CANARY - a reactive defense mechanism for Controller Area Networks based on Active RelaYs

Bogdan Groza
Politehnica Univ. of Timisoara

Lucian Popa
Politehnica Univ. of Timisoara

Pal-Stefan Murvay
Politehnica Univ. of Timisoara

Yuval Elovici
Ben-Gurion University of the Negev

Asaf Shabtai
Ben-Gurion University of the Negev

Abstract

We are rethinking the decades-old design of the CAN bus by incorporating reactive defense capabilities in it. While its reliability and cost effectiveness turned CAN into the most widely used in-vehicle communication interface, its topology, physical layer and arbitration mechanism make it impossible to prevent certain types of adversarial activities on the bus. For example, DoS attacks cannot be stopped as the physical layer gives equal rights to all the connected ECUs and an adversary may exploit this by flooding the network with high priority frames or cause transmission errors which may move honest ECUs into the bus-off state. In response to this, we propose a reactive mechanism based on relays placed along the bus that will change the network topology in case of an attack, i.e., a moving target defense mechanism, allowing a bus guardian to filter and redirect legitimate traffic. We take care of physical properties of the bus and keep the 120 Ω load constant at the end of the lines whenever relays are triggered to modify the topology of the bus. We build a proof-of-concept implementation and test it in a laboratory setup with automotive-grade controllers that demonstrates its functionality over collected real-world in-vehicle traffic. Our experiments show that despite short term disturbances when the relays are triggered, the frame loss is effectively zero.

1 Introduction and motivation

The Controller Area Network (CAN) is a bus standard designed by BOSCH in the 80s which became the most widely used networking layer inside cars in the decades that followed. While famed for its design simplicity, reliability and cost effectiveness, the recent years have unfortunately and unsurprisingly proved that the lack of security on CAN opens road for numerous exploits of modern vehicles.

The security limitations of the CAN bus are twofold. On one hand CAN has no intrinsic security - this is now widely known and accepted. Adding cryptography (for authentication and encryption) may solve the problem in this respect and the industry is heading in this direction [2]. But on the other hand and equally important, even if cryptography is in place, the design of the CAN bus, its topology, physical layer and arbitration mechanism, set room for Denial-of-Service (DoS) attacks as frames with high priority (lower value identifiers) always win the bus and there are no guarantees for the arrival time of low priority frames in case of a flooded bus. Moreover, adversaries may cause transmission errors by simply flipping bits in legitimate frames and such transmission errors may eventually trigger the Bus-off state on honest nodes that comply with the error control mechanism of CAN. In this way, the transmission capabilities of legitimate nodes are cutoff, i.e., another type of DoS. The practical impact of a DoS is obvious as control will be lost on all vehicle subsystems once the bus becomes unavailable to legitimate electronic control units (ECUs).

The adversarial actions, reported by numerous papers, e.g., [5, 20], require a malicious device to be connected to the bus via an exposed port or corrupting (possibly from remote) a legitimate node that is already connected to the bus. The on-board diagnostics (OBD) port is a good candidate to gain access to the bus [29], though in some in-vehicle network deployments this port may be isolated from the rest of the ECUs by a firewall. Besides these exposed interfaces, CAN bus wires are accessible in various places, e.g., under the hood or behind car infotainment units, and an adversary may use any random location on the wires as a penetration point provided that he has physical access to the car. While corrupted ECUs may appear as a more distant possibility as most vehicle components come from trusted providers, recent research works have proved that legitimate devices, e.g., car headunits or telematic units, can be compromised from remote. Supply chain attacks may also lead to compromised devices being obliviously mounted in the car by honest manufacturers or repair shops. These three attack vectors, i.e., an adversary at the OBD port, one which taps the bus at some random location and a corrupted ECU, e.g., from the remote or by a supply-chain attack, are graphically depicted in Figure 1. To prevent such attacks, adding relays that allow for discon-

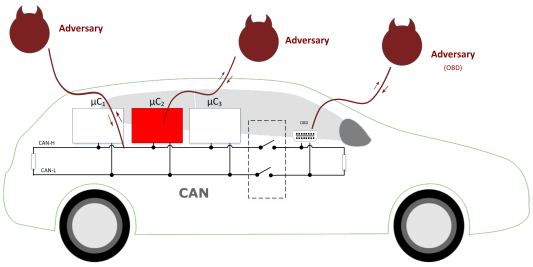


Figure 1: Addressed setting: an adversary taping to the bus, corrupting an ECU or connecting via the OBD port.

necting certain parts of the bus is a natural choice. An easy to imagine solution is the use of a relay on the OBD port which will disconnect it once an attack is detected (this is already suggested in Figure 1). Such a solution will be cost-effective but it would be too trivial to be able to hinder a well determined adversary that may use other entry points as well.

Challenges and contributions. The security of the CAN bus has been repeatedly studied in the past decade and the answer has always been the same: the CAN bus is insecure and adopting security mechanism uneasy due to various constraints while it remains nearly impossible to prevent certain types of attacks, e.g., DoS. Clearly, these attacks may have devastating effects on cars, passengers or even bystanders. In this context, devising a solution for physical separation of the ECUs on the bus and dynamic network reconfiguration seems to be promising. However, to facilitate practical adoption by the industry, the solution must be down-to-earth, cheap, easy to understand and implement. Nonetheless, the solution has to comply with the physical requirements of the CAN standard and of course it has to preserve message arrival time on bus, i.e., strict timings are mandatory for safety-critical tasks. In brief, simplicity and real-time demands must be met.

To the best of our knowledge, our work is the first to provides an effective solution for physical isolation of intruder nodes on the CAN bus and thus the first approach that can protect the CAN bus against DoS attacks. While we specifically target DoS attacks on the CAN bus, the proposed solution is by no means limited to this type of attack and we further demonstrate capabilities against other adversarial behaviors as well. We imagine a framework with moving target defense capabilities where relays are placed next to each ECU and by triggering the relays we can physically separate the left and right sides of the bus. Of course, other placements for the relays can be imagined, in all other existing vehicle sub-networks. For simplicity, we focus on the more conventional case of a single CAN bus. A specialized micro-controller, called Bus Guardian, is in control of the logic for intrusion detection, intruder isolation by relay switching and traffic filtering/redirection. Naturally, we try to prevent the loss of legitimate frames on bus disconnections and we filter and replay traffic in parts of the bus to which the adversary has no access. While a small percent of frames may be unavoidably lost, this seems clearly preferable when compared to a bus that is fully blocked by an adversary. In particular, in our experiments, we demonstrate that none of the legitimate frames are lost when exercising the new intruder isolation capabilities. One important aspect is that we take care of specific details of the physical layer, e.g., keeping a constant 120Ω termination at the end of the lines (according to the standard to avoid reflections) while the bus topology changes.

The main contributions of the proposed defense mechanism, i.e., CAN with Active RelaYs (CANARY), can be summarized as follows:

1. we propose a simple yet highly effective modification of the CAN bus that complies with CAN physical specifications, e.g., 120Ω end-of-line resistors, and allows for dynamic reconfiguration of the bus topology that will isolate nodes in certain parts of the bus,

2. we provide algorithms for detecting intrusion, node isolation and, more importantly, traffic redirection by which, once the intruder is located, incoming traffic is filtered and redirected to other parts of the network,

3. to prove the correctness of our approach, we provide realistic experiments with automotive-grade controllers and collected real-world in-vehicle traffic,

4. we show that frame loss due to relay action is essentially zero and the arrival time of legitimate frames is largely preserved, only a small number of frames being affected by the adversarial interventions and relay triggering.

Needless to say, the proposed solution does not exclude regular cryptographic authentication and intrusion detection, but complements them with a reactive defense mechanism.

Advantages of the proposed defense mechanism. A key aspect of the proposed defense mechanism is that it can be used to retrofit existing cars. For many decades, after-market solutions have successfully retrofitted cars with RF controls, intelligent alarm systems, remote start systems, GPS-related functionalities, multimedia units, etc. Similarly, relays may retrofit existing and forthcoming cars with an effective mechanism against attacks and entry points which the manufacturer did not consider. While active star topologies for CAN buses may solve most of the problems we address here, such topologies are very rare inside cars. Moreover, when present, star topologies are frequently implemented as hybrid star-bus architectures, where several buses are connected together through a gateway and cannot hinder a DoS on any of the connected sub-networks. This architectural choice affects cars which are in production today and which will be on road for the decades that follow. Changing the bus to a star topology after production will be extremely hard, if not impossible, mostly due to difficulties in fully rewiring the car. In contrast, relays can be more conveniently mounted in existing cars

at key locations by specialized workshops without changing the network architecture and much of the wiring. Modern transceivers, e.g., NXP TJA115X chips, incorporate DoS protection mechanisms but they are only effective against their own host controller when it attempts to flood and cannot stop other nodes from doing so. A knowledgeable adversary would not be so naive to use the self-limiting NXP transceiver when performing an attack. Nonetheless, it will be hard (or impossible) to retrofit existing cars with TJA115X transceivers.

Paper organization. The rest of our work is structured as follows. Section 2 provides a short background on CAN and Section 3 briefly surveys the related work. Section 4 holds the theoretical description of the proposed framework. In Section 5 we present our experimental setup and our proof-of-concept implementation. Section 6 holds the experimental evaluation of the proposed framework. Finally, Section 7 holds the conclusion of our work.

2 Brief background on CAN

The CAN bus was designed for the specific requirements of the automotive domain. It provides bit rates of up to 1Mbit/s and mechanisms for message prioritization as well as for efficient error detection and confinement. At the physical layer, CAN is implemented as a two wire differential line which must be properly terminated at each network end by a 120Ω resistor. While the use of CAN is not limited to a bus topology, e.g., star topologies can also be found in practice, bus topologies are the most often employed network designs with CAN due to their design simplicity.

The CAN frame may transport a payload of at most 8 bytes. Other frame fields are dedicated to the main mechanisms implemented at the data-link layer. The arbitration field, i.e., the identifier field (ID), the remote transmission request bit (RTR) plus the the identifier extension bit (IDE), are employed to determine transmission priority (i.e. frames with lower-valued IDs win the arbitration) when multiple nodes simultaneously start frame transmission. Frame IDs, i.e., 11 bits in standard CAN frames and 29 bits in extended frames, are defined at network design time to establish frame priorities. The 15 bit CRC field is used as part of the error detection mechanism. A network node that detects a transmission error immediately begins transmitting an error frame to signal this finding to all other nodes and stop the undergoing frame transmission.

CAN also implements an error confinement mechanism to prevent disturbances from faulty nodes. This mechanism uses two error counters, TEC and REC, for transmitted and received frames, which are incremented each time an error is reported and decremented after each successful message transmission or reception. All nodes start in the *Error Active* state in which they can interrupt frame transmissions with error frames. Once the error counters exceed the defined threshold, i.e., REC or TEC greater than 127, the ECUs transition in the *Error Passive* state, in which they cannot interrupt frame transmissions with error frames. They can return from this state when both TEC and REC are smaller than 128. The ECUs eventually reach the *Bus-off* state, in which the node will stop transmitting and acknowledging frames, if TEC becomes greater than 255. Notably, this error confinement mechanism has been exploited both to send legitimate ECUs into the Bus-off state [6] as well as against adversaries [26] (though, there are little chances that an adversary will comply with this since the Bus-off state can be bypassed from the software layer).

