A Real-time In-Vehicle Network Testbed for Machine Learning-based IDS Training and Validation

Hesamaldin Jadidbonab[1], Andrew Tomlinson[1], Hoang Nga Nguyen[1], Trang Doan[2] and Siraj Ahmed Shaikh[1]

[1]*Systems Security Group, Centre for Future Transport and Cities (CFTC), Coventry University, Coventry, UK*
[2]*CEM School, EEC, Coventry University, Coventry, UK*

Abstract

Modern vehicles are built of onboard networked computers, known as Electronic Control Units, to realise a range of functionality and features. Due to their connectivity, they are vulnerable to cyberattacks through various vectors including wired (e.g., CAN, Lin) and wireless (e.g., Bluetooth, WiFi) communications. To facilitate building and validating security measures against these attacks, it is vital to replicate the on-board networks and their components, along with a connected external environment representing required use cases and driving scenarios in an instrumental and controllable environment. In this paper, we propose a multi-component testbed representing a flexible and functional in-vehicle architecture for training and validating machine learning-based Intrusion Detection Systems (IDS).

Keywords

Automotive security, testbed, machine learning

1. Introduction

Modern vehicles comprise multiple networked computers, known as Electronic Control Units (ECUs), to realise a range of functionality and features such as driving and powertrain control, connectivity, sensing and body modules. Such ECUs are interconnected through onboard networks, including typically a data bus known as the Controller Area Network (CAN). As such modern vehicles are an example of a Cyber-Physical System (CPS).

Due to a combination of connectivity (over various interfaces) and complexity (of design and features), modern vehicles are also subject to cyber attacks. With trends towards interconnections of sensors, actuators and devices, modern cars now often possess interfaces enabling wired (e.g., USB) or wireless (e.g., Bluetooth, WiFi, Cellular) communication with the outside world. Therefore, the various computing systems embedded in modern cars can no longer be considered as a closed network, and opportunities for cyberattacks targeting the embedded automotive networks have become a reality. It has been already shown that a cyber attack can seriously impact the safety of a vehicle if packets are successfully sent on the vehicle's internal wired network [1]. A number of articles have investigated the potential security risks with modern vehicles focusing on specific attack goals and vectors (e.g. [2, 3, 4, 5, 6, 1, 7, 8, 9, 10, 11]).

AI-CyberSec 2021: Workshop on Artificial Intelligence and Cyber Security, December 14, 2021, Cambridge, England

✉ abc@coventry.ac.uk (H. Jadidbonab); abc@coventry.ac.uk (A. Tomlinson); abc@coventry.ac.uk (H. N. Nguyen); abc@coventry.ac.uk (T. Doan); abc@coventry.ac.uk (S. A. Shaikh)

CEUR Workshop Proceedings (CEUR-WS.org)

Counteracting automotive cybersecurity threats requires the investigations of threat models, counter measures, detection methods and resolution options. This presents a number of challenges specific to the automotive field. First, it is difficult to conduct experimentation on real systems. Conducting tests on live vehicles is costly and might result in vehicle damage or risks to property and life. It also presents challenges regarding ethics, insurance and legality, and the researchers may not have access to rolling roads or private tracks. Given such challenges, it is understandable why research into automotive cyberattack detection has tended to be done using offline experiments on previously captured data (e.g., [12], [13], [14], [15]). A second challenge concerns ensuring the scope and validity of testing. Testing across a representative range of conditions and operating scenarios would require capturing vast amounts of data, or controlling factors such as the weather, environmental conditions and traffic flows. These challenges may be compounded when testing attack detection methods employing machine learning, where the behaviours of the methods might not be fully described [16]. Moreover, vigorous testing needs to account for system interactions that might not be determinable from studying behaviours of the vehicle components in isolation.

To facilitate building and validation of security measures as well as to assure a secure automotive CPS, it is vital to replicate the on-board network and a series of components, along with a connected external environment representing required use cases and driving scenarios in an instrumental and controllable environment. Such a setting will be useful for:

- assessing the resilience of components, subsystems and the whole system, to cyber attacks, as part of a mix of commodity and bespoke multi-vector attacks;
- tracing and auditing low-level behaviours for evidence-driven assessment;
- amassing datasets for industry use and wider experimental-led collaborations with academic and developer communities.

While several in-vehicle network testbeds [17, 18, 5, 19, 20, 21, 22, 23, 24] have been proposed recently, they only focus on the in-vehicle network components and ignore providing multiple realistic driving scenarios. This means any dataset generated by these testbeds is not suitable for training and validating machine learning-based Intrusion Detection Systems (IDS) which are used to rapidly detect cyberattacks on in-vehicle networks [25].

In this paper, we propose a multi-component testbed representing a flexible and functional in-vehicle architecture for real environment trials to train, test, validate and demonstrate IDS solutions. Our contribution is the logical design and implementation of a real-time testbed that allows cybersecurity researchers and engineers an in-depth security evaluation of in-vehicular network components. This testbed will provide various features including:

- A CAN Bus representing a full-scale functional in-vehicle network;
- Support for generating realistic driving scenarios;
- A plug-and-play facility for testing and evaluation of hardware such as hacking devices, telematics, sensors, infotainment, in-cabin and body modules;
- Support for integrating security solutions for testing and evaluation purposes.

