A Stealth, Selective, Link-Layer Denial-of-Service Attack Against Automotive Networks

Andrea Palanca[1(✉)], Eric Evenchick[2(✉)], Federico Maggi[3(✉)], and Stefano Zanero[1(✉)]

[1] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy
andrea.palanca@mail.polimi.it, stefano.zanero@polimi.it
[2] Linklayer Labs, Toronto, Canada
eric@evenchick.com
[3] FTR, Trend Micro, Inc., Milan, Italy
federico_maggi@trendmicro.com

Abstract. Modern vehicles incorporate tens of electronic control units (ECUs), driven by as much as 100,000,000 lines of code. They are tightly interconnected via internal networks, mostly based on the CAN bus standard. Past research showed that, by obtaining physical access to the network or by remotely compromising a vulnerable ECU, an attacker could control even safety-critical inputs such as throttle, steering or brakes. In order to secure current CAN networks from cyberattacks, detection and prevention approaches based on the analysis of transmitted frames have been proposed, and are generally considered the most time- and cost-effective solution, to the point that companies have started promoting aftermarket products for existing vehicles.

In this paper, we present a selective denial-of-service attack against the CAN standard which does not involve the transmission of any complete frames for its execution, and thus would be undetectable via frame-level analysis. As the attack is based on CAN protocol weaknesses, all CAN bus implementations by all manufacturers are vulnerable. In order to precisely investigate the time, money and expertise needed, we implement an experimental proof-of-concept against a modern, unmodified vehicle and prove that the barrier to entry is extremely low. Finally, we present a discussion of our threat analysis, and propose possible countermeasures for detecting and preventing such an attack.

1 Introduction

The automobile, starting from the late seventies, has witnessed massive and radical changes over the years, due to the ever increasing addition of electronics and software. Almost every aspect of a car operation (e.g., steering, locks, windows, airbag deployment) is nowadays supervised by in-vehicle embedded systems, communicating among each other via an internal network typically

M. Polychronakis and M. Meier (Eds.): DIMVA 2017, LNCS 10327, pp. 185–206, 2017.
DOI: 10.1007/978-3-319-60876-1_9

based on the Controller Area Network (CAN) standard. The unavoidable consequence of this increased complexity and co-presence of electronic and computer-based components is a wider digital attack surface. The feasibility of such attacks has extensively been demonstrated by security researchers over the past decade [4,10,13,14,26], to the point that "car hacking" is now being taken into serious consideration by US government agencies [25], already acting toward strengthening automotive cybersecurity regulations [1].

Most of the attacks demonstrated thus far leverage one or more vulnerabilities with the aim of indiscriminately sending messages into the car's internal network and proving that it is possible to alter the behavior of safety-critical elements such as engine, brakes or steering. The frame-based nature of these attacks makes them effectively recognizable by proper intrusion detection or prevention systems (IDSs/IPSs), which monitor all messages circulating on the network and trigger countermeasures in case they detect that an attack is in progress. Previous work [6,12,13,20,21,26,27] has shown the feasibility of porting classic intrusion detection methodologies to the automotive domain, and car cybersecurity companies have already proposed aftermarket solutions for existing vehicles [2,17,23].

In this paper, we present a novel link-layer denial-of-service (DoS) attack that is inherently harder to detect via frame-level analysis mechanisms, because it does not require the transmission of *any* complete frame for its execution.

The attack is able to selectively cause malfunction or even a complete shutdown of any CAN node connected to the bus, including safety-critical components (e.g., electronic stability control, electric power steering). Since it exploits design weaknesses of the CAN protocol standard, any implementation and manufacturer is vulnerable, even beyond the automotive domain such as factory automation (e.g., CANopen- or DeviceNet-based machinery), building automation (e.g., elevator management), and hospitals (e.g., lights, beds, X-Ray machines).

The attack works locally, through the standard diagnostic port—which is mandatory in essentially every country [19]—or via a tampered/counterfeited/remotely-compromised replacement part. Therefore, the attacker model is rather generic, including for example a malicious mechanic, a malicious over-the-air (OTA) firmware upgrade, a malicious passenger or driver in a car sharing (or even self-driving car) setting, and similar scenarios.

In order to precisely evaluate the required time, level of expertise and cost, we concretely implemented a proof-of-concept of the attack against a modern, unaltered production vehicle (an Alfa Romeo Giulietta), and prove that it can be efficiently and conveniently mounted against a specific frame with 99.9974% accuracy using a development board as simple as an Arduino Uno.

In the end, we discuss examples of possible threats to car occupants, examine which are potential attack vectors and real-world scenarios where such attack could be staged by attackers, and propose possible remediation approaches.

In summary, our paper makes the following contributions:

– We describe a stealth, denial-of-service attack against the CAN standard, to which all CAN bus implementations are vulnerable.

– We demonstrate the attack feasibility by implementing a low-cost proof-of-concept against an unmodified vehicle and release full source code to the community.
– We propose practical solutions for detecting the attack in existing CAN networks and discuss possible network modifications for preventing it in future vehicles.

2 Background

2.1 Controller Area Network (CAN) Bus

The Controller Area Network (CAN) bus is a multi-master asynchronous soft real-time serial bus standard designed for the interconnection of multiple components called *nodes*. It was designed and first developed by Robert Bosch GmbH, released in 1986, and standardized in 1993 as ISO 11898.

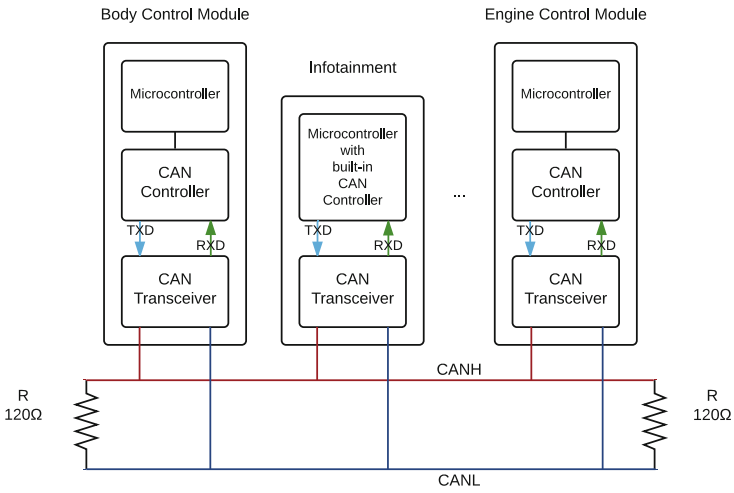


Fig. 1. Example architecture of a generic two-wire CAN network.

