iJRASET
International Journal For Research in Applied Science and Engineering Technology



# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 8 **Issue:** IX **Month of publication:** September 2020

**DOI:** https://doi.org/10.22214/ijraset.2020.31546

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429*
*Volume 8 Issue IX Sep 2020- Available at www.ijraset.com*

# DoS Attack Detection and Mitigation for Autonomous Vehicles using Raspberry Pi

Param D. Salunkhe[1], Uddesh N. Patil[2], Nehul J. Patel[3], Prakash V. Sontakke[4]
[1, 2, 3] *Student, Dept. of Electronics & Telecommunication Engineering, Pimpri Chinchwad College Of Engineering, Pune, Maharashtra, India - 411044*
[4] *Asst. Professor, Dept. of Electronics & Telecommunication Engineering, Pimpri Chinchwad College Of Engineering, Pune, Maharashtra, India - 411044*

***Abstract:*** ***Recent automotive systems are increasingly complex and networked using various protocols like CAN, LIN, MOST, FlexRay, etc. The introduction of these technologies adds to the vulnerability of the vehicle security and gives more opportunities for hackers and thieves to implement various malpractices for their own benefit or to inflict harm. Advancements in vehicular communication technologies have given rise to the possibility of various network-based attacks. This project focuses on the Controller Area Network (CAN) protocol and how its vulnerabilities can be exploited to perform malicious attacks like Denial of Service (DoS) attack on an in-vehicle network prototype. A prototype reverse parking assist system has been used as the system under test, throughout this paper. Here, we have also introduced a countermeasure for the DoS attack which shuts off the adversary node from the CAN bus, so that the ideal functioning of the network will resume.***
***Keywords: CAN bus, Raspberry Pi, DoS attack, ECU, Bus-off attack, etc***

## I. INTRODUCTION

New protection techniques in vehicles are being developed because of the introduction of Electronic Control Units (ECUs) that are software-driven along with the remote networks of present-day vehicles [1,2,3]. These patterns have presented increasingly remote facets that an enemy can misuse and, in the most exceedingly awful case, remotely control or manipulate the vehicle. It merits harping on the potential issues that hacking postures to associated vehicles and two such notable models incorporate those of a hacked Jeep Cherokee [4,5] and Tesla's Model X [6]. Security analysts have shown over the most recent two years that it is conceivable to not just hack into the frameworks of associated vehicles, yet to hold onto control of imperative capacities, for example, slowing down and directing [7,8]. As per the current trend of the market, the approach of vehicle users and manufacturers has been completely changed as they are keen to automate the system to make it more user friendly.

But this will make the system vulnerable to some cyber-attacks like DoS attack, SYN flood attack, etc. [1,9]. Hence a proper step should be taken in order to secure the system which can withstand such attacks. CAN (Controller Area Network) is a protocol designed by BOSCH in 1991 [10,11,12]. It has been widely adopted in the automotive sector in order to communicate different devices through a data serial bus, from the engine, suspension, and traction controls, to lights, doors, seats, instruments, and even light and environment devices. The vulnerability of the CAN network can be exploited using some attacks like spoofing, DoS [13], etc., which are often used for intruding in-vehicle networks and are described in [14,15,16]. In CAN network, spoofing a message injection has been reported to make it achievable to access Tyre Pressure Monitoring System (TPMS), Antilock Braking System (ABS), Door Control Unit (DCU), etc.[17,18,19].

This project aims to propose a system that aids the prevention of functional misbehavior of autonomous vehicles due to the infestation of a Denial of Service (DoS) attack.

## II. METHODS

### *A. Block Diagram*

As shown in Fig. 1., we have used a CAN bus that uses two-wires carrying different signals: one is called CAN-H (High) and the other CAN-L (Low). This CAN bus is connected to a CAN transceiver (TJA1050) [20], and CAN controller (MCP2515) [21]. These are the modules that are used to make CAN bus compatible with our 4 nodes which pose as Electronic Control Units (ECUs) in a car. The following figure represents the block diagram of a the proposed system.

©IJRASET: All Rights are Reserved

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429*
*Volume 8 Issue IX Sep 2020- Available at www.ijraset.com*

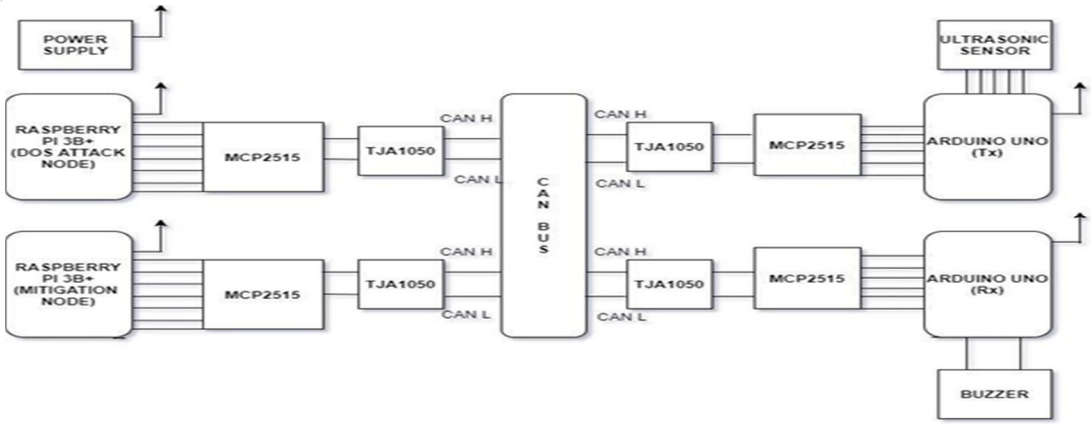



Fig. 1. Block Diagram

Out of these 4 nodes one is Attacking Node (Raspberry Pi), a countermeasure node (Raspberry Pi) and finally, two nodes (Arduino) that represent a prototype Reverse Parking Assist system which is to check the behavior of the system under various situations i.e., before the attack, after the attack, and after mitigation.

