Proceedings of the International Conference on Artificial Intelligence and Smart Systems (ICAIS-2021)
IEEE Xplore Part Number: CFP21OAB-ART; ISBN: 978-1-7281-9537-7

2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS) | 978-1-7281-9537-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICAIS50930.2021.9395871

Implementation of Replay Attack in Controller Area Network Bus using Universal Verification Methodology

Thirumavalavasethurayar P and Ravi T
Department of Electronics and communication
Sathyabama Institute of Science and Technology
Chennai, India
thiru_vlsi@zoho.com

***Abstract*—Controller area network is the serial communication protocol, which broadcasts the message on the CAN bus. The transmitted message is read by all the nodes which shares the CAN bus. The message can be eavesdropped and can be re-used by some other node by changing the information or send it by duplicate times. The message reused after some delay is replay attack. In this paper, the CAN network with three CAN nodes is implemented using the universal verification components and the replay attack is demonstrated by creating the faulty node. Two types of replay attack are implemented in this paper, one is to replay the entire message and the other one is to replay only the part of the frame. The faulty node uses the first replay attack method where it behaves like the other node in the network by duplicating the identifier. CAN frame except the identifier is reused in the second method which is hard to detect the attack as the faulty node uses its own identifier and duplicates only the data in the CAN frame**

Keywords: Controller Area Network (CAN), Universal Verification Methodology (UVM), Replay attack, Universal Verification Components (UVC), Application Specific Integrated Circuits (ASIC), Verification testbench, CAN Frames.

I. INTRODUCTION

CAN bus is a serial protocol which is commonly used protocol in automotive industry. CAN bus has two different frame formats, they are base frame format[1][2] and extended frame format. Base frame format fields are 1-bit start of frame (SOF), 11-bit identifier field (ID), 1-bit remote transmission request (RTR),1-bit Identifier extension bit (IDE), 4-bit data length code (DLC), 0-8 bytes of data, 15-bit cyclic redundancy check (CRC), 1-bit CRC delimiter, 1-bit acknowledge (ACK), 1-bit ACK delimiter and 7-bits of end of frame(EOF). Extended frame format is same as base frame format except the ID field which is 18bits. Other than the data frames there are other frames such as remote frame which is transmitted from the source node with the request from the sink node, overload frame which tells the source node that the sink is not ready to accept the data and error frame[3] which is sent when there is an error in the frame or when there is an bit stuffing error. The CAN node can transmit two values at the CAN bus which is dominant bit (0) and the recessive bit (1). If the two nodes are transmitting in the CAN bus at the same time, then the bus wins the arbitration transmits the data. Arbitration in the CAN bus is processed by comparing the bit values of both the nodes, the nodes which transmit the recessive bit losses the bus over the node transmits the dominant bit. The gap between two frame is defined by the inter frame gap. Three consecutive recessive bits are transmitted after the frame is identified as inter frame gap in the CAN bus. If six consecutive recessive(1) or dominant(0) bits are transmitted in the CAN bus, then the error frame is sent in the CAN bus to denote the bit stuffing error. So whenever the five consecutive 1/0 bits are transmitted in the CAN bus, node adds the opposite polarity of the last bit. The receiver node decodes this bit stuffing by eliminating the sixth bit followed by five consecutive recessive or dominant bits. The major disadvantage of the CAN bus is the lack of security or authentication[4][5]. The message is broadcasted by the source node in order for it to communicate to a particular node. Every other node including the source node can fetch the data. Thus, there is no message authentication in the CAN protocol. The data is eavesdropped by the nodes which are connected in the CAN bus. The replay attack[6] is the common error scenario in the CAN protocol. There are other serious attacks such as DOS attack[7] and spoof attack faced by CAN. The implementation of the CAN node and CAN network are coded in the universal verification methodology[8][9].

II. CAN NETWORK

Three CAN nodes are connected in the CAN bus to form the CAN network. All the nodes are designed as CAN 1.0 i.e. only the 11 bits identifier of base data format is supported and the remaining bits of the extended identifier are reserved bits. The data is packed in the sequence item based on the CAN frame format. The data generated in the sequencer are driven to the interface by the driver. The driver drives the CAN bus with the generated frame. The next frame is transmitted by the CAN nodes after the Inter frame gap delay. The connections are done using the sequence item port. The transmitted data is latched from the CAN bus by all the nodes. The latched data is compared with the transmitted data by every node to perform the arbitration mechanism. The identifier with the low value wins the arbitration because the node transmits the dominant bit ‘h0 wins the bus over the nodes which sends the recessive bit ‘h1. The base identifier of three nodes A,B and C are ‘h0c2, ‘h1ba and ‘h002 respectively. The highest priority node is C, followed by A, and then by B. Node C wins the arbitration even when all the other nodes transmit at same time. Node A wins the arbitration when it transmits along with node B. Node B transmits the data only if the bus is IDLE. Bit stuffing is implemented in all the nodes. If five consecutive 0s or 1s are transmitted then the sixth bit is the opposite of the previous five bits. The Cyclic redundancy check is done in the CAN frames. The CAN network with the three nodes[10] are shown in figure 1.

