Vehicle Safe-Mode
Limp-Mode in the Service of Cyber Security

Tsvika Dagan*, Mirco Marchetti†, Dario Stabili†, Michele Colajanni†, Avishai Wool‡
*‡ Tel-Aviv University, Israel
† University of Modena and Reggio Emilia, Italy

Emails: *TDagan02@gmail.com, †{mirco.marchetti, dario.stabili, michele.colajanni}@unimore.it, ‡yash@eng.tau.ac.il

***Abstract*—This paper describes a concept for vehicle *safe-mode*, that may help reduce the potential damage of an identified cyber-attack. Unlike other defense mechanisms, that try to block the attack or simply notify of its existence, our mechanism responds to the detected breach, by limiting the vehicle's functionality to relatively safe operations, and optionally activating additional security counter-measures. This is done by adopting the already existing mechanism of *Limp-mode*, that was originally designed to limit the potential damage of either a mechanical or an electrical malfunction and let the vehicle "limp back home" in relative safety. We further introduce two modes of *safe-mode* operation: In *Transparent-mode*, when a cyber-attack is detected the vehicle enters its pre-configured *Limp-mode*; In *Extended-mode* we suggest to use custom messages that offer additional flexibility to both the reaction and the recovery plans. While *Extended-mode* requires modifications to the participating ECUs, *Transparent-mode* may be applicable to existing vehicles since it does not require any changes in the vehicle's systems—in other words, it may even be deployed as an external component connected through the OBD-II port. We suggest an architectural design for the given modes, and include guidelines for a *safe-mode* manager, its clients, possible reactions, and recovery plans. We note that our system can rely upon any deployed anomaly-detection system to identify the potential attack.**

I. Introduction

A. Motivation

Modern vehicles are susceptible to cyber-attacks: this is since they are controlled by multiple dedicated computers (electronic control units - ECUs) that are typically connected not only to each other (e.g., over a CAN bus) but also to the outside world—often by wireless protocols (WiFi, Bluetooth, Cellular, etc.). These conditions, and the introduction of new technologies (that allow remote access to the vehicle internal systems) make vehicles vulnerable to potential new attack vectors of increasing number. Researchers have already shown that these attacks can be both feasible and severe (e.g., attacks on Jeep [1] and Tesla [2]).

Several defense mechanisms have been offered to identify attacks or block them - but probably none are perfect. This motivates us to look for a solution to limit the potential damage of an attack that already passed the vehicle's first line of defense. The vehicle's *Limp-mode*, that is designed to limit the damage of a mechanical malfunction, seems to be a good candidate for this purpose.

B. Related Work

Research into vehicle cyber-security has been growing since the first publication of Koscher et al. [3] in 2010. Using sniffing, fuzzing and reverse engineering of ECU's code, the authors succeeded in controlling a wide range of vehicle functions, such as disabling the brakes, stopping the engine, etc. Checkoway et al. [4] showed that a remote attack, without physical access to the vehicle, is also possible (via Bluetooth, cellular radio, etc.). Valasek and Miller [5] demonstrated actual attacks on Ford Escape and Toyota Prius cars via the CAN bus network. They affected the speedometer, navigation system, steering, braking and more. In 2015 it was reported [1], [6] that they remotely disabled a Jeep's brakes during driving, and caused Chrysler to recall 1.4M vehicles. Foster and Koscher [7] have also reported of the potential vulnerabilities in relatively new commercial OBD-II dongles (such as those used by insurance companies to track one's driving) which support cellular communication and may be even exploited via SMS. In 2016, a team of researchers from Keen Security Lab demonstrated a successful attack on the Tesla electrical vehicle [2], taking control over the vehicle through a bug in the Infotainment unit's browser, forcing the company to release an over-the-air software update.

Several ideas were offered to secure vehicles against cyber-attacks, including both active and passive solutions. One approach is to try and secure the internal communication of the vehicle - typically a CAN bus, by adding authentication to the messages (e.g., by using a cryptographic Message Authentication Code (MAC)). Several ideas were suggested, ranging from adding part of a MAC tag to the actual message's data field, to splitting the MAC into several pieces and layers as offered by Glas and Lewis [8]. Van Herrewege et al. [9] suggested to use a new light-weight protocol to better fit the CAN bus limitations. Their *CANAuth* protocol, also relied on the *CAN+* protocol of Ziermann et al. [10], which allowed them to split the authentication bits in between the sampling points of the bus. These solutions however require having a pre-shared key, which has its own key management challenges. A similar approach was adopted by the AUTOSAR standard, as defined by the Secure Onboard Communication (SecOC) mechanism [11], to add some authentication and replay prevention to the vehicle's internal networks.

Another, more aggressive, approach was offered by Matsumoto et al. [12] to try and destroy non-legitimate spoofed messages, by using a modified hardware to send *active-error* flags against identified spoofed messages. A centralized approach to combine the two previous ideas (using MAC for authentication and the *active-error* flags) was suggested by Kurachi et al. [13] to reduce the need to use modified hardware and share a key between all ECUs. In this approach a centralized modified ECU was used to both authenticate and destroy non legitimate messages. The later work of Kurachi et al. [14] demonstrated an actual implementation of a central gateway to include the above mechanism.

Another evolution of [12] was the work of Ujiie et al. [15] which replaced the usage of the MAC with other, non cryptographic, message analysis algorithms. They also implemented and tested their model in a real vehicle, taking into consideration important technical details, such as the error counters behavior, etc.

A different solution - the *Parrot* system [16], [17], was offered by Dagan and Wool to try and mitigate spoofing attacks. In this solution the defender launches a counter-attack of specially crafted defense messages, in order to intercept the attacker's next message, cause a set of collisions, and drive the attacker's ECU into a *bus-off* state (where it is temporally disconnected from the bus). This solution relies on some fine details of the CAN protocol [18] and can be implemented both in software and hardware.

Another approach is to try and identify un-authorized access to the internal network of the vehicle, by using Anomaly or Intrusion Detection Systems (IDS). Markovitz and Wool [19], [20] demonstrated the ability to classify the traffic over the CAN bus, where Marchetti et al. offered some anomaly detection mechanisms, based on an information theoretic algorithm [21] and on inspection of sequences of IDs [22]. Hamada et al. [23] offered to implement an IDS system that relies on the traffic density of some periodic messages.