*B. Flow Chart*

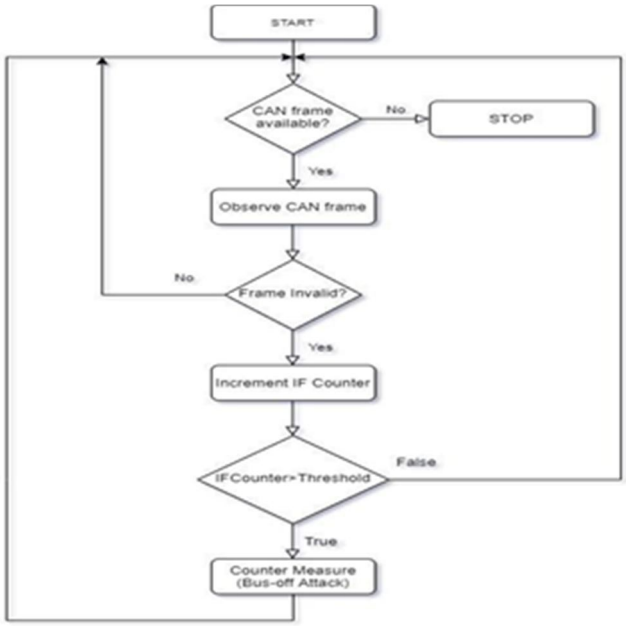



Fig. 2. Flow Chart

The flow chart shown in Fig. 2. exactly depicts how the DoS attack introduction is to be carried out and how the mitigation countermeasure is deployed. As shown in the flowchart we have used CAN bus sniffing. Here, the mitigation device (Raspberry Pi) will monitor the CAN bus, observe the incoming CAN frames on the bus. Each frame will be then compared against a set of predefined CAN frames that are already stored in the mitigation device program memory. If the incoming frame does not match any of the predefined frames, it will be considered as an Invalid Frame (IF). When an invalid frame is encountered, a counter named Invalid Frame Counter (IF counter) will be incremented. During the occurrence of a DoS attack, the attacker node crowds the CAN bus with multiple such invalid frames [22] and for every invalid frame, the IF counter will increment. Once the IF counter increments beyond a certain predefined threshold, the system is considered to have detected the attack and it starts with the counterattack process. In order to mitigate the attack, we are using an error handling property of CAN Bus called the CAN Bus-Off property, which is discussed in detail in the further sections.

©IJRASET: All Rights are Reserved

## III. METHODOLOGY

### *A. CAN Protocol*

CAN protocol is used to interconnect ECUs with a message bus. Basically, there are four types of CAN data frames: a data message frame, a remote frame for seeking communication of data, an error frame used to point out errors observed, and overload frames to introduce a setback among two frames[23].

The data frame of CAN Bus is shown in Fig. 3.

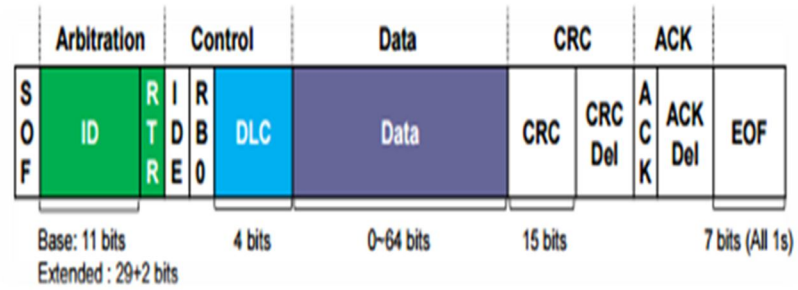



Fig. 3. CAN data Frame

The frame starts with a Starting bit (SOF). It is followed by the message ID. CAN is a broadcast type bus. Every instruction has a priority when it is on the bus. The ID field is responsible for deciding the message priority. The default size is 11 bits. Extended CAN ID is of 29 bits, obtained by setting the IDE bit. 4-bit Data Length Code (DLC) is where data length is stored. This is followed by a data or message field which can have up to 64 bits of the message. In the last byte of the data field, a 1-byte checksum is provided for most vehicles. This field is followed by the Cyclic Redundancy Check (CRC), ACK fields used for error control, and finally a string of 7 recessive bits indicating the end of the frame (EOF).

1) *Attributes of CAN Protocol:* In the CAN Protocol, there are two bits of which first is the dominant bit which is logical "0" and another is a recessive bit which is logical "1". When we transmit both the bits at the same time the resulting CAN bus will be dominant i.e. logical "0".To avoid transmission of the same bits continuously, the property of bit stuffing is used. Thus when there is a transmission of 5 same continuous bits is observed one complementary bit is introduced in the CAN-BUS. The transmission of CAN messages is only dependent on the destination CAN ID not on the source and this is one of the remarkable features of CAN Protocol. As it only works on destination IDs, the ECU will remain unknown about the source of the CAN ID transmitted on the CAN bus. Hence, here comes the first disadvantage that the attacks like spoofing can be easily introduced. A frame that indicates to the other nodes that an error has occurred, is called as an error frame. In CAN there are multiple types of errors. A bit error happens when there is a difference between the transmitted and received bits. To manage this, the ECU sends an error frame to inform the other ECUs, and in the CAN controller, the values of Receive Error Counter (REC) and Transmit Error Counter (TEC) get incremented. For every error received, there is an increment of 8 in TEC and 1 in REC. The CAN error state diagram is shown in Fig. 4 below.

