

# PROJECT REPORT

Gary Morrissey C00259786

## Table of Contents

Introduction.....	2
Background .....	2
CAN Bus .....	2
PEAK and PCAN System.....	3
Brute Force .....	3
Reverse Engineering .....	4
Implementation .....	4
Coding.....	4
Language.....	4
Interface.....	4
Hardware.....	5
Features .....	6
Testing and Results .....	10
Discussion.....	11
Original Concept .....	11
Problems Encountered .....	12
What I learned.....	13
Coding.....	13
Research.....	14
Strengths and Weakness .....	14
Strengths.....	14
Weaknesses .....	14
Overall.....	15
Future Improvements .....	15
Conclusion .....	16
Security Implications.....	16
Overall Conclusion .....	16
Plagiarism Declaration .....	17
References .....	17

# Introduction

The purpose behind this project was to reverse engineer a cars dashboard with the aid of a brute force attack through the cars CAN Bus. The CAN Bus is the Control Area Network of the car and is the transport layer for all communication throughout the vehicle, which I will go into detail in the '[Background](#)' section of this project. My application will initialize a connection to the CAN Bus and then allow the user to force a brute attack as well as send their own individual messages based off their findings from the brute force attack. The purpose behind the individual messages is for the user to narrow down the message that makes an affect to the dashboard. Once the user is happy a change is made, they can then save it to a file with the message ID, the message data and then they will be prompted to include a comment as to what the change was. The user then repeats this process until they are happy with the messages they have collected. In the end they will be left with a file containing each message saved along with their comments so they can refer back to it and prove they reverse engineered this section of the CAN Bus.

This application revolves around the connection to the vehicles CAN Bus. When the application launches, it asks the user to initialise the baud rate speed at which they wish to send their CAN Messages (Baud rate Speed explained here). Once the user enters either 125 or 500, then the application will automatically try to initialize a connection to the CAN Bus through the computers USB port. If this initialisation fails, then the application will show an error code and fail to launch. This ensures that if the user launches the application and isn't connected, they won't be pushing a brute force attack with no purpose.

The application I produced centralises the process for users to reverse engineer the CAN Bus, so while it does have a specific purpose behind it, it still creates an easy-to-use interface for the user to push the brute force attacks and easily alter any messages they wish to check before sending and saving in their file of successful messages.

## Background

### CAN Bus

As mentioned earlier, the CAN Bus is the Control Area Network of the car. It operates as the communication channel throughout vehicle. The CAN Bus has a number of ECU's connected all along it. These ECUs are Electronic Control Units that interact with a specific aspect of the vehicle. Meaning one ECU may be connected to the cars lights system, while another is connected to the power steering control etc.

The CAN Bus is made up of 2 different channels. A 125kbps Bus and a 500kbps Bus. The 125kbps is a slower speed bus and may be used for messages relating to radio adjustment or the air conditioning, while the 500kbps bus is used for faster communication such as the power steering control or operating the brakes system.

Each bus also contains a CAN high and CAN low wire. These operate how the messages are carried throughout the Bus, the High carrying the signal at a higher pulse rate, while the lower handles the communication at a lower pulse rate.

The messages that go through this bus are named CAN Messages. These messages contain the message ID, which starts from 000 in hex and goes all the way to FFF. They also contain the message DATA which in a standard PCAN system is 8 bits on length and each bit is a hexadecimal value varying from 00 at the way to FF. These messages are then sent out in a broadcast format throughout the CAN Bus. Each ECU then reads the ID and determines whether the message is intended for that ECU or not. If so, it allows the message to be processed and ran, if not the ECU ignores the request.

## PEAK and PCAN System

I have integrated the use of the PEAK systems functions as well as the use of their physical cables in the connection from my computer to the CAN Bus. The PCAN basic software available online [here](#), provides the appropriate configuration files to allow a user to enable their own application to communicate with the can bus through their cables interface.

The PCAN basic downloads provides the user with the relevant header files and dynamic link libraries (DLL). These files provide the functions needed to establish a connection to the CAN system and to send messages through the CAN Bus.

I have these files included in my application code and linked to the project dynamically to ensure a successful connection.

## Brute Force

Brute force attacks are a type of attack that uses a trial-and-error basis to access a system or retrieve data from a system. The brute force attack tries every possible combination of a payload until correct combination is found. This type of attack can be very time consuming, especially if you need to observe the affect each combination has on a system. This can be implemented using timer delays in the attacks execution so a human can perceive the affect that specific combination had.

In my case, I implemented this attack by iterating through each message ID from 000 to FFF and setting each bit to FF, sending the message, then setting each bit to 00 and sending the message for each ID. This allows me to see what ID affects the dashboard section of the CAN Bus I am testing my application on. These messages will contain a delay in between each sending function to allow myself or another user to see the affect on the dashboard.

With the information gathered from this, I can begin the reverse engineering process by altering the message bits manually with the ID's I found from the Brute Force attack.

## Reverse Engineering

Reverse Engineering is the process breaking down a system or software in order to understand how its operates and what its functionality is. This can be used to understand how a product or system works, to find any vulnerabilities or to replicate it. This process can be applied to many different aspects such as software or hardware.

For this project, the reverse engineering (RE) aspect is used to create a list of successful messages that affected the dashboard based off the findings from the brute force attack mentioned before. This RE process is manual in this case after the brute force attack. With the ID's collected from the attack, I then use the interface to manually insert my own message into the message data, then send and save said messages to a file that appends each message into it. After repeating the process until the user is happy, they found all relevant messages, the user is then produced with the file containing all successful messages, completing the reverse engineering process.

