CAN Demonstrator with Intrusion Detection System

Simulation Environment for Carrying Out and Analysing Hacker Attacks with a CAN Bus Demonstrator

Dirk Labudde and Heiko Polster
Forensic Science Investigation Lab
Hochschule Mittweida – University of Applied Sciences Mittweida, Germany
Fraunhofer Institute for Secure Information Technology, Germany

Email: dirk.labudde@hs-mittweida.de, heiko.polster@hs-mittweida.de

Abstract — **The paper describes a digital simulation environment to reproduce and test attacks on Controller Area Network (CAN) bus systems with a CAN bus demonstrator. It describes how the CAN bus demonstrator is structured and used to analyze and evaluate hacker attacks on automotive bus systems. Security researchers repeatedly find vulnerabilities in various software components of networked vehicles. Since investigations on the real system seem too costly, various attacks on a CAN bus system are to be tested and analyzed with the help of a CAN bus demonstrator. The system consists of a CAN bus demonstrator in the form of a model car and the development environment Vector CANoe. The development environment is connected to the model via two NodeMCU-ESP32 development boards, which are configured as WLAN CAN gateways, and a VN1610 CAN-USB interface. The CANoe tool is used to control the CAN bus demonstrator, simulate CAN bus attacks and analyze them using the intrusion detection system. The simulation environment shows that the CAN demonstrator is an effective method for imparting theoretical knowledge and solving practical tasks in the field of car forensic training.**

Keywords - cyberattacks; car forensics; can bus; demonstrator; intrusion detection system.

I. Introduction

The digital transformation has transformed the power vehicle from an isolated means of transport to a system that communicates incessantly with its environment. Thanks to the networking of vehicle functions (drive, steering, braking, comfort and infotainment) by bus systems and the communication of the vehicle with its surroundings via mobile communications, emergency call (eCall) or Car2X, vehicles today have become more efficient, more comfortable and safer. Today, a modern vehicle consists of approximately 100 million lines of program code [1], distributed over at least 50 installed Electronic Control Units (ECU). The statistics from [2] show a forecasted share of connected cars in the USA, China and the European Union up to the year 2035. Some 72% of motor vehicles in the European Union will be connected by 2035.

Reference [3] shows that by 2023 it is estimated that there will be worldwide 775 million vehicles connected via in-vehicle telematics or a smartphone app. This rapidly increasing number of connected vehicles, coupled with the constant expansion of new interfaces with the environment, opens the possibility for unauthorized persons to compromise the system of the connected vehicle.

Safety researchers repeatedly find weaknesses in the numerous software components of networked vehicles. For example, most software errors in mobile communications connections and in tracking or emergency call systems that can be detected allow infiltration into infotainment systems to subsequently compromise the control electronics. In some cases, however, it is necessary for the functionality of the comfort applications in the vehicle to send vehicle data to the infotainment system. Here, it is important to prevent the flow of data in the other direction to avoid a compromise of the control electronics [4] - [6].

This work deals with an environment to reproduce and test attacks on Controller Area Network (CAN) bus systems using the Vector CANoe tool [18] and a CAN bus demonstrator. It describes how a CAN bus demonstrator is structured and shows how the demonstrator hardware can be controlled using the Vector CANoe tool. In addition, an implementation of an intrusion detection system is shown, with which CAN bus attacks generated in the CANoe tool can be analyzed.

Due to cost reasons and the dangers of tampering with vehicle systems during real use, a CAN bus demonstrator offers a useful alternative for performing and analyzing hacking attacks on automotive bus systems. The demonstrator provides a safe way to analyze what-if scenarios.

The content of the paper is structured as follows. Section II describes the hardware components and the structure of the CAN bus demonstrator. Section III deals with the software-side control of the CAN bus demonstrator using the CANoe tool from Vector. Section IV describes the implementation of an intrusion detection system. Finally, Section V summarizes the results achieved.

II. DEVELOPMENT OF THE CAN BUS DEMONSTRATOR

The mechanical basis of the CAN bus demonstrator is Joy-It's Robot Car Kit 4WD (Figure 1), which is equipped with four electric motors [8].