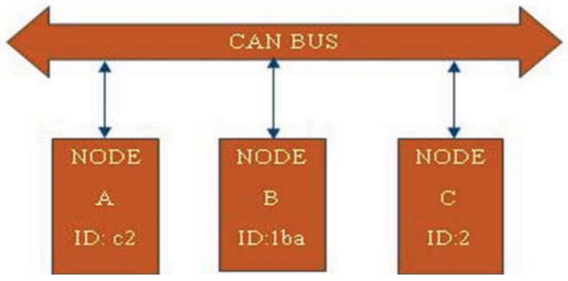
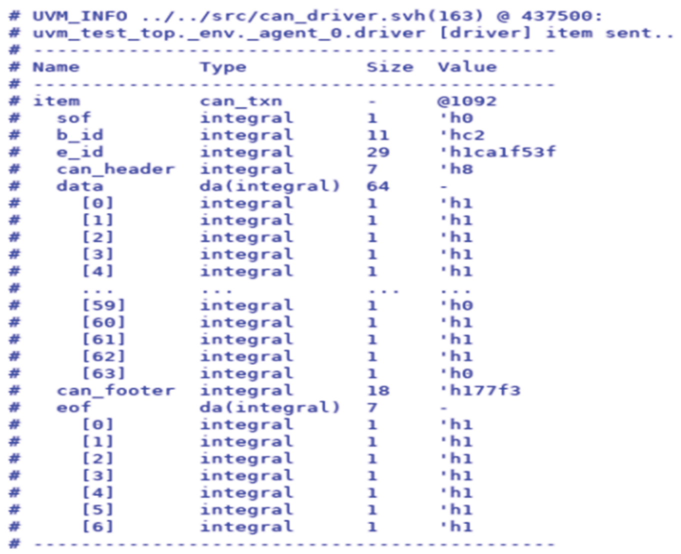


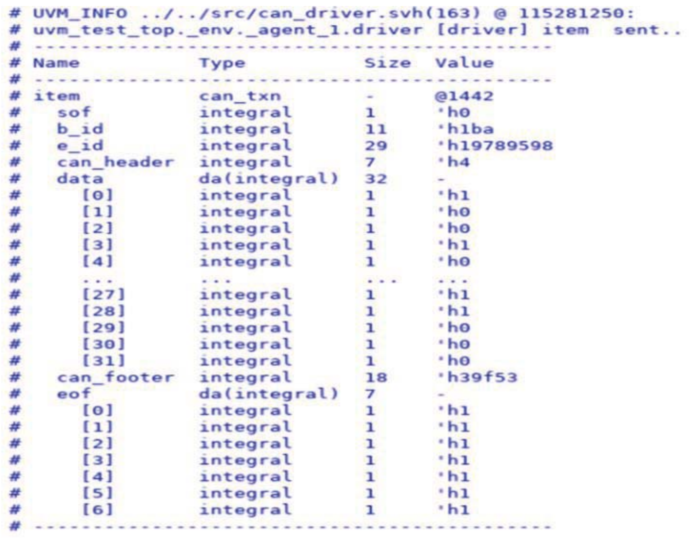
Fig. 1. CAN network with three nodes

The data of the node A, B and C are shown in the Figure 2,3 and 4, respectively. From the below figures, all the nodes generates recessive start of frame. RTR and IDE bits of all the nodes are dominant. CAN header from the below figures includes the RTR, IDE, reserved and DLC fields. CAN node A generates the frame with ID as C2, DLC as 8 and 64 bits of data. Node B generates the frame with ID as 1BA, DLC as 4 and 32 bits of data. Node C generates the frame with ID as 2, DLC as 5 and 40 bits of data. Extended ID field generates the random data in the sequence, but while transmitting from the driver the ID fields overrides with recessive bits. CAN footer from the below figure includes cyclic redundancy check, cyclic redundancy check delimiter, acknowledge slot, and acknowledge delimiter. CRC and ACK for all the nodes are calculated and generated in the sequence item. The end of frame is denoted by generating 7 consecutive recessive bits. CAN frame generated and driven from the CAN driver for 3 agents are 0,1 and 2 which is node A, B and C respectively.



