Controller Area Network and In-Vehicle Network Security Survey

COMP5900H Fall 2020 Project

Patrick Killeen, pkill013@uottawa.ca , December 10 2020

Abstract

Internet of things (IoT) is a popular trend, where an increasing number of resource-constrained devices are being deployed. With all these devices, a large amount of data is being produced. Modern vehicles may be placed under the category of IoT, since they have a series of on-board heterogeneous devices that have various resource requirements and produce a lot of data. These devices are called electronic control units (ECUs), and can range in numbers from 40 to 100 on board modern luxury vehicles. ECUs manage the sensors and actuators of vehicle components, and are connected to in-vehicle networks (IVNs). Several IVNs exist, including, CAN, FlexRay, MOST, LIN, and Ethernet. These vehicle network architectures enable features such as accident prevention, accident emergency response, predictive maintenance, and other remote diagnostic applications. The number of features, interfaces, and ECUs on board vehicles introduces many attack surfaces and potential vulnerabilities. Successful cyberattacks against these vehicles could lead to data theft, but more importantly, they could lead to loss of life.

Attacks against vehicles can be categorized into three types of attacks, namely: long-range wireless, short-range wireless, and physical access attacks. An example of a vehicle cyberattack follows: a denial-of-service attack can be mounted against a CAN bus by flooding the bus with high priority messages, preventing the normal operation of the vehicle. This example illustrates one of several CAN security issues (CAN also has no built-in form of addressing or authentication). However, CAN does have defined security standards, requiring authentication before performing sensitive ECU actions (such as reading ECU memory locations, for example). A series of successful vehicle attacks have been performed in the literature. In general, the security researchers were able to fully control a car by using the following methodology: a) sniffing the CAN bus and associating observed actions to replayed packets, b) reverse-engineering software to find vulnerabilities, c) using debugging tools to extract the knowledge required for taking advantage of the vulnerabilities, and d) exploiting the vulnerabilities to write arbitrary messages onto the CAN bus, granting significant (or full) control over the vehicle. Most of the successful vehicle hacking experiments took advantage of common vulnerabilities. Therefore, it can be seen that in general, vehicles can be protected against cyberattacks by: deploying an intrusion detection system, following CAN standards, avoiding nonce re-use, avoiding buffer overflow vulnerabilities, securing each ECU in an IVN, closing unused ports, and enforcing frame-level encryption and message authentication on the IVN.

1 INTRODUCTION

The trend of Internet of Things (IoT) is leading to an increasing number of resource-constrained devices that are being deployed, which are used to sense the environment around them and act upon their environment. Modern vehicles may be categorized as part of the IoT. Modern vehicles typically have from 40 to 100 on-board devices [27], and these devices are called electronic control units (ECUs) (or named engine control units in some literature). There are tens of millions of automobiles in North America [17]. Vehicles are becoming increasingly complex as the number of features increase with newer vehicle models. By reviewing the literature [17], [26], [19], [2] vehicle features include: infotainment system updates, emergency response systems, navigation maps, anti-locking brakes, adjusting radio volume based on vehicle speed, and weather information. Furthermore, there is also this new trend that supports the notion of 'car-as-a-platform', which involves having application developers create and publish automobile applications on App stores [17]. One can only imagine the number of security issues that will arise from such a trend if vehicle cybersecurity research does not catch-up with this trend.

Therefore, with the increasing numbers of ECUs, features, and vehicles on the road, it is important to consider the cybersecurity of modern vehicles. As vehicles become increasingly more sophisticated, they are more vulnerable to cyberattacks [26][19]. In standard cybersecurity, availability, authenticity, integrity, and confidentiality are the typical requirements to secure a system, but vehicle cybersecurity slightly deviates from this model [26]. Vehicles may be hacked like any other computer system, however, a successful attack against a vehicle not only threatens the privacy of the vehicle's users, the attack could also lead to loss of life [19][21]. Unfortunately, securing vehicles is more difficult than securing standard systems on the Internet [26].

It is therefore important to understand the security concerns and threats against in-vehicle networks (IVNs). The present work reviews the literature about enabling vehicular technology and IVNs, while focusing on the Controller Area Network (CAN) IVN. The present work also explores threat models, successful attacks against vehicles, and how vehicles can be protected from such attacks. It is worth noting that the security of self-driving vehicles is beyond the scope of the present work.

1.1 Paper Overview

This paper is structured as follows: section 2 contains some related vehicle technology background to help a reader understand the vehicle security discussion in later sections. Specifically, this section covers IoT security, IVNs, and enabling vehicular technology; section 3 presents a detailed review about IVN and CAN bus security. Specifically, this section covers the proposed threat model, CAN bus security, hacking examples, a comparison of IVN security, and security solutions to help protect a vehicle against cyberattacks; and section 4 concludes the present work with a summary of the literature discussed in the present work and some future work that could improve the present work.

2 PROJECT TOPIC - BACKGROUND

2.1 Internet of Things - IoT

IoT consists of many interconnected devices that have networking, computing, and storing capabilities. These devices tend to be resource-constrained (for example, battery powered), heterogeneous in nature, and deployed in large numbers. These devices sense the environment around them using sensors, and act upon insights acquired using actuators [14]. Studying the security of IVNs (especially CAN) requires some understanding of the principles of IoT security, since IVNs in modern vehicles are resource-constrained networks of heterogeneous microprocessors that are connected to sensors and actuators.

2.1.1 Security Requirements and Considerations. In addition to availability, authenticity, integrity, and confidentiality, a literature review [1], [10], [13], [28], [18] reveals the following key IoT security requirements and concepts that should be considered when designing security for IoT:

- **Software and firmware patching**: when considering the firmware patching of devices, care must be taken otherwise the firmware/software update process may leave devices vulnerable. For example, Costin et al. [3] find that some publicly accessible firmware images include private keys used by many other devices.
- **Key management**: the management of keys should be properly conducted, otherwise, attackers may compromise keys or rogue users may retain access to a system even after having been revoked access.
- **Remote maintenance**: it should be easy to remotely maintain these devices, otherwise, devices that are deployed may be forgotten and could become prime targets for attacks as their software/firmware become out-of-date.
- **Default passwords**: the devices should also avoid having simple default passwords, else they risk being hacked by automated malware that scan networks and conduct dictionary attacks; for example, the Mirai virus [16] created a bot net of IoT devices by using dictionary attacks that used known commonly used default passwords.
- **Physical location**: in terms of physical location of a device, it should not be trivial to tamper with the device; for example: a) publicly exposing an OBD-II port inside a public transportation vehicle would be a poor choice, since passengers would have physical access to the OBD-II port, or b) placing a sensitive device in a room without security cameras would be unwise.
- **Multi-device compromise**: the security of all nodes in a resource-constrained network should be considered. An attack vector used to compromise a system's component may not pose as a serious threat; however, when multiple dependent attack vectors are used to compromise a variety of components, the collective malicious behavior of the components may in fact pose as a serious unforeseen threat, where individually the components pose only as a little threat [26]. For example, the Stuxnet attack used a worm that targeted multiple components of nuclear centrifuges used in a nuclear program of Iran.

2.1.2 Security Mechanisms. There are many options for addressing the security requirements of IoT. These options are discussed below:

- **Layering** the IoT architecture into software layers can help address the security requirements, since it enables applying security at each IoT layer [1]. For example, low-level perception nodes (such as sensors, for example) will naturally have less built-in security, middleware nodes may have the capabilities of applying more heavy-duty security mechanisms,

and application/cloud layer IoT services can be secured using standard security practices. Note that section 3 discusses some details and properties about these IoT layers, which are proposed by Killeen et al. [15].

- Since some IoT devices do not have the necessary resources for implementing full-fledged networking protocols, connecting the devices via a **gateway** (or aggregation node, in some literature [1]) can give the devices Internet access via standard networking protocols (for example, a Raspberry Pi could act as a gateway). This enables the use of more resource-intensive security mechanisms; for example, a firewall can be installed on a gateway to apply a security layer over many resource-constrained devices (a collection of sensors, for example) that could otherwise not implement a firewall [1]. Note that section 3.5.2 goes into more detail on how a gateway can be used for securing IVNs.
- Enforcing device manufacturers to follow **security standards** [18] [16] can also help IoT security. IoT device manufacturers would be forced to follow a government-issued IoT security legislations; for example, a legislation was put in place for car and airplane manufacturers to enforce user/driver privacy and strong vehicle cybersecurity [22], and according to Greenberg [7], this legislation was proposed to address the car hacking detailed by Greenberg [6]. Another example of a standard could be to enforce manufacturers to use unique default passwords for their IoT devices [16]. Without security standards, viruses like Mirai can cause a lot of damage [16].
- Lightweight **wireless network security** protocols can also be used to add confidentiality, integrity, and message authentication over a resource-constrained wireless protocol. For example, Karlof et al. [13] propose TinySec, a link layer security software implementation for resource-constrained devices in wireless sensor networks.
- **Patching** out-of-date software should be done regularly, since network scans can reveal vulnerable devices. Durumeric et al. [4] propose Censys, a system architecture that can be used to query vulnerable devices on the Internet. Censys can find devices that have OpenSSL versions that are vulnerable to the Heartbleed[1] bug.