Physical Layer. ISO 11898 CAN buses are characterized by two wires, CANH (high) and CANL (low), terminated at each end by a 120 Ω resistor.
As shown in Fig. 1, each CAN node comprises three parts:

Microcontroller: is responsible for sending and processing complete CAN frames to and from the CAN controller and supervising the CAN controller operation.

CAN Controller: implements the CAN specification. It synchronizes with the CAN signal, sends and receives logical data to and from the CAN transceiver, automatically adds stuff bits, and performs error handling. Notably, in our attack we leverage such error handling mechanism. Therefore, we describe thoroughly in Sect. 3.

CAN Transceiver: serves as an interface between the CAN controller and the physical bus by translating logical signals coming from the CAN controller into bus electrical levels.

Stuff bits are added whenever a transmitter detects five consecutive identical bits to be sent. When this happens, the transmitter automatically inserts a subsequent complementary bit in the transmitter bit stream. This so-called *stuff rule* is necessary to keep the nodes synchronized. CAN buses have no clock sync signal, and use a Non Return to Zero encoding.

The CAN standard mandates that one of the two logical values shall be *dominant* over the other one. In case one "dominant" and one "recessive" bit are sent at the same time, the bus state—and thus the logical signal received by all CAN nodes—is "dominant." Most CAN bus implementations feature a wired-AND configuration, hence the dominant bit is the logical 0 whereas the recessive bit is the logical 1. The state read by the nodes is determined by the voltage measured between CANH and CANL lines; whenever it exceeds a certain threshold (usually 0.9 V), a dominant state is encoded (recessive otherwise).

Data Link Layer. The CAN standard describes four types of frames: data, remote, error, and overload frame.

The data frame is composed of Start of Frame, Arbitration Field, Control Field, Data Field, CRC Field, ACK Field, and terminates with the End of Frame. The Arbitration Field contains the Frame Identifier, which identifies the meaning of the message content, and determines the frame priority when two or more nodes are contending the bus. The Arbitration Field is either 11 or 29 bits long, depending on the specification (CAN 2.0A or 2.0B).

The error frame consists of an Error Flag and an Error Delimiter. The Error Flag is characterized by six consecutive identical bits (dominant or recessive, depending on the current CAN controller error state, as explained in Sect. 3), which violate the bit stuffing rule. A node sends an error frame whenever an error is detected. In particular, there exist 5 types of errors: bit error, stuff error, CRC error, form error, and acknowledgment error.

A message is valid for the transmitter if there is no error until the end of End of Frame. A message is valid for the receivers if there is no error until the last but one bit of End of Frame.

2.2 Applications of CAN Bus

The CAN bus standard has been designed specifically for the automotive domain, which is where it finds most applications.

Although other protocols have been proposed through years—e.g., Local Interconnect Network (LIN), Flexray—, CAN has been established as the *de-facto* standard by car manufacturers due to its general-purpose ability of carrying data for a great variety of applications [15] (which, for instance, LIN is not able to ensure due to its slow speed and master-slaves architecture [16]) while still preserving competitive prices (cost per node of a CAN bus network is approximately half the cost per node of a Flexray network [18]). In addition to that,

some countries, such as the USA, recently started mandating the exclusive use of CAN for diagnostics purposes for all light duty cars sold on the market [24]: this further encouraged the majority of car makers into adopting CAN bus for the implementation of the entire car's internal network.

There are typically two types of CAN data frames in current automotive systems:

Standard messages: exchanged between two or more ECUs for regular communications, in order to coordinate for the correct execution of an application. For example, the frames sent from the engine control module to the instrument panel to display engine status.

Diagnostic messages: exchanged between diagnostic devices connected to the car internal network (e.g., via the on-board diagnostics OBD-II port) and one or more ECUs for diagnostic sessions. For example, for emission testing or, in case of malfunctioning vehicle, for checking diagnostic trouble codes.

One of the major applications of CAN bus standard messages in modern vehicles is for active safety systems, which reactively (and even proactively) intervene and correct car inputs in real time to avoid or minimize the effects of an accident, or to enhance the driving experience. In the past, active safety systems were included as standard equipment in luxury vehicles only; however, given the (measured) effectiveness of such systems in terms of road casualties and injuries reduction, governments started mandating a minimum set of active safety systems on all cars sold on their national market. At the same time, national crash-test evaluation agencies began fostering their adoption by means of safety ratings boosts. As a result, the majority of modern cars are equipped with on-board active safety systems.

Nevertheless, the CAN standard is not restricted to the automotive domain only. Beginning in 2002, with the Ducati 999, motorcycle manufacturers started adopting CAN buses, mainly due to the weight savings provided by the reduced wire harness requirements. The CAN standard is also employed for train-wide communication networks (e.g., linking door units, brake controllers coordination, passenger-counting units), maritime (e.g., controlled-by-wire ships), avionics (e.g., flight-state sensors, navigation systems, or communications with research PCs in the cockpit), or aerospace (e.g., fuel systems, pumps, or linear actuators). The CAN standard is also used for regulating CANopen- or DeviceNet-based machinery networks in industries (e.g., packaging machines, knitting systems or for semiconductor manufacturing), for managing operating rooms equipment in hospitals (e.g., lights, beds, X-Ray or other diagnostic machines), or for controlling elevators in modern, automated buildings.

2.3 Known Attacks

The constant addition and coupling of embedded systems inside vehicles and the inclusion of more and more interfaces with the outside world immediately raised concerns about the impact of vulnerabilities.

Researchers of ESCRYPT were the first to theoretically investigate the possible risks to which vehicles would be exposed in case of attack and to propose possible countermeasures [29].

Notably, starting from 2010, researchers have also been investigating *practical* attacks on CAN networks, especially focusing on frame-injection attacks because of their potential ability to deeply alter the vehicle's behavior. The authors of [8,10] first showed how a local attacker capable of injecting frames into the vehicle's network could control the majority of its subsystems, including safety-critical devices like engine or brakes, and even bypassing the driver inputs.

In the following years, the famous "Jeep hack" by Charlie Miller and Chris Valasek [14], anew completed via frame-injection attacks, further contributed to raising security awareness among car manufacturers.

2.4 Proposed Countermeasures

A survey conducted in March 2016 by the US Government Accountability Office [25] among major industry stakeholders identified the following countermeasures that could be applied to mitigate the impact of digital attacks against current and future cars:

Trusted Computing Base: hardware security modules or trusted software in order to preserve and guarantee ECUs integrity.

Network Segmentation: safety-critical ECUs decoupling from non safety-critical ECUs, or from ECUs featuring external interfaces, by confining them in different networks and restricting inter-networks communications via firewalls/gateways.

Cryptography: by means of ECUs code signing or frames encryption and authentication.

