CANvas: Fast and Inexpensive Automotive Network Mapping

Sekar Kulandaivel, Tushar Goyal, Arnav Kumar Agrawal, and Vyas Sekar,
Carnegie Mellon University

https://www.usenix.org/conference/usenixsecurity19/presentation/kulandaivel

This paper is included in the Proceedings of the
28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

CANvas: Fast and Inexpensive Automotive Network Mapping

Sekar Kulandaivel
Carnegie Mellon University
skulanda@andrew.cmu.edu

Tushar Goyal
Carnegie Mellon University
tgoyal1@alumni.cmu.edu

Arnav Kumar Agrawal
Carnegie Mellon University
akagrawa@alumni.cmu.edu

Vyas Sekar
Carnegie Mellon University
vsekar@andrew.cmu.edu

Abstract

Modern vehicles contain tens of Electronic Control Units (ECUs), several of which communicate over the Controller Area Network (CAN) protocol. As such, in-vehicle networks have become a prime target for automotive network attacks. To understand the security of these networks, we argue that we need tools analogous to network mappers for traditional networks that provide an in-depth understanding of a network's structure. To this end, our goal is to develop an automotive network mapping tool that assists in identifying a vehicle's ECUs and their communication with each other. A significant challenge in designing this tool is the broadcast nature of the CAN protocol, as network messages contain no information about their sender or recipients. To address this challenge, we design and implement *CANvas*, an automotive network mapper that identifies transmitting ECUs with a pairwise clock offset tracking algorithm and identifies receiving ECUs with a forced ECU isolation technique. *CANvas* generates network maps in under an hour that identify a previously unknown ECU in a 2009 Toyota Prius and identify lenient message filters in a 2017 Ford Focus.

1 Introduction

Recent efforts have demonstrated numerous vulnerabilities in automotive networks, particularly those that employ the CAN communication protocol. Although CAN is the prevailing standard for intra-vehicular communication due to its low cost and robustness, its broadcast nature has enabled many exploits initially exposed by the early work of Koscher et al. [20]. These exploits target the intra-vehicular CAN bus via either direct physical access [9, 20] or the remote exploitation of an ECU with existing direct access [26]. For the purpose of planning their well-known exploit [26], Miller et al. [25] analyzed the intra-vehicular networks of several vehicles, which revealed that the 2014 Jeep Cherokee was the "most hackable" based on its layout of ECUs. Once the authors gained access to the CAN via an exploited ECU, they simply had to discover which ECUs and real physical functions react to injected messages.

From these anecdotes, we can see that the set of ECUs and their inter-ECU communication channels determine the vulnerability of a vehicle's ECU network. Consequently, we argue that the automotive security world needs tools similar to Nmap [21], which are used to map the structure of modern IP networks. Such mapping tools prove useful in both attack and defense scenarios, such as identifying potentially malicious servers, attesting server configurations, and auditing firewalls by identifying available network connections. Analogously, with such a tool for scanning a car's network, we could (1) discover potentially malicious ECUs inserted through an attacker, (2) attest to the network configuration of ECUs over time, and (3) identify potential ECUs that are vulnerable to a recent type of attack (§2).

To aid in these scenarios, an ideal network mapper would require three main outputs: (1) the transmitting ECU for each unique CAN message, (2) the set of receiving ECUs for each unique CAN message and (3) a list of all active ECUs in the vehicle. To ensure that our network mapper is practical for our envisioned use cases, we ideally want our tool to be (a) *fast* to permit analysis of multiple vehicles at a time and limit the time a vehicle must be running and (b) *inexpensive* to avoid requiring costly equipment such as an oscilloscope or logic analyzer.

Unfortunately, extracting the necessary information to map these communication channels requires an unreasonable amount of effort. In the work by Koscher et al. [20], the authors analyzed the security of a vehicle's components by manually extracting ECUs to isolate and interact with them. This type of analysis requires significant time and effort or access to limited or proprietary information [25]. Second, obtaining vehicles for extended time and with permission to disassemble is costly and expensive. Considering new model years and over-the-air update capabilities, the frequency of analyzing an intra-vehicular network will quickly increase in time and cost requirements.

A key challenge we face in realizing this vision in practice is the lack of source information in CAN messages. CAN messages are "contents-addressed," i.e. messages are labeled based on their data and provide no indication to the message's sender. Another significant challenge in mapping a CAN bus is the broadcast nature of the CAN protocol; we cannot tell which ECUs have received a message. A CAN message is not explicitly addressed to its recipients, but a node can indicate it has correctly received a message (§3).

In this paper, we present *CANvas*, a system that demonstrates a *fast* and *inexpensive* automotive network mapper without resorting to vehicle disassembly (§4). Rather than require physically isolating each ECU, our key insight is to extract message information by re-purposing two observations from prior work:

- **Identifying message source** (§5): Prior work by Cho et al. [11] state that clock skew is a unique characteristic to a given ECU and thus build an intrusion detection system (IDS) that measures this skew from the timestamps of periodic CAN messages. Using this insight, we envision a mapper that computes clock skew per unique message and uses skew to group messages from the same sender. Unfortunately, due to shortcomings of their approach in our mapping context, we instead track the clock offset of two messages over time to determine their source.

- **Identifying message destination(s)** (§6): In another prior work [10], the authors propose a denial-of-service (DoS) attack that exploits CAN's error-handling protocol to disable a target ECU. Using this insight, the mapper could disable all but one ECU via this DoS attack and observe what messages are correctly received by the isolated ECU. However, due to shortcomings in their method w.r.t. our context, we develop a method to forcefully isolate each ECU and detect which messages the ECU receives despite the broadcast nature of CAN.

We implement the *CANvas* mapper on the open-source Arduino Due microcontroller with a clock speed of 84 MHz and an on-board CAN controller. We evaluate our mapper on five real vehicles (2009 Toyota Prius, 2017 Ford Focus, 2008 Ford Escape, 2010 Toyota Prius, and 2013 Ford Fiesta) and on extracted ECUs from three Ford vehicles. We show that *CANvas* accurately identifies ECUs in the network and the source and destinations of each unique CAN message in under an hour (§7).

Contributions and roadmap: In summary, this paper makes the following contributions:

- Designing an accurate message source identification algorithm that tracks a message's relative clock offset (§5);
- Engineering a reliable message destination identification method by isolating ECUs with a forced shutdown technique (§6);
- A real implementation that maps five real vehicles and extracted ECUs (§7) along with two real examples of motivating use cases for mapping (§2).

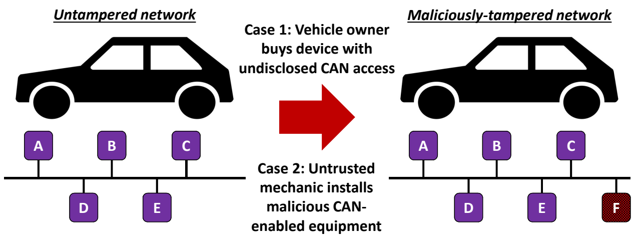

Untampered network
Case 1: Vehicle owner buys device with undisclosed CAN access
Maliciously-tampered network
A
B
C
D
E
Case 2: Untrusted mechanic installs malicious CAN-enabled equipment
A
B
C
D
E
F

Figure 1: A network mapper could discover potentially malicious ECUs from an untrusted party.

After defining the automotive network mapping problem and describing typical CAN bus setups (§3), we highlight the challenges of identifying message information via the CAN protocol and provide an overview of our approach (§4). Finally, we discuss open issues and limitations (§8) and related work (§9) before concluding the paper (§10).

2 Motivation

In this section, we discuss motivating scenarios for mapping in the context of intra-vehicular networks and describe characteristics of an ideal version of this security tool. To guide our design, we draw an analogy to Nmap [21], a popular network scanning tool that discovers hosts, services, and their interconnections in traditional computer networks. We identify a number of automotive-specific scenarios to illustrate the potential benefits of mapping, although this is not meant to be a comprehensive list.

Malicious ECU discovery: One main feature of Nmap is its ability to discover hosts, i.e. enumerate devices on the network. In the context of automotive networks, these "devices" are equivalent to a vehicle's ECUs. One major automotive cybersecurity concern (depicted in Figure 1) is the potential for an attacker to gain access to a physical network and add a new device [26], which could be a malicious ECU installed by an untrusted party or even by a vehicle owner who installs a CAN-enabled device purchased from an untrusted source. For an attacker that aims to insert this ECU into the network under the guise of a new equipment installation, the ECU could connect to the existing CAN bus and gain unfettered access to the CAN. If a defender performs a mapping through the vehicle's lifetime, they could verify changes to the network's ECUs. We provide an example of this scenario in §7 where we discover a previously unknown ECU that was installed in a modified 2009 Toyota Prius.

Continuous network attestation: Another popular use for Nmap is performing security audits to identify changes to a network [21]. Where such audits would identify new servers or a modification in a server's open ports, an audit in an automotive context could identify changes to the ECUs and their communication channels. With future over-the-air update capabilities, automakers will install new firmware or activate

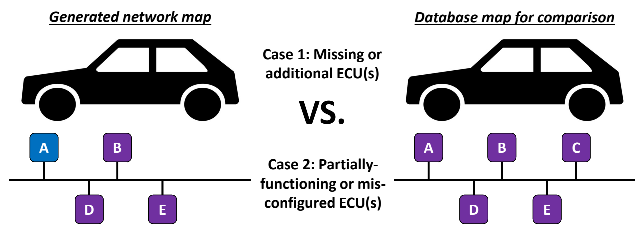

Generated network map
Database map for comparison
Case 1: Missing or additional ECU(s)
VS.
Case 2: Partially-functioning or mis-configured ECU(s)
A
B
C
D
E

Figure 2: A network mapper could compare a generated map to one found on an online database. Differences between these maps could identify changes in ECUs (Case 1) and/or their message configurations (Case 2).