2.1.3 Security Challenges. A literature review [11], [8], [13], [19] reveals the challenges of applying security to IoT include the following:

- **Key management.**
- Managing firmware image repositories for offline vs. online **firmware image signing**, which enables firmware patching of a large number of resource constrained devices.
- Some devices may be too **resource-constrained** to implement a cryptographic algorithm, while some networking protocols may have packets that are too small to add security parameters to.
- The number and **heterogeneity of devices** adds a layer of difficulty to securing IoT devices. For example, in the domain of vehicles, there may be a variety of types of on-board IVNs, and each vehicle manufacture will most likely implement a unique (proprietary to individual manufacturers) CAN message encoding/decoding scheme, which poses as a challenge to securing vehicles and IVNs.
- The physical nature of a IoT system means securing it requires consideration of the **physical aspect** of the system as well. A Linux cloud node may be assumed to always be physically present and always running, which arguably only requires software security considerations. However, for cyber physical systems (CPS)/IoT, physical attacks are possible and must be included in the threat model when designing the security of an IoT system (an attacker could steal a device or cut a wire, for example).
- The **novel types of threats** that arise against IoT, compared to the standard security solutions to secure the Internet, pose also as a challenge. For example, sonar waves could be used to attack a system via a microphone sensor, which clearly would be difficult to predict for inclusion of such an attack in a threat model.

2.2 In-Vehicle Networks - IVN

According to Pesé et al. [27], vehicles have become more reliant on digital components over the years [17]. ECUs are found on board vehicles and communicate between each other using an IVN to enable vehicle operation. Multiple IVNs may be interconnected by an on-board gateway. In the 1990s, vehicles had only a few ECUs, while modern vehicles today can easily be equipped with 40 to 100 ECUs. Some also estimate that the software on board modern vehicles has tens of millions of lines of code. Four popular IVNs are CAN, FlexRay [17], Local Interconnect Network (LIN), and Media-oriented System Transport (MOST). Modern vehicles typically have at least five IVNs on-board [26]. Ethernet can also be used as an IVN [11], although it is mentioned less frequently in the IVN literature. Figure 1 illustrates and example of an IVN architecture.

[1] A critical bug that affects a version of the popular open-source cryptographic library, OpenSSL [30].

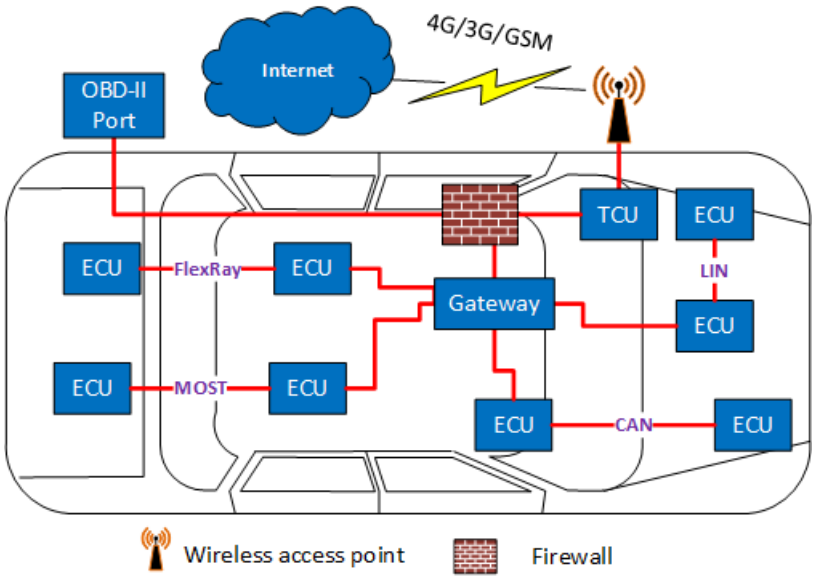

Figure 1: An example of a vehicle's IVN architecture. This figure is inspired from [11].

2.2.1 CAN - Controller Area Network. A literature review [43], [5], [20], [35], [14], [37], [27], [11], [17], [36] reveals that the CAN is a high-bandwidth (1 Mbps) low-resource network used by most vehicles. It was designed in the 1980s, and since 2008, it has become mandatory that all cars implement the OBD-II interface (discussed in section 2.2.2) over CAN [12]. CAN only supports broadcast messages. Peers in the network are ECUs. There are no addresses in the messages; the CAN messages mainly consist of a message ID and an 8-byte payload (the packet structure will be discussed shortly). The message ID is used to win wire contestation by using bit arbitration, when two ECUs write to the wire simultaneously. The message with the lowest ID wins the contest (more on this topic is discussed later in this section). DBC files, which contain proprietary sensor data decoding tables, can be used to decode proprietary CAN message data.

It is worth noting that Controller Area Network with Flexible Data Rate (CAN FD) is a more recent extended version of CAN that supports a data field of up to 64 bytes and a bus speed of up to 8 Mbps. Although, it is not found in literature as commonly as CAN is.

CAN Frame Structure: figure 2 illustrates a diagram of the important fields of a CAN frame. The top frame is the standard CAN frame, and the bottom frame is the extended-ID CAN frame. The message length of the standard frame and extended-ID frame is 11 bits and 29 bits, respectively. The fields that are important for understanding vehicle security are displayed, and all the other fields are not represented in this diagram ('reserved' replaces these irrelevant fields). It can be seen that CAN frames have a cyclic redundancy code (CRC) for error checking.

Message Arbitration - Resolving Collisions: recall that there are no packet collisions on the wire, which is achieved using message ID arbitration. The frame with the lower message ID has the highest priority. All ECUs are synchronized and read the bits one-by-one. If an ECU, E_a, writes a 1 and detects that another ECU, E_b, wrote a 0 on the wire, E_a will back-off and let E_b keep writing its message. This process can be thought of as taking the bit-wise 'and' of the bits written by all ECUs on the wire, and the conjunction of these bits end-up on the CAN bus. Example 2.1 presents an example of the message arbitration process.

Example 2.1. CAN Message Arbitration

An example of the bit arbitration process is illustrated in figure 3. In the table, it can be seen that the ECUs are writing messages onto the wire at the same time. The bottom row ('CAN Data' row) illustrates the bit that won the bit arbitration at each time step; that is, reading from the CAN bus would result in reading the bits from the 'CAN Data' row. When the ECUs write the first bit of their message ID, they each write a 1, and since no ECU wrote a 0, no ECU backs off. When the second bit of the message ID is written, each ECU writes a 0, and since no ECU wrote a 1, no ECU backs off. For the third bit of the message ID, ECU 3 writes a 1, but since a 0 was written on the wire, ECU 3 backs off. This process continues, and at the 7*th*

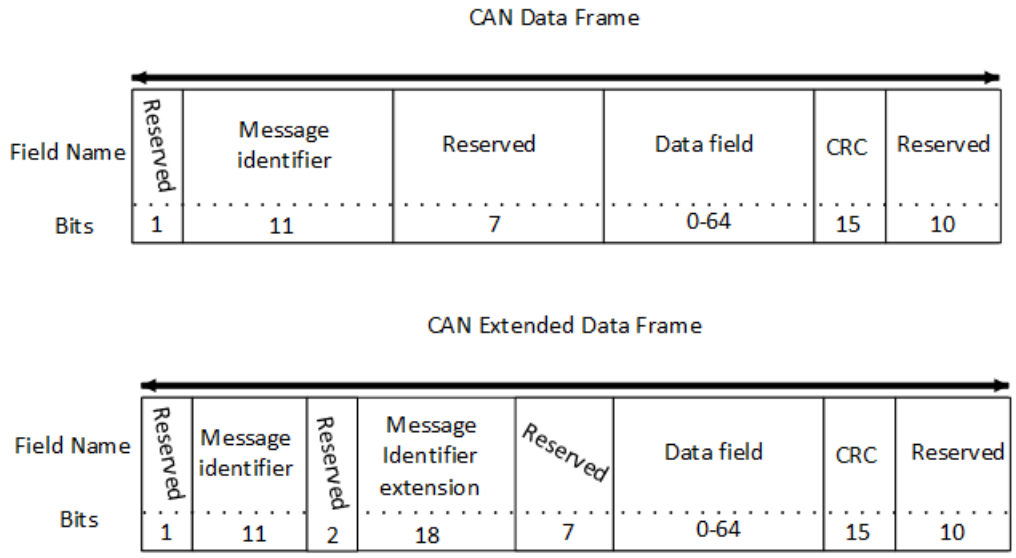

Figure 2: CAN frame fields that are relevant to vehicle security (top frame = standard CAN frame, and lower frame = extended-ID CAN frame). This figure is inspired from [37] and [17].

	Start Bit	ID Bits											The Rest of the Frame
		1	2	3	4	5	6	7	8	9	10	11	
ECU 1	0	1	0	**0**	1	1	0	**0**	1	1	0	1	
ECU 2	0	1	0	**0**	1	1	0	**1**	Stopped Transmitting				
ECU 3	0	1	0	**1**	Stopped Transmitting								
CAN Data	0	1	0	0	1	1	0	0	1	1	0	1	

Figure 3: CAN bus message arbitration example. This figure is inspired from [38].

bit, ECU 2 backs off, since ECU 2 wrote a 1 while a 0 was also written. ECU 1 ends up being the node that wins the message arbitration and gets to write its entire message on the wire.

CAN Frame Decoding: most CAN message encodings are proprietary to the ECU manufacturer. A DBC file is required, since it stores decoding information and descriptions for sensors, and other CAN message details. For example, OBD-II, a diagnostic standard for vehicle emission tests discussed in section 2.2.2, is a public standard that defines a DBC file, allowing emission tests to be carried out. Interested readers are referred to example 2.2, which presents how to decode a CAN message. Do keep in mind that the endianness[2] of the ECU's microprocessors must be kept in mind when decoding sensor values into a CAN data field.

Example 2.2. CAN Message Decoding

This example is taken directly from Killeen [14], one of the previous works of the present work's author. The DBC file used is the Society of Automotive Engineers J1939 digital annex of October 2017.

In this example, the CAN message data field to decode is '`3e 69 1f 30 10 0 59 ff`'. The data field contains the sensor reading of the 'engine intake air mass flow rate', which is a 2-byte sensor value found at second and third byte of the data field. The offset (b in equation 1) and the resolution (m in equation 1) are used to decode the value.

