Detecting CAN Attacks with J1939 Specification Powered State-based Rules

Matthew Rogers
Department of Computer Science
University of Oxford
Oxford, United Kingdom
matthew.rogers@cs.ox.ac.uk

Kasper Rasmussen
Department of Computer Science
University of Oxford
Oxford, United Kingdom
kasper.rasmussen@cs.ox.ac.uk

Jassim Happa
Information Security Group
Royal Holloway, University of London
Egham, United Kingdom
jassim.happa@rhul.ac.uk

Abstract*—Millions of modern day industrial vehicles use J1939, a software standard operating on top of CAN. Having a software standard allows for interpreting the data field without extensive reverse engineering, making hacking the CAN Bus easier. In this paper we propose an adversarial model for CAN and J1939, a rules based intrusion detection system for any J1939 system, an evaluation study and a set of recommendations for defending systems that make use of CAN. The adversarial model shows what attacks an adverary is capable of with the J1939 specification, and how a defender can measure the impact of these attacks through capability degradation, covertness, access, and fleet scalability. When designing security around our adversarial model into the CAN Bus we sought to minimize additions to the system, and modifications to existing Electronic Control Units (ECUs). We propose an intrusion detection system which utilizes state-based static rules for comparison of arbitrary data from any J1939 message, and only requires one new device silently connected to the CAN Bus, ensuring it has no effect on CAN Bus bandwidth. We wrote over 30,000 rules to maintain the specification's restrictions on data values, watch for inconsistency, and detect rogue device transmissions. Our experiment consisted of injecting malicious messages, firmware upgrades, jamming attacks, and intentional data mistakes into a C7 Caterpillar Diesel Engine Control Module using a physical implant and testing against rental truck data. After adjustments to transmission intervals not exactly matching the specification, we have zero false positives on a 90 minute data capture from a rental truck, and detected transmitting rogue devices, such as physical implants, maintenance devices, and telematic pivots, for fixed-rate messages on our testbench. In this paper we demonstrate the effectiveness of data based analysis, and modify the threat model for J1939 systems from rogue devices to existing devices.

1. Introduction

Since the early 2000s millions of industrial systems [1] have taken their existing Controller Area Network (CAN) Bus infrastructure and added a software standard, J1939 [2], to simplify communication between the different electronic control units (ECUs) controlling the vehicle. The standard was initially designed for ground vehicles, but is now common place across agriculture and forestry equipment, military vehicles, marine vessels, power generators, and much more. While this is useful for industrial systems, the underlying infrastructure is still CAN, a serial data data bus protocol with no authentication, or effective security mechanisms. For the last decade academia and enthusiasts continually showed hacking an automobile is possible with access to the CAN Bus, even going as far as remotely gaining access. The J1939 standard only simplifies the hacking process by removing the need for reverse engineering the proprietary CAN messages of consumer automobiles. Not only does this simplify hacking single vehicles, it makes attacks agnostic to installed ECUs, enabling non-targeted attacks across fleets of heterogeneous vehicles.

We propose using the J1939 standard for defensive purposes. Instead of relying purely on header and timing data we can analyze the data field, making sense of the 8 bytes of data previously left untouched for practicality's sake. We begin this research with 2 premises: we can only add a single device to the J1939 Bus without modifying any existing ECUs, and we cannot have any false positives. Modifying every installed ECU is expensive, and discourages future firmware upgrades, effectively discouraging security. False positives generally risk alert fatigue, causing true positives to go unnoticed. The safety critical systems typically found running J1939 are too valuable for any level of false positives to be acceptable.

To support these restrictions and the J1939 data field we propose an intrusion detection system (IDS) based on a rules framework. Each rule compares translated J1939 data fields between different message IDs, and are deliberately defined. By explicitly defining rules for the different states of the system we can easily interpret the actions of the adversary, and design them such that no false positives occur. We incorporated the J1939 standard into our rules, building many automatically, reducing the computational complexity required by a model, and naturally restricting the message space for any future model based work. With these rules and our discussion on offensive techniques enabled by J1939 we propose four primary contributions:

- We propose an adversarial model for attacks on J1939 with a focus on covert attacks. The impact of these attacks is measured by 4 criteria: covertness, access, capability degradation, and fleet scalability.
- We propose a set of recommendations for defending systems which make use of CAN.

- We propose a state-based rules framework based on the J1939 specification, designed for readable alerts, explicit attack coverage, and simple rule development.
- We developed a proof of concept rules based IDS which detects rogue transmitting devices and variations from the J1939 specification with zero false positives on 90 minutes of truck data using our rules framework.

We begin by providing a background on CAN and J1939 systems and previous work for defensive and offensive techniques in Section 2. Our system model in Section 3.1 defines a starting point for our attacked system and the capabilities of our IDS, while our adversarial model in Section 3.2 defines the capabilties of the adversary, as well as a series of examples attack attacks we tested on our testbench defined in Section 6. We analyze these example attacks in terms of our proposed metric for measuring attack impact. In Section 5 we propose our J1939 IDS rules framework and implementation. Section 7 and 8 provide the experiment and results from utilizing our earlier proposed attacks on our testbench. An analysis of the security guarantees provided our system and adversarial model is in Section 9. We lay out final thoughts on the pros and cons of a rules based approach and what to do after detection in Section 10 before concluding the paper in Section 11.

2. Background

In this section we will provide a background on the CAN and J1939 architecture, as well as existing literature on offensive and defensive techniques affecting these architectures.

2.1. Protocol Background

CAN is a serial data bus protocol invented in the 1980s by Bosch [3], and is used in most modern ground vehicles. It operates using two wires labeled CAN-L and CAN-H. Each device attached to the CAN Bus reads the differential voltage between CAN-L and CAN-H to determine the signal. A '1' is the default and referred to as the recessive state, with a differential voltage of 0. '0' is the dominant state with a differential voltage of approximately 2V. If any ECU is transmitting a '0' signal it overrides the recessive state, and transmits a 0 across the wire. This is built in the J1939 ID field as priority, using the first 3 bits to avoid collisions, as if any ECU detects another ECU transmitting with a lower priority value it will stop transmitting during the initial arbitration period. Each ECU then waits for 11 recessive bits before trying to transmit.

J1939 is a software standard created by the Society of Automotive Engineers [2], which tells ECUs how to interpret the contents of an extended CAN packet, as seen in Figure 1. Three bits of the message ID indicate priority, a function of the CAN protocol such that the messages with lower priority values will speak first in the event of multiple ECUs trying to speak at once. The unique message identifier is referred to as Parameter Group Number (PGN), and takes 18 bits. The last 8 bits of the PGN can be a destination address depending on bits 3-10, but the addressing scheme of J1939 is built in software, so our IDS can read any message regardless of the destination address. The final byte of the header is the source address (SA) of the message. From a security standpoint the addressing scheme is mostly interesting for traffic patterns within the CAN Bus, but can be spoofed by an attacker by changing this byte to any number. That said, the J1939 specification specifies the critical equipment that claims addresses 0..127 across all systems, allowing for some inference of the types of messages coming from each address. For the data section each PGN is broken down into Suspect Parameter Numbers (SPNs), that tell a receiver how to interpret different grouping of bits across the 8 bytes. Across the 1952 PGNs there are over 8500 SPNs, with most messages having a defined transmission interval, resolution, offset, data range, operational range, units, and description. We parse all of this data from the J1939 Digital Annex (DA), an excel spreadsheet containing most message related specification information, for our IDS, later described in Section 5.