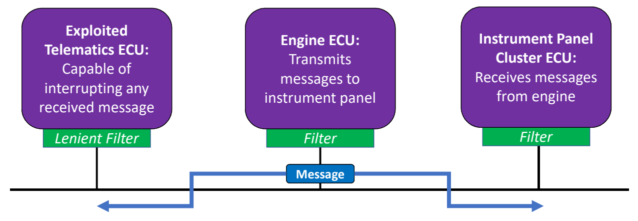

Exploited Telematics ECU: Capable of interrupting any received message
Lenient Filter
Engine ECU: Transmits messages to instrument panel
Filter
Instrument Panel Cluster ECU: Receives messages from engine
Filter
Message

Figure 3: Assume that only the instrument cluster should receive messages from the engine. If the exploited telematics ECU is able to receive engine messages, then an attacker [10] could shutdown the engine ECU via the exploited telematics ECU.

different features in an existing vehicle. As the configuration of the network can change over time, it is necessary for vehicle owners to attest to the vehicle's expected configuration. If a user does not own the vehicle over its lifetime as in the *malicious ECU discovery* scenario, we could implement an online database where vehicle owners could upload the outputs of their network maps for comparison against maps generated from brand-new vehicles. Any differences from the expected maps could indicate malicious or accidental network changes.

Lenient filter identification: Nmap is often used to perform port scanning to identify open ports [21], which are potential vulnerabilities. These "open ports" are analogous to the set of CAN messages that an ECU is able to correctly receive, which we refer to as the ECU's message-receive filter. Now consider an attacker who aims to target a safety-critical ECU (e.g. engine ECU) as depicted in Figure 3. If gaining direct access to the engine ECU proves infeasible, the attacker could access an ECU that is less critical and potentially has access to remote networks (e.g. telematics ECU). Using the ECU shutdown attack as discussed in recent work [10], our attacker can shutdown the engine ECU by gaining control of the telematics ECU and reprogramming it; the attacker simply needs to receive a message from the victim ECU to target it. To combat this, a defender could perform a similar analysis via network mapping and implement filters that prevent the message from being received to limit the damage from a potential shutdown attack. We provide an example of this scenario in §7 where we discover lenient message-receive filters in a 2017 Ford Focus.

Goals: In designing a useful automotive network mapper, we must consider a few requirements that we impose to ensure practicality in the context of our motivating scenarios:

Fast: First, we want to limit the amount of time a vehicle (and its ECUs) are turned on. Also, a fast mapping process will make it more practical for a user to verify the state of their vehicle's network after a repair. Considering these reasons, we aim to achieve a mapping time of under one hour.

Inexpensive: To permit greater access to the mapper, the mapper should consist of relatively inexpensive components and should avoid expensive tools, such as oscilloscopes and logic analyzers. We aim to limit costs to under \$100; a low-cost approach to network mapping will permit more users for our system.

Vehicle-agnostic: Every vehicle has a different setup of ECUs on the CAN bus and can employ additional features of the CAN protocol. For our mapper to be practical, it must work on many makes and models of vehicles as well as rely on only standard CAN features.

Minimally-intrusive and non-destructive: One extreme approach for mapping a vehicle requires physical disassembly, which is a very intrusive process and requires a great deal of access to the target vehicle. We should limit this access to simply connecting to a diagnostics port on the vehicle. If a CAN bus is not exposed on this port, we describe a method of getting access to these buses with minimal disassembly in §8. Additionally, the mapper must not cause any permanent damage to the vehicle or its network. Any of our methods can put the network into a non-ideal state (warning lights on, gear shift disabled, etc.), but as long as restarting the vehicle undoes any imposed errors, we satisfy this constraint.

3 Problem Overview

In this section, we give a concrete problem formulation for the network mapper and discuss technical challenges. We preface with some necessary background on CAN to understand the overall problem and mapping challenges.

3.1 CAN basics

To better understand the message information we hope to gain using a network mapper and the associated challenges in acquiring that information, we first discuss some necessary background on how the CAN protocol works.

CAN in modern vehicles: All vehicles produced for the U.S. market in 2008 and after are required to implement the CAN protocol for diagnostics purposes [4]. Many vehicles will often employ either one, two or three CAN buses. In the event of three CAN buses, it is likely that the vehicle has one bus for powertrain components (engine, transmission, etc.),

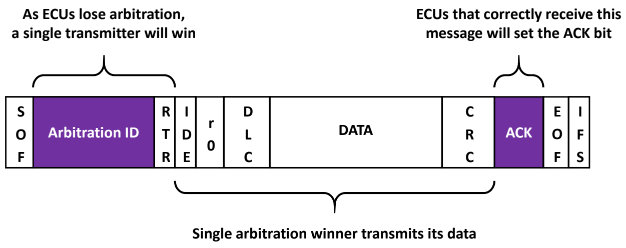

As ECUs lose arbitration, a single transmitter will win
ECUs that correctly receive this message will set the ACK bit
SOF
Arbitration ID
RTR
IDE
r0
DLC
DATA
CRC
ACK
EOF
IFS
Single arbitration winner transmits its data

Figure 4: Each CAN frame is transmitted on the bus bit-by-bit. A single transmitter wins arbitration and will listen to receiving ECUs during the ACK slot.

one bus for infotainment components (radio, etc.) and another for body components (door controller, headlights, etc.). These CAN buses are usually exposed through a vehicle's On-Board Diagnostics (OBD-II) port as detailed in §8.

Message broadcast bus: The CAN protocol [13–15, 32, 33] is defined as a message broadcast bus, which means that ECUs in the network communicate with each other via *messages*. These ECUs are connected to a shared network where all ECUs can receive all transmissions. Due to the nature of this broadcast bus, it is not possible to send a message to a specific ECU. In the CAN protocol, after a message is broadcast to the network, devices that correctly receive this message will acknowledge their reception.

Typical CAN setup: A typical CAN setup for a vehicle will grant each ECU with a unique set of IDs and each message will be labeled with an ID, which is then transmitted onto the bus. An ECU will be responsible for a subset of the message IDs seen in the network, and each message ID will only be sent by a single ECU. Each message is queued by a software task, process or interrupt handler on the ECU, and each ECU will queue a message when the message's associated event occurs.

CAN frame format: Each CAN message from an ECU uses its assigned message ID (interchangeably referred to as the ID or the arbitration ID), which determines its priority on the CAN bus and may serve as an identifier for the message's contents. These messages are transmitted and received at the physical layer by an ECU's CAN controller as CAN data frames in the format depicted in Figure 4. The key fields in the CAN data frame, as relevant to our work, are: the start-of-frame (SOF) bit, the arbitration/message ID field, the acknowledgement (ACK) slot and the end-of-frame (EOF) bits.

All ECUs in the network with a queued message simultaneously start to transmit their message at the same time. During the arbitration ID field, all but one ECU will eventually stop transmitting based on CAN's arbitration resolution. Once an ECU has won arbitration on the bus, it will be the only sender and transmit the remainder of the CAN data frame until the ACK slot. During the ACK slot, the transmitter now becomes

Scenario	*Enum.*	*Src. map*	*Dest. map*
Malicious ECU discovery	✓	✓	
Continuous network attestation	✓	✓	✓
Lenient filter identification	✓		✓

Table 1: Mapping requirements for motivating scenarios

a receiver on the bus and all other ECUs in the network that correctly receive a message will simultaneously send a dominant bit on the network. This slot is then followed by the EOF and the inter-frame space (IFS).

Message arbitration: To understand how ECUs communicate on the CAN bus, it is necessary to discuss the CAN message arbitration process [13–15, 33]. CAN is designed to support collision detection and bit-wise arbitration on message priority to allow higher-priority messages to dominate the network. The arbitration of these messages is performed on the message ID field of a data frame, where a lower ID indicates a higher priority. This priority-based arbitration process sets a 0-bit as dominant and a 1-bit as recessive. Since a 0-bit is dominant, a message with a lower ID will get priority on the CAN bus and will be sent before a message with a higher ID that is queued at the same time.

3.2 Mapping requirements

Unlike most traditional packet-switched networks, CAN messages do not have fields that identify the message's source and destination(s), which makes the mapping problem difficult. To develop a mapper that will aid in the motivating scenarios of §2, we formulate three required outputs for *CANvas*:

ECU enumeration: The importance of enumerating ECUs is evident in all of our provided scenarios as seen in Table 1; enumeration highlights new or absent ECUs. Note that in all of these scenarios, it is *not* necessary to know an ECU's type (engine, transmission, etc.) or its functionality (fan speed control, tire pressure sensing, etc.).

Formally, let E_i denote ECU i in a given vehicle that contains n total ECUs that are CAN-enabled. For each E_i in a vehicle's set of ECUs, $E_{1:n}$, the ECU is responsible for sending a specific set of m messages labeled with a unique arbitration ID from the set, $I_{E_i,1:m}$. This set of IDs is unique to E_i and no other ECU in the network should send the same ID. Given a CAN traffic dump from a vehicle, *CANvas'* enumerator should determine the number of ECUs, n, and differentiate between them to determine the set of ECUs $E_{1:n}$ for that particular vehicle.

Message source identification (§5): In the *malicious ECU discovery* and *continuous network attestation* scenarios, changes to the set of transmitted messages for each ECU can pinpoint a potentially malicious reconfiguration. This means that a goal for our mapper is to map each message ID to its source ECU.

Formally, given a CAN traffic dump from which we extract the set of uniquely-ID'd messages where l is the number of total unique message/arbitration IDs and $I_{1:l}$ is the set of unique IDs, we should be able to determine which ECU E_i sent each unique message. This step is very closely related to ECU enumeration; once we know which ECU E_i that an arbitrary ID I_j originates from, we can produce a mapping of the ID to its source ECU, $I_j \in E_i$. Using this mapping, we can group the IDs with a common source ECU and complete our enumeration.