A similar, although active approach, was offered to try and block un-authorized packets from entering the vehicle internal network, by using a secure gateway to separate the exposed vulnerable ECUs (such as the Infotainment system) from the rest of the network. Wolf et al. [24] suggested using a firewall to protect the sensitive portions of the vehicle's network, where Berg et al. of Semcon [25] suggested and implemented a prototype for a layered-architecture gateway, to protect the CAN bus from the Infotainment's IP domain of modern vehicles.

A different approach was to try and notify the driver on potential attacks [26], using different methods according to the notification severity. Note that the recently released UN-ECE Resolution on the Construction of Vehicles [27] Annex 6 (4.3.3) in fact requires driver notification in case a cyber-attack is detected.

There are several commercial companies attempting to cover various aspects in vehicle cyber-security [28], [29], [30], [31], [32] —some are still young and provide minimal details about their specific offerings.

Some leading manufacturers, such as NXP [33] and Bosch [34] offer a variety of products to secure the vehicles, ranging from Hardware Secure Modules (HSMs) to full fledged secure gateways. The existence of these products fits the wide-spreading holistic (in-depth / layered) approach for vehicle cyber-security, as described by Van Roermund et al. [35].

C. Contribution

This paper describes a concept for vehicle *safe-mode*, that may help reduce the potential damage of an identified cyber-attack. Unlike other defense mechanisms, that try to block the attack or simply notify of its existence, our mechanism responds to the detected breach, by limiting the vehicle's functionality to relatively safe operations, and optionally activating additional security counter-measures. This is done by adopting the already existing mechanism of *Limp-mode*, that was originally designed to limit the potential damage of a mechanical malfunction and let the vehicle "limp back home" in relative safety. We further introduce two modes of *safe-mode* operation to raise the flexibility and the number of potential integration plans that may fit the manufacturer's needs. In *Transparent-mode*, when a cyber-attack is detected the vehicle enters its pre-configured *Limp-mode*; In *Extended-mode* we suggest to use custom messages that offer additional flexibility to both the reaction and the recovery plans. While *Extended-mode* requires modifications to the participating ECUs, *Transparent-mode* may be applicable to existing vehicles since it does not require any changes in the vehicle's systems—in other words, it may even be deployed as an external component connected through the OBD-II port. We also suggest an architectural design for the given modes, and include guidelines for a *safe-mode* manager, its clients, possible reactions and recovery plans.

Organization: In the next section we describe some preliminaries. In Section III we introduce the *safe-mode* concept and a suggested architecture. Section IV describes various possible reactions, recovery plans and some potential related problems. We conclude with Section V.

II. Preliminaries

A. Limp-mode

Limp-mode (also known as *Fail Condition*) was originally designed as a safeguard to limit the potential damage of either a mechanical or an electrical malfunction, and let the vehicle "limp back home" for treatment, without risking further damage and without forcing the vehicle to a complete stop. In modern vehicles, *Limp-mode* is activated automatically after an ECU detects a malfunction in one or more vehicle subsystems.

It is possible to distinguish between two different types of *Limp-modes*: a *local limp-mode* that is limited to the operation of a single ECU; and a *global limp-mode* affecting the global state of the vehicle.

Local limp-mode is a feature often supported by micro-controllers used to implement ECUs. It is usually provided as a physical pin that, when activated by applying the proper voltage, makes it possible to override the normal behavior of the micro-controller and drive the output pins directly to pre-configured settings (see as an example the technical documentation of the DRV8305-Q1 automotive micro-controller [36]). *Local limp-mode* can be easily deactivated by restoring the normal voltage to the *Limp-mode* pin, thus restoring the normal operation of the micro-controller.

Global limp-mode is activated when one of the central ECUs connected to the in-vehicle network, usually the Body Control Module (BCM), or the Engine Control Module (ECM) detects possible fail conditions by analyzing the values of the messages received from the CAN bus (e.g., see the Central BCM produced by Infineon [37]). For instance, *global limp-mode* may be activated if the coolant temperature rises above safe values [38] or if the Powertrain control module detects a failure (or near-failure) condition in the transmission [39]. Depending on the type and on the severity of the failure, the central ECU triggers a set of operations that restrict the vehicle to a limited set of failsafe states. As an example, when in *Limp-mode* the vehicle speed might be electronically limited to a set threshold, the transmission might be fixed in a second gear or, if an issue related to the engine is detected, *Limp-mode* can shut the engine off and gradually reduce the vehicle speed to a complete stop. The exact counter-measures deployed when in *Limp-mode* depend on the specific settings defined by the car manufacturer.

Depending on the car maker and model, *global limp-mode* may be implemented by activating the *local limp-mode* of some peripheral ECUs, letting the main ECU directly control them.

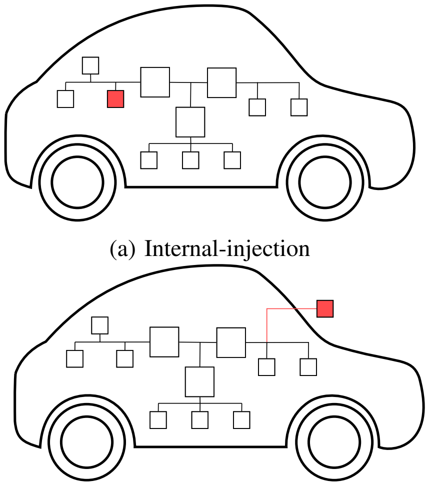
Deactivation of the *global limp-mode* also depends on the nature and severity of the detected failure. For example, *Limp-mode* that is activated due to the detection of transient failure conditions, is usually reset automatically after restarting the vehicle, or after a predefined amount of time. In some cases, the car owner can perform a sequence of operations that resets the *Limp-mode* for non-severe failures, such as switching the car ignition on, and pressing and releasing the throttle pedal for a given number of times [40]. On the other hand, more severe failures may require a manual reset of the *Limp-mode*, which is usually performed by operators of authorized car services by physically connecting to the OBD-II port and executing proprietary diagnostic protocols.