The paper is structured as follows. Section 2 presents and evaluates existing testbeds for automotive cybersecurity testing. It focuses on their configuration, hardware and software

Table 1
Types of Automotive Cybersecurity Testbeds

Name of Testbed	Test Platform	Reference
Open Car Testbed and Network Experiments (OCTANE)	Hybrid	[17]
Resistant Automotive Miniature Network	Hybrid	[23]
Design, Implementation, Validation and Implications of a Real-world V2I Prototype Testbed	Hybrid	[22]
Developing a QRNG ECU for automotive security: Experience of testing in the real-world	Hybrid	[24]
Testbed for Security Analysis of Modern Vehicle Systems	Hybrid	[21]
Mobile Testing Platform	Hardware	[5]
Hardware-in-loop based Automotive Embedded Systems Cybersecurity Evaluation Testbed	Simulator	[19]
Portable Automotive Security Testbed with Adaptability (PASTA)	Simulator	[20]
A Cyber Assurance Testbed for Heavy Vehicle Electronic Controls	Simulator	[18]

packages. It also discusses existing approaches to CAN attack detection and outlines the challenges specific to detection on CAN networks. Then, we recall the CAN-BUS protocol and present the relevant software and hardware used to implement a full-scale testbed including vehicle simulators and automotive in-vehicle communications in Section 3. The implementation detail of our testbed is presented in Section 4. Section 5 demonstrates the integration of the testbed with an exemplary IDS, which shows how the testbed can be used in future research. The IDS is used to detect typical attacks to in-vehicle communication CAN. Finally, the paper is concluded in Section 6.

2. Related work

2.1. Automotive Cybersecurity Testbeds

Testbeds can generally be categorised into three different types: simulation-based, hardware-based, and hybrid. Simulation-based testbeds rely solely or substantially on software to simulate the behaviour of ECUs and in-vehicle networks. Since they do not include real cyber-physical components, simulation-based testbeds are generally cheaper to build and provide a safer environment for the testers. Hardware-based testbeds, on the other hand, include real or emulated hardware components. As opposed to software-based testbeds, hardware-based testbeds enable testers to study interactions between components through physical inputs and outputs. Hybrid testbeds include both software and hardware components, offering the strengths of simulation-based and hardware-based testbeds. Table 1 presents an overview of the surveyed testbeds, indicating whether they are simulation-based, hardware-based, or hybrid. The mobile testing platform [5] is the only testbed that uses real physical components and a vehicle (go-cart) for investigating cybersecurity threats. OCTANE, Resistant Automotive Miniature Network, Realworld V2I Prototype Testbed and Developing a QRNG ECU for automotive security are hybrid testing environments. All other testbeds rely on virtual/software components only. Table 1 lists the most recent and relevant testbeds (introduced in the cited articles) to date. They are categorised initially based on their platform structure as previously noted.

Testbeds should also be evaluated based on their various characteristics, such as adaptability, portability, fidelity, cost and types of attacks, attack surfaces, attack targets, and communication protocols supported. This characterisation is essential to choose an appropriate platform with regard to the testing scenarios.

To facilitate the testbeds for more thorough and optimised testing, a generic testing framework based on existing and emerging standards for cybersecurity engineering has been recently introduced [26]. This framework enables the automotive cybersecurity testbeds to conduct automated security tests and a systematic process for verifying and validating automotive cybersecurity based on ISO/SAE DIS 21343. This framework is also flexible to allow extension beyond its common structure to utilise additional toolsets.

2.2. Automotive Cybersecurity Attacks and Detection

Attacks to the CAN network entail the attacker changing the packet broadcasts [27]. An attack might disrupt the packet broadcast rates, for example, to stage a DoS (Denial of Service) by flooding the network with fabricated packets, thus incapacitating the vehicle or some function. Although immediately disruptive, such crude attacks are considered relatively easy to detect. Potentially more difficult to detect are attacks that seek to control some functionality of the car or meaningfully alter information presentation. For example, by broadcasting fabricated packets with misleading data payloads. Such attacks might force the car into dangerous situations, systematically change the car performance or efficiency, or manipulate for financial gain or dissatisfaction [28]. These attacks might entail the fabrication of legitimate-looking packets from attack systems masquerading as legitimate control units [4, 29, 30].

A well-publicised attack [4] on a Jeep Cherokee required the attackers to broadcast fabricated packets that contained plausible, but altered speed data, thus fooling the car systems into believing the car was travelling more slowly – this, in turn, allowed other functions, such as park-assist, to be activated. A problem in detecting such an attack is that the manipulated data values remain within legitimate bounds, so might not look obviously malicious [31].

Two common approaches to detecting attacks on any computer network, including the CAN, are signature approaches and anomaly approaches [32, 33]. Signature approaches compare traffic patterns against rule bases. Although these approaches can be accurate, they assume the ability to determine, encode and distribute evolving attack models. This can be problematic for automotive CAN systems because (i) the data derivations are usually secret to the manufactures; and (ii) cars have diverse locations, usage patterns and lifecycles making it difficult to maintain and update signature databases; and, iii) the attack scenarios are still emerging. Signature approaches for detecting CAN attacks are thus generally seen as less favourable[34]. Anomaly approaches seek to use statistical or machine learning algorithms to identify data outliers indicative of attacks. They tend to be more prone to false detections, but hold the potential to detect novel attacks, are less reliant on maintaining updates, and can be applied generically across car models. But they assume that ample data is available for training and testing.

As discussed in Section 1, generating test data and testing detection methods presents challenges when attempted on real vehicles. This has led researchers to attempt to develop and validate security methods using data from previously captured data logs, with the log manipulated offline to fabricate the likely results of an attack, and then used offline for testing (e.g., [12], [13], [14], [15]). Such methods are safer than attacking a real, driven vehicle, and give the researcher control over the attack manifestation on the data. However, they limit the researcher to only testing scenarios that have previously been captured. Also, the simulated attack data may be difficult to fabricate; and packet timings and interactions between CAN connected systems or ECUs are difficult to accurately mimic in the fabricated data. A further complication is presented by the packet arbitration and error confinement processes of the CAN protocol (discussed in Section 3.1). These govern the priority, curtailment and rebroadcast of CAN packets, and for the validity of testing would need replicating in fabricated logs.