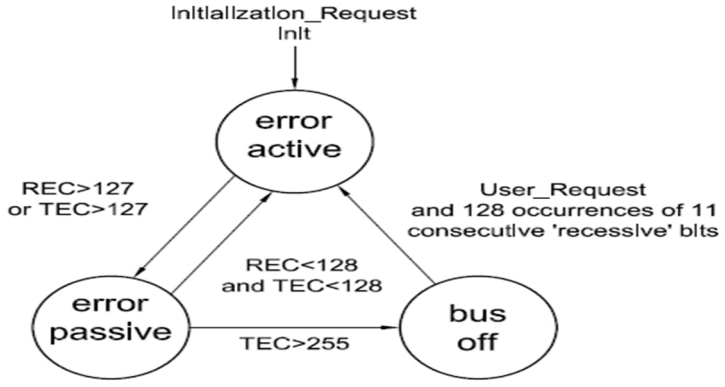



Fig. 4. CAN Error State Diagram

©IJRASET: All Rights are Reserved

*B. DoS Attack on CAN Bus*

The experimental hardware setup used for the project is shown below, in Fig. 5.

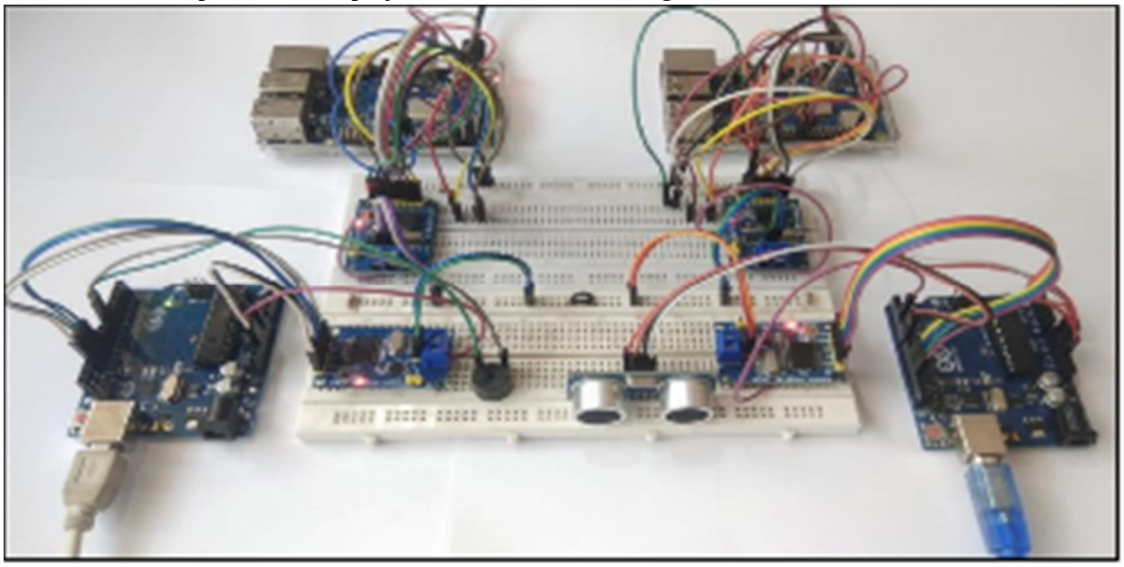


Fig. 5. Hardware Implementation

This setup comprises of four main parts: the CAN Bus, which is a simple two-wire system terminated with 120Ω resistances, a prototype model of a vehicular sub-system, the attacker node and the countermeasure node.

The DoS attack can be performed on any sub-system in the vehicle, examples including some crucial systems like the Airbag Deployment System, Anti-lock Braking System (ABS), and Adaptive Cruise Control System (ACC). In this project, we introduce the DoS attack on a prototype Reverse Parking Assist System which is designed using two Arduino Uno R3 development boards that communicate with each other over CAN using two MCP2515 SPI-to-CAN modules. Hardware implementation of this sub-system is shown below, in Fig. 6.

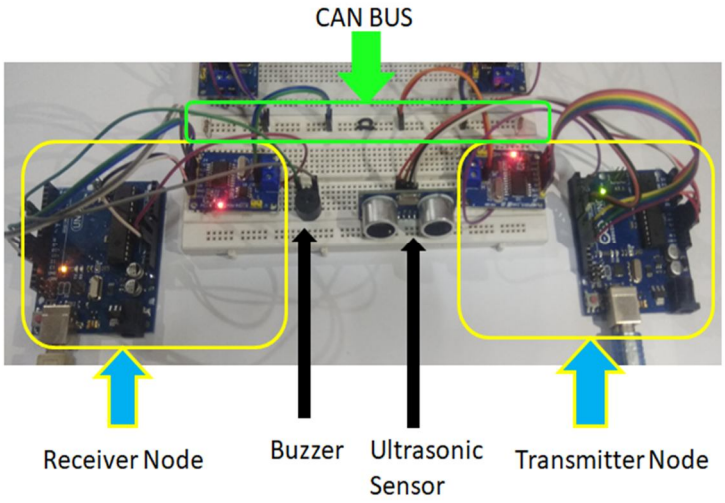



Fig. 6. A prototype of a Reverse Parking Assist System

The Attacker node or ECU maybe any pre-existing or externally added node which is forced to perform the DoS attack [24] by injecting useless, high-frequency garbage data bits over the CAN bus by winning the arbitration over the other nodes. Thus, the main nodes do not get the opportunity to transmit their data and they do not find the bus available at the time of transmission. For the prototype attacker ECU and countermeasure node, we have used Raspberry Pi 3B+ development boards. These again, use the MCP2515 controllers to communicate with the bus [25]. Hardware implementation is shown in Fig. 7.

