Practical DoS Attacks on Embedded Networks in Commercial Vehicles

Subhojeet Mukherjee[1], Hossein Shirazi[1], Indrakshi Ray[1],

Jeremy Daily[2], and Rose Gamble[2]

[1] Colorado State University, Fort Collins, CO 80523, USA
{subhomuk, shirazi, iray}@cs.colostate.edu
[2] University of Tulsa, Tulsa, OK 74104, USA
{jeremy-daily, gamble}@utulsa.edu

Abstract. The Controller Area Network (CAN) protocol has become the primary choice for in-vehicle communications for passenger cars and commercial vehicles. However, it is possible for malicious adversaries to cause major damage by exploiting flaws in the CAN protocol design or implementation. Researchers have shown that an attacker can remotely inject malicious messages into the CAN network in order to disrupt or alter normal vehicle behavior. Some of these attacks can lead to catastrophic consequences for both the vehicle and the driver. Although there are several defense techniques against CAN based attacks, attack surfaces like physically and remotely controllable Electronic Control Units (ECUs) can be used to launch attacks on protocols running on top of the CAN network, such as the SAE J1939 protocol. Commercial vehicles adhere to the SAE J1939 standards that make use of the CAN protocol for physical communication and that are modeled in a manner similar to that of the ISO/OSI 7 layer protocol stack. We posit that the J1939 standards can be subjected to attacks similar to those that have been launched successfully on the OSI layer protocols. Towards this end, we demonstrate how such attacks can be performed on a test-bed having 3 J1939 speaking ECUs connected via a single high-speed CAN bus. Our main goal is to show that the regular operations performed by the J1939 speaking ECUs can be disrupted by manipulating the packet exchange protocols and specifications made by J1939 data-link layer standards. The list of attacks documented in this paper is not comprehensive but given the homogeneous and ubiquitous usage of J1939 standards in commercial vehicles we believe these attacks, along with newer attacks introduced in the future, can cause widespread damage in the heavy vehicle industry, if not mitigated pro-actively.

Keywords: security, vulnerability, CAN, J1939, Data-link, Denial-of-Service

1 Introduction and Previous Efforts

Gone are the days when vehicles used to be driven solely by human-mechanical interactions. Since the advent of the Controller Area Network (CAN) in the early 1980s vehicle manufacturers have adopted a more cyber-physical approach to driving. Majority of the functions performed by vehicular mechanics are now mediated through embedded devices referred to as Electronic Control Units (ECUs). The ECUs help in executing critical (vehicle propagation, maintenance etc.) as-well as less critical (driver comfort, entertainment etc.) vehicular functions. While performing these functions, the ECUs interact with each other using fixed-length packets over the CAN bus. The CAN protocol follows a set of specifications [1] that enables it to support high-speed communications over a 2-wire serial broadcast bus. In addition, CAN allows assigning priorities to individual messages, thereby permitting higher priority messages to pass through at the time of contention. This not only allows ECUs to perform time critical functions like throttle and brake control but also perform less important functions like telematics and comfort management.

The CAN protocol facilitates in-vehicle message exchange. It does not however specify what messages are exchanged and how they are used by ECUs. It is often the responsibility of the vehicle manufacturer to implement protocols and standards which provide these functionalities. While passenger car manufacturers opt for proprietary standards, commercial vehicle vendors adopt a common set of standards specified by the SAE International. The standards are unified under the common naming convention SAE J1939 [9]. SAE J1939 is modeled on the ISO/OSI network protocol stack with the physical layer functionalities being realized by the CAN protocol. Together, the CAN protocol and J1939 specification sets help in accomplishing complex mechanical and electrical functions within a commercial vehicle.

Like other frequently used communication protocols and standards, CAN and J1939 are also accompanied by their fair share of security pitfalls. While attacks on the CAN protocol have been researched extensively of late [8, 12, 13, 4, 6], security aspects of the SAE J1939 specifications have been largely overlooked. More recently, Burakova et al. [2] attempted to replicate consumer vehicle specific attacks on their heavier counterparts by cleverly crafting, replaying and spoofing J1939 messages. The authors were successful in manipulating both critical (e.g. Engine RPM) and less critical features (e.g. Oil Pressure Gauge) to their desired levels. However, their work did not exploit any specifications made by the J1939 standards. In other words, these attacks are not specific to just trucks or other vehicles complying J1939 communications. In fact, by altering specifics of the attack vectors, similar attacks can be launched on consumer vehicles. Thus, to the best of our knowledge, this is the first work focused on discussing weaknesses in the SAE J1939 specifications. SAE J1939 is a collection of standards describing various functionalities at different layers. Currently, there are 17 such standards and each standard is a collection of different protocols and specifications. Documenting all possible attacks on J1939 is a time-consuming process. In order to scope our work, we limit our attacks to exploiting weaknesses in the the data-link layer protocols specified in the SAE J1939-21 standard document [10]. The reader can view this work as a proof-of-concept aimed at establishing the fact that attackers can exploit the protocol specified in the SAE J1939 standards to cause major damage.

Hoppe et al. [4] performed a practical security analysis of the CAN network and identified the basic weaknesses in the CAN protocol which allow targeted attacks to succeed. These weaknesses were modeled on the five central information security concerns, namely, confidentiality, integrity, availability, authenticity, and non-repudiation. After analyzing the majority of the current CAN security literature, we conclude that physical damage can be caused to the vehicle and the driver by exploiting the lack of integrity, availability, and authenticity services offered in the CAN bus. Deceiving ECUs to perform unintended actions (integrity and/or authenticity issues) or disabling the ability of the ECUs to perform regular tasks (availability issues) can result in problematic or undesirable consequences. Since J1939 uses CAN services at the physical layer, it is also susceptible to attacks launched by exploiting integrity, availability, and authenticity deficiencies. For example, J1939 allows some ECUs to command other ECUs to perform critical activities like transmission and torque/speed control. Impersonating as the former can allow attackers to control/modulate these vehicle critical functions. We refer to this as an injection attack. Similarly, attackers can inhibit the functionalities offered by one or more ECUs by overwhelming the performance capabilities of the ECUs or the bus. We refer to such an attack as denial-of-service (DoS) attack as it adversely affects the services provided by the ECUs or the CAN bus. Although both these attacks can lead to fatal consequences, in this work we limit out exploration to DoS attacks on the SAE J1939 data-link layer protocol. This is because injection attacks can be launched straightforwardly by searching for command messages from the J1939 Digital Annex [11] and injecting them into the CAN bus. On the contrary, DoS attacks require studying the workflow of the SAE J1939 data-link layer protocols, finding suitable attack vectors and drawing inferences by analytically observing of the bus traffic. The scientific challenges involved in executing DoS attacks make it more interesting from a research perspective compared to injection attacks.