3. Background

In this section, we recall the CAN-BUS protocol, the main in-vehicle communication of the testbed. We also briefly review the two simulators employed to realise driving scenarios and the CAN-BUS network activity. We also outline an example IDS approach which we later use as a demonstration of integration into our testbed.

3.1. CAN-BUS - an in-vehicle communication protocol

Internal vehicle networks which provide the ECUs communication are based on the CAN standard. The CAN 2.0 and SAE J1939 protocols are the two broadly used application protocols for passenger and heavy-duty vehicles, respectively. Features that make CAN desirable for automotive applications are its range of adoption, its lower cost and lower complexity (compared with FlexRay, TTE, e.t.c.), its robustness, and its bounded delays.

The physical structure of CAN comprises a shared physical medium to which multiple nodes are connected. These nodes can transmit/receive data over unshielded twisted pair cables. If any node transmits a logical zero then the bus only remains in state zero, even if all the other nodes transmit a logical one. As CAN is an event-based communication protocol, it requires a synchronised start of communication for all of its nodes. Each message is encapsulated in frames which can be transmitted periodically or occasionally between nodes. In general, the CAN frame includes seven distinguished sections as depicted in Figure 1. Most importantly, the arbitration holds an 11-bit Identifier and is used to determine the priority of a message, with smaller Identifiers having higher priority [35].

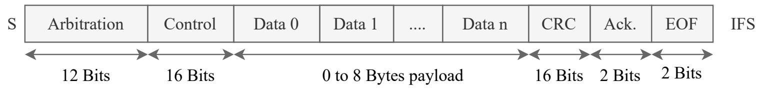


Figure 1: CAN data frame format

The CAN-BUS protocol specification is able to format error frames and overload frames. Each of these frames includes two sections of flag and delimiter field. Transmission of an error frame interrupts the normal frames, thus indicating the error condition to all nodes. Overload frames are used for flow control by the receiver [36].

Despite the popularity of the CAN, several related vulnerabilities and attacks have been reported [5]. There is no message authentication mechanism in the CAN, and most of its vulnerabilities are associated with this. This means that the protocol can not associate a message with its sender and thus fails to distinguish between a legitimate and a malicious message. Additionally, the absence of any encryption standard in the CAN protocol reduces the message's integrity and confidentiality [37]. Implementation of an encryption algorithm is also challenging due to the low computational power of the car's ECUs along with other limitations such as lack of backward compatibility, cost and insufficient implementation details, among others [38]. These reasons make it convenient for any device connected to the same physical CAN bus to listen and intercept messages.

3.2. CANoe - an in-vehicle network simulator

CANoe [39], from Vector, is a software application primarily used by automotive manufacturers and ECU suppliers for developing, simulating, and testing on-board networks and individual ECUs. An advantage of CANoe is the faithful simulation of on-board networks in terms of timing accuracy and the support of multiple vehicle bus systems. It creates virtual ECUs using Communication Application Programming Language (CAPL), an event focused language developed by Vector Informatik to support their CAN toolsets [40]. Additionally, CANoe allows hybrid network configurations where one part of an on-board network is virtually simulated while the other is physically set up.

In the testbed, the physical CAN bus network of the demonstrator is interfaced with the simulated one in CANoe via a CAN bus interface device.

CANoe enables communication with 3rd party applications to provide external inputs and outputs for the simulation. In particular, CANoe uses Fast Data Exchange protocol (FDX) which enables bi-directional data exchange with other applications. A vehicle simulated in CARLA provides car physical data which are sent to CANoe simulator by FDX. To maintain timing in CARLA and CANoe, the timers in Python program is deployed. These data provide inputs for virtual ECUs simulated in CANoe to function and generate CAN traffic on the simulated network as well as the physical one. Conversely, any CAN message occurring on the network and aiming at creating effects on the simulated vehicles are translated into control signals by CANoe and forwarded to CARLA also by FDX. They enable CARLA to simulate the corresponding effects on the simulated car. Therefore, CARLA functionalities are first to generate realistic car physical data and secondly to realise the effects of received CAN messages from CANoe on the simulated car.

CANoe enables the researchers with tools for real-time CAN bus network simulation. Various options such as message timing, easy observation of data traffic and comprehensive network analysis, time-synchronous analysis of multiple buses, Interactive sending of predefined messages and various types of stimulation of network traffic provide full control on the simulation; hence more realistic scenarios can be implemented.

3.3. CARLA - a car simulator

CARLA [41] is an open-source simulator for autonomous vehicles. The advantages of CARLA include autonomous driving baselines, Robot Operating System (ROS) integration, multiple repeatable and customisable traffic scenarios, flexible API (to control all aspects related to the simulation, including traffic generation, pedestrian behaviors, weathers, sensors), maps generation, traffic scenarios simulation and a wide range of sensors and sensor data (such as images, depth, semantics, Lidar). Furthermore, simulated cars can also be driven manually using car-like inputs such as steering wheels and foot pedals. Data from CARLA can be easily exchanged with 3rd party applications via Python programming. All required libraries come with CARLA. CARLA runs on a separate PC.

3.4. Example CAN bus IDS classifier

Later in this paper we show how IDS methods can be integrated with the testbed for testing, demonstration and validation. Our goal here is not to present a detailed explanation of any particular IDS method, but we use for demonstration a representative IDS example presented in [42], where the reader can find a full explanation of its methods and data parsing process.

The approach assumes that the role of each CAN packet is unknown to the IDS developer, which is likely given the manufacturer's secrecy regarding the CAN data together with CAN data variations specific to the make and model of cars. The approach first determines a cluster of CAN data fields which might be combined to provide a situational context. To achieve this, likely sensor data fields (including fields that might be composite) are determined by considering the statistical profiles and pairwise characteristics of the individual data fields from all the CAN packets identified. Next, a clustering algorithm is applied to determine whether the identified sensor data fields might be split into functional clusters.