2.2. Attacks

Hacking CAN involves reverse engineering a vehicle to determine the proprietary CAN messages, and then injecting arbitrary messages to manipulate the system into a desired state [4], [5]. Most work on offensive techniques focus on preventing the system from operating, manipulating the brakes, or other safety critical functionality, sometimes by remotely pivoting through a telematic device [6], [7], [8]. Trends indicate more safety critical systems will use telematic solutions for predictive maintenance [9], providing additional attack vectors. In J1939 hacking becomes simpler as the specification removes the need for reverse engineering, while only requiring the same hardware one used when hacking consumer CAN devices [10], [11]. We will elaborate more on specific attacks in Section 4.3, but generally past work focuses on denying service, modifying vehicle speed, or triggering the brakes at will as a proof of concept for safety disrupting effects. This paper contributes more work on covert attacks, such as enabling diagnostic errors, and ways to measure the impact of those attacks along capability degradation, fleet scalability, access, and covertness.

2.3. Defense

We often find timing solutions to serial data bus exploits. Cho and Shew built a timing model for each ECU and used variations in it for alerts. It had a 0.055% false positive rate [12]. In the MIL-STD-1553 space we found similar work building a Markov chain mode to detect variations in message order and timing [13]. These papers rely on serial data bus systems being much more regular than traditional TCP/IP networks. We have not found a timing defense explicitly for J1939 systems, though existing research which supports extended CAN messages inherently supports J1939. They would simply require retraining. In a similar vein Pawelec et al and Butler used machine learning to identify anomalies in the data field for CAN messages [14], [15]. Instead of trying to add security through anomaly detection, Murvay and Groza added authentication through modern cryptography [16]. We avoided this approach because it required modifying existing ECUs, and we had concerns about the overhead

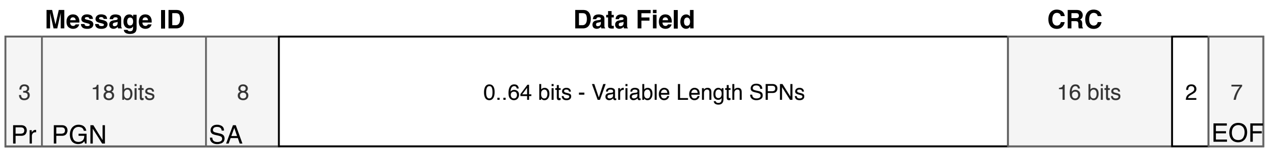


Figure 1. A J1939 Extended CAN packet where any unlabeled numbers are bit lengths, Pr is the 3 bit priority field, SA is the source address, and EOF indicates the end of frame.

for safety critical messages. All of the aforementioned anomaly detection based work functions best for fixed-rate messages. We found no intrusion detection work for CAN systems using explicit rules, nor analyzing the data field beyond a few hand selected fields for maintenance and data visualization purposes [17], [18]. Our rules based approach, in addition to timing based rules for fixed-rate messages, has a series of watchdog rules which watch for specification or state breaking messages, even for non-fixed-rate messages. We elaborate on watchdog rules in Section 5.2.3.

3. System and Adversarial Model

In this section we will define our system and adversarial model. The system model defines our CAN Bus infrastructure and how our IDS is connected. The adversarial model defines the capabilities of the adversary.

3.1. System Model

Our system model is defined in Figure 2. It is a CAN Bus which operates by using a differential signal between two wires designated CAN-L and CAN-H. Lowering the voltage of CAN-L and raising CAN-H transmits a '0'. ECUs transmit CAN messages on these wires, sending it to all nodes connected to the CAN Bus. 120 Ohm resistors act as terminal resistors between the two wires, ensuring the voltage of one does not affects the voltage in another, and providing resistance to RF interference. The specific ECUs vary by vehicle, but some example systems controlled by ECUs include: powertrain, transmission, engine, brakes, emergency brakes, gauge clusters, steering, and more. To listen to these ECUs any device with the appropriate connections only has to connect to these two wires. All ECUs in our system model utilize the J1939 standard, and are trusted devices using known firmware. A description of the specific devices connected to our system model can be found in Section 6.

Our IDS is connected to the two wires in our CAN Bus, giving it full access to the Bus. It is also capable of reading current across its own pullup resistors, those indicated in the CAN Transceiver. Full access implies reading all traffic on the Bus, and transmitting any data across the Bus. The IDS is then able to interpret any read data using the J1939 standard, and compares those data values against each other or values within the specification to send alerts. Alerts can be provided wirelessly or via a local indicator light, we will assume this added security device is secure from wireless threats, and can indicate if uninstalled from a system. The attacker's device is an external attack vector, as we will define in Section 4.1, and has full access to the Bus. The attacker has not added modifiers to the line, and

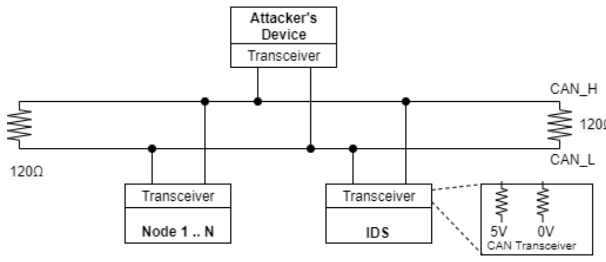


Figure 2. System Model for a CAN Bus system

so any transmitted signals are visible to our IDS. These transmissions can be J1939 packets, malformed J1939 packets, or messages manipulated by the attacker. Notably breaking the J1939 specification by transmitting outside the operational range or a field is still a valid packet, though our IDS will alert on these sorts of mistakes.

We identify two kinds of transmissions, fixed-rate and non-fixed-rate. Fixed-rate messages occur on a set interval, defined by the J1939 specification. Most ECUs directly connected to the Bus transmit at this set interval regardless of whether the feature is in an on or off state. We assume any sensors directly reporting data to the CAN Bus use fixed-rate transmissions. Non-fixed-rate messages are typically used for data on request, or sending updates such as diagnostic trouble codes.

3.2. Adversarial Model

For the purposes of this paper we will consider something successfully attacked if an attacker can transmit or modify arbitrary messages on the CAN Bus undetected. Our adversary could attack CAN systems for many reasons. They could seek to prevent the vehicle from starting, moving, or any other capability. Alternatively more covert attacks can increase costs for the system maintainer by delaying or causing early maintenance, providing an economic advantage to the attacker.

We assume the adversary is not manipulating the physical wire, or adding in line signal modifiers, as stated in our system model. We give the adversary full knowledge of the system, and an access vector onto the CAN Bus with read and write access to the Bus. With this access we summarize their capabilities as being able to pull the CAN Bus signal high or low by manipulating the voltage across each wire at any time.

This allows an attacker to inject arbitrary messages across the CAN Bus, or manipulate each bit of messages sent by legitimate ECUs. For the rest of this paper we will refer to injecting messages as Arbitrary Message Injection (AMI) and manipulating bits as a manipulation attack. AMI allows an attacker to change safety critical functions, enable diagnostic trouble codes, and manipulate any of the 8500 data fields in J1939. One of these messages in particular is used to flash firmware across the Bus allowing for every ECU to be infected once they can transmit J1939 packets. Firmware modification includes removing certain message parsing capabilities, such as responding to maintenance messages, firmware upgrades, or any other desired functionality. But these are for injecting new messages. Alternatively an attacker doing a manipulation attack can modify individual bits as they go across the wire, drastically changing message contents. As part of this manipulation they can pull the signal high effectively jamming the Bus. To the best of our knowledge existing CAN literature does not reference the possibility of manipulation attacks, instead focusing on AMI.

4. Practical Attacks on J1939 Systems

In this section we will describe how to attack and measure the impact of attacks across J1939 systems. Attacks on J1939 are primarily the same as CAN, but more attacks are practical due to the specification.

