A Precedence Graph-Based Approach to Detect Message Injection Attacks in J1939 Based Networks

Subhojeet Mukherjee*, Jacob Walker† and Indrakshi Ray‡
Department of Computer Science, Colorado State University
Fort Collins, CO, 80523, USA
Email: *Subhojeet.Mukherjee@colostate.edu, †Jacob.Walker@colostate.edu, ‡Indrakshi.Ray@colostate.edu
Jeremy Daily
Department of Mechanical Engineering, University of Tulsa
Tulsa, OK 74104, USA
Email: Jeremy-Daily@utulsa.edu

***Abstract*—Vehicles now include Electronic Control Units (ECUs) that communicate with each other via broadcast networks. Cyber-security professionals have shown that such embedded communication networks can be compromised. Very recently, it has been shown that embedded devices connected to commercial vehicular networks can be manipulated to perform unintended actions by injecting spoofed messages. Such attacks can be hard to detect as they can mimic safety critical actions performed by ECUs. We present a precedence graph-based anomaly detection technique to detect malicious message injections. Our approach can detect malicious message injections and is able to distinguish them from safety critical actions like hard braking.**

***Index Terms*—J1939, Application Layer, Message Injection, Precedence Graph, Time Series, Anomaly Detection**

I. Introduction and Related Efforts

A. CAN and SAE J1939

Modern vehicles have a number of embedded electronic devices that control the vehicle operation and are referred to as Electronic Control Units (ECUs). ECUs communicate with each other over the Controlled Area Network (CAN) bus. The CAN protocol [1] facilitates highly reliable communication over a multi-master broadcast bus by using a priority based arbitration scheme. However, it does not indicate how messages are used by ECUs. Such decisions are made using specifications and protocols designed to run over the CAN bus. While passenger vehicles mostly adopt manufacturer defined standards and specifications [2], commercial vehicles such as trucks and buses use a common standard specified in the SAE J1939 [3] documents. These common set of standards aid in reliable and interoperable communication between ECUs manufactured by different OEMs.

B. Command Injection Attacks and Detection Challenges

Researchers [4], [5] have shown that passenger vehicles can allow both physical and remote access to the CAN bus. Thus, an attacker may inject malicious messages into the CAN bus [6], [7], [8] and disrupt regular vehicle functions. Burakova et al. [2] showed that by injecting J1939 standardized control command messages (*command injection attacks*), driving critical ECUs such as the Engine ECU can be tricked into performing activities potentially causing damage to heavy vehicles. In order to formulate a successful attack, the attacker needs to have adequate information about the underlying network. Although the openly available J1939 standards makes this task easier for the attacker, OEM specific engineering aspects like whether a particular message is supported by an ECU or how an ECU reacts to a specific command, may be hidden from an attacker [10]. While developing the attacks, Burakova et al. [2] had to modify some of their initial strategies based on the observed ECU reactions. Based on this observation, we posit that before launching an actual attack, an attacker may probe or fuzz test the underlying network and the ECUs attached to it. In this paper, we attempt to detect such command injection attacks and associated probing/fuzzing activities.

Since J1939 standards are followed ubiquitously, a command message on the network can be easily identified at runtime. However, since commands are J1939 specific messages it may be hard to distinguish malicious injections from legitimate ones. Even if the attacker injects a bursty sequence of command messages to establish continuous control over an ECU, it may still be hard to distinguish, as similar bursty sequences may be observed when safety critical units like anti-lock braking and traction control systems attempt to stabilize the vehicle during hard braking or drive slips. Although the latter case denotes some form of driving anomaly, it is not necessarily malicious and in this work we attempt to distinguish between malicious command injections and safety critical legitimate ones.

A number of challenges are associated with this problem. First, we need to eliminate false positives (as mentioned in the previous paragraph). Second, malicious message injection detection needs to be done in (almost) real-time [11]. Third, the solution needs to be readily deployable across different models of commercial vehicles that are driven in various conditions. Traditional supervised anomaly detection systems rely on previously recorded information that represent the normal state of operations and compare runtime operations with the recorded traffic. Such solutions are infeasible as vehicles differ with respect to their normal characteristics as the drivers have different styles and the road conditions also vary.

C. Related Efforts

To the best of our knowledge, there is only one proposal [12] that has been made to detect intrusions on J1939-based networks. This paper proposes to use a machine learning based approach to solve the aforementioned problem. Machine learning based intrusion detection approaches [13], [14] have been adopted previously in passenger vehicles. However, such approaches require offline training data to characterize normal vehicular operations. In-vehicular communication traffic depends largely on the internal mechanics, the driver's behavior and environmental factors. Therefore, supervised anomaly detectors trained on sample dataset obtained from one driving condition may not be effective to detect attacks executed under different driving conditions. A second category of approaches look at timing or message frequency based anomalies [15], [16], [17]. Legitimate command injections can be bursty and situational and hence may be difficult to characterize normality using message transmission rates. Muter et al. [18] propose an entropy-based approach and flag peaking system entropy values as intrusions. However, they do not consider the operational situation of a vehicle [19]. For example, under abnormal driving conditions like hard brakes and drive slips, the entropy of the communication system can observe peaking values thereby increasing the chance of false alarms. Signature-based approaches [20], [21] have also been proposed, but creating and updating attack signatures in the domain of vehicular security can be difficult [11].

In this paper we make the following contributions:

1) We first propose an approach which models sequentially arriving discrete J1939 parameters using a modified representation of precedence graphs [22] which we refer to as *Report Precedence Graphs* (RPGs).
2) We then introduce two features that can be used to characterize normal driving behavior and legitimate/malicious command injections from periodically generated RPGs. The two features, *Normalized Graph Flux Capacity* (NGFC) and *Edge Weight Distribution Skewness* (EWS), are based on the structural nature of the graph and do not rely on individual message contents. Thus, J1939 specific domain knowledge is only used to generate RPGs and not for detecting injections. We thus posit, that our approach may be used equally efficiently on passenger cars if such RPGs can be built from manufacturer specifications.
3) Finally, we present a real-time detection algorithm which makes one-step-ahead [23] time series predictions, and compares the obtained NGFC and EWS values with their predicted counterparts to detect injection attempts. Using a moving window of 30 seconds, our approach detects malicious injections and distinguishes them from legitimate commands. Furthermore, since we do not make use of any offline training data or previously crafted attack signatures, our approach can be used readily on any vehicle, driven by any driver and under any environmental conditions.

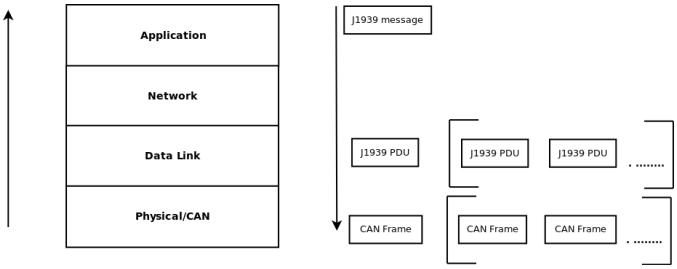


Fig. 1: SAE J1939 Protocol Stack

Identifier Field						Data Field
P	EDP	DP	PF	PS	SA	64 bits
3 bits	1 bit	1 bit	8 bits	8 bits	8 bits	

Fig. 2: J1939 Message [25]

The rest of the paper is organized as follows. Section II highlights some preliminaries of the J1939 protocol stack and introduces RPGs. Section III characterizes message injection attacks using the newly introduced features, namely, NGFC and EWS. Section IV demonstrates our approach for detecting intrusions in real-time. Section V showcases the ability of our approach to detect command injection attacks, even in the presence of safety critical activities like hard braking. Section VI highlights the contributions and suggests possible future directions.