Intrusion Detection or Prevention Systems (IDSs, IPSs): security appliances that monitor network traffic, try to establish if an attack is in progress and, in case of prevention systems, attempt to stop it automatically.

Among these, frame-analysis based detection or prevention systems are currently believed to be the most time- and cost-effective solution for circumventing security threats in CAN networks [6,12,13,20,21,26,27]. Indeed, frame-injection attacks are based either on the transmission of normal frames at a much higher rate than usual[1] or on the transmission of diagnostic frames that are not expected to be seen in standard circumstances. Hence, a proper detection system can recognize signs of such attacks. Moreover, the bus topology of CAN-based networks makes the deployment of IDSs or IPSs into current architectures effortless, to the point that companies have already developed aftermarket detection and prevention systems for current generation vehicles [2,17,23].

[1] The reason is that spoofed frames will be sent at the same time as legitimate frames. Thus, in order to trick the receiving ECU into considering only the maliciously crafted messages, these must be sent at a much faster rate with respect to the rightful ones.

2.5 Related Work

The idea of mounting denial-of-service attacks against CAN networks is not novel. In the aforementioned papers [8,10,14,26,29], several examples of DoS attacks via frames injection have been proposed (e.g., by sending frames that counteract driver inputs, or frames with the highest priority so as to indefinitely delay other nodes' transmissions). However, these kinds of attacks are effectively detected and stopped with frame-analysis based IDS/IPS approaches. Indeed, the essential aspect of such attacks is the injection of either unexpected frames or the transmission of frames at abnormal rates.

A more subtle denial-of-service attack exploiting CAN error handling and fault confinement protocols was published in July 2016 [5]. However, the attack is restricted to periodic messages only as it requires precise predictability of transmission instants of target frames, and it still involves the communication of a few complete messages for its execution. The attack presented in this paper, instead, is not affected by any restrictions and does not require any full message-sending capability at all.

These types of frame-less attacks were theorized in the past. For instance, in [29] the authors explore the feasibility of performing frame-less DoS attacks by sending well-directed error flags into the CAN network, forcing other nodes to reject a message. In [9] the authors briefly mention that similar situations could occur if a corrupted node started to upset CAN traffic bits. In [28] many bus networks (including CAN) are described as being vulnerable to "bit banging" attacks. However, all previous work described such attacks from a purely theoretical standpoint, without any proof-of-concept implementation nor in-depth threat-model analysis.

To the best of our knowledge, the only prior work which proposed an implementation of a mechanism capable of inserting faults in CAN networks is [7]. Yet, the research focused on injecting errors in CAN networks for pre-production testing purposes only, without covering any security considerations. Moreover, in order to perform such faults injection, the network had to be topologically altered to a non-ordinary star schema, tampering which a potential attacker is not expected to perform in a reasonable amount of time.

3 Protocol Analysis and Attack Description

In this section, we describe the two weaknesses that have been exploited by the proposed denial-of-service attack. The main one lies in how the CAN standard handles errors. A second weakness further exacerbates the impact of the first one, making its DoS capabilities more relevant. Finally, we present our attack, along with a description of its technical requirements.

3.1 CAN Error Handling Weakness

As mentioned in Sect. 2, there are five possible error types. For our attack, the relevant one is the bit error type.

By design, each node must monitor the bus signal every time it sends a frame. A bit error occurs (and must be detected within the sampling frequency) whenever a transmitting node notices that the logical value of the bus is different from the bit value that it is trying to send[2]. Should a node observe such condition, it must interrupt the frame transmission and send immediately an error frame, which breaks the stuff rule (or generates other errors) and causes all other nodes to reject the frame received up to this point, effectively denying the broadcast of that frame.

Therefore, considering that a dominant bit always overwrites a recessive bit, the transmission of just *one* single dominant bit, by *any* node, when a recessive bit is being transmitted, is enough to trigger a bit error, causing the other nodes to discard the current frame.

3.2 CAN Fault Confinement Weakness

The impact of bit errors is not limited to frame-wise DoS due to a second security weakness of the CAN standard induced by the automatic fault confinement protocol.

In order to automatically overcome node faults and avoid situations such as a malfunctioning node causing a complete bus failure, each CAN node can be in three distinct error states (Fig. 2), depending on how many errors a certain node has generated or observed:

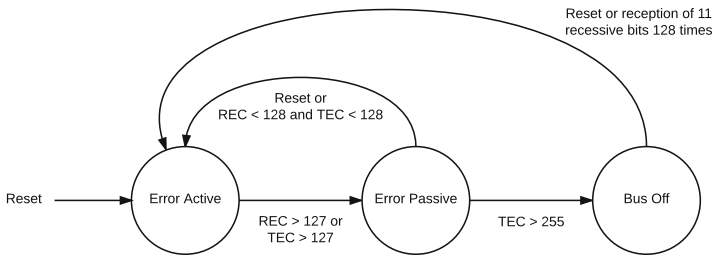


Fig. 2. CAN fault confinement finite state machine.

Error Active: the CAN node normally takes part in bus communications and sends an active error flag (six dominant bits) when it detects an error.

Error Passive: the CAN node can normally take part in bus communications, but can only send a passive error flag (six recessive bits) when it detects an error and must wait an additional 8-bit time before starting a new transmission.

Bus Off: the CAN node cannot take part in any bus communication, not even reading frames off the bus.

[2] With the exception of the Arbitration Field and the ACK Field, in which a bus value different than the transmitted one is an expected condition in regular CAN protocol operations.

Transitions between these three error states are determined by the values of two counters, the Transmit Error Count (TEC) and the Receive Error Count (REC). According to the protocol, whenever a transmitting node sends an error flag, its TEC must be increased by 8. This means that, after 16 invalid transmissions, an Error Active node with a zeroed TEC will go in Error Passive state ($TEC = 16 \times 8 = 128$), and after another 16 invalid transmissions it will enter the Bus Off state ($TEC = 256$), denying all bus communication until a bus idle condition or a reset command (or both) are observed. Unfortunately, forcing an idle condition is practically impossible, because it would mean disabling or disconnecting the majority of devices attached to the bus. Similarly, forcing a reset command, which can be done by the node's microcontroller, is problematic, because the Bus Off node could be a legitimate faulty node.

By means of the previous weakness, the practical consequence is that 32 straight bit overwrites on a frame sent by a node are sufficient for making that node unable to either send or receive any message on and off the bus, effectively denying the service that such node is implementing.

3.3 Technical Requirements

The attack is based on a deliberate violation of the CAN protocol, which mandates that all nodes that have lost arbitration shall in no way further interfere with CAN traffic.