In the remainder of the paper we use the term *Limp-mode* to refer to the *global limp-mode*.

B. *The Adversary Model*

We assume that an attacker is able to gain access to the CAN bus of a modern vehicle and to inject forged CAN messages. The amount and the nature of injected messages may vary depending on the final goal of the attacker. We identify two attack injection modes that model the attackers' abilities: *internal-injection* and *external-injection*.

- **Internal-injection:** In this attack scenario the adversary manages to gain full control over one or more ECUs. Hence the attacker can directly control some functions of the vehicle by altering the logic of the compromised ECUs. The attacker can also exploit the compromised ECUs to inject arbitrary messages over the CAN bus. This attack mode is represented by Figure 1a.
- **External-injection**: In this scenario the adversary cannot directly control any ECU. However, he can still inject arbitrary CAN messages over the CAN bus either by physical access (such as a direct connection to the OBD-II diagnostic port) or remote access (e.g., by transmitting malicious messages over a legitimate wireless channel). This attack mode is shown in Figure 1b. We note that most of the published attacks (recall Section I-B) fall into one, or both of these injection modes.



(a) Internal-injection

(b) External-injection

Fig. 1: Adversary models: the attacker's access points are marked in red.

III. Vehicle Safe-mode - suggested architecture

A. *Overview*

The concept of vehicle *safe-mode* proposed in this paper is similar in principle to the *Limp-mode* mechanism (recall Section II-A): The *safe-mode* mechanism is offered to let the vehicle "limp back home" in case a cyber-breach is detected, while reducing the potential damage of such an attack to the vehicle, its driver, the passengers, and its surroundings.

The Vehicle *safe-mode* system operates as follows: When a cyber-attack is detected, a *safe-mode* manager (*SMManager*, see Section III-C) puts the vehicle into a *safe-mode* condition—in which several operations are limited or disabled, by sending an alert triggering message (*TMessage*) to other ECUs. The *SMManager* bases its decision on any existing IDS-like systems, that flag suspicious cyber-related events. This decision should typically include the recommended level of alert and the chosen reaction that can be encoded into the broadcast *TMessages*. See Figure 2 for a system overview.

For possible deployment, we further present two modes of operation: In *Transparent-mode* (Section III-B1) the *SM-Manager* only causes the neighboring ECUs to enter into

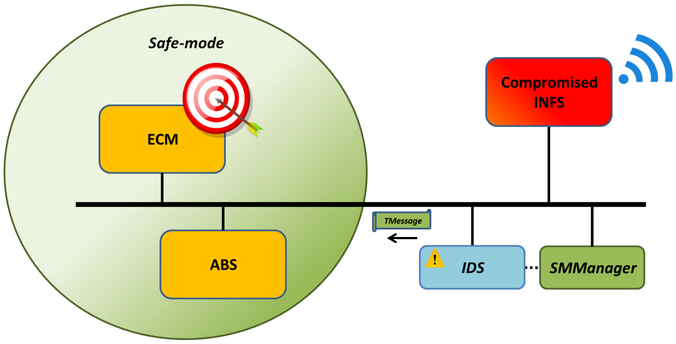


Fig. 2: The system overview. Note that the *SMManager* can be connected directly to the IDS system, or alternatively, get its feedback over the bus.

their pre-configured *Limp-mode* state, in order to limit the functionality of the vehicle and reduce the potential danger. The main advantage of this mode is its immediate applicability to virtually all modern vehicles, since the introduction of the *SMManager* is transparent with respect to all other ECUs. in particular, the system may be deployed by adding a single OBD-connected entity to include the *SMManager*, with optional IDS capabilities.

Alternatively, the *Extended-mode* (Section III-B2) requires adding a novel software component, called *safe-mode* client (*SMClient*, Section III-D) to chosen ECUs. The purpose of the *SMClient* is to process and react to the custom *TMessages* sent by the *SMManager*. Using this mode adds more flexibility to the system, by making it possible to design and implement customized reactions per individual ECU and state of alert.

Special care should be given to the recovery options (Section IV-B) - that make it possible to exit from *safe-mode* and to restore full vehicle functionalities. This is required to make sure that the attacker will not have an easy way to take the vehicle out of *safe-mode*, while simultaneously ensuring that the driver will not have a too-difficult time to return to normal operation.

B. Operation modes

1) Transparent-mode: In this mode of operation, the *SMManager's* goal is to put the relevant ECUs into *Limp-mode* in order to reduce the potential damage of an identified attack, by triggering the pre-existing *Limp-mode* mechanism of each relevant ECU. Doing so may be effective in reducing the potential damage to the vehicle and its passengers under the assumption that entering *Limp-mode* would typically limit the vehicle's operation in a way that may also help to maintain its safety (e.g., by putting the car on a rigorous speed limit, keeping it in a low gear).

For this purpose, the *SMManager* can maintain a list of all relevant CAN bus messages (or any other protocol in use) that typically cause each ECU to enter *Limp-mode*. This list can be maintained by a simple updatable table of the relevant *TMessages* per ECU (see Table I). Note that this table can include several different lines per ECU, in case there are multiple *TMessages* per ECU (in this case the *SMManager* can decide, per ECU, whether to send all or only some of the available *TMessages*).

TABLE I: a sketch of pre-existing *TMessages* that can trigger *Limp-mode*

ECU	Msg ID	Data
ECM	014	”Dangerous high engine temperature”
ECM	014	”Major engine malfunction”
ABS	004	”Dangerous low oil pressure”
TPM	020	”Dangerous low air pressure”

The properties of this mode potentially make the *Safe-mode* protection applicable to any existing vehicle, e.g., by connecting an after-market device (to include the *SMManager* and some anomaly-detection component) to its OBD-II port. A more sophisticated after-market device (e.g., one using a smart-phone) can include more sophisticated notification and recovery options to the vehicle's driver (Sections IV-A3, IV-B).

The drawback of this mode is that *safe-mode* reactions are bound to be the same reactions that the car maker already planned for the *Limp-mode*. Hence counter-measures that are designed specifically against cyber-breaches cannot be implemented.