©IJRASET: All Rights are Reserved

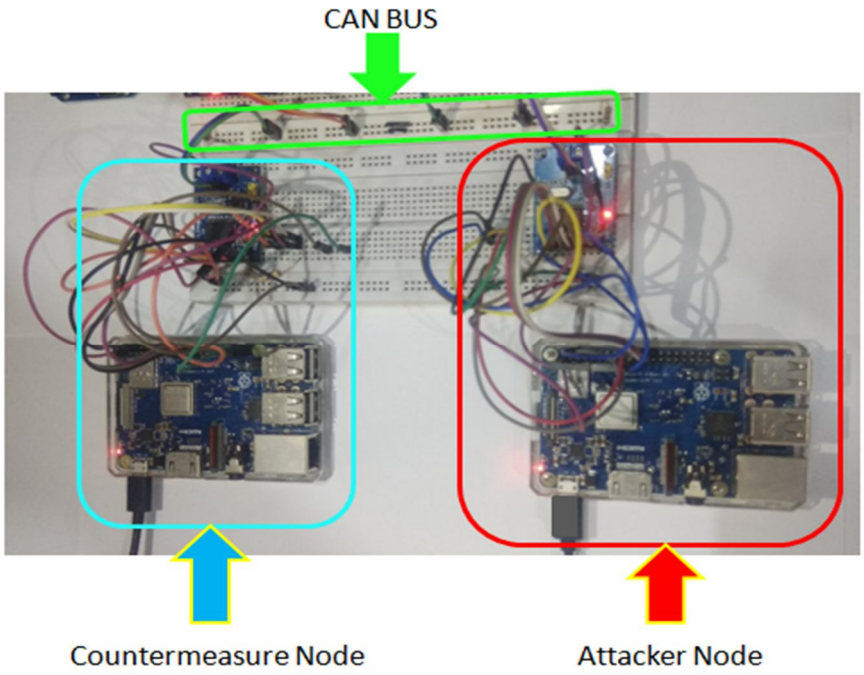



Fig. 7. Attacker and Countermeasure node hardware

As mentioned in the flow chart we have introduced the DoS Attack in a CAN bus by exploiting the frame structure of CAN ID. Here at first, the CAN Frame is observed for its validity and if it is invalid then the IF Counter will be incremented. Once the IF Counter is incremented above the threshold due to the continuous introduction of invalid frames the node will malfunction (DoS Attack has been introduced).
From the four nodes shown in the block diagram, one will be the attacker node which will continuously send invalid frames another is the victim node which will be under the DoS attack and another two nodes are designed to observe the output during the entire process.

*C. Mitigation of DoS Attack*

1) *CAN Bus Off:* In order to monitor bus health (and also their own health), CAN controllers must keep two counters, called transmit error counter along with error counter at receiving end that s REC. They start at zero and are incremented (upon error) and decremented (whenever the controller performs a successful Tx/RX) according to a group of rules specified by the CAN standard. The value of these counters affects the error-handling mode of the controller (error-active versus error-passive) and, ultimately (when the error counter of transmitter i.e. TEC exceeds the value 255), the alteration to the Bus off state. While it is in this state the controller switches off from the bus. So mandatorily, it stops transmitting and acknowledging frames. Whether or not the node keeps receiving frames depends on the implementation[11,26,27].
2) *Mitigation Methodology:* Here, we are using the property of CAN Bus to mitigate the DoS Attack caused due to intrusion of invalid CAN ID Data Frames. As the process of introduction of invalid CAN IDs goes on at one point that one particular node starts to misbehave where we conclude that Denial of Service i.e. DoS attack has been incorporated. To Secure the system i.e. to mitigate the system from ongoing DoS attack we Use the BUS-OFF property of CAN Module. As explained earlier there are two counters in the CAN Module of which the first is Error counter at transmission (TEC) and another is error Counter at the receiver (REC). As shown in the flowchart, a counter (invalid frame counter) is maintained by the countermeasure node which sniffs the CAN bus for invalid messages. Once the counter gets incremented beyond a predefined threshold, it will introduce this BUS-OFF Attack on the attacking node which is continuously attacking the victim node. So this attack will push the attacker node from error active state to error passive state and so next in the BUS-Off state due to which attacking node gets completely cut off from the unit. Due to this, the victim node will not receive any invalid CAN frame and will result in the proper working of all the remaining nodes.

©IJRASET: All Rights are Reserved

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429*
*Volume 8 Issue IX Sep 2020- Available at www.ijraset.com*

## IV. RESULTS

### *A. CAN Communication*

When the discussion comes to the CAN Nodes, the Communication between those two nodes is of utmost importance. Hence to achieve this One Raspberry Pi was assigned as the Sender node and the other was assigned as the receiving node. The receiver node is made ready to receive data via the CAN bus using the following command[28,29]:

*candump any*

Data was sent through the sender node using the command:

*cansend can0 -dataframe-*

For example,

*cansend can0 111#FF*

Figure 8 below, shows the data sent through the sender node that is data from node 1 which is a combination of Raspberry pi, CAN Controller and CAN transceiver.