Our goal in this paper is to demonstrate techniques by which the regular work-flow of the J1939 data-link layer protocols can be disrupted. However, we do not discuss the eventual effect of attack on the mechanical behavior of the vehicle. This is because we assume that some normal vehicular functions depend entirely on the seamless accomplishment of all the protocols involved in executing them and any disparity observed in the protocol flow should cause some adverse effect on the mechanical behavior of the vehicle. Documenting the exact effect is beyond the scope of this work.

The rest of the paper is organized as follows. Section 2 provides a brief overview of CAN protocol and J1939 standards with emphasis on the J1939 data-link layer [10]. Section 3 discusses our threat model, a concise categorization of the attacks performed in this paper, and the experimental setup used. Section 4 documents and analyzes three separate DoS attacks. Each attack is complemented with suggested mitigation techniques. Section 5 concludes the pa-

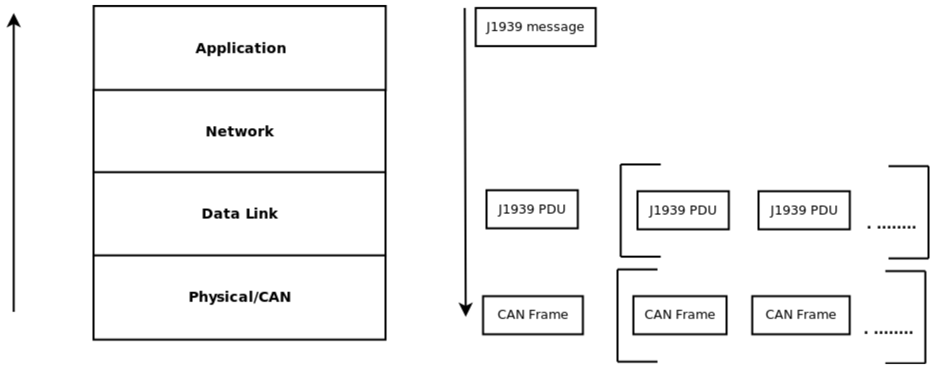


Fig. 1. J1939-OSI Model

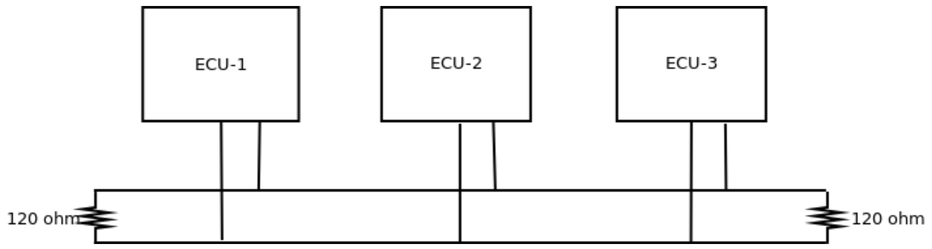
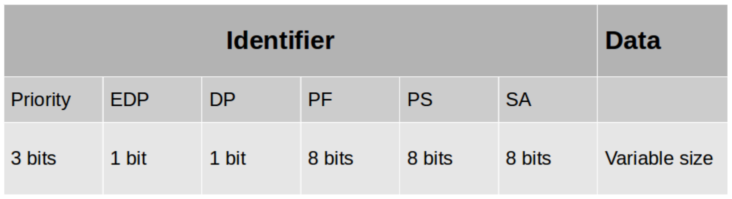


Fig. 2. Example CAN Network



Identifier						Data
Priority	EDP	DP	PF	PS	SA	
3 bits	1 bit	1 bit	8 bits	8 bits	8 bits	Variable size

Fig. 3. J1939 Message Format

per with an overview of the results achieved and indicates the future direction of advancements for both attack and defense strategies for the J1939 standards.

2 Background

Embedded communications in commercial vehicles are facilitated by the SAE J1939 [9] standards. As shown in Fig. 1, J1939 is modeled on the ISO/OSI protocol stack. A J1939 packet is created at the applications layer. As a packet moves down the layers it is optionally split up into two or more protocol data units (PDUs) at the data-link layer. This is because the physical layer operations are guided by the CAN protocol which allows a maximum of 8 data bytes in one CAN frame. Finally, the CAN frames are exchanged using CAN protocol specifications.

2.1 The Physical (CAN) Layer

Functions at the lowermost layer the of the J1939 protocol stack are handled by the CAN protocol [1]. The protocol handles transmission of J1939 packets over a 2-wire multi-master serial bus. CAN is a broadcast protocol and does not specify unicast message transfer. This means every node (ECU) on a CAN bus can see messages transmitted by every other node. Protocols running on top of the CAN bus, however, can implement functionalities to accomplish point-to-point message transfer. J1939, as will be seen later in this section, uses source and destination address fields to specify senders and receivers of CAN frames. An example CAN network is shown in Fig. 2. ECUs can transmit messages on the bus following a CSMA/CD bus access method. This means the ECUs can transmit messages on the bus only when it is free. If two ECUs transmit on a free bus at the same time, the protocol arbitrates between the two messages using the CAN message identifier. The CAN identifier is an additional 11 (standard) or 32 (extended) bit field prepended to an 8 byte CAN message. As it will be seen later, J1939 recommends the 29 bit identifier, hence the extended CAN identifier is used for arbitration purposes. Finally, in CAN bus terminology a 0 (dominant bit) on the bus is considered to be of higher priority than 1 (a recessive bit). This means, on the CAN bus, a message whose prefix is "000" overwrites the one whose prefix is "001".

