Error Handling of In-vehicle Networks Makes Them Vulnerable

Kyong-Tak Cho and Kang G. Shin
The University of Michigan
{ktcho,kgshin}@umich.edu

ABSTRACT

Contemporary vehicles are getting equipped with an increasing number of Electronic Control Units (ECUs) and wireless connectivities. Although these have enhanced vehicle safety and efficiency, they are accompanied with new vulnerabilities. In this paper, we unveil a new important vulnerability applicable to several in-vehicle networks including Control Area Network (CAN), the *de facto* standard in-vehicle network protocol. Specifically, we propose a new type of Denial-of-Service (DoS), called the *bus-off attack*, which exploits the error-handling scheme of in-vehicle networks to disconnect or shut down good/uncompromised ECUs. This is an important attack that must be thwarted, since the attack, once an ECU is compromised, is easy to be mounted on safety-critical ECUs while its prevention is very difficult. In addition to the discovery of this new vulnerability, we analyze its feasibility using actual in-vehicle network traffic, and demonstrate the attack on a CAN bus prototype as well as on two *real* vehicles. Based on our analysis and experimental results, we also propose and evaluate a mechanism to detect and prevent the bus-off attack.

1. INTRODUCTION

New security breaches in vehicles are emerging due to software-driven Electronic Control Units (ECUs) and wireless connectivities of modern vehicles. These trends have introduced more remote surfaces/endpoints that an adversary can exploit and, in the worst case, control the vehicle remotely. Researchers have demonstrated how vulnerabilities in remote endpoints can be exploited to compromise an ECU, access the in-vehicle network, and then control vehicle maneuvers [5, 10, 12]. In 2015, researchers were able to compromise and remotely kill a Jeep Cherokee running on a highway [13], which triggered a recall of 1.4 million vehicles. Such a reality of vehicle cyber attacks has made vehicle security one of the most critical issues to be addressed.

To detect and prevent vehicle cyber attacks, various types of security solutions, such as Message Authentication Code (MAC) and Intrusion Detection Systems (IDSs) for in-vehicle networks — akin to those in the Internet security — have been proposed [8, 14, 17, 21]. These solutions provide a certain level of security, but there still remain critical, uncovered vulnerabilities specific to the automotive domain.

We propose a new type of Denial-of-Service (DoS) attack called the *bus-off attack* which, ironically, exploits the error-handling scheme of in-vehicle networks. That is, their fault-confinement mechanism — which has been considered as one of their major advantages in providing fault-tolerance and robustness — is used as an attack vector. The attacker periodically injects attack messages to the in-vehicle network, deceives an uncompromised ECU into thinking it is defective, and eventually forces itself or even the whole network to shut down. In addition to its severe consequences, the following unique characteristics of the proposed bus-off attack differentiate itself from previously known attacks and make it a critical threat which must be countered.

1. The bus-off attack is easy to mount since the attacker is not required to reverse-engineer messages (i.e., figure out the meaning/purpose of messages) or their checksum algorithms for launching it. Thus, the attacker can easily mount the attack on various vehicles regardless of their manufacturer or model. This is in sharp contrast to previously demonstrated attacks, which required painstaking reverse-engineering procedures in order to take control of a vehicle [10, 12, 13].
2. As the symptoms of the bus-off attack resemble those of system errors such as improper termination [4], bit flip [22], and bit drop [15], it deceives state-of-the-art IDSs to think the network is erroneous, while it is actually under attack.
3. Although MACs for in-vehicle network messages may thwart most of the previously known attacks, they cannot prevent the proposed bus-off attack since it nullifies their functionalities. That is, not only contemporary insecure in-vehicle networks but also prospective *security-enhanced* ones will still be vulnerable to the bus-off attack.
4. Since the attack relies solely on low-level safety features of in-vehicle networks, it is independent of actual implementation subtleties of different ECUs.

Among the various in-vehicle network protocols, we focus on the vulnerability of the Control Area Network (CAN) protocol, which is the *de facto* standard for in-vehicle networks. The feasibility of the bus-off attack is, however, not limited to CAN but applicable to other in-vehicle network protocols, which are discussed further in Section 8. In this paper, we will primarily focus on *what* an adversary can do with a compromised ECU, rather than *how* the ECU was compromised in the first place, which has been covered well elsewhere [5, 7, 10, 12, 13].

This paper makes the following main contributions:

- Discovery of a new Denial-of-Service threat — bus-off attack — on in-vehicle networks which exploits their error-handling mechanism as an attack vector;

CCS'16, October 24-28, 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2976749.2978302

- Analysis and characterization of real CAN bus traffic, and the proof of the practicability of the bus-off attack;
- Implementation and demonstration of the bus-off attack on a CAN bus prototype and on two real vehicles; and
- Development and evaluation of a countermeasure that can detect and prevent the bus-off attack.

The rest of the paper is organized as follows. Section 2 provides the required background on CAN, and Section 3 details the proposed bus-off attack. Section 4 discusses the feasibility of the attack and Section 5 evaluates the attack on a CAN bus prototype and real vehicles. Section 6 discusses the limitations of state-of-the-art solutions in preventing the bus-off attack, and Section 7 details a new defense mechanism against it. Section 8 discusses further the severity of the bus-off attack as well as its applicability to other in-vehicle networks. Finally, we conclude the paper in Section 9.

2. PRIMER ON CAN

For completeness, we first review the main features of CAN related to the proposed attack.

2.1 CAN Frames

CAN interconnects ECUs/nodes through a message broadcast bus. Each node broadcasts periodic (and occasionally sporadic) data frames on the CAN bus to provide retrieved data. The transmitted data is received by one or more nodes on the bus and then utilized for maintaining data consistency and for vehicle control decisions.

Frame format. Each CAN frame is basically a sequence of dominant (0) and recessive (1) bits, and belongs to one of four different types: *data* frame which is used for sending retrieved data; *remote* frame for requesting transmission of a specified message; *error* frame used to indicate detected errors via error flags; and *overload* frame to inject delay between frames. Fig. 1 shows the base format of a CAN data frame. A data frame can carry up to 8 bytes of data, the length of which is specified in the 4-bit Data Length Code (DLC). For most passenger cars, a 1-byte checksum of each message is contained in the last byte of its data field [13]. Although the checksum is not part of the CAN specification, car manufacturers implement it using their own algorithms to provide an additional layer of protection to the CAN Cyclic Redundancy Check (CRC). Consecutive transmissions of CAN frames are separated by a 3-bit Inter-frame Space (IFS).

Message ID. Instead of containing the transmitter/receiver address, a CAN frame contains a unique ID, which represents its priority and meaning. For example, a frame containing wheel speed values might have ID=0x01 and frame containing battery temperature values might have ID=0x20. Only one ECU is assigned to transmit a given ID at a time, and the ID values are defined to be distinct from each other by the manufacturer. The base frame format has an 11-bit ID, whereas an extended format has a 29-bit ID. Since the use of base format is much more prevalent, we focus on the base format in this paper. Note, however, that the attack model proposed in this paper is not dependent on the type of format.

2.2 Arbitration

Once the CAN bus is detected idle, a node with data to transmit, starts its frame transmission (Tx) by issuing a Start-of-Frame (SOF). SOF provides *hard* synchronization between ECUs to make bitwise transmission and reception feasible. At that time, one or more other nodes may also have buffered data to transmit, and may thus concurrently access the bus. In such a case, the CAN protocol resolves the access contention via *arbitration*.

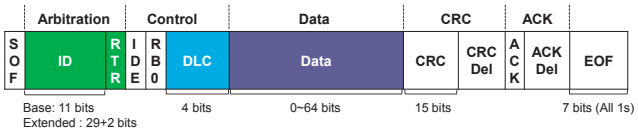


Figure 1: Format of a CAN data frame.

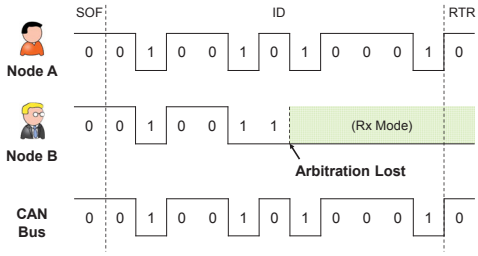


Figure 2: Example of CAN arbitration phase.

During transmission, each node sends its frame one bit at a time and monitors the actual output on the CAN bus. In the arbitration phase, since frame IDs are unique and the CAN bus logically behaves as a wired-AND gate (e.g., 0 & 1 = 0), some contending nodes would see a dominant (0) bit even though it has transmitted a recessive (1) bit. In such a case, they lose arbitration, withdraw from bus contention, and switch to receiver mode. In the end, only *one* arbitration-winner node is allowed to continuously access the bus for data transmission. This process enables higher-priority frames (i.e., lower IDs) to be transmitted before lower-priority ones. Once the arbitration winner has completed transmission of its frame ending with an End-of-Frame (EOF), after a 3-bit time of IFS, the bus becomes free again for access, i.e., idle. At that time, nodes that have buffered data or had previously lost arbitration, start another round of arbitration for access. For completeness, we illustrate in Fig. 2 how arbitration is done in CAN.

2.3 Error Handling

Error handling is built in the CAN protocol and is important for its fault-tolerance. It aims to detect errors in CAN frames and enables ECUs to take appropriate actions, such as discarding a frame, retransmitting a frame, and raising error flags. The CAN protocol defines no less than 5 different ways of detecting errors [3].

