Tanksale *Cybersecurity (2024)7:4*
https://doi.org/10.1186/s42400-023-00195-4

Cybersecurity

RESEARCH

Open Access

Intrusion detection system for controller area network

Vinayak Tanksale[1*]

Abstract

The rapid expansion of intra-vehicle networks has increased the number of threats to such networks. Most modern vehicles implement various physical and data-link layer technologies. Vehicles are becoming increasingly autonomous and connected. Controller area network (CAN) is a serial bus system that is used to connect sensors and controllers (electronic control units—ECUs) within a vehicle. ECUs vary widely in processing power, storage, memory, and connectivity. The goal of this research is to design, implement, and test an efficient and effective intrusion detection system for intra-vehicle CANs. Classic cryptographic approaches are resource-intensive and increase processing delay, thereby not meeting CAN latency requirements. There is a need for a system that is capable of detecting intrusions in almost real-time with minimal resources. Our research proposes a long short-term memory (LSTM) network to detect anomalies and a decision engine to detect intrusions by using multiple contextual parameters. We have tested our anomaly detection algorithm and our decision engine using data from real automobiles. We present the results of our experiments and analyze our findings. After detailed evaluation of our system, we believe that we have designed a vehicle security solution that meets all the outlined requirements and goals.

Keywords Controller area network, Deep learning, Intrusion detection system, Long short-term memory, Machine learning, Recurrent neural networks

Introduction

Vehicular technology has been steadily improving to enhance the safety and comfort of automobiles. Today's automobiles consist of a wide variety of networks such as Controller Area Network, Local Interconnect Network, and Media Oriented Systems Transport. The rapid and omnipresent expansion of intra-vehicle networks has increased the number of vulnerabilities to these networks. Most modern vehicle systems implement various physical layer and data link layer technologies. Such networks not only interface among themselves but also with external networks. Vehicles are becoming increasingly smart, connected, and part of the Internet. This has given rise to multiple attack surfaces and vectors to automobiles. In Miller and Valasek (2015) demonstrated successful hacking of a car in motion on an interstate by jamming the transmission system and disabling the brakes at low speeds.

The number, severity, and variety of security attacks on vehicles is increasing. From jamming transmissions to disabling lane control systems, such attacks are a major threat to the driver and their surrounding vehicles (Larson and Nilsson 2008; Dibaei et al. 2020). Over the years, multiple security solutions have been proposed (Woo et al. 2016; Kim et al. 2021). Examples of such solutions include firewalls, network segmentation, signature-based scanning, and intrusion detection systems. Vehicular technology is a mix of multiple physical and link layer technologies. As new technologies are introduced for vehicles, these technologies have to interface with legacy technologies. The legacy technologies are widely prevalent in all vehicles and they provide critical control

*Correspondence:
Vinayak Tanksale
vjtanksale@bsu.edu
[1] Department of Computer Science, Ball State University, Muncie, IN 47306, USA

functions. The advantages of legacy technologies are reliability and low-latency (Lv et al. 2021). Security was not an integral part of the design of legacy technologies (Dibaei et al. 2019).

As technology evolves for autonomous vehicles, the number of attack surfaces will increase. Most modern vehicles are equipped with Adaptive Cruise Control (ACC), Advanced Driver Assistance Systems (ADAS), and Light Detection and Ranging (LIDAR) systems. Using these systems, a lot of critical features such as speed, lane control, navigation, and others are automatically controlled with minimal driver interaction. Such systems will continue to evolve as vehicles make rapid progress towards full automation. For example, multiple video cameras on the periphery of the vehicle are currently being used to automatically and safely change lanes, exit from a freeway, and stop at traffic lights and signs. Although these advanced systems use newer and faster physical and link layer technologies, the critical control and safety systems still run on legacy networks. There is a need to develop security solutions that will protect critical systems from attacks.

This paper extends our prior work (Tanksale 2020b, 2021) in the area of intrusion detection for Controller Area Networks. We presented an LSTM network to detect anomalous CAN frames in Tanksale (2020b). We presented an anomaly detection function design process in Tanksale (2021). Each function outputs whether a given CAN frame is anomalous or not. We use multiple functions in our anomaly detection engine and have presented the rationale for this in our prior work. The major contributions of this paper are (1) a decision engine that uses the outputs of multiple functions and contextual information to determine if an intrusion is occurring, (2) a modified anomaly detection function design process, and (3) validation on multiple real-world datasets including various makes and models. The decision engine can be generalized to make binary decisions based on contradictory inputs from multiple sources. It carefully considers contextual information as part of its determination process. All of these are novel contributions of the work that is presented in this paper.

This paper is organized as follows. Section "Controller area network" provides a brief introduction to the Controller Area Network (CAN). Section "CAN security" describes security weaknesses and attacks on CAN communications. Relevant related research is summarized in section "Background and related work". Our prior work that describes a Long Short-Term Memory parameter selection algorithm (Tanksale 2020b) and design of anomaly detection functions (Tanksale 2021) is described in section "Summary of our prior work". This section also presents a modified anomaly detection function design process as a result of further evaluation of our earlier algorithm. Section "Decision engine" presents the rationale for the proposed decision engine parameters, a process to handle functions with contradictory outputs, design of the decision engine, and evaluation based on real-world datasets. We perform a detailed comparative analysis with other similar systems in section "System analysis and discussion" and conclude the paper in section. "Decision engine".

Controller area network

Controller Area Network is the most common serial bus system that is used to connect devices within automobiles. The connected devices are commonly called Electronic Control Units (ECU) although there is a subtle distinction that is outlined later in this dissertation. Figure 1 demonstrates how ECU nodes are connected to a CAN bus. An electronic control unit controls an electrical subsystem in a vehicle. Most newer vehicles contain an average of 120 ECUs. ECUs are used in transmission control, engine control, speed control, airbag control, powertrain control, and many other vehicle subsystems.

CAN with flexible data-rate (CAN FD) is the latest communication standard that provides high data rates. Classical CAN was introduced in 1986 and implemented in 1988 and CAN FD was launched in 2012 and internationally standardized in 2015 in ISO 11898-1. Table 1 summarizes the communication speed for intra-vehicle network technologies. Figure 2 shows the format of a CAN-FD data frame.

CAN security

Security weaknesses

CAN messages are broadcast and do not contain the sender's address. All frames are received by all ECUs and each ECU determines whether to act on the frame based on the message identifier. A well-known flaw in any broadcast transmission is that malicious nodes can easily passively eavesdrop on all the frames transmitted by other nodes. This will allow a rogue CAN sensor to

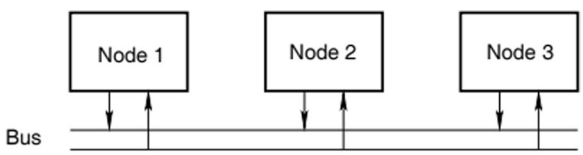

Fig. 1 CAN Bus and ECUs

SOF	Arbitration	Control	Data	CRC	ACK	EOF	IMF
1	12 or 32	8-9	up to 512	28 or 33	2	7	3

Fig. 2 CAN-FD frame format (size in bits)

Table 1 Communication speed

Type	Speed
Low-speed CAN	40–125 Kbps
High-speed CAN	40 Kbps–1 Mbps
CAN-FD	8 Mbps
MOST	100–150 Mbps
LIN	20 Kbps

read the CAN data traffic and later use it for a fabrication attack. Traffic on the CAN is not encrypted. CAN Message space is limited. It is fairly easy to capture CAN traffic and analyze it for traffic and message patterns. This allows attackers to passively monitor and collect detailed metrics about CAN traffic. All of this makes a replay attack fairly easy to execute. Lack of a sender's address in the CAN frame makes it a challenge to verify if a message was indeed sent by an ECU that is expected to send it. A rogue sensor could periodically report incorrect wheel speed or oil temperature values. Recall that a remote frame is used to solicit data from CAN sensors. Any malicious sensor can potentially respond to such a remote frame. Such malicious activity can potentially damage or disable critical control systems.

Apart from maintenance of sender integrity, there is no mechanism to verify data integrity (Xiong et al. 2019). Message Authentication Codes (MAC) are one solution to enforce data integrity. The payload of CAN frames can be at most 512 bits. There is not enough space in the data field for the actual message and a strong MAC. A majority of ECUs send very similar messages with only minor changes to the content of the message. This makes it easier to replay messages.

There are multiple interfaces into the CAN. The OBD-II port provides direct physical access to the CAN (Takefuji 2018). The OBD-II port only provides wired access to the CAN. A majority of modern automobiles are equipped with a multi-functional telematics system, which supports GPS, media entertainment, Bluetooth, cellular among others. All such interfaces are potential vulnerabilities that can be used in any of the aforementioned attack scenarios.

To summarize, the CAN protocol has the following weaknesses due to its design:

Communication

All nodes broadcast their messages on the CAN. A malicious node on the CAN can easily sniff all traffic. Message sent by a malicious node will be seen by all sensors. Broadcast is required for the network to function. Eliminating broadcast will necessitate a hardware and network change which is not practical.

Low-latency requirement

CAN messages are supposed to be sent and received in real-time. Any security mechanism may significantly add to the delay.

Lack of authentication

There is no support for source and message authentication. This makes the CAN network vulnerable to integrity violations and replay attacks. Authentication procedures will add to latency.

Our prior work (Tanksale 2019, 2020a, b, 2021) encompasses CAN security weaknesses and CAN security requirements.