## Implementation

### Coding

#### Language

In the development of my application, I chose to use C++ as my coding language. I chose this as I am familiar it due having a module on it in my 3<sup>rd</sup> year of college. I also have plenty of study notes on it that I could refer to if stuck. C++ is a widely used language and is renowned for its performance and execution speed. There are many online tutorials and guides that helped me in the development of my code throughout the process.

C++ also allows a user to tailor their code to interact with other software, such as the pcan basic software I interact with in my application. The language makes interacts with the pcan basic software through the PCANBasic.h file. Once this is included in the C++ file of the application, it can then access the functions and implement the variables required for the connection to the CAN System.

#### Interface

To develop my interface and GUI, I decided to use QT Creator and Designer. This is my first time creating a GUI interface for a project, so after some research into what the best fit for me would be, I decided upon using QT. I chose this as there are many online tutorials available on how to create a GUI and make it interact with your applications functions. QT has many libraries available which proved instrumental in the creation of my GUI that seamlessly interacts with my application functions.

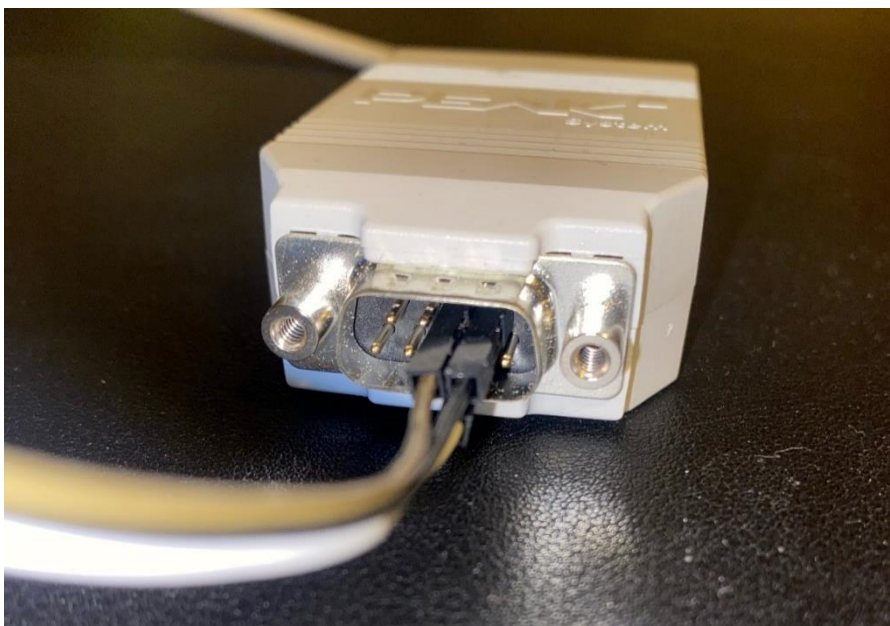
QT's features that I availed of are the use of signals and slots. These features allowed me to connect my GUI's line edit slots and push buttons to interact with the PCAN set functions from the PCAN libraries and my own personal functions to brute force attack the CAN bus, and to send, save and set messages.

## Hardware

This application initialises the CAN connection through the use of the PCAN-USB cable which is available from the PEAK System company. The cable allows a simple USB connection from the computer, which I then connect the specific pins on the other end of the cable to the CAN high and CAN Low pins of the desired CAN Bus speed I want to test. Below pictured is said cable.



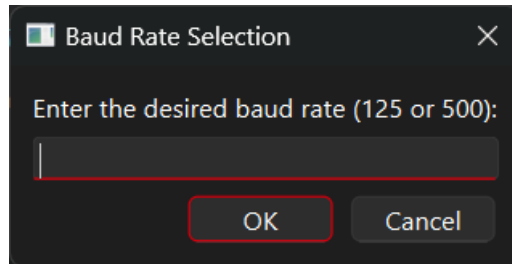
Also pictured below are the specific pins used to connect to the CAN Bus specific High and Low pins.



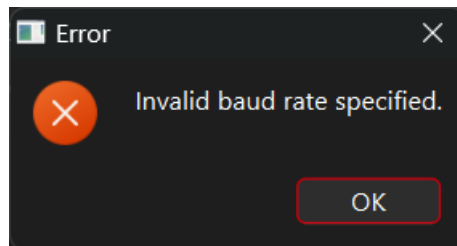
## Features

My application has many useful features to aid the user in their reverse engineering process.

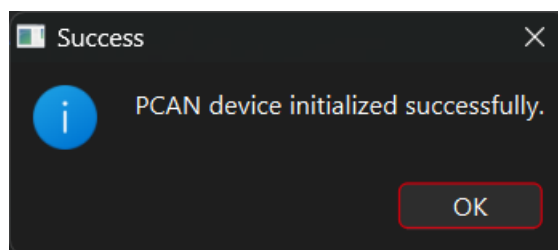
The first feature is its versatility in being able to connect to the 125kbps and 500kbps buses separately. When the application is opened, the user is prompted with an input box that asks the user to declare the bus they wish to connect to.



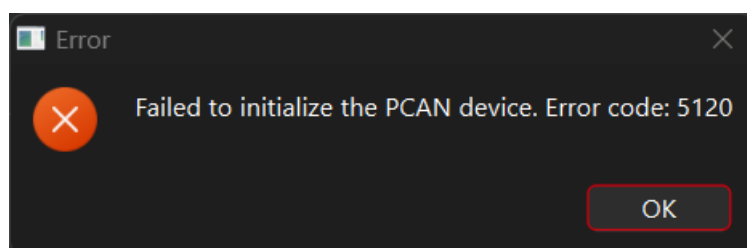
The user types in either 125 or 500 depending on the one they want to connect to. The input box is set to only accept these two inputs, so any other input will be invalid and the application shuts down.