- Bit Error: Every transmitter compares its transmitted bit with the output bit on the CAN bus. If the two are different, a *bit error* has occurred, except during arbitration.
- Stuff Error: After every five consecutive bits of the same polarity, an opposite polarity bit is stuffed for maintaining *soft* synchronization. Violation of this incurs a *stuff error*.
- CRC Error: If the calculated CRC is different from the received CRC, a *CRC error* is raised.
- Form Error: If the fixed-form bit fields (e.g., CRC delimiter, ACK delimiter, EOF, IFS) contain at least one illegal bit, a *form error* has incurred.
- ACK Error: When a node transmits a message, any node that has received it issues a dominant bit in the ACK slot. If none replies, an *ACK error* is raised.

Error counters. For any detected errors, the perceived node transmits an error frame on the bus and increases one of the two error counters it maintains: *Transmit Error Counter* (TEC) and *Receive Error Counter* (REC). There are several rules governing the

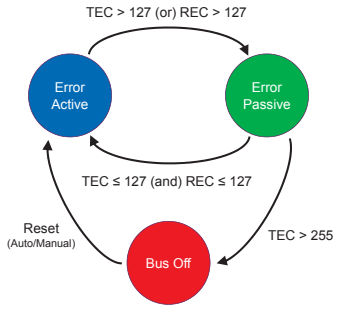


Figure 3: State diagram of fault confinement in CAN.

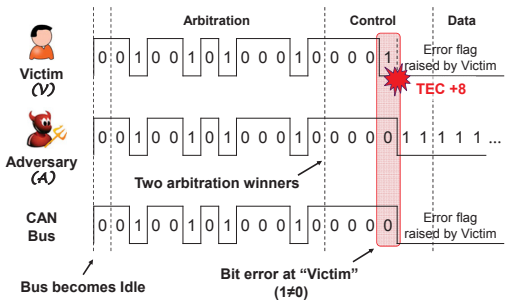


Figure 4: Example of a bus-off attack.

increase/decrease of these counters, but in essence, a node that detects an error during transmission increases TEC by 8, whereas if perceived during reception, REC is increased by 1 [3]. Moreover, for any error-free transmission and reception, TEC and REC are decreased by 1, respectively.

Fault confinement. To confine serious errors disrupting bus communications, each ECU's error mode is managed as in Fig. 3. All ECUs start in an *Error-active* mode and switch between different modes depending on their TEC and REC values. When TEC or REC exceeds 127 due to consecutive errors, the node becomes *Error-passive*, and only returns to its initial state when both go below 128. When TEC exceeds the limit of 255, the corresponding ECU — which must have triggered many transmit errors — enters the *Bus-off* mode. Upon entering this mode, to protect the CAN bus from continuously being distracted, the error-causing ECU is forced to shut down and *not* participate in sending/receiving data on the CAN bus at all. It can be restored back to its original error-active mode, either automatically or manually. However, since bus-off is usually an indication of serious network errors and may not be fixed by mere automatic re-initialization of the CAN controller, a user-intervened recovery or even a controlled shut-down of the entire system is recommended [18].

Bus-off recovery. The reasons for such different options are 1) the bus-off recovery mechanism depends on the software stack being used, i.e., how the system is designed by the manufacturer, and 2) ECUs have different ASILs (Automotive Safety Integrity Levels). Since the bus-off is a serious problem, in most cases, vehicle systems are designed to first enter a "limp home" mode (when it occurs) with all their parameters set to pre-set values, and thus run with reduced functionality (e.g., limited engine RPM). In this mode, warning lamps are lit up on the dashboard to alert the driver, and the vehicle runs only for some time before it is properly serviced by an OEM-authorized service center. Depending on the severity of the underlying issue, i.e., which ECU was shut down, a vehicle in the limp home mode will later be totally disabled.

Error flags. When an error is detected, the perceived node indicates to others on the bus via an error flag, which comes in two forms: active and passive. For any perceived errors, nodes that are in error-active mode issue an active error flag which consists of 6 *dominant* bits. So, the transmitted frame causes other nodes to violate the bit-stuffing rule, transmit their own error frame caused by the stuff error, and terminate any on-going transmissions or receptions.

For nodes that are in error-passive mode, they operate in the same way as error-active ones, except that they issue a passive error flag which consists of 6 *recessive* bits and have an 11 (not 3) bit-time of IFS if they were the transmitter of the previous message [3]. An error-passive node tries to signal its passive error flag until it actually observes 6 recessive bits on the bus, i.e., an indication of the error flag being properly sent. Since recessive bits are overwritten on the CAN bus by dominant bits, the thus-issued passive error flags may persist until the end of a frame.

3. ATTACK MODEL

We first discuss the adversary model under consideration, and then uncover a new vulnerability of in-vehicle networks.

3.1 Adversary Model

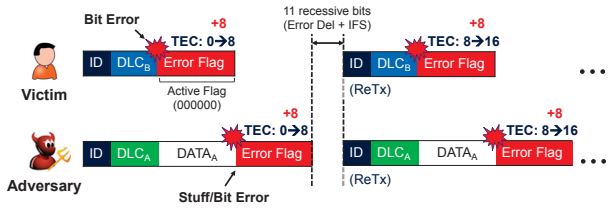
We consider an adversary whose objective is to shut down uncompromised (healthy) in-vehicle ECUs with a minimal number of message injections. Such an objective precludes other types of attacks (e.g., flooding) which, albeit their greater impact, require a large number of message injections and are thus easier to be detected. In Section 8, we will discuss more on such attacks, highlighting the severity of the bus-off attack. As in previously discussed attacks [5, 10, 12, 13], we assume that the adversary can physically/remotely compromise an in-vehicle ECU through numerous attack surfaces and means, and thus gain its control. In contrast, we do *not* require the adversary under consideration to reverse-engineer messages or checksums in order to achieve its goal of shutting down an ECU. Since the messages and the implemented checksum algorithms are different for different vehicle manufacturers and models, such reverse-engineering can be very painstaking, when the adversary wants to mount attacks on different vehicles.

Once an ECU is compromised, we consider the adversary to be capable of performing at least the following malicious actions. The adversary can *inject* any message with forged ID, DLC, and data on the bus as they are managed at user level. Also, since CAN is a broadcast bus, the adversary can *sniff* messages on CAN. Restrictions of message filters are detailed in Section 4.2. These are the *basic* capabilities of an adversary who has the control of a compromised ECU. Practicability of such an adversary model has already been proved and demonstrated in [5, 10, 12].

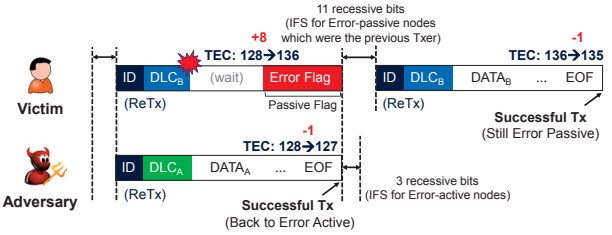
3.2 Bus-off Attack

We now introduce the *bus-off attack* which exploits the following feature of CAN: CAN's error handling automatically isolates defective or "misbehaving" ECUs — whose TEC>255 — into *bus-off* mode. Specifically, by iteratively injecting attack messages, the adversary coerces the TEC of an uncompromised/healthy victim ECU to continuously increase — deceiving it to think it is defective — and in the end, triggers the CAN fault confinement to force the victim or even the entire network to shut down.

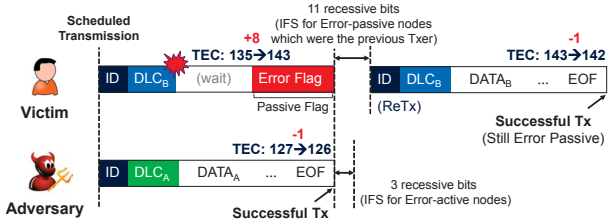
Increasing the victim's TEC. Suppose message $\mathbb{M}$ is periodically sent by some victim ECU $\mathbb{V}$. Then, an adversary $\mathbb{A}$ can succeed in a bus-off attack by injecting an attack message, which satisfies the following conditions.



(a) Phase 1 – Victim in error-active mode.



(b) Transition from Phase 1 to 2.



(c) Phase 2 – Victim in error-passive mode.

Figure 5: Two phases of bus-off attack.

C1. ID – *Same* ID as message $\mathbb{M}$;

C2. Timing – Transmitted at the *same* time as $\mathbb{M}$; and

C3. Contents – Having at least one bit position in which it is dominant (0), whereas it is recessive (1) in $\mathbb{M}$. All preceding bits should be the same as $\mathbb{M}$.

To describe the general attack model, for now we assume that the adversary can transmit a message that satisfies C1–C3. Details of how and why C1–C3 can be met will be discussed in Section 4. As shown in Fig. 4, when the bus becomes idle, not only the victim transmits message $\mathbb{M}$ but also the adversary transmits an attack message satisfying C1–C3. So, not one but two transmitters — $\mathbb{A}$ and $\mathbb{V}$ sending messages with identical IDs — win arbitration, and thus *concurrently* send their bit values of control, data, etc., on the bus. The two nodes' bitwise transmissions are synchronized in virtue of hard and soft synchronizations. Since the attack message meets C3, $\mathbb{V}$ sees an opposite polarity on the bus to the one it transmitted (this happens after an arbitration). As a result, the victim $\mathbb{V}$ experiences a bit error *forced* by $\mathbb{A}$, thus increasing its TEC by 8. By repeating this bus-off attack on the victim's messages, the adversary can make the victim's TEC to continuously increase, and force the victim to enter bus-off mode and disconnect from the bus. Although the victim is error-free in transmitting messages other than the targeted one, since TEC increases by 8 upon detection of each error but decreases only by 1 for each error-free transmission, an iterative bus-off attack rapidly increases the victim's TEC. The entire process of an iterative bus-off attack consists of the following two phases.