Having determined a candidate cluster, a buffer is used to store the most recent data value from each field in the cluster. Thus, at each packet broadcast, the current data values from the cluster are examined together as a snapshot. This presents the option for using the cluster as a contextual gauge for evaluating each of its components at the current point in time. Thus the cluster can be used for anomaly detection using a classifier, such that malicious changes in some of the data fields might appear anomalous according to the context described by the whole cluster.

A one-class classifier is used since this might determine outlier boundaries, even where the anomaly data is not yet available, or cannot be obtained, or is difficult to fully determine. Such one-class classifiers have been considered pertinent to automotive networks [43, 13]. For our demonstration, we use a One-class Support Vector Machine (OCSVM) for the classifier. OCSVMs try to find the largest margin between the projected training data and its origin, thus demarking data-dense regions in input space [44]. Target instances are thus classified according to their position relative to the margin. For this demonstration, we retain the parameters found [42] to be optimal with comparable CAN networks tested on other vehicles (Radial Basis Function, Nu 0.015 and gamma 0.5). For classifier training data, multiple snapshots are collated and processed offline. During online testing, the current snapshot is tested in real-time against the trained classifier.

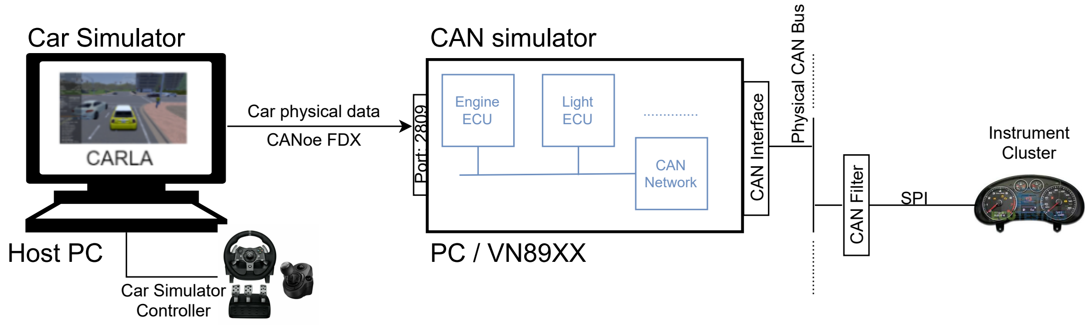


Figure 2: Test-bed configuration

4. Testbed

This section presents our testbed which facilitates building and validation of security measures on in-vehicle networks. First, the logical design of the testbed is introduced. Then, we will provide the detail of the testbed implementation.

4.1. Logical Design

The section 3 outlined the basic requirements that any automotive testbed should satisfy regarding vehicle-security technology development and research application. Logically, the demonstrator consists of a car simulator, an on-board network simulator, a physical network and a data storage service. Figure 3 depicts the logical setup of the testbed. A car simulator is used to generate car physical data for the on-board network simulator. These include engine information such as engine rev speed, vehicle velocity, gear, acceleration throttle, and brake status. The on-board network simulator is used to simulate the operation of an on-board network.

The simulated network contains several virtual ECUs to mirror those inside an actual car such as Engine ECU, Transmission ECU and Light ECUs. The virtual ECUs operate by leveraging the car physical data from the car simulator to generate data traffic on the simulated network. A physical network is attached to the simulated network via a suitable interface.

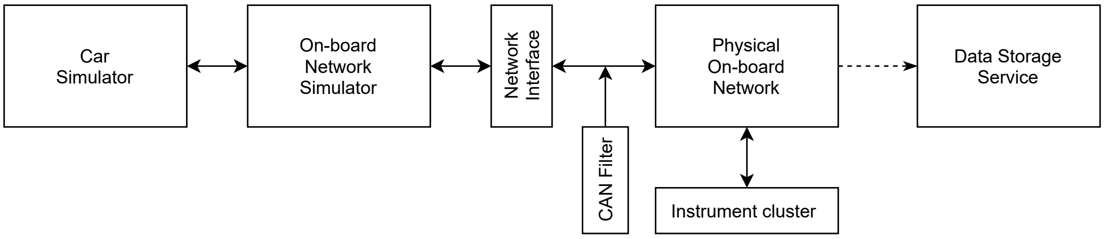


Figure 3: Logical setup of the demonstrator

In the next section, the components of the logical setup are realised by a physical setup as illustrated in Figure 2.

4.2. Implementation

In this section, implementation details of a testbed that can represent a vehicle IT network are described. The proposed automotive testbed includes a car simulator, an on-board network simulator, a physical network, a real car's instrument cluster and a spoofing hardware.

Figure 2 provides an overview of the testbed architecture. Most of the vehicle architecture and its CAN bus network is realised within a virtual environment using Vector CANoe network simulator. The network and car simulations is run on a Vector hardware simulator and a separate machine, respectively. The Vector hardware simulator is also employed as a middle layer, where an additional physical component requires to be integrated into the testbed. In the proposed testbed, a real car's instrument cluster and a CAN filter is interfaced by the Vector hardware simulator to form the physical CAN bus network. Additionally, this hardware expands the testbed's operational capability to perform real-time simulation. This hardware-in-loop simulation setup creates a realistic framework of physical ECUs talking with each other over a physical bus and with simulated ECUs via the virtual bus. Finally, in order to provide a real driving behaviour, a full set of driving wheel, pedals and gear Shifter is attached to the car simulator.

