Multi-layer Reverse Engineering System for Vehicular Controller Area Network Messages

Xiaojie Lin[a], Baihe Ma[a], Xu Wang[a], Ying He[a], Ren Ping Liu[a] and Wei Ni[b]

[a]School of Electrical and Data Engineering, University of Technology Sydney, Sydney, Australia
{Xiaojie.Lin-2, Baihe.Ma-1}@student.uts.edu.au, {Xu.Wang-1, Ying.He, RenPing.Liu}@uts.edu.au
[b]Data61, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Sydney, Australia
Wei.Ni@data61.csiro.au

***Abstract*—The undisclosed Controller Area Network (CAN) decoding specification is important to the in-vehicle network (IVN) research for both industry and academia. Researchers have developed several CAN reverse engineering systems to predict signal boundaries and labels in order to map out CAN signal decoding specifications. Existing works mainly use one parameter (i.e., bit flip rate) to determine CAN signals boundary, which results in biased slicing and labelling of CAN signals. In this paper, we propose a multi-layer CAN reverse engineering system to cluster signal boundary at byte-level and label sliced CAN signal blocks at bit-level. The proposed system avoids biased signal slicing and labelling by introducing multiple parameters in signal classification, while existing works only use the bit flip rate and the number of unique value. The feasibility and adaptability of the proposed system is assessed by deploying it into a web application as a functionality module. We evaluate the proposed system with CAN messages from real cars. Compared with existing reverse engineering models, the proposed system introduces multi-layer signal processing to avoid over-slicing and over-labelling problem.**

***Index Terms*—Reverse Engineering, Controller Area Network, In-vehicle Network, In-vehicle Sensor Security**

I. Introduction

Automotive industry experiences the technological evolution and the wave of autonomous vehicles. Cutting-edge technologies such as the Advanced Driver Assistance System have been introduced to modern vehicles. Modern vehicles have become complex with the increasing number of sensors and Electronic Control Units (ECU). A large amount of data are generated and transmitted internally on the In-vehicle Network (IVN). The IVN can be regarded as a collaborative system, where sensors and ECU of IVN cooperate with each other. The legacy in-vehicle communication system is exposed to cybersecurity risks as a result. Controller Area Network (CAN) was originally developed by Bosch in the 1980s [1]. It is widely used as the communication bus protocol for the body control and powertrain sub-systems in IVN [2], [3]. The adversary can easily sniff the broadcast CAN messages via access points such as the On-board Diagnostics (OBD) port and the wireless Tire Pressure Monitoring System (TPMS). Koscher et al. manually reverse engineered the CAN messages and gained full control over the Body Control Module (BCM) functions of the test vehicle [4]. The security problem in IVN has drawn researchers' attentions due to the weak security guarantee of CAN.

Many researchers propose CAN Intrusion Detection System (IDS) solutions. The existing works encounter obstacles in generating practical datasets for the performance evaluation [5]. The generated dataset is biased without the correct CAN decoding specification. The CAN decoding specification is described by the CAN Database Container (DBC) file. The DBC files vary in different vehicle models and makers [6]. The Original Equipment Manufacturers (OEMs) keep DBC files secret as an important intellectual property. The CAN DBC file is different per OEM and per vehicle model. This motivates the CAN reverse engineering research to determine the CAN decoding specification.

In this paper, we propose a multi-layer system (i.e., bit and byte layer) to automatically reverse engineer the CAN messages. The contributions are threefold. Firstly, the proposed system considers both the byte-level and bit-level features of the CAN messages in signal segmentation. Secondly, the proposed system introduces multiple parameters in signal slicing and labelling. Thirdly, the system can be integrated as a functionality module to generate labels with a web application.

We evaluate the proposed system with real CAN messages from cars. The evaluation results show that the multi-layer reverse engineering design generate more signal labelling blocks than only one layer system design. We also compared the proposed system with another reverse engineering model named READ [7]. Compared with READ, the proposed system uses multiple parameters which can avoid CAN signal over-slicing.

The rest of the paper is organised as follows: Section II discusses the related works and Section III describes the proposed multi-layer reverse engineering system for CAN messages, including the CAN signal segmentation and labelling. Section IV evaluates the proposed system, and Section V concludes the paper.