Phase 1 – Victim in error-active. Both adversary and victim nodes start in their default mode, error-active. After observing messages on the CAN bus, the adversary targets one of them for a bus-off attack. We refer to such a message as the *target message* and its transmitter as the victim. As mentioned earlier, the adversary then injects its attack message at the same time as the target message to increase the victim's TEC. Thus, as shown in Fig. 5(a), the victim experiences bit error, transmits an *active* error flag, and increases its TEC by 8. Since an active error flag consists of 6 consecutive dominant bits (i.e., 000000), either a stuff or bit error is triggered at the adversary node, and its TEC also increases by 8. After the error delimiter and IFS, the CAN controllers of the adversary and the victim automatically retransmit the Tx-failed messages again at the same time. So, the exact same bit error recurs until they both enter error-passive mode. What is significant about the attack in Phase 1 is that the adversary can coerce the victim to become error-passive with just one message injection.

Phase 1 to 2. After 16 (re)transmissions, as shown in Fig. 5(b), both the adversary and the victim become error-passive when their TEC=128. Again, for the retransmitted message, bit error occurs at the victim node. However, since the victim is now in error-passive mode, it attempts to deliver a *passive* error flag which consists of 6 recessive bits. At that time, since the adversary transmits its frame, the attempt to deliver the error flag will persist until the adversary's EOF. In contrast to Phase 1, the adversary node experiences no error and thus succeeds in transmitting its frame, whereas the victim will succeed later during its retransmission. In total, due to a bit error ($+8$) and a successful retransmission (-1), the victim's TEC changes $128 \rightarrow 136 \rightarrow 135$, whereas the adversary's changes $128 \rightarrow 127$. Accordingly, the adversary returns to error-active, while the victim remains error-passive. Up to this point, all but the first change in TEC are achieved via automatic retransmissions by the CAN controller, i.e., *the controller does it all for the attacker!*

Phase 2 – Victim in error-passive. Fig. 5(c) illustrates Phase 2 of the bus-off attack in which only the victim is error-passive. Once the scheduled interval of the target message has elapsed, the victim again transmits that message, and thus at the same time, the adversary re-injects its attack message. Since the victim is still in error-passive mode, as it was the case when transitioning from Phase 1 to 2, the adversary can decrease its TEC further by 1. On the other hand, the victim's TEC is again increased by 7 ($= +8 - 1$), thus keeping the victim in error-passive mode. In Phase 2, the adversary iterates this process for every periodically transmitted target message until the victim is eventually forced to bus off, i.e., TEC$>$255. This implies that the periodicities of the attack and the target messages are the same. As a result, the victim ECU becomes disconnected, and in the worst case, the entire network shuts down [18].

Although CAN messages' ID values do not contain information on their actual transmitters, their values and intervals together imply the messages' priority and safety-criticality. That is, if the attacker targets a message sent with high priority (i.e., a low ID value) and small message intervals, then the attacker would most likely disconnect a *safety-critical* ECU that sends important messages related to, for example, vehicle acceleration or braking.

Alternative bus-off attacks. The adversary may attempt to iterate this process for not only every periodic transmission but also every retransmission (as in Phase 1). However, as shown in Fig. 5(c), since error-passive nodes, which were the transmitter of the previous message, have a longer IFS than error-active nodes, the adversary cannot synchronize its injection timing with the victim's retransmission, thus failing to mount the attack.

Yang [23] reported that an uncovered fatal error can sometimes occur due to a misspecification of CAN; if a new frame is transmitted on the bus during the error delimiter subsequent to a passive error flag issued by an error-passive node, a form error incurs to that node, thus increasing its TEC. If the adversary were to exploit such an inherent specification error in CAN to mount a bus-off at-

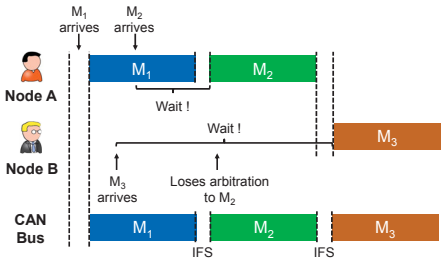


Figure 6: Example of preceded IDs.

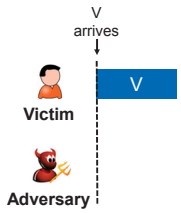
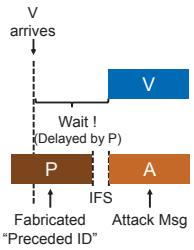


Figure 7: Fabricated preceded ID.



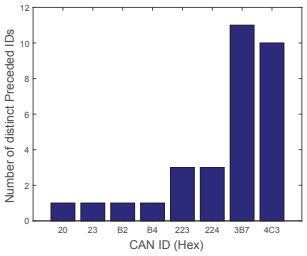


Figure 8: Distinct preceded IDs.

tack, it may be simpler to coerce the victim to bus off. However, the occurrence of that fatal error depends on what type of new frame is transmitted at that time as well as how high the bus load is. Note that it would be very difficult for an adversary to control such factors. That is, it is too restrictive for the attacker to achieve the bus-off attack in that way. By contrast, we propose and discuss a more general method for mounting the bus-off attack, i.e., succeeding the attack under any bus condition.

4. FEASIBILITY OF BUS-OFF ATTACK

To mount the bus-off attack, the adversary has to inject an attack message that satisfies conditions on ID, timing, and contents. We now discuss how and why each condition can be met, in the ascending order of difficulty.

4.1 Different Contents

To trigger a bit error at the victim and increase its TEC, the attack message must first satisfy C3: having at least one bit position in which its signal is dominant (0), whereas the victim's is recessive (1), and all preceding bits are identical. Since the bus-off attack also requires the attack and target messages to have the same ID (i.e., C1 – identical arbitration fields), the mismatch in C3 must occur in either the control or the data field. Note that this is infeasible in other fields such as CRC and ACK, because they are determined by the CAN controller, not by the user/adversary. Since CAN messages normally have DLC set to at least 1 and non-zero data values, one simple but most definite way for the adversary to cause a mismatch is to set the attack message's DLC or data values to all 0s. Also, given that DLC for each CAN ID is usually constant over time, the attacker can learn the value and set its attack message's DLC accordingly. This way, the adversary can satisfy C3.

4.2 Same ID

The next difficult task is to meet C1, which requires the attack message to have the same ID as the target. That is, the adversary must know in advance the ID used by the target message. The fact that favors the adversary is that CAN is a broadcast bus system. However, each ECU cannot acquire the IDs of all received messages except those passed through its message filter. We distinguish a *received* message from an *accepted* message, depending on whether it passed through the filter and then arrived at a user-level application. Since an adversary can read contents only from accepted messages, meeting C1 depends on how the filter is set at the compromised ECU.

Empty message filter. Some ECUs in vehicles have almost, if not completely, empty message filters so that they can receive, accept, and process almost all messages on the bus, thus making it trivial to satisfy C1. A typical example of these ECUs is the telematic unit, which has to operate in that way to provide a broad range of features, e.g., remote diagnostics, anti-theft. Another interesting aspect of this ECU is that, from a security viewpoint, it is regarded as one of the most vulnerable ECUs due to its wide range of external/remote attack surfaces. Note that several researchers have already shown the practicability of compromising the telematic unit [5, 7, 10, 12]. So, this implies that an adversary can compromise some empty filter ECUs more easily than those with non-empty filters, thus satisfying C1.

Non-empty message filter. Although the message filter of a compromised ECU is preset to accept messages with only a few different IDs, it does not restrict the adversary in attacking them. Furthermore, by directly modifying the message filter, the adversary can also mount the attack on messages that would usually have been filtered out. The two most common CAN controllers — Microchip MCP2515 and NXP SJA1000 — both allow modification and disabling of message filters through software commands, when the ECU is in configuration mode [1, 2]. For ECUs with the Microchip MCP2515 CAN controller, the configuration mode can be entered not only upon power-up or reset but also via user instructions through the Serial Peripheral Interface (SPI). Through the SPI, it is also possible for the user to read/write the CAN controller registers, including the filter register [1]. Thus, such user-level features for configuring the CAN controller allow attackers to easily enter configuration mode via software commands, and modify/disable the message filters, thus satisfying C1.

4.3 Tx Synchronization

Even though the adversary knows the ID and contents to use for his attack message, he should also know exactly *when* to send it. Unless the adversary is capable of sending the attack message at the same time as the target message, not only once but iteratively, he would fail to cause a bit error, or increase the victim's TEC.

Difficulties in synchronizing the Tx timing. C2 requires the transmission of the attack and the target messages to be synchronized with less than a bit resolution. If the attack timing is wrong by even one bit, there won't be two arbitration winners and the attack would thus fail. For synchronizing the timing of its transmission, the adversary may utilize the fact that CAN messages are usually sent at fixed intervals. For example, once the adversary learns that the target message is sent every T ms, it can attempt to transmit its attack message when T ms has elapsed since the target's last transmission. However, such an approach would be inaccurate due to jitters. Since jitters make the actual message periodicities deviate from their preset values [9], albeit leveraging message periodicity to fulfill C2, the attacker would have difficulties in synchronizing the transmission of its attack message with the target's.