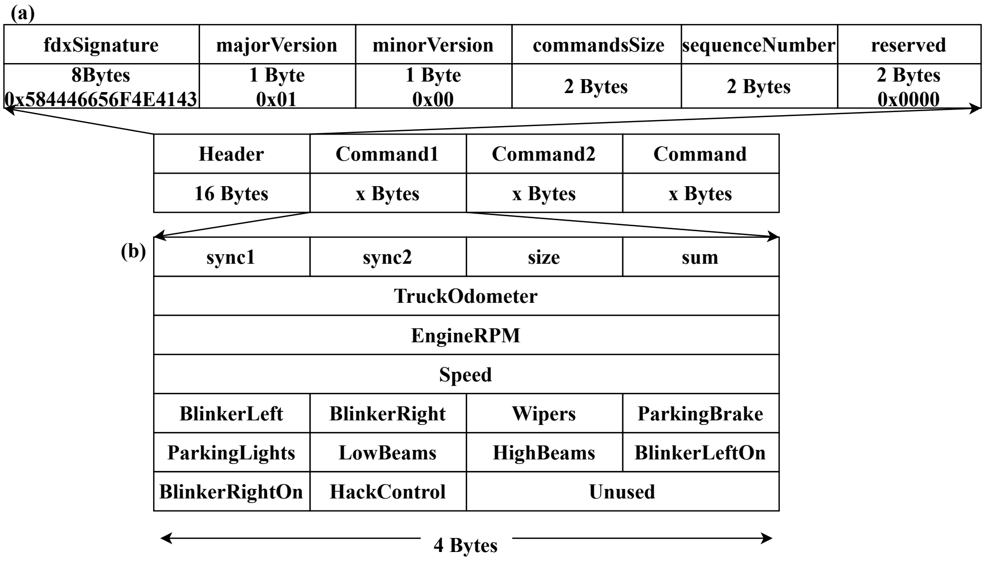


Figure 4: UDP datagram layout consisting two major sections: a) 16 bytes of datagram header consists of a FDX signature, b) 28 bytes of datagram command

CARLA is used as the car simulator to generate car physical data for the on-board network simulator. These include engine information such as engine revolution speed, vehicle velocity, gear, acceleration throttle, and brake status. Also, to achieve CAN bus network simulation, Vector CANoe is employed. CARLA generates realistic car physical data which are sent to CANoe by Fast Data eXchange (FDX).

CANoe FDX is a UDP-based protocol (User Datagram Protocol / IPv4) for simple, fast and real-time exchange of data between CANoe and other systems via an Ethernet connection. This protocol enables other 3rd party software both read and write access to CANoe system, environment variables, and bus signals. To this end, each FDX message is designed to contain the simulated car information including Engine RPM, Speed and Lights' status. These FDX messages are sent by CARLA. UDP is a commonly used protocol which uses a simple connectionless transmission model which means often there must be already a UDP stack available on the CARLA [45]. As there is no pre-defined UDP stack available on the CARLA, it must be first implemented prior to use. The transmitted data by CARLA simulator is achieved through reciprocal transmission of the UDP on a specific port. As a matter of principle, CARLA always sends a datagram to CANoe first and therefore has to know the IP address and port number used by CANoe for the FDX protocol. CANoe evaluates all incoming datagrams to determine the sender's IP address and port number and always responds only to the sender that requested the data. *UDPClient* class in python is an convenient way of sending and receiving UDP datagrams.

Therefore, to establish a *UDPClient*, the specification of a designated socket should be defined in CARLA python code in order to communicate with CANoe. The socket specification includes the port number and the IP address of the CAN simulator (Vector VN89XX). As illustrated in Figure 4b, a new command section including the selected telemetry data is created. The entire process is implemented in the CARLA running code as *send CANoe* function in order to continually send the telemetry data while the game is running.

Receiving these FDX messages requires configuring the CANoe environment in order to receive the incoming data in an appropriate order. The FDX message structure must be specified by a FDX description file and added to the CANoe configuration. The FDX description file is an eXtensible Markup Language (XML) file. It specifies the data contents and variables within UDP data groups that is being sent reciprocally by CARLA. The FDX description file comprises from two data groups: groupID '1' and '2'. Each data group can be used for sending and receiving data. In the proposed testbed, groupID '1' and '2' are used to receive data from CANoe and send data to CANoe, respectively [45]. The FDX option must also be enabled. The XML file can also be configured using the edit option within the CANoe environment. This includes assigning several parameters given to the data group members such as the data symbols, identifier, offset, size and type.

A separate machine is used to configure the simulated CAN bus network and their attached virtual ECUs. To maintain timing in CARLA and CANoe, the timers in Python program are deployed. The simulation configuration is then loaded into a Vector hardware simulator (VN8914) with the communication module (VN8972) to ensure the real time processing.

The simulated CAN messages are transmitted to the physical CAN bus and then received by the instrument cluster through the hacking device that can change the odometer's value.

The hacking device is a CAN filter that can manipulate the CAN traffic between the CAN bus and one or more ECUs. To realise the CAN filter, the ARM-based MCU STM32F105 from ST is used in the testbed. It features two CAN interfaces, one is connected to the CAN bus and the other is connected to the instrument cluster. We implemented and installed firmware into the MCU so that it is acted as a mileage corrector. The CAN filter then has four operational modes. In mode 1, it is inactivated. This means the CAN filter simply replay all the CAN frame it receives from the interface to the CAN Bus to the other interface connected with the instrument cluster. In modes 2, 3 and 4, the CAN filter is activated. This means it makes changes to some of the CAN frames to reduce the incremental rate of the odometer on the instrument cluster:

- In mode 2, the incremental rate is reduced by 90%, i.e., for every 10 miles of travel, the odometer increases by only 1 mile.
- In mode 3, the incremental rate is reduced by 50%, i.e., for every 10 miles of travel, the odometer increases by only 5 miles.
- In mode 4, the incremental rate is reduced by 100%, i.e., for every 10 miles of travel, the odometer does not increase.

One can switch between the modes by quickly flashing the full beam on the CARLA simulator four times. The CAN filter starts in mode 1. If it is in mode $n \in \{1, 2, 3\}$, then switching moves it into mode $n + 1$. If it is mode 4, then switching moves it back to mode 1.