4.1. Attack Vectors

We focus on 3 methods of gaining this capability: physical implants, infected maintenance devices, and telematic pivoting. In traditional CAN hacking a physical implant is connected via the OBD-II diagnostic port, though more covert options are available. The implant then transmits data, this is the simplest option. A maintenance device looks for diagnostic error codes and uploads firmware to vehicles. These firmware updates are downloaded from maintenance laptops which can be attacked to modify the firmware and upload malicious code to the target. Finally, with the growing trend of adding WiFi or other telematics to systems it is increasingly possible to find ways to pivot from the connection to the CAN Bus [7], [19], [6].

4.2. Measuring Attack Impact

How our adversary hacks a J1939 system may be similar across different industries and platforms, but the impact those attacks have can vary dramatically. We measure attack impact across 4 criteria: covertness, capability degradation, fleet scalability, and access. We measure capability degradation in respect to the missions of the infected system, such as mobility, safety, or system readiness. Fleet scalability is how many systems are infected by the attack, be it one or an entire fleet. Typically this is dependent on the attack vector, it is far easier to infect multiple devices via a telematics flaw or through an infected maintenance device than a hand installed physical implant. We define access by what triggers the attack. This could be a set time from infection, a geofence, or even sustained remote access. Covertness measures the likelihood of discovery, which in current systems is near zero as almost no data logging exists outside of information for insurance [20] or limited emergency info. If there is a data logger, greater reliance on telematics, serious capability degradation, and affecting more systems results in decreased covertness as it arouses suspicion and begins incident response processes. Additionally, the more messages sent the greater likelihood of being detected.

4.3. Example Attacks

In this section we will describe attacks and their potential impact through capability degradation, covertness, and our adversarial model. Access and scalability are determined by the attack vector. We assume they already have an attack vector installed, the cheapest likely being a Raspberry Pi with a CAN shield [21] for $80. These attacks will be referenced in later sections to demonstrate the bounds of our rules based IDS. While some of these attacks exist in other work [10], [11], we highlight them here due to their effectiveness, and to provide contrast for the rarely focused on covert attacks, such as the diagnostic attack featured in this section.

4.3.1. Jamming and Targeted Jamming. Jamming sets the Bus signal to high, causing all messages to be ignored. Nothing in the vehicle which relies on information from other ECUs will function. We have observed a similar attack on a real system, the vehicle usually glides to a stop or has a safety mechanism to violently trigger the brakes. A voltmeter will reveal this attack, though identifying the source automatically is more difficult. However, assuming our adversary is capable of infecting all ECUs with malicious firmware that jams on demand then it does not matter. Either way the defender must reflash all ECUs on the Bus with known good firmware.

Targeted jamming attacks individual ECUs by kicking them off the Bus, resulting in the target ECU no longer speaking on the network. This works by claiming the address of an existing ECU. A particular J1939 message accomplishes this by comparing 8 byte NAME fields, something easily spoofable for the attacker. The lower NAME value claims the address and the other ECU is left to claim another address. However, we observed that most critical ECUs are designed without the ability to claim a new address, instead sending out one message announcing they cannot claim any address, but otherwise remaining silent. Alternatively the adversary can claim every J1939 address ensuring no legitimate device can speak on the network [10].

An actor focused on short term disruption not concerned with being detected may do this attack. More destructive attacks are possible, but require more knowledge of the J1939 protocol, or system specific knowledge. Jamming requires almost no technical expertise, making it ideal for quick, resource limited actors.

4.3.2. Malfunctioning Attack. CAN sends a small error frame at the end of each packet set to all '1' bits. If any of these bits are set to zero all other ECUs on the Bus will ignore that message, and the source ECU will increment a faulty message counter value and transmit it out. If this faulty message counter gets above a certain value the source ECU is programmed to stop transmitting data to avoid flooding the Bus with bad packets [22]. An attacker can force this scenario by setting any of these final bits, as shown by Giannopoulos et al., who demonstrate this capability for an intrusion prevention system [23].

Performing this capability selectively is technically challenging, but also not covert on the CAN Bus. This technique is mostly useful for defenders, as it does not prevent an IDS from observing the messages, and flags them as suspicious, but is a valid way of effectively removing an ECU from the Bus.

4.3.3. Arbitrary Message Injection and Manipulation Attacks. Our adversary is capable of transmitting any data once on the Bus. This includes manipulating inputs to all ECUs, the more components connected to the Bus the more an attacker can manipulate. While testing these attacks we found some engine control modules (ECMs) were explicitly programmed to not accept data over the wire, instead it was a direct input to the ECM. AMI and manipulation are too wide of an attack surface to name all scenarios, but generally an adversary with full system knowledge can manipulate any data field, and do so in a way which appears like a maintenance fault. At scale these attacks would be suspicious, particularly message injection, but there are enough possible attacks to make a single signature ineffective.

4.3.4. Diagnostic Attack. A Diagnostic Attack manipulates diagnostic trouble codes to temporarily hinder systems and activate the maintenance process, wasting resources. Alternatively J1939 allows for clearing diagnostic trouble codes over the wire, causing maintenance faults to not be discovered until the system is more costly to repair. In terms of capability degradation turning on a diagnostic light has no direct consequences, but the long term impact is greater. Even with data logging this attack requires a single message and is covert without system specific knowledge of the parameters for each diagnostic trouble code. The longer this attack remains undetected the more strain is put on the maintenance supply chain, while eventual detection depreciates system operator trust in maintenance faults. This is explicitly a non-fixed-rate message attack.

Actors desiring a subtle, long term degradation of systems will use a diagnostic attack. This may be a nation state causing militaries to spend more on wasted maintenance, or a commercial competitor wanting to gain an economic advantage. We highlight this attack to shift the focus from the purely destructive attacks often discussed in automotive security. Jamming and AMI can have destructive effects, but causing maintenance faults, or non-optimal performance can be far more harmful in the long run and has a low chance of being detected.

5. Intrusion Detection in J1939

In this section we will discuss the full scope of our intrusion detection solution for J1939 systems. After providing a guarantee for detecting manipulation attacks we propose a rules based approach to securing J1939 systems from the aforementioned arbitrary message injection attacks.

5.1. Detecting Manipulation Attacks

Our IDS relies on watching the data field for anomalies. If an attacker is able to manipulate any bit of any message, even those from trusted systems, then it becomes difficult to make any guarantee of security outside of detecting an attacker making a mistake or acting too quickly. Additionally if a manipulation attack is undetected, timing based security models are broken, as the contents have changed without any timing variation. CAN has a built in error detection mechanism for manipulation attacks, but this only applies outside the arbitration field, meaning an attacker could modify the priority, PGN, or source address fields. Modifications to the priority and source address fields are easy to detect with rules described later in Section 5.2.4, but modifying the PGN changes how the message is interpreted. Modifying the PGN means our IDS interprets the message as another, allowing an attacker to send their desired message directly after that message, violating any timing based security guarantees. Given this we must detect manipulation attacks at a lower level than J1939.

We detect manipulation attacks by measuring the current across the CAN Transceiver, depicted in Figure 2. Let us consider the scenario of an attacker manipulating a 0 to a 1. Here the differential voltage between CAN-L and CAN-H is approximately 2V and the adversary is trying to set the differential voltage to 0V. To do so they must lower the voltage of CAN-H and increase the voltage of CAN-L. In raising the voltage of CAN-L such that the differential voltage is 0V the attacker is rapidly increasing the current across the CAN Bus. The same voltage increase occurs in CAN-H when the attacker manipulates a 1 to a 0. Each CAN Transceiver connected to the CAN Bus, including our IDS, contains pullup resistors wired in parallel to each other pullup resistor. It follows that our IDS sees this sudden increase in current. Using this technique we are able to detect any manipulation attack. This means all attacks involving J1939 packets must be done by injecting new packets.