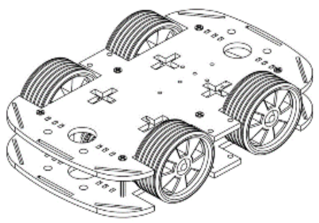


Figure 1. Schematic illustration Robot Car Kit 4WD by Joy-IT

The electronics of the CAN bus demonstrator are based on the classic basic structure of on-board automotive electronics, which consists of a central gateway and classic electronics, which consists of a central gateway and classic vehicle bus systems such as Powertrain CAN, Comfort CAN, Diagnostic CAN, etc. (Figure 2).

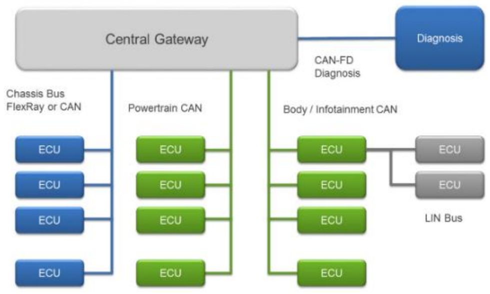


Figure 2. Basic structure of the on-board electronics of a motor vehicle - [7]

For the CAN bus demonstrator, a CAN bus system was designed (Figure 3), which includes a central gateway, a motor control unit, a control unit for lighting and a control unit for other electronic components (distance measurement, battery monitoring and horn).

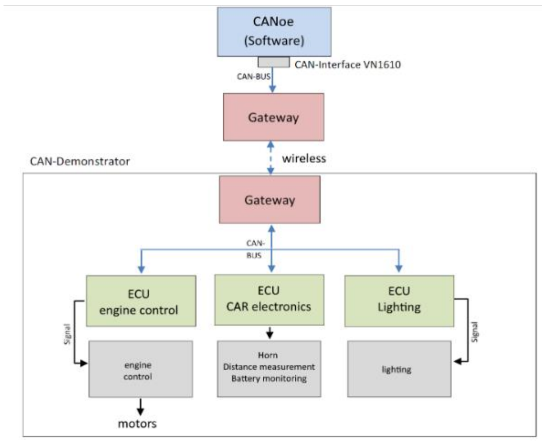


Figure 3. Block diagram of the CAN bus demonstrator with connection to the Vector CANoe tool

The CAN Bus Demonstrator is connected to the Vector CANoe tool via a wireless connection and a Vector CAN interface VN1610.

Motor electronic control unit

In addition to an Arduino Nano [9], the engine driver TB6612 [10] from Adafruit is used to control the engines (Figure 4).

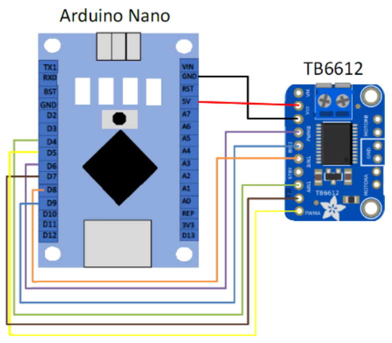


Figure 4. Sketch of the physical connection between the Arduino Nano and motor driver for motor control – adapted from [11]

This breakout board receives the corresponding control information to control the engines via a physical connection via the ATmega328P-AU implemented on the Arduino Nano. The speed regulation is realized by means of pulse width modulation.

Car electronic control unit

The demonstrator has a total of 10 Light-Emitting Diodes (LED). At the front are four LEDs, two each for the turn signals (1, 4) and headlights (2, 3) which can be adjusted in brightness by means of Pulse Width Modulation (PWM) depending on the functionality (dipped beam or main beam). Another four LEDs are located at the back. Two for the indicators (5, 10), two for the tail lamp (6, 9) and one each for the rear-end lamp (8) and the rear fog lamp (7). Equivalent to the headlights, the taillights are controlled by PWM depending on the function (dipped beam or brake beam) in brightness.