[2]Endianness defines where the most significant byte of a binary word is stored [39].

$$
\begin{aligned}
&\texttt{data field} = \texttt{3e 69 } \mathbf{1f\ 30} \texttt{ 10 00 59 ff} \\
&b = 0\ kg/h \\
&m = 0.05\ \texttt{(kg/h)/bit} \\
&x = \texttt{1f 30} = (\texttt{1f30})_{16} = (7984)_{10}\ \texttt{bit - in Little Endian systems} \\
&x = \texttt{1f 30} = (\texttt{301f})_{16} = (7984)_{10}\ \texttt{bit - in Big Endian systems} \\
&\\
&y = mx + b \\
&y = (0.05\ \texttt{(kg/h)/bit}) \cdot (7984\ \texttt{bit}) + 0\ \texttt{kg/h} \\
&y = 0.05 \cdot 7984\ \texttt{kg/h} + 0\ \texttt{kg/h} \\
&y = 0.05 \cdot 7984\ \texttt{kg/h} + 0\ \texttt{kg/h} \\
&y = \mathbf{399.2\ \ kg/h}
\end{aligned}
\tag{1}
$$

It can be seen from equation 1 that after extracting the encoded parameter from the CAN data field and applying the '$y = mx + b$' equation to it, the air intake sensor value is 399.2 kg/h.

2.2.2 On-Board Diagnostics - OBD-II. According to Pesé et al. [27], a standard was put in place such that all vehicles sold in the US since 1996 had to support an OBD-II interface [17], which provides, over the OBD-II protocol, read-only diagnostic information regarding emission tests and other related sensor data (e.g., mass air flow, engine speed, vehicle speed, and intake temperature). Note that that the OBD-II protocol only supports read operations, while the physical OBD-II port/interface may be used for writing and reading raw CAN messages. Furthermore, an on-board gateway typically offers an OBD-II port.

2.2.3 Flexray. The FlexRay bus is typically used for automotive power control systems. Its bandwidth supports up to 10 Mbps. FlexRay uses time division multiple access and flexible time division multiple access to meet the real-time requirements of the mission-critical components relying on FlexRay. A FlexRay message can hold up to 254 bytes of data in its data field. FlexRay uses a CRC for error checking [11].

2.2.4 Media-oriented System Transport - MOST. A literature review [26], [9], [34] reveals that MOST is designed for multi-media (video, audio, navigation, and communication systems). There are three versions of MOST, namely, MOST25, MOST50, and MOST150, each supporting up to 25 Mbps, 50 Mbps, and 150 Mbps, respectively. MOST supports up to 64 devices in a ring network topology, and has a master-slave communication model. It has audio and video encoding schemes built into its specification, which make it ideal for multi-media support. It also supports an Ethernet interface, which makes bridging MOST to Ethernet networks straight forward. The data field size varies depending on which MOST protocol is chosen for the Open Systems Interconnection (OSI) network stack layers[3]; a couple examples follow: a) the Packet Data protocol has a 48-byte data field in the link layer (MOST25), and another data link layer protocol supports a data field of length up to 1014 bytes (MOST25/50), and b) the data field length of the Application Message Service protocol of MOST supports a data field length of up to 65535 bytes.

2.2.5 Local Interconnect Network - LIN. The LIN bus is typically used for controlling vehicle seats, doors, wipers, sunroofs, and other body-related components. A LIN message can contain up to 8 bytes of data in its data field. A frame also includes a checksum for error detection. The bandwidth of LIN supports up to 20 kbps, and LIN uses a master-slave communication model [11].

2.2.6 Ethernet. IEEE 802.1 AVB and TTEthernet are examples of Ethernet-based IVN protocols for infotainment applications. Ethernet enables cloud-to-vehicle communication, including remote updates and diagnostics, since the Internet backbone is primarily composed of Ethernet. The bandwidth of Ethernet supports between 100 to 1000 Mbps. The data field of an Ethernet packet can contain more than 1000 bytes, and Ethernet uses CRC for error checking [11].

2.2.7 Comparison. This section compares the similarities and differences between the IVNs. Table 1 summarizes key differences between CAN, CAN FD, FlexRay, LIN, MOST, and Ethernet. The following remarks can be made: a) CAN and LIN have the smallest data fields; b) FlexRay and Ethernet have the larger data fields; c) CAN and LIN have the slowest bit rates; d) MOST and Ethernet have the faster bit rates. Note that it is difficult to directly compare MOST's data field to the other IVNs' data fields, since MOST has different data field sizes depending on the chosen OSI layer protocol.

[3]The OSI model defines the following network abstraction layers: physical layer, data-link layer, network layer, transport layer, session layer, presentation layer, and application layer [41].

IVN protocol	Bit rate	Data field size (bytes)	Description
CAN	1 Mbps	8	Mandatory for OBD-II interface since 2008
CAN FD	8 Mbps	64	Extended version of CAN
FlexRay	10 Mbps	256	Automobile power control systems
LIN	20 kbps	8	Automobile body control
MOST	150 Mbps	(*)	Multi-media
Ethernet	100 - 1000 Mbps	>1000	Internet backbone

Table 1: Summary of key IVN differences. (*): MOST's data field size depends on the chosen OSI layer MOST protocol, so the data field varies.

Furthermore, when choosing which IVN to use, it depends on the domain the IVN is being used in. Choosing CAN would be the most straight forward choice, since CAN has been mandatory for automobiles since 2008. For a rear-view camera streaming feature, MOST would probably be the best IVN choice, since it is a multi-media protocol. In a car-as-a-platform domain, MOST and/or Ethernet would probably be good choices, since MOST bridges well with Ethernet and connecting an Ethernet-based IVN to the cloud is straight forward.

2.3 Enabling Vehicular Technology

This section presents a variety of enabling vehicular technologies.

2.3.1 Vehicle Telematic Services. Vehicles are equipped with a telematic control unit (TCU), which is responsible for bridging Internet access (via cellular network) to the IVNs [21]. A TCU enables a pleasant variety of remote vehicle applications and services. A literature review [17], [2], [23] reveals that car manufactures support remote telematic systems including Ford's Sync, GM's OnStar, Chrysler's UConnect, Toyota's SafetyConnect, Lexus' Enform, BMW's BMW Assist, BMW's Connected Drive, and Mercedes-Benz's mbrace . Such systems provide features like remote assistance, emergency response notification using GPS location, predictive maintenance notifications, unlocking doors remotely, and other similar application features.

2.3.2 Diagnostic Tools. Popular diagnostic tools (or OBD-II scan devices) include: Ford's NGS, Nissan's Consul II, and Toyota's Diagnostic Tester. These devices will be referred to as 'PassThru' devices in the rest of the present work, and are programmed using Windows-based software. The software solutions used to reprogram ECUs using these tools include: Toyota's TIS, Ford's VCM, Nissan's Consult 3, and Honda's HDS [2].

2.3.3 Interfacing Tools. This section presents a few key interfacing tools/cables that can be used to read and write from a CAN bus. CANtact is a device that acts as an interface bridge between an OBD-II port and a USB port on a computer. This tool can be used with open-source software (written in Python) to mount attacks against a CAN bus [26]. CANCapture ECOM cable is a CAN-to-USB off-the-shelf cable that is used to interface standard devices (a laptop, for example) to a CAN bus [17]. Technoton's CANCrocodile is another useful tool. It is a contactless CAN bus reader that enables non-intrusive CAN bus reading [31]. Intrepid Control System's neoVI FIRE is a hardware interface that enables streaming CAN messages to a computer from an OBD-II port via USB [12].

2.3.4 Infotainment Systems. Infotainment (information and entertainment) systems are responsible for managing the in-vehicle user console and offer the driver a variety of features including: a Wi-Fi hotspot, weather information, a navigation system, and a media player. As a result, such systems have many user interfaces (USB, Bluetooth, Wi-Fi, etc.). According to Lin et al. [21], infotainment systems typically run the following operating systems (OS): QNX, Green Hills, Linux, Windows CE, and possibly an Android-based virtual machine. Furthermore, Miller et al. [25] found that a 2014 Jeep Cherokee's infotainment system (UConnect) runs the QNX OS with a 32-bit ARM processor.

3 VEHICLE SECURITY

This section discusses key concepts related to vehicle security. One of the most vulnerable components of a vehicle is the infotainment system [21]. It is also important to think about: a) the inter-dependencies between the ECUs and their functions, and b) the consequences of exploiting a group of ECUs collectively [26], since exploiting a group of ECUs could grant control over mission-critical functions of a vehicle if ECUs were poorly designed. Furthermore, Checkoway et al. [2] found that compromising one IVN bus can lead to compromising the entire vehicle.

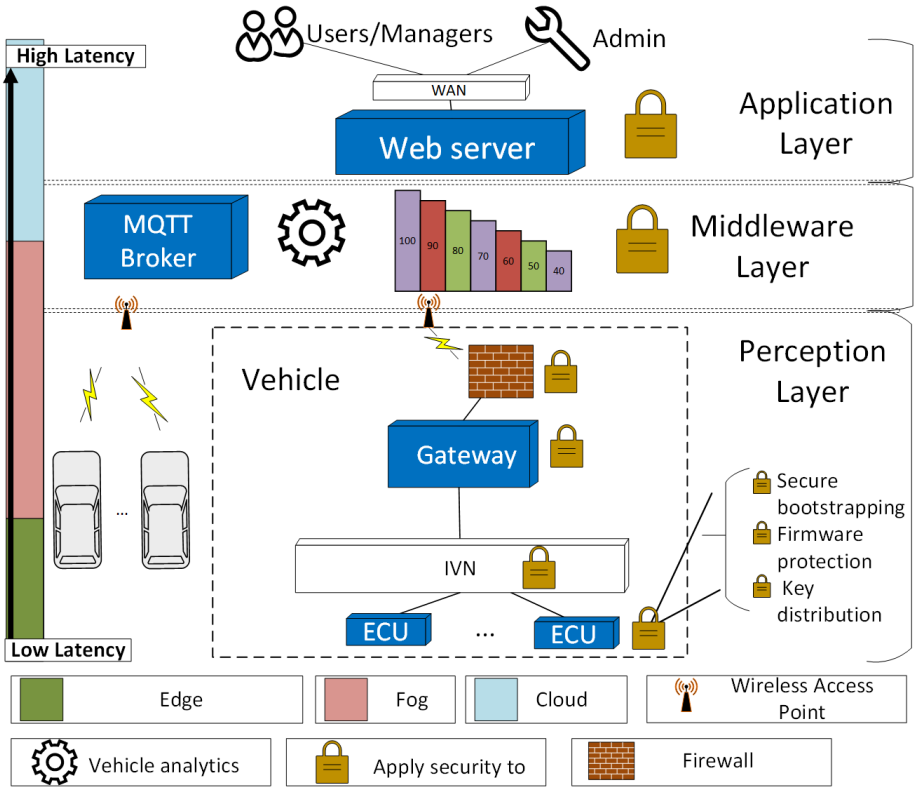

Figure 4: An illustrative example of applying security to IoT software layers and vehicle security layers in a fleet management system. This figure is inspired from Hu et al. [11] and Killeen [14].