Attacks on CAN communication

The following attack scenarios are possible:

- *Modification*—Malicious ECU sniffs frame and changes frame data
- *Interception*—Passively scan all traffic on CAN
- *Fabrication*—Malicious ECU generates frame that is supposed to be generated by other ECU(s)
- *Interruption*—Denial of Service attack where malicious ECU continuously sends frames with lower IDs to thwart transmission of higher priority frames

The U.S. Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) recently published an alert about a selective denial-of-service attack against the CAN standard which doesn't involve the transmission of any frames for its execution, and thus would be undetectable via frame-level analysis (NCCIC/ICS-CERT 2017). Some recent experiments have revealed vulnerabilities

00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	05	75	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	05	75	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	05	75	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	05	75	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	05	75	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	05	75	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	05	75	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS
00	00	00	00	CN	28	77	CS	00	00	00	00	CN	28	77	CS

Fig. 3 Normal CAN sequence (left), modified CAN sequence (right)

in the CAN and ECU architecture. White-hat hackers were able to gain access to the transmission system of a vehicle by gaining access to it via the infotainment system (Meyer 2019). Attackers are likely to focus on vehicle entry-points such as Bluetooth, over-the-air diagnostics, Wi-Fi, Zigbee, infotainment systems, and mobile applications.

The attack shown in Fig. 3 was demonstrated by Miller and Valasek (2015) on a Toyota Prius. The CAN message that sends the vehicle's speed to the speedometer for display was manipulated. As a result, the speedometer was displaying the incorrect speed which results in a dangerous driving condition. The Counter (CN) and the Message ID (0x00B4) were unchanged. The Checksum (CS) was recomputed for the modified data. The current vehicle speed of 64.23 mph (0x2877) was replaced with 8.66 mph (0x575) in the highlighted frames. This caused the driver of the vehicle to accelerate and the vehicle reached a dangerously high speed.

Background and related work

In this section, we review prior research, in the areas of confidentiality and integrity of CAN communications, that utilizes classifiers, intrusion detection, and various types of deep neural networks. Kleberger et al. (2011) survey the current research related to securing the connected car, with a focus on the security of the in-vehicle network. Taylor et al. (2016) use Long Short-Term Memory (LSTM) networks to detect sequential anomalies in CAN data, however their approach results in higher than acceptable false positive rate. Desta et al. (2020) propose an LSTM network to predict the next CAN arbitration ID and compare it with the actual arbitration ID, however their proposed scheme is vulnerable to replay attacks. A broad survey of current intrusion detection systems for all types of in-vehicle networks is presented in Wu et al. (2020). Nie et al. (2020) propose an intrusion detection system, using convolution neural networks, for Internet of Vehicles using road side units. We adopt some of their feature extraction techniques in our research.

Gated Recurrent Units (GRU) use fewer gates and parameters than LSTM, however, prediction accuracy of LSTMs is higher than the prediction accuracy of GRUs (Tanksale 2023) for temporal vehicle data. A system proposed in Lee et al. (2022) uses periodic properties of CAN messages to detect masquerade attacks. There are many problems with this approach. It does not detect attacks on CAN messages that are not periodic. Their results and analysis are based on data from 2 vehicles which is not nearly enough to validate any system. The system relies on projecting when a normal ECU suspends message transmission and then flags frames sent after that time as intrusive. Another approach uses Convolution Neural Networks and Gated Recurrent Units to flag intrusions (Javed et al. 2021). This system performs well

to detect intrusions and considers different types of attacks on the CAN. One of the attacks that this system specifically addresses is an impersonation attack. One potential drawback of this approach is that the hacker can send fabricated frames with the same frequency as normal CAN messages and such an attack may not be detected. CAN message sequencing and corresponding timing analysis is used to detect intrusions (Marchetti and Stabili 2017). This system is vulnerable to fabricated and replay CAN frames that are inserted using the same periodicity as normal CAN frames. Attacks on multiple CAN parameters are not detected with high sensitivity. Generative pretrained transformer (GPT) is a well-researched technique for natural language generation. Researchers have used GPT to detect intrusions in CAN (Nam et al. 2021). Their system is designed to detect only injection attacks. It works well when a limited number of CAN message types are used in the attack. The researchers recognize that their system will not detect all types of attacks and attacks launched using multiple CAN message types.

Jin, Chung, and Xu propose a signature-based intrusion detection system (Jin et al. 2021) that can detect replay and interruption attacks. Katragadda et al. (2020) propose a system to detect low-rate replay attacks however it their solution works only for specific types of replay attacks. Wu et al. (2018) use statistical measurements related to the message id to attempt to thwart replay and reverse-engineering attacks on the CAN. A basic LSTM network was used to classify CAN frames as normal frames or attack frames in Hossain et al. (2020). There is no novel approach proposed here and although accuracy is extremely high there is no discussion of sensitivity.

Support vector machines (SVM) were first introduced by Cortes and Vapnik (1995); Vapnik (2013). A Support Vector Machine-based classification and prediction approach is detailed in Tanksale (2019). A weakness of this approach is the absence of contextual time-series information in classifying CAN frames. Kang and Kang propose an efficient intrusion detection system based on a deep neural network for the security of in-vehicular network (Kang and Kang 2016). The use of Support Vector Machines (SVM) for the detection of DoS attacks have been discussed in Mukkamala and Sung (2003). The performance of the proposed method has been validated experimentally and shown that proposed SVM-based detection approach achieves very high detection accuracy. A general classification problem in n-dimensional space is defined in Cristianini et al. (2000). Attacks on safety-critical CANs are summarized in Fröschle and Stühring (2017) and we incorporate these ideas in some of our experiments. A general machine learning based detection system is presented in Minawi et al. (2020) but it was not tested on real world attack data. Zhou et al. (2019) propose a novel intrusion detection system using ECU fingerprinting by calculating statistical features of the bit time of recessive and dominant bits.

Sekar et al. (2002) combine specification-based with statistical anomaly detection techniques to ease the task of model construction and to reduce false alarm rate. The authors acknowledge that such a system is vulnerable during training. SVM is a supervised machine learning model which is well-known for its great performance in pattern recognition and classification tasks with high dimensional data (Peng et al. 2015). Nguyen and Armitage (2008) describe machine learning techniques for Internet traffic classification. The techniques described therein do not rely on well-known port numbers but on statistical traffic characteristics. Multiple ECUs on a high-speed CAN bus and a low-speed CAN bus, connected using a bridge, were simulated using CANoe (Vector https://www.vector.com/int/en/products/products-a-z/software/canoe/) software to implement a replay attack by Hoppe and Dittman (2007). An ECU on the low-speed CAN bus was able to capture messages from and replay those messages to the high-speed CAN bus.

Machine learning is often employed to implement anomaly-based intrusion detection. The network traffic is collected from the Network Interface Card or from a packet capture file containing previously captured network traffic. The packets are then filtered and sent to a feature extraction engine, which computes flow-based and header-based attributes. These attributes are assembled into a feature vector, which provides the input data for the training or classification phases of a classifier. Tavallaee et al. (2008) proposed an anomaly detection scheme using the correlation information contained in groups of network traffic samples. The main idea is to compare the signs in the covariance matrix of a group of sequential samples with the signs in the covariance matrix of the normal data obtained during the training process. Machine learning techniques have been widely used in detecting network anomalies because machine learning can construct models automatically based on the given training data. Machine learning techniques have achieved good performance on anomaly-based detection systems. Some typical methods used in network traffic anomaly detection include Bayesian networks, support vector machine (Sung and Mukkamala 2003), fuzzy logical (Hoang et al. 2009), genetic algorithm (Li 2004), and decision trees.

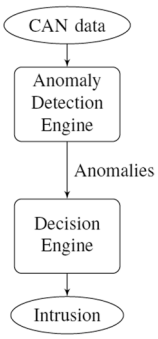

Fig. 4 Intrusion detection system

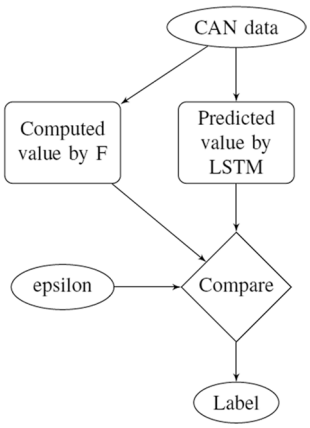

Fig. 5 Labeling done by a function

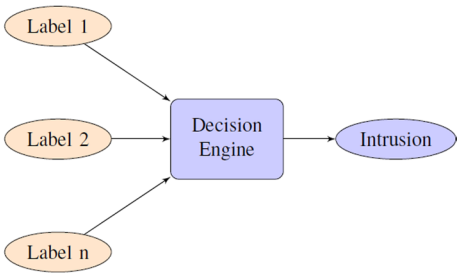

Fig. 6 Intrusion detection system

To summarize, a variety of intrusion detection systems have been proposed over the years. CAN data is temporal-based as future vehicle behavior is dependent on past and current behavior. Dependence on periodic properties (Lee et al. 2022) of CAN is vulnerable to attack frames that are inserted at regular intervals in the CAN traffic. Systems that use CNNs and GRUs (Javed et al. 2021) also depend on periodic properties of the CAN network. Classification and prediction models are unable to address contextual time-series-based nature of the CAN data (Tanksale 2019; Mukkamala and Sung 2003). LSTM networks are designed to predict time-series data based on long-term dependencies. They have been successfully implemented in speech recognition and sentiment analysis (Huang et al. 2022; Oruh et al. 2022). Hence, our implementation of a system to predict values of functions based on CAN parameters uses LSTM networks.

Summary of our prior work

We propose a comprehensive intrusion detection system for the Controller Area Network. The overall architecture for such a system is shown in Fig. 4. The two main components of this system are an anomaly detection engine and a decision engine. The anomaly detection engine is described in detail in our prior work (Tanksale 2020b, 2021). The focus of this paper is the decision engine which is described in section "Decision engine". In this section, we briefly summarize our prior work.

We presented an LSTM parameter selection algorithm in Tanksale (2020b). We presented a design process for anomaly detection functions in Tanksale (2021). Multiple strong functions are constructed using our function design process and LSTM networks are used to predict the values of these functions. We compare the predicted value ($\widehat{y_t}$) with the computed value (y_t) of the function. Let z_t be a variable used to indicate if a CAN frame is labeled as anomalous by the function. $z_t = 1$ if the frame is anomalous, $z_t = -1$ if it is not. If $|y_t - \widehat{y_t}| > \epsilon$, then we label the frame as anomalous. Figure 5 outlines this process for one function.