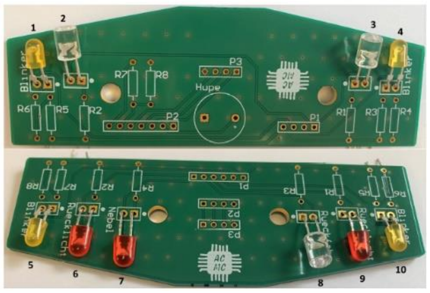


Figure 5. Overview of the implemented lighting elements

The LEDs are supplied by the corresponding Arduino Nano and are regulated via pre-resistors. Printed Circuit Boards (PCB) at the front and rear are used as a mounting solution (Figure 5).

The 2-pin speaker connector SUMMER AL-60P12 [12] is used to imitate the horn. This is regulated by the electronic control unit (Arduino Nano). Power is also supplied by the Arduino Nano.

The distance measurement is realized by means of ultrasonic sensors HC-SR04 [13]. Measurements are only taken in the direction of travel. From a distance of 100 cm to an object, CAN messages containing the distance are sent to the CAN bus at intervals of one second. If the distance decreases, the transmission interval increases. Beginning at a distance of 50 cm to the object, the message is sent every 30 seconds and at 30 cm every 200 ms. At standstill, no distance measurement takes place. The two ultrasonic sensors were soldered to the designed circuit boards at the front and rear. They belong to the electronic control unit that also supplies them with power.

To monitor the charge state of the accumulator in the CAN demonstrator, the voltage is connected to the electronics ECU (Arduino Nano) via an analog-to-digital converter. When a certain threshold is reached, a message is sent, the demonstrator starts to honk and stops. Because the maximum of the battery voltage is 8. 4 V, but the Arduino Nano only has an operating voltage of 3. 3 V, a voltage divider is used.

Gateway electronic control unit

The central gateway in the demonstrator is a NodeMCU-ESP32 [14]. For optimal use of the CAN, a high-speed CAN transceiver VP230 [15] is connected to the NodeMCU-ESP32. Since the VP230 cannot be conveniently attached to the NodeMCU-ESP32 and a handy solution had to be found especially for the external gateway, a compact board14 was designed for the connection of the mentioned gateway to the CAN analysis software (Figure 6), which in addition to a slot for the NodeMCU-ESP32 (also) includes an RS-232 interface Connection point according to CAN in automation Draft Standard 102 (CiA/DS 102) [16] for the connection of the CAN interface VN1610.

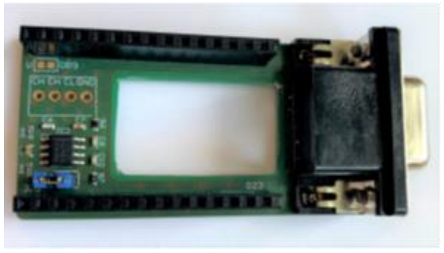


Figure 6. Circuit board of the VP230 on the client side

A small board was also designed for the transceiver in the demonstrator (Figure 7), which contains the VP230 and a three-pin terminal for CAN-High, CAN-Low and ground.

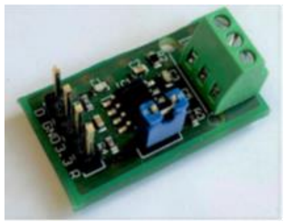


Figure 7. Circuit board of the VP230 on the server side

Power supply

For the ideal power supply, an optimal power circuit diagram was developed, considering the consumption values of the individual components and a sensible arrangement (Figure 8). Modules connected directly to the accumulator have an integrated voltage regulator.

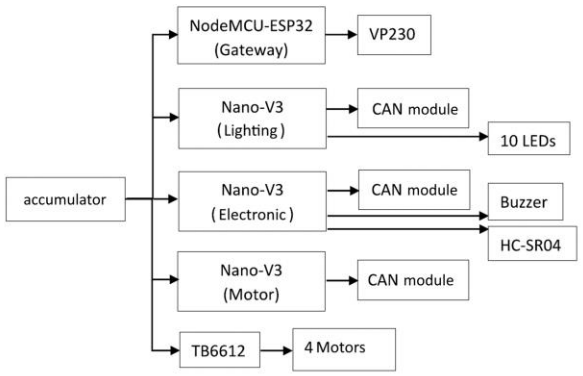


Figure 8. Overview of the power supply in the CAN demonstrator

To regulate the power supply and to be able to switch the CAN demonstrator on and off, a switch was integrated into the experimental setup.

