# BetterCAN Documentation

***Release March 2018***

**William Putnam**

**May 23, 2018**

# Contents

**NOTE:** *This documentation is a work in progress and is being expanded when and where the opportunity arises. Please be patient.*

CHAPTER 1

# Contents

## 1.1 Introduction

This page briefly introduces BetterCAN and how it works.

### 1.1.1 What is BetterCAN?

BetterCAN is a update on the controller area network (CAN) bus standard to increase the overall security of the network and all the devices on it. From a structural standpoint according to the OSI model, the CAN bus network is based on the physical, data link, and application layers. This structure allows the CAN bus to send messages at almost real-time speeds. However, there are no protections built into this protocol, which can allows any external threat to easily hijack the network and control the devices connected to it.

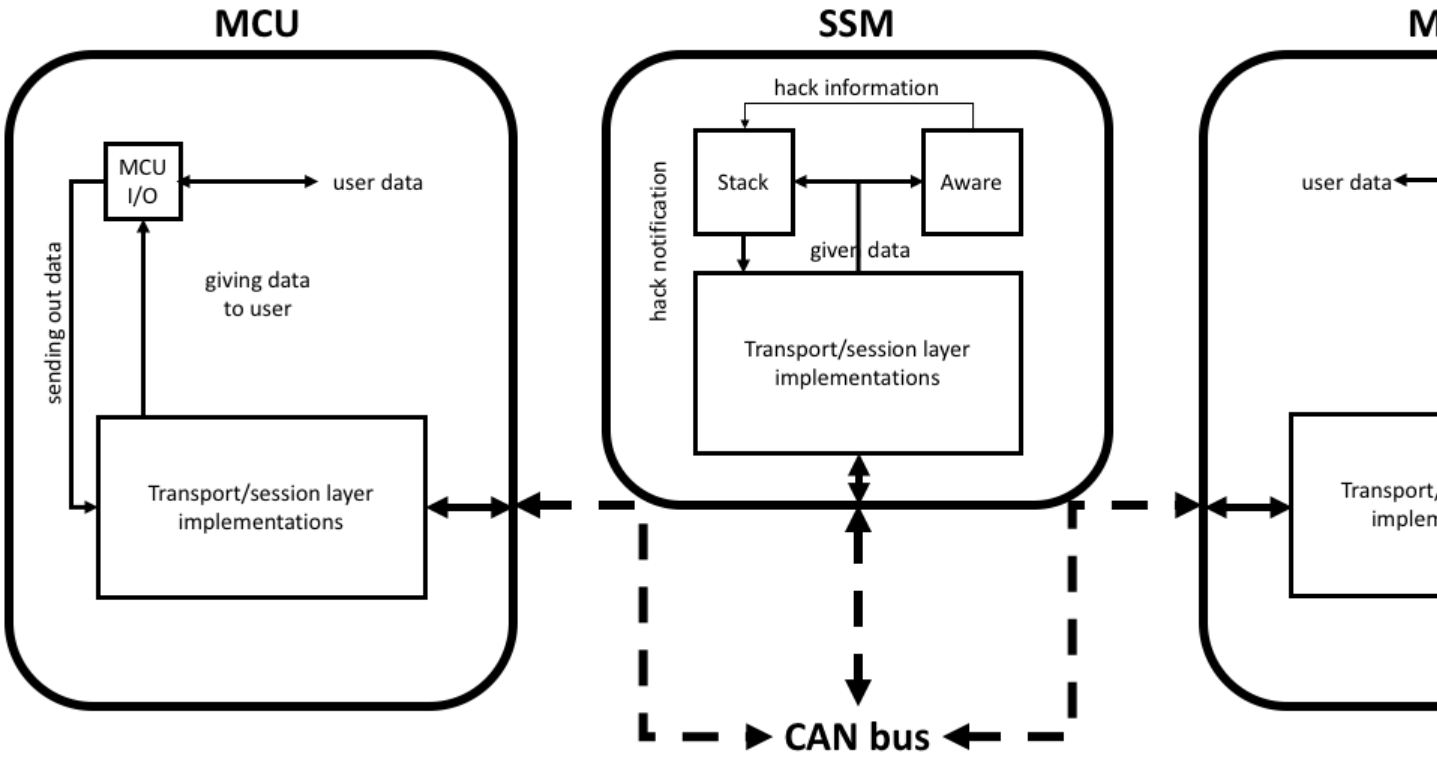
BetterCAN introduces a transport and session layer to the network, based on TCP and TLS protocols. Thanks to the introduction of these features, it becomes much harder for external threats to intercept, inject, or alter any messages being sent on the network.