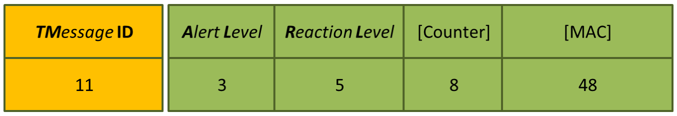
Special care should be taken under this mode to make sure that no collision will occur between the *SMManager's TMessages* and genuine messages of the original responsible ECU (see Section IV-C).

2) Extended-mode: In this mode of operation, the *SMManager* is able to put chosen ECUs into a customized *safe-mode*, rather than into their pre-configured *Limp-mode*. This mode offers more flexibility to the designer, at the cost of adding at least some software update—the *SMClient*—to participating ECUs.

This mode of operation gives us the freedom to chose any reaction, per ECU, to reduce the potential damage of a cyber-breach to the overall safety of the vehicle and passengers. This freedom also provides us more possibilities to react differently according to the type and severity of the identified attack, as further described by the *SMManager's* chosen alert-levels and triggered reaction (Sections III-E and IV-A).

In addition, this mode of operation can make the vehicle's *safe-mode* more robust against potential manipulations of an adaptive attacker, since it allows defending the mechanism itself (e.g., by adding some authentication to the triggering *TMessages*, etc.). This mode can be also used to actually fight some of the attacks e.g., by requiring the addition of some authentication to all of the critical CAN bus messages (Section IV-A2) when under a spoofing attack (saving this overhead during quiet times).

Another potential advantage of this mode, is the extra flexibility that is given to choose the driver notification and recovery options; custom messages can notify the driver (e.g., through the Infotainment or Cluster units) about the identified attack and the state of alert (Section IV-A3); Proper notification can also let the driver decide whether the chosen



***TMessage* ID**	***Alert Level***	***Reaction Level***	[Counter]	[MAC]
11	3	5	8	48

Fig. 3: Possible structure of an *Extended-mode TMessage*. The numbers represent the field length in bits. Note that the ID field is a regular CAN-ID-field, while the other fields fit into the CAN 8 byte data-field; Both the counter and the MAC fields are optional; *Transparent-mode TMessages* are regular (*Limp-mode* triggering) CAN messages.

reaction is sufficient, or alternatively the *safe-mode* state can be manually overridden (Section IV-B).

In this mode the *SMManager* can maintain a table of all relevant triggering *safe-mode TMessages*, per ECU/Alert-level, to include the type of reaction, as further defined in Section IV-A1. We note that a similar table can be used for both modes of operation, even though the *Transparent-mode* should be able to use a simpler one.

A typical custom *TMessage* should be based on the underlying protocol (typically the CAN protocol). Unlike the *Transparent-mode TMessage*, it can contain, apart from its message ID, the vehicle's Alert-level *AL*, the required Reaction-level *RL*, and optionally a replay counter and a truncated MAC of a chosen algorithm (e.g., HMAC). A suggestion for such a CAN based message, with an 8-byte data field, is depicted in Figure 3.

An *SMClient* (Section III-D) should be added, optionally as a software patch, to any participating ECU to allow proper identification, processing and reaction to the custom *safe-mode TMessages*.

The *SMManager* can also be responsible for the necessary key management and distribution, in case the *safe-mode* system incorporates authentication codes in the *TMessages*. Several solutions can be chosen to cover key management, ranging from factory serialization to specialized solutions, as offered by Mueller and Lothspeich [41].

Finally we would like to note that combinations of the two presented modes may also apply, allowing vehicles to utilize a mixture of *SMClient*-supportive and non-supportive ECUs.

C. Safe-mode Manager

The *SMManager* is responsible to process the IDS feedback, calculate the vehicle alert-level (AL), decide on the relevant reaction-level (RL), and finally put the vehicle into, and out of, *safe-mode*, by sending the relevant *TMessages*.

The *SMManager* may also be responsible for any related key-management aspects in case of using cryptography for either the protection of the *TMessages* or the switch into secure-communication when under attack.

Regardless of the selected configuration, the *SMManager* should be able to receive the IDS alerts, either directly from the bus, or from its hosting ECU (which can also comprise of both the *SMManager* and the IDS).

We also note that the *SMManager* can be implemented in either software or hardware—a hardware implementation should increase the cyber-resistance of the suggested mechanism, while possibly increasing its cost and making deployment more challenging.

1) Topology: The *SMManager* can be implemented differently according to the topology of the internal networks and the computational load of each ECU. In this section we propose two different topologies: as an *Independent SMManager*, or as an *Incorporated SMManager*.

The ***Independent SMManager*** is the implementation of the *SMManager* on a dedicated hardware module. This option allows both possibilities for an internal and an external module. The internal *SMManager* can be seen as a dedicated ECU, responsible for collecting the different notifications across the internal network in order to properly start the vehicle *safe-mode* if necessary.

The external *SMManager* can be implemented as a dedicated dongle connected through the OBD-II interface (Figure 4a). In order to work as an external module, the *SMManager* must be able to observe the data packets flowing on the internal network, and to broadcast the necessary *TMesagges* when needed: in particular this means that the OBD-II interface must allow message transmission into the network, and must be connected to the relevant CAN bus segment(s). The external module approach allows implementations of the *safe-mode* logic in vehicles that were designed without it, thus extending the proposal of this paper to past and present vehicular systems.

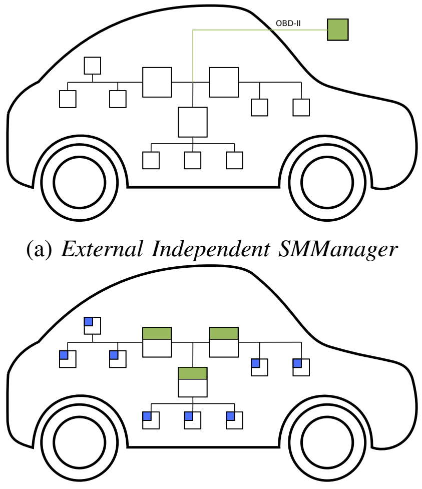
The ***Incorporated SMManager*** is the implementation of the *SMManager* on existing ECUs of the internal network, as part of the vehicle specification (as shown in Figure 4b). This option makes the *SMManager* part of the whole system *by design*. An *Incorporated SMManager* allows three different topologies for its implementation:

- *Centralized SMM:* the logic for the *SMManager* operations is part of the code of a centralized ECU (e.g., the ECM or BCM)
- *Distributed SMM:* the logic for the *SMManager* operations is spread over multiple ECUs across the network, each one with its specific set of operations needed for monitoring and eventually triggering the vehicle *safe-mode*.
- *Hybrid SMM:* a composition of the two previous topologies: Different instances of the same *SMManager* are responsible of monitoring and collecting different pieces of information, which they filter and forward to the centralized *SMManager*—which ultimately decides whenever it is necessary to start the vehicle *safe-mode*.