Final assembly of the CAN bus demonstrator

In the final step, all previously described hardware components were attached to the Robot Car Kit. Figure 9 shows the final test setup.

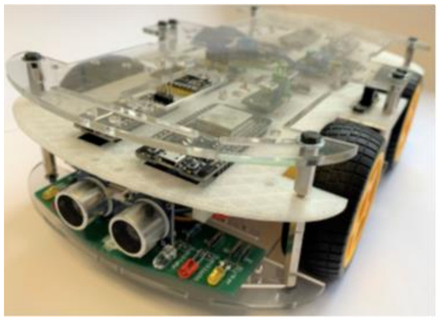


Figure 9. Final structure of the CAN demonstrator

III. CONTROL OF THE CAN BUS DEMONSTRATOR

When developing new control units for motor vehicles, it is possible to simulate CAN bus systems using professional simulation environments. The tool CANoe of the company Vector is used for this purpose. This tool allows virtual ECUs to be tested in real CAN bus environments via a hardware interface, e. g., Vector VN1610. Within the environment, all valid CAN messages for the respective CAN system are stored in a database. These messages can be sent and received in the simulation using interactive generators or virtual ECUs. The virtual ECUs are equipped with their special functionality using the Communication Access Programming Language (CAPL). This type of programming is event oriented. The control of the environment is realized with panels in which corresponding controls such as buttons, switches, visual displays etc., are to be deployed. These controls can be linked to system variables or message content. Extensive analysis tools are available within the environment.

A CANoe project was created for controlling the CAN bus demonstrator (Figure 10).

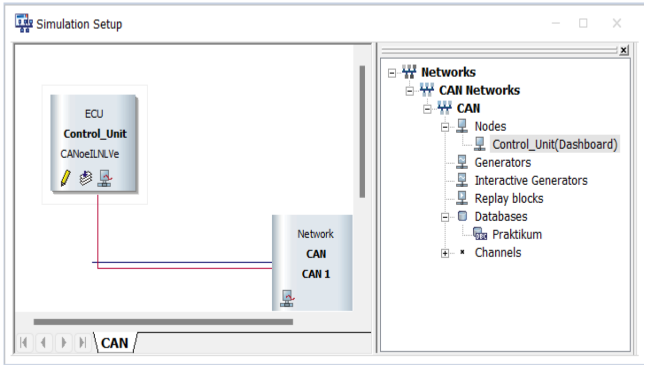
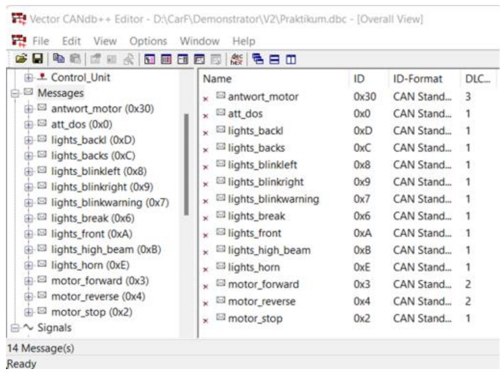


Figure 10. Simulation setup of the CANoe project

The setup of the CANoe project is limited to a virtual ECU. This is linked to a CAPL file in which the functions for the output of the CAN messages are implemented. CAPL is a procedural, C-like programming language developed by Vector. Execution is controlled by program blocks by events. CAPL programs are developed and compiled with their own browser. All objects contained in the database (messages, signals, environment variables) and system variables can be accessed. In addition, CAPL offers a variety of predefined functions. The CAN messages are sent from the virtual ECU when the dashboard changes the values of the checkboxes and controls and their associated system variables.