II. Background and Preliminaries

A. SAE J1939 Application and Data-Link Layer Specifications

The SAE J1939 specifications are organized based on the OSI seven layer networking model. Of the seven layers, the most critical are the *Application Layer* [24] and the *Data-link Layer* [25]. The Data-link layer specifications enable reliable data-transfer between ECUs and the Application layer specifies how the recipient ECUs interpret the messages and realize various vehicular parameters embedded in them. The J1939 protocol stack[1] is shown in Fig. 1. Messages created at the Application layer must be transmitted through the Physical/CAN layer. J1939 standards allow message sizes to be greater than maximum limitation imposed by the underlying CAN standards. Consequently, J1939 messages are broken down into smaller units and then transmitted on the physical bus as sequence of bits constituting CAN frames. Here we focus only on J1939 messages that fit into one CAN frame and refer to it as a *message*.

ECUs, listening on a CAN network, read CAN frames from the bus and process them following upper layer standards, like SAE J1939. After removing CAN specific information bits from the message, the remaining bits are interpreted using the J1939 Protocol Data Unit (PDU) format shown in Fig. 2. A

[1]SAE specification documents are available for 4 of the 7 OSI layers at http://www.sae.org/standardsdev/groundvehicle/j1939a.htm

J1939 message is split into an *Identifier* field and a *Data* field. The Identifier field consists of 6 different components namely, *Priority (P)*, *Extended Data Page (EDP)*, *Data Page (DP)*, *PDU Format (PF)*, *PDU Specific (PS)* and *Source Address (SA)*. P and SA denote the criticality and originator ECU of the message respectively. EDP and DP are predefined 1-bit binary values specified in the Data-link layer documentation [25]. The information contained in the EDP, DP, PF and PS fields can be combined to obtain two critical J1939 concepts namely, *Parameter Group Number (PGN)* and the *Destination Address (DA)* for a J1939 message. PGNs are unique numbers, assigned to each message, that can be used to determine the meaning of a J1939 message. For example, messages related to torque or speed control correspond to PGN 0 (0000_{16}). If the value in the PF field is less than 240, the PGN is calculated only on the basis of the EDP, DP and PF fields, while the PS field expresses the destination address. Otherwise, the PGN is calculated using bits from the EDP, DP, PF and PS fields, and the destination address is interpreted as 255. Thus, J1939 standards allow broadcast messages as well as destination specific messages. J1939 messages corresponding to a particular PGN can be transmitted periodically or in an ad-hoc manner [9].

Values of different vehicular parameters are encoded in the data field of the J1939 message. In order to obtain these values, the PGN for a message is first calculated using the logic expressed above. The J1939 Digital Annex [26] maps every PGN to a set of *Suspect Parameter Numbers* or SPNs. SPN determines how J1939 data is utilized by application software and firmware. Each SPN is attributed to a *starting bit position* in a message and a *length* which can be used to extract the exact set of message bits that correspond to that SPN. These bits are then converted to application understandable forms using *resolution* and *offset* factors, also assigned to the same SPN. To clearly comprehend the above mentioned procedure let us consider the example J1939 message (split into *ID* and *Data* fields) shown below.

$$ID \rightarrow \underbrace{110}_{P}\overbrace{\underbrace{0}_{EDP}\ \underbrace{0}_{DP}\ \underbrace{11111101}_{PF}\ \underbrace{11100001}_{PS}}^{PGN}\underbrace{00110001}_{SA}$$

$$Data \rightarrow \underbrace{01100100}_{SPN2609}111111\ \underbrace{01}_{SPN7853}\ 11111111111111..$$

Since the value contained in the PF field is greater than 240, the PGN is calculated using all of EDP, DP, PF and PS fields. In this case the binary to decimal conversion yields PGN 64993, which denotes "Cab A/C Climate System Information" [26]. From the J1939 Digital Annex, this PGN is associated with 2 distinct SPNs:

- **2609** "Cab A/C Refrigerant Compressor Outlet Pressure", expressed in byte 1 (from left) and calculated by multiplying the decimal equivalent with the resolution "16kPa/bit" and adding to it an offset of "0".
- **7853** "Air Conditioner Compressor Status", expressed in byte 2 (from left) bits 0 to 1 and used in binary form.

Report	Interpretation	SA	PGN
0	Accelerator pedal 1 in low idle condition	00_{16}	61443
1	Brake pedal released	00_{16}	65265
2	Proprietary retarded control mode	$0F_{16}$	61440
3	Low idle governor retarded control mode	$0F_{16}$	61440
4	Accelerator pedal 1 not in low idle condition	00_{16}	61443
5	Brake pedal depressed	00_{16}	65265

TABLE I: Report Interpretations from RPG in Fig. 3

The actual value of the "Cab A/C Refrigerant Compressor Outlet Pressure" can thus be calculated as $01100100_2 * 16_{10} + 0_{10} \rightarrow 100_{10} * 16_{10} \rightarrow 1600\ kPa$. Similarly, the "Air Conditioner Compressor Status" is set to 01_2 which denotes that the "Air Conditioner Compressor is ON" [26]. Note that the message bits which are not used for parameter instantiations are set to 1.

According to the J1939 specifications, some SPNs are used to denote states of various vehicular functional parameters. For example, the 2 bit SPN 597 can take 4 values, two of which denote states like brake pressed (00_2) and released (01_2). We refer to these SPNs as *state SPN*s ($sSPN$).

B. Report Precendence Graphs (RPG)

RPGs are used to express the temporal relationships between various vehicular parameters. We begin by defining the concept of a report.

Definition 1 (Report)**.** *Reports are the basic units of state information received on the CAN bus. A report can be expressed as a set of sSPN-value-source triples contained in a single J1939 message.*
$r = \{(sSPN, v, sa) |\ (\exists PGN,$
sSPNs are assigned to *PGN* and transmitted by source *sa*$) \wedge$
$(v \in \{0,1\}^{\text{length of sSPN}})\}$

A report is a collection of sSPN-value pairs from a single message transmitted by a specific source. Since messages on the J1939 network are distinguished using PGNs, every report is associated with a single PGN. For example, $\{(597,00_2,00_2),(598,01_2,00_2)\}$ is a report generated from a message associated with the PGN 65265 and transmitted by source 00_{16} corresponding to the Engine ECU. This denotes the state information (`Brake pedal released`, `Clutch pedal depressed`). Since the same PGN can be transmitted by different ECUs, multiple reports can be associated with the same PGN, albeit being distinguished by the source address of that ECU. Note that, some SPN and sSPN values may denote parameters not being supported by the vehicle. These are denoted by sets of '1's in the binary representation of the corresponding sSPN values and we ignore them when generating reports.