The adversary must be able to directly read the RXD signal coming from the transceiver (which transports the current logical CAN bus value) and manipulate the TXD signal entering into the transceiver (which transports the logical value the CAN bus will be driven to), as depicted in Fig. 3. This requires the microcontroller to be directly attached to the transceiver, a common architecture among ECU manufacturers [11], such as in the case of the Renesas V850ES/FJ3

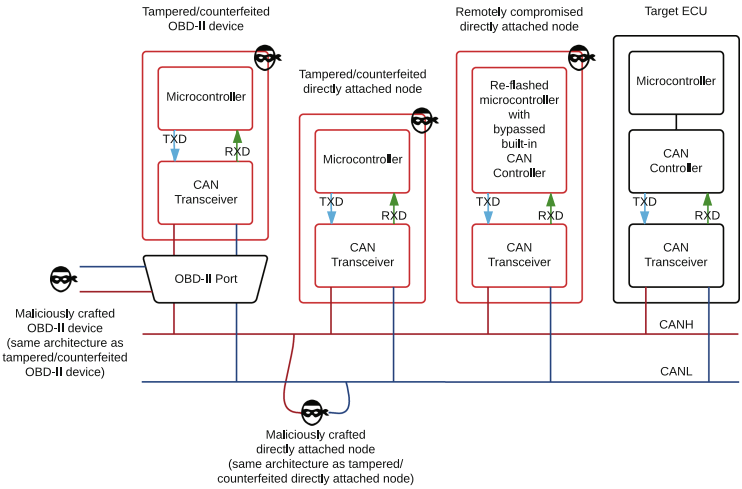


Fig. 3. Examples of attacking nodes architectures.

of the Jeep Cherokee's Uconnect [14], due to the induced cost-effectiveness and space-saving reasons.

The microcontroller is simply required to support pin-edge-change external interrupts and a timer that can be set to trigger at custom values (i.e., to match the CAN bus bit rate), both largely diffused features in modern microcontrollers (the aforementioned V850ES/FJ3 supports both). Obviously, the microcontroller must be fast enough to account for interrupt latency, pin read-write latency and compilation-induced overhead, a requirement which is nowadays not restricting thanks to the availability of low-price multi-core high-frequency microcontrollers (e.g., Parallax Propeller).

3.4 Proposed Attack Algorithm

The attack, which runs in a microcontroller attached to the CAN bus, consists of a setup phase (Algorithm 1), whose goal is to prepare the microcontroller for the attack execution, and an interrupt service routine (ISR) (Algorithm 2), which will monitor the bus and, when necessary, will execute the actual attack payload.

The setup algorithm is executed only once, when the microcontroller boots. The procedure consists in setting the TXD signal to recessive and initializing a buffer of size B—which, during the attack, will always contain the last B bits read from the CAN bus—with a series of 1s (as the ISR will start its execution after the first RXD falling edge for synchronization purposes, thus after a series of 1s have been transmitted on the bus). The size B of the buffer depends on the implemented CAN specification (11 bit or 29 bit): For instance, if the attacker wants to disable a node which is sending frames with a 29 bit ID, a buffer of at least $B = 29$ bits is needed. Then, the algorithm sets the timer expiration value—which regulates the rate at which the attack ISR will be executed—to match the target CAN bus bit rate. Finally, it waits for the first RXD falling edge and, when perceived, activates the Attack ISR.

Algorithm 1. Setup procedure

```
1: procedure SETUP
2:     TXD ← Recessive
3:     Buffer ← 111...1
4:     Set timer to expire every CAN bit time seconds
5:     Wait until RXD falling edge
6:     Activate Attack ISR
7: end procedure
```

The attack ISR is executed periodically, at the same rate of the CAN bus signal. The algorithm first checks if the frame currently being transmitted on the bus has the target ID. If the frame is a target frame, the algorithm overwrites the first recessive bit with a dominant bit. Else, if the frame is not a target frame, the algorithm updates the buffer by sampling the bus signal and appending the sampled bus value to the buffer (Fig. 4).

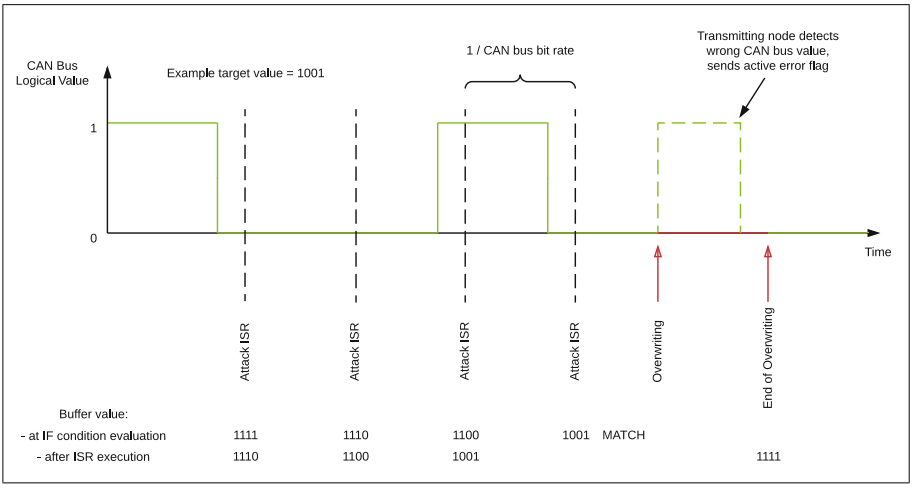


Fig. 4. Visualization of the proposed attack algorithm on a time graph.

Algorithm 2. Attack ISR

```
1: procedure ATTACK
2:     if Current frame in Buffer is a target frame then
3:         Wait until first recessive bit
4:         TXD ← Dominant
5:         Wait CAN bit time seconds
6:         TXD ← Recessive
7:     else
8:         Append RXD value to Buffer
9:     end if
10: end procedure
```

4 Experimental Proof-of-Concept Implementation and Testing

In this section we describe how an adversary can implement our attack. Our goal is twofold. First, we want to assess the technical feasibility of our attack and quantify its performance on a modern automobile. Secondly, we want to show how low the barrier to mount the attack is nowadays, given the ample availability of rapid-prototyping frameworks (e.g., Arduino).

A full demonstration video of the attack in action is available at https://www.youtube.com/watch?v=PmcqCbRMCCk and the source code running on the attacking device at https://github.com/stealthdos/CAN-Denial-of-Service.

4.1 Target Automobile

The automobile at our disposal for the test was a 2012 Alfa Romeo Giulietta 2.0 JTDm-2. The car features two CAN buses: a high-speed CAN (class C,

according to SAE networks classification) working with 29 bit IDs (500 kbps), and a medium-speed CAN (class B) working with 29 bit IDs (50 kbps). Both lines are reachable via the OBD-II port (Fig. 5).