D. Safe-mode Client

In order to support the *Extended-mode*, an *SMClient* is required. This client should be added, e.g., as a software patch, to any participating ECU to allow proper identification, processing and reaction to the custom *SMManager TMessages*.

In this mode, the client can maintain a list of update-able reactions per *TMessage* encoded reaction-level (*RL*). These reactions can be chosen by the manufacturer with the goal



(a) *External Independent SMManager*

(b) *Distributed Incorporated SMManager*

Fig. 4: *Safe-mode Manager* suggested topologies. The *SMManagers* are marked in green, while the *SMClients* are marked in blue. Note that Figure (a) shows one example for a system in *Transparent-mode*, while Figure (b) shows an example for a system in *Extended-mode*.

of limiting the potential damage of the possible attacks to the overall vehicle and passengers' safety.

If the *TMessages* are authenticated, to prevent adversarial manipulations, then the *SMClient* should also validate the authentication tag embedded in the *TMessage* using the algorithm in use, (e.g., the 48 bit truncated HMAC, recall Figure 3) on the relevant section of the received *TMessage*. If the authentication also involves an anti-replay counter, then the *SMClient* must also validate that the counter value c is acceptable (e.g., $Vmax < c < Vmax + k$ where $Vmax$ demotes the maximal value observed on previous *TMessages* and k is a configured window-size).

Upon receiving a relevant (optionally authenticated) *TMessage*, the *SMClient* should put its hosting ECU in *safe-mode*, by performing the relevant (per *RL*) pre-configured actions according to its reaction table (e.g., ignore non critical messages, limit the operation, etc., see Section IV-A2 for further details).

The *SMClient* must also support the chosen recovery mechanism (see Section IV-B) to allow proper recovery of its hosting ECU at the right time and under the right conditions. The recovery can be done either unilaterally (e.g., after a reset, or after X seconds, etc.), or by a special recovery-triggering message (with a unique *TMessage* ID or *RL*), or according to other pre-defined conditions. We note that special care should be given to this procedure to keep this mechanism both robust and applicable.

E. The vehicle Alert-level

Independently of the chosen implementation (Transparent or Extended), the *SMManager* is responsible for evaluating the vehicle's Alert-level *(AL)*. Different levels of alert reflect different threat levels and imply the deployment of appropriate reactions, as will be further discussed in Section IV-A1.

To evaluate the current *AL*, the *SMManager* relies upon any anomaly detection system deployed within the vehicle. In particular, intrusion detection systems (IDS) represents the main source of information useful for *AL* evaluation. IDS for in-vehicle networks of modern vehicles have already been proposed in the literature [21], [22], [23]. All these systems analyze different features of the messages broadcast over the CAN bus and issue alerts whenever evidence of an attack is found. The *SMManager* collects and analyzes all of the security related alerts (or lack thereof) and modifies the current *AL* accordingly.

For concreteness, as an example we suggest that the *AL* can be comprised of five different levels to represent the severity of the alert, denoted by *AL1* (low severity) to *AL5* (critical severity).

IV. Possible reaction, recovery plans, and potential problems

A. Reaction

In this section we suggest several steps that can be taken by the *SMManager* after the detection of a cyber-breach. The reaction of the *SMManager* comprises of two main parts:

- *Notification:* optional feedback to the driver and the vehicle surroundings about the identified attack and the chosen reaction.
- *Action: under-the-hood* counter-measures to limit the potential damage of the attack, narrow the possibilities of the attacker, and even, under some cases, try to stop the attack.

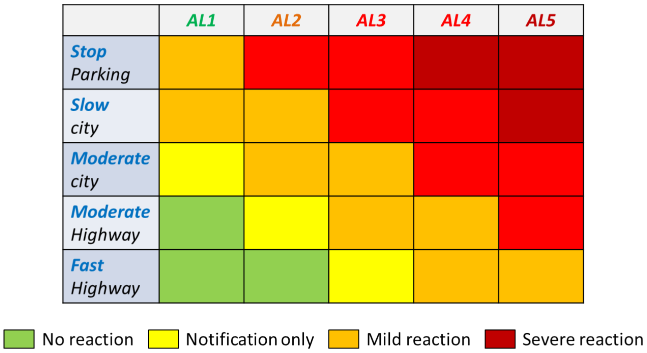
Both the notifications and actions can be triggered sequentially or simultaneously, depending on the alert-level and on the content of the *Reaction-Matrix* as further explained below.

1) Reaction-Matrix: The *Reaction-Matrix* is the structure used by the *SMManager* to determine the reaction-level *(RL)* that encodes the required protective steps and notifications. The calculation of the *RL* depends on two metrics:

- the current Alert-level (*AL*, recall Section III-E)
- the current Vehicle-condition (*VC*).

The value of the current *VC* represents the current conditions of the vehicle dynamics, including speed, yaw, roll, pitch, lateral acceleration and outputs of the ABS and ESP systems. Intuitively, it is important to consider the current vehicle conditions to make sure that reactions decided by the *SMManager* are appropriate, and do not cause more harm than the attack itself.