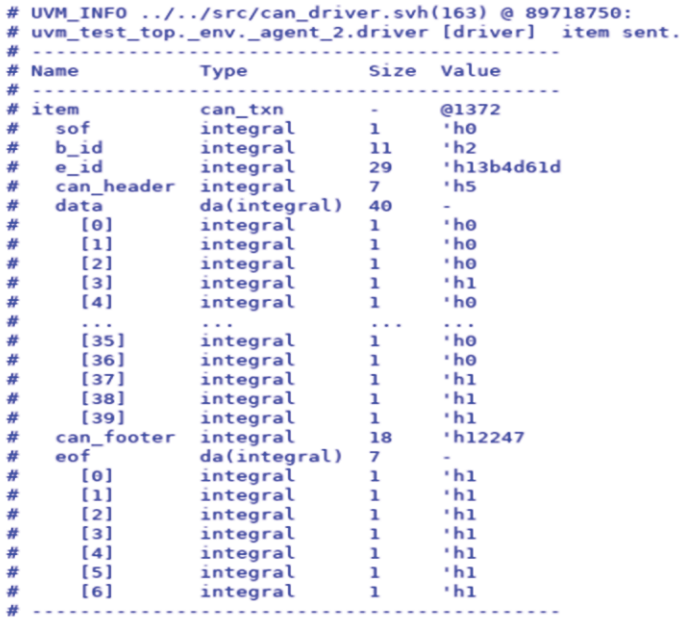
```
# UVM_INFO ../../src/can_driver.svh(163) @ 437500:
# uvm_test_top._env._agent_0.driver [driver] item sent..
# ----------------------------------------------------
# Name              Type             Size  Value
# ----------------------------------------------------
# item              can_txn          -     @1092
#   sof             integral         1     'h0
#   b_id            integral         11    'hc2
#   e_id            integral         29    'h1ca1f53f
#   can_header      integral         7     'h8
#   data            da(integral)     64    -
#     [0]           integral         1     'h1
#     [1]           integral         1     'h1
#     [2]           integral         1     'h1
#     [3]           integral         1     'h1
#     [4]           integral         1     'h1
#     ...           ...              ...   ...
#     [59]          integral         1     'h0
#     [60]          integral         1     'h1
#     [61]          integral         1     'h1
#     [62]          integral         1     'h1
#     [63]          integral         1     'h0
#   can_footer      integral         18    'h177f3
#   eof             da(integral)     7     -
#     [0]           integral         1     'h1
#     [1]           integral         1     'h1
#     [2]           integral         1     'h1
#     [3]           integral         1     'h1
#     [4]           integral         1     'h1
#     [5]           integral         1     'h1
#     [6]           integral         1     'h1
# ----------------------------------------------------
```

Fig. 2. Data of node A



```
# UVM_INFO ../../src/can_driver.svh(163) @ 115281250:
# uvm_test_top._env._agent_1.driver [driver] item  sent..
# ----------------------------------------------------
# Name              Type             Size  Value
# ----------------------------------------------------
# item              can_txn          -     @1442
#   sof             integral         1     'h0
#   b_id            integral         11    'h1ba
#   e_id            integral         29    'h19789598
#   can_header      integral         7     'h4
#   data            da(integral)     32    -
#     [0]           integral         1     'h1
#     [1]           integral         1     'h0
#     [2]           integral         1     'h0
#     [3]           integral         1     'h1
#     [4]           integral         1     'h0
#     ...           ...              ...   ...
#     [27]          integral         1     'h1
#     [28]          integral         1     'h1
#     [29]          integral         1     'h0
#     [30]          integral         1     'h0
#     [31]          integral         1     'h0
#   can_footer      integral         18    'h39f53
#   eof             da(integral)     7     -
#     [0]           integral         1     'h1
#     [1]           integral         1     'h1
#     [2]           integral         1     'h1
#     [3]           integral         1     'h1
#     [4]           integral         1     'h1
#     [5]           integral         1     'h1
#     [6]           integral         1     'h1
# ----------------------------------------------------
```

Fig. 3. Data of node B



```
# UVM_INFO ../../src/can_driver.svh(163) @ 89718750:
# uvm_test_top._env._agent_2.driver [driver]  item sent.
# ----------------------------------------------------
# Name              Type             Size  Value
# ----------------------------------------------------
# item              can_txn          -     @1372
#   sof             integral         1     'h0
#   b_id            integral         11    'h2
#   e_id            integral         29    'h13b4d61d
#   can_header      integral         7     'h5
#   data            da(integral)     40    -
#     [0]           integral         1     'h0
#     [1]           integral         1     'h0
#     [2]           integral         1     'h0
#     [3]           integral         1     'h1
#     [4]           integral         1     'h0
#     ...           ...              ...   ...
#     [35]          integral         1     'h0
#     [36]          integral         1     'h0
#     [37]          integral         1     'h1
#     [38]          integral         1     'h1
#     [39]          integral         1     'h1
#   can_footer      integral         18    'h12247
#   eof             da(integral)     7     -
#     [0]           integral         1     'h1
#     [1]           integral         1     'h1
#     [2]           integral         1     'h1
#     [3]           integral         1     'h1
#     [4]           integral         1     'h1
#     [5]           integral         1     'h1
#     [6]           integral         1     'h1
# ----------------------------------------------------
```