Definition 2 (Report Precedence Graph)**.** *The Report Precedence Graph (RPG) is a labeled directed graph $G = (R, T, L)$ where each node $r \in R$ is a report and each edge $\langle r_i, r_j \rangle \in T$ denotes report r_j transmitted after report r_i ($r_i \prec r_j$ with no other r_k such that $r_i \prec r_k$ and $r_k \prec r_j$), a total of $l_{i,j} \in L$ number of times.*

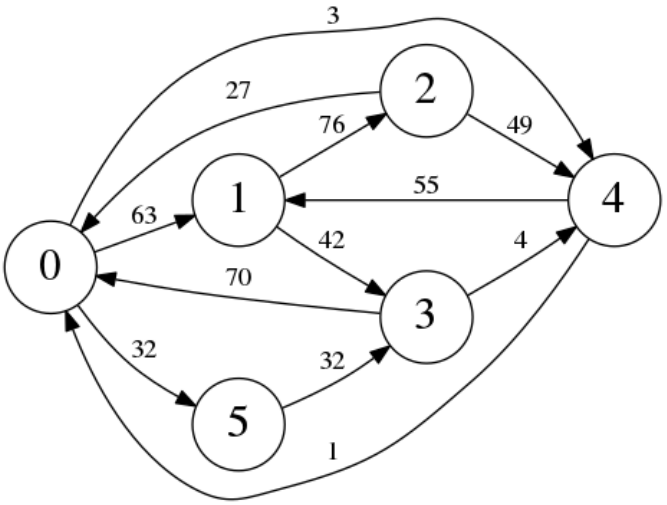


Fig. 3: Runtime Instance of a Report Precendence Graph

Fig. 3 shows a 15 second snapshot of an RPG built from 3 distinct PGNs transmitted on a heavy vehicle CAN bus by the engine (00_{16}) and the retarder ($0F_{16}$). Table I shows the interpretation [26] of the RPG in Fig. 3. The edge labels in the graph denote the number of times $report_i$ precedes $report_j$. Assuming that the only ECUs connected to a CAN bus are the engine (00_{16}) and retarder ($0F_{16}$) and only 3 distinct PGNs are transmitted, a third party tool sniffing the bus can construct the RPG by following the steps shown below.

1) Observe the PGNs in the message, such as, 61443, 65265 or 61440.
2) Interpret the vehicle parameters embedded in the message by evaluating SPN values.
3) Express a report as a collection of sSPN-value pairs from a given source. As an example, Report 0 from the engine is constructed by observing just one SPN-value pair $\{(558, 01_2)\}$ from PGN 61443.
4) When a new message is seen on the bus, repeat steps 1 to 3 and create an edge from the previously seen report to the newly formed report to denote temporal precedence. Note that the RPG does not represent transitive precedences (definition 2).

This procedure appears in lines 1-10 of Algorithm 1 in Section IV.

Two observations can be made from the RPG in Fig. 3:

- Firstly, reports generated from PGN 65265 (1 and 5) are always preceded by reports generated from PGN 61443 (0 and 4). Similarly, reports generated from PGN 61440 (2 and 3) are always preceded by reports generated from PGN 65265 (1 and 5). This implies that in the period of 15 seconds during which this RPG was built PGNs 61443, 65265 and 61440 were seen on the bus in the following order: 61443 $\prec$ 65265 $\prec$ 61440. We believe this order depends on engineering and/or specification related factors like ECU firmware programming logic, PGN transmission rates etc.
- Secondly, all reports generated from one PGN are not equally likely to precede all reports generated from another PGN, even if the first PGN is seen before the second in the bus traffic. For example, although all of reports 2, 3, 4 and 5 occur in the given time period and PGN 65265 is seen before PGN 61440, report 2 is never preceded by report 5. In other words, driving states seen in that period of time did not require the retarder to be under proprietary control when brakes were being pressed. This reflects temporal behavioral aspects of the driver and the vehicle.

The RPG thus reflects both behavioral and engineering/specification related aspects of the vehicle.

III. Characterizing Network Intrusions Using RPGs

We now highlight features of the RPG that can be used to characterize normal driving scenarios and command injections.

A. Normal Driving

The driving pattern of a truck depends on how the driver responds to external stimuli. Predicting such subtle external changes a priori is hard and cannot be easily inferred from the in-vehicle bus traffic. For instance, just by observing the bus traffic, it is difficult to predict the external road conditions and the driver's response. However, once such a change in external stimuli happens and the driver reacts, regular precedence patterns can be observed in the RPG. Fig. 3 shows one such scenario where proprietary control mode of the retarder is activated only when the brake pedal is released. Note that, such patterns reflect the engineering and behavioral aspect of the driver/vehicle under certain driving states.

At any given point there may exist some less frequent report precedences. For example, in Fig. 3 we observe that reports 4, 0 and 3 precede each other less frequently. Such unexpected precedences that are rare do not hinder our detection approach presented in the next section.

B. Command Injections

Fig. 4 shows the RPG after malicious spoofed messages were injected in the same time frame of generation as the original RPG shown in Fig. 3. The newly introduced attack report is a `commanding RPM to the engine` and is labeled 6 in Fig. 4. Burakova et al. [2] theorized that prolonged injection of this command, with high RPM values, can damage the engine. As mentioned in section I-C, this can also be a fuzzing or probing attempt to observe the response of the engine ECU to such a command. To mimic both the above mentioned scenarios, the RPG in Fig. 4 was generated by repeating the injected messages at a 50% probability for a period of 15 seconds. Similar to [15], we assume that the injected messages exist on the bus along with normal messages.

There are two observations to be made from Fig. 4.

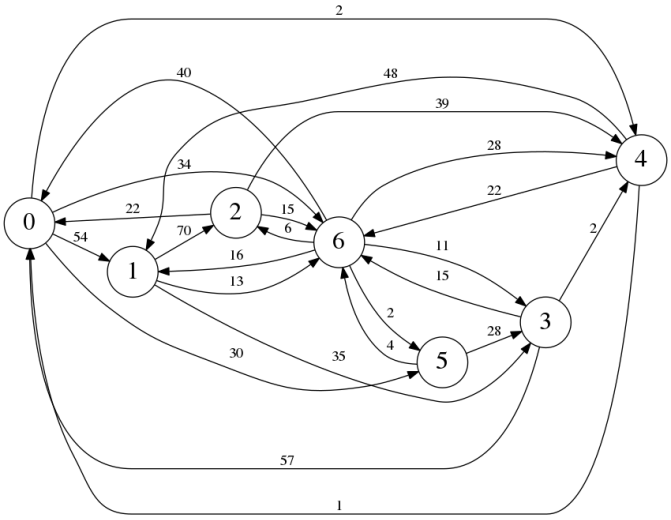
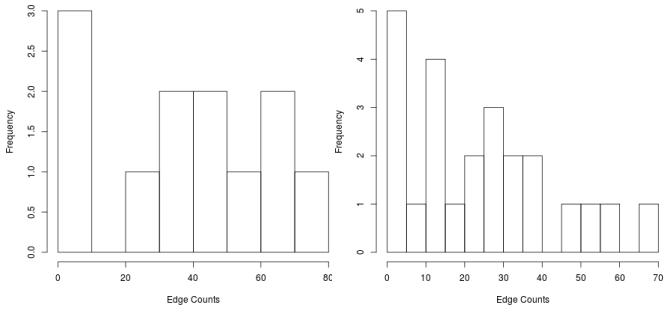


Fig. 4: Runtime Instance of a Report Precedence Graph Under Message Injection Attack



(a) RPG Fig. 3 - Normal Driving Conditions

(b) RPG Fig. 4 - Message Injection Attacks

Fig. 5: RPG Edge Weight Distributions

a) Average Flux Capacity of an RPG: At the time of normal driving the precedence patterns seen in the RPG reflect the normal engineering and behavioral aspects of the vehicle. Thus, if the RPG lacks some of these patterns significantly in the bus traffic, it indicates some form of anomalous activity. If the attacker has no prior knowledge of how the vehicle is engineered or how a driver behaves, the precedence patterns will be random and there will be no strong coherence with report precedences generated from normal traffic.