We use multiple functions as part of our anomaly detection engine. Figure 6 outlines the design of our intrusion detection system thus far.

Attack model

To introduce anomalies in normal CAN traffic data, we modify CAN variable values using multiple techniques. For each of our experiments, the testing data contained 1% malicious CAN frames. The following modifications were made to CAN variable values:

Scale	Multiply the CAN variable values with a scaling factor.
Shift	Shift the CAN variable values.
Random	Randomly generate CAN variable values to replace actual values.
Replay	Repeat earlier CAN variable values.

The attacker can place malicious CAN frames in various locations when executing an attack. When inserting

	Brake position	Brake pressure	Wheel speed	Current gear	Lateral acceleration	Steering angle	Accelerator pedal position	Longitudinal acceleration	Engine coolant temperature	Intake air temperature	Fuel consumption	Inhibition of engine fuel cutoff	Engine in fuel cutoff	Fuel pressure	Current spark timing	Torque of friction	Engine torque	Flywheel torque	Spark angle from TCU	TCU engine torque request	Air compressor activation	Clutch operation ack
Brake position		X	X				X	X			X	X	X				X					X
Brake pressure	X		X		X		X	X			X	X	X				X			X		X
Wheel speed	X	X		X	X		X	X														
Current gear			X					X			X			X						X		X
Lateral acceleration		X	X			X	X									X		X				
Steering angle					X																	
Accelerator pedal position	X	X	X		X			X			X											
Longitudinal acceleration	X	X	X	X			X							X								
Engine coolant temperature										X			X	X								
Intake air temperature									X												X	
Fuel consumption	X	X		X			X		X			X	X	X								
Inhibition of engine fuel cutof	X	X									X											
Engine in fuel cutoff	X	X							X		X											
Fuel pressure				X				X	X		X											
Current spark timing																						
Torque of friction					X																	
Engine torque	X	X																	X			
Flywheel torque					X																	
Spark angle from TCU																	X					
TCU engine torque request		X		X																		
Air compressor activation										X												
Clutch operation ack	X	X		X																		

Fig. 7 CAN variables relationship matrix

anomalous CAN frames to test our system, we chose multiple placement locations:

Random Malicious frames are placed in random locations in the frame sequence.

Single group Malicious frames are grouped together in one random location.

Multiple groups Malicious frames in n equal-sized groups are placed in n periodic locations.

Function design

Based on observations from prior experiments, we need to build a function that addresses the weaknesses of the preliminary function that we used. We are using CAN variables that are related to the critical functionality of the vehicle. A vast majority of the variables are collected and processed by the engine control module and the transmission control module. The matrix in Fig. 7 lists the CAN variables.

Using the mechanical and electrical properties of the CAN variables and their logical relationships, we built a

relationship matrix as shown in Fig. 7. This matrix will be used to build rules that define relationships between CAN variables using various operations.

Operations and rules

Multiple types of relationships and dependencies exist between CAN variables. They can be directly or inversely proportional to one another, or they can be threshold or distance based, or they could be correlated. To capture the various relationships between CAN variables, we define a list of operations:

1. correlation
2. multiplication
3. distance
4. boolean operations

The Pearson correlation coefficient of two variables p and q over n measurements is defined as

$$corr_n(p,q) = \frac{n\Sigma p_j q_j - \Sigma p_j \Sigma q_j}{\sqrt{[n\Sigma p_j^2 - (\Sigma p_j)^2][n\Sigma q_j^2 - (\Sigma q_j)^2]}} \quad (1)$$

The rules file contains all operations between CAN variables. For ease of implementation and automation, each line in the rules file defines one function in postfix notation. A number is assigned to each CAN variable and a letter is assigned to each operation. An example rule is $< 0, 13, \mathtt{a} >$ which represents the correlation (a) between brake position (0) and fuel pressure (13). Another rule is $< 9, 10, \mathtt{c} >$ which is postfix representation of the multiplication of intake air temperature and multiplicative inverse of fuel consumption. Few other examples of rules are $< 11, 12, \mathtt{g} >$, $< 19, 0, \mathtt{a} >$, and $< 2, 3, \mathtt{c} >$.

Function properties

As discussed earlier, a function with only two arguments can be easily attacked. Hence, we need a minimum value for the number of CAN parameters passed to a function. We need to ensure that changing one CAN variable creates a cascade effect requiring the attacker to change multiple CAN variable values. Also, if a CAN variable is related to multiple variables, then any change in this variable's value should significantly affect the value of the function. These desired function properties can be modeled as a graph.

We use an undirected graph to model the relationship between two CAN variables in a function. We represent CAN variables, used in the function, as vertices in this graph. Operations between variables used in this function will be represented by an edge of this graph. Let $G = (V, E)$ be an undirected graph where V is the vertex set and E is the edge set. An edge in an undirected graph is a set $\{u, v\}$ where $u, v \in V$ and $u \neq v$. The degree of a vertex is the number of edges incident on it. The degree of the graph is the maximum of its vertices' degrees. A path $< v_1, v_2, \ldots, v_k >$ forms a cycle if $k > 0$, $v_0 = v_k$, and edges on the path are distinct. The cycle is simple if $v_1, v_2, \ldots, v_k$ are distinct. A Hamiltonian cycle is a simple cycle that contains each vertex in V.

To enforce the cascade effect mentioned earlier, we can require our graph to have a Hamiltonian cycle. Changes to the value of a CAN variable, that is related to multiple variables in the function, will require changes to values of multiple CAN variables. This will make it difficult for the hacker to modify all required CAN variables such that the computed value of the function is close to its predicted value. To enforce this, we need to ensure that the degree of at least one graph vertex is close to the degree of the graph. It is possible that based on the variables that are part of the function and the existing relationships between them, we may generate a graph that does not meet the desired properties. Hence, we introduce the concept of an *artificial edge* between vertices. Such as edge does not represent any physical or mechanical relationship between the respective CAN variables. We use artificial edges so that we can efficiently achieve our desired properties. We need to be careful that we do not use multiple artificial edges as that will weaken the function. Hence, as we add artificial edges to our graph, we make sure that the number of artificial edges is always less than a third of the number of natural edges. Thus, we propose that the graph representing an anomaly detection function (function graph) meet the following properties. Let m represent the number of CAN variables used in the function.

Property 1 $m \geq 5$.

Our experiments have shown that we need a minimum number of CAN variables passed to a function to ensure that the hacker cannot easily attack the function.

Property 2 *Function graph must contain at least one node of degree* $\geq m - 2$ *or a Hamiltonian cycle.*

A Hamiltonian cycle is a closed loop through a graph that visits each node exactly once. Changes to the value of a CAN variable that is related to multiple CAN variables within a function causes significant changes in the

function's computed value. The hacker will need to modify the values for all the CAN variables that are related to the first variable. A similar cascade effect can be achieved if the CAN variables passed to a function are related to each other in a cycle. In both cases, it becomes difficult for the hacker to launch a successful attack.

Property 3 *Total number of artificial edges in the function graph must be less than a third of the total number of edges.*

The computed value of a function is composed of operations defined between pairs of CAN variables. Given these operations between CAN variables, if we are unable to construct a function that is not vulnerable to attack, then we add new relationships between CAN variables. The purpose of this is to design a function that is not vulnerable to attack. The new relationships are added one at a time and each new added relationship weakens the functions. Our experiments have demonstrated that the number of new relationships between CAN variables can at most be a third of the number of actual relationships without making the function vulnerable.

Property 4 *Node(s) with degree* $\geq m-2$ *from Property 2 can have at most one artificial edge.*

The rationale for having a CAN variable related to multiple CAN variables is that it will be difficult for the hacker to modify all CAN variables so that the computed value of the function does not change. If new relationships between CAN variables are added to such a variable then they need to be limited so that the function is not vulnerable.

Property 5 *Total number of artificial edges within the Hamiltonian cycle from Property 2 must be less than a third of the total number of edges in that Hamiltonian cycle.*

The goal of having CAN variables related to each other in a cycle is to make it challenging for the hacker to attack the function. The number of new relationships added to the function must be limited or else the function will become vulnerable.

Function design process

In this section, we describe an algorithm for designing anomaly detection functions that meet the properties listed in section "Function properties". All of our functions contain at least 5 CAN variables. To begin, we create a set of all m-tuples (m=5) of CAN variables and our algorithm tries to construct a function using each tuple. A random m-tuple is selected from the set of all m-tuples. All rules that contain these m CAN variables are considered. A function graph is then constructed using selected rules and the m CAN variables. A function graph must meet the 5 properties outlined in section "Function properties". If it does not, then we add an artificial edge and check to see if all properties are being met. If such a function cannot be constructed, then we discard this m-tuple and select the next one.

A function may contain one or more artificial edges to satisfy the graph's properties. An artificial edge does not define an operation. By adding an artificial edge we are adding a new term in the function definition. For this reason, we need to assign one of our operations to the artificial edge. Also, this newly added operation must not contribute to the function's output as much as the other edges. Hence, we assign a weight between (0, 1] to this operation. We use a grid-search process to create different variations of the function. To do so, once a function graph meets all required properties, we construct multiple variations of the function using all operations on all artificial edges, with all weights, of the function graph. Each variation of the function is trained using the LSTM training process detailed earlier. We test the model generated by each variation of the function by introducing all four anomaly types in all three locations in the test dataset. For each anomaly type and location, we measure the sensitivity and specificity. We calculate the mean sensitivity and specificity over all anomaly types and locations. Game-theory principles are used to find the optimal threshold values for intrusion detection system metrics (Laszka et al. 2016; Creech and Hu 2014). These values are passed as input to the algorithm ($\eta_{se} = 0.92$ and $\eta_{sp} = 0.97$). We only consider function variations that achieve a mean sensitivity of at least η_{se} and a mean specificity of at least η_{sp}. Such a function variation is added to our list of function candidates. Once we have completed evaluating all possible function variations, we choose the function variation with the highest sensitivity. This function variation is added to our final list of functions. We repeat this process, making sure that we don't select the same m-tuple, until we construct the desired minimum number of functions. If all m-tuples are exhausted, then we increase the value of m and repeat the process. Algorithm 1 outlines the process of designing and selecting anomaly detection functions.