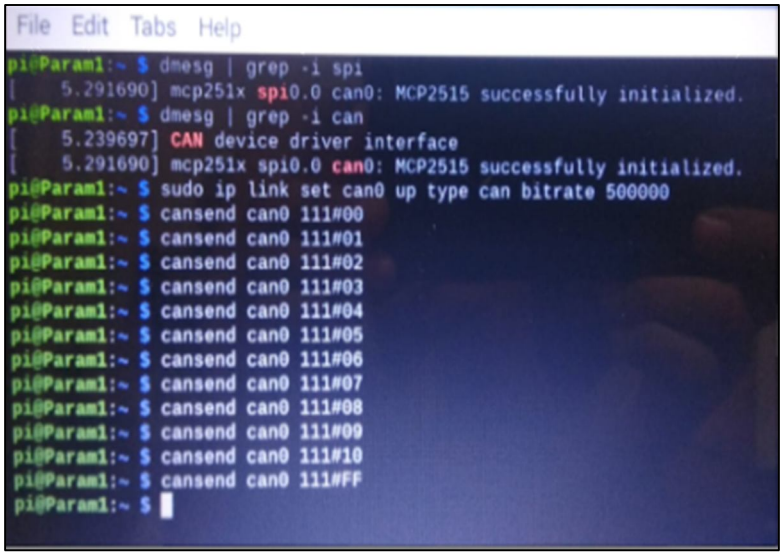



Fig. 8. Data sent through the sender node

Figure 9 below, shows the data received on the receiving node 2. This is achieved using the command: *candump any*. This command is used by the terminal to sniff the incoming data on any R-Pi supported CAN channel [30].

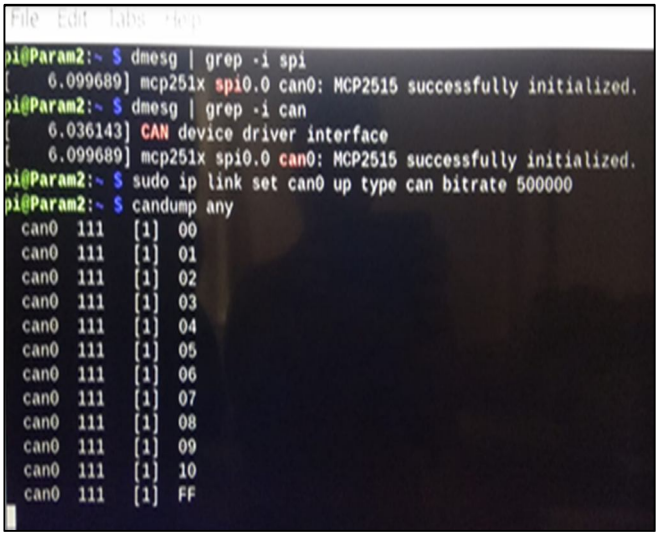



Fig. 9. Data received on the receiving node

©IJRASET: All Rights are Reserved

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429*
*Volume 8 Issue IX Sep 2020- Available at www.ijraset.com*

*B. DoS Attack Implementation*

Figure 10 and Fig. 11below, show the data on CAN bus in normal operating conditions i.e., DoS attack hasn't been performed yet. The ultrasonic sensor node (transmitter) sends data "1 1 1 1 1 1 1 1" when it detects no obstacle.

Now, 1 is converted to ASCII by the SPI protocol, which makes it 49. This 49 is then read by the Raspbian terminal in hexadecimal format, which makes it 31. Thus, when "1 1 1..." is transmitted, "31 31 31..." is observed on the terminal in Fig. 10. The buzzer node (receiver) responds to this with no buzzer action and a message "no obstacle detected". The code for the same is run using Python 2.7 and the output of the same is shown in Fig. 11.

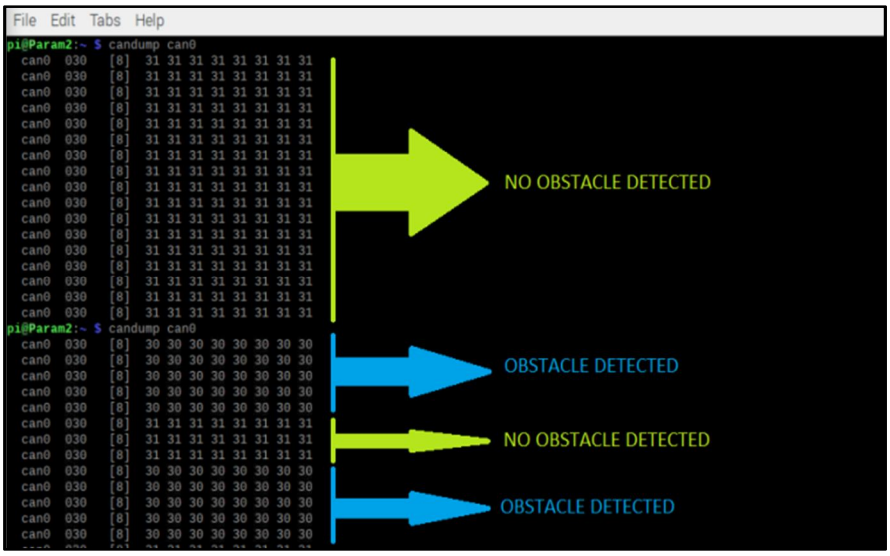



Fig. 10. Data observed on R-Pi terminal screen before DoS Attack

It is also shown in Fig. 10 that when the transmitter node detects an obstacle, it sends the data "0 0 0 0 0 0 0 0" on the CAN bus, which is indeed, read by the receiver node terminal as "30 30 30 30 30 30 30 30". To this, the receiver node responds by activating a buzzer. A message saying, "obstacle detected" is displayed as an output of the receiver's Python code. This message can be used to alert the passenger in the car. This output is shown in Fig. 11.