II. Related Work

The CAN message is unauthenticated and encrypted due to its lightweight design. These features make CAN vulnerable to IVN attacks such as frame sniffing, frame injection, and replay attack [8]. The intuitive way of reverse-engineering CAN

messages is to manually infer functions of certain CAN identifier (CAN ID) by observing the consecutive CAN messages. To improve efficiency and accuracy, several automatic CAN reverse engineering systems have been proposed in recent years.

Automatic reverse engineering system is first developed in [9]. The authors calculated the maximum number of distinct values for all possible bit field assembles. In [7], Marchetti and Stabili used the bit-flip rate to calculate CAN signal boundary and labels. This is the first work that evaluates the system with the real CAN messages collected from different cars. This is also the first work that uses DBC files to validate the system. In [10], Pesé et al. used the bit flip rate in the CAN reverse engineering system, named LibreCAN. The LibreCAN extracts signals and translates the kinematic body-related data. The authors evaluated LibreCAN on real CAN data from cars and trucks.

Additional data resources and parameters for labelling CAN signals are explored. In [10], Pesé et al. collected the motion sensor data from a smartphone and the OBD-II data to classify signals. In [11], Kang et al. leveraged the OBD-II diagnostic data as additional reference to label signal blocks. Hoog et al. used the Artificial Neural Network to integrate prior knowledge of various vehicles and to produce more references [12]. The authors used the Pearson Correlation Coefficient and the FastDTW [13] to measure the correlation between the CAN signals and the artificial references. The mixed characteristics of different vehicles degrade the performance of the FastDTW. Wen et al. proposed a novel approach to use the car companion mobile apps commands as additional reference in [14]. However, most of the identified commands relate to the OBD-II diagnostic commands and miss the CAN specification details. In [15], Verma et al. took the open-source DBC files from the OpenDBC repository[1] as the ground truth of system evaluation.

The existing works have limitations as follows. Firstly, the existing works use a single parameter to slice the signal boundary. The bit flip rate or the maximum number of distinct values is used in past works. Secondly, limited labelling resources lead to small amount of identified signals. The translation of signals to vehicle functions mainly depends on the OBD-II diagnostic messages. Finally, most of the existing works overlook the evaluation of system feasibility and adaptability.

III. Proposed System

In this paper, we propose a new reverse engineering system which utilizes multiple parameters to slice and label CAN signals. Our system introduces multiple parameters to reduce error rate. We determine byte-level clusters to avoid over-slicing at bit-level, while existing models only process at bit-level. The output of our reverse engineering system is the predicted CAN decoding specification with sliced and labelled CAN signals. The architecture of the proposed reverse engineering system is shown in Fig. 1.

[1]openDBC

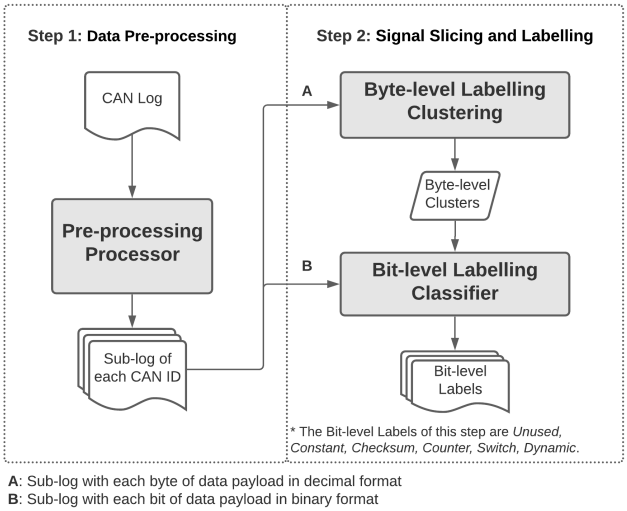


Fig. 1. Proposed reverse engineering system.

The proposed system is a multi-layer system with breakdown reverse engineering tasks. There are two steps in the proposed system.