3 Related work

It can be easily seen from the above description that CAN provides no security mechanisms and that security was not considered as a goal during its design time more than three decades ago. As a consequence, CAN is vulnerable to spoofing and replay attacks as reported in [5,20] and to DoS attacks in particular [21,24].

Attack prevention and detection is subject to many recent lines of work on in-vehicle network security. The use of additional hardware is common in addressing CAN bus security [16]. Matsumoto et al. [19] are the first to propose the idea of an intrusion prevention mechanism that destroys intruder frames by generating error frames. Several different lines of work [11,16] adapt and implement this approach in a centralized form while a software-based implementation alleviating the need for specialized CAN controllers is proposed in [9]. Another approach proposed for attack prevention is ID-hopping which involves constantly modifying CAN frame identifiers through a secured procedure only available to legit nodes. Such an approach was first proposed by Humayed and Luo [15] which are using a software-based implementation that requires the involvement of a gateway node in the ID-hopping procedure. An improved approach based on a dedicated CAN controller which reduces computational and communication overheads while providing increased ID entropy is proposed in [31].

The prevention mechanisms explored so far in related research works are effective against replay and spoofing attacks. However, preventing DoS attacks is more difficult. The most simple form of DoS attack, mentioned for the first time in [30], exploits the CAN arbitration mechanism which establishes transmission priority based on message identifiers, i.e., the lower the ID value, the higher the priority. Thus, continuously sending frames with the highest priority would prohibit any legit transmissions. Another attack approach reported in [24] and [21] is to manipulate CAN transmissions directly at the physical layer to prevent correct generation and interpretation of CAN symbols. This type of attack can be used to completely block CAN communication or can be even targeting specific messages or nodes [21]. Several lines of work have proposed solutions for some types of DoS attacks. The work in [6] introduces a mechanism used to detect and prevent a

DoS attack by resetting the targeted ECU and preventing it to reach the Bus-Off state. ID-hopping is efficient in preventing DoS attacks targeted to specific messages. However, none of these related works can help against a generalized DoS attacks that prevents all CAN transmissions by flooding the bus with a high-priority ID.

Countermeasures such as disconnecting adversarial segments of the bus are to the best of our knowledge yet unexplored. Interestingly however, the idea of using relays to disconnect sections of the CAN bus was previously employed by several works for fault detection and recovery on CAN [25]. More recently, similar topologies with relays were studied in the context of fault diagnosis by [32] and [33]. Note however that these works are using basic relays to simulate broken wires. The pairwise 2-pole-relay-resistor structure from our setup (detailed later in Figure 3) is unique to CANARY and to the best of our knowledge has not been proposed elsewhere. Another approach proposed for fault isolation is the use of an active star topology where all nodes are connected to a central node. This node acts as a router which isolates traffic from nodes found to be faulty or ones transmitting other IDs than they are supposed to according to a routing table [23].

There is also a large body of works that addresses intrusion detection systems (IDS) on CAN. Our work does not rely on a specific IDS, we use Bloom filters [3] because of their compact representation. The use of Bloom filters in the context of CAN buses has been also explored in [12]. There are of course many other solutions and any of them can be integrated in the IDS from the current proposal. Several works have focused on basic aspects of CAN traffic to detect intrusions such as the frequency of frame arrival time [28], the Hamming distance between frames [10] the entropy of CAN frames [22], [18] or timing characteristics of a remote frame [17]. Other works have focused on physical characteristics such as clock skews [7] or voltage levels [8]. Some recent overviews on existing proposals for securing the CAN bus can be found in [1] and [4].

4 Proposed framework: modified CAN topology and the defense mechanism

This section provides the description of the modified topology that stays at the core of our experimental setting and provides an overview of the proposed solution.

4.1 Modified CAN-bus topology

We begin with providing an overview of the new network architecture in Figure 2. The network topology shows relays placed next to each node and the Bus Guardian recording traffic to the left and right sides of the network. Upon detecting an intrusion, the Bus Guardian will trigger the relays to locate the intruder, isolate it to the left or to the right side of the network, and then it will filter and redirect traffic from one

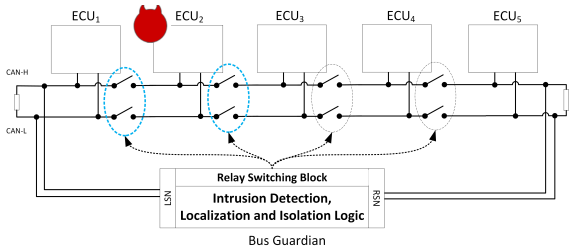


Figure 2: Brief schematic of the network topology, relay placement, Bus Guardian and an adversary near ECU_2.

side of the network to the other. We also suggest potential adversarial presence near ECU_2. By triggering the relays to the left or right of the ECU, the adversary can be isolated to the left or to the right of the network, and incoming traffic filtered and redirected to the side which is free of the adversary. The adversary can be also cut-off from the network by triggering both the left and right relays that surround him but by doing this, one may also remove a legitimate ECU, such as ECU_2, which is undesired. Consequently, isolating the intruder successively to the left and right sides of the network, filtering incoming traffic and redirecting it is the preferred solution.

Figure 3 shows the relay placement around a single node, i.e., ECU_i. There is one relay on the CAN-Low line, i.e., $\mathsf{R}_{\mathsf{low},i}$, and one on the CAN-High wire, i.e., $\mathsf{R}_{\mathsf{high},i}$. The relays will be triggered at the same time such that the impedance at the end of the line remains 120Ω when the relays simultaneously switch from position (1) to position (2) effectively closing the bus after ECU_i. The effects of relay switching on normal bus operation are discussed later in the experimental section. In brief, the time to switch the relays from our setup is around $5ms$ which may result in a brief disturbance of the bus. Since the time that a frame spends on the bus is around $200\mu s$ for a 500Kbps bus (a commonly employed speed), and applications usually work at a 50% bus-load, an average of a dozen frames may be occurring on the bus during this $5ms$ interval. Since each sender will get a transmission error in such circumstances (due to the existing error control mechanisms on the CAN bus) and will automatically attempt to re-send the frame, the number of lost frames is actually zero. This is later proved in our experiments. Nonetheless, the $5ms$ switching time was achieved with some off-the-shelf JQC-3F-5VDC relays that required no special adaptations for our setup. If needed, for more demanding applications, much faster relays are available that can operate well beyond the $1ms$ range and which can ensure that bus disturbances will last for at most the period of a single CAN frame, i.e., $\approx 200\mu s$ for the 500Kbps CAN. Figure 3 extends this graphical depiction by showing the relay placement in case of the five nodes from our setup. By switching any pair of relays $\mathsf{R}_i =< \mathsf{R}_{\mathsf{low},i}, \mathsf{R}_{\mathsf{high},i} >, i = 1..n$ the bus is cut after ECU_i, effectively splitting the bus into two distinct sub-networks, while the Bus Guardian can still route traffic from one side to the other.

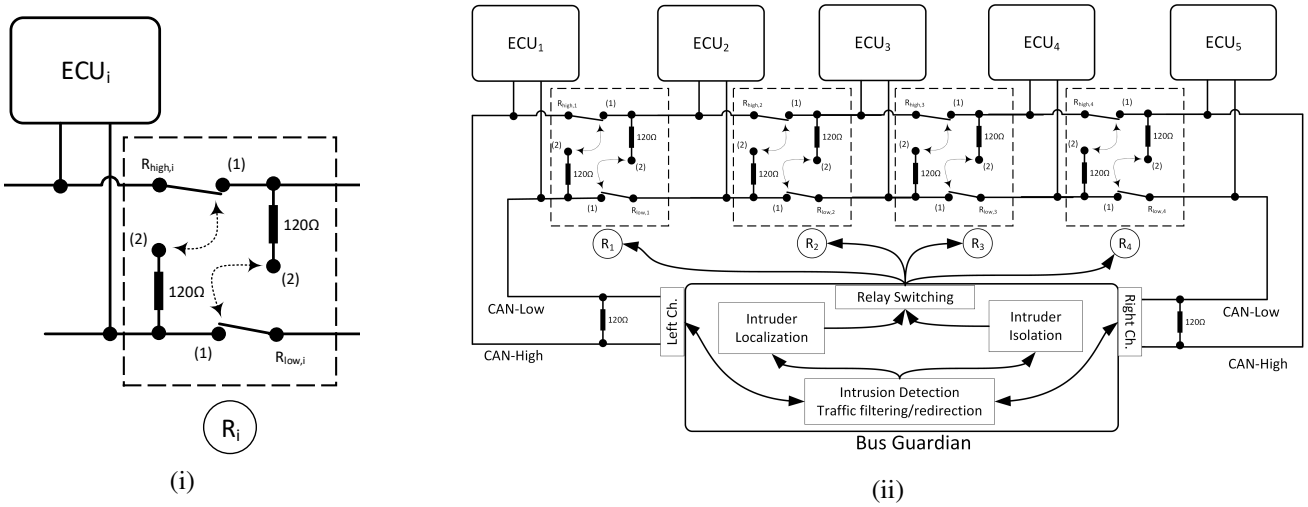


Figure 3: Relay placement near a single node (i) and detailed schematic with relays near five nodes and a Bus Guardian (ii)

4.2 Adversary types and countermeasures

Based on the exact intruder location, we envision three types of adversaries against CANARY's defense mechanism:

- Type (I) adversaries are the easiest to address. They are adversaries that tap the bus, or compromise a unit, at a location which can be completely isolated. Traffic redirection from the left and right sides of the bus may be needed, if the adversary is between two relays in the middle of the bus. This can be done at 0% frame loss as shown later in the experiments. A Type (I) adversary is shown in Figure 4 (i) which depicts a compromised ECU_5 that is isolated by R_5. If the compromised unit is non-essential, e.g., an OBD diagnosis tool or an Android head unit, this case does not require traffic redirection.

- Type (II) adversaries will tap the bus in the vicinity of a controller located at the end of the bus. In this case the relays will disconnect the adversary from the bus along with the legitimate node, e.g., R_1 in Figure 4 (ii). Traffic will be redirected from both sides (and filtered when coming from the adversary side). The worst damage that the adversary could inflict is to cause a DoS with no recoverable traffic from its bus segment, but the rest of the network remains unaffected. A practical example may be a compromised peripheral, e.g., a controller of some vehicle body element (mirrors, windows, etc.) that may be isolated at the cost of losing the functionalities nearby but without affecting the rest of the vehicle functionalities (doors, ignition, etc.).

- Type (III) adversaries are the most dangerous. They tap the bus near an ECU in the center of the bus making it impossible to isolate the adversary between two relays without the legitimate ECU (which in this case we assume to carry some essential functionalities). Traffic has to be redirected from both sides (and filtered) but the adversary may cause a full DoS on the side where it was isolated. Since we cannot afford to drop the functionalities on either sides (to the left or right of the adversary) nor on the legitimate ECU nearby, traffic has to be load-balanced between the left and right sides, e.g., this is done by alternatively triggering relays R_2 and R_3 in Figure 4 (iii). Worst case, the adversary can cause a full DoS on the side where it is isolated, but due to the load-balancing, the DoS will be halved on both sides of the network and only the nearby ECU will be continuously affected. A practical example could be a compromised legitimate ECU located at a key position on the bus.