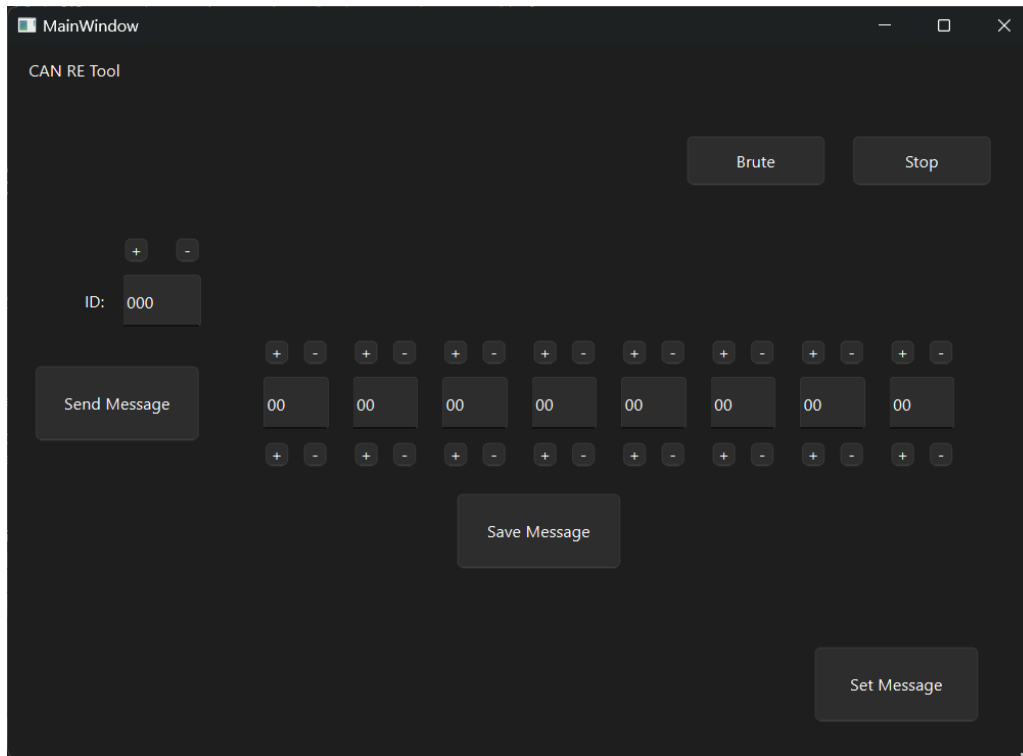
If the input is valid and the user has their device connected via the PCAN-USB Cable, they will be greeted with a dialog box saying the connection was successful.



If the cable is not connected with a valid input, the user is then greeted with a dialog box letting them know the CAN connection failed.



Once the connection is successful, the application will load the GUI. The GUI contains all the buttons used to set, save and message the PCAN messages that are set in the input boxes as seen below.



These input boxes correspond to the bits in the CAN message data being sent. The boxes are set to only accept input of a hex value with a maximum length of two characters, so they only accept from the range of 00 to FF. This is similar to the ID input box in the top left. This box also only accepts a hex value input but with a maximum length of three characters, so it only accepts values from 000 up to FFF.

Over the top of the eight input boxes to set the CAN Message there are a plus and minus button. The plus button sets its corresponding input box to FF and the minus button sets its corresponding input box to 00. This is a time saving tool so the user can just click a button to set the data to FF rather than click into it and typing it out each time for testing purposes.

Similarly, under the eight input boxes there is another set of plus and minus buttons. The intention behind these buttons was to increment and decrement the value in its corresponding slot by 1 respectively. Unfortunately, I encountered an issue with this function as it increments and decrements the value by 2 instead of one. This can still be useful for testing purposes as if the user wants to check each possible value the bit, once they reach FF, they can manually type in 01 and then use the increment button to repeat the process for the odd number combinations. So Initially they will check 00,02,04 etc to FF, then the second time they will check 01,03,05 etc up to FE.