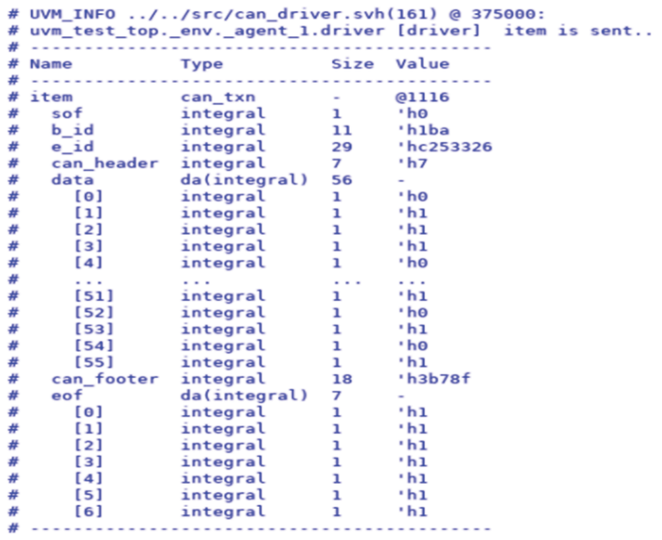
Fig. 4. Data of node C

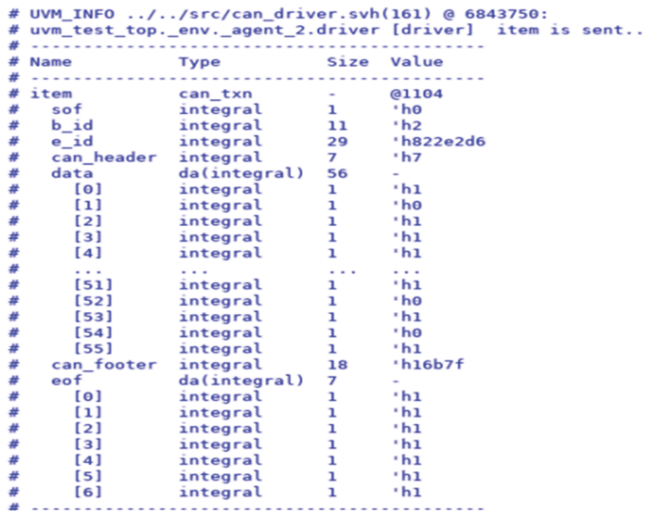
III. REPLAY ATTACK

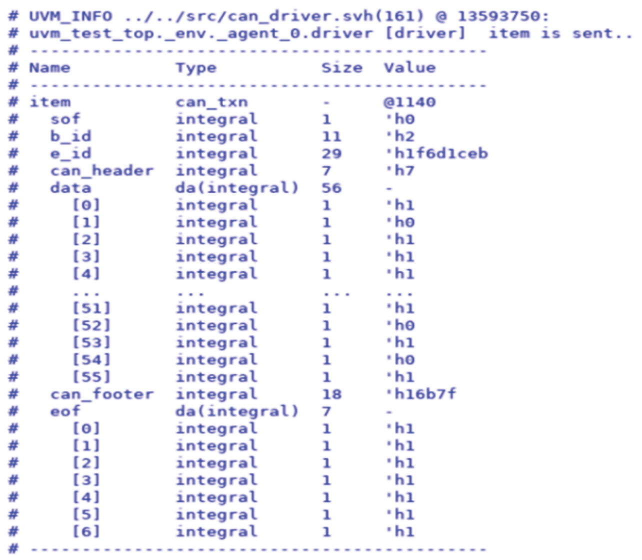
The replay attack[11][12] is the process of transmitting the received data from other CAN nodes in the bus. In this paper, the received data is transmitted in two ways. In the first method, the entire frame is transmitted in the bus[13] and in the second method the data is transmitted excluding the ID field.

In the first method, CAN node C generates the data and transmitted to the CAN bus. Node B and A received the message. Node B uses the message whereas node A stores the CAN frame internally in order to spoof it later[14]. After certain amount of delay when the CAN bus is IDLE, node A transmits the saved message from node C. Now the C and B receives the faulty data from node A. Since the CAN bus is used in automobile industry. This replay attack can cause serious damages. Node C latches the data in the driver or monitor. Node C uses the unpack function in the sequence item to unpack the frame and stores the values in their respective field. Node C compares the ID value of the received frame with its own ID. If node C ID matches with the received frame and if node C is not transmitting, then the node C detects the replay attack and sends the error frame. If the node C ID matches with the received frame and the node C is transmitting data, then the arbitration process starts. If the node C wins the arbitration then the node C frame is transmitted and replayed data is not transmitted in the CAN bus. If the node C loses its arbitration, then the replayed message is transmitted in the CAN bus. Node C checks the received frame ID with its own and compares the data with received frame, if the ID matches and data mismatches, then the replay attack is identified and error frame is transmitted from the node C. Checker logic is included in the driver or the monitor of the CAN agent to detect the replay attack. Error frame generation logic is done in CAN sequence item and sequencer. The generated error frame from sequencer is sent to the driver via the sequence item port or to the monitor via the analysis port. The error frame is sent to the CAN bus through the CAN interface. The error frame is six consecutive dominant or recessive bits followed by the eight recessive bits. So the frame generation can be done directly in the driver instead of the sequencer.