Message destination identification (§6): For the *continuous network attestation* scenario, we want to look for changes in what messages an ECU correctly receives as this could also indicate a potentially malicious reconfiguration. This component plays an important role in the *lenient filter identification* scenario, where an attacker could shutdown an ECU from an unintended message recipient.

We assume that at least one ECU in the network will correctly receive each message in the network. Formally, given the set of l unique IDs, $I_{1:l}$, from a traffic dump, we should be able to determine the set of ECUs, $E_{1:k}$, that correctly receive a message labeled with an arbitrary I_j. The expected output of this component should be a mapping of an ID to its destination ECUs, $I_{j,E_{1:k}}$.

3.3 Challenges in an automotive context

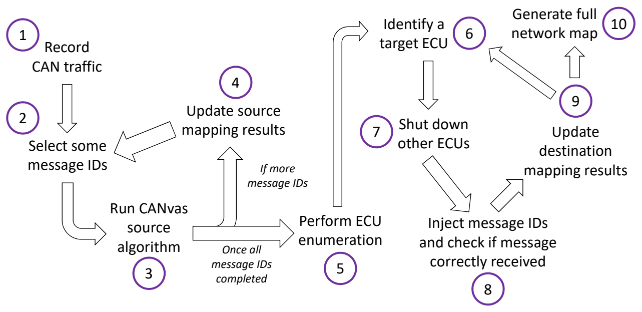
However, to achieve these mapping goals, we encounter two major challenges to determining the source and destination ECUs for CAN messages: (a) CAN lacks identifying source information and (b) CAN implements a broadcast protocol, which naturally implies that all nodes receive all messages. We discuss how we approach and solve these challenges in §5 and §6.

Lack of source information: If a message sent from ECU E_i has no identifying information, then it is non-trivial to determine that E_i sent the message. Since CAN messages are considered to be "contents-addressed" [13–15, 33], the value of the message ID is only related to the message's data and priority. In practice, the source ECU has no weight in determining the chosen arbitration ID for a particular message.

Broadcast protocol: We define destination as an ECU that correctly receives a message at the CAN controller level. Unfortunately, determining which ECUs correctly receive a message is non-trivial as an ECU connected to the CAN bus cannot detect which of its messages are received by certain ECUs. The ACK bit itself only indicates that some ECU has received the message, not which particular ECU(s) have received it. As multiple ECUs will set the ACK bit when a message is received, we cannot simply use this ACK bit to determine the set of ECUs $E_{1:k}$ that receive an arbitrary I_j.

4 System Overview

In this section, we provide a high-level overview of the *CANvas* network mapper.



1
Record CAN traffic
2
Select some message IDs
Run CANvas source algorithm
3
4
Update source mapping results
If more message IDs
Once all message IDs completed
Perform ECU enumeration
5
Identify a target ECU
6
7
Shut down other ECUs
Inject message IDs and check if message correctly received
8
9
Update destination mapping results
Generate full network map
10

Figure 5: *CANvas* obtains source mapping results by step 4. Then, it will enumerate the ECUs in step 5. *CANvas* then performs destination mapping and generates the full map at step 10.

4.1 High-level idea

***CANvas* mapping overview:** We split *CANvas* into two main components: (1) a source mapper and (2) a destination mapper. As detailed in §3, we satisfy our ECU enumeration requirement by simply using the output of source mapping. For (1), we passively collect several minutes of CAN traffic. After an offline data collection, the source mapper uses the data to produce a mapping of each unique CAN ID to its source ECU and subsequently, by grouping IDs with a shared source, a list of all active source ECUs on the bus. For (2), we interact with the network directly and perform an online analysis to determine message destination. *CANvas* systematically isolates each ECU, which will most likely cause the vehicle to enter a temporary error state that the user can reset.

User capabilities: We assume that the user has access to the OBD-II port of the vehicle and can connect the *CANvas* mapper directly to the CAN bus with the ability to read and write to the bus. We also assume that the vehicle even has a CAN bus and that the standard CAN protocol is implemented, which most vehicles will reflect [11]. The user should also be able to transition the vehicle's ignition switch between the LOCK, ACC and ON positions as the user will have to reset the vehicle after each iteration to exit the error state.

Scope and evasion: We assume that the vehicle does not implement countermeasures that will alter timing of message transmissions, potentially to prevent intruders from identifying transmitting ECUs. We also assume that the vehicle cannot identify a maliciously-triggered error and prevent intruders from abusing CAN's error-handling protocol to shutdown an ECU. The vehicle should not employ an intrusion detection system capable of preventing an ECU suspension. We further discuss adversarial evasion and other scenarios for bus configurations in §8.

4.2 *CANvas* workflow

The workflow of *CANvas* involves four major steps seen in Figure 5:

1. *Data collection:* The CAN pins of the OBD-II port provide access to the frame-level signals and the message-level data. *CANvas* will read this traffic for several minutes and timestamp each received message. From this traffic, we will obtain the set of unique message IDs observed in the network and a set of timestamped data for each ID.
2. *Source mapping:* With the list of all unique message IDs, the source mapper will extract the timestamped CAN traffic for each ID and determine which IDs share the same source as detailed in §5. To do this, we select two message IDs and run their CAN traffic through our comparison algorithm, which will determine if the two IDs originate from the same ECU.
3. *ECU enumeration:* Using the set of matching ID pairs from source mapping, the enumerator will simply group pairs that originate from the same ECU. The output of this step will be a list of ECUs and associated source IDs.
4. *Destination mapping:* Using the ECU enumeration output, the destination mapper will identify the ECUs that correctly receive a given message ID. *CANvas* will isolate a target ECU by performing a shutdown on all other ECUs, which we discuss in §6. Once an ECU is isolated, we inject all unique observed message IDs and determine which ECUs receive the message.

5 ID Source Mapping

In this section, we describe an approach to map each CAN message to its source.

Intuition: Due to the absence of source information in a CAN message, we must rely on some uniquely identifying characteristic that can be tied to a particular ECU. Following observations from prior work [11, 29] and CAN documentation [2, 14], we consider *clock skew* as a candidate fingerprinting mechanism. In particular, time instants for in-vehicle ECUs rely on a quartz crystal clock [14], and we can use the relationship between these clocks to identify a transmitting ECU. We first define the following terms considering two clocks, $\mathbb{C}_1$ and $\mathbb{C}_2$:

- **Clock frequency**: The number of cycles per true second, e.g. if $\mathbb{C}_1$ operates at 16kHz, then $\mathbb{C}_1$ cycles 16,000 times every one true second.
- **Relative clock offset**: The difference in time reported by $\mathbb{C}_1$ and $\mathbb{C}_2$, e.g. if $\mathbb{C}_1$ reports time t_1 of 4.1ms and $\mathbb{C}_2$ reports t_2 of 4.2ms, their offset $O_{\mathbb{C}_1,\mathbb{C}_2}$ is 0.1 ms. Where only one clock is denoted for relative offset, the other clock is the clock of the receiving node.
- **Relative clock skew**: The difference in clock frequencies of two clocks, or the first derivative of offset w.r.t. true time, e.g. if $\mathbb{C}_1$ operates at 16kHz and $\mathbb{C}_1$ operates at 16.1kHz, their skew $S_{\mathbb{C}_1,\mathbb{C}_2}$ is 100Hz. Where only one clock is denoted for relative skew, the other clock is the clock of the receiving node.

Two clocks with a relative clock offset of 0 are considered to be *synchronized*, and two clocks with a nonzero relative clock skew are said to "skew apart," or have an increasing relative offset over time [2]. Since the CAN protocol does not implement a global clock, it is considered to be unsynchronized as each ECU relies on its own local clock.

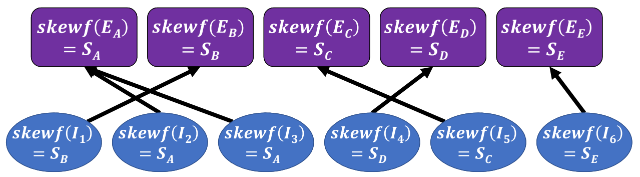


Figure 6: *CANvas* aims to cluster message IDs with a similar relative skew or offset.

Observation 1: ***The clock offset and skew of an ECU relative to any other ECU is distinct, thus providing us with a uniquely identifying characteristic for source mapping.***

High-level idea: To map each unique ID to its transmitting ECU, we break the module into two steps as Figure 6 illustrates: (1) computing either the skew $skewf(I_i)$ or offset $offsetf(I_i)$ of each ID I_i and (2) then clustering IDs with the same skew or offset where each cluster denotes a distinct source ECU, E_{src}. This module outputs a mapping of source ECUs to their set of source IDs. The main input to this module is a passively-logged CAN traffic dump, which contains entries in the form of $(I_i, t_{I_i,n})$ where I_i is the ID of the message and $t_{I_i,n}$ is the timestamp of the n^{th} occurrence of I_i.

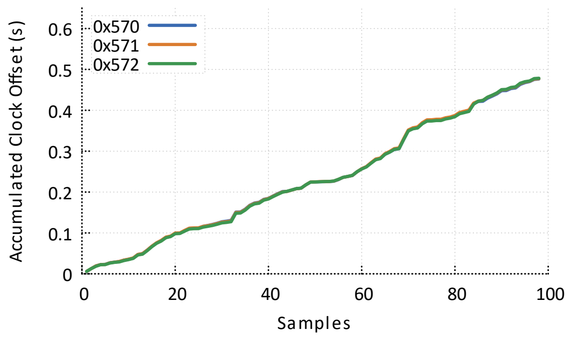
5.1 Prior work and limitations

Cho et al. [11] use clock skew as a means of building an intrusion detection mechanism to identify an attack by a malicious ECU. Specifically, this work uses timestamps of periodically-received message IDs and posit that IDs with the same skew originate from the same ECU.