Next, we have the Brute button and the stop button. The brute button implements a brute force attack on the CAN Bus. This button initially sets the ID to 000, then sets each message bit to FF and sends the message. It then sets the message data to 00 for each bit and sends the message. This is the first iteration of the loop. It then increments the ID by one and repeats the process. This means the application will send a message set to 'FF FF FF FF FF FF FF FF' and another set to '00 00 00 00 00 00 00 00' for each possible ID combination from 000 all the way to FFF. After each message is sent, there is a timer delay of two seconds to allow for a change to occur if the message is successful. The user of the application will have to physically keep an eye on their test subject. So, in my case I had to look at the dashboard apparatus seen [here](#), while the brute force attack was being sent. This two second delay also allows the user to notice the change to the test subject and then click the stop button. While the brute force attack is in motion, the application will write to a file the messages that have been sent. This file is named BruteMessages.txt and saves to the users' desktop for ease of access. Below is the file and the layout for how the messages are saved.

```
File Edit View
ID: 000 Message: 00 00 00 00 00 00 00 00
ID: 000 Message: ff ff ff ff ff ff ff ff
ID: 001 Message: 00 00 00 00 00 00 00 00
ID: 001 Message: ff ff ff ff ff ff ff ff
ID: 002 Message: 00 00 00 00 00 00 00 00
ID: 002 Message: ff ff ff ff ff ff ff ff
ID: 003 Message: 00 00 00 00 00 00 00 00
ID: 003 Message: ff ff ff ff ff ff ff ff
ID: 004 Message: 00 00 00 00 00 00 00 00
ID: 004 Message: ff ff ff ff ff ff ff ff
ID: 005 Message: 00 00 00 00 00 00 00 00
ID: 005 Message: ff ff ff ff ff ff ff ff
ID: 006 Message: 00 00 00 00 00 00 00 00
ID: 006 Message: ff ff ff ff ff ff ff ff
ID: 007 Message: 00 00 00 00 00 00 00 00
ID: 007 Message: ff ff ff ff ff ff ff ff
ID: 008 Message: 00 00 00 00 00 00 00 00
ID: 008 Message: ff ff ff ff ff ff ff ff
ID: 009 Message: 00 00 00 00 00 00 00 00
ID: 009 Message: ff ff ff ff ff ff ff ff
ID: 00a Message: 00 00 00 00 00 00 00 00
ID: 00a Message: ff ff ff ff ff ff ff ff
ID: 00b Message: 00 00 00 00 00 00 00 00
ID: 00b Message: ff ff ff ff ff ff ff ff
Ln 1, Col 1 | 344,064 characters | 100% | Windows (CRLF) | UTF-8
```

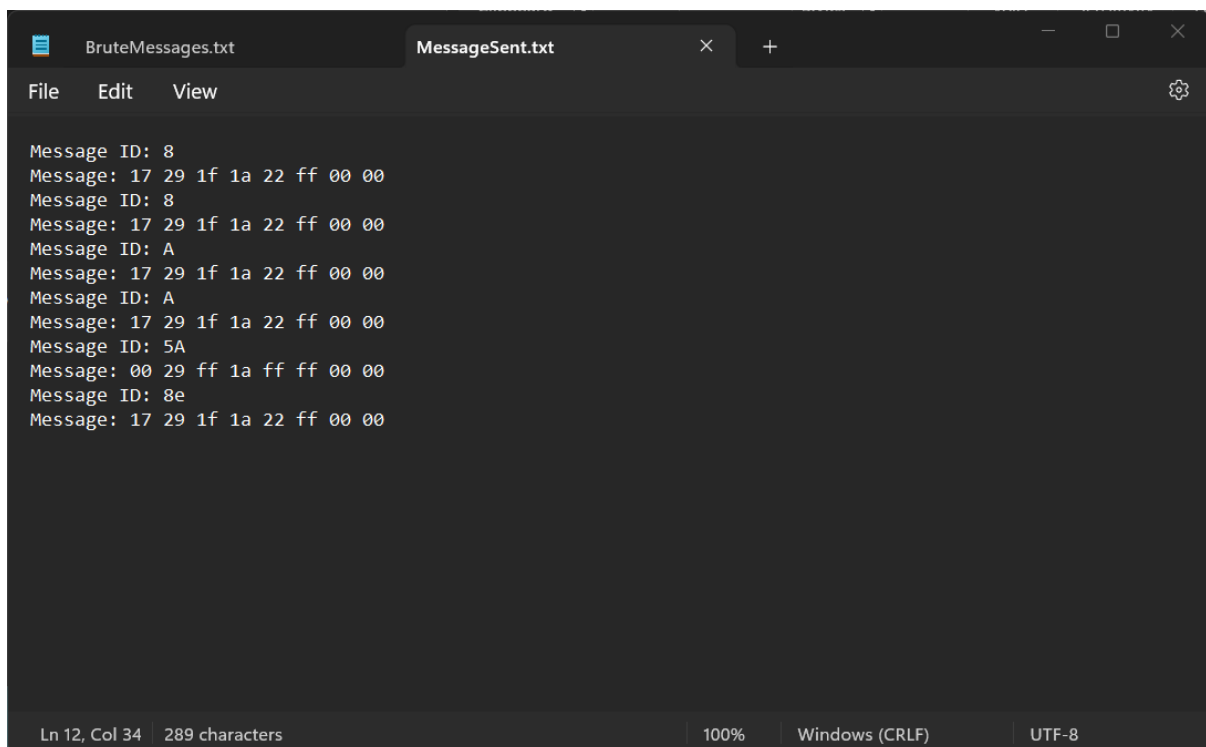
If the user notices a change has occurred during the brute force process, they can click the stop button which will stop the attack. The user can then go to the BruteMessages file and check which was the last message sent. Let's say the last message was ID 201 and message 'FF FF FF FF FF FF FF FF', this then lets them know that this ID correlates to the change they noticed which they can then take note of so they can manually check specific messages later in the RE process.

An important detail to note, the brute process can only execute once. So if the user wishes to push a second brute force attack, then they will have to relaunch the application.

Lastly, we have the send message, set message and save message buttons. Each of these messages are similar in their use of the data set in the ID input box and the eight data input boxes. But they differ in their execution of the messages.

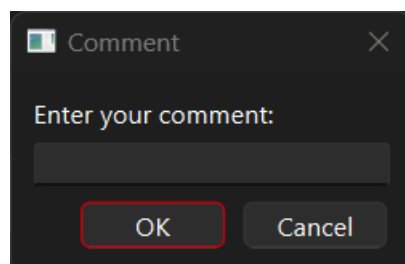
The Send Message button is used to send the CAN Message to the CAN Bus. It takes in the data from the ID and Data boxes, constructs the message format and then sends it. This button is used for the manual checking process to check if the message set affects the test subject.