The transmitted frames of node B and C are shown in Figure 5 and 6. The replayed frame of A is shown in the Figure 7. The waveform of CAN network is represented in Figure 8. The waveform shows the three nodes data separately and the line signal as CAN bus. The agent_0, agent_1 and agent_2 in the log file represents the node A,B and C. The frame transmitted from the driver_0 has the base identifier value as ‘h2, but the ID of the agent_0 is ‘hc2. It is evident that the node A has reused the frame of node C which is already sent in the CAN bus. But the node C can detect the attack by comparing the transmitted identifier with its own identifier. Thus, in this method, attack can be detected by one of the sink nodes which match the identifier of the replayed message.


```
# UVM_INFO ../../src/can_driver.svh(161) @ 375000:
# uvm_test_top._env._agent_1.driver [driver]  item is sent..
# ------------------------------------------------------
# Name            Type            Size  Value
# ------------------------------------------------------
# item            can_txn         -     @1116
#   sof           integral        1     'h0
#   b_id          integral        11    'h1ba
#   e_id          integral        29    'hc253326
#   can_header    integral        7     'h7
#   data          da(integral)    56    -
#     [0]         integral        1     'h0
#     [1]         integral        1     'h1
#     [2]         integral        1     'h1
#     [3]         integral        1     'h1
#     [4]         integral        1     'h0
#     ...         ...             ...   ...
#     [51]        integral        1     'h1
#     [52]        integral        1     'h0
#     [53]        integral        1     'h1
#     [54]        integral        1     'h0
#     [55]        integral        1     'h1
#   can_footer    integral        18    'h3b78f
#   eof           da(integral)    7     -
#     [0]         integral        1     'h1
#     [1]         integral        1     'h1
#     [2]         integral        1     'h1
#     [3]         integral        1     'h1
#     [4]         integral        1     'h1
#     [5]         integral        1     'h1
#     [6]         integral        1     'h1
# ------------------------------------------------------
```
Fig. 5. Transmitted frame of node B


```
# UVM_INFO ../../src/can_driver.svh(161) @ 6843750:
# uvm_test_top._env._agent_2.driver [driver]  item is sent..
# ------------------------------------------------------
# Name            Type            Size  Value
# ------------------------------------------------------
# item            can_txn         -     @1104
#   sof           integral        1     'h0
#   b_id          integral        11    'h2
#   e_id          integral        29    'h822e2d6
#   can_header    integral        7     'h7
#   data          da(integral)    56    -
#     [0]         integral        1     'h1
#     [1]         integral        1     'h0
#     [2]         integral        1     'h1
#     [3]         integral        1     'h1
#     [4]         integral        1     'h1
#     ...         ...             ...   ...
#     [51]        integral        1     'h1
#     [52]        integral        1     'h0
#     [53]        integral        1     'h1
#     [54]        integral        1     'h0
#     [55]        integral        1     'h1
#   can_footer    integral        18    'h16b7f
#   eof           da(integral)    7     -
#     [0]         integral        1     'h1
#     [1]         integral        1     'h1
#     [2]         integral        1     'h1
#     [3]         integral        1     'h1
#     [4]         integral        1     'h1
#     [5]         integral        1     'h1
#     [6]         integral        1     'h1
# ------------------------------------------------------
```
Fig. 6. Transmitted frame of node C


```
# UVM_INFO ../../src/can_driver.svh(161) @ 13593750:
# uvm_test_top._env._agent_0.driver [driver]  item is sent..
# ------------------------------------------------------
# Name            Type            Size  Value
# ------------------------------------------------------
# item            can_txn         -     @1140
#   sof           integral        1     'h0
#   b_id          integral        11    'h2
#   e_id          integral        29    'h1f6d1ceb
#   can_header    integral        7     'h7
#   data          da(integral)    56    -
#     [0]         integral        1     'h1
#     [1]         integral        1     'h0
#     [2]         integral        1     'h1
#     [3]         integral        1     'h1
#     [4]         integral        1     'h1
#     ...         ...             ...   ...
#     [51]        integral        1     'h1
#     [52]        integral        1     'h0
#     [53]        integral        1     'h1
#     [54]        integral        1     'h0
#     [55]        integral        1     'h1
#   can_footer    integral        18    'h16b7f
#   eof           da(integral)    7     -
#     [0]         integral        1     'h1
#     [1]         integral        1     'h1
#     [2]         integral        1     'h1
#     [3]         integral        1     'h1
#     [4]         integral        1     'h1
#     [5]         integral        1     'h1
#     [6]         integral        1     'h1
# ------------------------------------------------------
```
Fig. 7. Replay attack frame of node A

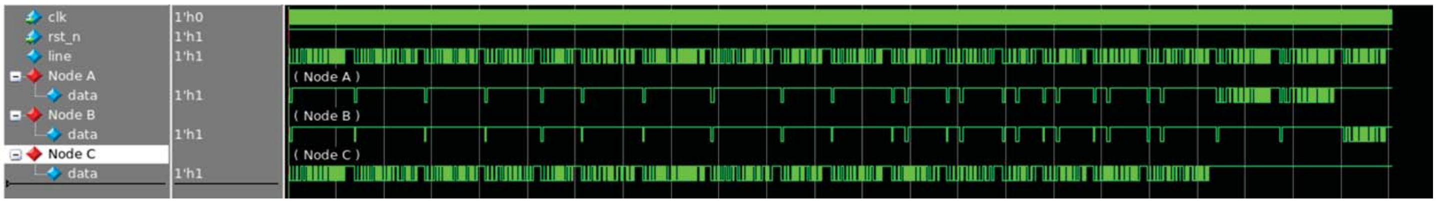
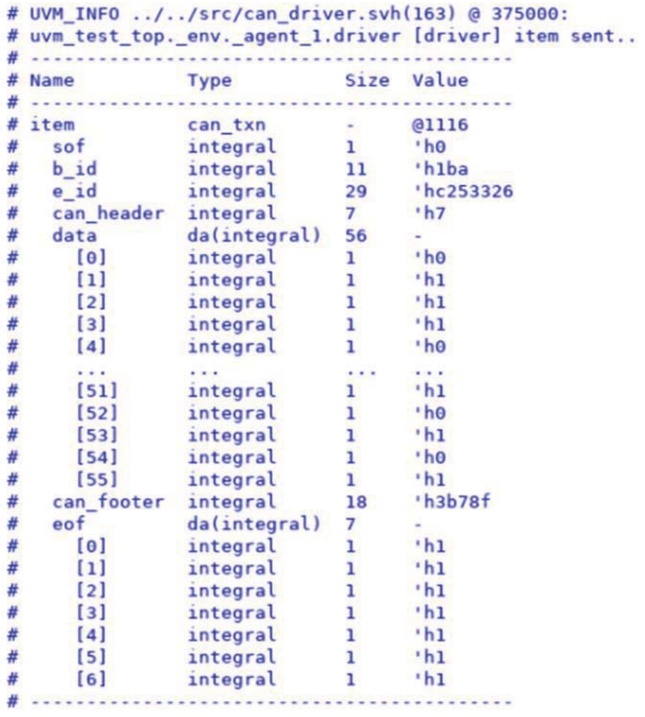