5.2. Rules Based Intrusion Detection in J1939

We started researching security of J1939 with the premise that modifying ECU firmware is not practical. With systems having up to 30 ECUs [22], each with separate suppliers it seems unsustainable. Instead we add a single security device to the CAN Bus, and exploit the broadcasting nature of CAN to capture every packet. By parsing the J1939 DA we interpret every bit in every packet on a standardized system. We use this deep packet inspection with a static rules based intrusion detection system. Given this known information, and a desire to have no false positives or negatives we chose static rules with the premise that some values should always cause alerts. A model based approach, [12], [15], can detect new attacks, but lacks the complete system clarity a list of rules provides. Additionally an attacker can get around existing models by slowly shifting the data field. For such safety critical systems a guarantee of detection for set conditions is preferred, though a model could be effective in combination with our rules framework.

A moving system has states that should never be reached, such as a new device claiming to be an engine as may happen in a targeted jamming attack. We incorporate the idea of states into our rules based approach by adding a precondition field. If this precondition field is evaluated to false, the rule will not trigger, but if the precondition(s) are true, we can test the condition. For instance, the preconditon could check if the engine is powered off.

Our rules understand conditional operands ('$>$','$<$','$! =$','$==$'), and the in set operation. They are written using a custom markup language to reference

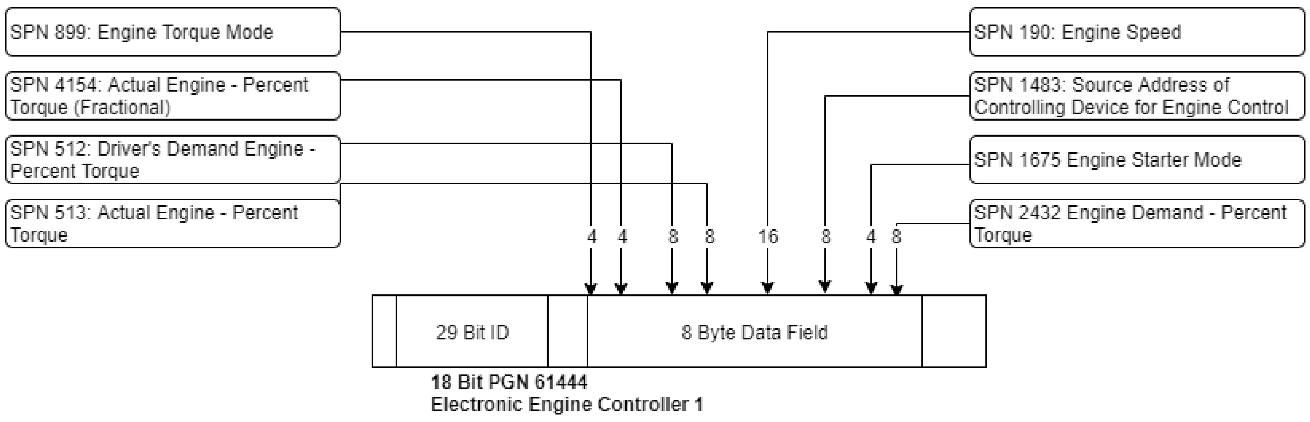
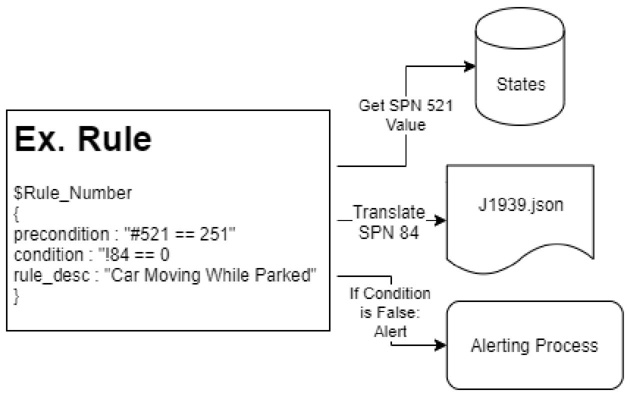

SPN 899: Engine Torque Mode
SPN 4154: Actual Engine - Percent Torque (Fractional)
SPN 512: Driver's Demand Engine - Percent Torque
SPN 513: Actual Engine - Percent Torque
SPN 190: Engine Speed
SPN 1483: Source Address of Controlling Device for Engine Control
SPN 1675 Engine Starter Mode
SPN 2432 Engine Demand - Percent Torque
4
4
8
8
16
8
4
8
29 Bit ID
8 Byte Data Field
18 Bit PGN 61444
Electronic Engine Controller 1

Figure 3. SPN Breakdown for PGN 61444: Electronic Engine Controller 1



```
$Rule_Number
{
precondition : "#521 == 251"
condition : "!84 == 0
rule_desc : "Car Moving While Parked"
}
```

Figure 4. Example rule detecting movement while a vehicle is parked. Triggering on PGN 65265 (which contains SPN 84)

different fields of the J1939 packet for incoming and previous messages, while using SPNs to reference subsets of the data section. We do this by parsing the DA for the data fields as shown in Figure 3. Figure 4 gives an example of our rules syntax. Here we trigger the rule if the PGN contains SPN 84. We evaluate the precondition by grabbing the last known value for SPN 521 from another J1939 message. If SPN 521 was 251 then the system is in park so we evaluate the rule. SPN 84, grabbed from the IDS triggering message indicates ground based wheel speed, if the vehicle is in park we assume this is 0, otherwise we alert. This rule is an example of a state that should not occur and indicates an attacker injecting messages at the wrong time, or a maintenance fault. It is worth noting that the J1939 DA does not have consistent floating point precision for listed values, so any work analyzing the data field should recalculate all values listed in the standard using the resolution, SPN length, and offset for accuracy.

Rules are designed around acting as a watchdog on messages between ECUs, and ensuring all messages meet the specification without false positives. The specification sets transmission intervals, operational ranges, error values, reserved values, etc.. for most PGNs and SPNs. As a watchdog we look for SPNs in one message that dictate the value of SPNs in another message. For instance PGN 0 can set the engine into speed override mode, overriding engine speed to SPN 898; engine speed is in PGN 61444, SPN 190. The rule verifying this is true can be seen in Table 1 under 'Engine RPM Inconsistent with Override'. We evaluate this rule if the incoming message PGN is 61444. '#695 == 1' is a precondition that checks we are in speed override mode, if the last message with PGN 0 we received had SPN 695 set to 1, then we alert if the engine speed from the incoming message is not equal to that set in SPN 898 from the last message with PGN 0. If the rule requires fudging values or raises a false positive it is better as a trained rule, or heuristic.

While rules are able to ensure data consistency once the system is operating, we found it necessary to create a basic learning system to maintain consistency between reboots. These rules check for training before running, and are only evaluated until they pass once to save on performance. A '?' precondition checks for training, then if the source address is not one sent while training we alert. The training should only be done once the car is fully booted (engine and battery on), otherwise it could learn uninstalled states for engine messages. The logic for this is akin to that used in our previously mentioned precondition field

5.2.1. Data Pipeline.

Whenever we receive a new J1939 packet it is passed to our IDS object. For each PGN we look up which SPNs we are tracking for any rules. After updating these tables we evaluate any rules tied to that source address or PGN. Evaluating a rule requires looking up each value in the table, or the triggering messages, then checking if the condition is true or false. A false sends an alert, informing the operator and maintenance crew. The full pipeline can be seen in Figure 5.