The Set Message button is used to take in the data from the ID and Data boxes and append it to a file named MessageSent.txt. The purpose of this is to create a file of any message sent so the user doesn't repeat any messages that they have already checked. This file will look like the image below, showing the ID and the message data.

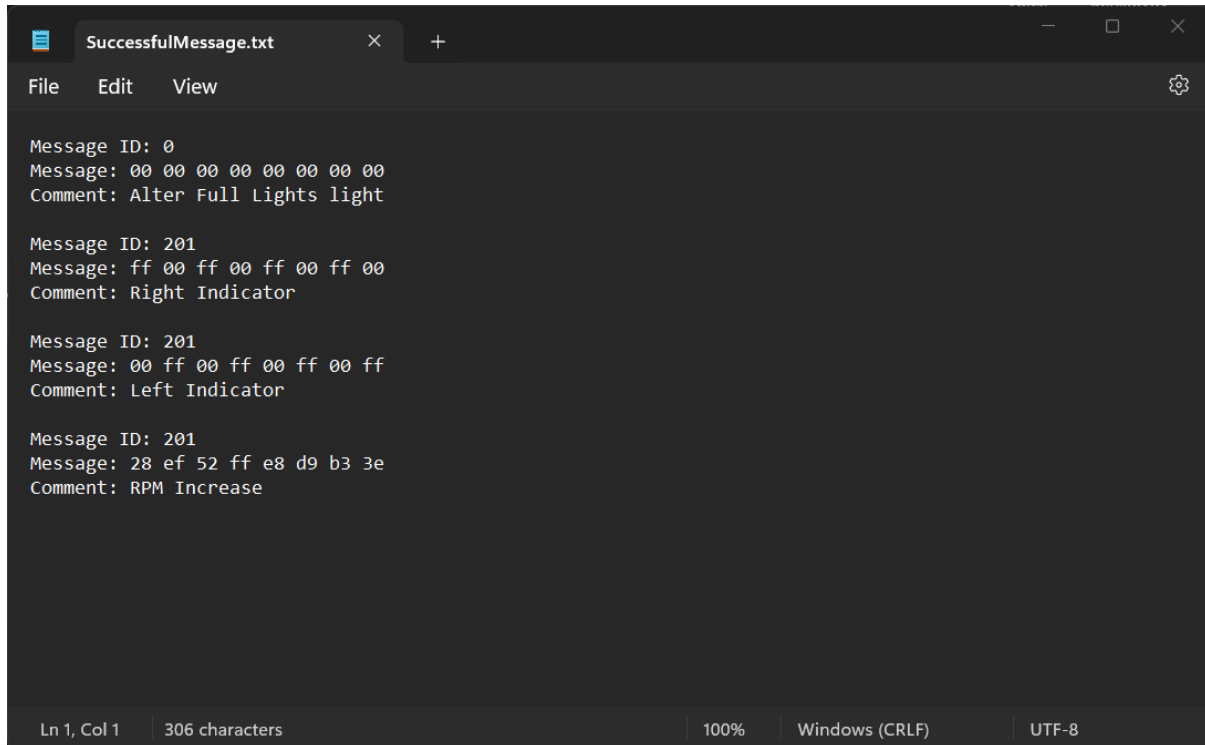


```
BruteMessages.txt | MessageSent.txt
File Edit View
Message ID: 8
Message: 17 29 1f 1a 22 ff 00 00
Message ID: 8
Message: 17 29 1f 1a 22 ff 00 00
Message ID: A
Message: 17 29 1f 1a 22 ff 00 00
Message ID: A
Message: 17 29 1f 1a 22 ff 00 00
Message ID: 5A
Message: 00 29 ff 1a ff ff 00 00
Message ID: 8e
Message: 17 29 1f 1a 22 ff 00 00
Ln 12, Col 34 | 289 characters | 100% | Windows (CRLF) | UTF-8
```

Finally, the Save Message button functions as it says. It takes in the data from the ID and Data boxes. If the user is happy with a message and found it to affect the test subject, they can use this to save the message to a file named SuccessfulMessages.txt. When the user saves the message, they are prompted with a comment box to take note of the change the message had on the test subject. Below is the comment box the user is prompted with.



Below we have the SuccessfulMessage file that stores the messages the user deems to have an affect on the test subject along with the comment saying what the affect was. This file is constantly appended during the RE process and once the user is satisfied that they have collected enough messages to affect their test subject, they can then open this file with a full list of messages completing the RE process.



```
File Edit View
Message ID: 0
Message: 00 00 00 00 00 00 00 00
Comment: Alter Full Lights light

Message ID: 201
Message: ff 00 ff 00 ff 00 ff 00
Comment: Right Indicator

Message ID: 201
Message: 00 ff 00 ff 00 ff 00 ff
Comment: Left Indicator

Message ID: 201
Message: 28 ef 52 ff e8 d9 b3 3e
Comment: RPM Increase

Ln 1, Col 1 | 306 characters | 100% | Windows (CRLF) | UTF-8
```

## Testing and Results

For the purpose of this project, my test subject was a dashboard clocks and screen from a Mazda 6. Pictured below is the physical test subject. To power this, I used a power control unit connected to the ground and +12v pins along the top of the subject. On the right-hand side, we have 2 sets of 2 pins. On the top we have the CAN High and CAN Low pins for the 500kbps bus. Then on the bottom we have the CAN High and CAN Low pins for the 125kbps bus. From the PCAN-USB cable mentioned [here](#), with the pins seen below we can connect directly into the cable using a combination of male and female jumper wires. For testing purposes of this project, I first launched the application and connected the wires to the 125kbps bus which showed me it affected the clocks on the dashboard. Once I was satisfied with the RE process, I closed the application and relaunched it connecting the wires to the 500kbps and repeated the process again. This showed me that this bus affected the digital displays.