To quantify occurrences of random precedence we make use of the concept of *flux capacity*. Flux capacity was introduced by Martinez et al. [27] to quantify the amount of information flow that passes through gene regulators. It is defined as the product of the in-degree and out-degree of a node. In this work, we use it to denote the number of two-edge paths passing through a report r in an RPG.

$$\begin{aligned} fc_{r_i} &= |(r_j, r_i)| * |(r_i, r_k)| \ \forall (r_j, r_i), (r_i, r_k) \in T \\ &= d^{in}(r_i) * d^{out}(r_i) \end{aligned}$$

For example, the flux capacity of report 6 in the Fig. 4 is calculated as 6*6 = 36. This is also the number of two edge paths passing through report 6.

Using the concept of flux capacity we now define the concept of *Normalized Graph Flux Capacity* (NGFC) as:

Definition 3 (Normalized Graph Flux Capacity)**.** *The Normalized Graph Flux Capacity of an $RPG = (R, T, L)$ denotes the amount of randomness observed in report precedences.*

$$NGFC = \frac{\sum_{i=1}^{|R|} fc_{r_i}}{|R|^3}$$

NGFC is thus the cumulative flux capacity of all reports in the RPG, normalized to the cubic power of the number of reports R. The choice of the cubic power in the denominator is motivated by the extreme case, where, if an RPG is a complete graph having n nodes, will be of the order of n^3 ($\sum_{i=1}^{n}(n-1)*(n-1)$). Thus, as random precedences begin to be seen in the RPG and every report becomes equally likely to be preceded by every other report, the NGFC value starts approaching 1 ($\frac{n^3}{n^3}$). This phenomenon can be observed in Fig. 4 where the NGFC is calculated to be 0.27, almost double of what is calculated from Fig. 3 i.e 0.12.

b) Skewness in Edge Weight Distribution of an RPG: Another important feature that can be observed from the RPG in Fig. 3 and Fig. 4 is the skewness of distribution of the edge weights (*EWS*). As observed from Fig. 5, the distribution of the edge weights under an attack becomes increasingly more right-skewed. In other words, the level of positive skewness increases in the Fig. 5b as compared to Fig. 5a. Python-Scipy's skewness [28] measure[2] for the distribution in Fig. 5a is obtained as -0.113 and that for the distribution in Fig. 5b is obtained as 0.745.

The increase in positive skew under an attack can be attributed to the same factor as that observed in the previous case. Due to the random precedences observed after the introduction of report 6 in Fig. 4, the injected report interleaves with normal reports and the original edge weights in Fig. 3 reduce, making room for new, less-weighted edges. For example, the weight for edge $\langle r_0, r_1 \rangle$ reduces from 63 to 54 and newer lower weight edges such as $\langle r_6, r_5 \rangle$ and $\langle r_3, r_6 \rangle$ are introduced. In other words, the incapability of the attacker to predict the normal report precedences leads to the introduction of random, less-weighted edges, thereby increasing the overall skewness.

IV. Detecting Intrusions in Real Time

RPGs are useful in observing temporal relationships between various vehicular parameters which describe the behavior of the driver or functionalities performed by the vehicle. Moreover, RPGs can be used to detect command injections in embedded vehicular networks using NGFC and EWS. We now describe how these two features are used to detect command injection attacks.

[2]Measured as $g = \frac{m_3}{m_2^{3/2}}$, where m_3 and m_2 are the third and second moments about the mean.

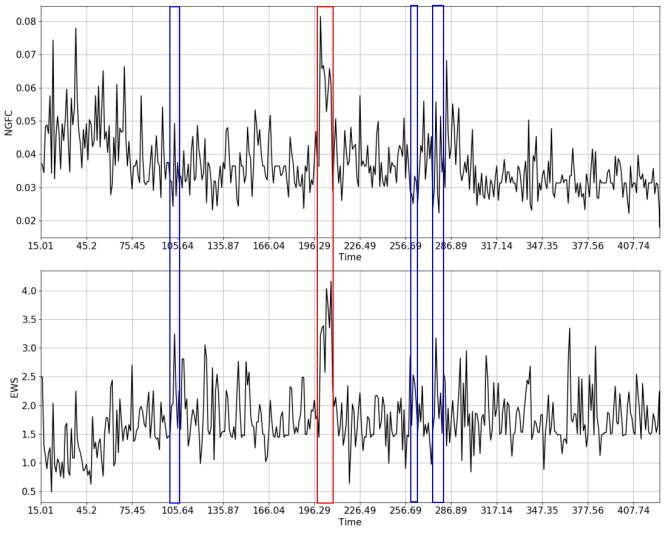


Fig. 6: Sampled NGFC and EWS Time Series

A. Representing Runtime Vehicular Traffic as NGFC and EWS Time Series

1) Sampling Window: NGFC and EWS are calculated on pre-built RPGs. In order to build an RPG one needs to observe a set of J1939 messages on the CAN bus for some period of time. We refer to this period as *sampling period* during which NGFC and EWS values are sampled. Calculating graph-based and statistical features can be time-consuming. ECUs often operate under strong resource constraints [29] and thus processing large graphs in real time might lead to undesired performance bottlenecks. It is therefore desirable to calculate NGFC and EWS during a sampling period, while a new RPG is built in parallel. Then again, the sampling period should not be too large, as in that case, one might under-sample the feature values, therefore hindering the detection process. Keeping these factors in mind, we decided to choose a sampling period of 1 sec. We also noticed that most critical J1939 PGNs (having priority ≤ 3) are transmitted at rates of 1 sec or lower. A sampling period of 1 sec would thus allow us to see most critical J1939 messages on the bus.

2) NGFC and EWS Time Series: NGFC and EWS values measured in the sampling period of 1 second are shown in Fig. 6 as time series. These series were generated from one of the recorded traffic datasets we used later (section V) for evaluating our detection algorithm. While building the RPG from which these NGFC and EWS values were calculated we considered every state SPN that was transmitted on the bus. We now demonstrate specific behavior of the time series (with reference to Fig. 6) which can be used to individually classify normal driving, legitimate and malicious message injections.