Algorithm 1 Algorithm for designing anomaly detection functions

Input: $(NUMFUNC, \eta_{se}, \eta_{sp})$

$m \leftarrow 5$
$counter \leftarrow 0$
while $counter < NUMFUNC$ **do**
    Let Ψ be the set of all m-tuples of CAN variables
    while Ψ is not empty **do**
        Select random m-tuple from Ψ
        Select all rules containing selected m CAN variables
        Construct function graph
        while property 3 true **and** graph does not meet all properties **do**
            Add artificial edge between any two nodes that are not connected
        end while
        if no graph found **then**
            $\Psi \leftarrow \Psi \backslash \{m\text{-tuple}\}$
            goto 6
        end if
        for all operations on all artificial edges with all weights **do**
            Select parameters for LSTM network
            Train the LSTM network
            for all anomaly types and all locations **do**
                Test the LSTM network
                Measure *sensitivity* and *specificity*
            end for
            Compute mean *sensitivity* and mean *specificity*
            if mean *sensitivity* $> \eta_{se}$ and mean *specificity* $> \eta_{sp}$ **then**
                Select function if *sensitivity* is higher that current selection
            end if
        end for
        if function selected **then**
            $counter \leftarrow counter + 1$
        end if
        if $counter = NUMFUNC$ **then**
            break
        end if
    end while
    $m \leftarrow m + 1$
end while
return *list of functions*

Decision engine

The decision engine is the second component of our intrusion detection system for the CAN. The detection of anomalous CAN frames by the various functions that make up the anomaly detection engine does not necessarily indicate that an intrusion is occurring. The decision engine utilizes the output of the anomaly detection engine and other contextual data to determine if the vehicle is being intruded on.

Intrusion

We define an intrusion into a vehicle as the malicious presence of anomalous CAN frames on the CAN bus. CAN frames that are sent in response to an intrusion may not be detected as anomalous by the anomaly detection engine. However, the decision engine should be able to determine these as intrusive. Our decision engine design takes into consideration contextual and temporal data in addition to the output of the anomaly detection engine.

In the next sections, we design a model for the decision engine that uses multiple parameters to determine if the vehicle is being intruded upon.

Decision engine rationale

Each anomaly detection function labels a CAN frame as anomalous or not. If all functions unanimously agree in their labeling of a frame, then the decision engine makes the same decision that all functions agree on for that frame. When there is no unanimous agreement between the functions for a particular frame, we propose to evaluate various parameters and then decide if the frame is part of an intrusion.

We need to determine each CAN sensor's role in the composition of a function. A vertex in a function graph that represents a CAN variable could be connected to multiple vertices. In this case, any change in the value of this CAN variable has a significant impact on the output of the function. As a consequence, this will impact the labeling of this message frame as anomalous or not. Additionally, if this sensor is closely related to the CAN message frame that is being analyzed then the labeling done by the function is important. The classification done by a function that contains multiple such variables is very important in making a decision on the frame. If the RPM sensor is being attacked and the CAN message frame is a response to an RPM request, then this is important contextual information and the decision engine needs to weigh this accordingly. If a function contains multiple CAN variables that are not related to the CAN message then the classification of the message by this function is less significant and the decision engine needs to weigh this accordingly.

As mentioned earlier, it is possible that a vertex representing a CAN sensor is connected to a majority of vertices in the function graph. In addition, it is possible that this sensor is closely related to the message. Consider a situation where such a sensor is used in multiple functions. If the labels by these functions are in agreement, then the decision engine needs to consider this. Alternatively, if the labels by these functions are contradictory to each other then the decision engine needs to evaluate this situation in detail. The decision engine should utilize the role of these CAN variables in other anomaly detection engine functions and the corresponding labels of these functions.

Decision engine parameters

Based on the design discussion from the earlier section, we use multiple parameters to construct the decision engine. We consider the relationship between each CAN variable and the message when determining intrusions. We define a value γ as a way of measuring relationships between CAN variables and messages. γ values are determined based on input from *subject matter experts* and $\gamma \in (0, 1]$. A γ value closer to 1 indicates that the sensor is closely related to the message. For example, γ(wheel speed sensor, request RPM message) is close to 1 whereas γ(door sensor, request oil temperature) is close to 0. Let γ_{ij} be the γ value for CAN variable c_j and message M_i. For each function F_k, per message M_i, we construct a vector $\overrightarrow{\gamma_{ik}} = < \gamma_{ij} >$.

It is possible that all or a majority of CAN variables passed to a function F_k are closely related to message M_i. On the other hand, it possible that none or very few of CAN variables passed to F_k are closely related to M_i. In the former case, the labeling done by a function needs to be given more importance since all or majority of variables are closely related to the message under consideration. In the latter case, the labeling done by this function needs to be given less importance since none or very few of the CAN variables are related to the message. With each function making its own labeling for each message, we need a measurement to decide which functions' labeling for that message carries more weight. We use this principle to construct a measurement later in this section.

If any two CAN variables are physically or mechanically related to each other, then there exists a natural edge between them in the function graph. The degree of a vertex in a function graph is the number of natural edges that are incident on it. This degree is directly proportional to the effect this variable has on the labeling done by this function. To take into account the effect of changing a CAN variable (as part of an attack) on the labeling done by a function, we need to consider the ratio of the degree of this vertex to the degree of the graph of the function (using only the natural edges). The higher the ratio, the more effect a change in the value of the CAN variable will have on the labeling done by function. We define

$$r_j = \frac{degree\ of\ vertex\ c_j}{degree\ of\ graph} \tag{2}$$

which gives us the vector $\overrightarrow{r_k} = < r_j >$ for function F_k. For example, given the following function graph in Fig. 8 for the function defined in Eq. 3

$$F(x_0, x_1, x_2, x_3, x_4) = \frac{x_0}{x_1} + corr(x_0, x_4) + \frac{x_3}{x_2} + corr(x_0, x_3) + corr(x_0, x_2) \tag{3}$$

where x_0: brake position, x_1: brake pressure, x_2: wheel speed, x_3: accelerator pedal position, x_4: engine torque, its $\overrightarrow{r} = < 1, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{1}{4} >$. This implies that CAN variable

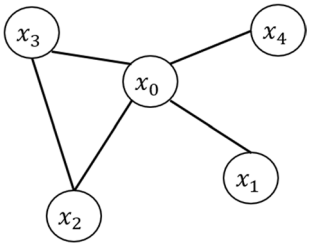

Fig. 8 Graph for function defined in Eq. 3

x_0 has the most effect on the labeling done by this function whereas CAN variables x_1 and x_4 have the least effect on this function's labeling.

To summarize, for each function F_k per message M_i, we have $\overrightarrow{\gamma_{ik}}$ that measures how closely related each sensor is to the frame being considered. We also have $\overrightarrow{r_k}$ that measures how much each CAN sensor in a frame proportionally contributes to the labeling done by the function. $\overrightarrow{r_k}$ is independent of M_i. As mentioned earlier, if a particular CAN parameter with a high r_j value is under attack then any changes to its value will affect the labeling done by the function. In addition, if this CAN parameter is closely related to the message, then the labeling done by this function is important. To measure the effect of both of these, we multiply the two for each CAN sensor c_j per function F_k per message M_i. This value will $\in (0, 1]$ because $\gamma \in (0, 1]$ and $r \in (0, 1]$. Observe that if this value is closer to 1, then it indicates that this sensor is closely related to the message and is connected to a lot of the other sensors used by the function. For example, if the vertex representing the accelerator pedal position is connected to a majority of vertices in the function graph and if the message being analyzed is about the RPM (high γ value), then the product of γ and r for the accelerator pedal position will be close to 1. Any malicious change in the value of this sensor will affect the computed value of the function. As a result, the message frame will be labeled as anomalous by the function. Hence, we call this quantity the evidence provided by the sensor to support the function's labeling (z_{ik}).

We collect evidence for each of the variables to support the function's labeling and call it an evidence score for message M_i provided by the variables in function F_k. Each of the functions are composed using a differing number of CAN variables. To be able to compare the evidence score of multiple functions with varying number of parameters we normalize its value. More formally, evidence score denoted by ES_{ik} for message M_i and function F_k that is composed of n_k CAN variables is computed as

$$ES_{ik} = \frac{\overrightarrow{\gamma_{ik}} \cdot \overrightarrow{r_k}}{n_k} \tag{4}$$

Each function F_k makes a labeling z_{ik} for message M_i. Consider two functions that contradict each other in their labels for message M_i. There are multiple possibilities to consider to resolve this contradictory labeling between each pair of functions. We need to analyze the evidence scores for both functions. If both functions present high evidence scores then we need to closely examine the composition of each function. If one of the two functions presents high evidence score and the other does not, then we need to give more weight to the labeling of the function that presents high evidence score. Consider the case where both functions do not share a lot of CAN variables between them and both present high evidence scores. This implies that for message M_i, each function presents high evidence score while not sharing a lot of the CAN variables with the other function, but contradicts the other function in its labeling. To resolve this, we need to consider the role each CAN sensor in each function plays in the labeling of all other anomaly detection engine functions. If the functions share a lot of variables, then there is no way to resolve the contradiction and we make no decision on intrusive behavior. To determine the extent to which functions share CAN variables, we propose to use a distance measure $d(F_p, F_q)$ between two functions. Let C_p be the set of CAN variables that are passed to F_p and let C_q be the set of CAN variables that are passed to F_q. We need to know how similar (or dissimilar) these two functions are in terms of the CAN variables used in each of them. The Jaccard similarity coefficient is used to measure the similarity between two sets and the Jaccard distance is used to measure the dissimilarity. Jaccard distance d is defined as

$$d(F_p, F_q) = 1 - \frac{|C_p \cap C_q|}{|C_p \cup C_q|} \tag{5}$$

We use the Jaccard distance to determine how far apart the two functions are in terms of CAN variables. A higher value of the Jaccard distance indicates that the functions do not share a lot of CAN variables. In the next two sections, we discuss the design of the decision engine.