5.2.2. Rules Creation Methodology.

Rules are created in three ways:

- We manually identified an attack and sought ways to mitigate it.
- We automatically parsed the J1939 DA and found a clear limitation of values where an attacker can make a mistake (1 byte of data, operationally only 4 bits are used).
- We manually looked for specific states and rules based on the bounadaries of those states for critical ECUs.

Finding data tied to an attack is fairly self-evident. Normally a timing based rule, or explicitly checking the

TABLE 1. EXAMPLE RULES USING OUR CUSTOM SYNTAX

Alert Description	Trigger	Precondition(s)	Condition
Rogue Firmware Update	PGN 26624	#9761 == 0[a]	!9773 != 1[b]
Transmission Interval Variation	PGN 65265	N/A	@65265 == 100[c]
ECU Disconnected	PGN 60928	N/A	> == 254[d]
CAN Error Signal	PGN 61667	N/A	!7750 ≥ 1[b]
Error State	PGN 256	N/A	!4255 == 14[b]
Reserved Bit Overwritten	PGN 0	N/A	!690 == 2[b]
Engine RPM Inconsistent with Override	PGN 61444	!695 == 1[b]	!190 == #898[a]
Address Claimed out of Sequence	SA 1	N/A	@>0 < 5000[e]
Data Outside Operational Range	PGN 0	!518 != 130[b]	-125 ≤ !518 ≤ 125[b]
Installed Feature Reporting Uninstalled	PGN 61444	;650 == 255[f]	!650 == 255[b]
Uninstalled Feature Reporting Installed	PGN 61444	;650 != 255[f]	!650 != 255[b]
Address Claimed While Moving	PGN 60928	N/A	#84 == 0[a]
Device ID Inconsistent for Address	PGN 60928	N/A	#2805> == !2805[eb]

[a]# is the last value for an SPN from another message [b]! indicates the SPN is from the triggering message [c]@ is the time in milliseconds since the provided PGN was transmitted [d]indicates a SA [e] >used after #, ; or @ indicates the last value/time for each source address [f] ; is the historical value for an SPN from the triggering message

conditions that set up the attack are enough, though setting the precondition(s) such that no false positives occur without test data is difficult. Sometimes the attack cannot be detected by a simple rule. In these cases the defender must weigh false positives, the realities of their incident response process, and the threat model of their system.

Most of our rule creation time was spent on parsing the J1939 DA for clear limits on the specification. Initial attempts utilized natural language processing libraries to find connections between data fields or messages automatically, these were mostly unsuccessful. The insights gained by parsing how the standard used keywords such as command, limit, request, inhibit, indicator, feedback, and others influenced the design of our IDS. Namely the decision to focus on state by adding the precondition field, and basing the rules syntax on SPNs. Some rules could be tied together using matching references in the SPN or PGN description field, but these sorts of searches were often only noise. Throughout this process we spent at least 150 hours parsing, analyzing, and trying to create rules from the J1939 DA. For defenders or any individuals wishing to expand a rules based IDS approach this is not very efficient once the baseline of rules and parsing is accomplished. Instead we recommend focusing rules specific to new attacks or specific systems.

Due to our ECM focused test bench, and generally thinking an Engine is the most important part of the vehicle, many of our rules are focused on Engine based messages. However, none focus on the desired limitations and normal behavior of a vehicle. A military vehicle and a truck may use the same ECM, but their normal operating parameters are different, as are their priorities. Developing rules takes time and so the analysis should be focused on whichever ECU or capability is deemed most critical. For a military system this may be maintaining mobility, or weapons systems, while a truck is likely focused on mobility and any mechanism controlling cargo. We may be able to automatically determine which ECUs are most critical through message frequency and the J1939 priority field, which indicates a higher priority at lower values from 0 to 7. These two metrics are positively correlated as shown in Figure 6, though this calculation depends on how many emergency ECUs, such as emergency brakes, are directly connected to the CAN Bus. This focus becomes more important as this work shifts from intrusion detection to intrusion prevention.

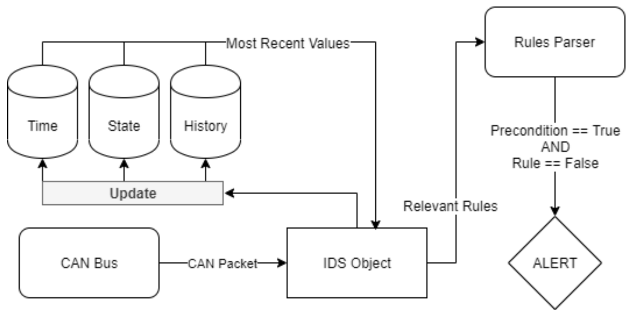


Figure 5. IDS Data Pipeline, where relevant rules are those applied to the incoming source address or PGN

5.2.3. Categories of Rules. Rules are split into two main categories: detecting transmitting rogue devices, and watchdogs on the data field. We define rogue devices as devices listed in Section 4.1 as attack vectors: physical implants, infected maintenance devices, and telematic pivots. These rogue devices can send arbitrary messages, at any time, from anywhere on the Bus. If a device is already transmitting their desired PGN on a fixed-rate (e.g every 100ms), any rogue transmissions will trigger an alert on this timing deviation through a transmission interval rule. As an attacker the simplest way around this is targeted jamming, over the wire firmware upgrades, or malfunction attack techniques [23]. Specific rules watch for these attacks. We theorize an attacker now has to physically remove an ECU from the CAN Bus, flash it with malicious firmware, and then reinstall it to inject messages with known transmission intervals undetected.

The second category of rules act as an ECU watchdog. Now an attacker must be listening across the Bus,

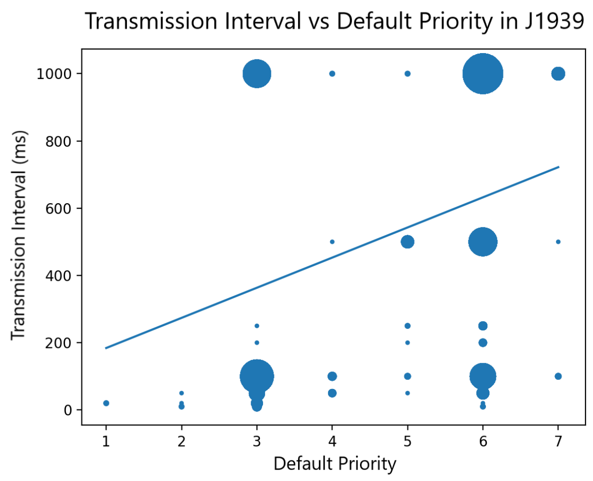


Figure 6. Transmission intervals scaling with default priority for J1939 messages with the single crash override priority zero message excluded. Data is sourced from the J1939 Digital-Annex

modifying their attack depending on the state of the system. If we allow our IDS the ability to request data across the network the attacker must also respond correctly on the fly, maintaining data consistency with known system configurations while incorporating their planned attack. Across 1160 PGNs not referencing a PDF in the specification we found an average SPN count of 6.3, with a maximum of 100 SPNs. Meaning an adversary would have to manage approximately 6.3 SPNs, though our observations indicate more safety critical systems tend to have more SPNs as they are more complex. An adversary with full knowledge of rules and the J1939 specification is capable of circumventing these rules, but increasing software complexity leads to a higher likelihood of bugs.