- **Normal Driving** During the periods of normal driving both time series fluctuate around some base value, with mostly small positive peaks. Normal driving is highly influenced by environmental stimuli. Moreover, certain factors like increased bus contention and fluctuating driver behavior can alter NGFC and EWS values. Under the assumption that such factors are independent and identically distributed, the sampled values tend to form a stationary time series. However, the time series may see short trends occasionally, denoting periods of increasing or decreasing stability. This can be seen in Fig. 6 where the NGFC and EWS values observe increasing and decreasing trends towards the beginning. This is because towards the beginning the vehicle sees new reports and new low weighted edges being introduced. However, such new introduction soon begins to dissipate leading to an almost stationary time series. For the sake of simplicity and under a lack of evidence from Fig. 6, we assume that NGFC and EWS time series lack any form of seasonal trends.
- **Legitimate Command Injections** As highlighted before, legitimate command injections are performed in safety critical scenarios. However, such activity is performed by specific ECUs that follow some pre-planned or programmed behavior. This means that NGFC and EWS values obtained during those period do not exhibit significant abruptness. This behavior can be seen in Fig. 6 and is highlighted in blue boxes.
- **Malicious Command Injections** The portion of the NGFC and EWS time series that show simultaneous significant abrupt peaks, denote some form of unnatural behavior leading to random low count precedences. This is the portion where malicious commands were injected into the recorded bus traffic. Although, the attack traffic used to generate Fig. 6 is a simulation, we believe in a real driving scenario even sharper abrupt peaks can be seen if the driver or the vehicle start taking rapid cautionary actions on sensing the attack, thereby generating previously unseen and possibly random precedences.

B. Detection Methodology

To detect message injection attacks in real-time we used a modified form of the methodology used by Brutlag et al. in [23]. Brutlag et al. suggest making one-step-ahead predictions using Holt-Winter's exponential smoothing. However, the NFGC and EWS time series do not exhibit any predictable seasonal trends, but may exhibit sudden short trends. Under this assumption, we make use of the simple exponential smoothing and Holt's exponential smoothing [30] technique to make future predictions, depending on whether the small set of predictor time series values exhibit stationarity. To verify the assumption of stationarity, we use Augmented Dickey-Fuller [30] unit-root test.

Algorithm 1 shows the real-time detection methodology used to flag command injection attempts. We begin by generating the RPG in real time. Lines 1 - 10 show this process. As newer messages are seen on the CAN bus, we generate reports out of incoming messages if they contain state SPNs. A new node is added to the RPG if it is not already contained in the set of existing reports R. A new precedence edge $\langle r_i, r_j \rangle$ is added if it is not a self-loop. Finally, the value for $l_{i,j}$ is

Algorithm 1 Real-Time Command Injection Attack Detection

Require: J1939 Traffic, confidence
$RPG = (R, T, L),\ R, T, L = \emptyset$
$r_{current} = null$
startTime = 0.0 sec
samplingWindow = 1 sec
dWindowCounter = 0
fcBuf, skBuf= $\emptyset$ ▷ queue data structure
1: **while** there are messages in traffic **do**
2: $r_i \leftarrow$ Report from J1939 Message
3: **if** $r_i \notin R$ **then**
4: $R \leftarrow R \cup r_i$
5: **end if**
6: **if** $r_i \neq r_{current} \wedge r_{current} \neq null$ **then**
7: $T \leftarrow T \cup \langle r_{current}, r_i \rangle$
8: Update L, set $l_{i,j} \leftarrow l_{i,j} + 1$
9: **end if**
10: $r_{current} = r_i$
11: **if** time $\geq$ startTime + samplingWindow **then**
12: dWindowCounter = dWindowCounter + 1
13: **if** dWindowCounter = 31 **then**
14: is_attack = SIG-INC(fcBuf,NFGC(R,T)) $\wedge$ SIG-INC(skBuf,EWS(L))
15: **if** is_attack **then**
16: **RAISE ALARM**
17: **else**
18: fcBuf.push(NFGC(R,T)), skBuf.push(EWS(L))
19: fcBuf.pop(NFGC(R,T)), skBuf.pop(EWS(L))
20: **end if**
21: dWindowCounter = dWindowCounter - 1
22: **else**
23: fcBuf.push(NFGC(R,T)), skBuf.push(EWS(L))
24: **end if**
25: startTime = time
26: $RPG = (R, T, L),\ R, T, L = \emptyset$
27: $r_{current} = null$
28: **end if**
29: **end while**
30: **procedure** SIG-INC(buf,current)
31: has_trend = **FALSE**
32: **if** Augmented_DickeyFuller_test(buf).p-value $\leq$ .1 **then**
33: has_trend = **TRUE**
34: **end if**
35: forecast_band = Holt-Winter's(buf,confidence,has_trend)
36: **if** current $\in$ forecast_band **then**
37: **return TRUE**
38: **else**
39: **return FALSE**
40: **end if**
41: **end procedure**

updated in the list of edge weights thereby maintaining up to date edge weights. If the *samplingWindow* expires (in line 11), we generate a new RPG (lines 25-27) for the next sample window. This process can be executed in parallel to the actual detection process (lines 11-41), thus speeding up calculations.

To predict the next future value, we use a set of 30 sampled predictor values in a moving window. Consequently, an initial set of 30 predictor values is buffered (line 23) first. From then on, for every newly sampled NFGC or EWS value, we compare the obtained value with the confidence band returned by the exponential smoothing and forecast technique. For this, we use R's native implementation of simple and Holt's exponential smoothing[3] and forecast[4] with and without the trend factor. If both obtained NFGC and EWS values are outside the forecast confidence band (lines 30 - 40), we raise an alarm and flag that sampled window as an *attack window*. The forecasting process returns wider confidence bands for bigger confidence limits, in order to enhance the prediction accuracy. Consequently, the detection accuracy of our approach depends heavily on the input confidence limit. If an attack is detected, we do not consider the sampled NFGC and EWS values obtained in that window for future predictions. This allows us to avoid treating abnormal values as predictors. Otherwise, we buffer the newly sampled values (line 18) and drop the oldest value from the bottom of the buffers *fcBuf* and *skBuf*.

V. Analysis and Discussion

We now discuss the accuracy of our approach in detecting command injection attacks.

A. Evaluation Metrics

In order to evaluate the capability of our detection approach to distinguish between normal traffic, legitimate and malicious command injections we choose 2 separate metrics:

- **Detection Percentage**: The detection percentage used is similar to the one used in [14]. Here we consider three separate detection percentages namely, *Normal Traffic* detection percentage (NDI), *Legitimate Command Injection* detection percentage (LDI), *Malicious Command Injection* detection percentage (MDI). The percentages are calculated as follows.
$percentage = \frac{number\ of\ flagged\ windows * 100}{total\ number\ of\ windows\ in\ traffic}$ Since our approach splits the total observed traffic into sampling windows of 1 second, we classify a window as containing *Normal Traffic*, *Legitimate Command Injection* or *Malicious Command Injection* if that window contains at least 0.25 seconds of traffic of the particular type. For example, a window must contain at least 0.25 second of attack traffic to be labeled as *Malicious Command Injection*. All windows, apart from the command injection windows are considered to be normal.
- **Precision**: Precision allows us to evaluate the strength of our approach in detecting malicious message injections in presence of both normal traffic and legitimate message injections. We, therefore, define precision as follows.
$precision = \frac{MDI * 100}{NDI + LDI + MDI}$

B. Experiment Dataset and Attack Simulation Parameters

a) Recorded Drive Around Traffic: We tested our approach on previously recorded CAN traffic datasets: the *Kenworth* dataset and the *Peterbilt* dataset.