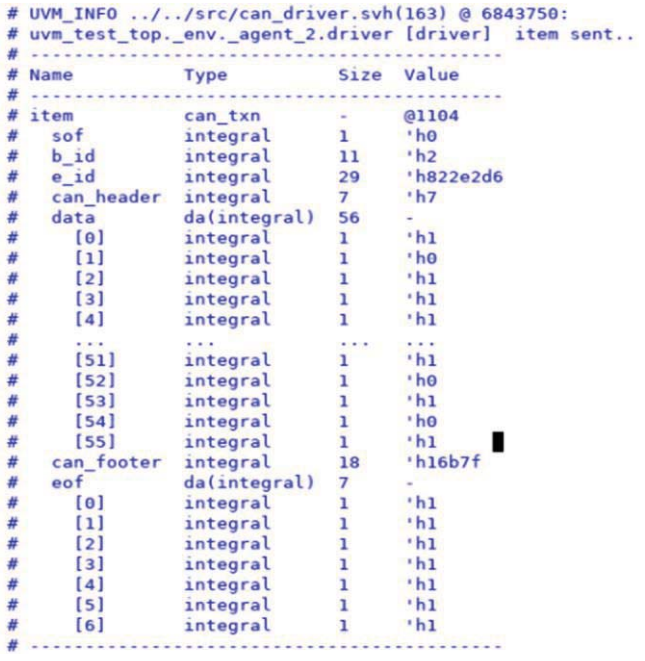
Fig. 8. CAN network waveform

The whole data is not re-used in the second scenario; only parts of the frames are re-used. The data after the identifier are used for the replay attack. Node A receives the node C frames and stores it internally. It replaces the C identifier with A identifier and transmits the data after some time. Now the node B and C received the data after some delay. Node B has less probability to identify this error. Node C can detect this error by comparing all the fields of the CAN frame which are already sent. But there may be a possibility that the node A data accidentally matches with the replayed data. Large numbers of buffers are needed to store the generated data and needs to be compared with the received data from other nodes. So the best and most common method to avoid replay attack is by adding counter value[15] or timestamp to the data. The frames of B,C and the replay attack frame of node A is shown in the Figure 9,10 and 11 respectively. Now the frames transmitted from the A has the ID as ‘hc2 but the rest of the data are similar with the node C. It is difficult to identify this error, because the identifier belongs to the source node. Thus, it is impossible for the other nodes to identify the attack. The workflow of partial frame and full frame replay attack identification method is shown in Figure 12 and 13 respectively.



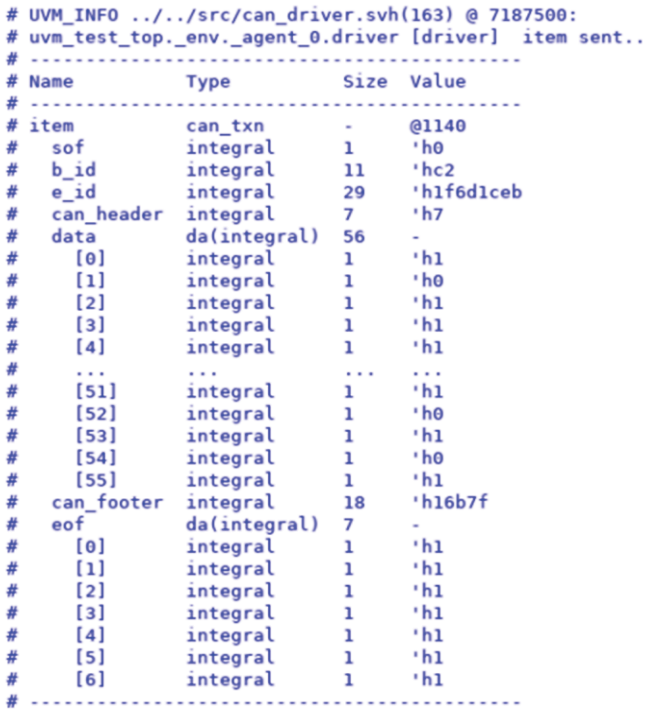
```
# UVM_INFO ../../src/can_driver.svh(163) @ 375000:
# uvm_test_top._env._agent_1.driver [driver] item sent..
# -------------------------------------------
# Name           Type           Size  Value
# -------------------------------------------
# item           can_txn        -     @1116
#   sof          integral       1     'h0
#   b_id         integral       11    'h1ba
#   e_id         integral       29    'hc253326
#   can_header   integral       7     'h7
#   data         da(integral)   56    -
#     [0]        integral       1     'h0
#     [1]        integral       1     'h1
#     [2]        integral       1     'h1
#     [3]        integral       1     'h1
#     [4]        integral       1     'h0
#     ...        ...            ...   ...
#     [51]       integral       1     'h1
#     [52]       integral       1     'h0
#     [53]       integral       1     'h1
#     [54]       integral       1     'h0
#     [55]       integral       1     'h1
#   can_footer   integral       18    'h3b78f
#   eof          da(integral)   7     -
#     [0]        integral       1     'h1
#     [1]        integral       1     'h1
#     [2]        integral       1     'h1
#     [3]        integral       1     'h1
#     [4]        integral       1     'h1
#     [5]        integral       1     'h1
#     [6]        integral       1     'h1
# -------------------------------------------
```