Hu et al. [11] present the following 8 requirements for IVN security:

(1) **Data origin authenticity**: the source of data should be authentic and reliable.
(2) **Integrity**: packets are not modified on the wire.
(3) **Access control**: only authorized entities should be able to access information.
(4) **Freshness**: packets have timestamps.
(5) **Non-repudiation**: an action of an entity cannot be denied.
(6) **Privacy/anonymity**: the data is confidential.
(7) **Confidentiality**: only authorized entities can access the data.
(8) **Availability**: services are always online.

Meeting these requirements could be achieved by: a) separating a vehicle's software into categories/layers and applying security mechanisms to each of these layers (which is an IoT security strategy discussed in section 2.1.2), and b) using security hardware [11].

Security Layers: Hu et al. [11] categorize vehicle security into four layers, namely, a) individual ECU layer, b) IVN layer, c) gateway layer, and d) firewall layer. Furthermore, by reviewing the IoT software layers presented in Killeen [14], it can be seen that applying software security to these four security layers will secure the IoT perception layer, and to secure an entire fleet system of vehicles, applying security mechanisms to the middleware and application layer is required. The middleware layer performs analytics and can store data from a fleet of vehicles (this layer could contain firmware repositories, for example). The application layer is the presentation layer used to display vehicle aggregates and other information to the user (for example, a user may receive a text from this layer, notifying the user to bring their car into the garage). Hu et al. [11] further discuss that security hardware in a vehicle must be able to secure microcontrollers by providing secure bootstrapping, firmware protection, and key distribution, which are security mechanisms that fall into the ECU security layer. Figure 4 illustrates a visual diagram of applying security mechanisms to these layers.

3.1 Threat Model

This section defines the threat model the present work focuses on for protecting vehicles (and IVNs) against threats. According to van Oorschot [33] (pp. 11 – 18), a threat model includes the assumptions made about the abilities of attackers, attack vectors, and other threats to consider when designing the security for a system.

A literature review [43], [25], [2], [26], [42] reveals the attack surfaces against vehicles that are discussed below, and is further discussed in sections 3.1.2, 3.1.3, and 3.1.4.

- **Long-range wireless**: an attacker attacks the vehicle from long range, for example via radio or cellular networks (which discussed further in section 3.1.2).
- **Short-range wireless**: a nearby attacker could attack the car via Wi-Fi, Bluetooth, RFID, or the vehicle's cellular network (which is discussed further in section 3.1.3).
- **Physical access**: an attacker attacks via a physical interface such as the CAN bus, OBD-II port, or USB port, for example (which is discussed further in section 3.1.4).

According to Li et al. [21], attacks can typically be categorized as either internal or external attacks; that is:

- **External attacks** attempt to compromise the TCU using an external remote connection. Once compromised, the TCU can be used to mount attacks against the ECUs on the same IVN as the TCU.
- **Internal attacks** attempt to break into the vehicle via the infotainment system or another internal vehicle system. Similarly, once the internal component is compromised, it can be used to attack any IVN it is connected to.

By examining the nature of these categories of attacks, it can be seen that external attacks may be conducted via long-range wireless or short-range wireless channels (over cellular data connection), while internal attacks may be conducted via short-range wireless channels or physical access.

3.1.1 Attacker Abilities. The assumptions made about attacker abilities are defined as follows; that is, an attacker: a) cannot brute-force a key, b) can extend the range of short-range wireless signals by using an amplifier [2], c) can reverse-engineer the meaning and nature of proprietary data found in CAN messages, by investing time and using tools such as LibreCAN [27], d) has indirect physical access to the vehicle (via an infected smart phone or malicious music file, for example), and e) can brute-force a Bluetooth PIN in an online fashion. Although it may initially appear unreasonable to assume an attacker can crack a vehicle's Bluetooth PIN in reasonable time, this assumption is made since Checkoway et al. [2] cracked a PIN in less than 13.5 hours, and they calculated that when in Bluetooth proximity of 1000 cars (in a garage, for example), the expected amount of time to crack the PIN (in an online fashion) is 1 minute. Section 3.4.2 provides more details on this work.

3.1.2 Long-range Wireless Attacks. This section presents details on long-range wireless attacks. These types of attacks involve attacking from a remote distance and can be classified into two categories, namely, a) attacks via **broadcast channels** (for example, satellite, FM and AM radio, and GPS), and b) attacks via **addressable channels** (for example, remote telematic systems). An attacker that attacks via broadcast channels may be a few kilometers away from the vehicle, while an attacker that attacks via addressable channels may be at even greater distances away from the vehicle (the distance depends on the cellular network infrastructure) [2]. Example 3.1 and 3.2 below are both examples of long-range attacks via addressable wireless channels.

Example 3.1. Long-range Wireless Attack - TCU Exploitation

An attacker attacks a vehicle, V_a, from a different country by taking advantage of a vulnerability in V_a's TCU. To attack the TCU, an attacker gains access to the fleet's cellular network by connecting a malicious device to the Wi-Fi hotspot of another vehicle, V_b. From V_b's Wi-Fi network, the attacker can attack V_a's TCU (or any other vehicle online and connected to the fleet's network). Note that Miller et al. [25] conducted a similar attack, which is discussed in section 3.4.4.

Example 3.2. Long-range Wireless Attack - Malicious Firmware Update

An attacker compromises a firmware image private signature key, allowing the attacker to sign malicious images and push malware updates onto a fleet of vehicles [19].

3.1.3 Short-range Wireless Attacks. This section presents details on short-range wireless attacks. These types of attacks involve attacking from a relatively close distance (1 to 100 meters away, for example). A few details are discussed below on the various attack vectors via short-range wireless mediums:

- **Bluetooth**: one of the most popular attack vectors for vehicle hacking.

- **Wi-Fi**: some luxury vehicles offer a Wi-Fi hotspot by using cellular data (for mobile Internet connection). An attacker that gains access to a vehicle's Wi-Fi network can then attack the vehicle's open-port services.
- **Cellular network**: attacking the car via a cellular network could involve using a femtocell[4] to force the vehicle to connect to a rogue cellular tower, at which point the vehicle can be attacked by malicious devices connected to the femtocell.
- **RFID**: since modern cars have keyless ignition based on RFID and a key fob, an attacker could target the RFID ignition system.

3.1.4 Physical Access Attacks. This section presents details on physical attacks. These types of attacks involve physically attacking the vehicle, either directly or indirectly, and assumes the attacker has some form of physical access to the vehicle. **Direct physical access** involves attacking the OBD-II port or CAN bus directly, and **indirect physical access** involves attacking via an indirect medium, such as a malicious smart phone that infects the vehicle via a USB port or a malicious music file, for example [2].

Overestimating Attacker Abilities: in the literature, there are some that argue that the abilities of the attacker are overestimated [2]; that is, some believe it is not reasonable to assume an attacker has physical access to a vehicle. To support this point of view, the argument is that if an attacker had physical access to a vehicle, she could simply cut the brake lines instead of conducting a complex cyberattack against the vehicle. On the other hand, ignoring the threat of direct (or indirect) physical access could be a dangerous assumption. For example, an attacker could physically install a malicious device onto the OBD-II port, which would only take a few moments (a direct physical attack). This attacker could be a valet that attacks the vehicles of a restaurant's clients or a malicious mechanic at a garage. Another example is an innocent user may have malware installed on his or her smart phone (or a malicious music file), and upon connecting it to the USB port (or playing the music file), the malware could attack the vehicle (an indirect physical attack). Clearly, assuming an attacker has direct physical access to a vehicle gives the attacker a lot of power. However, assuming the attacker has indirect physical access is a much more realistic assumption. The assumption should therefore depend on the domain the vehicle security model is being designed for.

3.2 Attacks Against Vehicles

This section presents a brief list of possible attacks against vehicles and some issues with vehicle cybersecurity (note that section 3.4 goes into more detail about the types of attacks by presenting vehicle hacking examples). **Attacks** against vehicles include:

- Taking advantage of a vehicle's wireless network connection [43][25][42], lack IVN isolation [43], and/or lack of IVN gateway [27].
- Accessing the IVN without authorization [42].
- Turning off all the lights of a car while driving to cause unsafe driving conditions [24].
- Conducting a denial-of-service (DoS) attack against the IVN [42].
- Updating ECU firmware without authorization.
- Controlling the vehicle [42].
- Spoofing the speedometer [17]: such an attack could be conducted by a corrupt law enforcement officer, making the speedometer display slightly slower speeds than the actual speed. The motivation would be to hand-off speeding tickets to innocent drivers.

3.3 CAN Security

This section discusses about the security standards and issues of CAN.