1. **Data collection and pre-processing**: CAN messages logs of the target vehicle is collected and separated by CAN ID.
2. **Signal slicing and labelling**: The CAN signals are identified and labelled as ***Unused***, ***Counter***, ***Checksum***, ***Switch***, ***Constant*** or ***Dynamic***. The separate CAN ID sub-logs are analysed at byte-level and bit-level.

The predefined labels summarize all categories of CAN signals. *Unused* signals stand for vacant bits that never changes; *Counter* signals represent the bits as incrementing counters or rolling counters of CAN frame; *Checksum* signals mean the cyclic redundancy check (CRC) of CAN messages; *Switch* signals indicate state signals with limited values (e.g., 1 for door open and 0 for door close); *Constant* signals imply those signals with unchanged values but not *Unused* bits; *Dynamic* signals represent kinematic values which changes linearly with motion (e.g., engine speed and vehicle speed).

Compared with the existing works, we introduce byte-level processing and multiple parameters (such as, xxxxx) in the proposed system to improve the labelling accuracy and coverage.

A. Data Collection and Pre-processing

The log of raw CAN packets contains messages of different CAN IDs with the data field in hex format. The proposed system firstly extracts different CAN IDs from the collected log. Then, the proposed system produces sub-logs for each CAN ID.

The active functions of the target vehicle are triggered to collect CAN message logs that contain comprehensive signals. The CAN log recorded by the PCAN software. The default format of the CAN log is shown in Fig. 2. The CAN log

Time Stamp	ID	Extended	Dir	Bus	LEN	D1	D2	D3	D4	D5	D6	D7	D8
1844512521.291080	411	FALSE	Rx	0	8	5	FF	3F	0	6	0	0	18
1844512521.291090	410	FALSE	Rx	0	8	9	0	1D	12	0	0	0	C1
1844512521.291100	514	FALSE	Rx	0	8	0	0	7B	0	4C	C1	D3	0
1844512521.291100	601	FALSE	Rx	0	8	8	0	0	0	0	0	0	0

Fig. 2. Sample CAN log recorded by the PCAN software.

	D1	D2	D3	D4	D5	D6	D7	D8
Input CAN log Hex Value	0x05	0xFF	0x3F	0x00	0x06	0x00	0x00	0x18
Byte-level sub-log Decimal Value	5	255	63	0	6	0	0	24
Bit-level sub-log Binary Value	00000101	11111111	01100011	00000000	00000110	00000000	00000000	00100100

Fig. 3. Data payload sample of the converted sub-log.

contains timestamp, CAN ID, frame type, CAN Bus channel, length of the data field and the value of each byte in hex format.

The pre-processing processor produces sub-logs of different CAN IDs from collected CAN logs. For each CAN ID, the pre-processing processor produces two sub-logs with the bit-level data field in binary format and decimal format, respectively. A sample of the converting result is shown in Fig.3.

B. Signal Slicing and Labelling

This step aims to slice and label signal blocks at the byte-level for initialling the signal delimiter. We observed DBC files from the OpenDBC repository and identified that most of CAN signals only occupy one or two bytes. Thus, we assume that signals representing different vehicle functions do not share byte with others. The proposed CAN reverse engineering system uses multiple parameters as given in Table I.

The labelling processors analyse the sub-logs M_C and m_C at the byte-level and bit-level, respectively. M_C and m_C include all CAN messages of the CAN ID C in the collected logs. M_C has the data payload in decimal format for each byte while that of $m_C$is in binary format for each bit. Algorithm 1 and Algorithm 2 describe the byte-level and bit-level labelling. The byte-level labelling occurs before the bit-level labelling to mitigate the decision error due to the same bit flip rate.

TABLE I
PARAMETERS DENOTATION

Notation	Description
B_i/b_k	Byte flip rate of $Byte_i$/Bit flip rate of Bit_k
V_i/v_{mn}	Distinct byte/bit value set of $Byte_i$/Bit_{mn}
U_i/u_{mn}	Distinct byte value rate of $Byte_i$/Bit_{mn}
P_i/p_{mn}	Byte-level/Bit-level value differences set of $Byte_i$/Bit_{mn}
A_i/a_k/a_{mn}	Average byte/bit/bit value of $Byte_i$/Bit_k/Bit_{mn}
M_C/m_C	Sub-log whose data payload is decimal/binary format for each byte/bit
Θ_i/θ_{mn}	Byte-level/bit-level labelling parameter of $Byte_i$/Bit_{mn}
G/g	Byte-level/Bit-level labelling function of signals
T_C	The number of CAN messages of the sub-log M_C or m_C
β/ϕ	Byte/Bit flip rate function