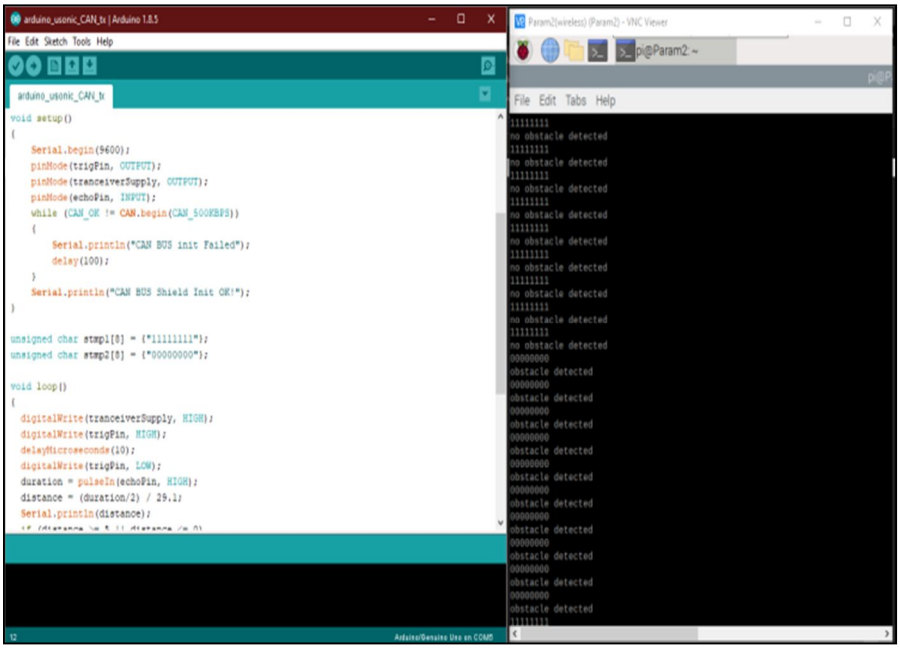


Fig. 11. Python code output before DoS Attack

©IJRASET: All Rights are Reserved

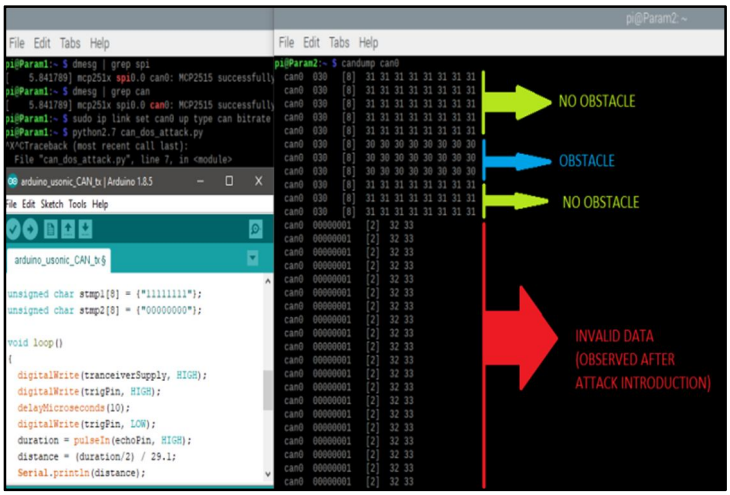



Fig. 12. Data observed on R-Pi terminal screen after DoS Attack

Figure 12 above, shows the data on the CAN bus, observed on the Raspbian terminal when the DoS attack is implemented on the prototype reverse parking assist system (system under test). The attacker node will have a higher priority message ID, so that it wins arbitration over the transmitter node and transmits 2 bytes of irrelevant, garbage data "2 3" (read as "32 33" on the receiver terminal) with higher transmission frequency than the transmitter node.

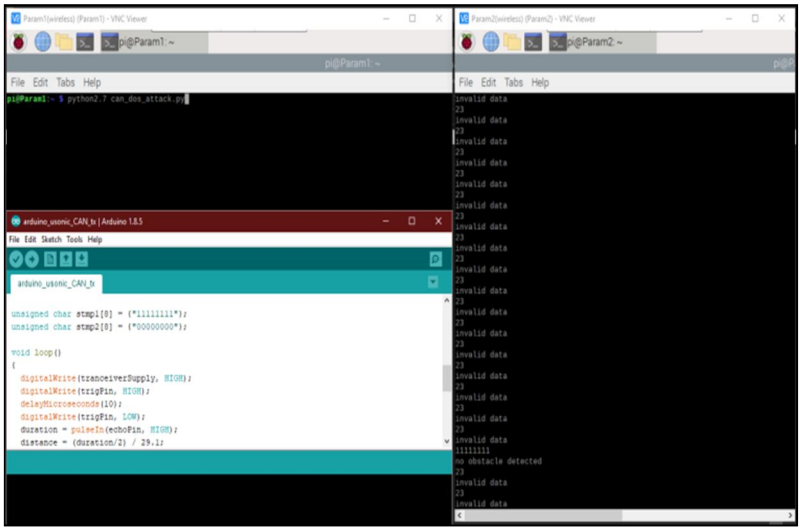


Fig. 13. Python code output after DoS Attack

The attacker overcrowds the CAN bus with this garbage data and thereby, doesn't allow the transmitter to send its messages as frequently as in the normal operation.
This is a Denial of Service (DoS) attack as the CAN bus is 'made unavailable' to the transmitter ECU. The receiver's Python code will respond to these garbage values with a message "invalid data". This output is shown in Fig. 13.