To compute the clock skew of an ID I_i over time, Cho et al. [11] perform the following steps: (1) compute I_i's expected period, μ_{T_i}, (2) compute the offset, O_i, by subtracting the expected timestamp (using μ_{T_i} from the actual timestamp), (3) take the average of O_i over a batch of N messages, (4) add $O_{i_{avg}}$ to an accumulated offset, O_{acc}, and (5) then compute the skew, S_{I_i}, by taking the slope of O_{acc} versus time. This work uses the Recursive Least Squares algorithm to minimize the errors. After every batch of N messages, O_{acc} increases by O_i, where k is the k^{th} batch. From this plot, since O_i should be constant, their formula for skew w.r.t. batch size sets S_{I_i} to:

$$skewf_i^{Cho}(N) = \frac{kO_i}{kN} = \frac{O_i}{N} \tag{1}$$

As an extension to this work, Sagong et al. [29] note that the skew of Equation 1 varies significantly based on N and use



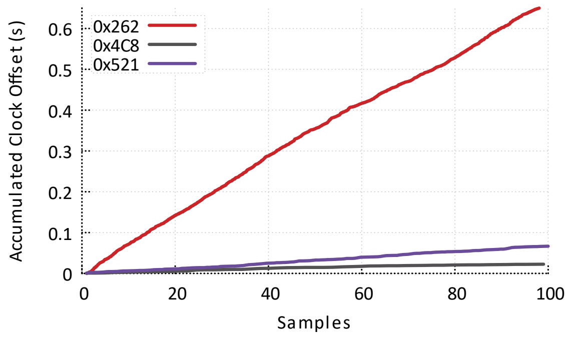


Figure 7: E_A transmits IDs 0x570, 0x571 and 0x572 at the same period and E_B transmits IDs 0x262, 0x4C8 and 0x521 at different periods. Above are plots of accumulated clock offset vs. samples for E_A and E_B using the algorithm by Cho et al. [11].

an updated formula for S_{I_i} w.r.t. batch size:

$$skewf_i^{Sagong}(N) = N \cdot \frac{kO_i}{kN} = O_i \tag{2}$$

Using data from a real vehicle, we now highlight a key limitation of Equations 1 and 2. Consider Figure 7: (1) E_A is the source of IDs 0x570, 0x571 and 0x572, which share the same period and (2) E_B is the source of IDs 0x262, 0x4C8 and 0x521, which each have different periods. In Figure 7, we use $skewf_i^{Cho}$ with $N = 20$ to plot the skew of all six IDs; $skewf_i^{Sagong}$ produces similar results. We can correctly conclude from Figure 7 that the IDs of E_A originate from a single ECU. However, from Figure 7, we will incorrectly conclude that IDs 0x262, 0x4C8 and 0x521 originate from three *separate* ECUs. Our analysis and experiments shed light on why these approaches fail–the skew value they compute is *period-dependent*.

As such, we update Equations 1 and 2 w.r.t. period T and batch size N:

$$skewf_i^{Cho}(N,T) = \frac{kO_i}{kTN} = \frac{O_i}{TN} \tag{3}$$

$$skewf_i^{Sagong}(N,T) = N \cdot \frac{kO_i}{kTN} = \frac{O_i}{T} \tag{4}$$

To potentially fix this issue, we can attempt a strawman that is not dependent on period or batch size.

$$skewf_i^{Straw}(N,T) = TN \cdot \frac{kO_i}{kTN} = O_i \tag{5}$$

Ideally, accounting for both batch-size and message-period (essentially batch-period, NT) using Equation 5 should give us a unique value that is common only among IDs from the same ECU. We apply Equation 5 for all I_i of a vehicle, and we attempt to establish distinct groupings of the computed skew for each ID, S_{I_i}, which would identify which I_i share the same E_{src}.

Unfortunately, this is a difficult task as I_i from the same E_{src} still do not have similar skews. This issue is further demonstrated as S_{I_i} varies across different data dumps or even segments of a given dump. Upon further inspection, we find that the measured S_{I_i} is affected by the deviation in an ID's period. This deviation in the period, σ_{p_i}, is attributed to sources of "noise", i.e. the period of a given message varies due to scheduling, queuing and arbitration delay. We also find that some I_i produce S_{I_i} with more deviation than others and produce widely-varying skew values, thus making our straw-man solution an unlikely candidate for source mapping.

Observation 2: ***We need a method of extracting the clock skew invariant that is: (a) independent of the period of I_i and (b) robust to noise in the period.***

5.2 Pairwise offset tracking

Issue with straw-man: In Equation 5, it is clear that, relative to the receiver, this "skew" function computes offset rather than true skew. Following our definitions in §5, a plot of relative offset over time should either be linearly increasing or decreasing if there is a nonzero skew between two clocks. In other words, if the relative skew between an E_{src} and the receiver is non-zero, then we should observe a gradual change in the offset. However, previous work [11, 29] fails to capture this change in offset over time.

Relative offset as a unique identifier: As mentioned in §5, clock offset and skew of an ECU relative to another ECU is distinct. We must note that the clock offset measured from one ID, I_1, of an E_{src} may not be the same as the offset of another ID, I_2, from E_{src}. If the initial transmission time of I_1 differs from that of I_2, the O_{I_1} could not equal O_{I_2}. Rather, the invariant here is the *change in relative offset*, ΔO_{I_i}; as the skew of E_{src} relative to the receiver is a constant nonzero value, the ΔO_{I_i} will be a constant nonzero as well (the derivative of offset is skew).

By measuring this change in offset, we can uniquely identify an E_{src}, but we must ensure our method of extracting this change in offset is (a) robust to a noisy period and (b) period-independent. To address the issue of noise in the period of I_i, p_{I_i}, we compute the relative offset between a pair

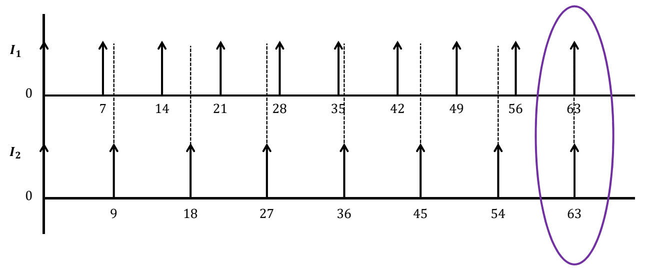


Figure 8: Timeline of two message IDs, I_1 and I_2, that have periods, p_1 = 7ms and p_2 = 9ms. Their hyper-period occurs every 63ms.

of two different IDs denoted by O_{I_1,I_2}. By performing this computation pair-wise, we expect O_{I_1,I_2} to have a deviation of approximately 0 if $I_1, I_2 \in E_{src}$ as the sources of noise for I_1, I_2 should mostly be shared. In reality, this deviation is very close but not exactly equal to 0; we define a practical threshold for this deviation in §7.

With a pairwise approach to computing O_{I_1,I_2} and the requirement for a period-independent approach, we face a new challenge: determining at what point in time to observe this relative offset regardless of the period of I_1 or I_2.

Observation 3: ***Compute offset at the* hyper-period *of I_1 and I_2, or the least common multiple of their periods.***

Measuring offset at the hyper-period: To guide our algorithm design for computing $\Delta O_{I_1,I_2}$ over time, we first model two periodically-transmitted IDs observed on the CAN bus. Consider two IDs, I_1 and I_2, from the same E_{src} which transmit at a period of p_1 and p_2, respectively. For example, let p_1 be 7ms and p_2 be 9ms. For now, we assume that the relative offset between I_1 and I_2 is 0. This offset should not change over time as they originate from the same E_{src}. To accurately compute the relative offset of these two IDs, O_{I_1,I_2}, we must select a time instant when the expected offset should also be 0: the hyper-period of I_1 and I_2, or the least common multiple of p_1 and p_2. As seen in Figure 8, this time instant occurs at 63ms, or the $lcm(7,9)$. Therefore, by computing the difference between the times reported from I_1 and I_2 every 63ms, or the hyper-period of I_1 and I_2, we can track the value of relative offset over time. If this relative offset is a nonzero constant, then the two IDs originate from the same ECU.

With an input of several minutes of timestamped CAN data to Algorithm 1, we can track relative offset over the timeline of two message IDs. Note that each timestamp has a noise component that stems from scheduling, queuing and arbitration delay. To compare whether two message IDs originate from the same ECU, we first assume that they are sent by separate ECUs. The two message IDs, I_1 and I_2, have periods, p_1 and p_2, and they have relative offsets, O_{I_1} and O_{I_2}. We draw the following relationships between these variables:

Algorithm 1 Pairwise offset tracking

```
 1: function PAIRWISECOMPARE($I_1, I_2, log_{I_1}, log_{I_2}$)
 2:     $p_1 = \lfloor \text{ComputeAveragePeriod}(log_{I_1}) \rfloor$
 3:     $p_2 = \lfloor \text{ComputeAveragePeriod}(log_{I_2}) \rfloor$
 4:     $n = \text{lcm}(p_1, p_2)/p_1$
 5:     $m = \text{lcm}(p_1, p_2)/p_2$
 6:     $pos_{I_1} = 0$, $pos_{I_2} = 0$
 7:     $\Delta_{I_1,I_2} = [\,]$
 8:     while $pos_{I_1} < \text{len}(log_{I_1})$ and $pos_{I_2} < \text{len}(log_{I_2})$ do
 9:         $\Delta_{I_1,I_2}$.append($log_{I_1}[pos_{I_1}] - log_{I_2}[pos_{I_2}]$)
10:         $pos_{I_1} += n$
11:         $pos_{I_2} += m$
12:     return true if $\sigma(\Delta_{I_1,I_2}) < threshold$ else false
13:     end function
```

- $p_2 = lp_1$, where l is the ratio of the periods.
- $O_{I_2} = jO_{I_1}$, where if j=1, then both IDs sent by same ECU; otherwise, they were sent by different ECUs.
- $n = ml$, where $LCM(n,m) = l$ as depicted in Figure 8.