2.2 J1939 Packet Formatting

The general format of the J1939 message is shown in Fig. 3. A single J1939 message can be partitioned into a 29 bit identifier (ID) section and variable size data section. Since the CAN protocol allows only 8 bytes of data in one frame, the variable size data section is broken up into 8 byte packets and appended with the identifier to form a J1939 PDU. At the physical layer, a few more CAN specific bits are added to the J1939 PDU and transmitted on the bus as a CAN frame.

Identifier Field The J1939 identifier is divided into 6 sub-fields.

- **Priority**: The 3 bit priority field is used to as a base for the CAN arbitration scheme. Priorities can vary from 000_2 (0_{10}) to 111_2 (7_{10}). The J1939 standard assigns a default priority of 011_2 (3_{10}) to vehicle control messages and 110_2 (6_{10}) to all other messages. The priority is ultimately specified by the original equipment manufacturer (OEM).
- **Extended Data Page (EDP)**: Currently the EDP bit is set to 0_2 (0_{10}) for all J1939 messages.

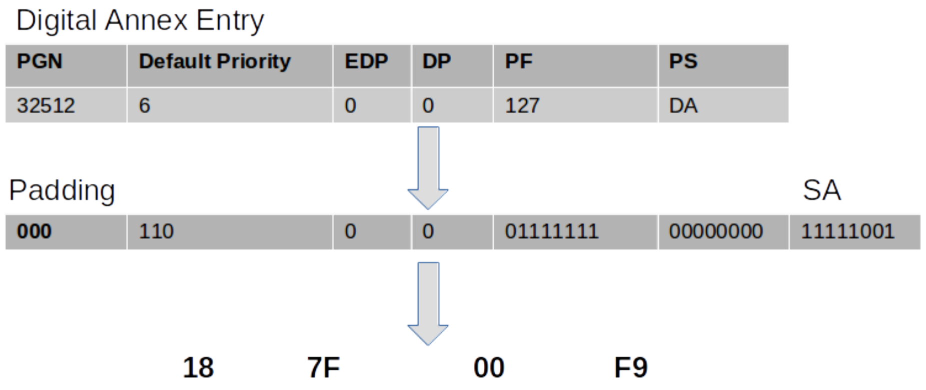


Fig. 4. Generating a J1939 Identifier from the Digital Annex

- **Data Page (DP)**: The DP bit can be set to either 0_2 (0_{10}) or 1_2 (1_{10}). The actual value of the DP bit for a particular message can be obtained from the J1939 Digital Annex [11].
- **PDU Format (PF) and PDU Specific (PS)**: In terms of message communication the PF and PS are the most significant bit fields. When put together along with EDP and DP they evaluate to what is referred to as the Parameter Group Number (PGN). PGNs are used to group J1939 messages according to their functionality. For example, messages related to torque or speed control are assigned the PGN 0_{10} (0000_{16}), whereas those related to tire sensor identification are assigned the PGN 32512_{10} ($7F00_{16}$). When encoded hexadecimals, the first ten bits of the PGN represent the PF and the last 8 bits represent the PS. When PF values are less than 240_{10} ($F0_{16}$) the PS field is used to specify the address of the intended receiver.
- **Source Address (SA)**: The source address field is used to specify the address of the sender. The source address field can be used to filter and process messages at the hardware level to avoid overloading the ECU firmware with unnecessary message processing. Source addresses can range from 00000000_2 ($0_{10}/00_{16}$) to 11111111_2 ($255_{10}/FF_{16}$). The J1939 Digital Annex [11] contains a list of suggested source address assigned to various functional ECUs.

To summarize, Fig. 4 shows how a J1939 identifier is constructed from J1939 standard entries. An additional 3 bit padding is added to convert the identifier into 32 bit CAN arbitration field. Since the PF is less than 240, the PS field denotes the receiver of the message (00_{16}) which in this case is the Engine#1 ECU. The SA field is used to denote the sender $F9_{16}$ which is the off-board diagnostic service tool.

Data Field J1939 message data field is constructed using Suspect Parameter Numbers (SPNs). Each PGN is associated with a set of SPNs. An SPN definition determines how a message (encoded in bits) belonging to a particular PGN is converted into application readable information. For example, the first 2 least significant bits in PGN 32512_{10} data is assigned the SPN 695_{10}. According to the SPN definition, the 2 least significant bits denote the Engine Override Control Mode and can assume any of 4 (2^2) states. The attacks demonstrated in this paper do not make use of SPN numbers and hence we do not discuss this further.

2.3 Message Transmission Rates

J1939 recommends transmitting messages at various rates depending on the PGN Transmission Rate specification available in the digital annex [11]. A broad categorization of the transmission rates is presented below. The categorization was done by thoroughly observing the Transmission Rate specification available in the digital annex.

- **Periodic**: Transmitted at various time intervals (seconds or milliseconds) as specified in the J1939 standards.
- **On-Request**: Transmitted on receiving a request.
- **Event-Based**: Transmitted at the occurrence of a specific event or interrupt.
- **Manufacturer Defined**: Transmission rates are defined by the manufacturer.
- **Requirement Based**: Transmitted only if required.
- **Conditional**: Dependent on ECU parameters like Engine Speed or other factors like state change.
- **Unspecified**: Transmission rates are not specified for these PGNs.
- **Hybrid**: Any combination of the above categories. For example, the time interval of periodically transmitted messages can vary depending on conditional factors.

2.4 J1939 Data-link layer

Fig. 1 shows 4 layers in a J1939 protocol stack. Each of these layers has one or more standard documentations associated with them. The documentations can be found in SAE standards repository (`http://www.sae.org`). The attacks documented in this paper employ extensive usage of the request message documented in the J1939-21 (data-link layer) [10] standard. The request message (PGN $59904_{10}/EA00_{16}$) is used to request a particular PGN from a single or a group of ECUs on the bus. Since the PF ($234_{10}/EA_{16}$) for the request PGN is less than 240_{10} ($F0_{16}$), the PS field is used to specify the address of the intended receiver of this message. This address can be destination specific like Engine (00_{16}), Brake ($0B_{16}$) or global broadcast (FF_{16}). A destination specific request is answered by the receiver with either the requested PGN or a negative acknowledgment. Acknowledgment messages are assigned to the PGN 59392_{10} ($E800_{16}$). As with the request PGN the acknowledgment can also be destination specific or broadcast. The first byte in the data field of an acknowledgment messages is the control byte. The mapping for the control byte values and the information conveyed by the respective acknowledgment messages are shown below:

		Identifier				**Data Bytes**							
Label	Sub Label	PGN	PF	PS	Default Priority	1	2	3	4	5	6	7	8
Request	N/A	EA00	EA	Dest-Addr	6	Requested PGN in reverse byte order			N/A				
Connection Management	Request to Send	EC00	EC	Dest-Addr	7	10	Total number of bytes to be transferred		Total number of packets to be sent	Max number of packets to be sent in response to 1 CTS: FF for any	Requested PGN in reverse byte order		
Connection Management	Clear to Send	EC00	EC	Dest-Addr	7	11	Number of Packets that can be send	Next sequence number to send	FF	FF	Requested PGN in reverse byte order		
Data Transfer	N/A	EB00	EB	Dest-Addr	7	sequence number	Data						

Table 1. Frequently Used PGNs

- 0_{10} (00_{16}): Positive Acknowledgment (ACK).
- 1_{10} (01_{16}): Negative Acknowledgment (NACK).
- 2_{10} (02_{16}): Access denied.
- 3_{10} (03_{16}): Cannot respond.
- 4_{10} (04_{16}) - 255_{10} (FF_{16}): Reserved for SAE assignment.

Requested PGNs are transferred either as a single packet (with 8 bytes or less of data) or multiple packets (with more than 8 bytes of data). In the second case, SAE recommends implementing a connection oriented multi-packet data transfer. A destination specific multi-packet data transfer (Fig. 5) starts by initiating a request (PGN $59904_{10}/EA00_{16}$). The requested party attempts to open a connection by sending a `Request to Send (RTS)` message (PGN $60416_{10}/EC00_{16}$). In response, the requester sends a `Clear to Send (CTS)` message (PGN $60416_{10}/EC00_{16}$). Upon receiving the CTS the requested party starts sending the data using the data transfer PGN ($60160_{10}/EB00_{16}$). On successful completion of the message transfer, the requester sends an `End of Message Acknowledgment (EndOfMsgACK)` (PGN $60416_{10}/EC00_{16}$). A summary of the PGNs used in our attacks (request, connection management and destination specific data-transfer) is shown in Table 1.

3 Preliminaries

The contents of this section convey the preparatory information for the attacks demonstrated in the next section. This includes the threat model under which our attacks were performed, a concise categorization of our attacks, and the experiment set-up we used to conduct the DoS attacks.

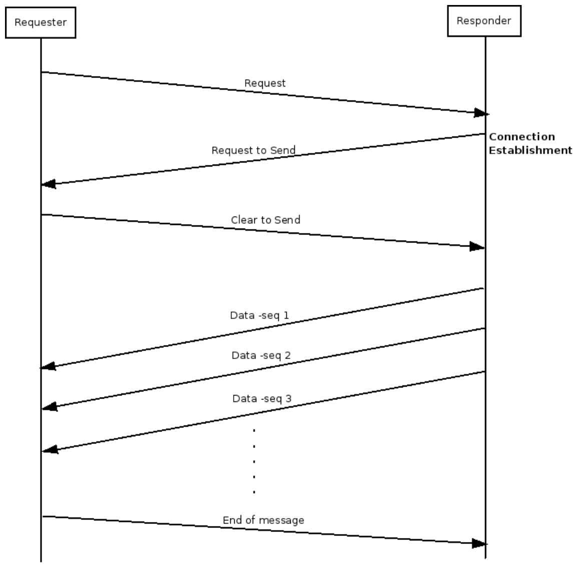


Fig. 5. Requested Multi-packet PGN transfer

	Type of Message		**Exploit**	
Attack Name	Request	Connection Management	Implementation Issues	Specification Issues
Request Overload	Yes	No	No	Yes
False RTS	Yes	Yes	Yes	Yes
Connection Exhaustion	Yes	Yes	No	Yes

Table 2. Attack Categorization

3.1 Threat Model

For the purpose of this work, we assume an active adversary with direct access to the CAN bus. By active, we mean that the adversary is capable of injecting any message into the CAN bus and disrupting the regular operations. The capabilities of the adversary are however restrained by the computational power of the device which is used to inject these messages. This device can be physically attached to the bus (a compromised Entertainment ECU or a pass through device attached to the OBD-II port) or connected remotely to wireless interfaces on the vehicle bus such as the telematics units, Tire Pressure Monitoring System (TPMS) or Bluetooth unit [3]. The use of any of these attack surfaces constricts the attackers resource significantly, either due to low computation power or considerable network delay.

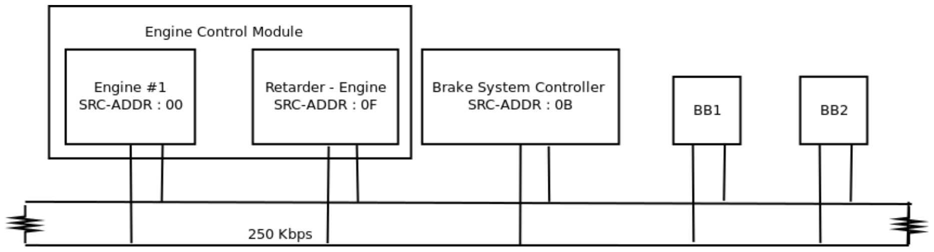


Fig. 6. Experiment Test-Bed Schematic

3.2 Attack Categorization

Three separate DoS attacks are demonstrated in Section 4 of this paper. In this subsection, we attempt to classify the attacks on the basis of a few factors: type of message (PGN) used for attack and exploit (flaws in implementation and/or specifications). The categorization is shown in Table 2. The leftmost column of the table, lists the three attacks namely, `Request Overload`, `False RTS` and `Connection Exhaustion`. The details about execution and findings from these attacks are reported in the next section (4).

3.3 Experiment Test-Bed

All attacks were conducted on a test-bed consisting of a single high-speed CAN bus with a baud rate of 250 kbps. The normal bus load was measured at 14%-15% using the canbusload utility from the can-utils [7] program built for SocketCAN in Linux. A schematic of the test-bed is shown in Figure 6. The test-bed consisted of an Engine Control Module (ECM) and a standalone `Brake Controller` ($0B_{16}$). The ECM includes an `Engine-#1` ECU (SRC: 00_{16}) and a `Retarder-Engine` ECU (SRC: $0F_{16}$) [3]. The make and model of the ECUs are not revealed to protect vendor reputation.