Preceded ID. In order to overcome these difficulties, the adversary can exploit another fact of CAN: nodes, which have either lost arbitration or had new messages buffered while the bus was busy, attempt to transmit their messages as soon as the bus becomes idle.

We define a *preceded ID* of $\mathbb{M}$ as the ID of the message that has

completed its transmission right before the start of $\mathbb{M}$'s. Consider an example where node A transmits messages with ID=$\mathbb{M}_1$, $\mathbb{M}_2$, and node B transmits a message with ID=$\mathbb{M}_3$ which has the lowest priority among them. As shown in Fig. 6, if these messages are arriving and being queued at the depicted times, $\mathbb{M}_1$ and $\mathbb{M}_2$ would be the preceded IDs of $\mathbb{M}_2$ and $\mathbb{M}_3$, respectively, with only a 3-bit IFS separating the corresponding message pairs. In other words, the transmissions of $\mathbb{M}_2$ and $\mathbb{M}_3$ are forced to be buffered until their preceded ID messages have been transmitted on the bus. Since message priorities and periodicities do not change, such a feature implies that one particular CAN message may always be followed by another specific message, i.e., there is a *unique* preceded ID for that specified one. As an example, if the periodicities of their transmissions are either same or integer multiples (e.g., 5ms for $\mathbb{M}_1$, $\mathbb{M}_2$, and 10ms for $\mathbb{M}_3$), then $\mathbb{M}_2$ would always be the preceded ID of $\mathbb{M}_3$, i.e., be the *unique preceded ID*. Hence, regardless of jitter, the exact timing of message transmissions becomes rather predictable and even determinative: 3 bit-time after the preceded ID's completion.

Bus-off attack by exploiting preceded IDs. In the above example, to attack $\mathbb{M}_3$, the adversary can monitor the CAN bus, learn its preceded ID of $\mathbb{M}_2$ or even $\mathbb{M}_2$'s preceded ID of $\mathbb{M}_1$, and buffer an attack message with ID=$\mathbb{M}_3$ when receiving one of them. Then, its CAN controller would always transmit the attack message after $\mathbb{M}_2$'s transmission (i.e., concurrently with the target message), and the adversary will thus succeed in the bus-off attack. Likewise, the adversary can target $\mathbb{M}_2$ by buffering its attack message with ID=$\mathbb{M}_2$, as soon as it receives its preceded ID of $\mathbb{M}_1$. If the preceded ID is unique, then the bus-off attack can be iterated for its every reception and thus consecutively increase the victim's TEC.

Even though the target message does not have such a preceded ID, an adversary can fabricate it in order to synchronize the timing and thus succeed in mounting the attack. Consider an example shown in Fig. 7 where a victim node periodically transmits message $\mathbb{V}$, which has no preceded IDs. In such a case, just before the transmission of $\mathbb{V}$, the adversary can inject some message $\mathbb{P}$ and an attack message $\mathbb{A}$, sequentially. Hence, $\mathbb{V}$'s transmission gets delayed until the completion of $\mathbb{P}$, i.e., the adversary fabricates $\mathbb{P}$ as the preceded ID of $\mathbb{V}$, and thus the attack message is synchronized with its target. Our evaluation results will later demonstrate the efficiency of bus-off attacks based on the above approaches.

4.4 Preceded IDs in Actual CAN Traffic

The key point in meeting C2 and succeeding in the bus-off attack is leveraging the preceded IDs of the target message. Depending on the configuration and scheduling of messages, some target messages may (or may not) have a preceded ID. We first consider the case in which the adversary is targeting messages with *genuine* (unfabricated) preceded IDs. Hence, it is essential to verify their existence in actual CAN traffic as well as usefulness in satisfying C2. That is, the following questions should be answered:

- **Existence** – Are there such preceded IDs in real in-vehicle network traffic?
- **Uniqueness** – If yes, how many distinct preceded IDs are there for a specified message?
- **Pattern** – If more than one, are there any patterns in the preceded IDs which can be utilized?

We answer these questions via an analysis of actual CAN traffic data. We use CAN data that was recorded from a 2010 Toyota Camry by Ruth *et al.* [20]. During a 30-minute drive, the data was logged by a Gryphon S3 and Hercules software [20]. According to the logged data, there were 42 distinct messages transmitted on the CAN bus: 39 of them sent periodically at intervals ranging from 10ms to 5 secs, and 3 of them sent sporadically.

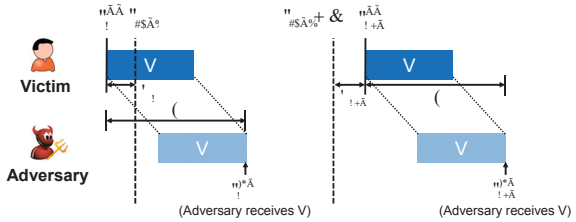


Figure 9: Timing of preceded ID message injection.

Existence and uniqueness. Of these, we first identify the ones of interest to us: messages that are *always* sent right after another's completion, i.e., have preceded IDs (e.g., $\mathbb{M}_2$ or $\mathbb{M}_3$ in Fig. 6). Of the 39 periodic messages seen on the bus, we were able to find 8 of such type. Fig. 8 shows the number of distinct preceded IDs of the ones labeled in the x-axis. For example, message 0xB2 had a unique preceded ID 0xB0, i.e., 0xB2 always followed 0xB0, where both were sent every 10ms. On the other hand, message 0x3B7 had 11 different kinds of those IDs. The result showing that 10% of the periodic messages have a unique preceded ID answers the question of their existence as well as uniqueness in actual CAN traffic, and thus implies that the bus-off attack exploiting genuine preceded IDs is indeed feasible in actual vehicles. Since in-vehicle messages have fixed priorities, are sent periodically, and some have to be sent consecutively by an ECU (e.g., two messages containing front and rear wheel speed values), we believe preceded IDs are prevalent in all other types of passenger cars as well.

Patterns in preceded IDs. Another interesting aspect of the CAN bus traffic, which caught our attention was that there were notable *patterns* in the preceded IDs. As shown in Fig. 8, message 0x223 with periodicity 30ms had 3 distinct preceded IDs, meaning that observing those IDs may not help determine the transmission timing of 0x223. However, we were able to extract an interesting pattern in their transmission: the $6n$-th transmission of 0x20 was the unique preceded ID of the $2n$-th transmission of 0x223, where n is an integer. That is, even though a unique preceded ID was not observable for every transmission, it was for every n-fold transmission, i.e., there exists a pattern in preceded IDs. Therefore, by observing the CAN bus traffic, acquiring knowledge of genuine preceded IDs, and thus meeting not only C1, C3 but also C2, the adversary can succeed in mounting bus-off attack.

4.5 Fabrication of Preceded IDs

Even though the targeted messages do not have a preceded ID, as shown in Fig. 7, the adversary can fabricate it to meet C2. Therefore, we will henceforth refer preceded ID messages to ones which are fabricated by the adversary. The injection timing, quantity, and the contents of the preceded ID message are important for the adversary to mount a bus-off attack with preceded IDs.

Injection timing. To succeed in the bus-off attack via fabrication of preceded IDs, it is essential for the adversary to inject that fabricated message right before the target message. In other words, the adversary is required to estimate when the target message would be transmitted on the bus. Although most in-vehicle messages have fixed periodicity, randomness incurred from jitters makes such estimation rather difficult. As shown in Fig. 9, consider a target message $\mathbb{V}$ with periodicity of T, which is expected to be transmitted at times t_{orig}, $t_{orig}+T$, and thereafter. Note that T is a predefined and constant value for periodic messages. However, due to the jitters of J_n and J_{n+1} — caused by variations in

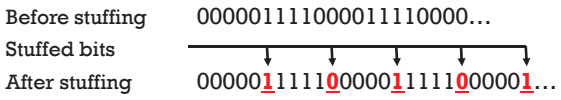


Figure 10: Maximizing Tx duration via bit-stuffing.

the transmitter node's clock drift, task scheduling, execution time, etc. [9] — the victim's messages are transmitted on the bus at times t_n^{vic} and t_{n+1}^{vic}, where n is the sequence index. From the adversary's perspective, due to an incurred delay of $\mathbb{D}$ from transmission and reception, it receives message $\mathbb{V}$ at times t_n^{adv} and t_{n+1}^{adv}. Note that $\mathbb{D}$ includes delays for message transmission, propagation, and processing. Since the number of bits in a certain message is almost a constant and the bit timing of CAN already takes into account of the signal propagation on the bus, without loss of generality, we assume $\mathbb{D}$ to be constant for a given message $\mathbb{V}$ [3, 16].