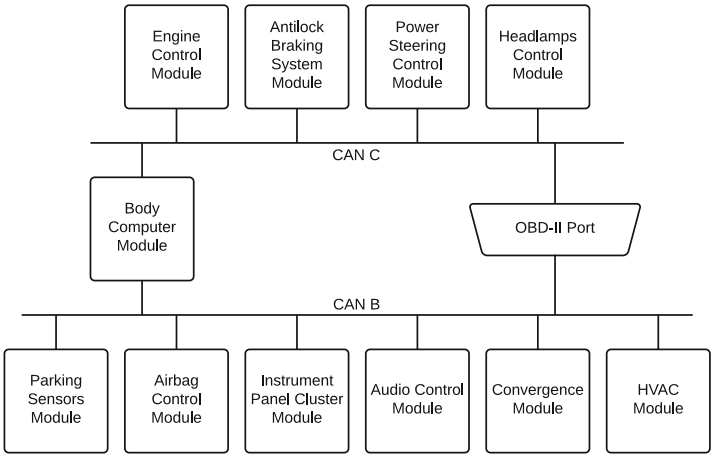


Fig. 5. Architecture of the 2012 Alfa Romeo Giulietta 2.0 JTDm-2 internal CAN networks.

For ethical reasons, we performed the proof-of-concept attack against a CAN B bus node, namely the parking sensors module. This choice is guided by the fact that CAN B buses typically do not connect safety-critical nodes: This should reduce the chances that, by simply reading this paper and taking our open-source prototype, a malicious attacker or a "script kiddie" could directly reuse our attack on safety-critical subsystems connected to CAN C buses. This does not imply any loss of generalization: bit rate apart, CAN Bs and CAN Cs operate identically.

4.2 CAN Traffic Analysis

In order to capture CAN traffic, we purchased for $30 a Scantool OBDLink SX USB-to-OBDII cable. The device features an embedded STN1130 microcontroller that, besides emulating the very common ELM327 1.3a AT instruction set, allows to capture even partial or erroneous CAN frames thanks to the additional ST commands. Moreover, it performs frame decoding automatically, writing ID-Data Length Code (DLC)-Data Field directly on the serial port.

We plugged the device into a laptop, then into the OBD-II connector, and we started listening on the serial port to capture all CAN messages sent on the bus for a fixed amount of time (until we noticed no different frames) in various conditions (e.g., with neutral gear, with reverse gear, with reverse gear and near an obstacle).

We filtered the captured CAN traffic and inspected it manually to remove uninteresting CAN frames. Eventually, we isolated the frame responsible for notifying the obstacle position, sent by the parking sensors module. Some examples of that frame follow:

```
CAN ID: 0x06314018;
Data Length Code: 8 bytes;
Data Field:
- Ignition off: frame not sent;
- On, N: C000000F0F000000;
- On, R, no obstacle: 0000000F0F000000;
- On, R, central obst: 0300000X0XY00000,
  X: chime sound frequency,
  Y: distance on driver's LCD.
```

An attacker who wants to target another device would have to either perform the same procedure offline, on another instance of the same car, or know the CAN ID in advance. Overall the whole procedure could require from minutes (e.g., for capturing the frame issued by a dashboard button) to hours or days (e.g., when trying to thoroughly reverse engineer a complex active safety protocol). The corner case is when the target CAN bus is not directly reachable via the OBD-II port. In this case, however, the attacker could simply reach the bus line by other means, as thoroughly discussed in Sect. 5.

4.3 Attacking Device Implementation

We implemented our attack as a hand-crafted OBD-II dongle, which can be physically plugged into the car's OBD-II port. Its architecture is reported in Fig. 6. We opted for an Arduino Uno Rev 3 and a Microchip MCP2551 E/P, the cheapest (total expense was around $25) and most common microcontroller and CAN transceiver available on the market which are capable of fulfilling the aforementioned minimum requirements.

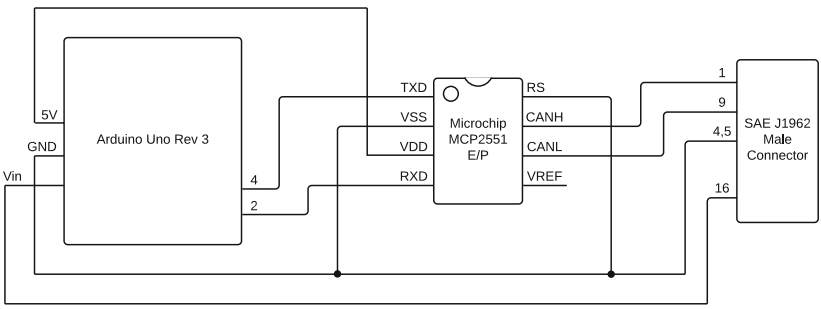


Fig. 6. Schematic of the crafted attacking device.

In order to execute the attack payload against the target vehicle, all it is necessary to do is plug the device into the car's OBD-II port. The operation requires less than 30 s, the device will be instantly powered by the 12 V battery and will immediately start the algorithm.

4.4 On Bench Testing

To adequately test our attacking device implementation and investigate its reliability, we implemented a bench test CAN bus.

On-bench Attack Test. In addition to the OBDLink SX and the attacking device, we used a breadboard and two 120 Ω resistors for creating a CAN network, a 12 V rechargeable battery for simulating the car battery, and a Linklayer Labs CANtact 1.0, an open-source, Python-scriptable, low-cost USB-to-CAN adapter. This setup ensured that we had at least two nodes, as required by the CAN specifications for correct operation.

First, we tested whether, without the attacking device, both nodes were able to correctly exchange messages with each other. Then, we connected the attacking device to the CAN bus and tried to send (target) CAN frames; first, from the OBDLink SX, then from the CANtact. In both cases the attacking device managed to correctly "kill" the target frames: The receiving nodes were not able to retrieve the message. Moreover, we confirmed that the CAN fault confinement weakness (Sect. 3.2) caused the CANtact node to enter the Bus Off state, after exactly 32 erroneous frames. Note that, being a mere testing tool, the OBDLink SX does not implement the CAN automatic fault confinement protocol, but retries to send an erroneous message for 160 times before halting the transmission.

Reliability Measurement. In order to investigate the reliability of our attacking device in a realistic scenario—comprising both target and non-target CAN frames—we developed a Python CAN-fuzzing script. The script automatically generates random, yet valid, CAN frames, sends them through the OBDLink SX, waits to receive them from the CANtact, and compares the received frames with the original ones. We left the script running for 24 h and report the results in Table 1.