### *C. Expected Result of Mitigation*

As explained in methodology, after detecting the attack on the system, i.e. on detecting the invalid frames, once the IF counter gets incremented above the decided threshold, the attacker node will acquire the message ID of the attacker node by sniffing the CAN bus and launch a Bus-off attack on the attacker. This will force the intruder into the Bus-off error state and stop transmitting its messages over the CAN bus. This will stop the bus from overcrowding. The transmitter node can then send its CAN frames on the bus at the required time intervals, properly. This will thus, ultimately result in the ideal behavior and functioning of the CAN bus and the prototype reverse parking assist system.

©IJRASET: All Rights are Reserved

## V. CONCLUSION

The proposed methodology will prove to be effective in regaining control over the disturbed CAN communication.
It can be implemented in an existing in-vehicle network, without any significant modification in the overall system.
Here successful exploitation of some properties of CAN is achieved. The proposed mitigation system is cost-effective.

## VI. FUTURE SCOPE

It can be implemented in an In-vehicular CAN network to protect the control units from external network-based attacks.
A thorough study will explore some more ways to exploit the vulnerabilities of CAN and implement the countermeasures resulting in safe and secure control units.

## VII. ACKNOWLEDGEMENT

It is our pleasure to present some words of acknowledgment to the ones who have helped and mentored us in various ways throughout this project. Our group especially thankful to the staff of E&TC dept., Pimpri Chinchwad College of Engineering who provided guidance and progressive suggestions to make the project more effective and efficient. We are also grateful to our colleagues whose help and suggestions were an important part of the successful completion of the current state of our project. Finally, we are immensely thankful to our parents for their unparalleled motivation, appreciation, and support through the ups and downs we faced while working on this project.