By computing the difference between every n occurrences of I_1 and every m occurrences of I_2, which occurs at the *hyper-period* of I_1 and I_2, we produce the following equation:

$$O_{I_1,I_2} = (mp_2 + O_{I_2} + i_2) - (np_1 + O_{I_1} + i_1)$$

We find that when we average the result of the above equation across the entire data log, the expected value is 0 if I_1 and I_2 originate from the same ECU. In reality, this value is close to 0 due to the deviation of a message's period. From experimental data, we define a threshold of 1ms for the change in relative offset, where a value under the threshold will classify the two IDs with the same source ECU. Using this approach to revisit the setup described in Figure 7, we correctly conclude that IDs 0x262, 0x4C8 and 0x521 originate from the same ECU.

Practical challenges: While the above approach is correct, there are a number of other practical challenges we need to address to ensure accurate mapping:

1. *Large hyper-period*: Consider a hyper-period that is "large", or on the scale of several minutes, e.g. the hyper-period of $p_1 = 980$ms and $p_2 = 5008$ms is over 20 minutes. Since we only extract one relative offset value per hyper-period, we would need hours of CAN traffic to produce a valid result. To ensure that our mapper is fast, this length of traffic log is unreasonable; we want to produce a full network map in under an hour. Fortunately, with a pairwise approach, we can choose to *not* attempt a comparison when the hyper-period is large; for example, if we assume that the E_{src} of I_1 also transmits another ID, I_3, where the hyper-period of I_1 and I_3 is small, we can still determine that $I_1, I_3 \in E_{src}$.
2. *Large period deviation, σ_{p_i}*: In early experiments, we discovered messages that had a large measured σ_{p_i} (we define

large as $\sigma_{p_i} \geq 0.1 p_i$) and, at first, assumed that these messages were either aperiodic or sporadic (aperiodic with a hard deadline). However, upon closer inspection, we noticed that these messages appeared to be periodic in nature. We observed three different patterns that altered the measured σ_{p_i}: (1) the period simply had a large σ_{p_i}, (2) periodic messages would occasionally stop transmitting for some time, and (3) periodic messages were missing their deadlines. With a large enough σ_{p_i}, the deviation would conceal an inconstant ΔO_{I_i} and make it difficult to detect a mismatch. We experimentally find that a σ_{p_i} greater than 8% of p_i results in incorrect outputs. Therefore, *CANvas* will choose to test I_i on the following cases when its σ_{p_i} is under a defined threshold, which we set to $\sigma_{p_i} \leq 0.08 p_i$ from our experiments.

3. *Periodic messages that occasionally stop:* We find that some I_i are periodic and will stop transmitting for some time, causing a measured σ_{p_i} to be large. To combat this issue, we only perform pairwise offset tracking when the given message was actively transmitting. In the event we compare two I_i that both occasionally stop and there is no overlap of active transmissions, we then rely on our pairwise approach to match the I_i to another ID from the same E_{src}.
4. *Messages that miss deadlines*: For some I_i with a large σ_{p_i}, we observe two different inter-arrival times: p_i and $2p_i$. When a task on one of the ECUs misses its deadline and cannot produce a message on time, it will skip that cycle and transmit during the next cycle [2]. Thus, when a deadline is missed, we will observe an inter-arrival time of $2p_i$. In this situation, there are two options: (1) perform relative offset tracking on portions of the log when deadlines are not missed or (2) interpolate the missed inter-arrival times. If a message frequently misses its deadline, the first option is not viable. To interpolate a missed arrival time, we insert a psuedo-entry in the traffic log with a timestamp equal to the average of the preceding and the following timestamp.

Factors for mapping time: For source mapping, we experimentally find that 30 minutes of data provides enough samples for larger hyper-periods to map accurately. While this stage has static run-time, the variation in time requirements will be dependent on the number of observed messages IDs. The more message IDs that exist in the network, the longer the mapping time takes; vehicles with more message IDs take longer to complete mapping due to an increase in message-pairs. However, to further reduce mapping time, mapping messages with small periods requires much less traffic data. To save additional time if necessary, it is recommended to reduce the traffic log length for high-frequency messages. Also, if there are few large periodic messages or if those messages are not relevant for whatever reason, the length of the initial traffic log can be reduced as necessary instead of the recommended 30 minutes.

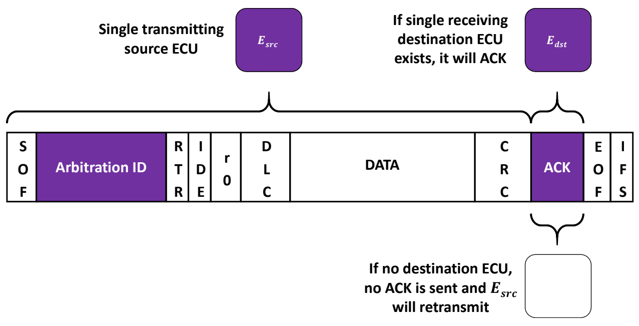

Single transmitting source ECU
E_{src}
If single receiving destination ECU exists, it will ACK
E_{dst}
SOF
Arbitration ID
RTR
IDE
r0
DLC
DATA
CRC
ACK
EOF
IFS
If no destination ECU, no ACK is sent and E_{src} will retransmit

Figure 9: Observing ACK bit with single ECU in network.

6 ID Destination Mapping

The goal of the destination mapping module is to accurately associate each ID with its set of receiving ECUs. The key consideration here is to maximize the accuracy of our mappings within our defined time constraint. In this section, we describe an approach to map each CAN message to one or more destination ECUs as defined in §4 and then present a systematic procedure that reliably determines which messages an ECU correctly receives.

6.1 Problem formulation

Intuition: As defined in §4, the destination(s) of a particular CAN message are those ECUs who correctly receive a given message. Despite the broadcast nature of CAN, if an ECU does not correctly receive a message, it will not set the ACK bit; however, if other ECUs receive this message, they will set the dominant ACK bit. Unfortunately, an ACK observed by the transmitting ECU only means that *some* active ECU correctly received the message. Therefore, with multiple active ECUs in the network, we cannot identify which ECUs were the destination for a given message.

Consider the scenario in Figure 9 where there was only one active destination ECU, E_{dst}, in the network other than the transmitting source ECU, E_{src}. For each message sent by E_{src}, a set ACK bit (performed only by E_{dst}) would indicate that only one ECU received the message: E_{dst}. Thus, in this scenario, E_{src} could simply inject all possible I_i and detect which messages have a set ACK bit. The major challenge here is identifying a method of isolating an E_{dst} and "removing" all other ECUs from the network. We define the bare minimum of "removal" as preventing an ECU from participating in the acknowledgement process.

Observation 4: ***Our idea for performing this removal is to transition an ECU into an error-state that prevents it from setting the ACK bit for any message.***

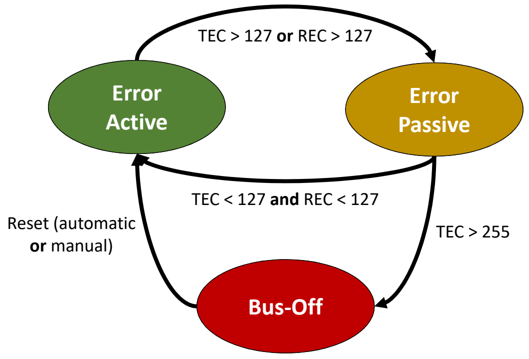


Figure 10: CAN transitions between three error states: error-active, error-passive and bus-off.

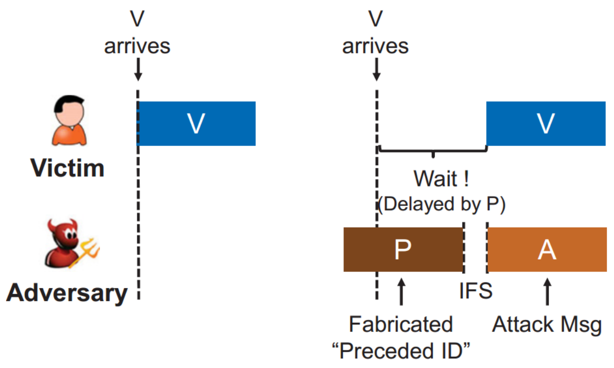


Figure 11: Injecting a fabricated message to impose a bus-off [10].

We now introduce the error-handling mechanism for CAN [2, 10], which follows the state diagram in Figure 10. Each ECU has two error counters: one for errors detected as a receiver (the Receive Error Counter, or REC), and another for errors detected as a transmitter (the Transmit Error Counter, or TEC). The TEC increments much faster than the REC as the transmitter is more likely to be at fault; the TEC increments by 8 while the REC increments by 1. If a message is received correctly, the error counter will decrease by 1. We describe the three CAN error-states and, under what conditions, the ECU will transition:

- **Error-active:** When an error is detected by an ECU in error-active, it will transmit an active error flag, or 6 dominant bits, that destroy the bus traffic. When either the TEC or REC increments past 127, the ECU transitions to error-passive.
- **Error-passive:** When an error is detected in error-passive, the ECU transmits a passive error flag, or 6 recessive bits, that do *not* destroy the bus traffic. Once the TEC or REC increases above 255, the ECU goes to bus-off.
- **Bus-off:** In this state, the ECU effectively removes itself from the network; it will not transmit anything onto the bus, including setting the ACK bit.