Fig. 6 also shows two BeagleBone Black (`BB1` and `BB2`) devices with custom built heavy vehicle communication protocol transceivers. The BeagleBones act as regular ECUs or other embedded devices connected to the bus. All attacks were performed using these devices. Both the BeagleBones hosted 32 bit Ubuntu@Linux operating systems running on an ARM processor with 500 MB of RAM. Although we had can-utils [7] at our disposal we preferred to use the python3 implementation[4] of the the SocketCAN driver [5] to conduct the attacks. This is because SocketCAN offered much more flexibility in implementing a morphed version of the J1939 data-link layer protocols for the purpose of the attacks.

Ten different snapshots of the CAN bus traffic was taken for 10 seconds each. It was observed that the traffic pattern outlined in Table 3 was exactly same for

[3] The names of the ECUs are obtained from the J1939 Digital Annex Source Address Tab

[4] `http://python-can.readthedocs.io/en/latest/socketcan_native.html`

Identifier	Priority	PGN	SRC	Count	Measured interval in ms	Matching Annexed Interval in ms
0CF00300	High	61443	00	200	50	50
0CF00400	High	61444	00	500	20	20
18E0FF00	Low	57344	00	10	1000	1000
18EBFF00	Low	60160	00	130	76.9230769231	Prop
18EBFF0F	Low	60160	0F	6	1666.6666666667	Prop
18ECFF00	Low	60416	00	12	833.3333333333	Prop
18ECFF0F	Low	60416	0F	2	5000	Prop
18F0000F	Low	61440	0F	100	100	100
18F00100	Low	61441	00	100	100	100
18F0010B	Low	61441	0B	99	100	100
18FD7C00	Low	64886	00	10	1000	Prop
18FDB300	Low	64947	00	20	500	500
18FDB400	Low	64948	00	20	500	500
18FEBD00	Low	65213	00	10	1000	1000
18FEBF0B	Low	65215	0B	100	100	100
18FEC100	Low	65217	00	10	1000	1000
18FEDF00	Low	65247	00	500	20	Prop
18FEE000	Low	65248	00	100	100	100
18FEE400	Low	65252	00	10	1000	1000
18FEEE00	Low	65262	00	10	1000	1000
18FEEF00	Low	65263	00	20	500	500
18FEF000	Low	65264	00	100	100	100
18FEF100	Low	65265	00	100	100	100
18FEF200	Low	65266	00	100	100	100
18FEF500	Low	65269	00	10	1000	1000
18FEF600	Low	65270	00	20	500	500
18FEF700	Low	65271	00	10	1000	1000
18FEFF00	Low	65279	00	1	10000	10000

Table 3. Test-Bed Traffic

all 10 snapshots. Only two distinct priorities were observed on the bus: $011_2/3_{10}$ ($0C_{16}$ with padding) and $110_2/6_{10}$ (18_{16} with padding). The Measured Intervals were calculated by dividing 10000 ms (10 seconds) by the individual message counts. The Matching Annexed Intervals were obtained for each observed PGN from the digital annex [11]. If the Matching Annexed Intervals did not match the Measured Intervals it was assumed that they were pre-programmed by the vendor and marked “Prop”.

4 Attacks

In our pursuit to find weaknesses in the J1939 data-link layer specifications, we performed three separate DoS attacks that were briefly introduced in section 3.2. We now present the details of the attacks. The documentation process for each attack is subdivided into 5 components:

1. **Background Theory**: We begin by introducing the core J1939 concept exploited for the attack.
2. **Proposed Attack**: An attack is proposed based on the background theory.

3. **Execution**: The attack is executed.
4. **Observation and Analysis**: The effect of the attack is evaluated by studying the network traffic and optionally using fitting metrics and charts. If required statistical significance testing is performed to gauge the truth value (Success or Failure) of the attack.
5. **Suggested Mitigation Techniques**: Finally, we suggest some probable mitigation techniques for the described attack.

4.1 Request Overload

Background Theory The J1939-21 standard suggests an algorithm to filter received messages at the microprocessor level. The intended use of this algorithm is to reduce the load on the application. For a destination specific request, the filtering algorithm recommends queuing message bytes (for further processing) if the destination address in the message identifier matches the device's source address. Once a request is queued, the ECU is expected to see if the PGN is supported by it. If supported, the ECU should reply back with the PGN.

Proposed Attack Sending a large volume of request messages for a supported PGN should increase the computational load of the recipient ECU to an extent where it might not be able to perform regular activities like transmitting periodic messages.

Execution We wrote a Python script to send repeated requests for ECU component id (PGN $65259_{10}/EBFE_{16}$) to the Engine-#1 ECU (refer to Fig. 6). We chose the Engine ECU as the target because we wanted to reduce the count of high-priority messages on the bus and the normal bus traffic from Table 3 shows that Engine-#1 is the only ECU transmitting high priority ($0C_{16}$) messages. The component id is a multi-packet (greater than 8 bytes) PGN. Responding to the request thus requires the ECU to perform slightly more activity than responding with a single-packet PGN.

Our attack script expected three arguments (used as independent variables for further analysis), namely, (1) number of concurrent threads, (2) injection time interval and (3) source address. The first two arguments allowed us to strengthen the magnitude of the attack. The final argument was varied to spoof the sender of the injected message. Three values were chosen for the spoofed address: $0B_{16}$ (Brake Controller), 00_{16} (Engine-#1) and $F9_{16}$ (Off Board Diagnostic Service Tool). The idea was to observe whether replies sent by the Engine-#1 to the brake, to itself or to a non-existent ECU alters its behavior in any way.