Despite a negligible fraction of false negatives—caused by distortions or spikes in the signal due to imperfect connections or hardware noise, and by Arduino interrupts timing drifts—, we measured a 99.9974% accuracy, which makes our basic and remarkably low-cost device already suitable for effectively performing the attack in a real-world situation.

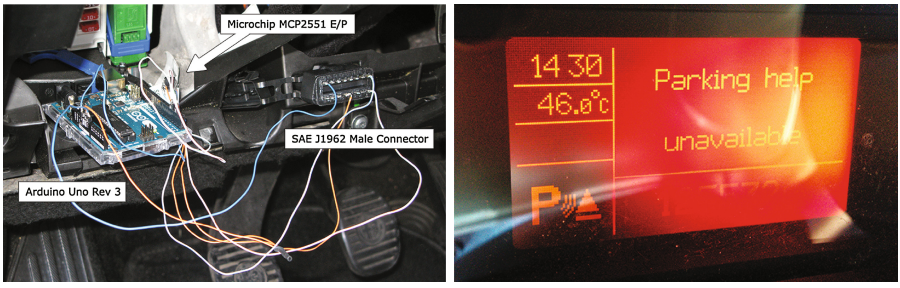
4.5 On Vehicle Testing

Finally, we tested our attack on our testing vehicle, an unmodified 2012 Alfa Romeo Giulietta.

Table 1. CAN fuzzer/checker test statistics.

Description	Value
Test duration	24 h
Total number of frames sent	9,403,842 frames
Average throughput	108.84 frames/s
Average frame length	101 bits
Average CAN utilization	0.21985834
Number of correctly processed frames	9,403,598 frames
Number of false positives	0 frames
Number of false negatives	244 frames
Accuracy	**0.99997405**

After plugging the attacking device to the OBD-II port, the parking sensors immediately stopped working altogether: Neither visual information nor warning proximity chime could be heard, even in the presence of a very close obstacle, and the dashboard display notified the driver about the malfunctioning subsystem (Fig. 7a and b).



(a) The attacking device attached to the Giulietta's OBD-II port.

(b) The parking sensors malfunction reported on the driver's LCD.

Fig. 7. Our attack in action on a real-world modern vehicle.

The subsequent fabrication of an ad-hoc forked cable, which allowed to connect both the attacking node and the OBDLink SX to the OBD-II port at the same time, and the following CAN traffic capture also revealed that the parking sensors module reached the Bus Off state—only 32 transmission attempts were recorded—, without any doubt confirming the complete denial-of-service accomplishment.

5 Threat Model Discussion and Remediation Approach

5.1 Threat Assessment

In this section, we discuss the practical impact of our work in terms of threats arising if an adversary decides to mount this type of attack against a vehicle in the real world, even in the presence of frame-analysis based detection or prevention systems. In all cases, the envisioned attacker is only required to execute our denial-of-service attack. Therefore, differently than previous work, we do not require that the attacker has frame-injection capabilities.

Active Safety Systems Attacks. One of the main purposes of CAN standard messages is to support active safety systems communications. Despite their undeniable usefulness, active safety systems may induce double-edged sword situations while driving, especially when the drivers have grown accustomed to their presence, and begin to blindly rely on them. As a result, an abrupt malfunction in such systems may cause unpredictable and potentially dramatic consequences. Given this premise, a potential threat is that an adversary could start using our attack to induce specific "faults" in the CAN frames generated by active safety systems. For instance, mounting our attack on traction control systems may lead to perilous vehicles loss of control; on autonomous cruise control systems may lead to vehicles not autonomously stopping as expected by drivers, a failure which, last year, caused even fatal accidents [22].

Car Ransom. Although CAN is not suitable for supporting steer-by-wire or brake-by-wire functionalities, CAN has been employed in the past to carry throttle-by-wire messages. For instance, as described in [26], the 2010 Toyota Prius internal combustion engine throttle actuator is controlled by CAN frames sent from the power management control ECU to the engine control module. An adversary may use our attack against such frames, causing inability for the driver to control throttle position and thus to move the vehicle. Though this would not necessarily generate hazardous conditions, a financially motivated attacker, after exploiting a vulnerability in an externally reachable module (e.g., the infotainment system), could leverage the DoS to mount a ransomware-like attack and later show the classic message on the infotainment display, in an utter similar fashion to desktop computers ransomware. An analogous condition might also be caused by blocking the frames sent by the key-less access control unit at car startup to all other modules, preventing anti-theft systems from being disengaged and hence the car from being started.

Theft Support. Both the aforementioned attack scenarios, despite perfectly feasible, would require a previous substantial reverse-engineering effort. In this section we discuss a third option, focusing on resource-bound attackers. We also assume that the attacker has a very narrow time window to gain physical access to the target vehicle.

Most modern premium cars' door locks are controlled by CAN B-connected ECUs (for instance, the 2014 Jeep Cherokee [14]), which is typically accessible via the OBD-II port. Isolating the frames responsible for locking the doors is much

simpler and faster than reverse-engineering active safety equipment messages. Indeed, the aforementioned door locks are under complete control of the user: A single press of the lock-unlock button on the driver's door corresponds exactly to one fixed set of frames issued to the door modules actuators. Therefore, in a matter of minutes, an adversary may isolate the frames responsible for doors locking, modify a software parameter in the attacking device in order to DoS locking frames, and then leave the device plugged into the car's OBD-II port, preventing car doors from being locked again after being unlocked. The attacking device architecture can be as simple as our experimental proof-of-concept or may include other components for additional functionality: For instance, GPS or GSM shields in order to track the vehicle position or command the attack payload execution remotely. The result of this attack is the ability, for the attacker, to gain cost-effective aposteriori access to the car, allowing her to subsequently steal any valuable goods or replacement part inside the vehicle.

5.2 Threat Vectors Analysis

After discussing the possible threats that a potential attacker may pose by selectively stopping specific CAN frames, we hereby examine the attack vectors that can be leveraged.

The easiest way by which the attack can be mounted is via a crafted device attached to the OBD-II port. As a matter of fact, in most vehicles the OBD-II port serves as a direct interface into all car internal buses, provides 12 V direct current output for powering connected devices, and is conveniently located underneath the steering wheel. Therefore, in a matter of seconds, an adversary with physical car access is able to install a working attacking device inside a car. Real-world scenarios in which this may happen are numerous, and include for instance valet parking, car sharing, car renting, car lending or self-driving car settings.

A similar situation may arise if the car owner decides to use a rogue (trojanized/counterfeited) aftermarket OBD-II device. The reasons for doing so could be various. For instance, the owner may opt for a low-cost "compatible" replacement part, be willing to obtain discounted fees from insurance companies by installing so-called "black boxes", could be interested in performing do-it-yourself car diagnostics, or simply for enriching car infotainment functionality.