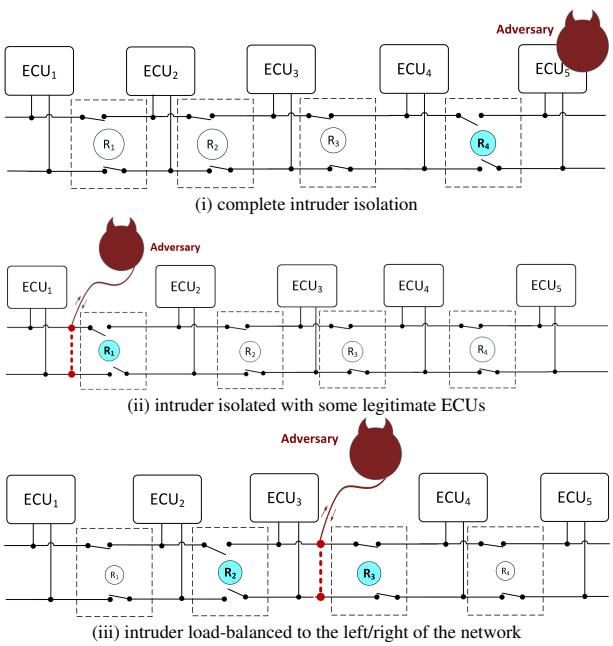


Figure 4: The three types of intruder locations along relays and the corresponding defense mechanisms in CANARY

Briefly: Type (I) adversaries can be fully isolated at no cost for legitimate nodes, Type (II) adversaries can be isolated while possibly losing communication with some (hopefully non-essential) ECUs and Type (III) adversaries cannot be fully isolated but have to be load-balanced to the left and right sides of the network. Other variations can be imagined. For example, depending on specific implementation details, if the functionalities of ECU_3 from Figure 4 are non-essential (iii), the Type (III) adversary in the middle of the bus may be also cut along with ECU_3 while traffic is redirected between the left and right sides. Ultimately, in case of a complete DoS it may be even preferable to isolate the adversary along with ECU_3 by cutting their bus segment. Such a decision however, depends on the specific functionalities that the ECU is implementing, for the exposition in this work we cannot delve into such details. For a crisper image, Figure 4 mostly depicts external adversaries, but as stated in the text, the adversary may be a compromised unit as well. As expected, since Type (III) adversaries are the hardest to address we focus most of our work on this type of adversary and the load-balanced defense mechanism. Type (I) and (II) adversaries can be addressed by immediate simplifications.

4.3 Overview of Bus Guardian activities

We provide a short overview of the actions of the Bus Guardian in the flowchart from Figure 5. The actions of the Bus Guardian begin by recording and filtering traffic to detect intrusions. Once the intrusion is detected, the intruder is located by Algorithm 1 and then it is isolated, traffic filtered/redirected according to Algorithm 2.

To detect intrusions, in the traffic filtering block from Figure 5, we suggest three types of simple and efficient mechanisms that can be used: *(1)* Bloom filters to detect frames with IDs that are unknown to the bus, i.e., fuzzing attacks, *(2)* a possible extension with counting Bloom filters, i.e., to spot replay attacks, and *(3)* monitoring for the arrival rate of frames on the bus λ which is a good indicator for DoS attacks. CAN deployments inside cars usually work at a 50% effective busload which corresponds to an arrival rate of $\lambda \approx 2000$ frames per second. If the busload peaks well over 50% for extended periods of time, e.g., several milliseconds or beyond, it may be a good indicator that a DoS attack is taking place. Since the filtering mechanism that we further use requires specific computations on each packet, in order to save computational it may be preferable to use the arrival rate of the packets for activating the filters. Determining the current rate λ requires only to count packets in a specific window, e.g., $100ms$, and a deeper inspection of the packets can be triggered only when needed. On the high-end controller that we used, the computations required by the filters were easily handled and consequently in all the experiments that follow we detect intrusions based on any single packet that is not part of the trace (regardless of the current bus rate λ which we suggest as an option for implementation on low-end controllers). In our application we use a regular Bloom filter to detect unknown IDs and a secondary Bloom filter to detect multiple occurrences of the same same frame during small time intervals, i.e., replay attacks. Finally, any existing IDS can be incorporated inside CANARY's Bus Guardian. The main focus of this work and our primary contribution are the intruder isolation and load balancing mechanisms which have never been addressed before and which are the only mechanisms proposed so far that can alleviate DoS attacks.

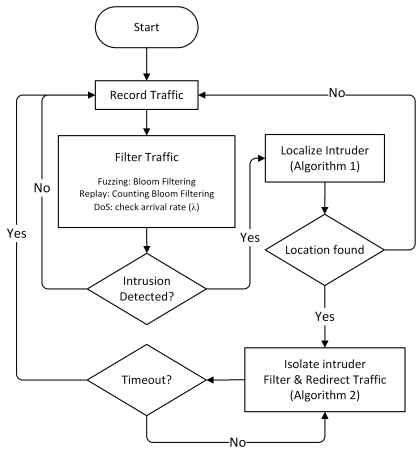


Figure 5: Flowchart of the Bus Guardian actions

The Bloom filter [3] is an array of m bits that is modified by the output of k hash functions. For each message that the filter learns, each of the k hash functions selects an index $1..m$ in the filter and the corresponding bit is set to 1. This happens during the training phase. Later, to check that a message is recognized by the filter, the k indexes are verified and the message is recognized if and only if all their corresponding bits in the filter are set to 1. This structure can be naturally extended to count for multiple occurrences of the same object by replacing the bits inside the filter with counters, i.e., counting Bloom filters. A survey on various types of Bloom filters is available in [27]. We calibrated a set of regular Bloom filters and tested their efficiency on a CAN trace collected from a real-world vehicle (more details on this data can be found in the experimental section). Half of the messages were turned into adversarial injections with randomized IDs that are not present in the legitimate trace. None of the genuine frames are misclassified, this is the expected behavior for Bloom filters which have a zero false negative rate. From our implementation we determined that a filter size of 512-1024 bits provides excellent classification results with a false positive rate well below 0.1%. Storing 100 IDs would require at least 1100 bits (considering 11-bit IDs) and almost three times as much if 29-bit IDs are used, thus the 512-1024 bits provide a more compact representation (the size of the Bloom filter does not increase with the size of the IDs). We choose to rely on the

non-cryptographic hash function MurMur[1] which is very fast, i.e., it requires $0.572\mu s$ for one computation on our controller. This kind of functions are recommended for hash tables that do not require cryptographic security, making them ideal for Bloom filtering. To filter one message, the computational time peaked at $2.86\mu s$ when $k = 7$ and $3.89\mu s$ when $k = 10$.

4.4 Intruder localization agorithms

Having a mechanism for filtering frames in place, we can proceed to the intruder localization algorithm once the malicious frames are reported. As expected, the intruder localization algorithm performs a binary search on the network by disconnecting parts of it and analyzing incoming traffic on the left and right sides of the network. While an algorithm that successively disconnects each segment of the network is straight-forward to implement, its disadvantage is that it will fully disconnect segments of the bus, causing losses among legitimate frames, until the intruder is located. Also, if the adversary taps the bus near an honest node, then the algorithm will isolate the adversary along with the legitimate node which is undesired. An adversary may even exploit this algorithm by sending intrusion frames to mislead it and cut legitimate segments of the bus.

To circumvent these issues, Algorithm 1 uses only the relays from one side of each node. This way, it preserves all the network traffic to the left and to the right of the bus, the traffic can be filtered and redirected. The algorithm starts by setting the left index $l = 1$, the right index $r = n$ and loops until the left l and right r indexes are next to each other. At each step it disconnects only the relays in the middle, i.e., the relays at index $index = \lfloor (l+r)/2 \rfloor$. Then it repeatedly filters traffic for time T, i.e., until $(t'_{cur} - t_{cur}) > T$. If needed, in case of intermittent adversaries, the relays may be switched on-event whenever an intruder frame is detected. Note that a single intruder frame is sufficient to detect the intrusion and on-event triggering will make the relays converge to the location of the adversary while all the existing traffic is perfectly redirected to the left and right sides of the network. For each receive event on the left side of the network, i.e., *RxLeft*, or on the right side of the network, i.e., *RxRight*, the algorithm filters the frame and sets the intrusion flag, i.e., *ileft* or *iright*, then redirects the frame to the left or right accordingly. When time T elapses, the relays at position *index* are reconnected and the algorithm first checks if intrusions were detected on both sides of the network. If this is the case, then the algorithm returns -1 since the intruder cannot be isolated to the left or to the right of the network. Otherwise if the intruder is on the left, i.e., $ileft = true$, then the right index is modified, i.e., $r = \lfloor (l+r)/2 \rfloor$, else if the intruder is on the right, i.e., $iright = true$, then the left index is modified, i.e., $l = \lfloor (l+r)/2 \rfloor$. When the loop ends, the algorithm returns the index *index* of the relay next to the intruder.

[1] https://github.com/aappleby/smhasher/blob/master/src/MurmurHash2.cpp

Algorithm 1 Binary localization algorithm (single relay)

```
1: procedure DETECT NODE
2:     l = 1, r = n
3:     while (r − l) ≠ 1 do
4:         index = ⌊(l + r)/2⌋
5:         Disconnect(index)
6:         ileft ← false
7:         iright ← false
8:         t_cur ← GetTime()
9:         repeat
10:            if RxLeft then
11:                frame ← Receive(LChannel)
12:                frame′ ← Filter(frame)
13:                if frame′ ≠ frame then ileft ← true
14:                elseBufferedSend(frame′, RChannel)
15:            if RxRight then
16:                frame ← Receive(RChannel)
17:                frame′ ← Filter(frame)
18:                if frame′ ≠ frame then iright ← true
19:                elseBufferedSend(frame′, LChannel)
20:            t′_cur ← GetTime()
21:        until (t′_cur − t_cur) > T
22:        Reconnect(index)
23:        if ileft = true ∧ iright = true then return -1
24:        if ileft = true then r = ⌊(l + r)/2⌋
25:        if iright = true then l = ⌊(l + r)/2⌋
26:    end while
27:    return index
28: end procedure
```

4.5 Traffic redirection: bridged and load-balanced retransmission

An easy to address situation is that when the adversary is located alone on a segment of the network. If this is the case, the segment can be cut-off from the network and traffic bridged from one side to the other. But if this is not the case, and the adversary cannot be fully isolated, then a load-balanced retransmission that alternatively switches the adversary from the left to the right is needed - we discuss this mechanism in what follows.