Fig. 9. Frame of node B



```
# UVM_INFO ../../src/can_driver.svh(163) @ 6843750:
# uvm_test_top._env._agent_2.driver [driver]  item sent..
# -------------------------------------------
# Name           Type           Size  Value
# -------------------------------------------
# item           can_txn        -     @1104
#   sof          integral       1     'h0
#   b_id         integral       11    'h2
#   e_id         integral       29    'h822e2d6
#   can_header   integral       7     'h7
#   data         da(integral)   56    -
#     [0]        integral       1     'h1
#     [1]        integral       1     'h0
#     [2]        integral       1     'h1
#     [3]        integral       1     'h1
#     [4]        integral       1     'h1
#     ...        ...            ...   ...
#     [51]       integral       1     'h1
#     [52]       integral       1     'h0
#     [53]       integral       1     'h1
#     [54]       integral       1     'h0
#     [55]       integral       1     'h1
#   can_footer   integral       18    'h16b7f
#   eof          da(integral)   7     -
#     [0]        integral       1     'h1
#     [1]        integral       1     'h1
#     [2]        integral       1     'h1
#     [3]        integral       1     'h1
#     [4]        integral       1     'h1
#     [5]        integral       1     'h1
#     [6]        integral       1     'h1
# -------------------------------------------
```

Fig. 10. Frame of node C



```
# UVM_INFO ../../src/can_driver.svh(163) @ 7187500:
# uvm_test_top._env._agent_0.driver [driver]  item sent..
# -------------------------------------------
# Name           Type           Size  Value
# -------------------------------------------
# item           can_txn        -     @1140
#   sof          integral       1     'h0
#   b_id         integral       11    'hc2
#   e_id         integral       29    'h1f6d1ceb
#   can_header   integral       7     'h7
#   data         da(integral)   56    -
#     [0]        integral       1     'h1
#     [1]        integral       1     'h0
#     [2]        integral       1     'h1
#     [3]        integral       1     'h1
#     [4]        integral       1     'h1
#     ...        ...            ...   ...
#     [51]       integral       1     'h1
#     [52]       integral       1     'h0
#     [53]       integral       1     'h1
#     [54]       integral       1     'h0
#     [55]       integral       1     'h1
#   can_footer   integral       18    'h16b7f
#   eof          da(integral)   7     -
#     [0]        integral       1     'h1
#     [1]        integral       1     'h1
#     [2]        integral       1     'h1
#     [3]        integral       1     'h1
#     [4]        integral       1     'h1
#     [5]        integral       1     'h1
#     [6]        integral       1     'h1
# -------------------------------------------
```

Fig. 11. Replay attack frame of node A

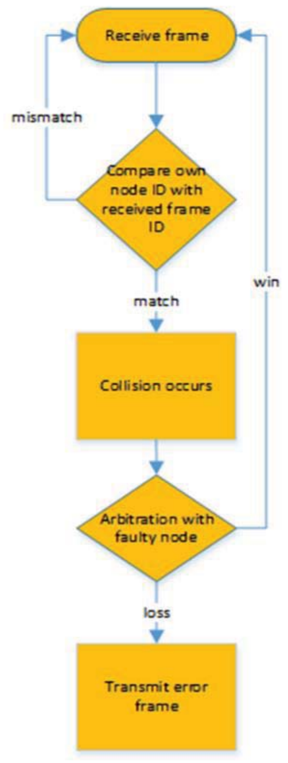

Receive frame
mismatch
Compare own node ID with received frame ID
match
win
Collision occurs
Arbitration with faulty node
loss
Transmit error frame