5.2.4. Implementation. By parsing the J1939 DA we created 30,000 rules which require no training and rely on systems following the specification. We created an additional 10,000 rules across 3 categories relying on a simple training model to track persistence of transmitted fixed-rate PGNs, installed features, and utilized source addresses. Examples of these rules can be seen in Table 1. All J1939 DA parsing, rule generation, and IDS functionality is written in Python3.7, primarily for rapid feature development as we gained new insights into the J1939 standard. We were unable to completely automate rule generation as the J1939 Digital Annex is written for engineers to manually reference, and is sometimes inconsistent between messages. Parsing the DA revealed enough patterns or information to write many of the rules, though several had to be handwritten for specific attacks such as targeted jamming. All rules are verified through over 80,000 total unit tests.

6. Methodology

To test the effectiveness of our rules we created a testbench using a C7 Caterpillar Diesel Engine Control Module (C7 ECM), CommaAI Panda, Neousys Rail Certified Computer, engine emulator, and CAT controller. The full testbench can be seen in Figure 7. The Neousys computer reads data off the CAN Bus, uploads it to Elastic Search / Kibana (ELK) [24] for data analysis, and sends all data to our IDS both locally and off-device. The Panda is designed for crowd sourcing autonomous driving data, but it is also capable of sending arbitrary extended CAN packets, acting as our physical implant. For maximum testing time we run the C7 ECM using a custom engine emulator to maintain realistic engine traffic. We confirmed the representativeness of this data by finding it similar to 1.5 million CAN packets collected from a truck using J1939.

The CAT controller is connected to a maintenance computer, which displays some of the J1939 data we are interested in. We used this data to verify our J1939 parsing is working correctly, and observe effects from attacks. After verifying data parsing we parsed the J1939 DA to create the rules mentioned in Section 5.2.4.

7. Experiment

In previous sections we make two claims:

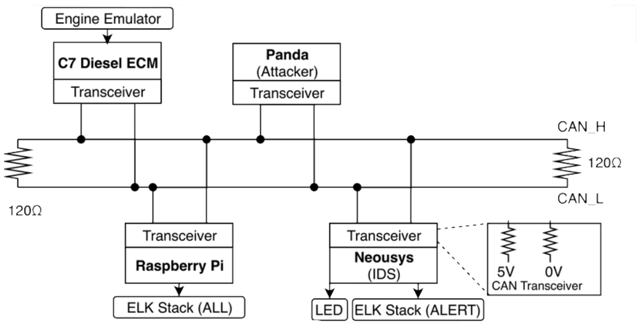
- We can detect transmissions from rogue devices.
- We can detect variations from the J1939 specification.

The Panda in our testbench serves as a physical implant, with all the capabilities prescribed to our adversary. We still assume the J1939 wire has not been manipulated, and direct line signal manipulators have not been attached. Through the Panda we transmitted a targeted jamming attack, an on the wire firmware upgrade, a diagnostic attack, and crafted messages with different safety critical data. Specifically for AMI we sent packets which: manipulated engine speed, enabled cruise control, and enabled engine speed override mode, though our perfect adversary could set any value for any data section. To verify a normally operating system does not result in false positives we ran our IDS against a rental truck for 90 minutes. We did not run attacks against this truck, instead relying on our testbench for safely observing the effects of attacks on the Engine.

Variations from the specification largely rely on the attacker making a mistake. To simulate this while attacking we flipped byte endianness, and set bits to '1' in some SPNs with unusual bit lengths or positions (odd numbered lengths, beginning in the middle of a byte, etc..). While this is not entirely representative, as researchers we are aware of all rules created, but also the plethora of parsing and packet mistakes we made in building the infrastructure for this experiment.

8. Results

On initial rules testing we found transmission interval rules generated false positives. Transmission intervals were consistent across messages sent by a device, but were typically within 5ms of the specification. So a message designed to transmit every 100ms will instead transmit every 103ms. Operational ranges, uninstalled values, and the rest of the J1939 data conventions were maintained by our testbench. While this is fixable without trained rules by broadening transmission intervals, for more frequent messages an attacker is able to inject up to four malicious messages per one legitimate message in the worst case. Given this we believe it is better to create a trained timing



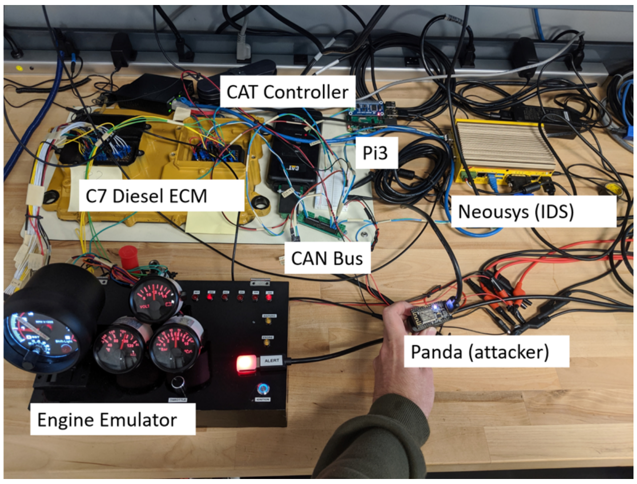


Figure 7. System model and testbench. ECM data is collected and analyzed in ELK stack, and locally on the Neousys computer for IDS functionality. The Panda injects extended CAN Packets

model for transmission intervals akin to Cho's work [12], but with the J1939 DA as a guide for messages that will flip transmission intervals in different states.

Over 90 minutes we collected 1.5 million CAN packets from our rental truck. In those 1.5 million packets we saw 82 out of 1952 PGNs, 3504 of our 30,000 rules, and we enabled an additional 20 rules designed around the expected network addresses for highway systems. Across these 82 PGNs and 3504 rules we observed zero false positives in normal operation. A chart displaying this data can be seen in Figure 8.

We detected the targeted jamming attack through the victim ECU transmitting it lost claim to an address immediately after the attack. Firmware upgrades were detected by logging any firmware activity, this is a simple solution but without data logging was impossible. Once we could detect attempts to silence known ECUs we found with slightly trained transmission interval rules ECUs transmitted consistently enough to detect transmitting rogue devices for fixed-rate messages. AMI was always undetected by a perfectly informed adversary transmitting non-fixed-rate messages, or messages not normally seen on the device given our current rules, though this was circumventable through a few minutes of training for which PGNs are transmitted by each source address. We found this list remains constant after a few minutes of operation. With ECU documentation for what triggers each diagnostic trouble code it is possible to monitor recent data for the proper preconditions with a rule. Currently an attacker can transmit a command message to enable any diagnostic lights, set any diagnostic alerts, or clear diagnostic alerts. With AMI an attacker can set up the conditions for a diagnostic alert, but that could involve an ECU they have not infected, and fixed-rate messages, effectively giving the non-fixed-rate message a transmission interval defense. When we simulated making a mistake we triggered watchdog alerts, but this is pure luck based on the SPN layout of the Electronic Engine Controller 1 message we injected. Diagnostic Attacks were not detected by our automatically created rules.