Upon testing my application on this test subject, I found that the pins are very delicate and fragile. This led to quite a few setbacks as when disconnecting the cables, they had the tendency to snap or bend. This resulted in having to reconstruct the test subject several times.

During the testing process, I also found that when a message was sent, it may take a couple seconds for the affect to be visible on the test subject. This led to me implementing the timing delay on the brute force attack in order to see the changes and the record them efficiently. The delay also allowed me appropriate time to click the stop button once I noticed the change.

Through my own testing of the subject, I found that the messages:

ID:433 with DATA FF FF FF FF FF FF FF FF affected the brightness of the on-board screens

ID:401 with DATA 00 04 04 04 04 04 04 FF caused the dashboard to beep.

These finding helped me in the development of the application and its functions.

## Discussion

### Original Concept

Originally, my intention was to make the application have a camera implementation where the camera notices a change in the dashboard during the brute force attack and takes a picture. This picture would then be saved to an SQL database in a table named 'Potential Positive' along with the message that was sent before the picture was taken.

The idea was then to pull this table from the database and manually check the messages. If the user was happy the potential message did make an affect to the test subject, then they could store it in another table named 'Successful Messages'. In the end the user could then pull this table and be produced with a file of all successful messages along with a comment on what affect they had, completing the Reverse Engineering process.

## Problems Encountered

In relation to the camera implementation, the idea was to put the test subject in a dark room and if a light appeared on the test subject for the camera to take a picture and save it with the message sent at the same time. This proved difficult as this concept meant any changes to the clocks like increasing the rpm clock or speedometer would not be noticed or saved.

The coding aspect of this also proved difficult. While I am familiar with C++, I still had to learn a lot of its functionality and how to implement them to work with my GUI. I also had to learn how to use QT creator for the GUI which left less time to work on the camera implementation.

Due to these reasons, I opted out of this idea and chose to have a manual checking process instead during the brute force attack.

In relation the SQL database, the main reason I wanted to have this was to work alongside the camera function to store the images. So, when I dropped this idea, I also decided to switch this database to using a simple text file for outputting the successful messages.

Another problem I encountered was the implementation of the functions on some of the buttons on the main window. As mentioned before, I had a problem with the plus and minus buttons underneath the message data slots. This problem seems to be related to how I convert the input box values to hexadecimal in order to increment or decrement. The input box calls a function named `onTextChanged()` and validates the input field to a hex value. I believe that the problem lies in the plus buttons function as it calls to the `onTextChanged()` function again which may cause the conversion of the value to increment by two instead of 1. I discuss this [here](#) with what I would do in a future implementation.

Another problem I found was an issue with the set, save and send message buttons. They function properly as stated before, but they repeat their function a second time. So, when I set a message, the application sets the message twice. When I click the send message, it sends the message twice and then when I click the save message button, it saves the message twice to the successful message file.

The final problem I will discuss is with the brute force attack process. As discussed earlier, this process can only be executed once when the application is launched. After debugging the code for this function, I found it was due to an issue with the Brute Force attack thread. This is a separate thread to the mainwindow thread as if they were all on the one thread, the user would not be able to click the stop button. This is because the brute process would pause the GUI and not allow the user to interact with it until its process has finished. This problem is something I would develop further on in the future.

While these problems I encountered were not intended to happen, the overall functionality of the application works. The user can still successfully reverse engineer a CAN Bus, it just may take a little longer than my original concept intended on taking.

In this final iteration of my project, I found an error in the send message function. The function successfully sends the message initially set, then it loses connection to the CAN Bus. This means in order to test several messages I have to relaunch the application. This is not ideal in the RE process and is an aspect I wish to fix given more time.

A bug I have been receiving near the end of the production of this application is that the file brute force process breaks down after the 10th iteration of the loop. But then the next time launched would iterate fully through. This is a problem with the memory of the application and is another aspect I would fix given more time.

## What I learned

### Coding

I learned a lot during the development of this project, in progressing my coding ability, how to interact applications with companies' software files provided online and how to create a graphical user interface to interact with my code functionality.

While I did study C++ in my 3<sup>rd</sup> year of college, it had been almost 9 months since I had used the language. This meant that I had to read my notes from college as well as completing my own study of online tutorial and forums on how to develop my application in C++. I found that w3 schools, stackoverflow and codecademy had sufficient information on the language to help me.

Upon my research at the start of the project, I had to decide on how to develop my GUI to interact with my code's functionality. This research led me to find several different ways to create this application such as the use of a framework. This showed different frameworks like QT, GTK+ and wxWidgets.

I decided to use QT as I found it easy to integrate my codes functionality with and it has the capability to create projects for windows, macOS and Linux as well as android and IOS. For this project I just focused on the windows implementation, but it does mean I could develop it further in the future if I wanted to.

QT also has an abundance of information online on how to create your application and how to link the GUI to your code. For example, the QT framework allows you to drag and drop different widgets for your interface. Once I was happy with how the interface looked, I could create the widget slots that contained my functions to run when that widget was interacted with. So, if I clicked by 'Send Message' button, in that slot, I implemented my send message function.

Another aspect of QT I learned was the proper way to connect your widgets to your source files in order detect the widgets efficiently. Connecting these widgets manually ensures the QT Framework doesn't have to generate the connections automatically, which can leave room for error and some widgets not connecting because of it.

