SECURITY OF AUTOMOTIVE EMBEDDED SYSTEMS

Abstract

In the time of advancing autonomous vehicle technologies, the demand for secure embedded systems is growing. The reliability of vehicle systems depends on real-time data to ensure the safety of the vehicles and its passengers. It is imperative that these systems are fortified against attacks that could compromise data integrity and the real-time operations.

Specifically, this research investigates if two Arduino Megas, utilizing CAN Bus Shields, maintain both data integrity and real-time operations when subjected to a Glitching Attack. An experiment designed using the ChipWhisperer CW 308 Lite board performs a glitching attack on a target board with an Arduino Mega.

To assess the feasibility of impact of an attack, the Arduinos executes code that simulates the headlights of a vehicle. Metrics related to impact of attack are collected and presented.

Motivation/Problem

In the wave of automation, vehicles are being redesigned to be more reliant on software and electronics rather than its mechanics. Vehicles are relying more on automation to reduce accidents caused by drivers.

There are risks with automation that must be considered when designing an automated vehicle. The system can be vulnerable to wired or wireless attacks. These attacks can cause the driver to lose control of the vehicle. There is a high reward to successfully incorporate embedded systems, but this brings about new risks that must be discovered.

Can an embedded system in a vehicle maintain data integrity while under a physical attack?

Embedded Systems:
- A combination of electronic hardware and software that completes a specific set of functions within a larger system

Data Integrity:
- Integrity requires that the data is not modified between the sender and receiver

Physical Attack:
- Hardware connected to the target device that can introduce a glitch in the voltage or clock to disrupt the device or extract data

Hypothesis & Methods

In order to assess the risk of a glitch attack on a CAN bus, a headlight sub-system is modeled. Two Arduino Megas with CAN BUS shields form the two endpoints of the embedded system. One of the endpoints serves as the input controller (e.g., light switch inside a vehicle) and the other serves as the headlight controller. Communication between the two endpoints occurs using the 11898 CAN Bus protocol, facilitated by the CAN Bus shields.

A ChipWhisperer Nano will be placed on the CAN Bus communication wires or input controller source, replicating a plausible attack scenario as vehicle CAN Buses are easily accessible. The experiment consists of repeated application of the ChipWhisperer Nano glitching feature at different locations (e.g., input controller, CAN Bus itself), time offsets (from reset), glitch pulse width, and frequency and observation of the interruption (corruption), or lack there off, of the messages sent from the input controller to the headlight controller. Observation can be done through visual inspection and/or CAN bus protocol observations.

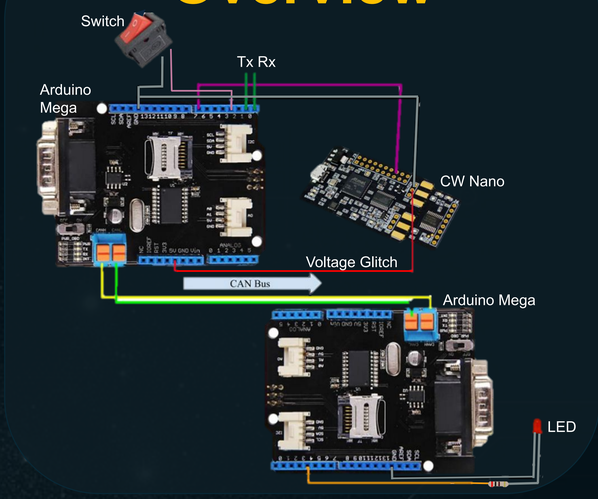
Current Work

The hardware was acquired and wired together according to the flow chart above. The CW Nano tutorials have been completed and the device has been made ready for implementation. The CAN BUS and Arduinos were tested using preliminary code to ensure initial functionality including a sender and receiver:

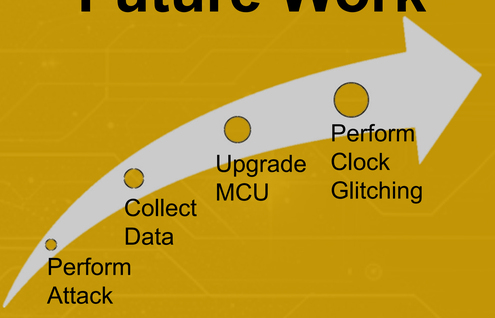
```ino
// Sends an alternating 1 and 0 signal every 500ms to toggle LED
#include <SPI.h>
#include "mcp2515_can.h"
#define SERIAL Serial

// Standard digital PIN shield uses to communicate SPI to Arduino
const int SPI_CS_PIN = 9;
mcp2515_can CAN(SPI_CS_PIN); // Set CS pin

void setup() {
    SERIAL.begin(115200);

    while (CAN_OK != CAN.begin(CAN_500KBPS)) {    //baudrate = 500k
        SERIAL.println("CAN BUS Shield init fail");
        SERIAL.println(" Init CAN BUS Shield again");
        delay(100);
    }
    SERIAL.println("CAN BUS Shield init ok!");
}

unsigned char stmp[1] = {1}; //initial message 1 (LED_ON)

void loop() {
    CAN.sendMsgBuf(0x01, 0, 1, stmp); //send data
    delay(500);                       //message every 500ms

    SERIAL.println("CAN BUS sendMsgBuf ok!");

//Alternate 0 and 1
    if(stmp[0] == 1) {
      stmp[0] = 0;
    } else {
      stmp[0] = 1;
    }
}
```

```ino
//Blinks LED based on input from send Arduino
#include <SPI.h>
#include "mcp2515_can.h"

const int SPI_CS_PIN = 9;
const int CAN_INT_PIN = 2;

mcp2515_can CAN(SPI_CS_PIN); // Set CS pin

void setup() {
    SERIAL_PORT_MONITOR.begin(115200);
    pinMode(4, OUTPUT); //Setting digital pin 4 for LED

    while (CAN_OK != CAN.begin(CAN_500KBPS)) { // baudrate = 500k
        SERIAL_PORT_MONITOR.println("CAN init fail, retry...");
        delay(100);
    }
    SERIAL_PORT_MONITOR.println("CAN init ok!");
}

void loop() {
    unsigned char len = 0;
    unsigned char buf[1];
    CAN.readMsgBuf(&len, buf);  // read, len, buf
    unsigned long canId = CAN.getCanId();

    SERIAL_PORT_MONITOR.println("-----------------------------");
    SERIAL_PORT_MONITOR.print("Get data from ID: ");
    SERIAL_PORT_MONITOR.println(canId);

    SERIAL_PORT_MONITOR.print(buf[0]);
    SERIAL_PORT_MONITOR.print("\t");

    digitalWrite(4, buf[0]); //Set LED to input
    SERIAL_PORT_MONITOR.println();
}
```

Overview



Future Work



Acknowledgements

Thank you to the DRACO Lab and the University of Central Florida for providing the space and equipment for this research.

SHERIDAN SLOAN
sheridan.sloan@ucf.edu
KATHERINE DOYLE
katherine.doyle@ucf.edu

DRACO LAB | www.ece.ucf.edu/DRACO
Dr. Mike Borowczak, Lab Director

Dept. Of Electrical & Computer Engineering
College of Engineering and Computer Science
University of Central Florida