The basis of the CANoe project is the data base in which the CAN messages and signals required to control the CAN bus demonstrator are defined.


Vector CANdb++ Editor - D:\CarF\Demonstrator\V2\Praktikum.dbc - [Overall View]
File Edit View Options Window Help

Control_Unit
Messages
- antwort_motor (0x30)
- att_dos (0x0)
- lights_backl (0xD)
- lights_backs (0xC)
- lights_blinkleft (0x8)
- lights_blinkright (0x9)
- lights_blinkwarning (0x7)
- lights_break (0x6)
- lights_front (0xA)
- lights_high_beam (0xB)
- lights_horn (0xE)
- motor_forward (0x3)
- motor_reverse (0x4)
- motor_stop (0x2)

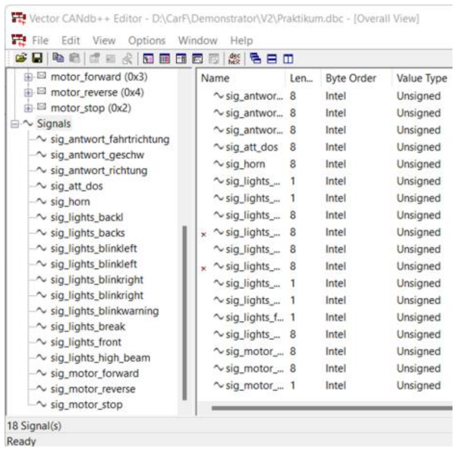
Signals

Name	ID	ID-Format	DLC...
antwort_motor	0x30	CAN Stand...	3
att_dos	0x0	CAN Stand...	1
lights_backl	0xD	CAN Stand...	1
lights_backs	0xC	CAN Stand...	1
lights_blinkleft	0x8	CAN Stand...	1
lights_blinkright	0x9	CAN Stand...	1
lights_blinkwarning	0x7	CAN Stand...	1
lights_break	0x6	CAN Stand...	1
lights_front	0xA	CAN Stand...	1
lights_high_beam	0xB	CAN Stand...	1
lights_horn	0xE	CAN Stand...	1
motor_forward	0x3	CAN Stand...	2
motor_reverse	0x4	CAN Stand...	2
motor_stop	0x2	CAN Stand...	1

14 Message(s)
Ready

Figure 11. CAN messages of the database

Figure 11 shows the database with the 14 implemented CAN messages for the CANoe project. These messages have been assigned the necessary signals (Figure 12).


Vector CANdb++ Editor - D:\CarF\Demonstrator\V2\Praktikum.dbc - [Overall View]
File Edit View Options Window Help

- motor_forward (0x3)
- motor_reverse (0x4)
- motor_stop (0x2)

Signals
- sig_antwort_fahrtrichtung
- sig_antwort_geschw
- sig_antwort_richtung
- sig_att_dos
- sig_horn
- sig_lights_backl
- sig_lights_backs
- sig_lights_blinkleft
- sig_lights_blinkleft
- sig_lights_blinkright
- sig_lights_blinkright
- sig_lights_blinkwarning
- sig_lights_break
- sig_lights_front
- sig_lights_high_beam
- sig_motor_forward
- sig_motor_reverse
- sig_motor_stop

Name	Len...	Byte Order	Value Type
sig_antwor...	8	Intel	Unsigned
sig_antwor...	8	Intel	Unsigned
sig_antwor...	8	Intel	Unsigned
sig_att_dos	8	Intel	Unsigned
sig_horn	8	Intel	Unsigned
sig_lights_...	1	Intel	Unsigned
sig_lights_...	1	Intel	Unsigned
sig_lights_...	8	Intel	Unsigned
sig_lights_...	8	Intel	Unsigned
sig_lights_...	8	Intel	Unsigned
sig_lights_...	8	Intel	Unsigned
sig_lights_...	1	Intel	Unsigned
sig_lights_...	1	Intel	Unsigned
sig_lights_f...	1	Intel	Unsigned
sig_lights_...	8	Intel	Unsigned
sig_motor_...	8	Intel	Unsigned
sig_motor_...	8	Intel	Unsigned
sig_motor_...	1	Intel	Unsigned

18 Signal(s)
Ready

Figure 12. Signals of the Can messages

A dashboard was created using the panel designer built into the CANoe (Figure 13).