Algorithm 1: Byte-level labelling algorithm

Input : sub-log M_C, sub-log messages amount T_C
Output: $byteLabels$, $byteMagnitude$, $byteDistinctValue$, $byteSignalDifferences$

while $Byte_i$ *in CAN Frame Data Field of* M_C **do**
  Compute the average value A_i of $Byte_i$;
  Compute the byte flip rate B_i of $Byte_i$;
  Append the distinct value of $Byte_i$ into $byteDistinctValue[i]$;
  Append the value differences of $Byte_i$ into $byteSignalDifferences[i]$;
  Compute the distinct byte value rate U_i;
  Compute the $byteMagnitude[i] \leftarrow \Theta_i = B_i \times U_i$;
  Label $Byte_i$ and make $byteLabels[i] \leftarrow G(\Theta_i, A_i)$;
end

1) ***Byte-level****:* CAN frame uses 11 bits (or 29 bits) of arbitration field to represent CAN identifier and uses 64 bits (i.e., 8 bytes) of data field to store transmitted data. $Byte_i$ is the i-th byte in the data field of the CAN frame, $1 \leq i \leq 8$. B_i denotes the byte flip rate of the $Byte_i$ with the sub-log M_C as in (1). Note that $0 \leq B_i \leq 1$. There are T_C rows of CAN messages in the sub-log M_C. M_C and m_C have the same amount of CAN messages. $Byte_{ij}$ is the decimal format $Byte_i$ of the j-th message in the sub-log M_C, $1 \leq j \leq T_C$. β is the byte flip rate function and can be given by (2). β produces the value as 0 when $Byte_i$ in adjacent messages are the same.

$$B_i = \frac{\sum_{j=1}^{T_C-1} \beta(M_C, i, j)}{T_C - 1}. \tag{1}$$

$$\beta(M_C, i, j) = \begin{cases} 0 & \text{if } Byte_{i(j+1)} = Byte_{ij}; \\ 1 & \text{if } Byte_{i(j+1)} \neq Byte_{ij}. \end{cases} \tag{2}$$

The proposed system uses multiple parameters, including the average byte value A_i, the distinct byte value set V_i, the distinct byte value rate U_i, and the byte-level value differences set P_i of $Byte_i$. The system traversals $Byte_i$ of each CAN message in the sub-log M_C and calculates the average value of $Byte_i$ as A_i. The distinct values of $Byte_i$ are put into V_i. (3) defines the distinct byte value rate U_i of $Byte_i$ in M_C, where $|V_i|$ is the number of values in V_i. The proposed system also calculates the difference of $Byte_i$ between adjacent messages in M_C and put the difference of $Byte_i$ into P_i. P_i indicates the value changing rule for the *Counter* and *Checksum* signals.

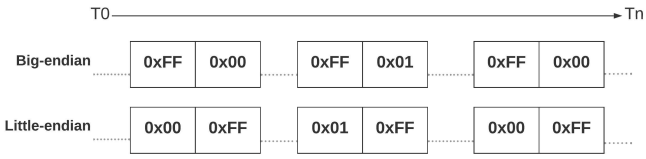


Fig. 4. Endianness decision from the value change.

$$U_i = \frac{|V_i|}{T_C}, \quad 1 \leq |V_i| \leq T_C. \tag{3}$$

The proposed system defines the labelling parameter Θ_i to determine the byte-level label of $Byte_i$ as in (4). Θ_i uses the byte flip rate B_i and the distinct byte value rate U_i of $Byte_i$ to maximise the dissimilarity of different labels. It is worth noting that $0 \leq \Theta_i \leq 1$.