As an example, if counter-measures were to be deployed without considering the vehicle conditions, the *SMManager* might decide to exclude the electronic stability protection or other advanced driving assistance systems. While this decision may be the most appropriate for a vehicle running at low speed on a straight road, it may cause severe safety risks to a vehicle in dangerous driving conditions (e.g., at high speed, or under high lateral accelerations). To prevent similar situations, the



	AL1	AL2	AL3	AL4	AL5
Stop Parking	🟧	🟥	🟥	🟫	🟫
Slow city	🟧	🟧	🟥	🟥	🟫
Moderate city	🟨	🟧	🟧	🟥	🟥
Moderate Highway	🟩	🟨	🟧	🟧	🟥
Fast Highway	🟩	🟩	🟨	🟧	🟧

🟩 No reaction 🟨 Notification only 🟧 Mild reaction 🟫 Severe reaction

Fig. 5: An example for a 5x5 *Reaction-Matrix*. Note that the rows represent the current *VC* (speed/location), the columns represent the chosen *AL* (1-5), and the colors of the internal blocks represent the chosen *RL* (nothing to severe).

reaction-matrix makes it possible to react to the same *AL* by deploying different counter-measures based on different vehicle conditions.

The calculated *RL* is used to determine the most appropriate reaction, aiming to bring the vehicle to a safe state that nullifies or limits the safety consequences of the detected cyber-breach.

An example for a Reaction-Matrix is depicted in Figure 5 to include the different *RL* to match every possible combination of a given 5x5 *AL/VC* structure.

2) Actions: After activating *safe-mode*, the *SMManager* triggers different actions in order to react to possible dangerous situations, according to the reaction-matrix. These actions could take place before, after or together with the notification phase, and it is essential to define their timing sequence according to every case (e.g., activating drastic actions, like bringing the car to a complete stop *before* driver and external notification, could have dangerous repercussions).

Actions differ according to the triggered reaction-level *(RL)*, and can have varying intensity according to the previously raised *AL*. While lower *RL* could trigger very limited actions in order to recover from a less dangerous situation, a higher *RL* will trigger more invasive actions, aimed to react to more dangerous situations.

The selected triggered actions can range from the already existing *Limp-mode* operations (e.g., limit the vehicle speed) to custom steps such as those presented below:

Ignore all non-critical messages: This action has a twofold advantage: it allows ECUs to ignore attacks that leverage non-critical messages; and further, it reduces the computational power required by ECUs to operate the vehicle, thus leaving more room for computations related to the safe-mode. We note that each ECU can maintain a list of non-critical messages.

Shutdown particular ECUs: In case of identifying a compromised ECU that puts the vehicle in danger. This action could be triggered after the identification of a Denial-of-Service attack on the internal network that leverages messages produced by the victim ECU.

Reset particular ECUs: Similar to the above, only less aggressive. This action can be used when dealing with a relatively important ECU, or as an initial step in a graceful shutdown. We note that both the reset and the shutdown options should be chosen with great care, to make sure that they won't put the vehicle in a dangerous situation (e.g., in the case that the selected ECUs are critical ones, the reset action is a viable option only if the expected outcome of this action is absolutely safe.)

Trigger the usage of authentication to some (CAN) messages: using cryptographic primitives (e.g., truncated HMAC) in order to mitigate spoofing attacks of critical messages. We note that using this option only when under attack (and between chosen ECUs) reduces the related overhead (traffic/computation wise) of a similar permanent solution.

Trigger the encryption of some (CAN) messages: using cryptographic primitives in order to encrypt the data of selected (CAN) messages when under attack. We note that both this and the previous options can be implemented in software or hardware, and that hybrid solutions may also be chosen to maximize the strength of the chosen solution, utilize the hosting ECU at best (e.g., by using the hardware capabilities of existing HSMs to establish a secure-like channel, alongside weaker ECUs with software only implementations of the chosen algorithms to provide some basic authentication.)

Segment isolation - the submarine model: In a typical segmentation of the vehicular networks - *Powertrain*, *Body* and *Infotainment*, a bus gateway can isolate a compromised segment of the network from the others. This solution allows fast reaction after the detection of a potential intrusion on any segment of the CAN bus, thus limiting the intrusion only to the affected network and preventing its spreading to the other segments. Further segmentation can be recommended to allow better flexibility. We note that special care should be taken if choosing this option to make sure that critical ECUs could still communicate.

Secondary emergency CAN bus: Implementing a secondary limited CAN bus, connecting only the critical ECUs on a different interface, could prevent some of the segment-isolation potential problems. We note that this solution can also be used to raise the accuracy of any existing IDS, by adding some redundancy to the system; During the vehicle normal operation, the secondary CAN bus could be used in a redundant way, sending duplicated packets (already sent on the primary CAN bus) of selected messages. Intrusion Detection Systems' could compare the two different networks in order to detect any intrusion on the primary CAN bus.

3) Notification: Notifications can be both internal or external. Internal-notifications are used to notify the driver that the *SMManager* is performing different actions in order to react to the calculated *AL*. These notifications can be acoustic, visual or even include haptic feedback on different parts of the cockpit, like the steering wheel or the pedals. A more articulated schema for vehicle internal-notifications can be found in [26].

The necessity to externalize the notification is extremely important and needs to be taken in consideration. External-notifications are mostly used in order to notify other drivers, vehicles, and nearby pedestrians of a potentially dangerous situation. External-notifications can consist of visual feedback, e.g., blinking turning indicators, brake lighting signals or even dedicated custom “under-attack” lights. More sophisticated external-notifications can be designed e.g., using vehicle-to-vehicle (V2V) technology to make adjacent vehicles enter a preventive “safe-mode”, or use vehicle-to-infrastructure (V2I) communication channels to trigger roadside actions - to warn and protect adjacent entities.

B. Recovery plans

The *Recovery* is the last phase of the vehicle *safe-mode*, and can take place only after the reaction has terminated. The recovery procedure is aimed to allow bringing the vehicle back to normal operation at the proper time—after the attack is considered to be over, or under safe-confinements (e.g., engine off, in an authorized garage, etc.).

The *SMManager* is responsible to decide when and where the recovery operation can begin (e.g., per ECU, *AL*, *VC*). We further suggest several modes of recovery: Self, Driver-initiated, Garage-authorized.

Self-recovery can allow the procedure to be started by the *SMManager* itself, without even notifying the driver. A self-started recovery procedure should be applied only if non-critical parts of the network were involved in the attack, or when the identified attack was not severe.

Driver-initiated recovery can be used when some interaction with the driver is required (e.g., by physically approving the initiation of the procedure). We note that this option can contribute to the robustness of the mechanism, by reducing the possibility of the attacker to initiate the recovery procedure during the attack.