3.3.1 Security Standards. This section lists some of the important standards regarding ECU access control and other policies that are expected to be enforced when manufacturing ECUs. A list of important CAN ECU security standards from Koscher et al. [17] follow:

- When a sensitive action (such as flashing firmware or reading memory locations, for example) is requested from some ECU, E_a, by an ECU (or PassThru device), E_b, on the CAN bus, the following sequence of events occurs: a) E_a sends a challenge (a seed) to E_b, b) E_b applies the cryptographic challenge-response algorithm and sends back the response to

[4] A femtocell is miniature modem that simulates a cellular network tower, which can be used to maliciously force cellular network devices to connect to it or to boost the cellular network connection strength of honest devices [25][40].

E_a, c) E_a compares the response with the key paired with the challenge/seed, and d) when the response is correct, the sensitive action is taken by E_a, otherwise no action is taken, since E_b does not have the authorization.
Note that E_a only stores the challenge-response pairs and avoids storing the challenge-response algorithm. As such, ECUs should store seed-key pairs for satisfying the challenge-response algorithm required for authentication. This is an interesting standard, since typically the reverse is done in practice; that is, the algorithm is usually public while the key is private.

- It should not be possible to flash/update ECU firmware when the engine is running.
- When a firmware flash request is received by an ECU over the CAN bus, a challenge-response scheme should be followed for authentication.
- Some parts of ECU memory (where the keys or programs are stored, for example) can be defined as inaccessible (via the CAN bus) under any circumstances; that is, memory at these locations cannot be read over the CAN bus, only the ECU's processor can read the sensitive memory locations.
- ECUs from low-speed/low-importance CAN buses (those that control the radio, for example) cannot be used to flash ECUs in high-speed/high-importance CAN buses (those that control mission-critical equipment like the engine, for example). This is presumably the case to avoid having the CD player turn off the engine accidentally (or maliciously).

3.3.2 Security Challenges and Issues. This section discusses some challenges and issues related to CAN security. A literature review [11], [43], [42], [5] reveals that the CAN bus tends to suffer from the following **issues**:

- It was not designed to be secured against hackers.
- It has no built-in form of message authentication or encryption; that is:
  - ECUs are not able to verify the authenticity of the sender, since the CAN message ID does not represent the identity of the sender (the message ID instead represents the purpose of the message).
  - The small packet size of the protocol limits message authentication support; for example, AES-128 cannot be used to encrypt a CAN frame's data field, since the data field is 8 bytes in length while AES-128 has a 16-byte block size.
- It does not support network isolation features.
- It is more difficult to apply standard network security mechanisms, since CAN lacks addressing.
- Eavesdropping: since CAN is broadcast-based, any rogue ECU may tap into the network and listen.
- It has a wire arbitration weakness: a rogue ECU could continuously send high priority messages to force other ECUs to back-off from the wire indefinitely.
- It has a lack of freshness; that is, there are no timestamps in CAN messages, so an attacker could replay old messages without being easily detected.

It is worth noting that CAN has security by obscuration, since generally CAN message payloads are proprietary, therefore, reverse-engineering may be required, which is tedious and not scalable [27]. Nevertheless, it was shown to be still possible to attack a vehicle even with this obscuration (section 3.4 goes into detail about such examples).

Furthermore, vehicles tend to have multiple IVNs for different functions, and these IVNs are usually all connected/bridged together to support numerous telematic system services (such as the real-time diagnostics offered by GM's OnStar, for example). As such, compromising less critical components can lead to compromising mission-critical components, since they are connected to each other. This illustrates the challenge of the features vs. security trade-off. Without having all the components connected together, the vehicle would be less vulnerable; but on the other hand, there are features (especially infotainment-related) that would be impossible to implement [17], and this would make it more difficult to sell a vehicle, since these luxury features have market value.

Firmware Updates: the challenge of updating IoT device firmware and managing firmware image repositories extends to vehicle security. Secure and efficient ECU firmware updates is not a trivial problem to solve.

When vulnerabilities in ECU software are discovered, it is required that all the ECUs in a fleet of vehicles be updated [19]. The standard in the industry is to store signed ECU firmware images in remote repositories. Firmware images may be paired with metadata, and at times, vehicle manufactures may not force the signing of the metadata in their security model. As such, an attacker could maliciously alter the metadata of an ECU's firmware update. Therefore, it is necessary to sign the metadata, and this can be done using either offline signing, online signing, or a combination of both. There is no best signing method, since they each have advantages and disadvantages, which are discussed below:

- **Online singing** supports frequent firmware updates, since singing can be done in an automated fashion. However, an attacker could install malware onto a fleet of vehicles by compromising the private signature key; that is, online signing requires less computational effort by the attacker but is more automated.
- **Offline signing** requires more computational effort by the attacker to install malicious firmware onto vehicles, but this approach is less automated, since it requires more human intervention. This means frequent firmware updates are not trivial when using offline singing, but it makes it harder for attackers. An example of an offline signing scheme is found in example 3.3.

Example 3.3. Offline Firmware Signing Model

Vehicles are required to be brought into the garage to get firmware updates as part of the maintenance procedure. This has the advantage of avoiding malicious (or accidental) firmware updates that 'brick'[5] the vehicle. An en-route vehicle that suffers from a failed firmware update, and as a result gets bricked, could be very dangerous. On the other hand, when a vulnerability is discovered, the process of bringing a vehicle to the garage would be time-consuming, and attackers could take advantage of the grace period where a fraction of the vulnerable vehicles did not have a chance to be brought into the garage for the vulnerability update.

3.4 Hacking Examples

This section discusses successful attacks against vehicles that are found in the literature. Before 2011, there were no reported instances where a remote attacker could take control of a vehicle. In 2010, Koscher et al. [17] (discussed in section 3.4.1) were able to control a vehicle via physical access to the OBD-II port. In 2011, Checknoway et al. [2] (discussed in section 3.4.2) successfully gained full control of a Sedan via Bluetooth, cellular network, the OBD-II port, and a vulnerable music player. In 2013, Miller et al. [24] (discussed in section 3.4.3) hacked a Ford Escape and Toyota Prius via physical access, and they were able to control most of the vehicle. Finally, in 2015, Miller et al. [25] (discussed in section 3.4.4) were able to remotely exploit and control a vehicle. A few other successful attacks in the literature are presented in section 3.4.5.

3.4.1 Koscher et al. 2010. Koscher et al. [17] found in their experiments that their vehicle did not follow all the standards detailed in section 3.3.1. The authors were able to: a) flash the ECUs when their car was moving, without authentication, b) read the reflashing keys from memory by first authenticating with a key (it should not be possible to read reflashing keys by design of the standard) and in another instance they read the entire memory (including the keys) of an ECU without authentication, and c) flash an ECU on a high-speed IVN using an ECU from a low-speed IVN.

Their attack **methodology** is as follows: a) they sniffed the CAN bus while performing deterministic actions (for example, turning on the headlights). Replay attacks confirmed the actions that could be performed by writing a special packet to the CAN bus, and the actions that could not be performed; and b) they performed fuzzing[6] to observe potential effects on the vehicle and to help reverse-engineer the system. After applying this methodology, by plugging in a laptop into the OBD-II port of the car, they were able to:

- Control the radio and deny the user access to the radio controls.
- Spoof the fuel gauge and speedometer display.
- Control the door locks, the trunk lock, the headlights, the horn, and other components on the vehicle's body.
- Control parts of the engine (for example, increasing the RPM temporarily). One important note is that they were able to turn off the engine when the car was running at 40mph.
- Control the brakes.
- Control the heating and cooling.
- Prevent the car from turning off or being turned on.

Disconnecting the laptop from the CAN bus reverted the car back to normal.

3.4.2 Checknoway et al. 2011. Checknoway et al. [2] explored many attack vectors to control a late model Sedan. The successful attacks they performed are summarized below:

- **Physical attacks**: they reverse-engineered a vehicle's media player and discovered a buffer overflow vulnerability. By using an unused UART interface on the media player, they were able to access a debugger that gave them the insight required to take advantage of the buffer overflow, which allowed them to execute arbitrary code by playing a malicious

[5] Bricking is computer science slang that means a node becomes completely and permanently unresponsive (like a brick).

[6] Fuzzing involves supplying random inputs to a system with the goal of testing, finding bugs, and observing other unexpected behavior [29][33].

music file. The authors were also able to compromise a PassThru device via shell command injection and a weak root password.
  - What is important to note about the music file is that the malicious WMA file was played correctly by a PC, but when played by the vehicle's media player, the file could be used to send arbitrary CAN messages on the IVN.
- **Short-range wireless attacks**: the authors reverse-engineered an ECU to gain access to a Unix-like OS, and that allowed them to examine the Bluetooth implementation. They found custom Bluetooth code that had '`strcpy`' calls (that is, was weak to buffer overflow attacks, which are discussed in section 3.5.4). They could attack the vehicle's Bluetooth ECU in the following two situations:
  (1) When a malicious device was already paired to the car. For example, a virus-infected smart phone that is paired with the vehicle via Bluetooth could be used to mount an attack against the vehicle.
  (2) Without the pre-paired device, by cracking the Bluetooth PIN in an online fashion. They found that pair requests could be done without any user interaction, which means brute-forcing the pairing PIN was possible if prolonged access to a running vehicle was available. The online PIN cracking time was directly linked to the response time of the vehicle's Bluetooth ECU. They were able to crack the PIN in at most 13.5 hours and at least 0.25 hours in their experiments. They also note that this type of attack can be conducted in parallel against many vehicles, reducing the expected PIN cracking time of a vehicle to one minute, if 1000 vehicles are being attacked in parallel and the goal is to crack any PIN from amongst the 1000 vehicles (an attacker could be in a popular garage, for example).
- **Long-range wireless attacks**: the authors reverse-engineered the aqLink protocol used by the vehicle's TCU (a Airbiquity aqLink modem), and they found a buffer flow vulnerability. They were not able to directly exploit the vulnerability due to the slow bit rate of the modem (although they were later able to exploit it by taking advantage of another vulnerability). The authors decided to target the authentication system between the vehicle's TCU and the command center. They found that nonce[7] challenges were generated from a deterministic random number generator; that is, each time the car's TCU restarted, the random number generator reset as well. Hence, the system was weak to replay attacks. They also found that for 1 out of every 256 challenges, a carefully crafted incorrect response could be interpreted as correct.