The filtered CAN messages are then sent to the real instrument cluster CAN transceiver and the IDS.

5. IDS Integration

In this section we show how the testbed employing a hacking device such as described in Section 4 can be used for testing a potential IDS implementation. We take the approach outlined in Section 3.4 whereby, following CAN packet log analysis to determine data field clusters, a classifier is trained offline using captured CAN logs, but is then tested in real-time on the testbed.

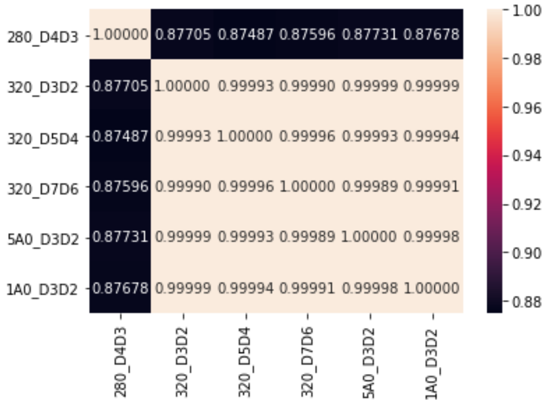
5.1. Cluster Detection and Offline Training

Analysis of the packet flow generated by the CARLA simulator showed an average of 643 packets broadcast per second, with a mean broadcast interval of 0.0016 seconds. Twenty-four different CAN packets were identified. The approach outlined in Section 3.4, suggested a correlated cluster which was used for the subsequent snapshots and anomaly detection (Fig. 5). For the training data, the log from a 30-minute journey was captured and converted into a series of snapshots. The classifier was then trained offline using snapshot records selected at random. The fitted model and the scaling objects are saved using Python's pickling framework, so they can be loaded into our intrusion detection script and used for the classifier.

5.2. Online Testing

The online testing using the trained classifier and the testbed is summarised in Figure 6. We run our Python intrusion detection script on a laptop connected to the testbed via a Kvaser Leaf Light HS v2 logger.

The script reads the CAN messages via the CAN channel presented by the logger. This setup requires drivers (e.g. Kvaser Drivers for Windows) freely available from Kvaser [46], and the python-can library [47] to connect to the CAN channel. Python-can presents each packet as it is broadcast, including the ID and eight data fields.



	280_D4D3	320_D3D2	320_D5D4	320_D7D6	5A0_D3D2	1A0_D3D2
280_D4D3	1.00000	0.87705	0.87487	0.87596	0.87731	0.87678
320_D3D2	0.87705	1.00000	0.99993	0.99990	0.99999	0.99999
320_D5D4	0.87487	0.99993	1.00000	0.99996	0.99993	0.99994
320_D7D6	0.87596	0.99990	0.99996	1.00000	0.99989	0.99991
5A0_D3D2	0.87731	0.99999	0.99993	0.99989	1.00000	0.99998
1A0_D3D2	0.87678	0.99999	0.99994	0.99991	0.99998	1.00000

Figure 5: Correlation matrix of CAN data fields from a correlated cluster

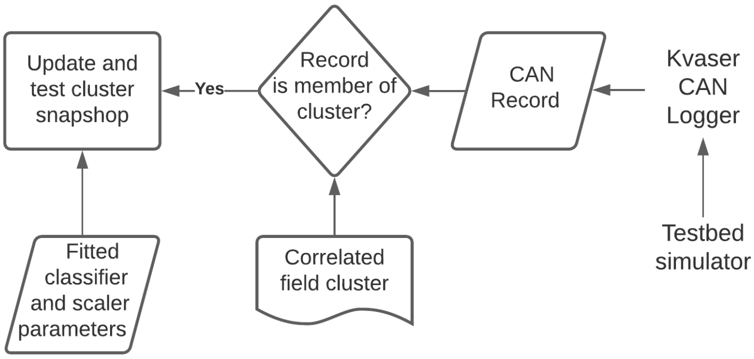

Update and test cluster snapshop
Yes
Record is member of cluster?
CAN Record
Kvaser CAN Logger
Testbed simulator
Fitted classifier and scaler parameters
Correlated field cluster

Figure 6: Overview of the online process for the IDS

As each packet is detected, the script checks the ID for a match against the cluster list. If the packet ID is in the cluster list, its relevant data fields are parsed, combined if needed, and the resulting converted integer field values are updated in the snapshot array. The updated array is then fed to the classifier, which uses the pickled fitted model and scaling objects gathered during the training stage. The snapshot is then classified according to the OCSVM prediction. If the prediction is an anomaly, the CAN packet is considered an attack packet.

5.3. Experiment Results

Prior to testing on the testbed, we tested the classifier by running the Python detection script against output generated by replaying previously captured log files to a virtual CAN network. The data for this used different journeys to those used for training. Each log file represented a 30-minute journey representing one of the operational attack modes described above. Each log file was fed into the trained IDS to make predictions if the CAN filter is activated, i.e., in modes 2, 3 or 4. The evaluation is facilitated by two counters, “inliers” and “outliers”, implemented in the IDS to count how many times it predicts “inactivated” and “activated”. Collecting the values of the counters in each run, we obtained a result described in Table 2. For these results "accuracy" is 0.7636, "recall" is 0.6964, "precision" is 0.9788, and "F1" is 0.8138.

Table 2
Results from testing the IDS via replay of CAN logs on a virtual CAN network

CAN Filter Mode	Predicting Activation	Predicting Inactivation
Mode 1 (inactivated)	25155	556672
Mode 2 (activated)	406834	156727
Mode 3 (activated)	382645	172793
Mode 4 (activated)	373270	177498

In comparison, we evaluated the IDS online by plug-n-playing into our testbed. The CAN messages between the instrument cluster and the CAN filter were collected by the Kvaser logger. We ran the test four times, each time running 30 minutes journeys where the CAN filter was set in one of the four modes. In spite of the high throughput of records broadcast, the script was able to process each in real-time. Clearly, an in-vehicle version would need to be run on different hardware compared to the laptop we used, though the testbed would facilitate the evaluation of alternative implementations. Similar to the previous test, we obtained a result described in Table 3.