Functions with contradictory labels

Using the design principles described in section "Decision engine rationale" and the measurements from section "Decision engine parameters", we now design a decision engine to determine intrusions. Given a message frame M_i, the decision engine's goal is to determine if an intrusion is occurring. If the decision engine cannot conclusively make a determination that a message frame is part of an intrusion, then the frame is classified as part of normal CAN traffic.

Each function has labeled a frame as anomalous or not. There is an evidence score for each frame per function. We propose to use a heuristically-determined threshold for evidence score. Any evidence score above this threshold will be considered a strong evidence score. It is assumed that each frame is non-intrusive unless the decision engine determines it to be intrusive. A tally is used to keep track of the interim results and determinations made by the decision engine. For each function that possesses a high evidence score and has labeled the frame as anomalous, the tally leans towards intrusion. Similarly, for each function that possesses a high evidence score and has labeled the frame as normal, the tally leans towards non-intrusion. In our experience, it is rare that all functions unanimously agree in the labeling of a frame.

Our approach seeks to resolve the contradictions in the labels of a message frame by performing further analysis using the measurements defined in section "Decision engine parameters". The decision engine does a pair-wise comparison of functions whose labels are contradictory to each other. For each such pair, the tally will either lean towards intrusion or non-intrusion or not lean towards either direction. Within each function pair, one function may possess a strong evidence score and the other may not. This implies that the labeling of the function with the strong evidence score is more significant than the labeling of the other function. In this case, the decision engine weighs the labeling of the function with the strong evidence score. This weight is determined by comparing the evidence scores of both functions. As a result, the tally will lean in one of two directions—intrusion or non-intrusion. Another case to consider is when both functions in the function pair do not present strong evidence. The tally does not lean in either direction in such a situation.

Consider the situation where both functions in the function pair possess strong evidence. If both functions do not share any CAN variables or share a few variables, then it implies that their labeling was done using sets of CAN variables with little overlap. On the other hand, if these functions share a majority of CAN sensors then that implies that both functions possess strong evidence towards contradictory labeling and they came to that decision using mostly the same CAN variables. In this situation, the decision engine cannot make a determination on which way the tally would lean for this function pair. The decision engine uses a heuristically-determined threshold to determine if functions are apart from each other using the Jaccard distance measurement mentioned in section "Decision engine parameters". If the Jaccard distance is more than this threshold, then we propose to analyze this further.

To recap, we are analyzing two functions, each possessing strong evidence scores, that are sufficiently distant from each other. Since there is little overlap in the CAN variables used in these functions, the decision engine looks at each CAN variable used by each function and analyzes its effect in all other functions of the anomaly detection engine. Each such variable used in the other anomaly detection engine functions possesses a relationship to the message frame (γ) and a ratio defined in Eq. 2. The function in which this CAN variable is used has its labeling as well. We weigh this labeling using the γ value and the ratio from Eq. 2. As this process is being completed for all CAN variables used in the function-pair, we accumulate the weighed labels. At the end of this process, the decision engine cumulatively analyzes the weighed labels and makes a determination as to which direction the tally should lean after resolving the contradictory labels of the functions in the function-pair. We reiterate here that there has to sufficient information to determine if a frame is intrusive, if not then it is deemed non-intrusive. Hence, we use a heuristically-determined threshold that the weighed labels should meet for the tally to lean towards an intrusion.

The entire analysis of a message frame is now complete. The decision engine is ready to determine if this frame is intrusive. Throughout its analysis, the decision engine has been collecting information using the tally. As mentioned earlier, there must be sufficient information to determine if a frame is intrusive. Hence, we use another heuristically-determined threshold which the tally has to meet for the decision engine to determine if this message frame is intrusive. The lack of sufficient information results in the frame being labeled as non-intrusive.

Decision engine design

The overall decision engine algorithm is detailed in Algorithm 2. Further analysis, which is required for one situation, is detailed in Algorithm 3.

Algorithm 2 begins by initializing a *decision* variable that is eventually used to determine if an intrusion is occurring. If all functions agree in their respective z values, then all functions unanimously agree in their labeling and that result is returned. However, if that is not the case, then we perform further analysis. The labeling (z) of a function, whose evidence score is higher than the threshold β, is considered significant and the *decision* variable is updated accordingly. The algorithm then considers functions that do not agree in their z values. It does so by analyzing each pair of functions that contradict each other in their labels. The reason for doing so is to determine how much one function's result is significant over the other function. If the evidence presented by both functions is strong but they don't agree in their labels, then we look at and compare the composition of both the functions. The Jaccard distance is used to determine how similar the two functions are in terms of CAN parameters used by each function. A distance of 1 indicates there are no common CAN parameters between

the two functions. A distance of 0 indicates that both functions share all of their CAN parameters. For the case where one function presents a strong evidence score and the other does not, we compute the ratio of the stronger evidence score to the sum of the evidence scores. This ration is used as a weight and is multiplied with the z value of the function with the stronger evidence score.

Now that we have addressed the cases where only one function presents strong evidence or both functions present weak evidence, we consider the case where both functions present strong evidence. If both functions present strong evidence and are sufficiently distant from each other, as measured by the Jaccard distance, then we analyze them further with the *RESOLVE* procedure outlined in Algorithm 3. Let α be the threshold to determine if the functions are sufficiently apart. The *RESOLVE* procedure returns either 0 or 1 or -1. Return value of 1 contributes towards intrusion, -1 contributes towards non-intrusion, and 0 indicates that no decision can be made. The return value is added to the *decision* variable in Algorithm 2. Within the *RESOLVE* procedure, we take the union of CAN variables that are part of the two functions being considered. Next, we consider all the functions, that are part of the anomaly detection engine, that contain the above set of CAN variables. Given the contradictory labels of the two functions being considered, we want to consider the labels of other functions that contain these CAN variables. We also want to weigh the function labeling by taking into account the γ value of the CAN variable and CAN message and the degree ratio (r). The *result* variable, initialized to 0 when the procedure begins, keeps track of the weighted labels of the functions. If *result* value is greater than or equal to a threshold (ϕ) then we return either 1 or -1, else we return 0.

After analyzing a CAN message frame through the various data paths and decisions, the algorithm makes a determination on intrusion based on the value of the *decision* variable. A non-negative value greater than a threshold (θ) suggests that an intrusion is occurring. If a conclusive decision cannot be made for the CAN message frame under consideration, then the algorithm determines that no intrusion is occurring.

Algorithm 2 Decision engine algorithm

Input: $(M_i, \alpha, \beta, \theta)$ - where α is limit on distance between functions, β is floor for strong evidence, θ is threshold for making decision

$decision \leftarrow 0$
if $z_r = z_s \ \forall r, s$ where $r \neq s$ **then**
  return z_r
end if
for all F_k with $ES_{ik} \geq \beta$ **do**
  $decision \leftarrow decision + z_k$
end for
for all (F_p, F_q) where $p \neq q$ and $z_p \neq z_q$ **do**
  if $ES_{ip} \geq \beta$ **then**
    if $ES_{iq} \geq \beta$ **then**
      if $d(F_p, F_q) \geq \alpha$ **then**
        $decision \leftarrow decision + \text{RESOLVE}(F_p, F_q)$
      end if
    else
      $decision \leftarrow decision + \frac{ES_{ip}}{ES_{ip}+ES_{iq}} z_p$
    end if
  else
    if $ES_{iq} \geq \beta$ **then**
      $decision \leftarrow decision + \frac{ES_{iq}}{ES_{ip}+ES_{iq}} z_q$
    end if
  end if
end for
if $decision > 0$ and $decision \geq \theta$ **then**
  return 1
end if
return -1

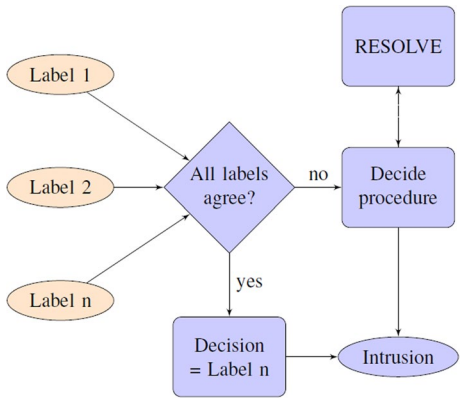

Fig. 9 Intrusion detection system

Algorithm 3 RESOLVE procedure

Input: (M_i, F_p, F_q, ϕ)

procedure RESOLVE(F_p, F_q)
  $result \leftarrow 0$
  $X \leftarrow C_p \cup C_q$
  for $c_j \in X$ **do**
    for all F_k that contain c_j, where, $k \neq p, k \neq q$ **do**
      $result \leftarrow result + \gamma_{ij} r_{jk} z_k$
    end for
  end for
  if $|result| \geq \phi$ **then**
    return $\frac{result}{|result|}$
  else
    return 0
  end if
end procedure

Figure 9 outlines the design of the intrusion detection system.

Datasets

Table 2 lists some of our datasets. Each entry consists of the data source, vehicle make and model, and number of CAN frames (rounded to nearest thousandth).