- **The Kenworth Dataset**: The Kenworth dataset set was generated at the University of Tulsa and recorded from a PACCAR MX powered Kenworth tractor driven around

[3]https://stat.ethz.ch/R-manual/R-devel/library/stats/html/HoltWinters.html
[4]https://www.rdocumentation.org/packages/forecast/versions/7.3/topics/forecast.HoltWinters

Injection Parameter			Normal Traffic			Legitimate Command Injections			Malicious Command Injections			Precision		
Start Time	**Duration**	**Probability**	**80**	**90**	**99**	**80**	**90**	**99**	**80**	**90**	**99**	**80**	**90**	**99**
Begin	**1**	**0.1**	2.02	0.50	0.00	16.67	16.67	0.00	100.00	100.00	0.00	84.26	85.35	0.00
Middle	**1**	**0.1**	0.76	0.76	0.00	16.67	16.67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
End	**1**	**0.1**	0.76	0.76	0.00	16.67	16.67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Begin	**7**	**0.1**	0.51	0.51	0.00	16.67	16.67	0.00	100.00	14.29	14.29	85.34	45.40	100.00
Middle	**7**	**0.1**	0.77	0.77	0.00	16.67	16.67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
End	**7**	**0.1**	0.77	0.77	0.00	16.67	16.67	0.00	12.50	0.00	0.00	41.75	0.00	0.00
Begin	**15**	**0.1**	0.52	0.52	0.00	16.67	16.67	0.00	40.00	6.67	6.67	69.94	27.94	100.00
Middle	**15**	**0.1**	0.76	0.76	0.00	16.67	16.67	0.00	6.67	6.67	0.00	27.66	27.68	0.00
End	**15**	**0.1**	0.79	0.79	0.00	16.67	16.67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Begin	**1**	**0.5**	1.26	0.50	0.00	16.67	16.67	0.00	100.00	100.00	0.00	84.80	85.35	0.00
Middle	**1**	**0.5**	0.76	0.76	0.00	16.67	16.67	0.00	100.00	100.00	0.00	85.16	85.16	0.00
End	**1**	**0.5**	0.76	0.76	0.00	16.67	16.67	0.00	100.00	100.00	50.00	85.16	85.16	100.00
Begin	**7**	**0.5**	0.51	0.51	0.00	16.67	16.67	0.00	57.14	57.14	14.29	76.88	76.88	100.00
Middle	**7**	**0.5**	0.77	0.77	0.00	16.67	16.67	0.00	62.50	62.50	0.00	78.18	78.18	0.00
End	**7**	**0.5**	0.77	0.77	0.00	16.67	16.67	0.00	100.00	100.00	87.50	85.15	85.15	100.00
Begin	**15**	**0.5**	1.04	0.00	0.00	25.00	16.67	0.00	26.67	20.00	6.67	50.59	54.54	100.00
Middle	**15**	**0.5**	0.79	0.79	0.00	16.67	16.67	0.00	93.75	26.67	20.00	84.30	60.43	100.00
End	**15**	**0.5**	0.79	0.79	0.00	16.67	16.67	0.00	100.00	68.75	6.25	85.14	79.75	100.00
Begin	**1**	**0.9**	0.76	0.50	0.00	16.67	16.67	0.00	100.00	100.00	0.00	85.16	85.35	0.00
Middle	**1**	**0.9**	0.76	0.76	0.00	16.67	16.67	0.00	100.00	100.00	100.00	85.16	85.16	100.00
End	**1**	**0.9**	0.76	0.76	0.00	16.67	16.67	0.00	100.00	100.00	100.00	85.16	85.16	100.00
Begin	**7**	**0.9**	0.51	0.51	0.00	16.67	16.67	0.00	25.00	0.00	0.00	59.27	0.00	0.00
Middle	**7**	**0.9**	0.77	0.77	0.00	16.67	16.67	0.00	100.00	100.00	37.50	85.15	85.15	100.00
End	**7**	**0.9**	0.77	0.77	0.00	16.67	16.67	0.00	100.00	100.00	100.00	85.15	85.15	100.00
Begin	**15**	**0.9**	1.04	0.00	0.00	25.00	13.33	0.00	33.33	0.00	0.00	56.14	0.00	0.00
Middle	**15**	**0.9**	0.78	0.78	0.00	16.67	16.67	0.00	33.33	33.33	20.00	65.63	65.64	100.00
End	**15**	**0.9**	0.79	0.79	0.00	16.67	16.67	0.00	100.00	100.00	100.00	85.14	85.14	100.00
Total			0.82	0.65	0.00	17.29	16.55	0.00	62.63	51.70	24.56	63.57	53.84	51.85

TABLE II: Analysis results on Kenworth Dataset

a block. Messages on the CAN bus were logged for a drive cycle of around 410 seconds thus producing a total of 410 detection windows. The total number of messages observed on the high speed CAN bus were 137317, which included 49 state SPNs thereby generating a total of 45 legitimate reports. There were three hard braking events in the recorder bus traffic at around 102-103 second, 260-262 second and 275 -278 seconds, which cumulatively occupied a total of 12 *Legitimate Command Injection* windows. This dataset allowed us to detect the accuracy of approach in detecting attacks during a steady driving process.

- **The Peterbilt Dataset**: The Peterbilt dataset (http://tucrrc.utulsa.edu/J1939.html) was taken from a 2008 Peterbilt straight crane truck with a Paccar PX-8 engine. Messages on the CAN bus were logged for a drive cycle of around 74 seconds thus producing a total of 74 detection windows. The total number of messages observed on the high speed CAN bus were 16204, which included 60 state SPNs thereby generating a total of 25 legitimate reports. There was one hard braking events in the recorder bus traffic at around 55 - 59 seconds, which cumulatively occupied a total of 5 *Legitimate Command Injection* windows. This dataset allowed us to detect the accuracy of approach in detecting attacks during a short and erratic driving process.

b) Simulated Injection Attacks: Burakova et al. [2] described their attacks on paper and included publicly hosted video links in their publication. However, to the best of our knowledge, there is no publicly available recorded network traffic showcasing the attacks they demonstrated. Furthermore, it is not always possible to record traffic which includes attacks on actual vehicles. Owing to the above mentioned challenges and keeping in tune with other researchers [14], [16], we decided to simulate message injection attempts by manually injecting malicious traffic into the Kenworth and Peterbilt datasets. Although, such a simulation does not effectively reproduce the behavior of the entire vehicular system under attack, we believe that in a real attack scenario the amount of randomness in the RPG will increase further due to unplanned defensive actions taken by the driver, even though such actions may not be successful in stabilizing the vehicle.

The attack traffic was modeled on the command injection attack performed by Burakova et al. [2]. Messages with PGN 0 (intended to command high RPMs to the Engine) were manually inserted into the recorded traffic at various positions, with different probabilities and for various time periods. We did not override any existing message [15] in the recorded traffic, as such attempts have been shown to be detectable by previous message frequency-based IDS. The tampered traffic was then replayed to mimic a drive cycle under attack and our detection algorithm was run to detect malicious message injections.