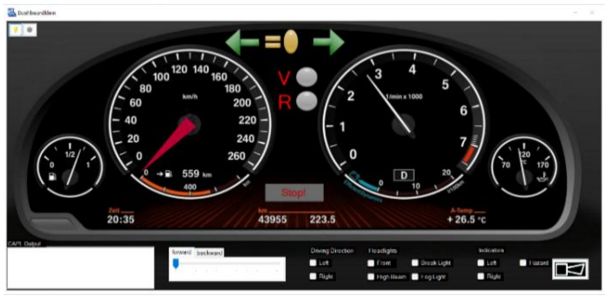


Figure 13. Dashboard for controlling the CAN bus demonstrator

Driving direction and lighting can be controlled via the checkboxes. In addition, the CAN bus demonstrator can be moved forward and backward via the virtual accelerator pedal. The speed is displayed on the dashboard in the speedometer.

IV. IMPLEMENTATION OF THE INTRUSION DETECTION SYSTEM

Intrusion detection systems are often used to detect and report an attack on a system. However, there is currently no way in vehicles to digitally trace and prove an attack on the CAN bus in digital forensics. Bresch and Salman [17] developed an intrusion detection system that can be integrated into the CANoe and therefore also be tested for the demonstrator. They propose two anomaly-based algorithms based on message cycle time analysis and plausibility analysis of messages (e. g., speed messages). Since the messages for the acceleration of the demonstrator are like those in the Bresch and Salman [17] experiment, the functions of the intrusion detection system only needs to minimally adjusted. In the case of attacks, messages are sent whose payload either contains a too high change of speed transmitted to the actual state or whose frequency deviates from that of normal operation. The Intrusion Detection System covers precisely these two scenarios. Figure 14 shows the basic program sequence of the Intrusion Detection System.

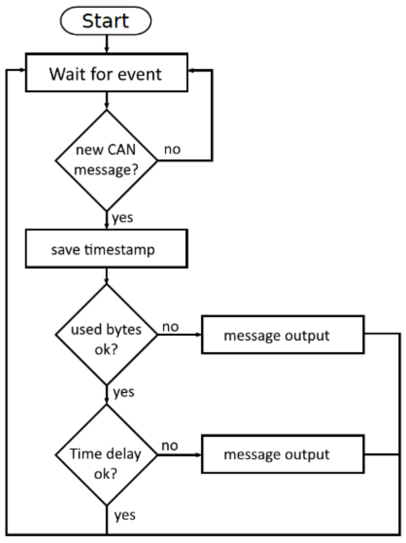


Figure 14. Sketch of the program flow of the intrusion detection system

To find out the threshold value for the message frequency, messages were sent via the panel, see Section III, to simulate an undisturbed ride. The interval between messages never fell below 300 ms/was never under 300 ms, making it a good threshold for the for the time between successive messages. The Intrusion Detection System is implemented in the CAPL file of the virtual ECU.

A panel was created to carry out the CAN bus attacks (Figure 15).

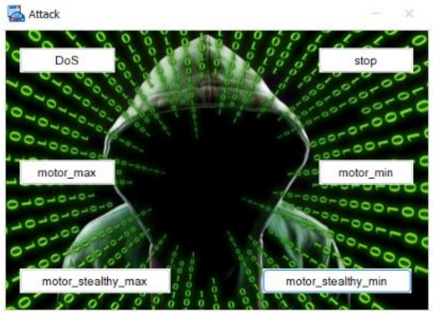


Figure 15. Panel for controlling the CAN bus attacks

The buttons in the panel can be used to trigger the CAN attacks Denial of Service (DoS) implemented in the CANoe project, creeping increase and decrease of speed, maximum and minimum speed, as well as the stopping of the engines.