Thus, the only remaining randomness in the timing of message transmission is jitter (e.g., J_n). Jitter is known to follow a Gaussian distribution due to randomness in thermal noise, which also follows a Gaussian, and the Central Limit Theorem, i.e., composite effects of many uncorrelated noise sources approach a Gaussian distribution [9]. So, we can consider J_n and J_{n+1} as outcomes of a Gaussian random variable $J \sim \mathbb{N}(0, \sigma_v^2)$. Thus, the times when the adversary receives $\mathbb{V}$ can be expressed as:

$$\begin{aligned} t_n^{adv} &= t_n^{vic} + \mathbb{D} = t_{orig} + J_n + \mathbb{D} \\ t_{n+1}^{adv} &= t_{n+1}^{vic} + \mathbb{D} = t_{orig} + T + J_{n+1} + \mathbb{D}, \end{aligned} \tag{1}$$

where $J_n < 0$ and $J_{n+1} > 0$ in Fig. 9. Then

$$t_{n+1}^{vic} = t_n^{adv} + T - \mathbb{D} + J_{n+1} - J_n = t_n^{adv} + T - \mathbb{D} + J^*, \tag{2}$$

where $J^* \sim \mathbb{N}(0, 2\sigma_v^2)$ since its outcomes are $J_{n+1} - J_n$. Note that in Eq. (2), J^* is the only random variable whereas others are either constant or measurable by the adversary. Such an equation shows that the adversary can indeed obtain an approximate estimation of when the victim would transmit its message, i.e., the target message, at the *next* sequence. As shown in Fig. 7, for the fabrication of preceded IDs to be effective, the adversary has to 1) start transmission of its preceded ID message(s) before the target and 2,3) hold the CAN bus, i.e., make the bus busy, until it becomes sure that the attack and target messages would synchronize. That is, the adversary must meet the following three conditions:

$$\begin{aligned} &1)\ t_{fab} < \min(t_{n+1}^{vic}) = t_n^{adv} + T - \mathbb{D} + \min(J^*) \\ &2)\ t_{fab} + \mathbb{H} > \max(t_{n+1}^{vic}) = t_n^{adv} + T - \mathbb{D} + \max(J^*) \\ &3)\ \mathbb{H} = \kappa\mathbb{F} > \max(J^*) - \min(J^*), \end{aligned} \tag{3}$$

where t_{fab} denotes when the adversary starts to inject its fabricated preceded ID message(s). Moreover, $\mathbb{H}$ denotes the duration of the adversary holding the bus, which is equivalent to κ preceded ID messages each sent for a duration of $\mathbb{F}$. Since J^* is a bounded Gaussian random variable, the boundaries can be approximated as $|max(J^*)| = |min(J^*)| \simeq \mathbb{I}\sqrt{2}\sigma_v$, where σ_v is measurable, and $\mathbb{I}$ an attack parameter. Since J^* is Gaussian, setting $\mathbb{I} = 3$ would provide a 99.73% confidence and $\mathbb{I} = 4$ a 99.99% confidence. In total, to fully exploit the fabricated preceded IDs, the adversary has to start injecting them prior to $t_n^{adv} + T - \mathbb{D} - \mathbb{I}\sqrt{2}\sigma_v$. Note that the adversary should not lower t_{fab} beyond $\max(t_{n+1}^{vic}) - \mathbb{H}$, which can be set once $\mathbb{H}$ is determined.

Number of messages. Once satisfying 1) in Eq. (3), the adversary has to satisfy 3) — occupy the bus at least for the duration of $\max(J^*) - \min(J^*) = 2\sqrt{2}\mathbb{I}\sigma_v$, which can be met via $\kappa\,(\geq 1)$ injections of preceded ID messages. Since the adversary's objective is to mount the bus-off attack with a minimal number of injections, κ should be kept to minimum by maximizing $\mathbb{F}$. To maximize $\mathbb{F}$, i.e., the duration of its preceded ID message occupying the bus, the adversary can exploit the bit-stuffing rule of CAN: after every 5 consecutive bits of the same polarity (e.g., 00000), an opposite polarity bit is stuffed. By fabricating its preceded ID message with DLC=8 and the data field as shown in Fig. 10, the adversary can maximize the number of stuffed bits and thus $\mathbb{F}$ to at least $\mathbb{F}^* = (8L + 44 + \lfloor 8L/4 \rfloor)/S_{bus} = 124/S_{bus}$, where 44 denotes the number of bits exterior to the data field, L the DLC=8, and S_{bus} the bus speed. Note that at least $2L$ bits are added to the fabricated message according to the CAN's bit-stuffing rule. Hence, if we consider a CAN bus with $S_{bus} = 500$Kbps, using a single injected preceded ID message, the adversary can take control of the bus for at least 0.248ms. Such analyses suggest that for targeting a message with a jitter deviation of σ_v, the adequate number of preceded ID messages can be expressed as $\kappa = \left\lceil \frac{\max(J^*) - \min(J^*)}{\mathbb{F}^*} \right\rceil = \left\lceil \frac{2\sqrt{2}\mathbb{I}\sigma_v S_{bus}}{124} \right\rceil$. For example, if $\sigma_v = 0.025$ms and $S_{bus} = 500$Kbps, the adversary is only required to inject $\kappa = \lceil 0.8554 \rceil = 1$ preceded ID message with $\mathbb{I} = 3$, i.e., 99.73% confidence. To ensure the effectiveness of the fabricated preceded IDs with a near-perfect confidence, the adversary can set $\mathbb{I} = 4$ and thus inject $\kappa = \lceil 1.1405 \rceil = 2$ messages at time t_{fab}. Our evaluation will later show that one injection of fabricated ID messages ($\kappa = 1$) can be sufficient for a bus-off attack.

Contents of the preceded ID message. Other than the control and data fields, the adversary also has to carefully decide which ID to use for fabricating the preceded ID messages. If only one preceded ID message is to be used for the attack, the adversary can exploit the next seemingly free ID. To be as elusive as possible, the ID value can be changed for each attempt of attack and be chosen from those least frequently sent on the bus. If two preceded ID messages are to be used, the adversary can similarly inject the first one with any free ID but should inject the second one with an ID having higher priority (smaller value) than the target.

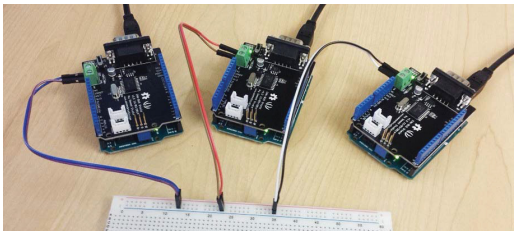


Figure 11: CAN bus prototype.

5. EVALUATION

We now evaluate the feasibility of the proposed bus-off attack on a CAN bus prototype and two *real* vehicles. For in-depth analyses of the bus-off attack, we first evaluate the attack on the CAN bus prototype, and then extend the evaluation to real vehicles.

5.1 Bus-off Attack

As shown in Fig. 11, we configured a CAN prototype in which all 3 nodes were connected to each other via a 2-wire bus. Each node consists of an Arduino UNO board and a SeeedStudio CAN bus shield, which is composed of a Microchip MCP2515 CAN controller with SPI interface, MCP2551 CAN transceiver, and a 120Ω terminal resistor to provide CAN bus communication capabilities.

Evaluation setup. The CAN bus prototype was set up to operate at 500Kbps as in typical in-vehicle CAN buses. Three interconnected nodes were each programmed to replicate the scenario

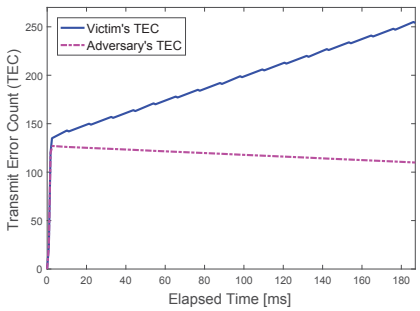


Figure 12: TECs during a bus-off attack.

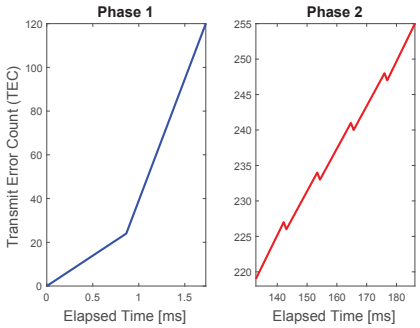


Figure 13: Victim's TEC during Phase 1 and 2.

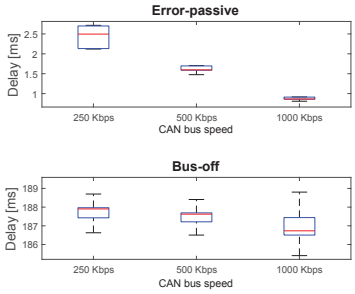


Figure 14: Delays of bus-off attack under different bus speeds.

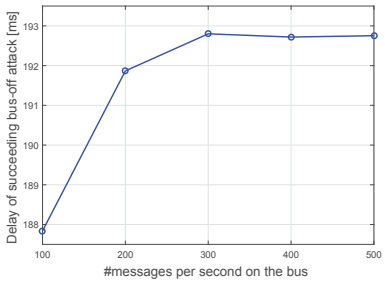


Figure 15: Delays of bus-off attack under different bus loads.