Example

The decision engine's purpose is to look at all contextual CAN data and the labels of the multiple functions of the anomaly detection engine and determine if an intrusion is occurring. To that effect, we generate functions using the function design process. These functions are now part of the anomaly detection engine. We train the anomaly detection engine using the same techniques we used in all earlier experiments. We compute the Jaccard distance metric for all pairs of functions. The evidence score metric depends on the message and is computed while the decision engine is running for each message. Consider the following five functions that are generated by the function design process. Function F_1 is defined in Eq. 6 and its function graph and CAN parameters are shown in Fig. 10. Function F_1 meets the Hamiltonian cycle property.

Table 2 Training, validation, and testing data

Source	Make, Model, Year	Frames
Miller and Valasek https://illmatics.com/carhacking.html	Toyota Prius	17K
Lee et al. (2017)	Kia Soul	4600K
Dupont et al. (2019)	Opel Astra, Renault Clio	520K
Cephas Barreto (2018)	Chevrolet Agile	23K
Personal vehicle	Toyota Venza	157K
Miller and Valasek https://illmatics.com/carhacking.html	Ford Escape	13K
Personal vehicle	Chrysler Town & Country	25K

$$F_1(x_0, x_1, x_2, x_3, x_4) = \frac{x_0}{x_1} + corr(x_1, x_2) + corr(x_2, x_3) + \frac{x_3}{x_4} + w_0 * operation_0(x_4, x_0) \tag{6}$$

Function F_2 is defined in Eq. 7 and its function graph and CAN parameters are shown in Fig. 11. Function F_2 meets the Hamiltonian cycle property.

$$F_2(x_0, x_1, x_2, x_3, x_4) = corr(x_0, x_1) + \frac{x_1}{x_2} + w_1 * operation_1(x_2, x_3) + \frac{x_3}{x_4} + x_4.x_0 \tag{7}$$

Function F_3 is defined in Eq. 8 and its function graph and CAN parameters are shown in Fig. 12. Function F_3 meets the minimum degree property as the degree of vertex x_1 is 3 (number of vertices is 5).

$$F_3(x_0, x_1, x_2, x_3, x_4) = x_2.x_0 + corr(x_4, x_1) + \frac{x_2}{x_1} + corr(x_4, x_3) + \frac{x_3}{x_1} \tag{8}$$

Function F_4 is defined in Eq. 9 and its function graph and CAN parameters are shown in Fig. 13. Function F_4 meets the minimum degree property.

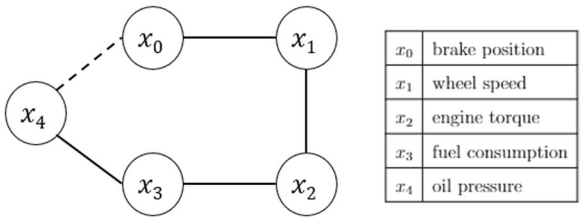

Fig. 10 Function graph and CAN parameters for F_1 (Eq. 6)

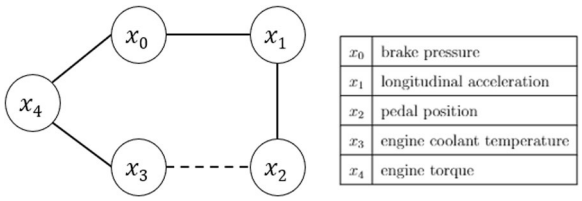

Fig. 11 Function graph and CAN parameters for F_2 (Eq. 7)

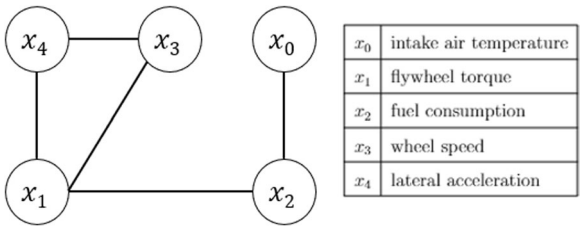

Fig. 12 Function graph and CAN parameters for F_3 (Eq. 8)

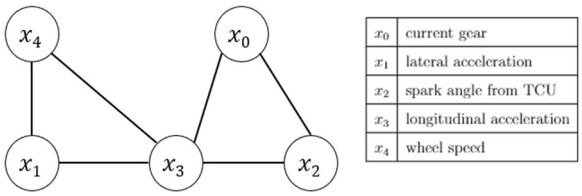

Fig. 13 Function graph and CAN parameters for F_4 (Eq. 9)

$$F_4(x_0,x_1,x_2,x_3,x_4) = corr(x_2,x_3) + \frac{x_0}{x_2} + x_3.x_1 + corr(x_0,x_3) + \frac{x_3}{x_4} + x_4.x_1 \quad (9)$$

Function F_5 is defined in Eq. 10 and its function graph and CAN parameters are shown in Fig. 14. Function F_5 meets both the Hamiltonian cycle property and the minimum degree property.

$$F_5(x_0,x_1,x_2,x_3,x_4) = \frac{x_2}{x_1} + corr(x_0,x_1) + w_1 * operation_1(x_3,x_4) + x_4.x_0 + corr(x_4,x_2) + \frac{x_3}{x_2} \quad (10)$$

Although the remote transmission frame and corresponding data frame use the same message ID, the RTR bit is used for discernment. The data frame is prioritized over the remote frame as it contains a dominant RTR bit. Since this is part of normal CAN operation and since we want to model the same behavior, we do not pre-process RTR frames in any special manner. We want to launch our attack in a way to trick the overall system. We manipulate 5 CAN parameters using the techniques mentioned earlier. The attack is designed to increase the speed of the vehicle by changing multiple values related to the speed and acceleration. Changing multiple values is an attempt to trick the system into believing that there is no intrusion occurring. We launch our attack by making the following changes:

- Multiply wheel speed by 1.2
- Set brake position to 0
- Multiply engine torque by 1.5
- Change current gear to the next lower position
- Multiply longitudinal acceleration by 1.2

The decision engine algorithm was ran on the Lee et al. (2017) dataset for Cars 7 and 8. During testing, we analyzed evidence scores of all functions per frame and determined the floor value (β) to correctly determine intrusions. For each intrusive frame, we analyzed the *decision* variable value to determine a value for θ. We tested the *RESOLVE* function by calling it on all pairs of

Table 3 Evidence score for some example messages per function

CAN message	Function				
	F_1	F_2	F_3	F_4	F_5
Request engine speed	0.71	0.63	0.23	0.25	0.82
RPM response	0.68	0.65	0.19	0.22	0.77
Change gear	0.77	0.52	0.62	0.71	0.51
Increase speed	0.82	0.49	0.59	0.75	0.48
Change steering angle	0.47	0.51	0.23	0.12	0.57
Request fuel pressure	0.39	0.27	0.3	0.37	0.32
Decrease brake pressure	0.7	0.62	0.19	0.64	0.78

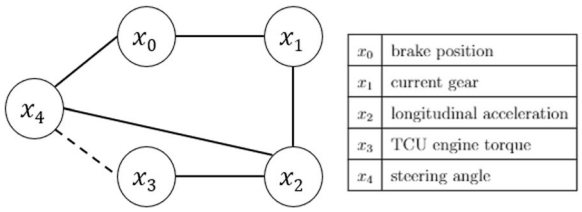

Fig. 14 Function graph and CAN parameters for F_5 (Eq. 10)

Table 4 Jaccard distance

Functions	*d*	Functions	*d*
F_1, F_2	0.8889	F_2, F_4	0.8889
F_1, F_3	0.75	F_2, F_5	1
F_1, F_4	1	F_3, F_4	0.8889
F_1, F_5	0.8889	F_3, F_5	1
F_2, F_3	1	F_4, F_5	0.75

functions and analyzing the value of ϕ for correct determination of intrusion. Table 3 lists the evidence score per function for a few of the CAN message frames in this dataset.

The Jaccard distance d between each pair of functions is shown in Table 4. The Jaccard distance ranges from 0.75 to 1.00 which implies that there are not a lot of CAN variables that are common between the functions.

The decision engine performs a detailed analysis for each frame. Consider a decision engine that determines if a frame is intrusive based on a simple majority of functions' labeling. We measured sensitivity and specificity for such a decision engine. Here are some decision engine observations:

1. All function labels unanimously agree for 7% of CAN frames.
2. Simple majority decision engine's sensitivity is 0.9132 and specificity is 0.7021.
3. Evidence score ranges from 0.12 to 0.93.
4. Jaccard distance between functions ranges from 0.75 to 1.00.
5. Minimum evidence score of 0.63 is required to correctly identify intrusive frames.
6. $\theta = 2.7$ to achieve 99% sensitivity.
7. $\phi = 0.72$ for 99% sensitivity to determine intrusion and $\phi = 0.532$ for 99% sensitivity to detect normal frame.
8. $\alpha = 0.8$ for RESOLVE function to get called and correctly determine result.

Based on the above observations, we set the values for the decision engine's thresholds. We evaluated the decision engine using Car 9 and Car 10 data and Table 5 lists these results. The results indicate that our system can detect the intrusion with high sensitivity and low false-positive rate. Additionally, comparing our decision engine with a simple majority decision engine, we can see that sensitivity has improved and the false positive rate has reduced drastically. We believe that the detailed analysis done by the decision engine per frame significantly reduces the false positive rate and improves the sensitivity. In the next section, we will compare our anomaly detection and decision engines with systems proposed by other researchers.