Algorithm 2 Single relay, load balancing

```
1: procedure LOAD-BALANCED RETRANSMISSION
2:     while true do
3:         SwitchRelays()
4:         FilterRedirectTraffic()
5:     end while
6: end procedure
```

Algorithm 3 Switch Relays

```
1: procedure SWITCH RELAYS
2:     if (t/T_relay)mod2 ≠ (t_last/T_relay)mod2 then
3:         t_last ← t
4:         if (t/T_relay)mod2 = 1 then
5:             ileft ← true
6:             iright ← false
7:             Disconnect(index)
8:             Reconnect(index + 1)
9:         else
10:            iright ← true
11:            ileft ← false
12:            Reconnect(index)
13:            Disconnect(index + 1)
14: end procedure
```

Algorithm 4 Filter and redirect traffic (buffered)

```
1:  procedure FILTER TRAFFIC
2:      if RxLeft then
3:          frame ← Receive(LChannel)
4:          if ileft then frame ← Filter(frame)
5:          BufferedSend(frame, RChannel)
6:      if RxRight then
7:          frame ← Receive(RChannel)
8:          if iright then frame ← Filter(frame)
9:          BufferedSend(frame, LChannel)
10: end procedure
```

Load-balanced retransmission. Algorithm 2 provides the main loop of the traffic filtering algorithm with load-balancing capabilities. The intruder is isolated either to the left or to the right side of the network. This is done in step 3 of the algorithm which switches the relays to the left or to the right of the adversary location. The recorded traffic is then filtered and replayed on the other part of the network in step 4. By switching the adversary from one side of the network to another, we assure a load-balanced network and the effects of a DoS attack will be halved on each side since the adversary has access only to half of the bus each time. The procedure for switching relays is depicted in Algorithm 3. This algorithm simply switches from the left to the right side of the node *index* at time intervals T_{relay}. The steps for filtering and retransmission are given in Algorithm 4. The algorithm simply checks the left (line 2) and right (line 6) sides if there is a new incoming frame. If this is the case, the frame is recorded (lines 3 and 7), filtered if the intruder is isolated in the corresponding side (lines 4 and 8) and then replayed on the other side of the bus (lines 5 and 9). The transmission is buffered since the bus may be busy on the side where retransmission is attempted. More discussions on the size of the buffer follow in the experimental section. In case when the adversary performs a more aggressive DoS attack, there will be few or no legitimate frames at all on the side of the adversary since the bus is flooded by illegitimate frames. Moreover, legitimate frames from the other side cannot be redirected since there is no room left on the bus. For this case, buffered retransmission also helps since frames are kept in the buffer and sent when the adversary is isolated to the other side of the network.

We also provide a graphic depiction of the load-balancing retransmission in Figure 6. The depiction is provided for three consecutive steps with the right-side network (RSN) and left-side network (LSN) successively off and isolated from the adversary (the duration of a step is of 100*ms* similar to the experiments that follow). Incoming traffic from the side affected by the adversary is filtered. The filtering box is shaded when incoming traffic is free of adversarial interventions, i.e., the adversary is isolated on the other side of the network and thus the filter is inactive. We also depict the arrival rate of frames from the right and left side, i.e., λ_{left} and λ_{right}, as well as the arrival rate for adversarial frames, i.e., λ_{Adv}, as well as the arrival rate after filtering, i.e., λ^{φ}_{left} and $\lambda^{\varphi}_{right}$.

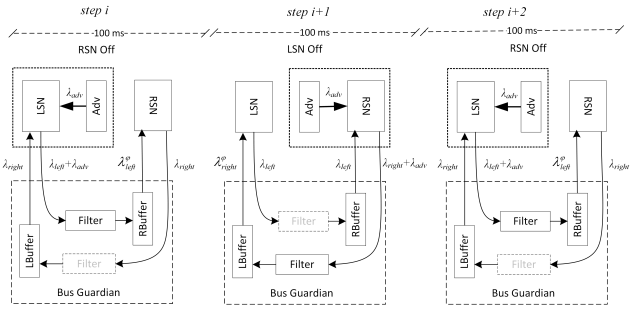


Figure 6: Schematic for three consecutive steps of the load-balanced retransmission

5 Setup and implementation

We now discuss implementation details regarding the Bus Guardian and give details on the recorded in-vehicle traffic that we used for building a realistic testbed.

5.1 Implementation of the Bus Guardian

Figure 7 shows the experimental setup of our work. The Bus Guardian with the role of intrusion detection, localization and prevention consists of an Infineon AURIX TC297 development board. In order to monitor the bus traffic we used two CAN transceivers connected to the microcontroller's pins, one which was already available on the board and an external MCP2551 transceiver. Both transceivers are provided with a 120Ω bus termination. For implementing a CAN network with multiple nodes we added 3 MCP2551 CAN transceivers controlled by an Infineon AURIX TC277 development board representing nodes 2 to 4 from the setup presented in Figure 3. Nodes 1 and 5 from the same figure are connected to a Vector Breakout Box D62Y9 which is controlled from the Vector CANoe 8.5 environment running on a PC. This environment was used to provide the legitimate bus traffic from our experiments (which was collected from a real-world vehicle), and each node was set to output half of the original vehicle trace to the left and half to the right.

The relays used in our experiments are JQC-3F-5VDC relays which require a supply voltage of 5V, have three different contacts and can connect one of two different contacts at a time to the third contact based on their enabling pin status. In our setup there are two PCBs with 4 relays and each one of the 8 relays are controlled by an individual pin of the Bus Guardian connected to its enabling pin through a jumper wire. In order to provide the required voltage to both of the relay boards and the external CAN transceivers we used a power supply connected to the breadboard with an input voltage of 12V and an output voltage of 5V. In addition to the supply and ground lines from the power supply, the breadboard contains all the connections done with jumper wires between the

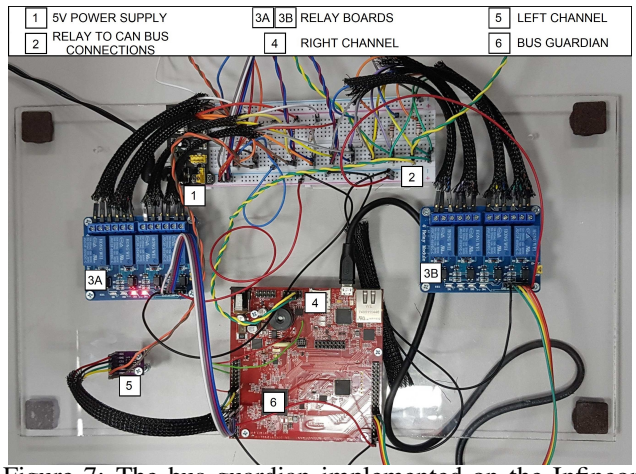


Figure 7: The bus guardian implemented on the Infineon TC297 and the relay blocks from our experimental model

CAN nodes and the relays required to disconnect the CAN lines linking any neighboring nodes and to add 120Ω between newly connected CAN-High and CAN-Low lines using the relay switches.

5.2 Collected in-vehicle traffic for the experiments

To create a realistic test-bed, we use real-world CAN bus traffic recorded in a high-end vehicle. The traffic is replayed on the bus in our experiments with the help of the CANoe environment via a CANCase device which assures accurate reproduction of the in-vehicle network traffic. The log file that we use accounts for 90 identifiers with cycles from 10*ms* up to 2*s* and a busload of around 40% on the 500Kbps CAN. Most of the identifiers however, have a periodicity between 10*ms* and 500*ms*. Figure 8 shows the arrival time for two IDs, one with a 20*ms* cycle (left) and the other with 40*ms* (right). The arrival time is stable, with very small variations (generally under 500μs) for each of the IDs. We also consider to look at the delays between consecutive frames, i.e., the inter-frame space (IFS). The IFS in the trace is critical since the Bus Guardian should parse frames by running the Bloom filter and distinguish between genuine and adversarial frames then retransmits frames to the other side of the bus.

For a 500Kbps baud rate the time for sending a frame on the bus varies roughly between 90μs to 270μs depending on the size of the data-field and the number of stuffing bits. The Bus Guardian must cope with these delays when classifying frames. For the existing bus traffic, the situation is more optimistic: by analyzing the trace only 0.5% of the IDs arrive with an inter-frame space lower than 200μs. This is expected at a bus load smaller than 50% (the bus is free at least half of the time). Figure 9 shows the inter-frame time for the first 2000 frames (left), only a few frames arrive with an inter-frame space lower than 200μs. On the right side of Figure 9

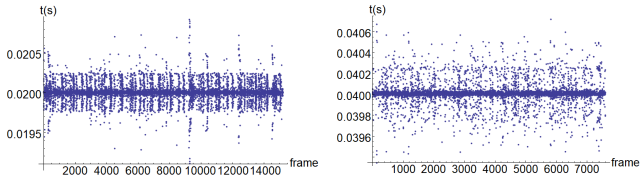


Figure 8: Collected in-vehicle traffic: variations in the arrival time for an ID at 20*ms* (left) and one at 40*ms* (right)

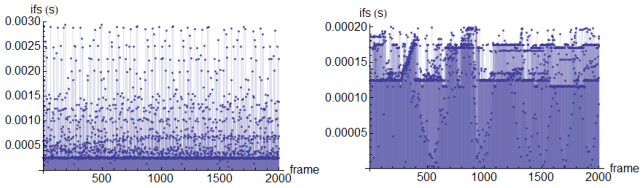


Figure 9: Collected in-vehicle traffic: IFS for the first 2000 frames (left) and same frames in the 0-200μs interval (right)

we depict the inter-arrival time for the first 2000 frames with an inter-frame space lower than 200μs, while this happens only rarely, and even if this is the case most of the frames leave a space of 100μs. The filters that we use do cope with these delays in the order of hundred micro-seconds. Nonetheless, traffic redirection must also work under these constraints which are not easy to meet and we are later forced to use a buffered retransmission to avoid losing frames.

6 Framework evaluation

In this section we follow two research directions. One of them is to determine how traffic filtering and redirection performs under specific attacks, such as fuzzing and DoS, the other is to determine how frame arrival time is affected by the defense mechanism. Notably, there are little side-effects and no frame loss due to the relay action. We also provide a more comprehensive analysis of the proposed framework.

6.1 Testbed overview

We provide an overview of our evaluation setup in Figure 10. Traffic collected from a high-end vehicle is replayed to the left and right sides of the network, i.e., half of the collected in-vehicle trace to each side by using the two channels of a Vector Breakout Box D62Y9 device connected as ECU_1 and ECU_5. We emphasize that the Vector Breakout Box is an industry-standard tool that perfectly mimics the behavior of the real-world vehicle bus and it is commonly used by the industry for system design and testing. Of course, due to potential damage and costs, it would have been uneasy for us to cut wires and mount relays inside the real car while the observed behavior on the in-vehicle network traffic would have been likely identical.