In terms of performance our Python implementation took approximately 60 microseconds to analyze from

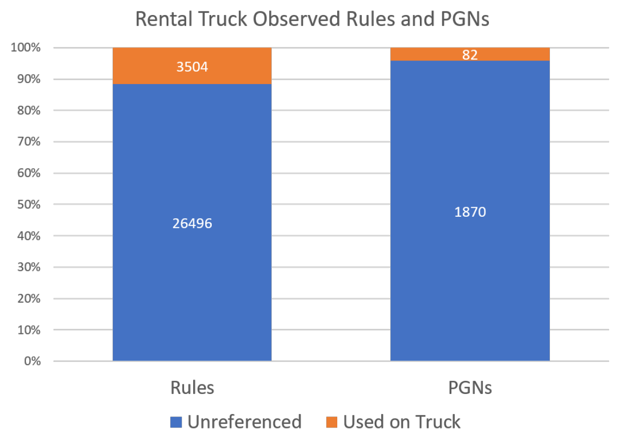


Figure 8. An analysis of 1.5 million truck messages where we found 11.68% of our rules were triggered, and 4.2% possible PGNs were sent

ingesting the full message. Based on data captured from a rental truck over 90 minutes, on average one message is transmitted every 60 microseconds. Meaning if a message with a large number of rules is being processed it could take a bit longer and we would fall behind. If re-implemented in efficient C, a field programmable gate array (FPGA), or a device with more system resources we are confident our approach can handle a heavy traffic CAN Bus.

9. Security Analysis

In the adversarial model we defined a successful attack as manipulating data or injecting arbitrary messages across the CAN Bus undetected. To provide a security guarantee against an attack we will focus on how we detect manipulation attacks and arbitrary message injection. We are able to detect manipulation attacks by measuring for sudden current increases across the pullup resistors in the CAN transceiver of our IDS, as described in Section 5.1. Now an attacker must inject new messages onto the CAN Bus to perform a successful attack.

To perform an AMI attack, an attacker must send a fixed-rate, or non-fixed-rate message. We provide different security guarantees based on this rate. Let us discuss the security model for an adversary transmitting fixed-rate messages. First the attacker must decide on a PGN and SA. Due to the training mentioned in Section 5.2.4 the attacker must use a fixed-rate PGN and corresponding source address previously seen across the CAN Bus. When the attacker transmits their desired fixed-rate PGN our IDS reads the previously seen PGN and checks the transmission rate of that message. If the time between the attacker's message and the previous time we saw that PGN is not equal to the known transmission interval, we alert. Given this, an attacker must ensure that no other ECUs are sending their desired PGN. It follows that to transmit their desired PGN the attacker must prevent the original ECU from speaking. There are 4 ways for an attacker to prevent the original ECU from speaking outside of physically removing it: flash the original ECU with malicious firmware to control all transmissions, execute a malfunctioning attack as described in Section 4.3.2, perform a targeted jamming attack as described in Section 4.3.1, or perform a jamming attack, also described in Section 4.3.1, when that device is transmitting. We can already detect a jamming attack, as the attacker must force the line high continuously, effectively a prolonged manipulation attack from the perspective of our IDS. The other 3 techniques each cause a specific PGN to be transmitted across the CAN Bus. Our IDS will trigger based on these PGNs as shown in Table 1, meaning an attacker must prevent our IDS from observing the PGN. This is impossible based on the assumptions of our system model.

Now let us assume an adversary is transmitting non-fixed-rate messages. These include diagnostic messages, data sent on request, and operations that only occur in certain states. Watchdog rules allow for codifying system knowledge into the IDS, meaning if we have sufficient documentation or system expertise we can ensure conditions that legitimately trigger these non-fixed-rate messages actually occur. As these conditions are contained in fixed-rate messages, an attacker must set those conditions via fixed-rate messages before transmitting the non-fixed-rate message. In J1939 systems we have not observed a non-fixed-rate message which causes the vehicle state to change directly, as most messages broadcast on or off at a fixed-rate, meaning our security guarantees for fixed-rate messages apply to non-fixed rate messages. This requires a defender with the time to develop rules, and full knowledge of the system, limiting the effectiveness of this approach from system to system. More broadly we use watchdog rules to detect deviations from the J1939 specification. Specification based rules provide no security guarantee, instead relying on the attacker making a mistake; however, these can apply to all J1939 systems without training.

Given the above we conclude that an attacker must use a supply chain attack or physically replace the ECU sending their desired message to send fixed-rate messages, as we can detect modifications to the ECU over the Bus, requiring any firmware modifications to be done while uninstalled. We guarantee detecting state-based non-fixed-rate messages, and we simplify the process for embedding this knowledge into our IDS with the custom rules syntax for comparing arbitrary data fields. Through our IDS we propose an extensible security solution which switches the security model for CAN systems from a perimeter based model, to an assumed breach model, suspicious of every transmission.

10. Discussion

In this section we will discuss the pros and cons of our rules based approach, differentiating emergencies from cyber attacks, parsing the J1939 DA, what to do after detection, and general security advice for system operators and designers.

10.1. Designing Rules

As covered in Section 5.2.2, we spent a majority of the initial research time on looking through the J1939 DA, trying to automatically find patterns across messages and SPNs. To build on this work rules analysis would be ECU focused, detecting attacks such as a diagnostic attack by documenting the exact state required for that diagnostic trouble code to trigger. The data required differs from our largely more general rules based on the J1939 DA. If documentation exists, we could research that state, but otherwise we will need to reverse engineer the firmware for the triggering conditions. We believe a defender with close ties to the manufacturer can easily get this data and make these rules assuming the documentation refers to the triggering data fields in terms of J1939 SPNs.

10.2. Emergencies and Maintenance Flaws

A fundamental problem with vehicle security is emergency scenarios appearing as anomalous messages. Any timing based model will always flag on them, and if we built one of these models into an intrusion prevention system we could jeopardize the safety of someone in a real emergency. Here lies a real benefit of rules based on state. If we are confident in what triggers the emergency scenario we can at least turn false positives into meaningful information. For instance, lets say we have a vehicle with an emergency brake, and a forward facing camera that reports on the distance to the nearest obstacle in front of it. J1939 has a specific PGN for this camera, so we can reference this distance SPN, and set a precondition check that the camera does not see anything within some close distance. We trigger this rule on the emergency brake message, and alert if the emergency brake is set to true while the camera sees nothing. Now we will not alert for a legitimate rear end crash, that one may slam the brakes or use an emergency brake for, but there could be another scenario for setting off the emergency brake, or perhaps the camera was unable to recognize a specific object. From a maintenance perspective that is useful information, making our false positives a bug tracker of sorts. We've observed a similar case with poorly mounted ECUs continuously disconnecting and reclaiming their address. The address reclaiming looks like an attacker kicking off a legitimate device before spoofing it, but its also an indicator to a technician to more securely mount those ECUs once we have identified it as a false positive.

10.3. Reliability of the J1939 Specification

The J1939 DA is generally upheld, but the full accuracy is reliant on the engineer's implementation. For this work we parsed the J1939 DA using a Python script we developed to turn it from an XLS file into the JSON format. In doing so we ran into many issues with inconsistent floating point precision, fields with missing values, and seemingly inconsistent methodologies for denoting SPN length, the formatting of variable length messages, and the actual length of multipacket messages as seen from our testbench to name a few. Commercial databases for reading the J1939 specification exist, though for maximum control of the data we recommend parsing the DA directly. This allows for adjustments to any defensive work reliant on an ECU breaking the specification. The biggest risk of false positives from rules is an ECU outright breaking the J1939 specification. We have yet to observe this in our test bed or in the trucking data, but the DA lists 1863 unique manufacturers, and we have not tested they all follow the specification, only the ones included in our experiment.

