**DoS Attack Detection and Mitigation on Autonomous vehicles using Raspberry Pi:**

**Attributes of CAN Protocol:**Two basic bits determine the transmission mechanism in the CAN Protocol: the recessive bit (logical "1") and the dominant bit (logical "0"). The dominant state of the CAN bus is represented by logical "0" when both bits are transmitted simultaneously. Bit stuffing is a method used to stop the transmission of identical bits again and again. When five consecutive similar bits are identified, this approach adds a complimentary bit to the CAN bus. CAN message transmission, in particular, ignores the source and depends only on the destination CAN IDs. This feature presents a serious vulnerability because it makes it simple for hackers to carry out attacks like spoofing. Error frames are also used to report errors to other nodes as they happen. The CAN architecture contains several forms of mistakes, bit errors being the result of differences between the bits that are broadcast and received. In order to handle these errors, the ECU notifies other ECUs via error frames, which causes the CAN controller's Receive Error Counter (REC) and Transmit Error Counter (TEC) values to increase. TEC increases by 8 and REC increases by 1 for every error that occurs.

**Attack Implementation:**The experimental hardware arrangement that was used for this research consists of four main parts: the attacker node, the countermeasure node, a prototype vehicle subsystem, and the CAN Bus, a straightforward two-wire system ended with 120Ω resistances. Critical automotive systems including the Airbag Deployment System, the Anti-lock Braking System (ABS), and the Adaptive Cruise Control System (ACC) are susceptible to Denial of Service (DoS) attacks. The purpose of this study is to introduce a denial-of-service attack against a prototype reverse parking assist system, as shown in below Figure, which was built utilizing two Arduino Uno R3 development boards in communication with two MCP2515 SPI-to-CAN modules.

A diagram of a circuit board

Description automatically generated

Fig: Prototype of Reverse parking System.

Any pre-existing or externally introduced node that is forced to carry out the DoS attack by flooding the CAN bus with pointless, high-frequency garbage data bits to gain arbitration over other nodes is known as the attacker node, or Electronic Control Unit (ECU). As a result, primary nodes are prevented from sending their data and come across an unavailable bus while doing so. Using Raspberry Pi 3B+ development boards, MCP2515 controllers are used to interface with the bus for the attacker ECU and countermeasure node prototypes, as shown in Figure.

A diagram of a circuit board

Description automatically generated

Fig: Attacker and Countermeasure Node Prototype.

By taking advantage of CAN ID's frame structure, the DoS attack is injected into the CAN bus. First, the validity of the CAN Frame is examined, and if it is found to be invalid, the Increment Frame (IF) Counter is raised. When there are constant invalid frame introductions that push the IF Counter beyond the threshold, the node malfunctions, signifying that a DoS attack has been successfully launched. Of the four nodes shown in the figures, one is the attacker node that broadcasts incorrect frames nonstop, while the other is the node that is being attacked by a denial-of-service attack.

**Attack Explanation:** The buzzer node (receiver) and the ultrasonic sensor node (transmitter) can communicate with each other when the CAN bus is initially configured and functioning normally. A series of "1" data bits are sent over the CAN bus by the ultrasonic sensor when it detects there are no obstacles. The SPI protocol transforms these "1" bits to ASCII, which is then converted into the hexadecimal representation "31" on the Raspbian terminal. In response, the buzzer node just stays inactive and flashes the message "no obstacle detected."

Similarly, a series of "0" data bits are sent over the CAN bus by the ultrasonic sensor in response to the detection of an obstacle. On the receiver node terminal, this data is converted to "30". The buzzer node responds by turning on and notifying people that an "obstacle had been detected."

However, to obstruct authorized communication, an attacker node that launches a Denial of Service (DoS) attack on the CAN bus floods the bus with useless high-frequency garbage data bits. To effectively overload the transmitter node in this case, the attacker node sends "2 3" data bytes with a higher priority message ID. The transmitter node is unable to send its messages quickly due to the flood of junk data. The overload caused by the DoS attack prevents valid communication from occurring on the CAN bus. Python code in the receiver recognizes an increase of invalid data and reacts by displaying the message "invalid data," which denotes that the attack has affected the system's ability to function.

**Computation/Communication Resources needed for Attack:** To carry out the Denial of Service (DoS) assault on the Controller Area Network (CAN) bus, the attacker needs an MCP2515 controller and a Raspberry Pi 3B+ development board. In addition, any pre-existing or externally inserted node that wins the arbitration against the other nodes and is compelled to carry out the DoS attack by transferring useless high-frequency garbage data bits over the CAN bus may be the attacker node.

**Attack Mitigation:**  
 An efficient countermeasure is put in place to reduce Denial of Service (DoS) assaults on the CAN bus, making use of the built-in BUS-OFF feature of the CAN protocol. A specifically built countermeasure node must continuously monitor the bus for the existence of invalid messages to implement this countermeasure. When the countermeasure node notices an unusual flood of false frames that might indicate a denial-of-service effort, it launches a BUS-OFF attack directed at the attacker node. By forcing the attacker node into an error-passive state, the BUS-OFF attack successfully stops the malicious activity (as shown in Fig1). After that, the attacker node is switched to the BUS-Off state, which cuts it off from the network and stops it from sending any further illegal data. The integrity and functionality of the CAN bus are maintained by isolating the attacker node, permitting appropriate communication to resume uninterrupted. This mitigation strategy ensures that the vehicular subsystems relying on the CAN bus remain protected from disruptive attacks, safeguarding the overall functionality and security of the automotive system.

A diagram of a error

Description automatically generated

Figure 1: CAN Error State Diagram [1]

**References:**  
[1] Salunkhe, P. D., Patil, U. N., Patel, N. J., & Sontakke, P. V. (2022). DoS attack detection and mitigation for autonomous vehicles using Raspberry Pi. *Int. J. Res. Appl. Sci. Eng. Technol.*, *8*(9), 657-666.  
[2] Cho, K. T., & Shin, K. G. (2016, October). Error handling of in-vehicle networks makes them vulnerable. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 1044-1055).