Observation and Analysis As seen from Fig. 7 and Table 4, performing the attack caused regular messages on the bus to drop significantly. High Priority message (blue curve) count dropped by an average of 46.64% with the maximum drop obtained at *spoofed-SRC: F9, num-thread: 8, interval: 1.2.* Low priority

Attack Parameters			**Average Message Count per Source Address**							
			00				**0B**		**0F**	
			High P		Low P		Low P		Low P	
Source	Thread (no)	Interval (ms)	Count	Decrease(%)	Count	Decrease(%)	Count	Decrease(%)	Count	Decrease(%)
0B	1	0.4	216	38.29	257	60.16	117	-17	23	56.6
0B	1	0.8	153	56.29	114	82.33	120	-20	12	77.36
0B	1	1.2	123	64.86	86	86.67	140	-40	8	84.91
0B	4	0.4	297	15.14	502	22.17	111	-11	40	24.53
0B	4	0.8	221	36.86	319	50.54	119	-19	28	47.17
0B	4	1.2	197	43.71	219	66.05	122	-22	17	67.92
0B	8	0.4	215	38.57	285	55.81	125	-25	21	60.38
0B	8	0.8	115	67.14	98	84.81	129	-29	5	90.57
0B	8	1.2	117	66.5	46	92.87	118	-18	8	84.91
F9	1	0.4	235	32.86	302	53.18	118	-18	27	49.06
F9	1	0.8	136	61.14	118	81.7	128	-28	6	88.6
F9	1	1.2	121	66.4	75	88.37	119	-19	5	90.6
F9	4	0.4	310	11.43	524	18.76	109	-9	45	15.09
F9	4	0.8	239	31.71	317	50.85	118	-18	29	45.28
F9	4	1.2	207	40.86	253	60.78	130	-30	20	62.26
F9	8	0.4	221	36.86	301	53.33	125	-25	21	60.38
F9	8	0.8	131	62.57	118	81.7	127	-27	8	84.91
F9	8	1.2	104	70.2	63	90.23	128	-28	6	88.7
00	1	0.4	223	36.29	309	52.09	129	-29	25	52.83
00	1	0.8	145	58.57	106	83.5	120	-20	7	86.7
00	1	1.2	116	66.86	100	84.5	130	-30	6	88.7
00	4	0.4	283	19.14	465	27.91	112	-12	41	22.64
00	4	0.8	235	32.86	229	53.64	121	-21	20	62.26
00	4	1.2	232	33.71	306	52.56	121	-21	31	41.51
00	8	0.4	215	38.57	314	51.32	109	-9	17	67.92
00	8	0.8	128	63.43	111	82.7	114	-14	5	90.5
00	8	1.2	110	68.5	70	89.1	140	-40	7	86.7

Table 4. Request Overload Effect on Normal Traffic: Percentage Reduction in Regular Message Volume [Raw Statistics]

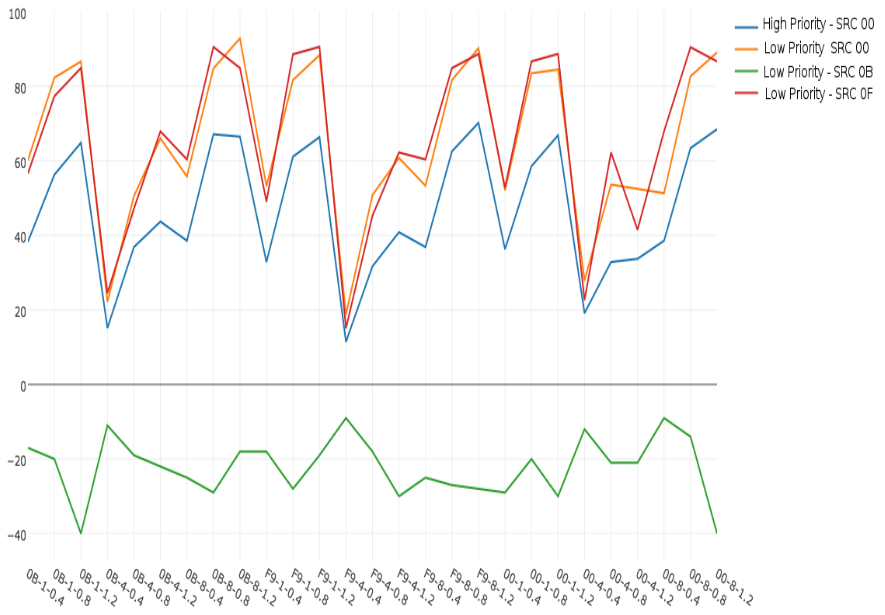


Fig. 7. Request Overload Effect on Normal Traffic: Percentage Reduction in Regular Message Volume

message count, on the other hand, dropped significantly for both the Engine-#1 (SRC: 00_{16}) and the Retarder (SRC: $0F_{16}$) although the average drop was almost equal ($\tilde{6}5\%$) for both. The peak drops for Engine-#1 (SRC: 00_{16}) and Retarder were observed at the following points *spoofed-SRC: 0B, num-thread: 8, interval: 1.2* and *spoofed-SRC: F9, num-thread: 8, interval: 1.2* respectively. The least amount of drop in count (for orange,red and blue lines from Fig. 7) was observed at the point *spoofed-SRC: F9, num-thread: 4, interval: 0.4*.

Pearson correlation coefficients for each independent variable (argument) and the reduction percentages for high priority messages (high priority messages were chosen for this purpose since they are hard to suppress on a CAN bus) are shown below:

- **Source**: -0.01 (negative weak correlation). As the source address increases from 00 to F9, reduction percentages drop [weakly].
- **Thread**: 0.137 (weak positive correlation). As the number of threads increase reduction percentages increase [weakly].
- **Interval**: 0.66 (strong correlation). As the interval increases reduction percentages increase [strongly] .