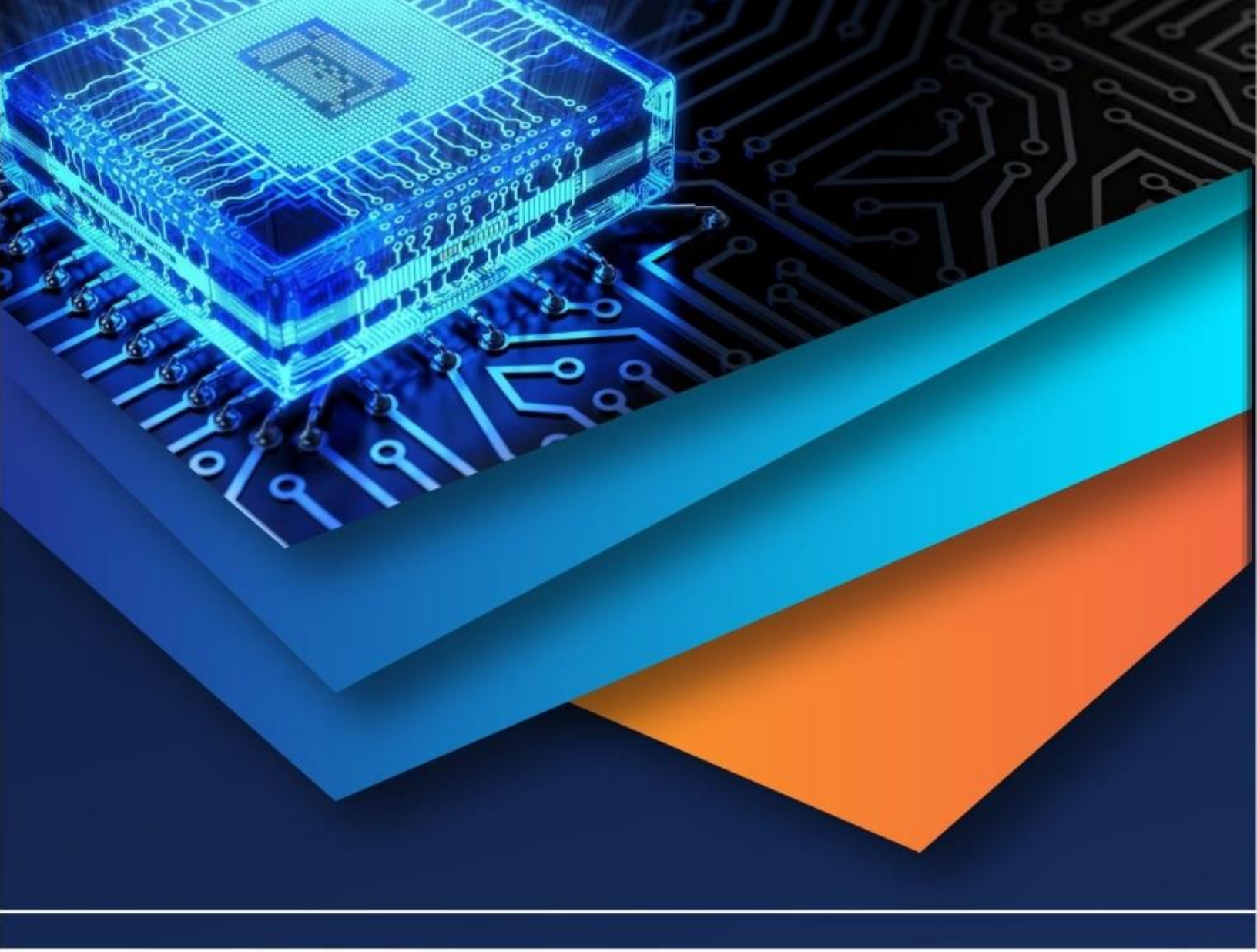
## REFERENCES

[1] V. L. L. Thing and J. Wu, "Autonomous Vehicle Security: A Taxonomy of Attacks and Defences," 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Chengdu, 2016, pp. 164-170.
[2] Lin, C.W., Sangiovanni-Vincentelli, A.: "Cyber-Security for the Controller Area Network (CAN) Communication Protocol." ASE Sci. J. 1(2), pp. 80-92 (2012)
[3] Checkoway, S., McCoy, D., et al.: "Comprehensive Experimental Analyses of Automotive Attack Surfaces". In USENIX Security, 2011.
[4] C. Miller and C. Valasek "Adventures in automotive networks and control units", Defcon 21, 2013.
[5] C. Miller and C. Valasek "Remote exploitation of an unaltered passenger vehicle", Black Hat USA, 2015.
[6] Nie, S., Liu, L., Du, Y.: Free-fall: "Hacking a Tesla vehicle from wireless to CAN bus". Black Hat USA, 2016
[7] L. Liang, K. Zheng, Q. Sheng and X. Huang, "A Denial of Service Attack Method for an IoT System," 2016 8th International Conference on Information Technology in Medicine and Education (ITME), Fuzhou, 2016, pp. 360-364
[8] A. Chattopadhyay and K. Lam, "Security of autonomous vehicle as a cyber-physical system," 2017 7th International Symposium on Embedded Computing and System Design (ISED), Durgapur, 2017, pp. 1-6.
[9] Martijn Bruning, "An Analysis of the DDoS Potential of Single Board Computers: A Raspberry Pi Case Study" University of Twente, The Netherlands in International Journal of Computer Science and Network Solutions (IJCSNS)
[10] A. U. Jadhav and N. M. Wagdarikar, "A review: Control area network (CAN) based Intelligent vehicle system for driver assistance using advanced RISC machines (ARM)," 2015 International Conference on Pervasive Computing (ICPC), Pune, 2015, pp. 1-3.
[11] CAN Specification Version 2.0 By BOSCH., 1991
[12] ISO 11898:2015 Road Vehicles - Controller Area Network (CAN), 2015
[13] Mukherjee, S., Shirazi, H., Ray, I., Daily, J., Gamble, R.: Practical DoS attacks on embedded networks in commercial vehicles. In: 2016 International Conference on Information Systems Security (ICISS 2016), pp. 23–42 (2016)
[14] Cho, Kyong-Tak & Shin, Kang, "Error Handling of In-vehicle Networks Makes Them Vulnerable". ACM SIGSAC proceedings of Conference on Computer and Communications Security, pp. 1044-1055, 2016
[15] K. Iehira, H. Inoue and K. Ishida, "Spoofing attack using bus-off attacks against a specific ECU of the CAN bus," 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, 2018, pp. 1-4
[16] Palanca, Andrea & Evenchick, Eric & Maggi, Federico & Zanero, Stefano. (2017). A Stealth, Selective, Link-Layer Denial-of-Service Attack Against Automotive Networks. 185-206. 10.1007/978-3-319-60876-1_9.
[17] Song, H.M., Kim, H.R., Kim, H.K.: "Intrusion Detection System Based on the Analysis of Time Intervals of CAN Messages for In-Vehicle Network", ICOIN (2016)
[18] Wolf, M., Weimerskirch, A., Paar, C.: "Secure In-Vehicle communication". In: Lemke, K., Paar, C., Wolf, M. (eds.) Embedded Security in Cars, pp. 95-109. Springer, Heidelberg (2006)
[19] Dagan, T., Wool, A.: "Parrot, A Software-Only Anti-Spoofing Defence System For the CAN Bus". In: 5th Embedded Security in Cars (ESCAR Europe)(2016)
[20] Datasheet: NXP (Philips) TJA1050 CAN Tranceiver [Online]
[21] Datasheet: Microchip MCP2515 Stand-Alone CAN Controller with SPI interface [Online]
[22] Godbole, N., Belapure, S., "Cyber Security", WILEY publications, pp. 158-163, 2011.
[23] KVASER: CAN Protocol Tutorial
https://www.kvaser.com/can-protocol-tutorial/

©IJRASET: All Rights are Reserved

[24] Si, W., Starobinski, D., Laifenfeld, M.: Protocol-compliant DoS attacks on CAN: demonstration and mitigation. In: 2016 IEEE 84th Vehicular Technology Conference (VTC-Fall) (2016)
[25] M. Takada, Y. Osada and M. Morii, "Counter-Attack Against the Bus-Off Attack on CAN," 2019 14th Asia Joint Conference on Information Security (AsiaJCIS), Kobe, Japan, 2019, pp. 96-102
[26] Souma, D., Mori, A., Yamamoto, H., & Hata, Y. (2018). Counter Attacks for Bus-off Attacks. Developments in Language Theory Lecture Notes in Computer Science, 319–330. DOI: 10.1007/978-3-319-99229-7_27
[27] Souma, D., Mori, A., Yamamoto, H., & Hata, Y. Counter Attacks for Bus-off Attacks. https://www.iit.cnr.it/strive2018/presentazioni/counter_attacks_for_bus-off_attacks.pdf
[28] Installing Python-CAN on Raspberry Pi 3 Model B+: https://skpang.co.uk/blog/archives/1220
[29] Python-CAN Documentation for Raspberry Pi : https://python-can.readthedocs.io/en/2.1.0/index.html
[30] Raspberry Pi model 3B+ product manual [Online]

©IJRASET: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)