Thus, it is evident that we can isolate an ECU by transitioning all other ECUs to the bus-off state.

6.2 Limitations of prior work

Imposing bus-off state: The challenge in transitioning an ECU to bus-off is to determine what kind of error to produce and how to produce it. We look to previous work [10] that aims to shutdown an ECU for the purpose of an attack. The authors aim to shutdown an ECU by causing an error in the target ECU. By exploiting the error-handling protocol in CAN, where bus-off effectively removes an ECU from the network, they choose to increment the error counter of a target by causing a bit error. This error occurs when a transmitting ECU reads back each bit it writes; when the actual bit is different, the ECU invokes an error.

Since only one ECU is expected to win the bus arbitration as detailed in §2, the authors point out that two winners would potentially cause a bit error. For example, suppose that the victim ECU transmits a message with ID 0x262. If the attacker ECU also transmits ID 0x262 at the exact same time as the victim, both ECUs will win arbitration. However, to ensure that the victim has a bit error, the attacker's message will set its DLC, or data length count, to 0 (most practical messages contain at least some data). After a sufficient number of these attack messages, the victim ECU will transition into the bus-off state.

The main challenge here is synchronizing the attack message with the victim message so they both enter arbitration simultaneously. Their insight as depicted in Figure 11 is to inject a message of higher priority around the time when the victim should transmit. The higher priority message will block the victim until the bus is idle, where it will then transmit. The attacker will load its attack message immediately after the higher priority message is transmitted, thus allowing both the victim and attack message to arbitrate simultaneously. Since there is noise in the true transmission time of the victim's first attempt at transmitting, there is a chance that the attacker will need to make multiple attempts to cause an error. The number of injection attempts needed to cause a single bit error, κ, is defined as the following where $\mathbb{I}$ is a confidence attack parameter (high parameter value means higher confidence in attack), σ_{p_v} is the jitter deviation of the victim's period, and S_{bus} is the speed of the bus in Kbps:

$$\kappa = \left\lceil \frac{2\sqrt{2}\mathbb{I}\sigma_{p_v} S_{bus}}{124} \right\rceil \tag{6}$$

The authors state that only one of these injections is needed to cause a bit error if setting $\mathbb{I} = 3$ and at most 2 if setting $\mathbb{I} = 4$, given that the period deviation is 0.025ms.

Straw-man limitations: Suppose we used the above ap-

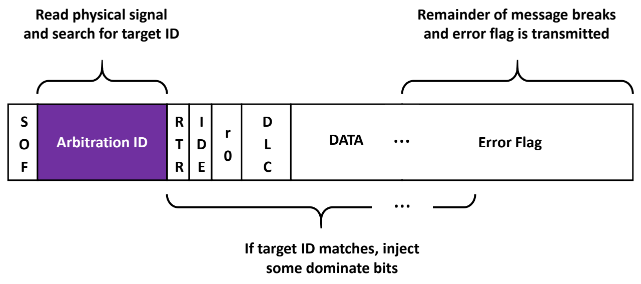


Figure 12: *CANvas* identifies target message by end of ID field and injects dominant bits during the DATA field.

proach to cause a bus-off in a real vehicle. Unfortunately, in sample traffic dumps from two real vehicles, the smallest deviation that we observed was approximately 0.15ms. Using the equation given by Cho et al. [10], the number of preceded message injections per error is 8 when the period deviates by at least 0.205ms; if 8 injections are required, any successful bit error would be undone by successful message transmissions. We look at available traffic logs used in the works by Miller et al. [24]. For this traffic log, the majority of the messages have a period deviation over 0.205ms. In other words, assuming the best case scenario of 0.15ms, we would need to inject at least 6 higher-priority messages, or preceded messages, for a bus speed of 500Kbps. Considering that each successful transmission by the victim ECU decrements the TEC by 1, we would effectively only increase the TEC by 2 with each successful attack (instead of the expected 8). Since the majority of messages have a period deviation greater than 0.205ms, it is highly unlikely to use this method for isolating an ECU.

Observation 5: ***We need a method of transitioning an ECU into the bus-off state that is reliable and robust even when the period deviates by more than 0.025ms.***

6.3 Forced ECU isolation

High-level idea: To map each unique ID to its set of destination ECUs, we break the module into two steps. We repeat these two steps for all n ECUs in the network. The first step is to isolate the target ECU and shut off all others by transitioning the non-target ECUs to the bus-off state. As there are n ECUs in the network, we will need to "bus-off" $n-1$ ECUs for each ECU, i.e. we will need to perform the bus-off at least $n(n-1)$ times. Once we isolate an ECU, we then inject the set of all I_i and observe which messages have a set ACK bit, thus identifying the set of I_i where the target ECU is an E_{dst}.

Inducing a direct bit-error: Isolating an ECU via the bus-off method requires a quick and effective approach. Since we are not limited to operating through the interface of a CAN controller, we can directly view the CAN frames in real-time via digital I/O pins. However, since we are using a microcontroller that operates at the same voltage of the CAN controller, we do not operate at the true CAN voltage. Instead, we tap directly between the interface of the Arduino's CAN controller and the CAN transceiver, where we can safely access the bus data. At this junction, we observe that the data on the line is within the Arduino's voltage and contains the full data frame, including SOF, ACK and EOF bits. With this access to the full data frame rather than just the components of the CAN message, we can directly induce an error on the bus and thus achieve the bus-off attack as seen in Figure 12.

Observation 6: ***By reading the ID of the message in real-time, we can choose to attack any ID by simply driving a dominant bit to the CAN transceiver.***

Note that the bus-off method requires attacking a message ID every time it occurs until the ECU enters the bus-off state. However, in the event that a message has a very long period, the time to perform the bus-off will not satisfy our speed requirement. As such, we can employ the result of *CANvas'* source mapping component by identifying the ID with the smallest period per ECU and attacking just that ID. In practice, we have found that every real ECU we have encountered has at least one ID that operates under 100ms. Thus, this approach makes the destination mapping component of *CANvas* fast.

Determining message receive filter: Now that we can isolate a single ECU in the network, we can simply inject all messages in the observed ID space and determine which messages are correctly received by the ECU. However, to view the ACK bit at the network level, which is not visible to the user, the obvious option is to use a logic analyzer. As this does not satisfy our requirement for low-cost mapping, we seek an alternative. We observed that if a message is sent to a single ECU and it does not correctly receive the message, the transmitter will re-attempt to send the message until it is received correctly. As such, if we transmit a message and see a continuous stream of the same ID from our transmitter, then we may conclude that the message ID is not received by the isolated ECU.

Practical challenges of mapping a real vehicle: Since our approach to destination mapping involves shutting off multiple ECUs at a time, we encounter a couple of challenges in a real vehicle setting: (1) ECUs that auto-recover and (2) ECUs that are persistently active. We now define these scenarios and provide a detailed approach to addressing these practical challenges:

1. *ECUs that auto-recover*: In our earlier experiments, we performed a simple experiment to verify the potential of an isolation method. We attempted to transition all ECUs in the network to the bus-off state by shorting the CAN bus pins, which would effectively cause a transmit error for all ECUs and force them into bus-off. However, after removing the short, we saw that some CAN messages were still transmitted onto the network, clearly indicating that some

ECUs left the bus-off state. We find that these ECUs would wait a predefined amount of time before re-transmitting again as these ECUs were critical to the vehicle's powertrain (engine, hybrid, etc.) [14]. In this situation, we would transmit a portion of the injected messages onto the bus and then re-isolate our target ECU when a non-target starts to transmit again. This approach is only reasonable for recovery times on the scale of seconds.

2. *ECUs that are persistently active*: Out of the set of ECUs that did auto-recover, we also noticed that one ECU seemed to be persistently active. In other words, there appeared to be no delay between a transition into the bus-off state and the next transmission from the ECU. Upon closer inspection, we found that this ECU would auto-recover only after 128 occurrences of 11 recessive bits [27]. In this situation, we must "hold" the bus open by constantly transmitting false messages from our device to trick the recovering ECU into thinking that the bus is still active.

Factors for mapping time: For destination mapping, the run-time is dependent on the number of ECUs and increases with more ECUs. We acknowledge the potential of long run-times for vehicles with 70+ ECUs if all were CAN-enabled. To combat this, we suggest performing the bus-off on the ID with the smallest period per ECU to reduce the time attributed to achieving ECU isolation. Also, for our two vehicles, all observed IDs were active when the vehicle was simply in ACC rather than ON so there may be no need to crank the engine per ECU.

7 Evaluation

In this section, we show that *CANvas*:

1. identifies an unexpected ECU in a '09 Toyota Prius,
2. identifies lenient message-receive filters in a '17 Ford Focus,
3. produces a sound source mapping of two real vehicles and accurately identifies the source of approximately 95% of all I_i in the network and a complete destination mapping with an isolation technique that is 100% reliable,
4. successfully demonstrate our forced ECU isolation on three extracted ECUs,
5. and produces source mapping of three additional vehicles.