Table 3
Results from live-testing the Online IDS on the testbed

CAN Filter Mode	Predicting Activation	Predicting Inactivation
Mode 1 (inactivated)	359336	143963
Mode 2 (activated)	392678	177223
Mode 3 (activated)	426183	143715
Mode 4 (activated)	376455	193490

The testbed detection (Table 3), which was processed in real-time, in comparison to the virtual CAN reply (Table 2), showed more false positives in classifying. Using this method, we obtained "accuracy" 0.6052, "recall" 0.6991, "precision" 0.7689, and "F1" 0.7323. The accuracy reduces to 60% on the testbed, compared with 76% when tested using replay on the virtual network. We repeated the testing using additional journeys and training data but obtained similar results.

The reasons for the reduced detection accuracy on the testbed are still being investigated. The log was replayed at the fastest available replay broadcast speed for the virtual CAN, even so, perhaps a difference in the speed of transmission on the testbed was sufficient to hamper calculations on the laptop running the IDS script. It is also possible that behaviours were observed on the testbed car operation that was not captured for the training data or the test log used for the virtual CAN testing. It is thus possible that the trained model is overfitted and more training data might be required to improve the live prediction performance on the testbed. A further cause of poor prediction might stem from the data parsing algorithm deployed. Again, more research is needed to refine and test this. Whatever reason is eventually uncovered, the results indicate the need for further exploration for real-time tracking. Importantly, they highlight the benefit of using the testbed for validating the trained model against live data where driving behaviours and experiences might be different from those captured when the datasets were collected for training. The approach adopted, which assumes the CAN data dictionary and data derivations are unknown, clearly cannot give a complete picture of the CAN data, or reveal what the data represents. Also, without a more thorough analysis, it is difficult to determine whether convoluted manipulations or conditional calculations might have been used to derive the data. These would reduce the detection reliability.

6. Conclusion

In this paper, we presented a novel approach for building an automotive security testbed that comprises both a full-scale in-vehicle network and a car simulator to generate network traffic in realistic driving scenarios. The in-vehicle network utilises Vector CANoe software and hardware to resemble a real CAN Bus within a vehicle to a high degree of precision. These enable a flexible configuration of the network in terms of (1) receiving input from external sources to enrich simulations and (2) allowing plug-n-playing both virtual and physical components. To this end, we realise driving scenarios by employing CARLA, a car simulator, which is highly customisable with respect to driving environments (e.g., driving maps and traffic participants), as input to the CAN Bus to generate realistic CAN traffic. The testbed, therefore, can be used to generating datasets and testing machine learning-based IDS.

This was illustrated by the example of training and testing an IDS implementation. Testing a classifier in real-time using the testbed revealed poorer detection rates compared to testing performed by running the trained classifier offline against previously captured logs. The precise reasons for the poorer detection and the circumstances in which classifier detection deteriorates, need investigating - something the testbed can facilitate through its ability to replicate diverse driving scenarios. It is conjectured that the difference between driving behaviours and experiences, and those captured when collecting datasets for training might have contributed to the reduction of the implemented IDS accuracy when its performance online on the testbed is compared against running a virtual simulation. These test results reveal the benefit of using the testbed for validating the trained model against live data in real-time. Thus, testing using the live-journey replicating testbed can uncover issues in attack detection that might be missed in offline testing.

While this paper only focused on the IDS as anomaly detection, other well known detection methods such as potentially against nearest neighbour, fuzzy logic and neural net based methods will be bench-marked against each other to further investigate the role of testing platform on their accuracy. Additionally, more physical components will be added to the test bench in order to include more abuse cases. Also, the data exchange with CARLA will become bi-directional to study the effect of anomaly in the driving behavior.

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, et al., Experimental security analysis of a modern automobile, in: 2010 IEEE Symposium on Security and Privacy, IEEE, 2010.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, et al., Comprehensive experimental analyses of automotive attack surfaces., in: USENIX Security Symposium, volume 4, 2011.

[3] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, Survey on security threats and protection mechanisms in embedded automotive networks, in: IEEE/IFIP DSN, 2013, pp. 1–12.

[4] C. Miller, C. Valasek, Remote exploitation of an unaltered passenger vehicle, Black Hat USA 2015 (2015) 91.

[5] C. Miller C., C. Valasek, Car hacking: for poories, Technical Report, 2015.

[6] T. Hoppe, S. Kiltz, J. Dittmann, Security threats to automotive can networks–practical examples and selected short-term countermeasures, in: SAFECOME, Springer, 2008.

[7] U. E. Larson, D. K. Nilsson, Securing vehicles against cyber attacks, in: Proceedings of the 4th annual workshop on Cyber security and information intelligence research, 2008.

[8] P. R. Thom, C. A. MacCarley, A spy under the hood: Controlling risk and automotive edr, Risk Management Magazine 55 (2008) 22.

[9] M. Wolf, A. Weimerskirch, C. Paar, Security in automotive bus systems, in: Workshop on Embedded Security in Cars, Bochum, 2004.

[10] M. Wolf, A. Weimerskirch, T. Wollinger, State of the art: Embedding security in vehicles, EURASIP Journal on Embedded Systems 2007 (2007) 1–16.

[11] Y. Zhao, Telematics: safe and fun driving, IEEE Intelligent systems 17 (2002) 10–14.