Using all the above vulnerabilities, the authors were able to gain full control of the vehicle by taking advantage of the long-range wireless vulnerabilities and by using a malicious music file found on an iPod.

3.4.3 Miller et al. 2013. Miller et al. [24] perform their security research on a 2010 Ford Escape and 2010 Toyota Prius. The authors wrote their own ECOMCat software (in the C programming language) to read/write from/to the CAN bus using an ECOMCat cable. The authors analyzed the CAN architecture of both vehicles, and they found that some functionality of the vehicle could not be controlled (this notion is discussed below).

The authors identify a few **challenges/issues** when attacking the vehicles namely:

- Some components cannot be controlled from the OBD-II port, since they are not connected to the port; for example, the acceleration could not be controlled since the only module that automatically controlled the acceleration was the cruise control (which was directly connected to the ECU that managed acceleration).
- Not all CAN messages lead to a direct action by the vehicle. Some CAN messages were just reporting the state of the vehicle (for example, how much a vehicle has decelerated). As such, it is not as simple as replaying any message to control the vehicle. It requires time and effort to reverse-engineer CAN messages and examine how they affect the car.
- Messages that have an effect on actions taken by the car may not be trivially spoofed, since other honest ECUs may also be broadcasting the authentic version of the spoofed message. This creates message conflict that may or may not allow a spoofed message to control the car.
- At times, ECUs have simple rules stating that packets should only be processed in certain vehicle states; for example, the steering wheel should only rotate in response to an automated parallel park packet when the vehicle is in reverse, meaning all spoofed packets get ignored when not in reverse.
- A spoofed packet may lose the message arbitration contest, and as a result, the packet would get ignored.

The authors were able to conduct the following **successful attacks via the raw CAN bus** against one (or both) of their experimental vehicles:

- Change the dashboard display to incorrectly show that the doors were open even though they were closed.
- Spoof the speedometer display on the dashboard (Toyota and Ford).

[7] Number only used once.

- Spoof the odometer by flooding the bus with a series of increased spoofed readings (Ford).
- Fool the navigation system with spoofed locations (Ford).
- Flood the bus with a CAN message with ID 0, causing a DoS attack against the vehicle. The steering wheel became hard to rotate, and would not rotate more than 45 degrees (Ford).
- Control the steering wheel angle at speeds below 5mph (limitless control for the Toyota), which is accomplished by taking advantage of the parking assist system (Ford).
- Control and lock the brakes by taking advantage of the collision prevention system (Toyota).
- Control the acceleration briefly after pressure on the gas pedal was released (Toyota).
- Control the steering wheel completely by taking advantage of the lane assist system (Toyota).

The authors were able to conduct the below successful **attacks using diagnostics**. The authors accomplished these attacks by first reverse-engineering the Ford Integrated Diagnostics (FID) software to obtain a secret key that was shared between the FID and the ECU. They were able to acquire an entire list of keys from the FID software. This gave them access to sensitive ECU functions, such as reading out ECU memory, for example. To obtain the secret keys required for performing diagnostic actions to the Toyota, they reverse-engineered the Toyota Calibration Update Wizard software.

- Engage the brakes to prevent acceleration, which only worked on stopped cars (Ford).
- Flush the brakes, rendering the brakes useless (Ford).
- Turn off all the lights of the car, which includes the radio, brake lights, and interior lights. This only worked on stopped cars, but it persisted even when the car started to move (Ford).
- Kill the engine at any speed (Ford).
- Force the interior lights to permanently blink until the ECU is reprogrammed (Ford).
- Kill the engine (Toyota).
- Turn the headlamps on and off (Toyota).
- Turn the horn on and off (Toyota).
- Tighten the seat belts at any time (Toyota).
- Lock and unlock doors (Toyota).
- Control the fuel-gauge meter (Toyota).

The successful **firmware update attacks** the authors performed are discussed below, which involved reprogramming ECUs.

- Ford: the authors found an ECU that runs on a Motorola HCS12X microprocessor, and examined its memory using a debugger. They were able to discover what function calls were required to write and read to/from the CAN bus, and they were able to execute arbitrary code on the ECUs. This was accomplished by taking advantage of the *RequestDownload* ECU service.
- Toyota: they reverse-engineered the update process. The two ECU processors under attack were a NEC v850 variant and Renesas M16/C. They reverse-engineered how the secret keys for the firmware updates were generated.

3.4.4 Miller et al. 2015. Miller et al. [25] analyzed the security of a Jeep. Their goal was to write CAN messages to potentially control the vehicle.

One of the **challenges** they faced, which is similar to one of the challenges discussed in section 3.4.3, is that flooding the CAN bus with a spoofed message may or may not have resulted in the target ECU interpreting the spoofed message correctly. Furthermore, some ECUs may have stopped performing actuation if there were inconsistencies in messages (contradicting messages in small window of time), while other ECUs did not have this defence mechanism (they simply took action based on the last message received).

They were able to conduct the following **short-range wireless and indirect physical access attacks**:

- They found that the on-board vehicle Wi-Fi hotspot was vulnerable to exploitation. By analyzing the binary of the OS of the UConnect system, they were able to reverse-engineer the code that chose the default password of the WPA2-secured Wi-Fi. They found the password was generated based on the current time, which was always fixed at the same value, since the Wi-Fi module booted-up before the vehicle had time to receive the actual time from the command center.
- After cracking the password in an online fashion, they scanned the Wi-Fi network and found many open ports. Among the services was a D-bus service, where the authors could send commands to the daemon.
- They were able to gain root access by updating the OS of the UConnect system, which they accomplished using a fraudulent version of the OS found on a USB drive. The fraudulent version had the root password changed, so they could

then enable SSH. Although, they found that gaining root access to head unit (the ECU that is connected to all CAN buses, a.k.a the gateway) was not necessary. Via Wi-Fi, using the D-Bus port to access the 'NavTrailService' service, they were able to execute any command with root privileges using the 'execute' feature of D-Bus. In other words, the head unit suffered from over privileges.

To summarize, the authors were able to execute arbitrary commands on the head unit via USB or Wi-Fi with root privileges. To further generalize their attack, they showed that they were able to access the D-bus service via telnet using a laptop (or smart phone) connected to the vehicle's Sprint[8] network, which was a **long-range wireless attack**. They were able to control the Jeep remotely using CAN messages by applying the following methodology:

(1) Identify vulnerable vehicles by scanning the Sprint cellular network.
(2) Exploit the D-bus service to enable the SSH service.
(3) Update the firmware on board the Jeep to allow transmitting arbitrary CAN messages over the CAN bus (requires a reboot).
(4) Use an SPI interface to write CAN-messages to the CAN bus.

Steps (1) and (2) enabled remote control of the on-board infotainment system (volume control, for example), and steps (1) to (4) enabled physical control of the vehicle. Note that they did not attack other vehicles on the Sprint network for ethical reasons; they only exploited their own vehicle.

3.4.5 Other. This section briefly introduces other successful attacks against vehicles in the literature. Möller et al. [26] state that security researchers were able to reverse-engineer the OS running on a vehicle. The researchers were able to identify what version of the Bluetooth library was used by the vehicle. They found the library was a popular embedded Bluetooth library, which contained buffer flow vulnerabilities. These buffer overflows enabled arbitrary code execution on the TCU of the vehicle. A literature review [23], [2], [26] reveals that researchers from the University of California San Diego and the University of Washington were able to: a) execute arbitrary code on a vehicle's TCU, by exploiting a Bluetooth vulnerability in the ECU responsible for managing Bluetooth, b) compromise the vehicle with a malicious CD, and c) compromise a vehicle remotely via a cellular network connection.

3.5 Defence Mechanisms

This section discusses various security mechanisms that can be applied to protect a vehicle from cyber threats. According to Möller et al. [26], a vehicle security solution should be able to detect, deter, and avert a cyberattack against a vehicle. Furthermore, vehicle security solutions can be categorized into three categories [11]:

(1) Message authentication.
(2) Data encryption.
(3) Intrusion protection.

3.5.1 Encryption and Authentication. A literature review [21], [11] reveals that encryption and message authentication can be used to secure CAN, although, resource constraints should be kept in mind, since ECUs may not be powerful enough to run a cryptographic algorithm. The bus load, network latency, and key management requirements must also be considered.

According to Hu et al. [11], CAN may be secured using message authentication codes (MACs) and digital signatures, which could be embedded into the data frames. However, care must be taken when using such techniques, since computing MACs and digital signatures will introduce latency, which may defy the real-time nature of the CAN bus. Furthermore, the data field of IVN frames are limited in size. For example, a CAN frame only has an 8-byte data field, so storing a 16-byte MAC is not directly possible. To address this, the MAC could be truncated by only taking the first 8 bytes of the MAC. This would be at the expense of the authentication security strength, since the MAC would have fewer bits when using this approach. Another option could be to send multiple messages to send the MAC/signature in its entirety, but this would increase the bus message rate. For example, to send a 16-byte MAC, two CAN messages would need to be sent, increasing the bus load by at least a factor of 2. One may consider using CAN FD for the 64-byte data field to address the data field length issues.

To summarize, in terms of **technical requirements**, Hu et al. [11] state the following points should be considered when applying encryption and authentication security mechanisms to vehicles:

(1) Cryptographic algorithms can be improved via hardware, and should be easily applicable to the target IVN (do not apply AES-128 with CAN, since the 16-byte block size is too big for CAN's data field, for example).