Setup and methodology: Our experimental setup includes five real vehicles and several synthetic networks to demonstrate the above benefits. Below is a brief description of the *CANvas* hardware implementation, five real vehicles and our synthetic network of real ECUs:

- *Mapping device:* To interface with a CAN bus, our mapping device consists of three components: an Arduino Due microcontroller with an 84 MHz clock and an on-board CAN controller, a TI VP232 CAN transceiver, and a 120Ω resistor. To gain direct write access to the bus for destination mapping, we connect a digital I/O pin to the driver input pin of the transceiver.
- *'09 Toyota Prius and '17 Ford Focus:* The Prius contains eight original ECUs that transmit on a single CAN bus at 500 kbps. The Focus contains eleven original ECUs that transmit on three CAN buses at varying speeds; as our model of the Focus is the standard edition, only the high-speed 500 kbps bus has more than one active ECU. We obtain ground truth for our experiments by physically taking apart the car and gaining direct access to the ECUs by splicing directly into the CAN wires as seen in Figure 13. We use a paid subscription to both Toyota and Ford's mechanics' manuals [3, 6] for guidance on disassembly of vehicle components. Due to the non-destructive design of *CANvas*, our interaction does not impose any permanent errors to the vehicle.
- *'08 Ford Escape, '10 Toyota Prius and '15 Ford Fiesta* We obtain CAN traffic from three additional vehicles for testing only our source mapper, as we did not have permission to inject data. We use data from the '09 Prius and '17 Focus to partially confirm our source mapping output without disassembling these vehicles.
- *Synthetic networks:* To further validate the capability of our mapper, we perform additional experiments on three real engine ECUs extracted from a '12 Ford Focus, '13 Ford Escape and '14 Ford Escape.

7.1 Discovering an unexpected ECU

We now describe a real scenario where, in the process of designing *CANvas*, we discovered an unexpected ECU in our Prius. Using the results of our source mapping on the '09 Prius as seen in Table 2, we noticed that there were a total of nine ECUs when only eight were expected. Even after manually disconnecting all eight known ECUs, we still observed CAN traffic, specifically IDs $I_{570-572}$, coming from a single ECU. By looking at the history of the vehicle and systematically disconnecting various systems, we discovered that this ECU was installed as part of a modification from several years ago. The Prius had an additional battery installed to grant it all-electric capabilities, and with the use of the network mapper, we now know that a new CAN-enabled device was added. If we took a network map of the vehicle when first purchased or used an online database as mentioned in §2, we could easily compare our results with published results and identify the unexpected ECU. We confirm that these IDs are new by comparing our IDs to a same-generation Prius [23].

7.2 Identifying lenient filters

As detailed in §2, a real concern for network security is the ability to shut-down an ECU by simply receiving the target's CAN messages. Using the results of *CANvas'* destination mapping, we can identify several instances where an ECU is expected to only receive messages from a subset of other ECUs but still receives all other messages. We have found that all ECUs in the Focus and Prius do not employ any filter on the receipt of incoming messages. In Ford's Motorcraft TechInfo

Figure 13: Images of the vehicles we used for ground truth: the 2009 Toyota Prius and the 2017 Ford Focus.

ECU #	*Source message IDs*	*Actual ECU*
A	020, 030, 0B1, 0B3, 0B4, 230, 4C3, 591	Skid control ECU
B	022, 023	Yaw rate sensor
C	025, 4C6	Steering sensor
D	038, 03A, 03E, 120, 244, 348, 527, 528, 529, 540, 5B2, 5C8, 5EC, 602	Hybrid vehicle control ECU
E	039, 3C8, 3CF, 526, 52C, 5CC, 5D4, 5F8	Engine control module
F	262, 4C8, 521	Power steering ECU
G	3C9, 3CB, 3CD	Battery ECU
H	553, 554, 57F, 5B6	Gateway ECU
I	570, 571, 572	*Unknown ECU*

Table 2: 2009 Toyota Prius source mapping output

Service [3], we can see simple diagrams of how the ECUs communicate as part of the vehicle's systems. For example, the Focus' braking system involves communication between the instrument panel cluster, the transmission ECU, the body control ECU and the engine ECU. Now suppose an attacker takes over the infotainment unit of the Focus, has complete access to rewrite the ECU's code and gains the ability to inject CAN messages as described in §2. The attacker can launch a bus-off attack and shut-down the transmission ECU simply because the infotainment ECU receives its messages. It is evident that these devices need filters on what messages are received by their CAN controllers.

7.3 Mapping our test vehicles

We now present results and observations from mapping both the Prius and Focus.

Source mapping results: Using a threshold of 1ms and 30 minutes of traffic collection, we get a false positive rate of 0% for both vehicles, permitting us to get a sound source mapping output. Out of a total of 59 unique message IDs, our pairwise timing comparison resulted in 102 matching pairs for the Prius. By performing a simple grouping of these pairs as detailed in §5, we get the output as seen in Table 2. While the majority of the IDs observed on the Prius have a strong periodic characteristic, we discuss some special cases we encountered. Most of the messages were under five seconds except for I_{57F} with a period of 5 seconds and I_{602} with a period of 60 seconds. The majority of our messages matched with multiple IDs from the same ECUs but due to the large period of I_{57F} and I_{602}, they only had a single match. However, due to our pairwise approach, we can still map these two IDs using a shared matching pair as discussed in §5. We also encounter a few examples of messages that miss their deadline and wait until the next cycle to re-transmit. For the Focus, we observe messages that miss their deadlines and either transmit two messages on the next cycle or drop the missed message and wait for the next cycle. In these cases, we simply remove the inter-arrival times that exceed two standard deviations from the average period and interpolate for the removed timestamps as discussed in §5.

Destination mapping results: With a CAN bus running at 500 kbps, we discover that all of the ECUs in the Prius do not implement any filtering between the network and the CAN controller. When each ECU is isolated, we see that all IDs are properly acknowledged by the receiving ECU. We do observe two ECUs that recover quickly from the bus-off method, specifically the engine control module and the skid control ECU. With the other ECUs in the vehicle, it was sufficient to perform our bus-off once and the ECU would stop transmitting. For these two ECUs, we selected the smallest period ID and held the bus open by injecting false messages to keep the two ECUs from auto-recovering. Additionally, we discovered that the Focus also do not implement any sort of filtering for the IDs we observe on the CAN. From these findings, we can conclude that attacking via the reception of a message for these vehicles could prove trivial due to the lack of filtering between the network and the controller. In general, the maximum number of manual transitions of the ignition switch is equal to the number of detected CAN-enabled ECUs in the vehicle. For the keyless ignition of the 2009 Prius, we transition the ignition 7 times as two ECUs recover on their own (the Prius has 9 total CAN-enabled ECUs). For the keyed ignition of the 2017 Focus, we transition the ignition 7 times as two ECUs recover on their own (the Focus has 9 total CAN-enabled ECUs).

7.4 Mapping additional vehicles

Mapping real extracted Ford ECUs: We also obtained three Ford engine ECUs from a '12 Focus, '13 Escape and '14 Escape. By collecting data from these three ECUs, we found that they shared the many of the same message IDs and conclude that they are based off of the same engine controller configuration. As they all auto-recover, they were prime candidates for testing our forced ECU isolation technique.

We use *CANvas* on three other vehicles to look for data that seems logical to our findings from the test cars. For the Ford vehicles, we look for similarities with our extracted engine ECUs. For the '08 Escape, we found a set of IDs that we believe is the engine ECU and only has a subset of those found on our extracted ECU. For the '15 Fiesta, we also found a likely candidate for an engine ECU that has more IDs than our extracted ECUs. Since these vehicles range over three different Ford generations, it seems logical that the newer engine ECUs transmit more IDs. Additionally, we find a few similarities between the '09 and '10 Prius. We found an ECU on the '10 that is likely to be the skid control ECU, which has similar IDs to the '09 Prius. These findings potentially demonstrate *CANvas'* source mapping capabilities.

8 Discussion

Adversarial evasion: For *CANvas'* source mapping, an adversary could attempt to modify the timestamps to trick *CANvas* into thinking that a pair of IDs originate from the same ECU when in fact the opposite is true, and vice versa. We acknowledge that an attacker who aims to spoof IDs from an implanted or compromised ECU breaks the assumption for message-source analysis. If the attacker performs an active attack (i.e. attack occurs during data capture) or simultaneously transmits with the spoofed ECU, then IDSes from several previous works could detect such an attack and thus we did not perform such experiments. *CANvas* instead could discover ECUs that do not actively inject messages but rather change the ID-ECU source mapping (a new ECU or existing ECU that sends different IDs). We also make the assumption that ECUs do not intentionally alter their timing due to the challenges that arise from scheduling real-time embedded systems. There are numerous challenges that automakers already face in achieving reliable and robust scheduling for their vehicles and any modification to the timing of CAN messages would add a great amount of complexity to the already complex challenge of scheduling. Additionally, as our destination mapping approach deals with the error-handling mechanism, it would also not be practical to change these basics of CAN.

Avoiding permanent damage: We take care to avoid any damage to our test vehicles. Even with our active interaction with the bus in destination mapping, most dash lights that turn on are simply reset by power cycling the car; it may sometimes be necessary to drive the car for a few minutes so the ECUs can identify the absence of a real error. After mapping, all of our vehicles operate with no error codes once the above steps have been followed. Sometimes, a persistent Diagnostic Trouble Code may exist in the network as indicated by the Malfunction Indicator Light (MIL, commonly known as a "check engine light"). To remedy this, a simple OBD-II scan tool can be used to reset these lights with no harm to the vehicle. In the event of network communication failure (e.g. bus-off), manufacturers implement a "limp-home" mode where ECUs will default to secondary programming and allow the vehicle to operate with limited capabilities [7]. It is possible for the CAN bus to be shorted (effectively causing a bus-off on all ECUs) during faults, repairs, etc. so this mode protects the vehicle from our methods. In our experiments, the engine did not need to be running as all ECUs became active with the ignition at ACC. However, this may not apply to all vehicles so it is possible that the ignition will need to be ON.