In conclusion, the development of this application not only enhanced my coding capability, It also deepened my understanding of integrating applications with external software files and creating an efficient GUI. Despite the gap since my last experience with C++, I believe I have

furthered my knowledge on language and will hopefully develop this knowledge further along with the development of my QT skills.

## Research

The actual research into the CAN Bus of the vehicle was a big learning experience for me. Before I started the project, I had no understanding of how a vehicles communication system worked.

Through the project research, I got to learn all about the fundamental principles of the CAN Bus, including its message format and the different signal speeds they are sent on.

To send my CAN Messages and push the Brute Force Attack, I had to familiarise myself with the message format, which consisted of the message ID, the data stored in the message and the specific speed bus it is sent on. From this I could see that the message ID and data were all represented in hexadecimal values, which led to me creating the relative functions to only accept hex value inputs in my input boxes and how to create the message variable to be sent.

Finally, I learned all about the functions available through the PCAN-Basic files available from the PEAK System website. Specifically, the functions to initialise the CAN Bus connection, the relevant variables to set the message and the function to send the message to the CAN Bus. The PCANBasic.h file contained all these functions for me to integrate them with my application and once this file was included in my header and source files, I was able to pull the relevant variables and functions.

Overall, my research of the CAN Bus protocol has been a valuable learning experience. Allowing me to gain the theoretical knowledge and practical skills needed in the design and implementation of this project's application.

## Strengths and Weakness

### Strengths

One of the strengths of my project is its understanding of the CAN Bus protocol and Brute Force attack and its use of the PCAN Basic software functions in communicating with the CAN Bus.

The functional GUI I developed through QT is also a strength in my project as it creates a user-friendly interface which allows the user to avail of the applications functionality efficiently.

Another strength in my application is its versatility in being able to connect to both 125kbps and 500kbps buses which is prompted by the users input upon launching the application.

The last strength with my project is its use of the Brute Force attack on the CAN Bus to reveal the ID that correlates to the test subject. This along with the documentation of the separate files allows the user to reverse engineer the CAN Bus.

### Weaknesses

There are a couple of weakness of this project that I wrote about previously in the document.

The first being the fragility of the test subject's hardware. My test subject's pins and wires, while served their purpose for testing, were prone to breaking and bending, resulting in having to replace them. This put a pause on the testing process of this project until I could get the necessary parts to fix the test subject.

Another weakness is the functionality behind the plus and minus buttons to increment and decrement the input box values. While these buttons increasing or decreasing by 2 instead of 1 is a slight problem with the manual checking process, it can be avoided by iterating from 00 to FF incrementing by 2 each time, then starting again at 01 and incrementing by 2 up to FE. This means the user won't miss any possible combination in that message bit.

Finally, the last weakness to discuss is the issue with thread management. The limitations of the brute force attack to a single execution upon the application launch is a big weakness as it doesn't leave room for error. For example, if the user clicked the button by mistake or stopped the attack by mistake, they would have to relaunch and start again.

## Overall

While this project does have its weaknesses, the positives out way the negatives as the overall operation of the application works fine and the reverse engineering process can still be completed. The weakness only relate to making the process a small bit easier, but again fixes to these are not necessary in the RE process.

## Future Improvements

As I spoke about in the '[Original Concept](#)' section, I wanted to implement a camera functionality to automatically detect on the test subject if a change occurred. Once the camera detects the changes, it will store the corresponding message sent.

This idea would make the overall RE process smoother and quicker. It definitely is achievable as in my original research document, I found that CANON have software files that allows you to access a CANON Camera and connect it to your application. Unfortunately, I could never figure out how to make the camera detect changes in the test subject, but given more time and research, I believe I could implement this function and possibly improve on the idea in the future.

Along with the camera implementation, I spoke about the use of an SQL Database to store the potential messages and successful messages. The use of this would make it easier to pull the messages straight from the database and transfer it to the GUI to be checked. Now without this, the user must take the message from the Brute Message file and manually input the message for checking if it affects the test subject. Similar to the camera function, given more time and research, this is an implementation I would use in an updated iteration of this application.

Finally, in a future implementation, I would try to fix the plus and minus buttons underneath the input boxes to successfully increment and decrement by 1 instead of 2. I believe with further development of the button's functions, I would be able to succeed in this. Upon initial debugging, I found that the issue may lay within the function to convert the input value to a hex value. The problem may also occur due to the similar problem of the save, set and send message buttons completing their functions twice. This may be due to how the buttons are connected to the source files. But again, with more time and debugging I believe this issue could be patched in a new iteration of the application.



# Conclusion

## Security Implications

As said before, the purpose behind this project is to reverse engineer a vehicle's CAN Bus with the aid of a Brute Force Attack. This project presents several security implications. These implications consist of risk to potential unauthorised access, safety concerns and overall ethical considerations.

The risk of unauthorised access is seen in the use of the brute force attack, which could inadvertently give the user access to critical systems in a vehicle. In the hands of an attacker, they could exploit this functionality to control essential systems such as engine management, braking systems or power steering. Access to these in the wrong hands could lead to safety risks for the vehicle's passengers or even other road users.

The second risk mentioned is safety concerns. During the brute force attack or even the manual checking process, the user could unintentionally send a message that alters a critical system which could lead to safety risks and once again lead to the vehicle's passengers or other road users at risk of danger. This differs to the unauthorised access risk as a malicious user can exploit it, fully aware of their intention, whereas this risk could happen from a curious user testing some messages, leading to the execution of a function on a critical system unbeknownst to the user.