Positive correlation for factors *Thread* and *Interval* explain the existence of the lowest and highest count reduction percentages at points *spoofed-SRC: F9, num-thread: 8, interval: 1.2* and *spoofed-SRC: F9, num-thread: 4, interval: 0.4*

Identifier	**Regular Count from Table**	**Attack count (F9,4,0.4)**
0CF00300	200	86
0CF00400	500	224
18E0FF00	10	4
18EBFF00	130	53
18ECFF00	12	4
18F00100	100	42
18FD7C00	10	4
18FDB300	20	6
18FDB400	20	9
18FEBD00	10	5
18FEC100	10	4
18FEDF00	500	203
18FEE000	100	36
18FEE400	10	5
18FEEE00	10	4
18FEEF00	20	9
18FEF000	100	45
18FEF100	100	35
18FEF200	100	40
18FEF500	10	4
18FEF600	20	9
18FEF700	10	3
18FEFF00	1	0

Table 5. Non-Parametric U-Test Samples

Finally, we performed a two-tailed Mann-Whitney U test to determine if our attack succeeded. We compared the counts of Engine-#1 transmitted messages on the bus before and after the attack were performed. The attack arguments were chosen to be from the point which produced the lowest message count reduction (*spoofed-SRC: F9, num-thread: 4, interval: 0.4*). The samples for the U-test are shown in last two columns of Table 4. After performing the non-parametric test, we obtained a p-value of 0.01468 and thereby concluded our attack produced significant differences ($p \leq .05$) in message count at a 5% confidence interval. Since the positive reduction percentages were obtained for all Engine-#1 message counts, we conclude that our attack was successful. Using the worst results to perform the significance tests allowed us to have the best notion about the performance of the attack.

It should be noted that this type of attack could be unintentional since third party telematics units often request component information from ECUs. While this is not an attack, a poorly programmed ECU on the network could have the same effect as shown above.

Suggested Mitigation Techniques One approach to prevent such an overloading scenario can be to program the ECU such that it drops incoming request packets if it has already responded to a request from the same source address within a pre-defined time interval. This, however, requires ECUs to maintain state information and can, in turn, lead to further resource exhaustion. Designers or developers can, however, opt for indigenous techniques to defend against this scenario. Another alternative can be to opt for proper intrusion detection systems (IDSs) with capabilities of distinguishing such attack traffic from normal traffic.

4.2 False Request to Send (RTS)

Background Theory The J1939-21 standard specifies that if multiple RTS messages are received from the same source address then the most recent RTS shall be considered and previously received RTSs shall be discarded without sending a notification to the sender of the RTS message.

Proposed Attack Consider a connection in progress where the requester receives an RTS from the recipient (Alice) of a request message. After receiving the RTS, the requester allocates a buffer having size equal to that received in bytes 2 and 3 of the data field in the RTS packet (refer to Table 1). The requester then sends a CTS requesting for given number of packets starting from sequence number 1. A clever attacker (Bob) can then send a crafted RTS packet (with a reduced data size in bytes 2 and 3 of the data field) to the requester spoofing the source address of the original recipient of the request. If the receiver of the spoofed RTS reallocates the buffer and keeps receiving data (PGN: $EB00_{16}$) packets from the original sender (Alice), the allocated buffer might overflow causing the ECU firmware to crash.

Execution To test this attack we used both BeagleBone Black devices connected to our test-bed. On one device (`BB1`) we ran a faulty script to receive multi-packet PGNs. The workflow of the program is shown below.

```
Send request;
In a separate thread:
      Sniff for RTS;
      On receiving RTS allocate/reallocate
      buffer space (buffer size = as obtained
      from bytes 2-3 of the RTS data field);
Send CTS;
Recieve data;
```

On the second BeagleBone Black device (`BB2`) we ran the attack script as shown below:

```
Sniff bus for CTS from attack target;
Send crafted RTS with lesser data size;
```

Observation and Analysis We ran both the scripts on the two BeagleBones for 10 consecutive occasions. It was observed that on all occasions the script running on BB1 crashed. This can be fatal for an ECU because crashing the firmware can render an ECU useless.

Suggested Mitigation Techniques It is extremely hard to defend against such attacks. If the ECU firmware developer decides to avoid re-allocating space on receipt of the second RTS, the attacker can spoof the first RTS and cause the exact same damage. The success of this attack can be attributed to two factors: the exploitable J1939 concept detailed as a part of the background theory and insufficient programming logic. Thus, according to us, the best defense against this attack is to avoid allocating space statically using the size specified in the RTS message. The receiving side can incrementally allocate 7 bytes [5] as newer packets arrive.

4.3 Connection Exhaustion

Background Theory The J1939-21 standard restricts that each pair of ECUs can have at most one connection at any given point of time. Moreover, J1939 allows requesters to keep connections open by sending CTS messages within a specified time period.

Proposed Attack The J1939 source address is an 8 byte field. This means there can be at most 255 different ECUs connected to a bus. If a driving critical ECU like a brake controller can support 255 different connections at the same time, an attacker can open 255 separate connections to that ECU and keep the connection open by sending periodic CTS messages. In such a case, no other ECU can open connections to the brake controller. In practice, the actual number of ECUs connected to the bus is most often much less than 255. This makes the task easier for the attacker. The quick scan of the network traffic can reveal the transmitting source addresses. The attacker can then spoof all available source addresses and open a connection to other ECUs thereby creating a mesh network of connections. In such a case no other ECU will be able connections to other ECUs.

[5] the first byte of a data packet is the sequence number

```
BB1->Engine-#1 request          00EA0011   EB FE 00 00 00 00 00 00
Engine-#1->BB1 RTS              18EC1100   10 2C 00 07 FF EB FE 00
BB1->Engine-#1 CTS              00EC0011   11 07 01 FF FF EB FE 00
BB1->Engine-#1 request          00EA000B   EB FE 00 00 00 00 00 00
Engine-#1->BB1 RTS              18EC0B00   10 2C 00 07 FF EB FE 00
BB1->Engine-#1 CTS              00EC000B   11 07 01 FF FF EB FE 00
Engine-#1->BB1 Data Transfer    18EB1100   01 43 4D 4D 4E 53 2A 36
Engine-#1->BB1 Data Transfer    18EB1100   02 43 20 75 30 37 44 30
Engine-#1->BB1 Data Transfer    18EB1100   03 38 33 30 30 30 30 30
Engine-#1->BB1 Data Transfer    18EB1100   04 30 30 2A 30 30 30 30
Engine-#1->BB1 Data Transfer    18EB1100   05 30 30 30 30 2A 78 30
Engine-#1->BB1 Data Transfer    18EB1100   06 36 42 42 42 42 42 42
Engine-#1->BB1 Data Transfer    18EB1100   07 42 2A FF FF FF FF FF
Engine-#1->BB1 Data Transfer    18EB0B00   01 43 4D 4D 4E 53 2A 36
Engine-#1->BB1 Data Transfer    18EB0B00   02 43 20 75 30 37 44 30
Engine-#1->BB1 Data Transfer    18EB0B00   03 38 33 30 30 30 30 30
Engine-#1->BB1 Data Transfer    18EB0B00   04 30 30 2A 30 30 30 30
Engine-#1->BB1 Data Transfer    18EB0B00   05 30 30 30 30 2A 78 30
Engine-#1->BB1 Data Transfer    18EB0B00   06 36 42 42 42 42 42 42
Engine-#1->BB1 Data Transfer    18EB0B00   07 42 2A FF FF FF FF FF
BB2->Engine-#1 request          00EA0011   EC FE 00 00 00 00 00 00
BB2->Engine-#1 request          00EA000B   EC FE 00 00 00 00 00 00
BB2->Engine-#1 request          00EA0011   EC FE 00 00 00 00 00 00
BB2->Engine-#1 request          00EA000B   EC FE 00 00 00 00 00 00
BB2->Engine-#1 request          00EA0011   EC FE 00 00 00 00 00 00
BB2->Engine-#1 request          00EA000B   EC FE 00 00 00 00 00 00
BB2->Engine-#1 request          00EA0011   EC FE 00 00 00 00 00 00
BB2->Engine-#1 request          00EA000B   EC FE 00 00 00 00 00 00
BB1->Engine-#1 CTS              00EC0011   11 07 01 FF FF EB FE 00
BB1->Engine-#1 CTS              00EC000B   11 07 01 FF FF EB FE 00
```