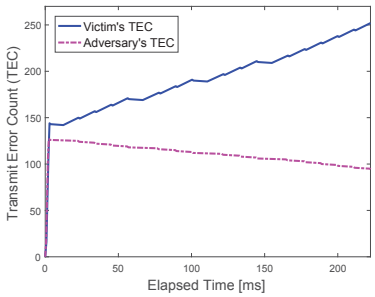


Figure 16: Changes in TECs when the preceded ID is fabricated.

shown in Fig. 6, which is also commonly seen in actual CAN bus traffic: every 10ms, node A was programmed to send two consecutive messages with ID=0x07 and 0x09, and B to send one message with a lower priority of ID=0x11. Node B buffered its message when a message with ID=0x07 was received. Note that 0x07 and 0x09 become the genuine preceded IDs of 0x09 and 0x11, respectively. We will later extend this evaluation of the bus-off attack to the case without assuming the availability of genuine preceded IDs. Last, but not least, the third node was programmed as an attacker which learns any preceded IDs on the bus and repetitively launches the bus-off attack. In our evaluation, message 0x11 from B was set as the attacker's target message. Hence, the attack message was set up to have the same ID=0x11 but with a different DLC (=0). Since the targeted message had a period of 10ms, the bus-off attack was repeated at the same time interval. For every message transmission/reception, the Transmit Error Count (TEC) of each node was read from its CAN controller register. The entire procedure of an iterative bus-off attack was re-initiated once a node entered the bus-off mode, and was examined 1,000 times.

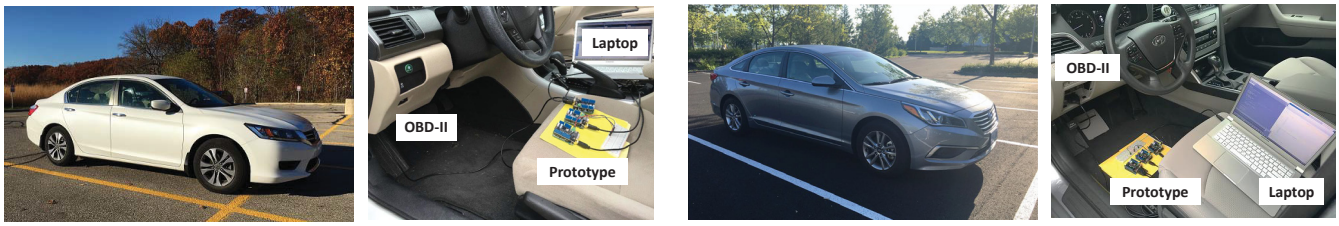
Changes in TEC during a bus-off attack. Fig. 12 shows how the TECs of the victim and the adversary change during an iterative bus-off attack. All 1,000 examinations showed near identical changes as shown in Fig. 12. In the initial stage of the attack, there was a steep rise in the TEC of both nodes. This is because in Phase 1, with just one attack message, bit errors incurred for not only the initial transmission but also all subsequent retransmissions as depicted in Fig. 5(a). Once the victim became error-passive, i.e., TEC$>$127, the attack entered its second phase. Here, with successful message transmissions, the adversary was able to recover back to, and remain as error-active. On the other hand, the victim experienced iterative bit errors during its transmissions, and eventually entered the bus-off mode when its TEC exceeded 255.

Properties of attack Phase 2. Fig. 13 shows a magnified plot of the changes in the victim's TEC during the attack. In Phase 1, TEC monotonically increased due to the errors in all (re)transmissions. On the other hand, once the attack entered Phase 2, the difference in error mode lets the victim succeed in its transmission only after experiencing a bit error. Thus, whenever an attack message was injected into the bus by the attacker, the victim's TEC was first increased by 8 and then immediately decreased by 1. The net TEC increase of 7 each time eventually forced the victim to be disconnected from the bus. These results confirm the properties of Phase 2 discussed in Section 3.2.

5.2 Attack Under Different Bus Conditions

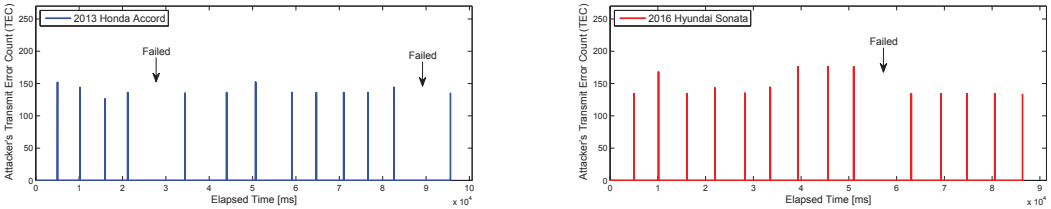
The bit-rate of the CAN bus can vary from 125Kbps to 1Mbps, depending on its purpose, and its bus load can also vary with time. Thus, in order to examine the practicability of a bus-off attack under different bus conditions, we conducted the same experiment in Section 5.1 1,000 times, while varying the speed and load of the bus. Each time we measured the average delays of the bus-off attack that forces a victim to enter error-passive and bus-off modes, as they represent the following metrics: (1) the required number of attack messages, and (2) the probability of the attack's success. If there was at least one attempt in which the attack failed, then the maximum deviation in delays would be at least the inter-attack interval.

Different bus speeds. Fig. 14 shows box plots of the average delays of the bus-off attack in coercing the victim to become error-passive and bus-off. The bus speed was varied from 250Kbps to its maximum of 1Mbps. As shown in Fig. 14 (top), for all bus speeds, it took much less than 10ms for the adversary to coerce the victim to become error-passive. Since the attack message was injected every 10ms, this implies that only a single injection of the attack message was required. Also, observing that the maximum deviation was less than 10ms, all attempted bus-off attacks succeeded, irre-



(a) Communication with ECUs in a 2013 Honda Accord. (b) Communication with ECUs in a 2016 Hyundai Sonata.

Figure 17: Bus-off attack experiments on (a) Honda Accord 2013 and (b) Hyundai Sonata 2016.



(a) 2013 Honda Accord. (b) 2016 Hyundai Sonata.

Figure 18: Transmit error counts of ECUs in the two real vehicles are increased via bus-off attack.

spective of the bus speed. The attacker was able to make the victim enter error-passive mode faster with the increase of bus speed, because a higher bus speed enabled frame (re)transmissions to complete more quickly. Fig. 14 (bottom) shows the total delay of the victim eventually becoming bus-off at different bus speeds. For all settings, the maximum deviation was less than 10ms, again implying a 100% success probability of the attack.

Different bus loads. Not only the speed but also the load of the bus was varied in evaluating the attack. To generate different bus loads, the node that was neither the adversary nor the victim injected 100∼500 messages per second. Their IDs were randomly chosen among the set of unused ID values, and their DLCs were set randomly between 1 and 8. Fig. 15 shows the average total delay of coercing the victim node to eventually bus off under the given bus loads. As the bus load increases, the overall delay is shown to rise, because some of the randomly injected messages won arbitration over the target and attack messages and thus delayed their transmission. Note, however, that since they both had the same IDs, they both won/lost the arbitration. Therefore, in all 1000 examinations with different bus loads, all trials of bus-off attack succeeded regardless of the bus load.

5.3 Periodicity vs. Preceded ID

To mount a bus-off attack, the attack message has to satisfy conditions C1–C3. Of these, C2 is the most difficult to meet, but can be satisfied by exploiting either of the following three scenarios:

- **Periodicity** – measure the Tx interval of the target message and exploit it for synchronizing the transmission timing.
- **Genuine ID** – assuming genuine preceded IDs are available, exploit them for the bus-off attack.
- **Fabricated ID** – fabricating and thus exploiting the preceded ID of a target message.

We evaluated all of these in order to verify their accuracies and efficiencies. For the first scenario, the adversary and the victim were programmed to send messages every 10ms with the same ID but different DLC values. The first transmissions from both nodes were initiated by a reference message sent by the non-victim node. For the second scenario of exploiting genuine preceded IDs, the nodes were programmed equivalently as discussed in Section 5.1. Finally, for the third scenario, we programmed the adversary to fabricate one preceded ID message per attack as shown in Fig 7. For each attack scenario, we examined 50,000 bus-off attack trials.

When periodicity was exploited to synchronize the Tx timing for a bus-off attack, due to jitters, only 58 out of 50,000 trials (0.12%) were able to trigger a bit error at the victim, thus eventually not triggering a bus-off. On the other hand, when genuine preceded IDs were assumed to be present and thus were exploited, all 50,000 trials succeeded in increasing the victim's TEC. Even without assuming that there is a genuine preceded ID for the target message, the adversary increased the victim's TEC 45,127 times (out of 50,000 trials) by fabricating it, i.e., a 0.9025 success probability. Note that in achieving such a high probability, one fabricated preceded ID was sufficient since the jitter deviation $\sigma_v = 0.023$ms. Although some attempts failed, due to the high success rate and the nature of change in TEC (i.e., $+8$ in TEC in case of error and -1 in the absence of error), iterative bus-off attacks eventually forced the victim to bus off as shown in Fig. 16. One can see that the change in TEC is slightly different from the one in Fig. 12 due to some failed attempts. These results show that a preceded ID — regardless of whether genuine or fabricated — is a good indicator for determining the exact timing of a specific message, and is indeed useful for mounting a bus-off attack.

5.4 Bus-off Attack on Real Vehicles

To evaluate the feasibility of bus-off attack further, we also conducted experiments on two real vehicles, 2013 Honda Accord and 2016 Hyundai Sonata shown in Figs. 17(a) and 17(b). During our experiments, the vehicles were immobilized for safety in an isolated and controlled environment. As shown in Fig. 17, through the On-Board Diagnostic (OBD-II) port, we were able to connect our CAN bus prototype to their in-vehicle CAN buses, both of which run at 500Kbps. Thus, the 3 prototype nodes were able to read all 40 distinct broadcast messages from the Honda Accord's CAN bus and 58 distinct messages from the Hyundai Sonata's CAN bus. Moreover, the nodes were capable of injecting and delivering arbitrary messages to the in-vehicle ECUs.