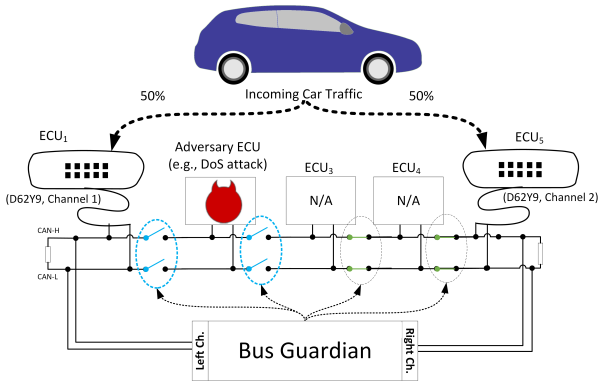


Figure 10: Schematic depiction of our experimental testbed

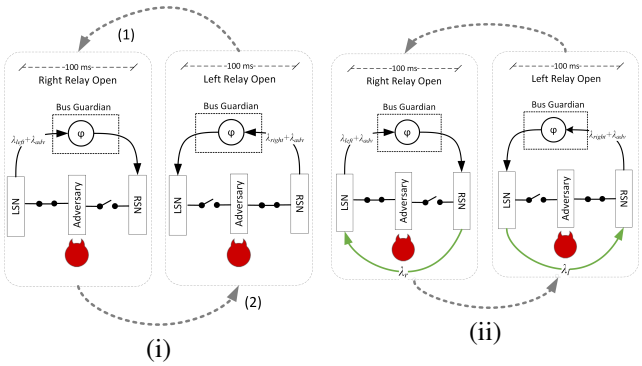


Figure 11: The two states of single-side traffic retransmission (i) and load-balanced retransmission (ii)

The second ECU from the left is designed to carry adversarial tasks, e.g., perform DoS or fuzzing attacks. The relays that surround him, encircled with dotted blue line, will alternatively open to the left and to the right, effectively disconnecting the adversary from the left-side network (LSN) or the right-side network (RSN). Traffic is filtered then redirected to the other side of the network by the Bus Guardian. We have additional relays in our setup toward the mounting point of ECU_3 and ECU_4 but these were not needed in our experiments, we kept them for potential future extensions.

6.2 Single-side traffic redirection in case of fuzzing attacks

Some basic tests of the relay impact on the adversary free bus can be found in Appendix A. We now add adversarial activity to the bus. Our framework is specifically designed to address DoS attacks. However, before evaluating DoS resilience, we also test the filters and relay behaviour in front of fuzzing attacks which is a common adversarial behavior that puts more stress on the filters (as the distribution of the IDs is randomized). In this type of attacks, the adversary injects random CAN frames that have random IDs and data fields. This kind of adversarial intervention is important because it can be employed in order to cause abnormal behaviour or learn how the system reacts to IDs that are not expected by the controllers.

Figure 11 (i) shows the testing strategy in case of single-side retransmission with the adversary in the middle of the network. Each $100ms$, or alternatively $1s$ in some experiments, the relays from the left or right sides are opened alternatively - the adversary is isolated either to the left or to the right side of the network. The traffic from the adversary side is filtered by the Bus Guardian and redirected to the other side. Under this second testing strategy with fuzzing attacks we consider traffic redirection only from the side which is affected by the intruder i.e., fuzzed by the adversary.

In this case the adversarial ECU is programmed to inject frames with random identifiers that are not part of the legitimate trace. For simplicity, we randomly replace frames from the legitimate trace with an attack frame that has a random ID that is not part of the legitimate ID set. The intrusions form roughly 50% of the trace. Since part of the genuine frames are now altered, they are not going to be transmitted to the left or right side since they are classified by our filters as intrusions.

Table 1 provides a summary in terms of: left transmitted count (LTC), left failed count (LFC), right transmission count (RTC) and right failed count (RFC). In case of the fuzzing attack the frame loss drops to half but this is expected since half of the frames on the bus are now adversarial frames and the LTC and RTC are also halved. By a careful analysis we observed that frame loss is not due to the relay actions but because of frame buffering. We determined that a 1 frame buffer reduces the frame loss to under 1% and a buffer of at most 32 frames reduces it to 0%.

6.3 Load-balanced retransmission in case of flooding and DoS attacks

Figure 11 (ii) shows the testing strategy for the load-balanced retransmission with adversarial activity in the middle of the bus. We specifically designed this experiment to respond to DoS attacks. Each $100ms$, the relays from the left or right sides are opened alternatively, isolating the adversary to the left or to the right side of the network. The traffic from the adversary side is filtered by the Bus Guardian and redirected to the other side, while traffic from the adversary-free side is directly transmitted to the other side (without filtering).

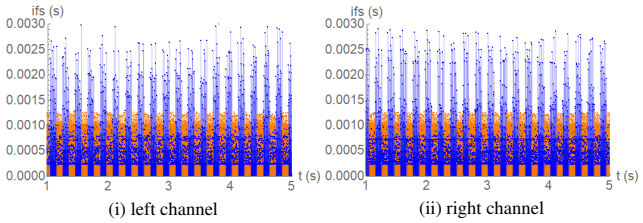
To begin with, we conducted four experiments in which the adversary injects an ID with high priority with a cycle time of $10, 1, 0.5$ and $0.3ms$ respectively. The time of the longest frame on the bus at 500Kbps is roughly $0.26ms$, so getting the cycle time closer to this value will result in a complete DoS of the bus. The larger $10, 1$ cycles where chosen for allowing us to test that retransmission works. When the attack gets closer to the $0.3 - 0.5ms$ range there are very

Table 1: Frame loss at various relay trigger rates with adversarial activity (fuzzing attacks) and single-side traffic redirection

Buffering	Retransmission	Filter size	Adversary	Relay Rate (s)	LTC	RTC	LFC	RFC	LF%	RF%
None	Single side	512	Fuzzing	1	57918	54444	1607	2031	2.7%	3.7%
	Single side	1024	Fuzzing	1	58111	54589	1527	1957	2.6%	3.5%
1 frame	Single side	512	Fuzzing	0.1	60258	57265	25	13	0.04%	0.02%
	Single side	1024	Fuzzing	0.1	60270	57271	16	30	0.03%	0.05%
32 frames	Single side	512	Fuzzing	0.1	60237	57234	0	0	0.0%	0.0%
	Single side	1024	Fuzzing	0.1	60261	57284	0	0	0.0%	0.0%

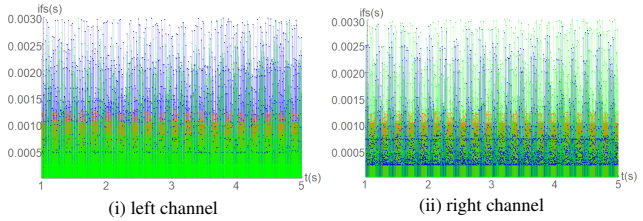
few legitimate frame to retransmit from the attacked side (more details concerning a full DoS are in the next paragraph). Figure 12 shows the inter-frame space on the left and right channel for $T_{relay} = 100ms$ during a flood with $1ms$ cycle time. Legitimate frames are printed in blue and adversarial frames are in printed in orange. Note that the left and right channels are asynchronous. In Figure 13 we also separate between left (green) and right (blue) side frames to show that legitimate frames occur on both sides. In Table 2 we summarize results for the 1ms flood (a partial DoS) which is more revealing for retransmissions since all frames can be successfully retransmitted. We determined that a buffer of one frame will make the failed retransmission to drop to less than 1%. To get retransmission errors down to 0% we need a buffer of 8 frames to the side that is free of adversarial interventions and a 32 frame buffer for the side where the adversary is present. The reason is that on the side where the adversary is present it is harder to find space on the bus for frame retransmission due to the higher busload. Thus the buffer must be capable to accommodate more frames. This happens for adversarial attacks at a rate of $1ms$, if the rate goes to $300\mu s$ frame retransmission becomes almost impossible on the adversary side regardless of the buffer size.

We now discuss the impact of a full DoS. Figure 14 provides plots for the case of an adversary that is programmed to loop and send high priority frames whenever there is room on the bus at $T_{relay} = 25ms$. This figure depicts the results in terms of busload and IFS on the left and right channels, contrasting legitimate (blue) frames with attack (orange) frames. The busload tops at the maximum of 500Kbps, i.e., 100% busload, and almost no legitimate (blue) frame manages to enter the attacked side, i.e., a full DoS. Legitimate frames (blue) may occasionally enter the channel when the relays are triggered due to brief disturbances in the adversary transmission. While the full DoS is more severe, the experimental outcome is in fact more simple to illustrate: the side where the adversary is isolated has no legitimate traffic (only orange frames) and once the adversary is shifted to the other side the recorded (buffered) traffic from the adversary-free side will be re-sent. We also note that, as CAN frames carry information from various sensors and actuators, it may not be necessary to replay all the recorded traffic but only the recent-most value for each ID. This allowed us to further simplify the buffering in our implementation since we only need to store and retransmit the last recorded value for each ID.



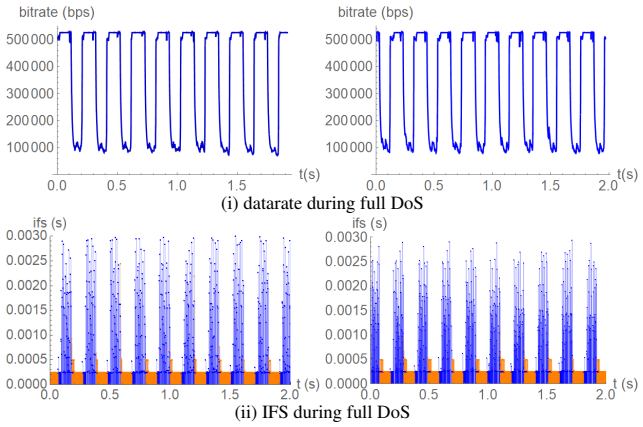
(i) left channel (ii) right channel

Figure 12: Interframe space for the left (i) and right channel (ii) for genuine (blue) and adversarial frames (orange) during 5 seconds of runtime ($T_{relay} = 100ms$)



(i) left channel (ii) right channel

Figure 13: Interframe space for the left (i) and right channel (ii) for legitimate frames from the left channel (green), legitimate frames from the right channel (blue) and adversarial frames (orange) during 5 seconds of runtime ($T_{relay} = 100ms$)



(i) datarate during full DoS

(ii) IFS during full DoS

Figure 14: Data rate (i) and inter-frame space (ii) during a full DoS on the left and right channels at $T_{relay} = 100ms$ (legitimate traffic in blue, adversary traffic in orange)

Figure 15 illustrates the inter-arrival time, denoted as dt, for an ID with $20ms$ cycle time during a full DoS with $T_{relay} = 25ms$. The left side where the ID originates is only

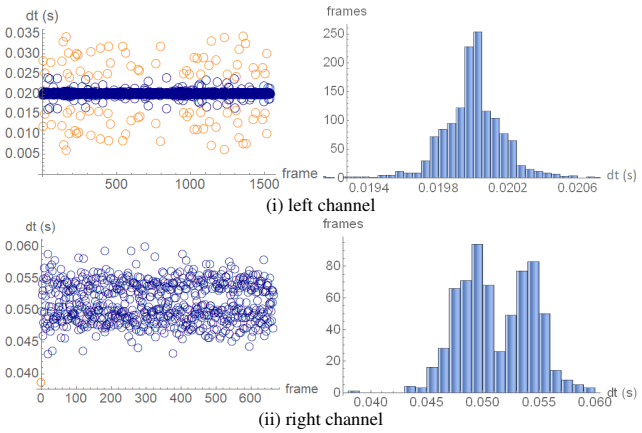


Figure 15: Arrival time for an ID with a 20*ms* cycle time on the left (i) and right channel (ii) at $T_{relay} = 25ms$ during a full DoS (ID belongs to the left side)

slightly impaired by the DoS and the mean arrival time remains the same. Delays of 25*ms* or more do occur, but the cycle time clearly remains centered around 20*ms*. On the right side we simply retransmit the recentmost value of the ID once the adversary is moved to the left side. This results in an inter-arrival time of $2 \times T_{relay}$, i.e., 50*ms* which can be easily explained as follows. When the right side is under a DoS (25*ms*), it is not possible to retransmit the ID from the left side (the bus is busy). When the adversary is moved to the left side, the recent-most value of the ID can be sent by the Bus Guardian to the right side - but this will be the only known value for another 25*ms* until the adversary is released from the left side. The same phenomenon was observed for all IDs which allows us to make a generalized statement: in case of a full DoS, the fastest cycle time that can be achieved with load-balancing on the side where the ID has to be retransmitted by the Bus Guardian is $2 \times T_{relay}$. As car diagnosis systems commonly report a time-out for a component after delays of several hundred milliseconds, we believe that the 50*ms* cycle time from our implementation (worst case under a full DoS that would otherwise lock the bus completely) should be sufficient for most messages to keep a vehicle functional. Finally, since the fastest messages on the CAN bus have a cycle time of 10*ms*, a $T_{relay} = 5ms$ should cope with any subsystem and is achievable with high performance solid-state relays.