System analysis and discussion

Intrusion Detection Systems for CAN have been proposed by other researchers. Each system uses a different approach so it will be useful to compare and contrast results. The researchers Bozdal et al. (2021); Seo et al. (2018, 2020) test their system using the Lee et al. (2017) dataset that we have used in our research. Three types of attacks—Denial of Service, Gear Spoofing, and Fuzzy—are used to evaluate their systems. We test our intrusion detection system using the same attacks and compare the results. In addition, Bozdal et al. (2021) use another dataset (Dupont et al. 2019) that we use to evaluate their system. Bozdal et al. (2021) propose a wavelet-based intrusion detection system for vehicular networks. CAN's transmission pattern is analyzed and any changes in its behavior is marked as an anomaly. Seo et al. (2018) use generative adversarial networks to design their intrusion detection system. They train two discriminators, one for known attacks and one for unknown attacks. A generator and the discriminator for unknown attacks are trained using an adversarial process. Seo et al. (2020) propose a specification-based intrusion detection system. Security specifications are designed by extracting expected system behavior. Their focal points for expected system behavior are timing and frequency of CAN messages.

The Denial of Service attack is launched by flooding the CAN network with frames with a message identifier of 0x000. This message has the highest priority and hence no other node in the network can transmit any messages as long as the malicious frames are being sent. This results in denying service to all nodes attached to the CAN network. Since the functions from the section "Decision engine" subsection "Example" use the same data set, we use the same functions and anomaly detection engine parameters to test our system. Data from Cars 9 and 10 was attacked by inserting messages with identifier 0x000 to execute the denial of service attack. We measured

Table 5 Decision engine results

	Sensitivity	Specificity
Car 9	0.9903	0.9891
Car 10	0.9872	0.9836

sensitivity of our system and compared it with other systems. It can be seen that our system performed as well as other proposed systems. The system designed by Bozdal et al. (2021) is based on detecting attacks that are launched towards a specific message identifier. Since the denial of service attack does not target a specific message, most likely their sensitivity is not as high as others. Our system performs better than Bozdal, Samie, Jennions and comparable to the others in terms of sensitivity. Our false positive rate is better than Olufowobi et al. and comparable to others. With respect to both sensitivity and specificity considered together, our system outperforms three systems. Results are summarized in Table 6.

The Gear Spoofing attack targets a specific message identifier. Recall that remote frames are used to request data. When an ECU receives a remote frame for data that it is responsible for, then this ECU is expected to transmit a response data frame immediately. In this attack, a rogue ECU responds to the current gear remote frame request with malicious response. We tested our system by injecting test data with spoofed current gear response frames, the same way in which the other systems executed this attack. Our system performs better than Seo, Song, Kim and comparable to the other two in terms of sensitivity. The results for specificity mimic the ones for the denial of service attack. Overall, our system performs better in terms of sensitivity and specificity considered together. Results are summarized in Table 7.

Fuzzy attack is a type of injection attack. CAN frames, with multiple targeted message identifiers, containing random data are transmitted on the bus by the malicious node. Unintended and dangerous consequences such as shaking of the steering wheel, unexpected gear shifts, blinking indicators on the panel, etc. Lee et al. (2017) occur as a result of such an attack. We repeated the same experiment on our system and compared the results (Table 8). For the fuzzy attack, our system performs comparable to Seo, Song, Kim and Olufowobi et al. in terms of sensitivity. However, sensitivity performance of Bozdal, Samie, Jennions is not good as the fuzzy attack targets random message identifiers and their system is designed to tackle attacks on specific message identifiers. Similar to the gear spoofing attack, the false positive rate of Olufowobi et al. is extremely high. Our false positive rate is comparable to Bozdal, Samie, Jennions.

Table 6 Denial of service attack

System	Sensitivity	Specificity
Proposed system	0.9952	0.9943
Seo et al. (2018)	0.9960	–
Olufowobi et al. (2020)	1.000	0.8929
Bozdal et al. (2021)	0.9415	0.9962

Table 7 Gear spoofing attack

System	Sensitivity	Specificity
Proposed system	0.9872	0.9928
Seo et al. (2018)	0.9650	–
Olufowobi et al. (2020)	0.9702	0.4256
Bozdal et al. (2021)	0.9845	0.9993

None of the peer-reviewed intrusion detection systems make a distinction between anomalies and intrusions. We believe that our decision engine improves the performance of the overall system by identifying intrusions with high sensitivity and reducing the number of false positives. In most approaches, presence of an anomaly is treated as an intrusion whereas our approach performs a detailed analysis to determine if an intrusion has occurred. Rare erratic driving behavior can cause irregular CAN traffic. Any security system should be able to handle such situations by reducing the number of false positives that will be triggered by such behavior.

The high false positive rate of Olufowobi et al. is most likely due to the fact that their system treats any anomalous frame as an intrusion. There is no further analysis of the frame to check if the anomaly is caused due to an intrusion or another reason such as rare erratic driver behavior or sudden change in driving conditions. Our function design algorithm takes into account multiple attack and deception techniques adopted by hackers. The proposed properties of our function graphs ensure that our anomaly detection system cannot be spoofed easily by manipulating multiple CAN parameters.

Limitations

We recognize that there are certain limitation to our proposed solution. We were limited by the amount of data that we had access to. We evaluated our solution on data from multiple major vehicle manufacturers but not from all of the major manufacturers. In certain datasets, we did not have access to as many CAN parameters as others thereby reducing the number of functions for the anomaly detection engine.

Table 8 Fuzzy attack

System	Sensitivity	Specificity
Proposed system	0.9948	0.9968
Seo et al. (2018)	0.9950	–
Olufowobi et al. (2020)	0.9958	0.5130
Bozdal et al. (2021)	0.8339	0.9977

Future work

We believe that selecting functions from a pool of functions for the anomaly detection engine will further strengthen our system. A random function selection algorithm that runs periodically, similar to a key refresh process in cryptography, will also be an improvement. Continuous training of the anomaly detection engine will ensure that the system's will perform as intended. We believe that overnight training, while the car is idle, will help maintain the performance of our system.

Conclusion

Designing, implementing, and enforcing security countermeasures in vehicles is a big challenge. As outlined earlier, security requirements and corresponding resource constraints, lack of security in the network design, and multiple attack surfaces are a few of the challenges for a robust vehicle security system. Our goal was to design a light-weight, almost real-time, and extensible security system for intra-vehicle networks. We designed unique measurements to be used by the decision engine. The evidence score underscores the importance of the relation between CAN messages types and CAN sensors. The evidence score uses a ratio that is used to determine the influence of CAN sensors within each anomaly detection engine function. We also propose a mechanism to resolve conflicts between anomalous frame markings of various functions. The Jaccard distance plays an important role in this process. Our decision engine presented in this paper and our anomaly detection engine presented in our earlier papers together deliver a vehicle security solution that meets all the outlined goals and requirements.

Acknowledgements
Not available.

Author contributions
Dr. Vinayak Tanksale is the sole author of the manuscript.

Funding
Not available.

Availability of data and materials
Most data used in the paper is publicly available and has been referenced accordingly. The rest of the data will be made available upon reasonable request.

Code availability
Not available.

Declarations

Ethical approval and consent to participate
Not available.

Consent for publication
Not available.

Competing interests
None of the authors have any competing interests in the manuscript.

Received: 31 July 2023 Accepted: 5 November 2023
Published online: 02 February 2024

References

Bozdal M, Samie M, Jennions IK (2021) Winds: a wavelet-based intrusion detection system for controller area network (can). IEEE Access 9:58621–58633. https://doi.org/10.1109/ACCESS.2021.3073057

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297. https://doi.org/10.1007/BF00994018

Creech G, Hu J (2014) A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns. IEEE Trans Comput 63(4):807–819

Cristianini N, Shawe-Taylor J, Shawe-Taylor DCSRHJ, Books24x7 I, Press CU (2000) An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, USA. https://books.google.com/books?id=_PXJn_cxv0AC

Desta AK, Ohira S, Arai I, Fujikawa K (2020) Id sequence analysis for intrusion detection in the can bus using long short term memory networks. In: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp 1–6

Dibaei M, Zheng X, Jiang K, Abbas R, Liu S, Zhang Y, Xiang Y, Yu S (2020) Attacks and defences on intelligent connected vehicles: a survey. Digit Commun Netw 6(4):399–421. https://doi.org/10.1016/j.dcan.2020.04.007

Dibaei M, Zheng X, Jiang K, Maric S, Abbas R, Liu S, Zhang Y, Deng Y, Wen S, Zhang J, Xiang Y, Yu S (2019) An overview of attacks and defences on intelligent connected vehicles. arXiv. https://doi.org/10.48550/ARXIV.1907.07455. arXiv:1907.07455

Dupont G, Lekidis A, Hartog JJ, Etalle SS (2019) Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2. 4TU.ResearchData. https://doi.org/10.4121/uuid:b74b4928-c377-4585-9432-2004dfa20a5d . https://data.4tu.nl/articles/dataset/Automotive_Controller_Area_Network_CAN_Bus_Intrusion_Dataset/12696950/2

Fröschle S, Stühring A (2017) Analyzing the capabilities of the can attacker. In: Foley SN, Gollmann D, Snekkenes E (eds) Computer Security - ESORICS 2017. Springer, Cham, pp 464–482

Hoang XD, Hu J, Bertok P (2009) A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference. J Netw Comput Appl 32(6):1219–1228. https://doi.org/10.1016/j.jnca.2009.05.004

Hoppe T, Dittman J (2007) Sniffing/replay attacks on can buses: a simulated attack on the electric window lift classified using an adapted cert taxonomy. In: Proceedings of the 2nd workshop on embedded systems security (WESS), pp 1–6

Hossain MD, Inoue H, Ochiai H, Fall D, Kadobayashi Y (2020) Long short-term memory-based intrusion detection system for in-vehicle controller area network bus. In: 2020 IEEE 44th annual computers, software, and applications conference (COMPSAC), pp 10–17

Huang F, Li X, Yuan C, Zhang S, Zhang J, Qiao S (2022) Attention-emotion-enhanced convolutional lstm for sentiment analysis. IEEE Trans Neural Netw Learn Syst 33(9):4332–4345. https://doi.org/10.1109/TNNLS.2021.3056664