In order to test the accuracy of the IDS, three different times were chosen to perform injection attacks: *begin* which was right after the initial 30 seconds of data buffering, *middle* and *end* which were towards the middle and end of the traffic. The three start times allowed us to observe the detection accuracies during various phases of the drive cycle. We chose 3 different attack injection durations and probabilities. Higher durations meant prolonged injections whereas higher probabilities meant higher number of injected messages. Thus, a short duration

Injection Parameter			Normal Traffic			Legitimate Command Injections			Malicious Command Injections			Precision		
Start Time	**Duration**	**Probability**	**80**	**90**	**99**	**80**	**90**	**99**	**80**	**90**	**99**	**80**	**90**	**99**
Begin	**1**	**0.1**	1.47	1.47	0.00	0.00	0.00	0.00	100.00	100.00	0.00	98.55	98.55	0.00
Middle	**1**	**0.1**	1.47	0.00	0.00	0.00	0.00	0.00	100.00	100.00	100.00	98.55	100.00	100.00
End	**1**	**0.1**	2.94	1.47	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Begin	**7**	**0.1**	0.00	0.00	0.00	0.00	0.00	0.00	16.67	16.67	0.00	100.00	100.00	0.00
Middle	**7**	**0.1**	0.02	0.00	0.00	0.60	0.00	0.00	100.00	1.17	100.00	99.39	100.00	100.00
End	**7**	**0.1**	3.13	1.56	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Begin	**15**	**0.1**	3.64	0.00	0.00	40.00	0.00	0.00	50.00	28.57	0.00	53.40	100.00	0.00
Middle	**15**	**0.1**	1.45	0.00	0.00	0.00	0.00	0.00	66.67	66.67	22.22	97.87	100.00	100.00
End	**15**	**0.1**	3.64	1.85	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Begin	**1**	**0.5**	1.47	0.00	0.00	0.00	0.00	0.00	100.00	100.00	100.00	98.55	100.00	100.00
Middle	**1**	**0.5**	1.47	0.00	0.00	0.00	0.00	0.00	100.00	100.00	100.00	98.55	100.00	100.00
End	**1**	**0.5**	2.94	1.47	0.00	0.00	0.00	0.00	100.00	100.00	0.00	97.14	98.55	0.00
Begin	**7**	**0.5**	0.02	0.00	0.00	0.00	0.00	0.00	100.00	100.00	14.29	99.98	100.00	100.00
Middle	**7**	**0.5**	0.02	0.00	0.00	60.00	0.00	0.00	100.00	100.00	100.00	62.49	100.00	100.00
End	**7**	**0.5**	3.13	1.56	0.00	0.00	0.00	0.00	100.00	100.00	0.00	96.97	98.46	0.00
Begin	**15**	**0.5**	0.00	0.00	0.00	0.00	0.00	0.00	107.14	107.14	78.57	100.00	100.00	100.00
Middle	**15**	**0.5**	1.69	0.00	0.00	100.00	100.00	0.00	100.00	100.00	70.00	49.58	50.00	100.00
End	**15**	**0.5**	3.64	1.82	0.00	0.00	0.00	0.00	35.71	28.57	0.00	90.76	94.02	0.00
Begin	**1**	**0.9**	1.47	1.47	0.00	0.00	0.00	0.00	100.00	0.00	0.00	98.55	0.00	0.00
Middle	**1**	**0.9**	1.47	0.00	0.00	0.00	0.00	0.00	100.00	100.00	100.00	98.55	100.00	100.00
End	**1**	**0.9**	2.94	1.47	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Begin	**7**	**0.9**	0.02	0.00	0.00	0.00	0.00	0.00	100.00	100.00	0.00	99.98	100.00	0.00
Middle	**7**	**0.9**	0.02	0.00	0.00	60.00	0.00	0.00	100.00	100.00	28.57	62.49	100.00	100.00
End	**7**	**0.9**	3.13	1.56	0.00	0.00	0.00	0.00	20.00	20.00	0.00	86.47	92.76	0.00
Begin	**15**	**0.9**	3.70	0.00	0.00	0.00	0.00	0.00	66.67	66.67	6.67	94.74	100.00	100.00
Middle	**15**	**0.9**	1.69	0.00	0.00	0.00	0.00	0.00	70.00	70.00	10.00	97.64	100.00	100.00
End	**15**	**0.9**	3.64	1.82	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Total			1.86	0.65	0.00	9.65	3.70	0.00	67.88	59.46	30.75	73.34	75.27	48.15

TABLE III: Analysis results on Peterbilt Dataset

Dependent Variables	Independent Variables			
	Normal Traffic Detection Ratio	Legitimate Command Injection Detection Ratio	Malicious Command Injection Detection Ratio	Precision
Start Time	0.08	-0.02	0.21	0.05
Duration	-0.07	0.03	-0.27	0.01
Probability	-0.05	0.01	**0.48**	**0.44**
Confidence	**-0.81**	**-0.86**	**-0.35**	-0.12

TABLE IV: Detection Factor Correlations for Kenworth Dataset

Dependent Variables	Independent Variables			
	Normal Traffic Detection Ratio	Legitimate Command Injection Detection Ratio	Malicious Command Injection Detection Ratio	Precision
Start Time	**0.37**	-0.03	**-0.35**	**-0.43**
Duration	0.06	0.2	-0.18	0.01
Probability	0	0.02	0.06	0.06
Confidence	**-0.64**	-0.22	**-0.33**	-0.22

TABLE V: Detection Factor Correlations for Peterbilt Dataset

command injection with high probability signifies bursty attack traffic, whereas a long duration low probability attack may signify a possible probing or fuzzing attempt.

C. Performance Evaluation and Discussion

Tables II and III show the detection percentages and precision values for different experiment treatments.

a) Attack Detection Accuracy: Malicious Command Injection detection percentages were fairly high for both datasets at 80% and 90% confidence intervals. The same was not observed for 99% confidence, since increasing the confidence interval increased the prediction confidence bands, thereby including some peaking NFGC and EWS values which were detected as attacks for 80% and 90% confidence respectively. Tables IV and V show negative correlations with the confidence interval showing that increasing the confidence decreases the attack detection accuracy. It can also be observed from Table II that our approach detected 24, 24, and 15 of the 27 attack attempts made on the Kenworth Dataset at different confidence intervals thus displaying very high chances for an attack to be detected at 80% and 90% confidence intervals. Similarly, from Table III we detected 22, 21, 13 of the 27 attack attempts made on the Peterbilt Dataset at different confidence intervals, thus reestablishing the efficacy of our approach. As mentioned earlier, we use only a small set of runtime predictor values (and no previous knowledge) to make one-step-ahead predictions. This makes our approach agnostic to particulars of different drive cycles. Observations from Tables IV and V show that none of the attack parameters affected the detection accuracy significantly across both datasets.

b) Normal Traffic and Legitimate Traffic Detection: For experiments performed on both the Kenworth and Peterbilt datasets, we observed significantly less detection percentages for both *Normal Traffic* and *Legitimate Message Injections*. We can thus conclude that our approach performs reasonably well in detecting attacks, while avoiding false positives even in presence of safety critical actions like hard-braking. The precision values from Tables II and III aid in strengthening this claim further. Because, decreasing confidence limits can increase both true positive and false positive rates (and vice-versa) we believe finding an optimal confidence limit is a separate challenge that should be addressed in the future. However, the initial results show that our approach works best for a 90% confidence limit.

c) Satisfying Other Technical Constraints: All bus messages must be represented in an RPG in our approach. Thus, a dedicated device for intrusion detection may speed up the

approach and also alleviate additional burden on other ECUs. Algorithm 1 shows that the detection procedure can be divided into two modules, building the RPG and detecting intrusions using the RPG, that can be executed in parallel. Building the RPG takes a fixed *samplingWindow* amount of time. The detection procedure depends on the size of the RPG (which sets a linear upperbound for NGFC and EWS calculations) and the size of input (*buf*) to the prediction algorithm. Since we fix the size of the input to 30, the detection time is only bounded by the size of the RPG formed in 1 second. Since most messages on the CAN bus are periodic and not many PGNs are supported for a given vehicle, the size of the RPG will be relatively small. For example, during the entire runtime, the Kenworth and Peterbilt datasets observed a total of 45 and 25 reports respectively and in a span of 1 second we expect this value to be even lower.