As further insights on the impact of a full DoS on cars we present more results from a CANoe car simulation in Appendix B.

6.4 Relay influence on message arrival time

Since in-vehicle networks handle safety-critical messages for which the arrival time is critical, we also evaluate the effects of filtering and retransmission on the cycle time of legitimate IDs that come from the left or right side of the network. By

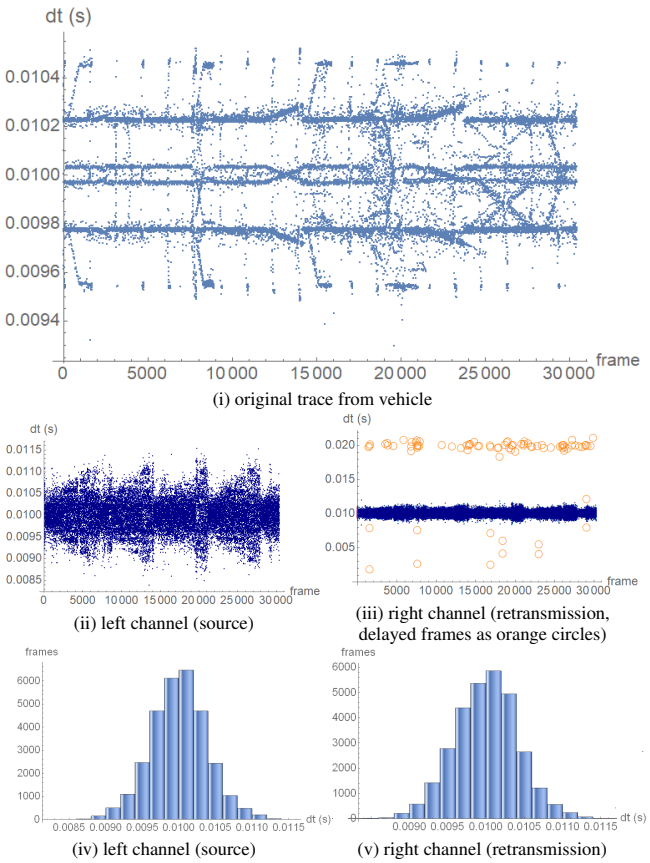


Figure 16: Message cycle time for a frame that originates on the left-hand side of the network, i.e., ID `0x2F`, on the: original trace (i), left channel (ii), right channel (iii) and histogram distribution on the left (iv) and right (v) channels

careful analysis of the recorded trace with CANARY's Bus Guardian active, we determined that the mean arrival time deviates by less than 1*ms* which should be acceptable for real-time demands.

Figure 16 shows the effect on ID `0x2F` coming from the left-hand side of the network. This ID has a cycle time of 10*ms*, the recorded delays between two consecutive occurrences of the ID is denoted as *dt* in the figures. The effects are minor on the left side of the network, a limited number of frames may be delayed or possibly lost when relays are triggered. But the cycle time remains steadily around 10*ms*. The plot for the right hand side of the network (where the ID is re-sent by the bus guardian) shows that the inter-arrival time may drift from the original 10*ms* to up to 20*ms*. These drifts occur on the right-hand side in case when frames are first lost due to relay triggering and then re-sent from the buffer. By computing the mean arrival time on the right side, we get 10.805*ms* compared to 9.99986*ms* on the left side and 9.99987*ms* on the right side in the original trace. This means that there are not many frames that drift from the expected cycle time of 10*ms* and is consistent with our estimation that

Table 2: Frame loss at various relay trigger rates with adversarial activity (DoS attack) and load-balancing

Buffering	Retransmission	Filter size	Adversary	Relay Rate (s)	LTC	RTC	LFC	RFC	LF%	RF%
None	Load balanced	512	DoS 1ms	0.1	110470	102111	8742	10782	7.9%	10.5%
1 frame	Load balanced	512	DoS 1ms	0.1	119124	112873	57	96	0.05%	0.09%
32 frames	Load balanced	512	DoS 1ms	0.1	119121	112939	0	0	0.0%	0.0%

7 – 10% of the frames require buffered retransmission that causes additional delays.

6.5 Immediate improvements: faster relays, more relays

Faster relays exist[2], e.g., solid state relays that operate in the $0.5-1ms$ range are common and have a higher life expectancy since they have no mechanical parts. But even the off-the-shelf relays that we used in the experiments ($5ms$ operation time) proved highly effective and eventually led to 0% frame loss. In case of the load-balancing algorithm, i.e., the worst case for an attack, the $100ms$ triggering rate results in little or no errors at all ($5ms$ out of $100ms$ means that relays impede a maximum of 5% from the total bus time). We now determine the theoretical upper-bound for the relay triggering rate based on relay operating time and bus parameters.

Maximum relay triggering rate. Since the bus is temporarily unavailable during the relay operation time (less than $5ms$ with the relays that we used), all frames that are scheduled for sending during this period will be automatically re-sent when the bus becomes available. We determined a theoretical upper-bound for the rate at which the relays can be triggered λ_{relay} depending on the relay operating time t_{relay}, frame arrival rate λ_{bus} and frame time t_{frame} as follows:

$$\lambda_{relay} \leq \frac{1-\lambda_{bus}\times t_{frame}}{t_{relay}}$$

The relation follows from the fact that for any fixed time interval T the bus has to be available to accommodate $\lambda_{bus}\times T$ frames. But during this period the bus will be unavailable for $\lambda_{relay}\times T\times t_{relay}$. This leads to the following condition $T-\lambda_{relay}\times T\times t_{relay}\geq\lambda_{bus}\times T\times t_{frame}$ which divided by T gives the upper bound for λ_{relay}. As a practical example, for a 500Kbps bus, setting $t_{frame}=200\mu s$, i.e., the average time of a CAN frame, and $\lambda_{bus}=2000$ fps, i.e., the usual 50% busload, having $t_{relay}=5ms$ we get $\lambda_{relay}=120$. That is, the relays can be triggered at $8.3ms$ cycles without losing frames on the bus. The upper side of Figure 17 summarizes these results by depicting the rate λ_{relay} as a function of relay operation time t_{relay} for a 500Kbps bus with $t_{frame}=200\mu s$. The lower side of Figure 17 extends this depiction for a frame arrival rate from 500 up to 4000 frames (a 50% bus-load for the highest CAN data-rate, i.e., 1Mbps). This is a theoretical bound, in practice,

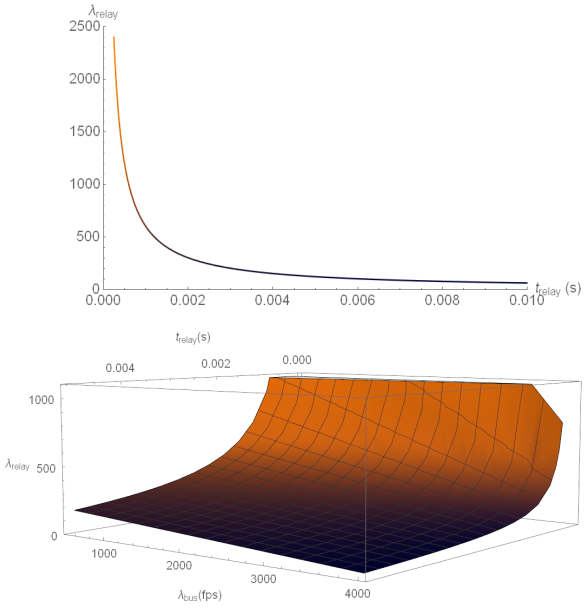


Figure 17: Maximum relay triggering rate λ_{relay} as function of relay operation time t_{relay} (up) and λ_{relay} as function of relay operation time t_{relay} and frame arrival rate λ_{bus} (down)

the CAN controller (or the upper-layer software) must be able to buffer frames in order to cope with the relay rate.

Adding more relays. Given the nature of in-vehicle networks, it is expected that adversaries will use predictable locations as entry points, e.g., the OBD port, the infotainment or telematics units, etc. Consequently, a small number of relays can be conveniently placed at key locations on the bus. However, for a comprehensive treatment, we cannot exclude the scenario where an adversary taps the bus at random locations. In this case adding more relays may increase the chances to isolate the intruder if he is unaware of the exact topology. The complexity of interwinding the relays with the ECUs increases exponentially. Concretely, for k controllers and n relays the number of placements corresponds to the number of $k+1$ *compositions* of integer n (the composition of an integer is a way of writing it as a sum of exactly k positive integers). Indeed, assuming that the relays can be placed anywhere, to the left and right of each ECU, there are $k+1$ bus segments starting from the left side of the first ECU, i.e., the first bus segment, to the right side of the last ECU, i.e., the $k+1$ bus segment. The number of k compositions of an integer n is given by the binomial coefficient $\binom{n-1}{k-1}$ and thus

[2] https://www.ni.com/ro-ro/innovations/white-papers/06/how-to-choose-the-right-relay.html

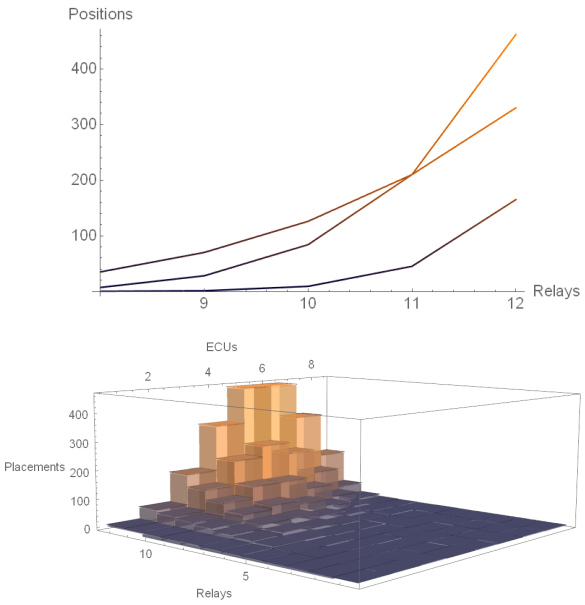


Figure 18: Relay placements for $n = 8..12$ relays on a $k \in \{4,6,8\}$ ECU network (up) and possible placements given $n \in [1,12]$ and $k \in [1..8]$ (down)

there are $\binom{n-1}{k}$ placements for n relays on the segments of a bus with k controllers. The upper side of Figure 18 shows the number of possible placements in a network with $k \in \{4,6,8\}$ controllers for $n = 8..12$ relays. The number quickly increases to 500 possible placements for 12 relays. The lower side of Figure 18 shows the number of possible placements as a function of the number of ECUs 1..8 and number of relays 1..12. The interwinding options quickly reach the order of several hundreds.