Lastly, we are left with the ethical considerations. For the purpose of this project, I wanted to show the possibility of the reverse engineering of a vehicle's CAN Bus. Now while this concept is interesting and quite technical, it can raise a red flag to ethical reasoning. If a user was to use this application and purposely manipulate a vehicle's systems without the proper authorisation it would raise questions on their actions. In this case the application should only be used on isolated test subjects the user knows to be safe and has permission to test on, or if testing on a full operational vehicle, they user must ensure they have the proper authorisation and permissions.

These security implications can be very serious in terms of real world application but also prevented once the application is applied in a safe manner with the essential permissions and proper authorisation in place.

## Overall Conclusion

This project report shows how the development and implementation of this application was successful in its attempt to reverse engineer a vehicle's CAN Bus with the aid of a brute force attack. The primary goal was to explore the functionality of the CAN Bus system and develop an application that allows a user to identify and manipulate messages to affect their test subject, in my case the dashboard apparatus.

In the report, I spoke about how I used the PCAN Basic functionality to initialise the connection and send the CAN messages in the format required. With these functions and information

gathered, I was able to succeed in the brute force, implement my own user functions, as well as have an application that makes the RE process a centralised and easy to follow system.

Through the testing analysis, I was able to find the flaws in my application and fix them as they were found. Unfortunately, I was unable to fix some of the functions that didn't operate to the full extend I intended, but overall, the GUI and its functions prove a purpose to help the user with the reverse engineering.

In conclusion, while my project operates as intended, it does have its weaknesses and concerns. These weaknesses can be the starting point in the development of a future iteration. This report also highlights the relevant security implications that too could be considered in a future iteration. Overall, this project contributes valuable insights into the understanding of a vehicle's communication system and shows the need for responsible research practices along with the responsible use of brute force attacks in a reverse engineering process.

## Plagiarism Declaration

1. I declare that all material in this submission e.g., thesis/essay/project/assignment is entirely my own work except where fully acknowledged.
2. I have cited the sources of all quotations, paraphrases, summaries of information, Tables, diagrams, or other material; including software and other electronic media in which intellectual property rights may reside.
3. I have provided a complete bibliography of all works and sources used in the preparation of this submission.
4. I understand that failure to comply with SETU's regulations governing Plagiarism constitutes a serious offence.

## References

1. <https://dewesoft.com/blog/what-is-can-bus>
2. <https://www.autopi.io/blog/can-bus-explained/>
3. <https://www.baeldung.com/cs/reverse-engineering>
4. <https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial>
5. <https://www.peak-system.com/PCAN-Basic.239.0.html?&L=1>
6. <https://www.qt.io/download-qt-installer-oss?hsCtaTracking=99d9dd4f-5681-48d2-b096-470725510d34%7C074ddad0-fdef-4e53-8aa8-5e8a876d6ab4>
7. <https://www.peak-system.com/PCAN-USB.199.0.html?&L=1>
8. <https://www.w3schools.com/cpp/>
9. <https://stackoverflow.com/questions/tagged/c%2B%2B>
10. [https://www.codecademy.com/catalog/language/c-plus-plus?g\\_network=g&g\\_productchannel=&g\\_adid=640978473026&g\\_locinterest=&g\\_keyw ord=learn%20c%2B%2B&g\\_acctid=243-039-7011&g\\_adtype=&g\\_keywordid=kwd-23862186&g\\_ifcreative=&g\\_campaign=account&g\\_locphysical=1007850&g\\_adgroupid=143797140465&g\\_productid=&g\\_source={sourceid}&g\\_merchantid=&g\\_placement=&g\\_partition=&g\\_campaignid=19230645493&g\\_ifproduct=&utm\\_id=t\\_kwd-23862186:ag\\_143797140465:cp\\_19230645493:n\\_g:d\\_c&utm\\_source=google&utm\\_medium=paid-](https://www.codecademy.com/catalog/language/c-plus-plus?g_network=g&g_productchannel=&g_adid=640978473026&g_locinterest=&g_keyw ord=learn%20c%2B%2B&g_acctid=243-039-7011&g_adtype=&g_keywordid=kwd-23862186&g_ifcreative=&g_campaign=account&g_locphysical=1007850&g_adgroupid=143797140465&g_productid=&g_source={sourceid}&g_merchantid=&g_placement=&g_partition=&g_campaignid=19230645493&g_ifproduct=&utm_id=t_kwd-23862186:ag_143797140465:cp_19230645493:n_g:d_c&utm_source=google&utm_medium=paid-)

[search&utm\\_term=learn%20c%2B%2B&utm\\_campaign=ESC\\_Language:\\_Basic\\_-\\_Exact&utm\\_content=640978473026&g\\_adtype=search&g\\_acctid=243-039-7011&gad\\_source=1&gclid=CjwKCAjwrlxhBbEiwACEqDJQXQDUrAvoR3reyCX1VXJS9dLJ0qbfWIFJVMU6ijQO91VKJKAT0uhoCdUEQAvD\\_BwE](https://www.google.com/search&utm_term=learn%20c%2B%2B&utm_campaign=ESC_Language:_Basic_-_Exact&utm_content=640978473026&g_adtype=search&g_acctid=243-039-7011&gad_source=1&gclid=CjwKCAjwrlxhBbEiwACEqDJQXQDUrAvoR3reyCX1VXJS9dLJ0qbfWIFJVMU6ijQO91VKJKAT0uhoCdUEQAvD_BwE)