### 1.1.2 How does it work [in a nutshell]?

BetterCAN is an expansion of the CAN protocol according to the OSI network.

There are three devices that are involved in sending a message. A sending device (MCU1) transmits a message to a third-party watchdog (signal security module, or SSM), which archives the data being sent without decrypting it. The SSM then sends the message to the destination device (MCU2), which replies with an ACK containing information that the SSM could use to determine if any message data was somehow compromised either from the end device or during transmission over the network.

The below diagram shows the general flow of the network:




All three devices parse incoming messages according to implemented transport and session layers, the latter of which utilizes 256-bit AES block ciphers to encrypt the contents of the session and application layers.

The SSM, in addition to parsing capabilities, runs two additional processes. One process stores [encrypted] data in a stack format (stack) and the other stores metadata of passed data and uses an awareness algorithm to detect any potentially suspicious or malignant activity (aware). In the event of a successful network breach, the SSM uses the awareness algorithm to determine unusual behavior, then terminate the connection while transmitting previous valid data values to the victim if needed.

For more detailed and technical information, please consult the `master's thesis` that this project was based on.

### 1.1.3 Where can BetterCAN be applied?

Although BetterCAN is still only a proof-of-concept, it has the potential to fully implemented onto existing CAN hardware, therefore eliminating the need for major hardware-based infrastructure upgrades.

BetterCAN was developed with the needs of the automotive industry in mind. It seeks to address the need for mission-critical speed and reliability while adding a much-needed layer of security. However, just about any security implementation comes at a performance cost. This performance cost can be partially adjusted by the system engineer, especially with respect to the awareness algorithm.

Of course, BetterCAN can be expanded outside the automotive industry. Because CAN is a simpler network than Ethernet, it can be used for many other applications, including but not limited to:

- fixed appliances, including medical devices and internal communications for Internet-of-things (IoT) devices
- other forms of transportation, such as trains and airplanes
- local networking hardware configurations, such as network routers

BetterCAN is an open source project, which means that anyone can apply BetterCAN for their own purposes in whichever field they work in that uses the CAN bus.

## 1.2 Legal

BetterCAN's code is released under the Apache 2.0 license. A copy of said license is in the top level of the project directory.

