboard makes a beep.
9. Sound 2	The dashboard makes a secondary beep.
10. Sound Reset	Resets the sound to none.
11. Warning 1	
12. Warning 2	
13. Warning 3	
14. Warning 4	
15. Waring Reset	Resets the current waring that is being displayed.
16. Submit	This button takes what was entered in the text field. Max is 12 characters, and the string is displayed on the radio display.
17. RPM	This is a slider that will adjust the RPM clock in the application and dashboard.
18. Speed	This is a slider that will adjust the speed clock in the application and dashboard.
19. Self-Reverse engineering	This section allows the user to send their own message into the CAN bus at high or low and if it is successful, they can save it.
20. Send Saved	Send the messages saved by the user.

Table 1: Functionality

User interface

The User interface is designed to be unique, interactive, fast, and easy to use. The application is broken down into two parts the dashboard and self-reverse engineering I will discuss in more detail how these differ.

Dashboard

The dashboard is designed for the least technical user, as this side has been reversed-engineered for a specific OEM provider, so the functions are pre-determined messages that the user selects, and the function is performed. This brings the ease-of-use aspect to the application. Each function has a specific button with either text or an image to identify the process that can be performed to make the application look and feel unique. The application is interactive, based on the action that the user performs, allowing it to display on both the user interface and the physical dashboard of the car simultaneously.

Self-Reverse Engineering

The self-Reverse engineering side of the application would be aimed at the technical user based on the functionality that can be executed i.e., the user can input their own message using hexadecimal values that can be sent to the CAN Bus network. The user has the option to send messages on a High or Low bitrate depending on the action the user is requesting. High is used for instances such as Speedometer messages, where Low bitrate is used for radio functionality. If the message has been executed on the interface successfully, the user has the option to name it and save this message for further use.

Reverse engineering

Reverse engineering was a big aspect to this project. How I started my reverse engineering was by using an application called “PCAN-view” which worked with the PCAN adapters to read CAN messages, it would gather all the messages going across the CAN network and take the CAN I.D placing it in a table and after the CAN I.D would be the CAN data but this would change all the time but the application would place this in the table too but would break it down into each data byte therefore allowing me to monitor the changes within the bytes.

With the PCAN-view I first start out by taking the CAN I.Ds that were captured and sending them back into the network using PCAN-view then monitoring if anything happened on the dashboard to identify what each CAN id controlled using this method I gained a list of CAN I.Ds but is still couldn’t match most of the ids to a component of the dashboard. I then went into the CAN data viewing each byte and changing them around I started by making all the bytes zero then continuing to go up through the hex table byte by byte this was a long process but the best outcome as from this I reversed most of the components from the list. But I reversed one component by taking the CAN data of the message and converting it back to ascii which brought back the word “Hello”, This was the message the radio displayed when the power was turned on for the dashboard.

Below you will find a table of the components that was reverse engineered:

Components
Speedometer
RPM
Radio Message
Radio display
Warning Messages
Engine and oil
ABS
Sounds

Table 2: Components

Starting Again

If I was to start this project again, I would like to have more time due to how much I had to learn to start the project. With this extra time, I would have been able to add some more features to the application i.e., record the cars action and play them back and advanced GUI using the QT framework.

Conclusion

In my conclusion I will be discussing three main ideas that have to be consider for securing the CAN Bus network, what OEM should be looking at and what security has been put in place so far.

Ideas

In the following papers “Evaluation of CAN bus security challenges” by (Bozdal et al., 2020), “Secure communication over CANBus" by (A. S. Siddiqui, Y. Gui, J. Plusquellic and F. Saqib,2017) and "A Secure Communication Method for CANBus," by (C. H. Park, Y. Kim and J. -Y. Jo, 2021) it mentions the following ideas: Network segmentation, Encryption, Authentication under the heading Counter measures for CAN attacks. “The attacks on CAN clearly show that the protocol is very vulnerable and requires cyber defence mechanisms for safe driving. The studies to solve this problem have mainly focused on four defence mechanisms: network segmentation, encryption, authentication” (Bozdal et al., 2020)