6.6 Further analysis: intermittent adversaries and multiple adversaries

Intermittent intrusions. Due to the efficient binary search, an intruder can be localized in roughly $\log_2(n)$ packets that are recognized by the filter as intrusions. An intruder may try to mount a low-rate attack or even send attack frames intermittently in order to avoid detection. If adversarial frames do not occur at some fix cycle time but occur independently in time (unaffected by each other), we can use the Poisson distribution to model the occurrence of at least one adversarial frame in a specific time interval T. That is, assuming that the time of occurrence for adversary frames follow a Poisson distribution with mean arrival rate λ_{adv} frames per second, the probability that k adversary frames occur in time T is:

$$\Pr[k] = \frac{(\lambda_{adv} \times T)^k e^{-\lambda_{adv} \times T}}{k!}$$

The probability to receive at least one adversary frame in time T immediately follows as $p_1 = 1 - Pr[0]$. This probability increases exponentially. Figure 19 depicts p_1 in relation with filtering time $T \in [0.010, 2]$ and $\lambda_{adv} \in [1, 10]$. An arrival rate of 10 frames per second corresponds to a $100ms$ cycle time which is a slow cycle time for in-vehicle network, cycle times usually go as low as $10ms$. But even if the adversary frames arrive at an average of 1 fps, the probability to receive one adversarial frame in $T = 4s$ is quite high at 98.16%. Parameter T can tuned in the localization algorithm, i.e., Algorithm 1, according to specific needs.

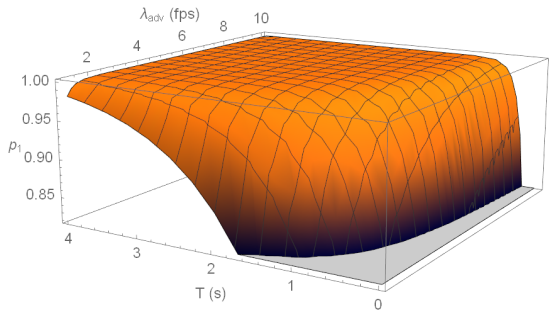


Figure 19: Probability to receive at least one adversarial frame p_1 during filtering time T at adversary rate λ_{adv}

Multiple adversaries. Our defense mechanism was designed to address a single adversary (or compromised unit) that taps the bus. We believe that this scenario covers most practical needs, but indeed, it may happen for adversaries to be present at multiple locations. For example, if adversaries are present at the two bus ends they may evade the localization mechanism since the attack comes from both sides of the bus and they may further cause a full DoS. As a direct extension to CANARY, to address this, we can add new transceivers to the bus as suggested in Figure 20. This would allow monitoring individual segments and removing them from the bus if needed, i.e., the Bus Guardian can disconnect any of the nodes in Figure 20 and individually monitor any of the nodes. However, this solution will increase the wiring complexity and implementation costs which may be unnecessary for most practical needs. A decision on the correct trade-off would require further investigations which due to obvious space constraints are unsuitable for the current work.

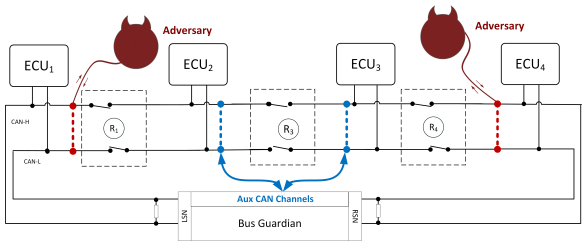


Figure 20: A multiple adversary scenario and multi-transceiver Bus Guardian

6.7 Remaining challenges and limitations

While our work is the first to propose an effective mechanism for defending the CAN bus against DoS attacks, the solution that we envision has several limitations which we now enumerate and leave as potential future work.

Re-certification and costs. Relays with automotive certification are common, in fact relays have been used inside cars long before the CAN bus. The topology induced by our modifications is still a bus compliant to ISO 11898 which should make certification feasible. Of course, certification will call for additional investigations on various issues, e.g., electromagnetic compatibility, which are out of reach for a first research communication. Porting the solution will induce additional costs related to wiring and components, i.e., the wires, relays, resistors and the bus guardian. However, the relay-resistor pairs should be placed only at critical positions around ECUs that can be easily corrupted or ports that are easy to access so that they can be isolated from the network. CAN buses usually connect less than a dozen ECUs, buses with 2-8 ECUs are common, and thus a dozen relays or so may be sufficient. The length of a CAN bus wire is typically between 3-15 meters and CANARY will require an amount of cable equal to the length of the bus to tap the two bus ends, one control wire and one power supply wire for each relay-resistor pair. At a minimum, CANARY may require similar modifications to existing after-market car access control systems, e.g., remote start-stop systems, that are connected to the CAN bus which hosts the electronic immobilizer of the car (usually the body control module) and which may also add relays to each door to facilitate remote access. By using relays, CANARY can disconnect (upon intrusions) the OBD port or the infotainment unit which are not vital for the car to function anyway. Due to physical difficulties in accessing random points of the in-vehicle wiring, adversaries will likely use predictable locations as entry points which can be efficiently protected by a small number of relays. Porting the full scale CANARY mechanism would indeed require more complex wiring. But according to recent estimates from the industry [13], common cars already have around 2.2 km of wires that connect 100 sensors and control units. Compared to these, CANARY should call only for a small additional fraction.

Further experiments. Our experimental model would greatly benefit from testing in a car-on-bench setup. The collected in-vehicle traffic that we use in the experiments perfectly mimics the behaviour of the bus from the real car but it is hard to predict the behaviour of the physical vehicle under an attack or when the active defense mechanism is triggered. Such tests will be particularly necessary if CANARY is placed behind some safety-critical subsystems and this type of evaluation would be required in case of re-certifications. Finally, a powerful attacker that has full knowledge of the car topology (including the wiring of the active defense mechanism) and which can tap the bus at any random location (possibly even in more than one location at the same time) would be extremely difficult or impossible to stop by the mechanism. In this respect, a systematic evaluation of all attacker entry points inside vehicles may provide better insights on the correct placement of the active relays.

Further applications of the proposed mechanism outside the automotive domain in areas where the CAN bus is also in use, e.g., industrial control systems, avionics, etc., may be also considered as future challenges.

7 Conclusion

Due to its bus topology and ID-oriented arbitration, the CAN bus remains vulnerable to message injections and in particular to DoS attacks. While cryptography can stop malicious messages from being accepted by legitimate nodes, it provides no solution against DoS attacks. The procedure presented in our work may help in this respect. Isolating the intruder and filtering/redirecting traffic provides an efficient mechanism that prevents the adversary from gaining full control over the bus. Since relays are cheap and the proposed algorithms easy to implement, there should not be many practical constraints in implementing the proposed solution or at least part of it. The relays that we use have an operation time of 5*ms* and changing them to faster solid-state relays will bring even better performances. Even with the relays from our setup, the frame-loss was reduced to zero by buffering frames during retransmissions. Notably, the relay action on the bus causes no frame loss since the clever error control mechanism of CAN sets room for transmitting the frames until the acknowledgement bit confirms successful reception. The few frames that were lost in our experiments were due to overlaps during retransmissions and the issue was solved by a rather small software buffer. The results from our experiments are on a real-world in-vehicle trace that was ported to our laboratory setup which proves the feasibility of practical use inside vehicles. While there are many works that focus on detecting intrusions on the CAN bus, there is still much work to be done in designing systems that can effectively prevent such intrusions. We hope that our work paves the way towards developing such systems.

Acknowledgments

We are grateful to the anonymous referees for their comments which helped us to improve our work. This work was supported by a grant of Ministry of Research and Inovation, CNCS-UEFISCDI, project number PN-III-P1-1.1-TE-2016-1317, within PNCDI III (2018-2020). http://www.aut.upt.ro/~bgroza/projects/presence/.

References

[1] Emad Aliwa, Omer Rana, Charith Perera, and Peter Burnap. Cyberattacks and countermeasures for in-vehicle networks. *arXiv preprint arXiv:2004.10781*, 2020.

[2] AUTOSAR. *Specification of Secure Onboard Communication*, 4.3.1 edition, 2017.

[3] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] Mehmet Bozdal, Mohammad Samie, Sohaib Aslam, and Ian Jennions. Evaluation of can bus security challenges. *Sensors*, 20(8):2364, 2020.

[5] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[6] Kyong-Tak Cho and Kang G Shin. Error handling of in-vehicle networks makes them vulnerable. In *Proc. ACM SIGSAC Conference on Computer and Communications Security*, pages 1044–1055. ACM, 2016.

[7] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium*, 2016.

[8] Wonsuk Choi, Kyungho Joo, Hyo Jin Jo, Moon Chan Park, and Dong Hoon Lee. Voltageids: Low-level communication characteristics for automotive intrusion detection system. *IEEE Transactions on Information Forensics and Security*, 2018.

[9] Tsvika Dagan and Avishai Wool. Parrot, a software-only anti-spoofing defense system for the can bus. *ESCAR EUROPE*, page 34, 2016.

[10] Stabili Dario, Marchetti Mirco, and Colajanni Michele. Detecting attacks to internal vehicle networks through hamming distance. In *IEEE Intl. Annual Conference-Infrastructures for Energy and ICT (AEIT)*, 2017.

[11] H. Giannopoulos, A. M. Wyglinski, and J. Chapman. Securing vehicular controller area networks: An approach to active bus-level countermeasures. *IEEE Vehicular Technology Magazine*, 12(4):60–68, Dec 2017.

[12] B. Groza and P. Murvay. Efficient intrusion detection with bloom filtering in controller area networks. *IEEE Transactions on Information Forensics and Security*, 14(4):1037–1051, April 2019.

[13] Ulrike Hoff and Dan Scott. Challenges for wiring harness development. *CAN Newsletter*, pages 14–19, 2020.

[14] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks–practical examples and selected short-term countermeasures. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2008.

[15] Abdulmalik Humayed and Bo Luo. Using id-hopping to defend against targeted dos on can. In *1st International Workshop on Safe Control of Connected and Autonomous Vehicles*, page 19–26. ACM, 2017.