The DoS attack is implemented in such a way that the highest priority message with the CAN identifier 0x00 is sent 20 milliseconds apart (Figure 16).

```
on sysvar att::dos{
  message 0x0 msg;
  int DoSTime = 20;
  DoSBreak = 1;
  setTimer(DoS,DoSTime);
  while(DoSBreak==1)  output(msg);}
```

Figure 16. CAPL function of the DoS attack

The slow increase or decrease of speed is realized with a system variable and two timers (Figure 17).

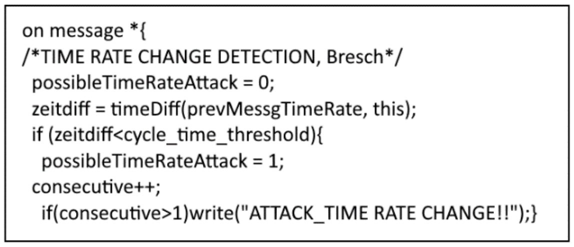
The first step is to evaluate the system variable *motor_stealthy_max*. Subsequently, the direction of travel is evaluated, and a corresponding timer is started depending on it.

```
on sysvar att::motor_stealthy_max{
  activespeed = speed; int time = 80;
 if(reverse) setTimer(timer2, time);
  else setTimer(timer1, time); }
on timer timer1{
   message motor_forward msg;
   if(speed < 255){
    speed ++;
    setTimer(timer1, time);  }
   msg.byte(0) = speed;
   output(msg); }
on timer timer2 {
   message motor_reverse msg;
   if(speed < 255){
    speed ++;
    setTimer(timer2, time); }
   msg.byte(0) = speed;
   output(msg);  }
```

Figure 17. CAPL functions for realizing the creeping increase in speed

After the set 80 milliseconds have elapsed, the variable for speed and the transmission of the corresponding CAN message is increased. Furthermore, the speed is increased to the maximum value.

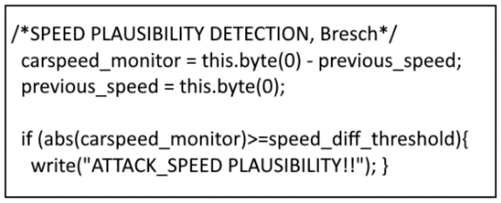
The IDS detect when the interval between sending the CAN message for speed change via a *TIME RATE CHANGE DETECTION* falls short and issues alerts in the CANoe tool console (Figure 18).



```
on message *{
/*TIME RATE CHANGE DETECTION, Bresch*/
  possibleTimeRateAttack = 0;
  zeitdiff = timeDiff(prevMessgTimeRate, this);
  if (zeitdiff<cycle_time_threshold){
    possibleTimeRateAttack = 1;
  consecutive++;
    if(consecutive>1)write("ATTACK_TIME RATE CHANGE!!");}
```

Figure 18. CAPL function for *TIME RATE CHANGE DETECTION* according to [17]

In addition to the *TIME RATE CHANGE DETECTION*, there is a *SPEED PLAUSIBILITY DETECTION* (Figure 19).



```
/*SPEED PLAUSIBILITY DETECTION, Bresch*/
  carspeed_monitor = this.byte(0) - previous_speed;
  previous_speed = this.byte(0);

  if (abs(carspeed_monitor)>=speed_diff_threshold){
    write("ATTACK_SPEED PLAUSIBILITY!!"); }
```

Figure 19. CAPL evaluation of *SPEED PLAUSIBILITY DETECTION* according to [17]

This information is also displayed in the console of the CANoe tool.

V. CONCLUSION

In this paper, we showed that a real CAN bus demonstrator can be recreated using the hardware presented. It is possible to carry out cyber-attacks on vehicle CAN bus systems using the CANoe environment. In addition to the implementation of the CAN bus attacks, the realization of a rule-based intrusion detection system to filter out malicious messages in the data stream of the CAN bus was demonstrated. This structure was already tested by students and received positive feedback throughout. In particular, solving practical tasks on the real CAN bus demonstrator was positively mentioned and motivated the participants to find creative solutions. In the future, it is conceivable to equip the demonstrator with further functionalities to realize further attack vectors on automotive systems.