Increasing the TEC of a real in-vehicle ECU. We first experimentally show that an attacker can synchronize its Tx timing with a real ECU, increase its TEC, and thus succeed in launching a bus-

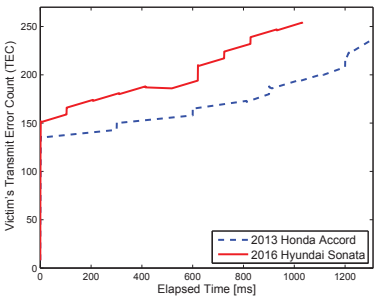


Figure 19: Iterative bus-off attack in a Honda Accord and a Hyundai Sonata.

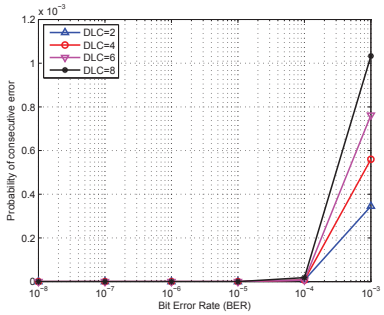


Figure 20: Probability of two consecutive errors.

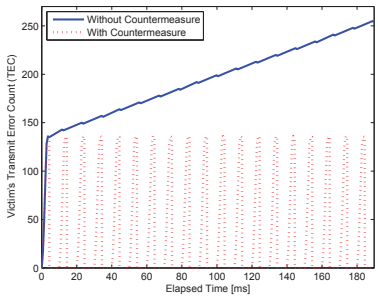


Figure 21: Efficiency of the proposed countermeasure.

off attack. For the evaluation on the Honda Accord, one prototype node was programmed as the attacker mounting a bus-off attack on one of its ECUs, which sent message 0x295 every 40ms. The attacker node made 15 attempts of the bus-off attack on that ECU by fabricating the preceded ID of 0x295. For the evaluation on the Hyundai Sonata, the node was programmed to mount the attack on an ECU which sent message 0x164 every 10ms. Similarly, 15 attempts of the bus-off attack via fabrication of preceded ID were made.

Due to restrictions in accessing the error counters of real in-vehicle ECUs, we validate the attack on real ECUs by exploiting the fact that in Phase 1 of the bus-off attack, the attacker's and the victim's TECs increase *equivalently* and both eventually exceed 127 within a very short period. So, Fig. 18 shows (i) how the attacker's TEC changed during Phase 1 of 15 bus-off attack attempts under both settings, and (ii) how the TEC of the in-vehicle victim ECU changed. In contrast, during Phase 2, the attacker's TEC does not reflect the victim's TEC. Since the results from only Phase 1 are valid, once the attacker's TEC exceeds 127, the node is reset to initialize its error counters. For the real ECU's TEC to be re-initialized so as to iterate Phase 1, the adversary mounted the attack every 5 secs, thus allowing the real ECU to decrease its TEC back to 0 as a result of its error-free transmissions during this period.

One can see from Fig. 18(a) that in 13 out of 15 attempts, the attacker's and hence the Honda Accord ECU's (victim's) TEC rose steeply, thus making both ECUs enter error-passive mode. As the attack was mounted via one fabricated preceded ID, 2 out of 15 attempts failed. The steep rises of TEC were again due to automatic retransmissions by the attacker node's and the in-vehicle ECU's CAN controllers. Similarly, Fig. 18(b) shows that, except for one attempt, the Hyundai Sonata ECU's TEC always increased when the bus-off attack was mounted. Although we could only show the result from Phase 1, it implies that the attacker can synchronize the Tx timing with a real ECU, and iterate such a process to continuously increase its TEC, eventually forcing it to disconnect from the in-vehicle network. Moreover, it shows that the attack can succeed regardless of the vehicle model/year, and corroborates an important fact of the bus-off attack: there is no need to reverse-engineer messages or checksums for mounting the bus-off attack, thus making it easier for the adversary to launch the bus-off attack.

Forcing an in-vehicle ECU to bus off. To further demonstrate the feasibility of the bus-off attack on real vehicles, we also evaluated a scenario in which one of the CAN prototype nodes was made to be the victim. The period of the target message sent by the victim was set to 50ms. Since the three CAN prototype nodes were capable of exchanging messages with real ECUs in both vehicles, they were successfully *added* to their in-vehicle CAN networks, and hence operate/act as if they were real ECUs. As a result, an attack on one of them would be equivalent to an attack on a real ECU. Unlike the prototype setting, however, their bus loads were significantly higher — due to traffic generated by real in-vehicle ECUs — during the attack, i.e., the attack was evaluated in highly complex CAN bus traffic. Fig. 19 shows the changes in TECs of the victim node being attacked on the 2013 Honda Accord and the 2016 Hyundai Sonata. Under both settings, through iterative bus-off attacks, the victim became error-passive within 2.4ms and eventually entered bus-off mode. Compared to the prototype setting, since the target message was set to have a larger interval and the bus loads were much higher, the overall delays of the victim entering error-passive and bus-off were larger.

These results on the two real vehicles confirm that the bus-off attack is indeed a severe, real problem.

6. RELATED WORK

As a countermeasure against attacks on in-vehicle networks, message authentication and intrusion detection systems (IDSs) have been the two main lines of defense.

Providing message authentication for CAN is difficult due to the limited space available for appending a Message Authentication Code (MAC) in its data field. Moreover, the requirement of real-time communication and processing makes the provision of authentication a non-trivial problem. Several schemes have been proposed to overcome these difficulties. The authors of [21] proposed to truncate MAC across multiple frames. Similarly, the authors of [17] proposed to use multiple CRC fields for including a 64-bit MAC. To achieve such authentications, the entire message has to be received by the transceiver and delivered to the upper layer. However, for the proposed bus-off attack, the adversary causes a bit error at the victim *during* message reception. Thus, even though a MAC is appended to the message, its functionalities will be nullified. Similarly, the functionalities of message checksums are also nullified.

Other than authentication methods, IDSs have also been proposed as countermeasures. The essence of state-of-the-art IDSs is to monitor the periodicity and contents of messages, and verify whether there are any significant changes to them. The authors of [14] proposed a method of measuring the entropy of an in-vehicle network and used the result as a specification of the behavior for an IDS. Similarly, a method of modeling the distribution of message intervals was proposed in [12] to define a norm behavior. Not only message frequency but also obvious misuse of message IDs (e.g., ID=0x00) were monitored in [8]. Although such IDSs are capable of detecting attacks to some extent, they overlooked the fact that message periodicity can change even in a normal or uncompromised environment. For example, if an error had occurred due to hardware/software fault, then that message would

automatically be retransmitted by the CAN controller, thus changing its transmission interval [19]. As a result, since an abnormal periodicity may be due to a system error as well as an attack, it is unreasonable to directly map such a symptom to an attack. For example, 16 consecutive error frames can occur due to not only a bus-off attack but also improper CAN termination [4]. This implies that for the proposed bus-off attack, the IDS may not tell if the error was due to an attack or a system error.

7. COUNTERMEASURES

The proposed bus-off attack is an important vulnerability, especially in view of its capability of nullifying MACs and checksums, and also deceiving IDSs to think there is a system error although the network is actually under attack. We propose a new defense mechanism which leverages the following features of the bus-off attack for its prevention.

In Phase 1, due to CAN's automatic retransmission,

F1. at least two *consecutive* errors occur during the transmission of frames. Thus, we watch for consecutive error frames with an active error flag.

In Phase 2, due to the difference in error modes,

F2. at the time when the (error-passive) victim's TEC increases, a message with the same ID will be successfully transmitted by some ECU on the bus.

F1 indicates a bus-off attack. We first simulated the probability of F1 under an uncompromised condition. We leveraged the same error model in [19] where bit error occurrences follow a Bernoulli distribution. In our simulation, for a given DLC and Bit Error Rate (BER), we randomly generated 100,000 different CAN messages and measured how many of them satisfy F1. The simulation result, plotted in Fig. 20, shows that even under an unusually high BER of 10^{-3} (usually $10^{-5} \sim 10^{-7}$ [19]), the maximum probability of F1 was only 0.11%. So, under a normal condition, the probability of F1 occurring 16 times in a bursty manner can be considered 0, whereas it was 1.0 during a bus-off attack. Considering this large discrepancy, we can consider F1 to indicate a bus-off attack. However, since F1 can also occur due to severe system errors such as improper bus termination [4], bit flip [22], and bit drop/insertion [15], F1 alone cannot be a definitive evidence of a bus-off attack.

F2 is the evidence. After occurrence of F1, once an error-passive ECU experiences a bit error again when transmitting a message with ID=$\mathbb{M}$, it can further monitor the CAN bus and check if there was any successful transmission of another message with the same ID=$\mathbb{M}$, i.e., occurrence of F2. This can only occur when two or more ECUs are sending the same message at the same time, which is not allowed on CAN and thus infeasible even in a severely erroneous network, while it is possible under a bus-off attack. From both F1 and F2, we can thus verify the occurrence of a bus-off attack, i.e., the observed symptoms are not caused by a system error. Following this reasoning, we propose the following defense mechanism: an ECU or its error counters are reset whenever F2 is observed after N consecutive error frames. We use $N = 16$ to reset the victim ECU upon occurrence of both F1 and F2.