Multiple CAN buses: For the typical OBD-II port, the CAN bus uses pins 6 and 14 on the connector. While many vehicles only have one CAN bus using these pins, it is possible for additional CANs to exist. These CAN buses may not be connected and they may employ different bus speeds. Sometimes, vehicles may also employ a gateway which handles how and which messages are passed between the various buses for reasons of fault confinement and network security. These CAN buses are often accessible at the OBD-II port but on different pins that are vendor optional: pins 3 and 11 and pins 1 and 8/9. In the case that a CAN bus is not exposed to the OBD-II, it is possible to access this bus by simply removing the door panel of a car and accessing the connector between the door assembly and the car body. This connector will likely contain the unexposed bus, which can be discovered as suggested by others [30].

Message acceptance filtering: CAN controllers have the option to employ a programmable acceptance filter where a message that is received by the controller can either be sent to the application layer or dropped after the message is received. It is possible to define message destination as a message that is "accepted" by an ECU rather than correctly received. This definition provides finer granularity on message destination and can prove useful for many other security scenarios; however, to identify what messages are accepted by an ECU, this may require vendor-specific methods. For example, in our experimental setup, we enable a CAN protocol feature called the overload frame [32]. If a vendor chooses to enable this feature, an accepted message can be determined by flooding the bus as fast as possible with a given message ID. When the receiving ECU gets behind on processing these messages, it will transmit an overload frame, indicating its acceptance filter allows the injected message ID; if the ID is dropped, then no overload frame will be present.

Non-transmitting ECUs: *CANvas* expects ECUs to transmit their messages periodically, but it is possible for ECUs to only activate under certain conditions or simply read from the network. As all ECUs that receive messages but have the ability to write to the network must participate in the ACK process, *CANvas'* forced ECU isolation technique can be used to identify the presence of a non-transmitting ECU. *CANvas* should detect these ECUs prior to starting to ensure that the detected ECUs do not interfere with destination mapping.

9 Related Work

We already discussed several of the key related work with respect to source and destination mapping. We discuss other related efforts here.

Automotive attacks: There have been a number of efforts at demonstrating vulnerabilities of automotive networks, including work on injecting messages [20], attacking keyless entry systems [8, 16, 28], and specific components such as TPMS [17, 18]. Our work can better inform such attack efforts and defenses by proactively identifying possible attack channels.

Intrusion detection for automotive: Given the growing security concerns, related work has also developed intrusion detection and firewall capabilities akin to traditional networks (e.g., [11, 19, 22, 29, 31]). Some of these may interfere with mapping efforts. More generally, however, these may have blind spots that a network mapper can highlight.

Alternative source identification: We acknowledge previous efforts that aim to identify message sources [12, 27]. While these efforts may prove valid, they either require many hours of data or require physical access to the bus for just source mapping. *CANvas* permits source mapping using a passively-recorded timestamped traffic log.

Authentication in CAN: We acknowledge that authentication for CAN devices may implicitly solve the source mapping problem. However, proposed authentication methods are rarely employed in real vehicles due to either the permanent addition of new devices or changes to the existing CAN protocol. Prior work, such as the TCAN system [5], requires the addition of a new device, access to two locations on the bus and a static authentication table. *CANvas*, however, acknowledges that timing characteristics can and will change due to clock drift. By comparing clock offsets, *CANvas* does not rely on static timing characteristics. CANvas does not even need physical access to the bus for source mapping as we only require a hardware-timestamped traffic log, and we operate solely from the OBD-II port without an additional permanent device.

Other work on ECU fingerprinting: Following initial efforts on fingerprinting [14, 27], other work has improved on their basic approach by identifying potential pitfalls [11, 12, 29]. As we show in our work, all of these still suffer from the same limitations in our context as they still assume either active access to the bus or very long traffic dumps.

10 Conclusions

In this work, we develop *CANvas*, a fast and inexpensive automotive network mapper. We have released our code and data under open-source licenses to enable further work in this area. A natural direction of future work is to add richer functionality, e.g. identifying the function of an ECU (transmission ECU, engine ECU, etc.), identifying gateway ECUs that potentially bridge multiple CAN buses and identifying vendor-specific message acceptance filters. Future work should also investigate network mapping on other automotive protocols, e.g. automotive Ethernet.

Acknowledgements

This work was funded in part by the PITAXVIII PITA award and the CNS-1564009 NSF IoT award. We gratefully acknowledge support from Technologies for Safe and Efficient Transportation (T-SET) University Transportation Center. This work was also supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We thank the anonymous reviewers and our shepherd Konrad Rieck for their helpful suggestions.

Availability

This work is made available [1] to encourage the community to add richer functionality and use *CANvas* to further the creation of automotive security tools.

References

[1] Canvas. https://github.com/sekarkulandaivel/canvas.

[2] Introduction to can. http://www.ti.com/lit/an/sloa101b/sloa101b.pdf.

[3] Motorcraft info service. https://www.motorcraftservice.com/.

[4] Obd-ii background information. http://www.obdii.com/background.html.

[5] Tcan: Authentication without cryptography on a can bus based on nodes location on the bus. https://autosec.se/wp-content/uploads/2019/03/3.-ESCAR-EU-2018.pdf.

[6] Toyota techinfo service. https://techinfo.toyota.com.

[7] What limp mode is, and why cars use it. https://repairpal.com/symptoms/what-is-limp-mode-why-cars-use-it.

[8] Ansaf Ibrahem Alrabady and Syed Masud Mahmud. Analysis of attacks against the security of keyless-entry systems for vehicles and suggestions for improved designs. *IEEE transactions on vehicular technology*, 54(1):41–50, 2005. https://ieeexplore.ieee.org/iel5/25/30186/01386610.pdf.

[9] 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*, pages 77–92. San Francisco, 2011. http://www.autosec.org/pubs/cars-usenixsec2011.pdf.

[10] Kyong-Tak Cho and Kang G Shin. Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1044–1055. ACM, 2016. https://dl.acm.org/citation.cfm?id=2978302.

[11] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *USENIX Security Symposium*, pages 911–927, 2016. https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_cho.pdf.

[12] Wonsuk Choi, Hyo Jin Jo, Samuel Woo, Ji Young Chun, Jooyoung Park, and Dong Hoon Lee. Identifying ecus using inimitable characteristics of signals in controller area networks. *IEEE Transactions on Vehicular Technology*, 67(6):4757–4770, 2018. https://ieeexplore.ieee.org/iel7/25/4356907/08303766.pdf.

[13] Robert I Davis, Alan Burns, Reinder J Bril, and Johan J Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007. https://link.springer.com/article/10.1007/s11241-007-9012-7.

[14] Marco Di Natale, Haibo Zeng, Paolo Giusto, and Arkadeb Ghosal. *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media, 2012. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.512.5543&rep=rep1&type=pdf.

[15] Mohammad Farsi, Karl Ratcliff, and Manuel Barbosa. An overview of controller area network. *Computing & Control Engineering Journal*, 10(3):113–120, 1999. https://ieeexplore.ieee.org/iel5/2218/17068/00788104.pdf.

[16] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science, 2011. https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/42365/eth-4572-01.pdf.

[17] Abdulmalik Humayed and Bo Luo. Cyber-physical security for smart cars: taxonomy of vulnerabilities, threats, and attacks. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, pages 252–253. ACM, 2015. https://dl.acm.org/citation.cfm?id=2735992.

[18] Rob Millerb Ishtiaq Roufa, Hossen Mustafaa, Sangho Ohb Travis Taylora, Wenyuan Xua, Marco Gruteserb, Wade Trappeb, and Ivan Seskarb. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *19th USENIX Security Symposium, Washington DC*, pages 11–13, 2010. https://www.usenix.org/legacy/event/sec10/tech/full_papers/Rouf.pdf.

[19] Min-Joo Kang and Je-Won Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, 11(6):e0155781, 2016. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0155781.

[20] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010. http://www.autosec.org/pubs/cars-oakland2010.pdf.

[21] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009. https://dl.acm.org/citation.cfm?id=1538595.

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

IEEE, 2012. https://ieeexplore.ieee.org/iel5/6238551/6239848/06240294.pdf.

[23] Jérôme Maye and Mario Krucker. Communication with a toyota prius. https://attachments.priuschat.com/attachment-files/2017/04/122809_Communication_with_a_Toyota_Prius.pdf.

[24] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. *Def Con*, 21:260–264, 2013. http://illmatics.com/car_hacking.pdf.

[25] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. *black hat USA*, 2014:94, 2014. http://illmatics.com/remote%20attack%20surfaces.pdf.

[26] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015:91, 2015. http://illmatics.com/Remote%20Car%20Hacking.pdf.

[27] Pal-Stefan Murvay and Bogdan Groza. Source identification using signal characteristics in controller area networks. *IEEE Signal Processing Letters*, 21(4):395–399, 2014. https://ieeexplore.ieee.org/iel7/97/4358004/06730667.pdf.

[28] Irving S Reed, Xiaowei Yin, and Xuemin Chen. Keyless entry system using a rolling code, February 4 1997. https://patentimages.storage.googleapis.com/c3/02/da/89f0cef9c2a9ea/US5600324.pdf.

[29] Sang Uk Sagong, Xuhang Ying, Andrew Clark, Linda Bushnell, and Radha Poovendran. Cloaking the clock: emulating clock skew in controller area networks. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 32–42. IEEE Press, 2018. https://dl.acm.org/citation.cfm?id=3207896.3207901.

[30] Craig Smith. *The Car Hacker's Handbook: A Guide for the Penetration Tester*. No Starch Press, 2016. http://opengarages.org/handbook/.

[31] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network. In *2016 international conference on information networking (ICOIN)*, pages 63–68. IEEE, 2016. https://ieeexplore.ieee.org/abstract/document/7427089/.

[32] CAN Specification. Bosch. 1991. http://esd.cs.ucr.edu/webres/can20.pdf.

[33] Ken Tindell, H Hanssmon, and Andy J Wellings. Analysing real-time communications: Controller area network (can). In *RTSS*, pages 259–263. Citeseer, 1994. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.5047&rep=rep1&type=pdf.