[8] cellular provider of the vehicle fleet [25].

(2) The bus load rates must be considered when choosing a cryptographic algorithm; that is, twice as many messages may be required on the IVN bus to apply frame-level encryption and message authentication, which may exceed the IVN's bandwidth.
(3) The real-time requirements of the IVN should be evaluated, given the additional network latency from the security mechanisms.
(4) Key management: keys should be securely stored, securely generated, and multiple keys should be used for many groups of ECUs.

3.5.2 Gateway. According to Pesé et al. [27], CAN can be secured by adding an IVN gateway (or head unit in some literature [25]). The gateway can host the OBD-II port, perform routing (preventing unwanted CAN messages, originating from the OBD-II port, from reaching undesired IVNs, for example), partition the IVNs, host firewalls, and enforce (either frame-level or application-level) access control lists [27][11]. Furthermore, the gateway should be configured to separate mission-critical IVNs from low-importance IVNs (separating a DVD player from the engine, for example) and separate other trusted vs. untrusted network components. The gateway is also a candidate device for running an intrusion detection system (see section 3.5.3) [11].

By reviewing the example IVN in figure 1, it can be seen that the gateway in this example enables many of the security features mentioned above.

3.5.3 Intrusion Detection System. A literature review [42], [11] reveals that intrusion detection systems (IDS) can be deployed to secure CAN. IDSs should be able to identify attacks, reduce attack damages, prevent attacks, restore the system into a stable secure state, and publish notifications upon attack detection (for example, sending a text message to a security analyst about a potential breach). One of the advantages of using an IDS is that the IVN structure need not be changed, since an IDS can simply read IVN traffic in a non-intrusive way (read-only) to detect attacks [21].

An IDS can either be deployed on gateways (a host-based IDS), on ECUs (a distributed IDS), or using a combination of both. Furthermore, there are two **types** of IDSs, namely, behavior-based and knowledge-based IDSs.

- **Behavior-based IDSs** analyze the statistical behavior and patterns of the network, and since data-drive predictive analytics models are robust at detecting new types of events/data [14], it can be inferred that behavior-based IDSs would best be used for detecting novel kinds of attacks.
- **Knowledge-based IDSs** use pre-defined expert knowledge that is represented in the form of rules, and these rules are enforced by a rule-engine. Furthermore, since knowledge-based predictive analytics models can only be used to detect pre-existing types of events/data [14], it can be inferred that knowledge-based IDSs would best be used for detecting known attacks. Furthermore, manufacturer usage description (MUD) files can be used as a knowledge-based IDS. MUD files specify what protocols an IoT device is expected to use, and how it will communicate. When the device deviates from its expected behavior, it can be flagged as potentially malicious. MUD files can also be used to prevent attacks by using the MUD file as firewall policies. If a device breaks the firewall policy, the communication of the device can be blocked [32].

In fact, combining both types of IDS would likely yield the best detection accuracy [11].

Unfortunately, there are **challenges** when deploying IDS to IVNs, namely:

- **ECU hardware constraints**: the ECU may not have enough resources to run an IDS.
- **Monetary cost constraints**: it may be infeasible, from a business point of view, to deploy an IDS.
- **Accuracy**: the IDS predictions may not be reliable enough to deploy the IDS onto mission-critical equipment. Furthermore, an important aspect to consider is the false positive (FP) rate of an IDS. A FP reading could potentially put the vehicle into a dangerous state if authentic behavior is incorrectly flagged as malicious and is blocked as a result.
- **Response time**: the speed at which predictions are made may not meet the real-time requirements of the system.
- **Standardization**: a lack of standardization makes implementing and deploying IDS troublesome.

Furthermore, IDSs cannot be used as the only defence mechanism on a vehicle, since they typically only detect the attacks (they do not prevent them). Furthermore, the IDS field of research also has open-ended issues, which include: a) improving IDS accuracy and response time, b) bridging the gap between the automotive functional safety and cybersecurity engineering for IVNs, c) securing IVNs consisting of heterogeneous hardware from a variety of vendors, d) securing multiple layers of IVNs, and e) applying machine learning to resource-constrained IVNs and obtaining datasets of intrusions.

Hu et al. [11] summarize the **policies** that a vehicle IDS should have, which are as follows:

- Real-time requirements of the IVN are met.
- Detection rates of the IDS should also be considered.

- The network capacity should be considered.
- It should be easy to update the IDS to support new policies when new types of attacks are defined.

IDS Prevention Examples: an example of having an IDS prevent an attack follows: Foster et al. [5] state that one of the ways to protect against a message flooding DoS attack involves ECU self-detection (a distributed IDS). A well-designed ECU will self-detect (detecting message contradictions, for example [25]) that it is being used to conduct a DoS and will enter a read-only mode to stop the attack.

3.5.4 Buffer overflow patching. Buffer overflows are popular vulnerabilities in C code. They involve copying a source input buffer into a destination buffer that is too small, resulting in stack corruption. A well-crafted input buffer could be malicious machine code that corrupts the return pointer and has the execution flow return to an attacker-defined function in the corrupted stack. Other types of buffer overflows exist, but such details are beyond the scope of the present work. Interested readers are referred to van Oorschot [33] (pp. 166 – 179) for a good technical review on buffer overflow vulnerabilities.

According to Möller et al. [26], making sure there are no buffer overflow vulnerabilities is important when programming ECU logic. This can be accomplished by using: a) code auditing, b) compiler tools (for example, StackShield, StackGuard, and Libsafe), c) non-executable stacks, and d) patching regularly. If possible, avoiding the C language all together could also prove to be a good solution.

3.5.5 LibreCAN. Reverse-engineering is one of the bottle necks in vehicle security research. The different CAN message structures/encodings, which depend on the manufacturer of the vehicle, make it time-consuming to perform security research. A vehicle made by one manufacturer most likely will not have the same CAN message encoding as another manufacturer's vehicle [12] (other than the public OBD-II diagnostics standard). This makes it hard to apply/generalize research results to a variety of cars. Pesé et al. [27] propose LibreCAN, an automated method for reverse-engineering CAN messages and used to infer the DBC files of the vehicle's CAN message architecture. LibreCAN gathers sources of truth by reading OBD-II data and smart phone kinematic motion data (where the smart of phone is installed inside the vehicle and is properly oriented), and LibreCAN reads proprietary CAN bus data. Using these data sources, LibreCAN cross-references the data to compile a reverse-engineered DBC file. This DBC file helps decode proprietary CAN messages, which facilitates IVN security research. What is interesting is that LibreCAN could also be used as a hacker tool. Furthermore, Jaynes et al. [12] propose a similar methodology for reverse-engineering proprietary CAN messages using machine learning.

3.5.6 Uptane. As mentioned previously, updating the firmware of ECUs on a vehicle is one of the challenges of IVN security. To address this challenge, Kuppusamy et al. [19] propose Uptane, a compromise-resilient security architecture for updating firmware on ECUs. Uptane balances the trade-off between patching convenience and security; that is, how easy it is to patch ECU firmware vs. the amount of time and effort an attacker has to invest to install malware onto the ECU. Some critical ECUs need to have the guarantee that malware has not been installed on them. This requires intensive computations from offline firmware image signing and verification. On the other hand, one may be able to afford saving computations from the firmware patching of ECUs that are of less importance to the mission-critical nature of the vehicle, and this can be achieved by relaxing key management and using online firmware signing at the cost of security. Uptane accomplishes this trade-off by using a series of firmware image signing keys, and by putting different importance onto each one.

However, a criticism about the security of this work can be made. In the hacking examples of the present work, which are discussed in section 3.4, many researchers were able to fully control an entire vehicle by simply breaking into a non-critical ECU (a CD player, for example). Although it is convenient to be able to manage the trade-off between the convenience of automated firmware patching vs. the strength of anti-malware installation mechanisms, if an attacker can simply break into the most vulnerable ECU (the weakest link), then Uptane provides no additional security (assuming the weak ECU can be used to control the entire vehicle); this is the case unless each ECU requires complete offline firmware imagine signing for the strongest security against malware updates. Nevertheless, from a theoretical point of view, Uptane remains a strong contribution assuming most of the security requirements of IVN security are met; for example, with IVN isolation, a CD player could not fully control the vehicle, it could only control the music player's volume and other music features.

3.6 IVN Security Comparison

This section presents a brief security comparison of each of the IVNs discussed in section 2.2. It can be seen that the technical differences between the IVNs (which are discussed in section 2.2.7) can be attributed to the differences in security concerns of the IVNs.

Hu et al. [11] state that MACs can be shorter in length compared to digital signatures. Therefore, they suggest using MACs for CAN and LIN, since both CAN and LIN have short data fields, and they suggest using digital signatures for Ethernet, FlexRay, and CAN FD due to their longer data field lengths. It is not clear what message authentication mechanisms can be used for MOST, since MOST's data field length varies in size depending on what MOST protocol is chosen at each OSI layer. Furthermore, CAN and LIN cannot use AES-128 to encrypt their data field, since AES-128 requires 16 bytes per block, and CAN and LIN frames only support 8-byte data fields. In general, the literature on IVN security focuses on CAN security, which may be attributed to the fact that CAN has been mandatory since 2008.