Efficiency of the proposed countermeasure. We evaluated the bus-off attack on our CAN bus prototype with the proposed countermeasure. According to the proposed defense, the nodes were programmed to reset when F2 was observed after 16 consecutive and bursty error frames. During an iterative bus-off attack, Fig. 21 shows how the victim's TEC changed with and without the proposed defense. By observing the presence of bursty error frames on the CAN bus, the victim's error mode mostly stayed as error-active, which is in sharp contrast to the case without any countermeasure, and also successfully transmitted its messages, efficiently preventing the bus-off attack.

Alternative countermeasures. In our proposed countermeasure, we looked for consecutive error *frames* (i.e., F1) to prevent the bus-off attack. We may also consider a countermeasure which verifies if consecutive errors incur at the same *bit* position, instead of *frames*. As this is, in fact, the strongest evidence of the bus-off attack, we may use it as the condition of triggering an ECU reset. It may also be considered as a new rule in CAN for *not* increasing the error counters. Then, as the detection of a bus-off attack can be made much faster with the stronger evidence, unwanted retransmissions, which were at least 16 in our proposed countermeasure, can be reduced further. Note that these retransmissions may affect other messages in meeting their hard/soft deadlines. Albeit effective, such a countermeasure accompanies numerous challenges and limitations: 1) detecting individual bit errors would require changes in hardware, thus incurring expensive development cost and 2) detecting which bit has currently changed and comparing it with the previously changed one would require additional memory in the CAN transceiver chip.

The proposed bus-off attack exploits the *periodic* feature of in-vehicle network messages for synchronizing its transmission time with the victim's. It is thus limited to only periodic messages but it is very effective since most in-vehicle messages are periodic. Therefore, an alternative countermeasure can be to transmit periodic messages with some random factors added to their Tx times (e.g., adding random jitters), making them somewhat aperiodic. However, adding random factors to Tx times can create serious unexpected problems such as priority inversion, message sequence inversion, and deadline violation [6]. As this incurs detrimental effects on the message scheduling mechanism, we did not consider it as a possible countermeasure against the bus-off attack.

8. DISCUSSION

Severity of the bus-off attack. Since contemporary in-vehicle networks are not equipped with security mechanisms, an adversary can mount not only the bus-off attack but also other types of (previously covered) attacks on in-vehicle networks. For example, the attacker can simply inject arbitrary messages on the CAN bus or monopolize the network by continuously sending the highest-priority frames. However, the following facts make the bus-off attack a more severe problem and an attack that an adversary might favor over other attacks. Previously demonstrated/covered attacks either show obvious misuse of message IDs or significantly increase the message frequency, which can easily be detected and then removed by existing IDSs [8, 12, 14]. In contrast, the proposed bus-off attack can be mounted without misusing IDs and requires only a small increase in message frequency — up to a frequency that renders the bus-off attack feasible even during a system error (e.g., bit drop/insertion). This may cause the existing IDSs to be confused whether the symptom is due to an attack or a system error. More importantly, unlike previously demonstrated attacks, the adversary can mount a bus-off attack without any knowledge on the meanings or purposes of messages, making their reverse-engineering unnecessary. Also, the attacker succeeds before the victim verifies a message's checksum, and hence can use an arbitrary checksum in the attack message, i.e., no need to reverse-engineer the implemented checksum algorithm. Even if MACs were used for in-vehicle network messages, their functionalities could likewise be nullified. Requiring far less painstaking (than reverse engineering) efforts — especially when the adversary wants to mount attacks on various types of vehicles — differentiates the bus-off attack from previ-

ously known vehicle attacks and can also be a strong motivation for the adversary to prefer the bus-off attack to others.

Limitations of the bus-off attack. In order to succeed in the bus-off attack, the attacker must synchronize its transmission timing with the victim's. To achieve this, the proposed bus-off attack leverages the CAN protocol's buffering strategy via a genuine/fabricated preceded ID message. As mentioned before, however, its exploitation is only feasible when the target message (from the victim) is sent periodically. In other words, the attacker cannot force a victim to bus off when he is sending messages aperiodically. Meanwhile, since most messages in CAN are sent periodically, the attacker can succeed in mounting the bus-off attack on most ECUs in the in-vehicle network.

Vulnerability of other in-vehicle networks. Although most modern vehicles are equipped with CAN, some may deploy other protocols, such as CAN-FD, TTCAN, and FlexRay for more complex operations. Note that CAN-FD is an enhanced version of CAN, providing flexible and higher data rates as well as a larger data field [15]. Since CAN-FD's basic components, arbitration, and error handling all conform to those in CAN, it is also vulnerable to the proposed bus-off attack. What makes CAN-FD more interesting is that an attacker can monitor the newly introduced Extended Data Length (EDL) and Bit Rate Switch (BRS) fields, recognize which messages are sent with high bit rates or large payloads — safety-critical messages — and target them for a bus-off attack.

TTCAN is a session-layer protocol in which its message transmissions are based on a static schedule, time-triggered paradigm to provide Tx determinism [11]. As a result, all ECUs transmit their messages only at their assigned time slots and are periodically synchronized through a broadcast reference message. Since the allocated time slots are predefined in a schedule matrix and are stored in each node, the attacker — having control of a vehicle running TTCAN — is provided with the knowledge of what messages are sent and when, thus making the bus-off attack easier.

In FlexRay, which is designed to be more reliable than CAN as in CAN-FD, the error modes are divided into 3 different modes: normal-active, normal-passive, and halt. Normal-active mode is essentially equivalent to the error-active mode of CAN, whereas normal-passive mode differs from the CAN's error-passive mode, as it does not allow the nodes to transmit in that mode. Accordingly, FlexRay becomes invulnerable to the proposed bus-off attack.

9. CONCLUSION

In this paper, we discovered a new vulnerability, called the *bus-off attack*, of in-vehicle networks. The attack exploits their error-handling scheme to disconnect an uncompromised ECU and/or even shut down the entire in-vehicle network. We analyzed its practicability and demonstrated the attack on a CAN bus prototype and two real vehicles. Based on our analysis and experimental results, we proposed a defense mechanism to prevent the bus-off attack. Even though the proposed attack has not yet been seen in the wild, it is easy to mount and also directly related to drivers/passengers' safety, and should thus be countered with high priority. Moreover, the facts that the proposed attack can nullify state-of-the-art solutions and is easy to launch, make it even more important to design and deploy its countermeasures. Thus, we recommend concerted efforts from both academia and industry to account for this vulnerability in the design of in-vehicle networks.

Acknowledgments

The work reported in this paper was supported in part by the NSF under Grants 1505785 and 1646130.

10. REFERENCES

[1] Microchip MCP2515 Datasheet. [Online].
[2] NXP SJA1000 datasheet. [Online] Available: www.nxp.com.
[3] CAN Specification Version 2.0. *Bosch*, 1991.
[4] Symptoms of Improper CAN Termination [Online] Available: http://digital.ni.com/. May. 2001.
[5] 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 *USENIX Security*, 2011.
[6] R. Davis, S. Kollmann, V. Pollex, and F. Slomka. Controller area network (can) schedulability analysis with fifo queues. *ECRTS*, 2011.
[7] I. Foster, A. Prudhomme, K. Koscher, and S. Savage. Fast and Vulnerable: A Story of Telematic Failures. In *WOOT*, 2015.
[8] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive can networks - practical examples and selected short-term countermeasures. In *Reliability Engineering and System Safety*, Jan. 2011.
[9] T. Hu. Deterministic and flexible communication for real-time embedded systems. *Ph.D Thesis*, 2015.
[10] 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 S&P*, May 2010.
[11] G. Leen and D. Heffernan. Ttcan: a new time-triggered controller area network. In *Elsevier Microprocessors and Microsystems*, 2002.
[12] C. Miller and C. Valasek. Adventures in automotive networks and control units. *Defcon 21*, 2013.
[13] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015.
[14] M. Muter and N. Asaj. Entropy-based anomaly detection for in-vehicle networks. *IEEE IVS*, 2011.
[15] A. Mutter and F. Hartwich. Advantages of CAN FD Error detection mechanisms compared to Classical CAN. In *iCC*, 2015.
[16] M. D. Natale, H. Zeng, P. Giusto, and A. Ghosal. Understanding and using the controller area network communication protocol: Theory and practice. In *Springer Science & Business Media*, 2012.
[17] D. Nilsson, D. Larson, and E. Jonsson. Efficient In-Vehicle Delayed Data Authentication Based on Compound Message Authentication Codes. In *VTC-Fall*, 2008.
[18] O. Pfeiffer, A. Ayre, and C. Keydel. *Embedded Networking with CAN and CANopen*. 2003.
[19] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in can. *Symposum on Fault-Tolerant Computing*, 1998.
[20] R. Ruth, W. Bartlett, and J. Daily. Accuracy of event data in the 2010 and 2011 toyota camry during steady state and braking conditions. In *SAE International Journal on Passenger Cars*, 2012.
[21] C. Szilagyi and P. Koopman. Low cost multicast network authentication for embedded control systems. In *Workshop on Embedded Systems Security*, 2010.
[22] E. Tran. Multi-Bit Error Vulnerabilities in the Controller Area Network Protocol. *Ph.D Thesis*, 1999.
[23] F. Yang. A bus off case of can error passive transmitter. In *EDN Technical Paper*, 2009.