Fig. 12. Workflow of partial frame replay attack identification method

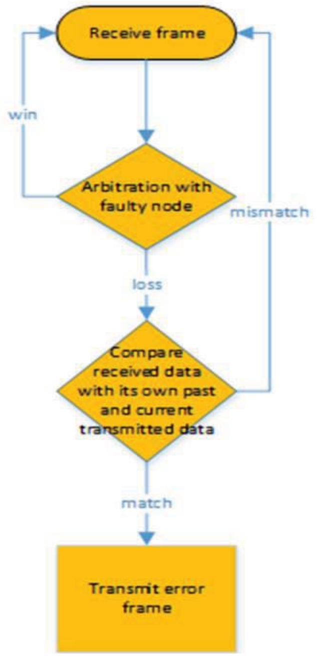

Receive frame
win
Arbitration with faulty node
mismatch
loss
Compare received data with its own past and current transmitted data
match
Transmit error frame

Fig. 13. Workflow of full frame replay attack identification method

IV. CONCLUSIONS AND FUTURE WORK

The replay attack is the major threat for the broadcast-based protocol. Most of the research papers implement post-silicon[15], but this paper implemented the replay attack in the pre-silicon phase. If the whole data is replayed then the ID gets compared and the replay attack is identified. If the part of the frame is replayed then data needs to be compared. The major limitation of this method is that each CAN node needs more buffers to store the transmitted data. Future work is to eliminate the risk by sending the timestamp or counter along with the data. If the timestamp matches with system time, then the message is the authenticated one. If the timestamp mismatches with the system time, then the data transmitted is already sent.

REFERENCES

[1] O. Cros and G. Chênevert, "Hashing-based authentication for CAN bus and application to Denial-of-Service protection," 2019 3rd Cyber Security in Networking Conference (CSNet), Quito, Ecuador, 2019, pp. 91-98, doi: 10.1109/CSNet47905.2019.9108978.

[2] S. Fassak, Y. El Hajjaji El Idrissi, N. Zahid and M. Jedra, "A secure protocol for session keys establishment between ECUs in the CAN bus," *2017 International Conference on Wireless Networks and Mobile Communications (WINCOM),* Rabat, 2017, pp. 1-6

[3] H. Giannopoulos, A. M. Wyglinski and J. Chapman, "Securing vehicular controller area networks: An approach to active bus-level countermeasures", *IEEE Veh. Technol. Mag.*, vol. 12, no. 4, pp. 60-68, Dec. 2017.

[4] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CanAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN Bus" in 9th Embedded Security in Cars Conf, Dresden, Germany, 2011.

[5] S. Araki, A. Tashiro, K. Kakizaki and S. Uehara, "A study on a secure protocol against tampering and replay attacks focused on data field of CAN," 2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW), Taipei, 2017, pp. 247-248, doi: 10.1109/ICCE-China.2017.7991088.

[6] M. R. Ansari, W. T. Miller, C. She and Q. Yu, "A low-cost masquerade and replay attack detection method for CAN in automobiles," *2017 IEEE International Symposium on Circuits and Systems (ISCAS),* Baltimore, MD, 2017, pp. 1-4

[7] W. A. Farag, "Cantrack: Enhancing automotive can bus security using intuitive encryption algorithms," in Modeling, Simulation, and Applied Optimization (ICMSAO), 2017 7th International Conference on. IEEE, 2017, pp. 1–5

[8] G. Sharma, L. Bhargava and V. Kumar, "Self-Assertive Generic UVM Testbench for Advanced Verification of Bridge IPs", *2017 14th IEEE India Council International Conference (INDICON)*, pp. 1-6, 2017.

[9] M. B. R. Srinivas and Sarada Musala, "Verification of AHB_LITE protocol for waited transfer responses using re-usable verification methodology" **,** Inventive Computation Technologies (ICICT), 2016**.**

[10] A. S. Siddiqui, Y. Gui, J. Plusquellic and F. Saqib, "Secure communication over CANBus," *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boston, MA, 2017, pp. 1264-1267.

[11] Z. King and S. Yu, "Investigating and securing communications in the controller area network (can)," in Computing, Networking and Communications (ICNC), 2017 International Conference on. IEEE, 2017, pp. 814–818.

[12] P. Noureldeen, M. A. Azer, A. Refaat and M. Alam, "Replay attack on lightweight CAN authentication protocol," 2017 12th International Conference on Computer Engineering and Systems (ICCES), Cairo, 2017, pp. 600-606, doi: 10.1109/ICCES.2017.8275376.

[13] S. Katragadda, P. J. Darby, A. Roche and R. Gottumukkala, "Detecting Low-Rate Replay-Based Injection Attacks on In-Vehicle Networks," in IEEE Access, vol. 8, pp. 54979-54993, 2020, doi: 10.1109/ACCESS.2020.2980523.

[14] S. Ohira, A. K. Desta, I. Arai, H. Inoue and K. Fujikawa, "Normal and Malicious Sliding Windows Similarity Analysis Method for Fast and Accurate IDS Against DoS Attacks on In-Vehicle Networks," in *IEEE Access*, vol. 8, pp. 42422-42435, 2020, doi: 10.1109/ACCESS.2020.2975893.

[15] K. Nishida, Y. Nozaki and M. Yoshikawa, "Security Evaluation of Counter Synchronization Method for CAN Against DoS Attack," *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, Osaka, Japan, 2019, pp. 166-167, doi: 10.1109/GCCE46687.2019.9015397.