Authorized-Garage recovery can be required when recuperating from a major attack (e.g., on critical ECUs), or when the attack was not fully terminated. This option requires bringing the vehicle to an authorized garage, and optionally the usage of special manufacturer tools, for further inspection and safe recovery.

We note that it may also be possible to initiate a remote-recovery procedure as an intermediate step, and that a combination of the above procedures may also be applicable.

The *SMManager* can use the following metrics in order to choose which recovery-mode can be allowed and under which conditions:

- ***iAL:*** the initial Alert-level,
  computed before the reaction phase
- ***RL:*** the previously calculated Reaction-level,
  computed before the reaction phase.
- ***aAL:*** the actual Alert-level,
  computed after the reaction phase
- ***aVC:*** the actual Vehicle-condition,
  computed after the reaction phase

C. Potential problems

The *safe-mode* mechanism may have some limitations and side effects as presented below. We recommend to take them into account when considering this solution.

False positives: The IDS or anomaly detection component is a critical, yet external, part of the *safe-mode* system, since it is responsible for providing the input to trigger the *SMManager*’s *safe-mode* reaction. This means that any problem or limitation of the IDS systems can affect the *safe-mode* mechanism. In particular, IDS systems are susceptible to false alarms, which means that a *safe-mode* state may be activated unnecessarily.

However, when we note the severity of potential false-negatives (unlimited vehicle operation under malicious control), one could argue that false-positives may be acceptable. This argument can be strengthened by the fact that *safe-mode* only limits the vehicle operation, and does offer some built-in recovery plans.

Either way, we can recommend taking the following two steps: Use more than a single IDS-like system as a source of input to the *SMManager*; and take the possibility of false-positives into account when configuring the *SMManager* triggering threshold and recovery plans.

Adversarial triggering: We note that an adaptive adversary may try to trigger the *safe-mode* mechanism on his own, by sending the relevant triggering *TMessages* when the system is deployed in *Transparent-mode*, or in a non-secure *Extended-mode*. However, this can be viewed as another flavor of false-positive case, and should be handled as described above.

Transparent-mode, *TMessage* collisions: Special care should be taken when using *Transparent-mode* to ensure that no CAN bus collisions will occur between the *SMManager* *TMessages* and genuine messages of the original responsible ECU. Collisions may happen since the *SMManager* may broadcast *TMessages* using the same ID, and but different content, than those broadcast by the ECU that is responsible for the given message ID. For example, if one of the *TMessages* is an over-heating alert message (recall Table I) with ID 014, and the ECM uses the same message ID to broadcast the engine temperature at a fixed frequency, even when conditions are normal; since the two messages have exactly the same CAN priority, it is possible that collisions may happen, (cf. [16]). To eliminate such possibilities, we recommend to either use a carefully chosen broadcast *Tmessage* schedule (e.g., broadcast immediately after the genuine message) or to simply avoid using the same ID. We note that in *Incorporated-mode* this problem cannot exist since the *TMessages* will use dedicated message IDs.

Transparent-mode, *TMessage* overriding: Another challenge in *Transparent-mode* is that of overriding. Under a similar scenario, even without the danger of CAN bus collisions. The ECU responsible for some message ID, which is oblivious to the cyber-attack in progress, may override the effect of a *TMessage* by its own genuine broadcast of a message with the same ID. Using the same example as before, the genuine ECM 014 message (normal temperature) can make the neighboring dependent ECUs understand that all is fine, or alternatively, make them go in-and-out of *Limp-mode* in a loop. Recommendations that are similar to those mitigating the *TMessage* collisions can be used to mitigate this challenge as well. Furthermore, broadcasting more than a single *TMessage*, at a fixed rate, may help ensure that the target ECU will remain in *Limp-mode* (even if getting occasional countering messages).

V. Conclusion and Future Work

This paper describes a concept for vehicle *safe-mode*, that may help reduce the potential damage of an identified cyber-attack. Unlike other defense mechanisms, that try to block the attack or simply notify of its existence, our mechanism responds to the detected breach, by limiting the vehicle's functionality to relatively safe operations, and optionally activating additional security counter-measures. This is done by adopting the already existing mechanism of *Limp-mode*, that was originally designed to limit the potential damage of either a mechanical or an electrical malfunction and let the vehicle "limp back home" in relative safety. We further introduce two modes of *safe-mode* operation: In *Transparent-mode*, when a cyber-attack is detected the vehicle enters its pre-configured *Limp-mode*; In *Extended-mode* we suggest to use custom messages that offer additional flexibility to both the reaction and the recovery plans. While *Extended-mode* requires modifications to the participating ECUs, *Transparent-mode* may be applicable to existing vehicles since it does not require any changes in the vehicle's systems—in other words, it may even be deployed as an external component connected through the OBD-II port. We suggest an architectural design for the given modes, and include guidelines for a *safe-mode* manager, its clients, possible reactions, and recovery plans. We note that our system can rely upon any deployed anomaly-detection system to identify the potential attack. We also identified potential challenges to our approach and suggested possible mitigations. We conclude by saying that building a prototype to implement and test the suggested mechanism, could help us better understand, configure, and evaluate the described mechanism.

References

[1] A. Greenberg, "Hackers remotely kill a Jeep on the highwaywith me in it," http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, 2015.

[2] D. Pauli, "Hackers hijack Tesla Model S from afar, while the cars are moving," http://theregister.co.uk/2016/09/20/tesla_model_s_hijacked_remotely, 2016.

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *IEEE Symposium on Security and Privacy (SP)*, May 2010, pp. 447–462.

[4] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028073

[5] D. C. Miller and C. Valasek, "Adventures in automotive networks and control units," http://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf, 2014, [Online; accessed 22-July-2015].

[6] A. Greenberg, "After Jeep hack, Chrysler recalls 1.4m vehicles for bug fix," http://www.wired.com/2015/07/jeep-hack-chrysler-recalls-1-4m-vehicles-bug-fix/, 2015.

[7] I. Foster and K. Koscher, "Exploring controller area networks," *USENIX ;Login: magazine*, vol. 40, no. 6, 2015.