Nonetheless, physical-access attacks are by no means limited to the diagnostic port. An adversary may or must opt for attaching and hiding the attacking device anywhere along the car internal network (see the attacker depicted under the two wires in Fig. 3). This, for instance, may happen in a malicious-mechanic scenario, while the car is undergoing tests or repair. The same holds for the installation of rogue replacement parts that require CAN bus connections for their operation, like aftermarket infotainment units, parking sensors modules, or anti-theft systems.

In addition, the denial-of-service attack may also be staged without requiring any physical interaction with the target vehicle at all. In this case, however, there must be an on-board CAN node with a fast-enough microcon-

troller that supports external and timer interrupts, and obviously there must be a vulnerability that allows an attacker to remotely re-flash the microcontroller firmware. Although this setting is certainly restrictive, in [14] the authors proved that, by leveraging a chain of vulnerabilities in the Harman Kardon Uconnect system of a 2014 Jeep Cherokee, it is possible to remotely re-flash the embedded Renesas V850ES/FJ3 microcontroller—responsible for the Uconnect CAN communications—with an ad-hoc firmware. Like many automotive-specific microcontrollers, the V850ES/FJ3 embeds an on-chip CAN controller—therefore, it is directly connected with CAN transceivers via re-programmable GPIO pins—and supports both edge-triggered and timer interrupts. As a consequence, the very same exploitation chain that led to the Cherokee remote compromise via CAN frames injection could be used for mounting our DoS attack as well.

In all depicted scenarios, the envisioned attacker is able to obtain persistent presence on the CAN network, even surviving through vehicle's power-cycles. In order to eliminate the threat, the vehicle's owner is required to physically disconnect the device from the vehicle's network in case of surreptitiously added or trojanized replacement part, or to reflash the compromised ECU with factory firmware in case of a remote reprogramming.

5.3 Detectability and Countermeasures

Last, we briefly compare our attack with detection approaches recommended so far in literature for identifying security incidents in current CAN networks, and propose possible detection and prevention solutions for recognizing and impeding the execution of the DoS, in the hope that car manufacturers would take them into account during the design of future vehicles' internal networks.

Comparison with Current Detection Mechanisms. CAN bus intrusion detection systems proposed up to now in literature are essentially based on the anomaly detection of a measure concerning well-formed frames, because, in the majority of attacks, in order to trigger actions on cyber-physical systems, a transmission of frames is required. The evaluated measures are the frequency of messages of a specific ID—such as in [13,21,26,27]—, the time differences of messages with a specific ID [20], the specification of the behavior of messages with a specific ID [12], or the clock skews of periodic frames, again given a fixed ID [6]. This attack, instead, is not based on the transmission of new frames, but on the transmission of bits concurrently to the transmission of a legitimate frame. If this is done as described in Sect. 3.4, there would not be anomalous activity such as, for instance, unexpected message transmissions or even something as subtle as spikes in the CAN signal: From the receiving devices' point of view, there would simply be a frame transmission interrupted by an error. In brief, a frame-analysis based IDS could only notice the effect of our attack, i.e. the lack of frames sent from a particular device, not the attack itself. At this point, it could effectively signal the anomaly to the driver, however the attack has

already successfully taken place at least one time and the target device has already reached Bus Off state.

Attack Detection. The most problematic challenge to address is the detection of a forthcoming attack before the denial-of-service has been executed, as the attacking node will not participate in any way with the CAN activity but will remain completely silent to all other nodes.

To mitigate the surreptitious addition of rogue devices into the car's internal network, we propose a novel solution based on simple electronics principles. All CAN nodes are characterized by a differential internal resistance (Rdiff), within a standard interval according to the CAN specifications. Such Rdiff influences the total bus load that a transmitting unit must drive in order to correctly send a dominant or a recessive bit on the bus. Any additional node attached to the bus would change the load, and thus would change the necessary current flow for driving a dominant bit on the bus. Therefore, a detection mechanism could find out when a (new) node is connected by measuring the amount of current necessary for a dominant condition at each vehicle startup and comparing this value with the previously registered ones.

This mechanism, unfortunately, can by no means protect from remotely compromised nodes. However, a remote vector for our attack would require prior re-flashing of a node's microcontroller, thus altering its functionality. This opens the possibility for detecting signs of such alterations (e.g., via code-integrity checks) before the actual DoS attack takes place.

While the attack is in progress, a possible way to distinguish a deliberate DoS from an occasional node fault stands in the determinism by which errors are manifested. The malicious node that executes our attack will *always* send a dominant bit at a certain position of a specific frame, resulting in that frames regularly triggering bit errors in the same way. This is very unlikely to happen in the case of a fault. A detection approach could be to account for errors statistics for all frames and identify suspiciously correlated error scenarios.

Attack Prevention. Since our attack relies both on link-layer and physical-layer weaknesses proper of the CAN protocol, caused by a lack of consideration over security requirements at design time, preventing the DoS without a major nodes and network architecture revision is *hardly* feasible. Nevertheless, there exist a number of solutions that could be considered during the design of next-generation vehicles.

Network Segmentation. The main precondition of our attack is that the attacking node must be able to physically sense the target frame during its transmission. Should the target and attacking nodes be attached to *separate* CAN networks, the DoS would not be possible. Therefore, network segmentation by means of trusted mediators (e.g., CAN firewalls) is a viable solution. This approach would not prevent an attacker from physically connecting the attacking device directly on target CAN bus, but, at least, would very likely contain damages by possible counterfeited or remotely compromised nodes.

Network Topology Alteration. A more radical segmentation approach consists in changing the network topology from the current bus topology to a star topology, with a trusted network dispatcher in the middle, as proposed in a few prior studies [3,9]. Unfortunately, this solution would dramatically increase the wiring harness and limit the flexibility of the network, which was one of the core reasons which favored CAN bus adoption in the past.

Diagnostic Port Access Control. Another countermeasure consists in securing the access to the OBD-II port, which is the easiest attack vector. Apart from physical access prevention (e.g., via a separate hardware key), another approach is to rely on an authentication gateway between the OBD-II port and the other networks, designed to exclusively allow transmission of OBD-II PIDs data queries to unauthenticated users, and full CAN access to authenticated personnel only. This could deter both our attack as well as previous attacks based on frames injection, without breaking the OBD-II diagnostics capability.