Javed AR, Rehman SU, Khan MU, Alazab M, Reddy TG (2021) Canintelliids: detecting in-vehicle intrusion attacks on a controller area network using cnn and attention-based gru. IEEE Trans Netw Sci Eng 8(2):1456–1466. https://doi.org/10.1109/TNSE.2021.3059881

Jin S, Chung J-G, Xu Y (2021) Signature-based intrusion detection system (ids) for in-vehicle can bus network. In: 2021 IEEE international symposium on circuits and systems (ISCAS), pp 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401087

Kang M-J, Kang J-W (2016) Intrusion detection system using deep neural network for in-vehicle network security. PLoS ONE 11(6):1–17. https://doi.org/10.1371/journal.pone.0155781

Katragadda S, Darby PJ, Roche A, Gottumukkala R (2020) Detecting low-rate replay-based injection attacks on in-vehicle networks. IEEE Access 8:54979–54993. https://doi.org/10.1109/ACCESS.2020.2980523

Kim K, Kim JS, Jeong S, Park J-H, Kim HK (2021) Cybersecurity for autonomous vehicles: review of attacks and defense. Comput Secur 103:102150. https://doi.org/10.1016/j.cose.2020.102150

Kleberger P, Olovsson T, Jonsson E (2011) Security aspects of the in-vehicle network in the connected car. In: 2011 IEEE intelligent vehicles symposium (IV), pp 528–533. https://doi.org/10.1109/IVS.2011.5940525

Larson UE, Nilsson DK (2008) Securing vehicles against cyber attacks. In: Proceedings of the 4th annual workshop on cyber security and information intelligence research: developing strategies to meet the cyber security and information intelligence challenges ahead. CSIIRW'08. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1413140.1413174

Laszka A, Abbas W, Sastry SS, Vorobeychik Y, Koutsoukos X (2016) Optimal thresholds for intrusion detection systems. In: Proceedings of the symposium and bootcamp on the science of security. HotSos'16. Association for Computing Machinery, New York, NY, USA, pp 72–81. https://doi.org/10.1145/2898375.2898399

Lee S, Jo HJ, Cho A, Lee DH, Choi W (2022) Ttids: Transmission-resuming time-based intrusion detection system for controller area network (can). IEEE Access 10:52139–52153. https://doi.org/10.1109/ACCESS.2022.3174356

Lee H, Jeong SH, Kim HK (2017) Otids: a novel intrusion detection system for in-vehicle network by using remote frame. In: 2017 15th annual conference on privacy, security and trust (PST), vol. 00, pp 57–5709. https://doi.org/10.1109/PST.2017.00017

Li W (2004) Using genetic algorithm for network intrusion detection. In: Proceedings of the United States Department of Energy Cyber Security Group 2004 Training Conference, pp 24–27

Lv Z, Lou R, Singh AK (2021) Ai empowered communication systems for intelligent transportation systems. IEEE Trans Intell Transp Syst 22(7):4579–4587. https://doi.org/10.1109/TITS.2020.3017183

Marchetti M, Stabili D (2017) Anomaly detection of can bus messages through analysis of id sequences. In: 2017 IEEE intelligent vehicles symposium (IV), pp 1577–1583. https://doi.org/10.1109/IVS.2017.7995934

Meyer S (2019) Vehicle hacking, the new data security threat. Accessed 17 Mar 2019. https://www.cpomagazine.com/cyber-security/vehicle-hacking-the-new-data-security-threat/

Miller C, Valasek C (2015) Remote exploitation of an unaltered passenger vehicle. In: Proceedings of the Black Hat USA 2015

Miller C, Valasek C. Car hacking data. https://illmatics.com/carhacking.html

Minawi O, Whelan J, Almehmadi A, El-Khatib K (2020) Machine learning-based intrusion detection system for controller area networks. In: Proceedings of the 10th ACM symposium on design and analysis of intelligent vehicular networks and applications. DIVANet'20. Association for Computing Machinery, New York, NY, USA, pp 41–47. https://doi.org/10.1145/3416014.3424581

Mukkamala S, Sung AH (2003) Detecting denial of service attacks using support vector machines. In: The 12th IEEE international conference on fuzzy systems, 2003. FUZZ'03., vol. 2, pp 1231–12362. https://doi.org/10.1109/FUZZ.2003.1206607

Nam M, Park S, Kim DS (2021) Intrusion detection method using bi-directional gpt for in-vehicle controller area networks. IEEE Access 9:124931–124944. https://doi.org/10.1109/ACCESS.2021.3110524

NCCIC/ICS-CERT: CAN Bus Standard Vulnerability (2017). Accessed 10 Aug 2018. https://www.cisa.gov/news-events/ics-alerts/ics-alert-17-209-01

Nguyen TTT, Armitage G (2008) A survey of techniques for internet traffic classification using machine learning. IEEE Commun Surv Tutor 10(4):56–76. https://doi.org/10.1109/SURV.2008.080406

Nie L, Ning Z, Wang X, Hu X, Cheng J, Li Y (2020) Data-driven intrusion detection for intelligent internet of vehicles: a deep convolutional neural network-based method. IEEE Trans Netw Sci Eng 7(4):2219–2230. https://doi.org/10.1109/TNSE.2020.2990984

Olufowobi H, Young C, Zambreno J, Bloom G (2020) Saiducant: specification-based automotive intrusion detection using controller area network (can) timing. IEEE Trans Veh Technol 69(2):1484–1494. https://doi.org/10.1109/TVT.2019.2961344

Oruh J, Viriri S, Adegun A (2022) Long short-term memory recurrent neural network for automatic speech recognition. IEEE Access 10:30069–30079. https://doi.org/10.1109/ACCESS.2022.3159339

Peng S, Hu Q, Chen Y, Dang J (2015) Improved support vector machine algorithm for heterogeneous data. Pattern Recognit 48(6):2072–2083. https://doi.org/10.1016/j.patcog.2014.12.015

Sekar R, Gupta A, Frullo J, Shanbhag T, Tiwari A, Yang H, Zhou S (2002) Specification-based anomaly detection: a new approach for detecting network intrusions. In: Proceedings of the 9th ACM conference on computer and communications security. CCS '02. ACM, New York, NY, USA, pp 265–274. https://doi.org/10.1145/586110.586146

Seo E, Song HM, Kim HK (2018) Gids: Gan based intrusion detection system for in-vehicle network. In: 2018 16th annual conference on privacy, security and trust (PST), pp 1–6. https://doi.org/10.1109/PST.2018.8514157

Silveira Barreto CA (2018) OBD-II datasets. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/83155 . https://www.kaggle.com/dsv/83155

Sung AH, Mukkamala S (2003) Identifying important features for intrusion detection using support vector machines and neural networks. In: 2003 Proceedings of the symposium on applications and the internet, pp 209–216. https://doi.org/10.1109/SAINT.2003.1183050

Takefuji Y (2018) Connected vehicle security vulnerabilities [commentary]. IEEE Technol Soc Mag 37(1):15–18. https://doi.org/10.1109/MTS.2018.2795093

Tanksale V (2019) Intrusion detection for controller area network using support vector machines. In: 2019 IEEE 16th international conference on mobile ad hoc and sensor systems workshops (MASSW), pp 121–126. https://doi.org/10.1109/MASSW.2019.00032

Tanksale V (2020a) Controller area network security requirements. In: 2020 International conference on computational science and computational intelligence (CSCI), pp 157–162. https://doi.org/10.1109/CSCI51800.2020.00034

Tanksale V (2020b) Anomaly detection for controller area networks using long short-term memory. IEEE Open J Intell Transp Syst 1:253–265. https://doi.org/10.1109/OJITS.2020.3043066

Tanksale V (2021) Design of anomaly detection functions for controller area networks. IEEE Open J Intell Transp Syst 2:312–321. https://doi.org/10.1109/OJITS.2021.3104495

Tanksale V (2023) Gated recurrent units for intrusion detection. In: 2023 IEEE IAS global conference on emerging technologies (GlobConET), pp 1–5. https://doi.org/10.1109/GlobConET56651.2023.10149912

Tavallaee M, Lu W, Iqbal SA, Ghorbani AA (2008) A novel covariance matrix based approach for detecting network anomalies. In: 6th annual communication networks and services research conference (cnsr 2008), pp 75–81. https://doi.org/10.1109/CNSR.2008.80

Taylor A, Leblanc S, Japkowicz N (2016) Anomaly detection in automobile control network data with long short-term memory networks. In: 2016 IEEE international conference on data science and advanced analytics (DSAA), pp 130–139

Vapnik V (2013) The nature of statistical learning theory. Springer, New York

Vector: CANoe. Vector. https://www.vector.com/int/en/products/products-a-z/software/canoe/

Woo S, Jo HJ, Kim IS, Lee DH (2016) A practical security architecture for in-vehicle can-fd. IEEE Trans Intell Transp Syst 17(8):2248–2261. https://doi.org/10.1109/TITS.2016.2519464

Wu W, Kurachi R, Zeng G, Matsubara Y, Takada H, Li R, Li K (2018) Idh-can: a hardware-based id hopping can mechanism with enhanced security for automotive real-time applications. IEEE Access 6:54607–54623. https://doi.org/10.1109/ACCESS.2018.2870695

Wu W, Li R, Xie G, An J, Bai Y, Zhou J, Li K (2020) A survey of intrusion detection for in-vehicle networks. IEEE Trans Intell Transp Syst 21(3):919–933. https://doi.org/10.1109/TITS.2019.2908074

Xiong W, Gülsever M, Kaya KM, Lagerström R (2019) A study of security vulnerabilities and software weaknesses in vehicles. In: Askarov A, Hansen RR, Rafnsson W (eds) Secure IT systems. Springer, Cham, pp 204–218

Zhou J, Joshi P, Zeng H, Li R (2019) Btmonitor: bit-time-based intrusion detection and attacker identification in controller area network. ACM Trans Embed Comput Syst 18(6):3362034. https://doi.org/10.1145/3362034

Publisher's Note