Network segmentation

Network segmentation is taking the CAN network and breaking it down into sub-networks. This would be beneficial because if one network got comprised it would not affect the others. The networks are interconnected by the ECU this is common in most cars today. But according to (Bozdal et al., 2020) “The method is simple to implement, but it is not effective if the gateway ECU is compromised”. This statement shows that even if you separate the network there are limits to how secure the cars can be, meaning the cars we drive today are not fully secure. The paper also mentions that “It also makes the maintenance of the system more difficult, along with the increased cost.” (Bozdal et al., 2020) with the increase in cost to implement this feature this could be a reason why lots of OEMs do not include the feature in their cars.

Encryption

Encryption would be the first thing that comes to many people’s minds when thinking of securing something, but for the CAN network it is not as simple as that. The CAN network is a critical part of the car. If you wanted to encrypt the messages you would use more of a dynamic key encryption as its less likely to be cracked over static key encryption but using this method causes latency issues according to (Bozdal et al., 2020) “The dynamic key can also cause latency on resource-constrained ECUs, and it is not acceptable for safety-critical real-time systems.” In a car this could have significant consequences, if you hit the brakes the ECUs would have to encrypt that messages before you send it and decrypt it when receiving it. This process would to be slow and could cause an accident. Implementing any sort of encryption needs lots of processing power which the current ECUs don't have. This is also mentioned by (Bozdal et al., 2020) "Another issue is the limited computational power of ECUs". In the following paper "Secure communication over CANBus" by (A. S. Siddiqui, Y. Gui, J. Plusquellic and F. Saqib,2017) it covers the impletion of using key pairs for encryption and decryption but using this method the ECU will need a great amount of power. Its also mention in both the above and following paper " A secure protocol for session keys establishment between ECUs in the CAN bus" by (S. Fassak, Y. El Hajjaji El Idrissi, N. Zahid and M. Jedra, 2017) the use of Elliptic Curve Cryptography with the key pairs.

Authentication

When it comes to authentication of the CAN bus, we can't identify what component of the vehicle sent the CAN messages its also mentioned in "A Secure Communication Method for CANBus," by (C. H. Park, Y. Kim and J. -Y. Jo, 2021) "Since CANbus protocol does not support source address authentication, every ECU node trusts every message". For example, if the network was accessed and messages where sent every node would accept the messages, as there is no way to verify the message that was sent. To solve this issue, you would need to implement non-repudiation which is the proof that the message sent cannot be denied from the sender and the receiver. With non-repudiation this will result in time delays as the messages will have to be verified before being accepted. An example of this would be the brakes could take a second before coming active. An implementation of non-repudiation with low latency time delay would solve a lot of problems within the security of the CAN bus network.

References

1. Bozdal, M. *et al.* (2020) *Evaluation of CAN bus security challenges*, *MDPI*. Multidisciplinary Digital Publishing Institute. Available at: https://www.mdpi.com/1424-8220/20/8/2364 (Accessed: April 7, 2023).

2. A. S. Siddiqui, Y. Gui, J. Plusquellic and F. Saqib, "Secure communication over CANBus," *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boston, MA, USA, 2017, pp. 1264-1267, doi: 10.1109/MWSCAS.2017.8053160. Available at: https://ieeexplore.ieee.org/document/8053160/ (Accessed: April 17, 2023).

3. C. H. Park, Y. Kim and J. -Y. Jo, "A Secure Communication Method for CANBus," 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), NV, USA, 2021, pp. 0773-0778, doi: 10.1109/CCWC51732.2021.9376166. Available at: https://ieeexplore.ieee.org/document/9376166 (Accessed: April 17, 2023).

4. S. Fassak, Y. El Hajjaji El Idrissi, N. Zahid and M. Jedra, "A secure protocol for session keys establishment between ECUs in the CAN bus," 2017 International Conference on Wireless Networks and Mobile Communications (WINCOM), Rabat, Morocco, 2017, pp. 1-6, doi: 10.1109/WINCOM.2017.8238149. Available at: https://ieeexplore.ieee.org/abstract/document/8238149 (Accessed: April 17, 2023).