Fig. 8. Connection Exhaustion Network Trace (Without End of Message ACK)

Execution None of the ECUs on our test-bed attempted to make destination specific connections to each other (refer to Table 3). However, for the purpose of testing this attack, we programmed BB1 to act as the attacker controlled device and BB2 to impersonate two different ECUs (Brake Controller (SRC: $0B_{16}$) and Cruise Control (SRC 11_{16})) and attempt to make connection requests to the Engine-#1 ECU. The BB1 device was programmed to create two connections with the Engine-#1 ECU requesting for the Component ID PGN ($FEEB_{16}$). BB1 was run slightly ahead oftime than BB2. This allowed BB1 to create the two connections with the Engine-#1 ECU.

Observation and Analysis Fig. 8 shows the network trace obtained from the CAN bus during the runtime of the attack. It can be seen that BB1 makes two connections in the beginning by sending a request, RTS and CTS packet for source addresses 11_{16} and $0B_{16}$. The Engine-#1 ECU then transfers data to BB1.

After sometime BB2 attempts to make two connections to the Engine-#1 ECU. For the purpose of this experiment, BB2 acts as the honest party(s). However, BB2 never receives RTS messages from the Engine ECU. At the end of the trace, it can be seen that BB1 keeps its connection open by sending periodic CTSs. As a result, any further connection attempts from BB2 would also be discarded leaving BB2 (acting as the Brake controller and Cruise Control device) starving for the required PGN.

Mitigation Techniques The following attack can have serious consequences on regular J1939 communications. This because, J1939 allows exchange of multi-packet proprietary messages. Disrupting exchange of all multi-packet messages can hamper proprietary message exchange. Authenticating the sending ECU can help in preventing this type of a scenario from happening.

5 Conclusion and Future Work

The J1939 standards are used extensively in commercial vehicles and industrial automation technology. The J1939 protocols run above the CAN bus. Although multiple research efforts have focused on discussing vulnerabilities in the CAN protocol, we believe this is the first work aimed at attacking the J1939 protocol specifications. We illustrated how attacks similar to those performed on the ISO/OSI protocol stack can be performed by a malicious adversary on J1939 protocols. Specifically, we demonstrated three specific denial-of-service attacks using the J1939 data-link layer request and connection management protocols.

Our future work includes uncovering new forms of attacks on the J1939 protocols. A major challenge is providing acceptable security solutions for such attacks. The attacks and the mitigating security solutions will be tested out in real-world scenarios to demonstrate their efficacy. We also plan to evaluate various security solutions in terms of their efficacy, resource utilization, usability, and cost. We will also explore trade-offs among proposed security solutions and provide recommendations for best practices.

6 Acknowledgments

This research was partially supported by the National Science Foundation under Grant No. 1619641 and Grant No. 1619690.

References

1. Bosch, R.: CAN specification version 2.0. Rober Bosch GmbH, Postfach 300240 (1991)
2. Burakova, Y., Hass, B., Millar, L., Weimerskirch, A.: Truck hacking: An experimental analysis of the sae j1939 standard. In: 10th USENIX Workshop on Offensive Technologies (WOOT 16) (2016)

3. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T., et al.: Comprehensive experimental analyses of automotive attack surfaces. In: USENIX Security Symposium. San Francisco (2011)
4. Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive CAN networks–practical examples and selected short-term countermeasures. In: International Conference on Computer Safety, Reliability, and Security. pp. 235–248. Springer (2008)
5. Kleine-Budde, M.: SocketCAN–The official CAN API of the Linux kernel. In: Proceedings of the 13th International CAN Conference (iCC'12), Hambach Castle, Germany CiA. pp. 05–17 (2012)
6. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., et al.: Experimental security analysis of a modern automobile. In: 2010 IEEE Symposium on Security and Privacy. pp. 447–462. IEEE (2010)
7. Linux-CAN / SocketCAN user space applications: Can-utils. `https://github.com/linux-can/can-utils`
8. Miller, C., Valasek, C.: A survey of remote automotive attack surfaces. Black Hat USA (2014)
9. Society of Automotive Engineers: Serial control and communications heavy duty vehicle network - top level document (2013), `http://standards.sae.org/j1939_201308`
10. Society of Automotive Engineers: Data link layer (2015), `http://standards.sae.org/j1939/21_201504`
11. Society of Automotive Engineers: J1939 Digital Annex (2015), `http://standards.sae.org/j1939da_201510/`
12. Studnia, I., Nicomette, V., Alata, E., Deswarte, Y., Kaâniche, M., Laarouchi, Y.: Survey on security threats and protection mechanisms in embedded automotive networks. In: Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on. pp. 1–12. IEEE (2013)
13. Wolf, M., Weimerskirch, A., Paar, C.: Security in automotive bus systems. In: Workshop on Embedded Security in Cars (2004)