[16] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata. CaCAN - centralized authentication system in CAN (controller area network). In *14th Intl. Conf. on Embedded Security in Cars (ESCAR)*, 2014.

[17] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. In *Procedings of PST (Privacy, Security and Trust)*, 2017.

[18] Mirco Marchetti, Dario Stabili, Alessandro Guido, and Michele Colajanni. Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms. In *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–6. IEEE, 2016.

[19] Tsutomu Matsumoto, Masato Hata, Masato Tanabe, Katsunari Yoshioka, and Kazuomi Oishi. A method of preventing unauthorized data transmission in controller area network. In *Vehicular Technology Conference (VTC Spring), 2012 IEEE 75th*, pages 1–5. IEEE, 2012.

[20] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. *Black Hat USA*, 2014.

[21] Pal-Stefan Murvay and Bogdan Groza. Dos attacks on controller area networks by fault injections from the software layer. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, pages 71:1–71:10. ACM, 2017.

[22] Michael Müter and Naim Asaj. Entropy-based anomaly detection for in-vehicle networks. In *Intelligent Vehicles Symposium (IV)*, pages 1110–1115. IEEE, 2011.

[23] R. Obermaisser and R. Kammerer. A router for improved fault isolation, scalability and diagnosis in can. In *2010 8th IEEE International Conference on Industrial Informatics*, pages 123–129, July 2010.

[24] Andrea Palanca, Eric Evenchick, Federico Maggi, and Stefano Zanero. A stealth, selective, link-layer denial-of-service attack against automotive networks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 185–206. Springer, 2017.

[25] H. Sivencrona, T. Olsson, R. Johansson, and J. Torin. Redcan/sup tm/: simulations of two fault recovery algorithms for can. In *10th IEEE Pacific Rim Intl. Symposium on Dependable Computing*, pages 302–311, 2004.

[26] Daisuke Souma, Akira Mori, Hideki Yamamoto, and Yoichi Hata. Counter attacks for bus-off attacks. In *International Conference on Computer Safety, Reliability, and Security*, pages 319–330. Springer, 2018.

[27] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2011.

[28] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. Frequency-based anomaly detection for the automotive can bus. In *World Congress on Industrial Control Systems Security (WCICSS)*, pages 45–49. IEEE, 2015.

[29] Haohuang Wen, Qi Alfred Chen, and Zhiqiang Lin. Plug-N-Pwned: Comprehensive vulnerability analysis of OBD-II dongles as a new over-the-air attack surface in automotive IoT. 2020.

[30] Marko Wolf, André Weimerskirch, and Christof Paar. Security in automotive bus systems. In *Workshop on Embedded Security in Cars*. Bochum, 2004.

[31] W. Wu, R. Kurachi, G. Zeng, Y. Matsubara, H. Takada, R. Li, and K. Li. IDH-CAN: A Hardware-Based ID Hopping CAN Mechanism With Enhanced Security for Automotive Real-Time Applications. *IEEE Access*, 6:54607–54623, 2018.

[32] Leiming Zhang, Yong Lei, and Qing Chang. Intermittent connection fault diagnosis for can using data link layer information. *IEEE Transactions on Industrial Electronics*, 64(3):2286–2295, 2016.

[33] Leiming Zhang, Fan Yang, and Yong Lei. Tree-based intermittent connection fault diagnosis for controller area network. *IEEE Transactions on Vehicular Technology*, 68(9):9151–9161, 2019.

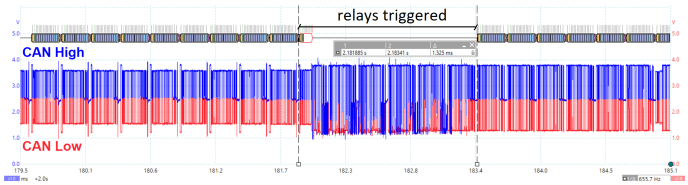


Figure 21: Oscilloscope plot of incoming frames during relay action

Appendix A - Testing relay impact on legitimate bus traffic

To get a baseline on performance, we first consider testing the relays and filtering mechanism in the absence of the adversary, i.e., in case of an adversary-free bus. Table 3 provides a summary of the frame loss due the action of relays with or without filtering in terms of: left transmitted count (LTC), left failed count (LFC), right transmission count (RTC) and right failed count (RFC). There is a slight increase of 1%-2% on the frame loss from the right side which can be explainable by possible differences in the relay blocks and slight asymmetries in the network traffic. Frame loss is not due to the relay actions but because of the buffering, i.e., the Bus Guardian does not manage to send frames as quickly as they arrive from one side to another. We solve this by proper buffering of incoming frames. Surprisingly, a 1 frame buffer reduces the frame loss to under 1%. For the case of a DoS attack at 1ms, as show in our experiments, a 32 frame buffer is needed. A buffer of this size is also sufficient for the adversary free bus as it reduces the number of lost frames to 0. The frame loss when using the Bloom filters is almost identical to the case in which no filtering is used, proving that filters don't affect performance. Triggering the relays will cause a brief disturbance on the bus and frames transmitted during this short period will be affected. Figure 21 depicts the effect of switching the relays on the bus. According to the data-sheet, the relays that we use have an operation time of 5ms, but as the plots show the actual time during which the bus is unavailable is much less, i.e., around 1.5ms. Conveniently, frames that are destroyed during relay switching are automatically retransmitted thanks to the clever design of CAN. Concretely, the sender node will get a transmission error and then automatically re-attempt to send the frame on the bus until it succeeds. The error counters are kept within acceptable margins as discussed next.

Impact on REC and TEC counters. Figure 22 shows the evolution of REC and TEC counters during 100ms and 1s relay triggering rate on one of the channels (the other channel looks identical). In both cases, the counters increase to at most 50 which keeps them in the Error Active state, i.e., the normal state of CAN nodes. There is still much room ahead until the Error Passive state (in which the nodes are still able to communicate but will not signal errors) and the counters are very far from the Bus off state. Nonetheless, the

Table 3: Frame loss at various relay trigger rates and buffer sizes without adversarial activity

Buffering	Retransmission	Filter size	Adversary	Relay Rate (s)	LTC	RTC	LFC	RFC	LF%	RF%
None	None	None	none	0.1	113673	106356	6856	8522	6.0%	8.0%
1 frame	Single side	1024	none	0.1	120333	114291	344	246	0.28%	0.22%
32 frames	Single side	1024	none	0.1	120687	114559	0	0	0.0%	0.0%

counters quickly decrease on the next successful transmissions/receptions so disturbances are short-lived and have little effects on the ECUs. We have also tried to trigger the relays at 10*ms* and the nodes still remained in the Error Active state while counters rarely increased up to 80.

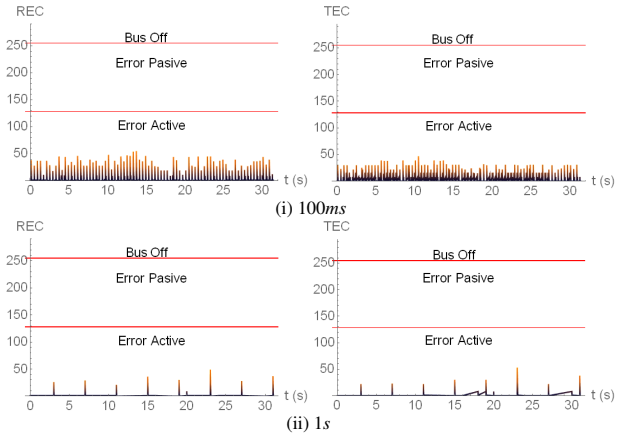


Figure 22: Evolution of REC and TEC counters with relays triggered at 100*ms* (i) and 1*s* (ii)

Appendix B - Further validations with a CANoe car simulation

The first attacks on the vehicular CAN buses were demonstrated by Hoppe et al. [14] as early as 2008 by using a CANoe simulation linked to a few car components on a bench. As an additional validation for the proposed solution, we use an existing car simulation from CANoe and show that (unsurprisingly) a DoS attack will completely halt all car functions while CANARY is able to fully alleviate the attack.

Figure 23 shows the default CAN demo from CANoe. Five IDs are broadcast related to car ignition, engine, ABS and gearbox having cycles of 20*ms* (for the ignition) and 50*ms* for the rest. Figure 24 (i) shows the plots with the recorded signals from the car in case of normal traffic. Then we mount a full DoS on the bus and in Figure 24 (ii) we show the effects: the entire bus is locked and incoming signals halt, i.e., the current value remains the last of the received values (which is incorrect). Then in Figure 24 (iii) with 50*ms* load-balancing the signals are almost identical to the adversary free bus. While there is still a long road ahead from this simulator

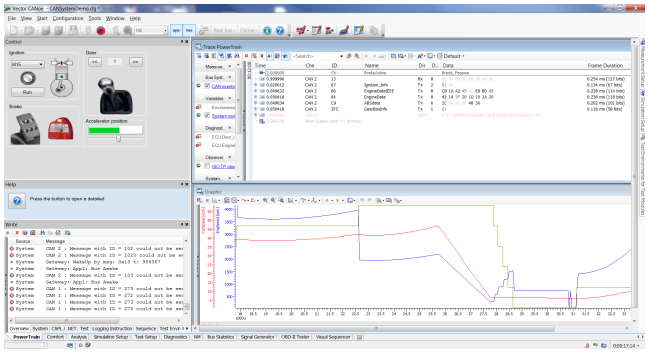


Figure 23: An existing in-vehicle CAN demonstration from CANoe

to a real-world demonstration, this at least proves that the attacks can be efficiently mitigated by CANARY within the simulation.

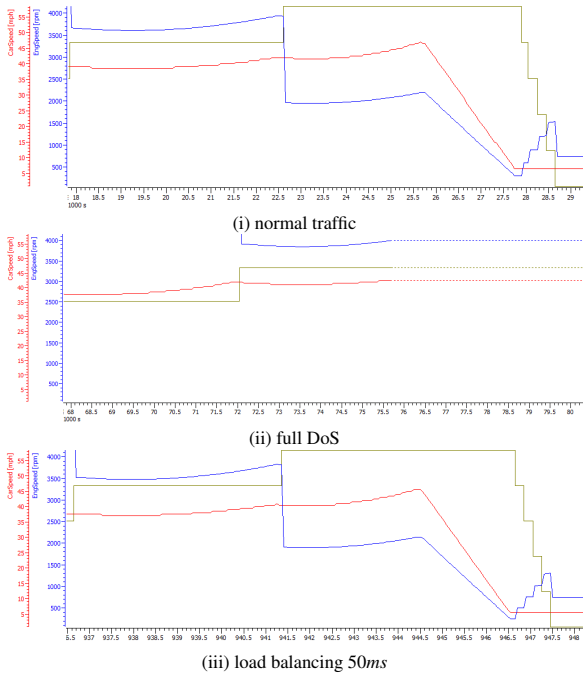


Figure 24: CAN bus signals as interpreted by the CANoe car simulator in case of: (i) normal traffic, (ii) full DoS and (iii) load-balancing at 50*ms*