VI. Conclusion and Future Work

We proposed a precedence graph based approach for real-time intrusion detection in heavy vehicles. The efficacy of our approach was proved by testing it on real-world data infused with injected messages. In future, we aim to extend the analysis process to ascertain optimum confidence bands for the detection procedure. Our approach currently detects malicious message injections at the application and data-link layers of the J1939 protocol stack by performing message level inspections. We aim to extend this approach to detect other malicious activities including message integrity violations and attacks at lower layers of the J1939 protocol stack.

Acknowledgement

The work was supported in part by NSF under award numbers CNS 1619641 and CNS 1715458.

References

[1] R. Bosch, "CAN specification version 2.0," *Rober Bosch GmbH, Postfach*, vol. 300240, 1991.

[2] Y. Burakova and B. Hass and L. Millar and A. Weimerskirch, "Truck Hacking: An Experimental Analysis of the SAE J1939 Standard," in *Proc. of WOOT*, 2016, pp. 211–220.

[3] *Serial Control and Communications Heavy Duty Vehicle Network - Top Level Document*, SAE International Std. SAE J1939_201 308, 2013. [Online]. Available: http://standards.sae.org/j1939_201308

[4] S. Checkoway and D. McCoy and B. Kantor and D. Anderson and H. Shacham and S. Savage and K. Koscher and A. Czeskis and F. Roesner and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *Proc. of SEC*, 2011, pp. 6–6.

[5] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *black hat USA*, vol. 2014, 2014.

[6] I. Studnia and V. Nicomette and E. Alata and Y. Deswarte and M. Kaniche and Y. Laarouchi, "Survey on Security Threats and Protection Mechanisms in Embedded Automotive Networks," in *Proc. of DSN-W*, 2013, pp. 1–12.

[7] T. Hoppe and S. Kiltz and J. Dittmann, "Security Threats to Automotive CAN Networks — Practical Examples and Selected Short-Term Countermeasures," in *Proc. of SAFECOMP*, 2008, pp. 235–248.

[8] K. Koscher and A. Czeskis and F. Roesner and S. Patel and T. Kohno and S. Checkoway and D. McCoy and B. Kantor, Brian and D. Anderson, Danny and H. Shacham, Hovav and S. Savage, Stefan, "Experimental Security Analysis of a Modern Automobile," in *Proc. of S&P*, 2010, pp. 447–462.

[9] S. Mukherjee and H. Shirazi and I. Ray and J. Daily and R. Gamble, "Practical DoS Attacks on Embedded Networks in Commercial Vehicles," in *Proc. of ICISS*, 2016, pp. 23–42.

[10] T. Hoppe and S. Kiltz and J. Dittmann, "Automotive IT-Security As a Challenge: Basic Attacks from the Black Box Perspective on the Example of Privacy Threats," in *Proc. of SAFECOMP*, 2009, pp. 145–158.

[11] ——, "Applying intrusion detection to automotive it-early insights and remaining challenges," *Journal of Information Assurance and Security (JIAS)*, vol. 4, no. 6, pp. 226–235, 2009.

[12] M. Butler, "An Intrusion Detection System for Heavy-Duty Truck Networks," in *Proc. of ICCWS*, 2017, pp. 399–406.

[13] S. N. Narayanan and S. Mittal and A. Joshi, "OBD_SecureAlert: An Anomaly Detection System for Vehicles," in *Proc. of SMARTCOMP*, 2016, pp. 1–6.

[14] M. J. Kang and J. W. Kang, "A Novel Intrusion Detection Method Using Deep Neural Network for In-Vehicle Network Security," in *Proc. of VTC Spring*, 2016, pp. 1–5.

[15] S. Otsuka and T. Ishigooka and Y. Oishi and K. Sasazawa, "CAN Security: Cost-Effective Intrusion Detection for Real-Time Control Systems," in *SAE Technical Paper*, 2014.

[16] A. Taylor and N. Japkowicz and S. Leblanc, "Frequency-Based anomaly detection for the automotive CAN bus," in *Proc. of WCICSS*, 2015, pp. 45–49.

[17] K. Cho and K.G. Shin, "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection," in *Proc. of USENIX Security*, 2016, pp. 911–927.

[18] M. Müter and N. Asaj, "Entropy-Based Anomaly Detection for In-Vehicle Networks," in *Proc. of IV*, 2011, pp. 1110–1115.

[19] I. Studnia and V. Nicomette and E. Alata and Y. Deswarte and M. Kaâniche and Y. Laarouchi, "Security of embedded automotive networks: state of the art and a research proposal," in *Proc. CARSSAFE-COMP*, 2013, p. NA.

[20] X. Zhang and W. Feng and J. Wang and Z. Wang, "Defensing the Malicious Attacks of Vehicular Network in Runtime Verification Perspective," in *Proc. of ICEICT*, 2016, pp. 126–133.

[21] U. E. Larson and D. K. Nilsson and E. Jonsson, "An Approach to Specification-Based Attack Detection for In-Vehicle Networks," in *Proc. of IV*, 2008, pp. 220–225.

[22] D. Dolev and M. K. Warmuth, "Scheduling Precedence Graphs of Bounded Height," *Journal of Algorithms*, vol. 5, no. 1, pp. 48–59, 1984.

[23] J. D. Brutlag, "Aberrant Behavior Detection in Time Series for Network Monitoring," in *Proc. of LISA*, 2000, pp. 139–146.

[24] *Vehicle Application Layer*, SAE International Std. SAE J1939/71_201 506, 2015. [Online]. Available: http://standards.sae.org/j1939/71_201506

[25] *Data Link Layer*, SAE International Std. SAE J1939/21_201 504, 2015. [Online]. Available: http://standards.sae.org/j1939/21_201504

[26] *J1939 Digital Annex*, SAE International Std. SAE J1939DA_201 510, 2015. [Online]. Available: http://standards.sae.org/j1939da_201510/

[27] N. J. Martinez and M. C Ow and M. I. Barrasa and M. Hammell and R. Sequerra and L. Doucette-Stamm and F. P. Roth and V. R. Ambros and A. JM Walhout, "A C. elegans genome-scale microRNA network contains composite feedback motifs with high flux capacity," *Genes & development*, vol. 22, no. 18, pp. 2535–2549, 2008.

[28] *scipy.stats.skew*, The Scipy community, 2017, Accessed May 26, 2017. [Online]. Available: https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.stats.skew.html

[29] P. Kleberger and T. Olovsson and E. Jonsson, "Security Aspects of the In-Vehicle Network in the Connected Car," in *Proc. of IV*, 2011, pp. 528–533.

[30] C. Goh and R. Law, "Modeling and forecasting tourism demand for arrivals with stochastic nonstationary seasonality and intervention," *Tourism management*, vol. 23, no. 5, pp. 499–510, 2002.