[12] A. Taylor, N. Japkowicz, S. Leblanc, Frequency-based anomaly detection for the automotive CAN bus, World Congress on Industrial Control Systems Security (2015) 45–49.

[13] A. Taylor, S. Leblanc, N. Japkowicz, Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks, 2016 IEEE DSAA (2016) 130–139.

[14] I. Studnia, E. Alata, V. Nicomette, M. Ka, I. Studnia, E. Alata, V. Nicomette, M. Kaˆ, Y. L. A, A language-based intrusion detection approach for automotive embedded network, in: IEEE (Ed.), 21st IEEE PRDC on Dependable Computing, Zhangjiajie, China, 2015.

[15] M.-J. Kang, J.-W. Kang, Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security, PLoS ONE 11 (2016).

[16] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, E. Vázquez, Anomaly-based network intrusion detection: Techniques, systems and challenges, Comput. Secur. (2009).

[17] P. Borazjani, C. Everett, D. McCoy, Octane: An extensible open source car security testbed, in: Proceedings of the Embedded Security in Cars Conference, 2014.

[18] J. Daily, R. Gamble, S. Moffitt, C. Raines, P. Harris, J. Miran, I. Ray, S. Mukherjee, H. Shirazi, J. Johnson, Towards a cyber assurance testbed for heavy vehicle electronic controls, SAE International Journal of Commercial Vehicles 9 (2016) 339–349.

[19] P. S. Oruganti, M. Appel, Q. Ahmed, Hardware-in-loop based automotive embedded systems cybersecurity evaluation testbed, in: AutoSec, 2019, pp. 41–44.

[20] T. Toyama, T. Yoshida, H. Oguma, T. Matsumoto, Pasta: portable automotive security testbed with adaptability, Black Hat Europ, Tech. Rep. (2018).

[21] X. Zheng, L. Pan, H. Chen, R. Di Pietro, L. Batten, A testbed for security analysis of modern vehicle systems, in: 2017 IEEE Trustcom/BigDataSE/ICESS, IEEE, 2017, pp. 1090–1095.

[22] A. Marchetto, P. Pantazopoulos, A. Varádi, Cvs: Design, implementation, validation and implications of a real-world v2i prototype testbed, in: IEEE VTC2020, 2020.

[23] C. Gay, T. Toyama, H. Oguma, Resistant automotive miniature network (2019).

[24] H. N. Nguyen, S. Tavakoli, S. A. Shaikh, O. Maynard, Developing a qrng ecu for automotive

security: Experience of testing in the real-world, in: ICST Workshops, IEEE, 2019.

[25] A. J. Tomlinson, J. Bryans, S. Shaikh, Towards viable intrusion detection methods for the automotive controller area network, in: ACM-CSCS, 2018.

[26] S. Marksteiner, N. Marko, A. Smulders, S. Karagiannis, F. Stahl, H. Hamazaryan, R. Schlick, S. Kraxberger, A process to facilitate automated automotive cybersecurity testing (2021).

[27] J. Hayward, A. Tomlinson, J. Bryans, Adding cyberattacks to an industry-leading can simulator, in: IEEE 19th QRS-C, 2019.

[28] T. K. T. N. (KTN), Automotive Cyber Security: An IET/KTN Thought Leadership Review of risk perspectives for connected vehicles, Technical Report, 2015.

[29] C. Valasek, C. Miller, Adventures in Automotive Networks and Control Units, Technical White Paper (2013) 99.

[30] S. Fröschle, A. Stühring, Analyzing the capabilities of the CAN Attacker, Lecture Notes in Computer Science 10492 LNCS (2017) 464–482. arXiv:9780201398298.

[31] D. S. Fowler, J. Bryans, M. Cheah, P. Wooderson, S. A. Shaikh, A method for constructing automotive cybersecurity tests, a can fuzz testing example, in: IEEE 19th QRS-C, 2019.

[32] J. Singh, M. J. Nene, A Survey on Machine Learning Techniques for Intrusion Detection Systems, IJARCCE 2 (2013) 4349–4355.

[33] D. Bhattacharyya, J. Kalita, Network Anomaly Detection, CRC Press, 2013.

[34] B. Groza, P. S. Murvay, Efficient Intrusion Detection with Bloom Filtering in Controller Area Networks, IEEE Transactions on Information Forensics and Security 14 (2019) 1037–1051.

[35] S. C. HPL, Introduction to the controller area network (can), Application Report SLOA101 (2002) 1–17.

[36] W. Voss, A comprehensible guide to controller area network, Copperhill Media, 2008.

[37] M. Müter, N. Asaj, Entropy-based anomaly detection for in-vehicle networks, in: 2011 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2011, pp. 1110–1115.

[38] N. Nowdehi, A. Lautenbach, T. Olovsson, In-vehicle can message authentication: An evaluation based on industrial criteria, in: Vehicular Technology Conference, IEEE, 2017.

[39] Vector, Canoe user manual, https://www.vector.com, 2021.

[40] CAPL Documentation, Vector Informatik GmbH, 2020. https://kb.vector.com/entry/48/.

[41] CARLA - Open-source simulator for autonomous driving research, https://carla.org, 2021.

[42] A. Tomlinson, J. Bryans, S. A. Shaikh, Using internal context to detect automotive controller area network attacks, Computers & Electrical Engineering 91 (2021) 107048.

[43] L. A. Maglaras, A Novel Distributed Intrusion Detection System for Vehicular Ad Hoc Networks, International Journal of Advanced Computer Science and Applications (2015).

[44] C. Chio, D. Freeman, Machine Learning and Security, 1st ed., O'Reilly, 2018.

[45] A. Huber, Fast Data Exchange with CANoe, 2020. https://assets.vector.com.

[46] Kvaser, Kvaser, 2021. URL: https://www.kvaser.com/.

[47] Python-CAN, python-can, 2020. URL: https://python-can.readthedocs.io/en/master/.