3.7 Summary

This section summarizes general techniques and strategies that can be used for protecting vehicles against common cyberattacks found in the literature. The defence strategies follow:

- Deploy a combination of a knowledge-based and behavior-based IDS on the vehicle.
- CAN ECU standards should be followed and authentication should be required for critical actions [17].
- Algorithms that generate nonces should never re-use the same nonce. Even after turning off the vehicle, an internal state should be maintained to avoid reuse of randomly generated numbers for challenges.
- Buffer overflow vulnerabilities should be avoided; that is, length checks should always be performed on external input parameters and tools should be used to detect buffer overflows.
- ECU firmware and metadata about the firmware should be signed and verified before flashing/updating the ECU. Furthermore:
  - The firmware image repositories should also be secured by using a variety of keys for online and offline signing.
  - Using a single key for online firmware signing should be avoided, since it could lead to an attacker installing malware onto a fleet of vehicles.
- Bluetooth pairing requests should require interactions from the user [2].
- Relying on security by obscurity should be avoided.
- Every device/ECU in a vehicle should be secured. It is not sufficient to secure a subset of the devices, since in general, any device can be used to fully control a vehicle.
- Writing known protocols from scratch should be avoided; instead, use popular open-source libraries that do not suffer from known vulnerabilities.
- Avoid exposing services on open ports (telnet, for example) [2]. Furthermore:
  - ECUs and gateways typically do not need open-port services for the vehicle to function.
  - The TCU should avoid accepting incoming service requests; instead, it should periodically ping the command center for pending requests.
- Implement IVN frame encryption and message authentication.
- Avoid exposing UART and other debugging interfaces when possible.

4 CONCLUSION

With the popular trend of IoT, an increasing number of resource-constrained heterogeneous devices are being deployed into the environment. These devices can sense and act upon their environment, creating a series of insightful data streams that may be analyzed for knowledge discovery. Among these IoT devices are the microprocessors found on board vehicles. Modern vehicles come with a variety of different features including remote diagnostics, accident prevention, and weather-related applications. These services are only possible with the help of the streams of sensor data and the series of actuators found on board vehicles. Furthermore, to support these features, vehicles need a cellular network connection for Internet access and an array of interconnected ECUs, which manage the physical components of the vehicles. As a result, with all this on-board vehicle technology, contrary to popular belief, vehicles are vulnerable to cyberattacks. Not only is data privacy a concern, but loss of life can also occur from a successful cyberattack against a vehicle. Therefore, it is important to understand the security requirements and issues of modern vehicles and their IVNs, which is closely related to IoT security research.

The present work provides an in-depth literature review of the related work and the state of the art of IVN and CAN bus security in the context of IoT security. IVNs include: CAN, FlexRay, MOST, LIN, and Ethernet, and a comparison of the technical details of each of these IVNs is provided. Note that the present work focuses on CAN security and only briefly covers the other IVNs. Furthermore, enabling vehicular technology is discussed, which includes vehicle telematic service systems, and diagnostic hardware and software. A review of vehicle security literature helped define a threat model for IVN security. This

threat model lists the assumptions made about the ability of the attacker and lists the attack surfaces found on a vehicle. In particular, the security of CAN is explored, which includes its security issues and its important security standards. A series of successful vehicle attacks and their details are also found in the presented work. The details include attack vectors used, vulnerabilities discovered, and methodologies applied in the vehicle hackings. The present work finishes by presenting defence mechanisms that can be used to protect vehicles against common cyberattacks and vulnerabilities found in the literature.

In terms of future work, the present work could be strengthened by: a) investigating the security of the other IVNs, since CAN was the main focus in the present work; b) adding a more detailed comparison of the IVNs and their security-related concerns; c) performing more research into tools like LibreCAN that make reverse-engineering CAN messages easier with automation, since these types of tools are interesting and seem to hold a lot of potential for vehicle security research; and d) doing a survey on security work that have used LibreCAN or similar tools in their studies.

REFERENCES

[1] Charles Bell. 2016. *MySQL for the internet of things*. Apress, 1–28.

[2] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. 2011. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. Volume 4. San Francisco, 447–462.

[3] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. 2014. A large-scale analysis of the security of embedded firmwares. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 95–110.

[4] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. 2015. A search engine backed by internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 542–553.

[5] Ian Foster and Karl Koscher. 2015. Exploring controller area networks. *USENIX Association ;login:* 40, 6.

[6] Andy Greenberg. 2015. Hackers remotely kill a jeep on the highway-with me in it. Retrieved 11/09/2020 from https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[7] Andy Greenberg. 2015. Senate bill seeks standards for cars' defenses from hackers. Retrieved 11/09/2020 from https://www.markey.senate.gov/news/press-releases/senators-markey-and-blumenthal-reintroduce-legislation-to-protect-cybersecurity-on-aircrafts-and-in-cars.

[8] Christopher Greer, Martin Burns, David Wollman, and Edward Griffor. 2019. Cyber-physical systems and internet of things. *NIST Special Publication*, 1900.

[9] Ing Andreas Grzemba. 2012. *MOST: the automotive multimedia network*. Franzis Verlag.

[10] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. 2016. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*, 461–472.

[11] Qiang Hu and Feng Luo. 2018. Review of secure communication approaches for in-vehicle network. *International Journal of Automotive Technology*, 19, 5, 879–894.

[12] Michael Jaynes, Ram Dantu, Roland Varriale, and Nathaniel Evans. 2016. Automating ECU identification for vehicle security. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 632–635.

[13] Chris Karlof, Naveen Sastry, and David Wagner. 2004. TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 162–175.

[14] Patrick Killeen. 2020. *Knowledge-Based Predictive Maintenance for Fleet Management*. Master's thesis. Université d'Ottawa/University of Ottawa. https://ruor.uottawa.ca/handle/10393/40086.

[15] Patrick Killeen, Bo Ding, Iluju Kiringa, and Tet Yeap. 2019. Iot-based predictive maintenance for fleet management. *Procedia Computer Science*, 151, 607–613.

[16] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. 2017. DDoS in the IoT: Mirai and other botnets. *Computer*, 50, 7, 80–84.

[17] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. 2010. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*. IEEE, 447–462.

[18] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. 2019. All things considered: an analysis of IoT devices on home networks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 1169–1185.

[19] Trishank Karthik Kuppusamy, Lois Anne DeLong, and Justin Cappos. 2018. Uptane: Security and customizability of software updates for vehicles. *IEEE vehicular technology magazine*, 13, 1, 66–73.

[20] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. 2017. OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 57–66.

[21] Xiangxue Li, Yu Yu, Guannan Sun, and Kefei Chen. 2018. Connected vehicles' security from the perspective of the in-vehicle network. *IEEE Network*, 32, 3, 58–63.

[22] Edward Markey. 2019. Senators Markey and Blumenthal Reintroduce Legislation to Protect Cybersecurity on Aircrafts and in Cars. Retrieved 11/09/2020 from https://www.markey.senate.gov/news/press-releases/senators-markey-and-blumenthal-reintroduce-legislation-to-protect-cybersecurity-on-aircrafts-and-in-cars.

[23] John Markoff. 2011. Researchers show how a car's electronics can be taken over remotely. Retrieved 11/09/2020 from https://www.nytimes.com/2011/03/10/business/10hack.html?_r=0.

[24] Charlie Miller and Chris Valasek. 2013. Adventures in automotive networks and control units. *Def Con*, 21, 260–264.

[25] Charlie Miller and Chris Valasek. 2015. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*.

[26] Dietmar PF Möller and Roland E Haas. 2019. Automotive cybersecurity. In *Guide to Automotive Connectivity and Cybersecurity*. Springer, 265–377.
[27] Mert D Pesé, Troy Stacer, C Andrés Campos, Eric Newberry, Dongyao Chen, and Kang G Shin. 2019. LibreCAN: Automated CAN message translator. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2283–2300.
[28] Frank Stajano and Ross Anderson. 1999. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *International workshop on security protocols*. Springer, 172–182.
[29] William Stallings and Lawrie Brown. 2012. *Computer security: principles and practice*. Pearson Education Upper Saddle River, NJ, USA, 390–391.
[30] Synopsys. 2020. The Heartbleed Bug. Retrieved 12/03/2020 from https://heartbleed.com/.
[31] Technoton. 2020. Contactless CAN J1939 reader. Retrieved 11/11/2020 from https://www.jv-technoton.com/products/contactless-readers/cancrocodile/.
[32] Hannes Tschofenig and Emmanuel Baccelli. 2019. Cyberphysical security for the masses: A survey of the internet protocol suite for internet of things security. *IEEE Security & Privacy*, 17, 5, 47–57.
[33] Paul van Oorschot. 2020. *Computer Security and the Internet: Tools and Jewels*. Springer Nature.
[34] Vector. 2020. Media Oriented Systems Transport (MOST). Retrieved 11/26/2020 from https://www.vector.com/int/en/know-how/technologies/networks/most/#c21310.
[35] Wilfried Voss. 2010. *A Comprehensible Guide to Controller Area Network*. Copperhill Media.
[36] Wilfried Voss. 2010. *A Comprehensible Guide to J1939*. Copperhill Media.
[37] Richard P Walter and Eric P Walter. 2016. *Data acquisition from HD vehicles using J1939 CAN bus*. Society of Automotive Engineers.
[38] Wikipedia. 2020. Can bus. Retrieved 11/26/2020 from https://en.wikipedia.org/wiki/CAN_bus.
[39] Wikipedia. 2020. Endianness. Retrieved 12/04/2020 from https://en.wikipedia.org/wiki/Endianness.
[40] Wikipedia. 2020. Femtocell. Retrieved 11/30/2020 from https://en.wikipedia.org/wiki/Femtocell.
[41] Wikipedia. 2020. OSI model. Retrieved 12/03/2020 from https://en.wikipedia.org/wiki/OSI_model.
[42] Wufei Wu, Renfa Li, Guoqi Xie, Jiyao An, Yang Bai, Jia Zhou, and Keqin Li. 2019. A survey of intrusion detection for in-vehicle networks. *IEEE Transactions on Intelligent Transportation Systems*, 21, 3, 919–933.
[43] Clinton Young, Joseph Zambreno, Habeeb Olufowobi, and Gedare Bloom. 2019. Survey of automotive controller area network intrusion detection systems. *IEEE Design & Test*, 36, 6, 48–55.