Encryption. Another option is to implement encryption to the ID and Data Field of CAN frames (e.g., via stream ciphers or block ciphers in stream mode). The attacking node would not be able to distinguish target frames from unrelated ones and, thus, would be unable to selectively attack certain ones. This approach would not prevent the attacking node from brute-forcing the ID space to inject faults in the whole CAN traffic. Nevertheless, this would make the attacking node noisy and thus easier to detect.

Other Protocols. The ultimate solution for preventing this kind of attack in automotive networks would be to use non-vulnerable protocols. For instance, albeit not immune to other security issues [29], Flexray is not susceptible to our attack because both logical 0s and 1s are represented by dominant conditions on the bus.

6 Disclosure

An official disclosure to the Computer Emergency Response Team (CERT) of the attack and its impact has been performed, in the hope to reach the greatest number of members of the CAN bus community.

7 Conclusions

In this paper, we have presented and analyzed a novel design-level DoS attack against CAN buses. The attack does not require the transmission of any complete data frame. All it demands is the transmission of 1 bit, resulting in being potentially capable of deceiving all frame-analysis based detection and protection approaches, which are currently believed to be the most time- and cost-effective solution for securing CAN networks from digital attacks.

As the leveraged weaknesses lie in the CAN design, and are by no means implementation or manufacturer specific, all instances of CAN bus networks

(including, but not limited to, land vehicles, maritime, avionic, medical or industrial applications) are vulnerable to this attack.

In our research we have focused on the impact on the automotive area. We implemented (and released to the public) an experimental proof-of-concept on a modern, unaltered vehicle, proving that the barriers for mounting the attack are slim. Then, we have described the possible threats against car owners and passengers descending from the discovery of our attack. Last, we have discussed the potential attack vectors, and proposed possible short- and long-term mitigation approaches.

The ultimate hope of the research is to instill awareness over the security risks that an aggressive and unrestricted interconnection approach of nodes—now equipped with external interfaces—via a fragilely security-wise designed network protocol such as CAN bus could pose, and propose practical solutions in order to ensure the security that a safety-critical components' backbone network is not expected to lack.

References

1. All bill information (except text) for s.1806 - SPY car act of 2015. https://www.congress.gov/bill/114th-congress/senate-bill/1806/all-info
2. Argussec: ARGUSidps. https://argus-sec.com/solutions/
3. Barranco, M., Rodriguez-Navas, G., Proenza, J., Almeida, L.: CANcentrate: an active star topology for CAN networks, pp. 219–228, September 2004. http://iestcfa.org/bestpaper/wfcs04/WFCS04_Barranco.pdf
4. Checkoway, S., et al.: Comprehensive experimental analyses of automotive attack surfaces. http://www.autosec.org/pubs/cars-usenixsec2011.pdf
5. Cho, K.T., Shin, K.G.: Error handling of in-vehicle networks makes them vulnerable. In: CCS 2016, USA, pp. 1044–1055 (2016). http://doi.acm.org/10.1145/2976749.2978302
6. Cho, K.T., Shin, K.G.: Fingerprinting electronic control units for vehicle intrusion detection, pp. 911–927. USENIX Association, Austin. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cho
7. Gessner, D., Barranco, M., Ballesteros, A., Proenza, J.: Designing sfiCAN: a star-based physical fault injector for CAN (2011). https://www.researchgate.net/profile/Julian_Proenza/publication/232259902_Designing_sfiCAN_A_star-based_physical_fault_injector_for_CAN/links/00b7d532161b659ee8000000.pdf
8. Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive CAN networks – practical examples and selected short-term countermeasures. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 235–248. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87698-4_21
9. Kammerer, R., Frömel, B., Wasicek, A.: Enhancing security in CAN systems using a star coupling router, pp. 237–246, June 2012. http://www.vmars.tuwien.ac.at/documents/extern/3116/canrouter_security.pdf
10. Koscher, K., et al.: Experimental security analysis of a modern automobile, May 2010. http://www.autosec.org/pubs/cars-oakland2010.pdf
11. Kvaser: Microcontrollers with CAN (2016). https://www.kvaser.com/about-can/can-education/can-controllers-transceivers/microcontrollers-with-can/

12. Larson, U.E., Nilsson, D.K., Jonsson, E.: An approach to specification-based attack detection for in-vehicle networks, June 2008. http://ieeexplore.ieee.org/document/4621263/
13. Miller, C., Valasek, C.: A survey of remote automotive attack surfaces, August 2014. http://illmatics.com/remote%20attack%20surfaces.pdf
14. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle, August 2015. http://illmatics.com/Remote%20Car%20Hacking.pdf
15. National Instruments: Controller area network (CAN) overview. http://www.ni.com/white-paper/2732/en/
16. National Instruments: Introduction to the local interconnect network (LIN) bus. http://www.ni.com/white-paper/9733/en/
17. NNG: Arilou cyber security. https://www.nng.com/arilou-cyber-security/
18. ON Semiconductor: Basics of in-vehicle networking (IVN) protocols. http://www.onsemi.com/pub/Collateral/TND6015-D.PDF
19. SAE International: Global OBD legislation update (worldwide requirements). http://www.sae.org/events/training/symposia/obd/presentations/2009/d1daveferris.pdf
20. Song, H.M., Kim, H.R., Kim, H.K.: Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network, January 2016. http://ieeexplore.ieee.org/document/7427089/
21. Taylor, A., Japkowicz, N., Leblanc, S.: Frequency-based anomaly detection for the automotive CAN bus, December 2015. http://ieeexplore.ieee.org/document/7420322/
22. The Verge: Tesla driver killed in crash with autopilot active, NHTSA investigating, June 2016. http://www.theverge.com/2016/6/30/12072408/tesla-autopilot-car-crash-death-autonomous-model-s
23. Towersec: ECUShield. http://tower-sec.com/ecushield/
24. United States Environmental Protection Agency: Control of air pollution from new motor vehicles and new motor vehicle engines. http://www.epa.gov/fedrgstr/EPA-AIR/2005/December/Day-20/a23669.htm
25. United States Government Accountability Office: VEHICLE CYBERSECURITY - DOT and industry have efforts under way, but DOT needs to define its role in responding to a realworld attack, March 2016. http://www.gao.gov/assets/680/676064.pdf
26. Valasek, C., Miller, C.: Adventures in automotive networks and control units, August 2013. http://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf
27. Valdes, A., Cheung, S.: Communication pattern anomaly detection in process control systems, May 2009. http://ieeexplore.ieee.org/document/5168010/
28. Waibel, A.: The art of bit-banging: Gaining full control of (nearly) any bus protocol, June 2016. https://www.youtube.com/watch?v=sMmc0hSi5rs
29. Wolf, M., Weimerskirch, A., Paar, C.: Security in automotive bus systems (2004). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.728&rep=rep.1&type=pdf