10.4. After Intrusion Detection

With an IDS the system maintainer must decide what to do after an attack. If the alert is on a rogue device rule, the maintainers may begin an incident response process to find that device and prevent fleet propagation. Physical implants can be hidden anywhere on the CAN Bus, or via a common maintenance port. Maintenance devices require hardening the computers they download firmware from, making it a more traditional security problem. Telematic pivoting removal varies by implementation, but is the most complicated, especially if a critical ECU is sending telemetry data [9]. Alternatively an IDS can be modified into an intrusion prevention system (IPS). For a rogue device, existing IPS techonology for CAN [23] could be effective if our IDS were reimplemented into an FPGA for speed. An adversary who infects existing ECUs is equally affected by this, but now the defender is disabling their own ECU. If it is safety critical, such as an ECM, the system will effectively no longer have an engine. Our testing and discussions with groups with offensive experience revealed vehicles will either maintain momentum and roll to a stop, or have a safety mechanism to activate the emergency brake. Either way if the mission is to maintain mobility, stopping is unacceptable. Emulating the firmware running on each ECU is possible for recovery but adds wiring costs to connect our security device to safety critical sensors and analog components. Though wiring is cheaper than having complete redundant devices as many military systems do.

10.5. Security Recommendations

Beyond adding security devices to these systems, system operators, maintainers, and designers can still increase the difficulty for hackers. Operators have the most experience with how the vehicle performs and are responsible for noticing any deviations in performance, and looking for any rogue physical implants. Maintainers should have consistent, and meticulously logged maintenance schedules, leaving no confusion about which firmware updates are legitimate. Part of this is checking hashes and verifying the firmware uploaded to a system is the same as downloaded by the ECU manufacturer. The maintainer should follow security best practices on their maintenance computer, installing OS updates, limiting third party software, and only using it for maintenance purposes to lower chances of infection.

Designers should focus on the risk associated with adding each ECU. An ECU from a manufacturer who writes insecure firmware can make any intrusion detection systems useless. Validating the security of an ECU may be difficult, but looking at the level of software documentation for the firmware is a good indicator for the level of software engineering, and thus quality assurance, that went into the ECU.

An ECU with an insecure telematic solution makes any physical security the vehicle has pointless. We suggest having a single ECU exfiltrating data off the CAN Bus. The device exfiltrating data should never need to send messages on the Bus and so measuring for voltage changes can be an easy indicator. Any wireless connections should be completely disconnected from the CAN Bus, instead using automotive ethernet or an equivalent protocol. If the vehicle design requires commands from a wireless connection directly to the CAN Bus, then industry standard authentication on that connection is necessary, otherwise an adversary can execute any AMI attack by hijacking the connection. Even with authentication, message sanitation is also recommended to prevent truly arbitrary messages, and firmware exploits.

11. Conclusion

In this paper, we propose a data focused intrusion detection system for J1939 with a focus on minimal installation costs, zero false positives, and no overhead to the CAN Bus. By parsing the specification into a machine readable format we can reference arbitrary data fields. We incorporate these data fields into a custom rules syntax for easy comparisons of the last value set for each of the 8500+ fields. With our rules syntax we created 30,000 rules which trigger depending on the PGN or source address of each message. A subset of rules detect transmissions of fixed-rate messages (e.g., every 100 ms) from rogue devices, and known methodologies for removing ECUs from the CAN Bus. Meaning an attacker cannot transmit fixed-rate messages where a legitimate ECU is already transmitting undetected. For non-fixed-rate messages we can detect them being transmitted with the appropriate state conditions. And so we effectively apply the same timing based fixed-rate-message security guarantees to non-fixed-rate messages with fixed-rate. We have created a small number of rules, with more system expertise and documentation we can continue to limit the attack space, until an attacker is unable to transmit any messages undetected in our system model.

We proved the efficacy of the rules framework through extensive attack testing against an ECM with a physical implant, and verified 0 false positives using a 90 minute truck data capture. Our shift from a trusted network to an assumed breach model allows for greater overlap with maintenance issues, and resilience to new offensive technologies. As new attacks are developed, new rules with greater system specific knowledge, such as those laid out for diagnostic attacks, can detect them without false positives. This gives defenders a chance to differentiate maintenance faults and accidents from the intentional, and mitigate risk of further fleet propagation. By explicitly defining each rule a defender can know their attack coverage, and have clear alerts with our knowledge of the data field. With our research we have severely restricted the physical threat model, provided an adversarial model with the insights of J1939 data, and proposed how a defender can use this same data to their advantage, while still maintaining system awareness and clarity on what attacks they can defend against.

References

[1] Jennifer Cheeseman Day and Andrew W. Hait. Number of truckers at all-time high. *US Census Bureau*, July 2019.

[2] Society of Automotive Engineers standard. https://www.sae.org/publications/collections/content/j1939_dl/ Accessed 11 November 2019.

[3] Robert Bosch. "CAN specification version 2.0". http://esd.cs.ucr.edu/webres/can20.pdfAccessed 11 November 2019, 1991.

[4] Roderick Currie. "hacking the can bus: Basic manipulation of a modern automobile through can bus reverse engineering". *SANS Institute*, 2017.

[5] Craig Smith. *The Car Hacker's Handbook: A Guide for the Penetration Tester*. No Starch Press, San Francisco, CA, USA, 1st edition, 2016.

[6] C. Miller and C Valasek. Remote exploitation of an unaltered passenger vehicle. defcon 23 (2015).

[7] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association.

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

[9] Chris Hartnoll. Rolls-Royce: Optimising jet engine maintenance with machine learning. *Harvard Business School*, 2018.

[17] F. Baronti, G. Manni, E. Marraccini, G. PalamÃ , R. Roncella, and R. Saletti. Monitoring system for the integrated acquisition of on-board sensor data in cruising motoryachts. In *2010 IEEE International Conference on Industrial Technology*, pages 252–258, March 2010.

[10] Subhojeet Mukherjee, Hossein Shirazi, Indrakshi Ray, Jeremy Daily, and Rose Gamble. Practical DoS Attacks on Embedded Networks in Commercial Vehicles. In *International Conference on Information Systems Security*, volume 10063, pages 23–42, 2016.

[11] Yelizaveta Burakova, Bill Hass, Leif Millar, and André Weimerskirch. Truck hacking: An experimental analysis of the SAE j1939 standard. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX, 2016. USENIX Association.

[12] Kyong-Tak Cho and Kang G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 911–927, Berkeley, CA, USA, 2016. USENIX Association.

[13] Orly Stan, Yuval Elovici, Asaf Shabtai, Gaby Shugol, Raz Tikochinski, and Shachar Kur. Protecting military avionics platforms from attacks on mil-std-1553 communication bus. arXiv, 07 2017.

[14] Krzysztof Pawelec, Robert A Bridges, and Frank L Combs. Towards a CAN IDS based on a neural-network data field. arXiv, 2017.

[15] Matthew Butler. An Intrusion Detection System for Heavy Duty Vehicle Networks. In *ICCWS*, volume 12, pages 399–405, 2017.

[16] P. Murvay and B. Groza. Security shortcomings and countermeasures for the SAE j1939 commercial vehicle bus protocol. *IEEE Transactions on Vehicular Technology*, 67(5):4325–4339, May 2018.

[18] D. S. Paraforos, V. Vassiliadis, D. Kortenbruck, K. Stamkopoulos, V. Ziogas, A. A. Sapounas, and H. W. Griepentrog. Automating the process of importing data into an fmis using information from tractor's can-bus communication. *Advances in Animal Biosciences*, 8(2):650–655, 2017.

[19] Ken Munro. Wi-fi as a weapon: how customer communication networks can be used to take over trains. In *InfraRail*, 2018.

[20] Adam Tanner. Data monitoring saves some people money on car insurance, but some will pay more.

[21] Coppher Hill Technologies. Pican2 - bus interface for raspberry pi.

[22] Wilfried Voss. Guide to sae j1939 - controller area network and j1939. https://copperhilltech.com/blog/guide-to-sae-j1939-controller-area-network-and-j1939/ Accessed 11 November 2019, 2018.

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

[24] Elastic. "https://www.elastic.co/products/elasticsearch"Accessed 11 November 2019.