REFERENCES

[1] Y. Simon, “100 million lines of code: What the Golf 8 can do” [Online]. Available from: https://www.kfz-betrieb.vogel.de/100-millionen-codezeilen-das-kann-der-golf-8-a-824084/ 2022.05.19

[2] Statista, “Connected cars as a share of the total car parc in China in 2021, with a forecast for 2025, 2030, and 2035” [Online]. Available from: https://www.statista.com/statistics/1276410/connected-car-fleet-as-a-share-of-total-car-parc-in-china/ 2022.05.19

[3] S. Smith, “In-Vehicle Commerce Opportunities Drive Total Connected Cars to Exceed 775 Million by 2023” [Online]. Available from: https://www.juniperresearch.com/press/in-vehicle-commerce-opportunities-exceed-775mn 2022.05.19

[4] K. Koscher et al., “Experimental Security Analysis of a Modern Automobile” [Online]. Available from: https://ieeexplore.ieee.org/document/5504804 2022.05.19

[5] K. Kochetkova, “Shock at the wheel: your Jeep can be hacked while driving down the road” [Online]. Available from: https://www.kaspersky.com/blog/remote-car-hack/9395/ 2022.05.19

[6] C. Thompson, “A hacker made a $30 gadget that can unlock many cars that have keyless entry” [Online]. Available from: https://www.businessinsider.com/samy-kamkar-keyless-entry-car-hack-2015-8 2022.05.19

[7] R. Lieder, “Gateway processor evolution in automotive networks” [Online]. Available from: https://www.can-cia.org/fileadmin/resources/documents/conferences/2017_lieder.pdf 2022.05.19

[8] Robot Car Kit 4WD, [Online]. Available from: https://joy-it.net/en/products/Robot03 2022.05.19

[9] Arduino Nano, [Online]. Available from: https://store.arduino.cc/products/arduino-nano 2022.05.19

[10] Adafruit TB6612 1.2AMotor Driver Breakout Board, [Online]. Available from: https://www.exp-tech.de/en/modules/motor-controllers/dc-motors/6501/adafruit-tb6612-1.2amotor-driver-breakout-board 2022.05.19

[11] TB6612 1.2A DC/Stepper Motor Driver, [Online]. Available from: http://wiki.t-o-f.info/uploads/Arduino/Adafruit_TB6612_1motor_bb.png 2022.05.19

[12] DISTRELEC GROUP AG: RND Components Electromagnetic Buzzer, [Online]. Available from: https://www.distrelec.de/Web/Downloads/_t/ds/RND%20430-00009_eng_tds.pdf 2022.05.19

[13] JOY-IT: Joy-IT Ultrasonic Distance Sensor, [Online]. Available from: https://cdn-reichelt.de/documents/datenblatt/A300/SEN-US01-DATASHEET.pdf 2022.05.19

[14] JOY-IT: NodeMCU ESP32, [Online]. Available from: https://joy-it.net/en/products/SBC-NodeMCU-ESP32 2022.05.19

[15] SN65HVD23x 3.3-V CAN Bus Transceivers, [Online]. Available from: https://www.ti.com/lit/ds/symlink/sn65hvd230.pdf?ts=1646221051596&ref_url=https%253A%252F% 2022.05.19

[16] CiA Draft Standard 102 Version 2.0, [Online]. Available from: http://affon.narod.ru/CAN/DS102.pdf 2022.05.19

[17] M. Bresch and N. Salman, “Design and implementation of an intrusion detection system (IDS) for in-vehicle networks” Department of Computer Science and Engineering (Chalmers), Chalmers University of Technology, Göteborg, 2017, [Online]. Available from: https://publications.lib.chalmers.se/records/fulltext/251871/251871.pdf 2022.05.19

[18] Vector, “Testing ECUs and Networks With CANoe” [Online]. Available from: https://www.vector.com/int/en/products/products-a-z/software/canoe/ 2022.05.19