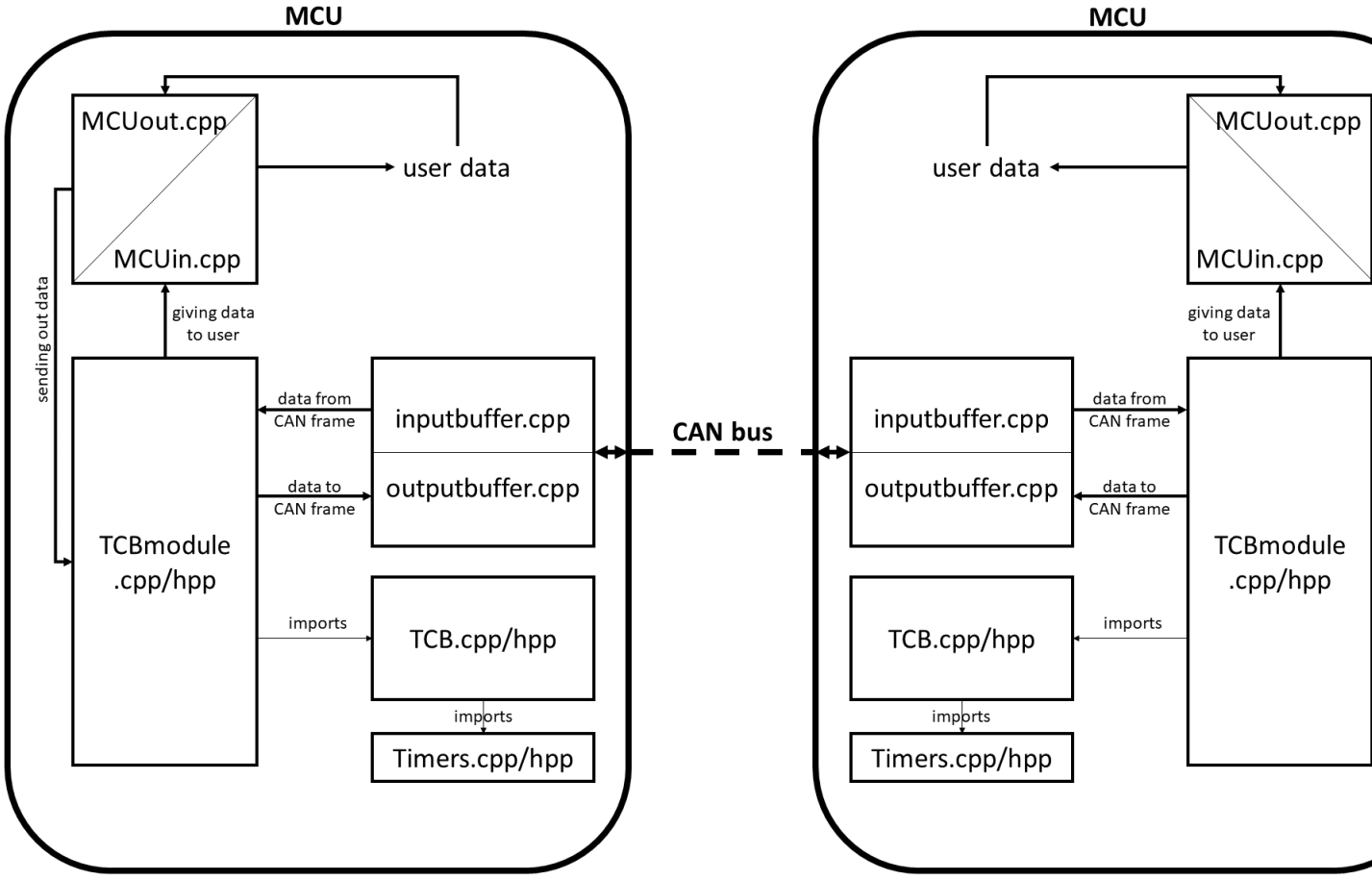
## 1.3 Contact

For any questions or discrepancies, please contact William Putnam at williamput3@aol.com.

If you find any issues with the code, please create an issue in the Github project.

## 1.4 Module layout

The below figure shows the internal structure of an MCU's communication, and the roles that each portion of source code plays in the system.



## 1.5 Functions

Below are basic explanations of each function in the source code by file group. (Singular files are explained under the section entitled *Other files*.

### 1.5.1 Aware files

- <*constructor*> - setting of the class object variables according to which connection it represents
- <*destructor*> - deletes the connection information from the awareness algorithm's vector
- *calcAvgs* - calculates the average value for each data set, and then passes the data on to the *decide function*
- *decide* - determines the type of behavior observed from the data provided by *calcAvgs*, and creates a tmp file with the related connection information
- *getSets* - adds a new data value to an awareness set in the class object, and records the time of the creation of a new data set
- *sameToID* - returns whether or not a supplied character matches the destination ID in the Aware object
- *sameFromID* - same as *sameToID*, but with the sender ID

### 1.5.2 TCB files

### 1.5.3 TCBmodule files

### 1.5.4 Timer files

- <*constructor*> - setting of the class object variables by type
- <*destructor*> - defined but empty
- *isExpired* - checks to see if the timer value for the Timer object has expired based on the type of timer it is
- *reset* - resets the timer value

## 1.6 Other files

- *CANinterface.sh* - establishes a single virtual CAN bus network interface
- *CANmult.sh* - establishes two virtual CAN bus network interfaces for MitM testing
- *keyGen.cpp* - creates a symmetric 256-bit AES to be pre-shared among all devices on the network
- *inputbuffer.cpp* - the data-link layer interface that receives messages on the CAN bus that match its assigned ID, and runs parallel to the TCB module
- *Makefile* - builds the source code and generates the executables required to run BetterCAN
- *MCUin.cpp* - part of the application layer for the MCU, where messages are simply read in after being parsed
- *MCUout.cpp* - part of the application layer for the MCU, where the transport and session layer parsers are established
- *outputbuffer.cpp* - the data-link layer interface that sends messages on the CAN bus to a specific destination ID, which is only ran when there is a message to send

- *SSMaware.cpp* - creates and manages a vector of connections for the awareness algorithm
- *SSMstack.cpp/hpp* - creates and manages the data stack where connection data is stored for parsing in the event of a network breach

CHAPTER 2

# Indices and tables

- genindex
- modindex
- search