[8] B. Glas and M. Lewis, "Approaches to economic secure automotive sensor communication in constrained environments," in *11th Int. Conf. on Embedded Security in Cars (ESCAR 2013)*, 2013.

[9] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011.

[10] T. Ziermann, S. Wildermann, and J. Teich, "Can+: A new backward-compatible controller area network (CAN) protocol with up to 16× higher data rates." in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.* IEEE, 2009, pp. 1088–1093.

[11] AUTOSAR, "AUTOSAR secure onboard communication (SecOC), version 4.3," https://www.autosar.org/standards/classic-platform, 2016.

[12] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, and K. Oishi, "A method of preventing unauthorized data transmission in controller area network," in *IEEE Vehicular Technology Conference (VTC Spring)*. IEEE, 2012, pp. 1–5.

[13] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata, "CaCAN—centralized authentication system in CAN (controller area network)," in *12th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*, 2014.

[14] R. Kurachi, H. Takada, T. Mizutani, H. Ueda, and S. Horihata, "SecGW secure gateway for in-vehicle networks," in *13th Int. Conf. on Embedded Security in Cars (ESCAR 2015)*, 2015.

[15] Y. Ujiie, T. Kishikawa, T. Haga, H. Matsushima, T. Wakabayashi, M. Tanabe, Y. Kitamura, and J. Anzai, "A method for disabling malicious CAN messages by using a centralized monitoring and interceptor ECU," in *13th Int. Conf. on Embedded Security in Cars (ESCAR 2015)*, 2015.

[16] T. Dagan and A. Wool, "Parrot, a software-only anti-spoofing defense system for the CAN bus," in *14th Int. Conf. on Embedded Security in Cars (ESCAR 2016)*, Munich, Germany, Nov. 2016.

[17] ——, "Testing the boundaries of the Parrot anti-spoofing defense system," in *5th Embedded Security in Cars (ESCAR USA'17)*, Ypsilanti, MI, USA, Jun. 2017.

[18] Robert Bosch GmbH, "CAN specification, version 2.0," http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf, 1991.

[19] M. Markovitz and A. Wool, "Field classification, modeling and anomaly detection in unknown CAN bus networks," in *13th Embedded Security in Cars (ESCAR'15)*, Cologne, Germany, Nov. 2015.

[20] ——, "Field classification, modeling and anomaly detection in unknown CAN bus networks," *Vehicular Communications*, vol. 9, pp. 43–52, 2017.

[21] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Sept 2016, pp. 1–6.

[22] M. Marchetti and D. Stabili, "Anomaly detection of can bus messages through analysis of id sequences," in *28th IEEE Intelligent Vehicle Symposium (IV2017)*, June 2017, pp. 1–6.

[23] Y. Hamada, M. Inoue, S. Horihata, and A. Kamemura, "Intrusion detection by density estimation of reception cycle periods for in-vehicle networks: A proposal," in *14th Int. Conf. on Embedded Security in Cars (ESCAR 2016)*, Munich, Germany, Nov. 2016.

[24] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," in *Workshop on Embedded Security in Cars*, 2004.

[25] J. Berg, J. Pommer, C. Jin, F. Malmin, and J. Kristensson, "Secure gateway - a concept for an in-vehicle IP network bridging the infotainment and the safety critical domains," in *13th Embedded Security in Cars (ESCAR'15)*, 2015, [Online; accessed 27-July-2015].

[26] T. Hoppe, S. Kiltz, and J. Dittmann, "Adaptive dynamic reaction to automotive IT security incidents using multimedia car environment," in *2008 The Fourth International Conference on Information Assurance and Security*, Sept 2008, pp. 295–298.

[27] UN-ECE, "UN-ECE Resolution on the Construction of Vehicles (RE.3) ECE/TRANS/WP.29/78/Rev.6 Annex 6 (4.3.3)," https://www.unece.org/trans/main/wp29/wp29wgs/wp29gen/wp29resolutions.html, July 2017.

[28] Arilou, "http://ariloutech.com," 2015, [Online; accessed 22-July-2015].

[29] TowerSec, “http://tower-sec.com,” 2015, [Online; accessed 22-July-2015].

[30] Argus Cyber Security Ltd., “http://argus-sec.com,” 2015, [Online; accessed 22-July-2015].

[31] Security inMotion, “http://www.security-inmotion.com,” 2015, [Online; accessed 22-July-2015].

[32] Karamba security, “https://www.karambasecurity.com,” 2017, [Online; accessed July-2017].

[33] NXP automotive, “http://www.nxp.com/applications/automotive,” 2017, [Online; accessed July-2017].

[34] Bosch mobility solutions, “http://www.bosch-mobility-solutions.com/en,” 2017, [Online; accessed July-2017].

[35] T. van Roermund, A. Birnie, R. Moran, and J. Frank, “Securing the in-vehicle network of the connected car,” in *14th Int. Conf. on Embedded Security in Cars (ESCAR 2016)*, Munich, Germany, Nov. 2016.

[36] Texas Instruments Incorporated, “DRV8305-Q1 three-phase automotive smart gate driver with three integrated current shunt amplifiers and voltage regulator,” http://www.ti.com/lit/ds/symlink/drv8305-q1.pdf, 2016.

[37] Infineon Technologies AG, “Driving the future of automotive electronics, automotive application guide,” http://www.infineon.com/dgdl/Infineon-Automotive_Application_Guide-ABR-v00_00-EN.pdf?fileId=db3a30431c48a312011c6696b47402cc, 2014.

[38] Hagens Berman Sobol Shapiro LLP, “Ford shelby gt350 mustang overheating,” https://www.hbsslaw.com/cases/ford-shelby-gt-mustang-overheating, 2005.

[39] Transmission Repair Cost Guide, “What is limp mode? causes and what to do,” https://www.transmissionrepaircostguide.com/limp-mode/, 2015.

[40] AnthonyJ350, “Nissan 350z/ infiniti g35 ecu reset (check engine light),” https://www.youtube.com/watch?v=5tQm28K32SQ, 2016.

[41] A. Mueller and T. Lothspeich, “Plug-and-secure communication for CAN,” *CAN Newsletter*, pp. 10–14, 2015.