$$\Theta_i = B_i \times U_i. \tag{4}$$

A threshold ε of Θ_i is used for distinguishing different signals labelling. However, it is difficult to differentiate the *Unused* and *Constant* signals since the two signals have the same value of Θ_i as 0. The proposed system uses the average byte value A_i of $Byte_i$ to solve this problem. (5) describes A_i. A_i of the *Unused* signal equals zero, which discerns the *Unused* and *Constant* signals.

$$A_i = \frac{\sum_{j=1}^{T_C} Byte_{ij}}{T_C}. \tag{5}$$

The byte-level labelling function G of the proposed system is given by (6). ε is set to determine labels. The ε_0, ε_1 and ε_2 are the decision bounds among the *Switch*, *Counter*, *Checksum* and *Dynamic* signals.

$$G(\Theta_i, A_i) = \begin{cases} Unused & \text{if } \Theta_i = 0 \quad and \quad A_i = 0; \\ Constant & \text{if } \Theta_i = 0 \quad and \quad A_i \neq 0; \\ Switch & \text{if } 0 < \Theta_i \leq \varepsilon_0; \\ Counter & \text{if } \varepsilon_0 < \Theta_i \leq \varepsilon_1; \\ Checksum & \text{if } \varepsilon_1 < \Theta_i \leq \varepsilon_2; \\ Dynamic & \text{if } \varepsilon_2 < \Theta_i \leq 1. \end{cases} \tag{6}$$

Byte-level labelling function G ends up assigning labels to each byte of the analysed CAN ID C. The labels produced from G is used for debugging purpose by comparing with the assigned labels from bit-level labelling function g.

The endianness is reverse-engineered when the signals occupy more than one byte. The left byte position reaches the 0xFF before the right byte position for the big-endian as shown in Fig. 4. The endinness of byte-level blocks determines the way to caculate the value of bit-level parameters.

Algorithm 2: Bit-level labelling algorithm

Input : sub-log m_C, sub-log messages amount T_C, $ByteLabels$, $ByteMagnitude$
Output: $bitLabels$, $bitDistinctValue$, $bitSignalDifferences$

while Bit_k *in the Byte-level signal block of* m_C **do**
  Compute the bit flip rate b_k;
  Compute the average bit value a_k;
end
Split the bit-level signal blocks by removing *Unused* bits;
foreach $Bit_{mn} \in$ *the bit-level signal blocks* **do**
  Compute the average value a_{mn};
  Append the distinct value of Bit_{mn} into $bitDistinctValue[mn]$;
  Append the value differences of Bit_{mn} into $bitSignalDifferences[mn]$;
  Compute the distinct bit value rate u_{mn};
  Compute the bit-level labelling magnitude θ_{mn};
  Label Bit_{mn} and make $bitLabels[mn] \leftarrow g(\theta_{mn}, a_{mn})$
end

*2) **Bit-level**:* Bit_k is the k-th bit position of the analysed byte $Byte_i$ or bytes, $1 \leq k \leq 64$. The system firstly calculates the bit flip rate b_k for each Bit_k. (7) and (8) describes the bit flip rate b_k and the the bit flip rate function ϕ, respectively. The k denotes the the k-th bit position Bit_k of the analysed byte or bytes. The j denotes the the j-th CAN message of the sub-log m_C, $1 \leq j \leq T_C$.

$$b_k = \frac{\sum_{j=1}^{T_C-1} \phi(m_C, k, j)}{T_C - 1}. \tag{7}$$

$$\phi(m_C, k, j) = Bit_{k(j+1)} \oplus Bit_{kj}. \tag{8}$$

The bit-level labelling processor truncates the byte-level signal block into multiple bit-level blocks Bit_{mn} by removing the *Unused* bits. The *Unused* bits are easily identified with both the b_k and the average bit value a_k equal 0. The average bit value a_k of Bit_k is shown in (9). The average bit value a_{mn} of Bit_{mn} is described in (10).

$$a_k = \frac{\sum_{j=1}^{T_C} Bit_{kj}}{T_C}. \tag{9}$$

$$a_{mn} = \frac{\sum_{k=m}^{n} a_k}{n - m + 1}, \quad m \leq n. \tag{10}$$

Given the signal blocks, the proposed system further defines more parameters. The distinct values of Bit_{mn} make up the distinct bit value set v_{mn}. The block takes up only one bit if m equals n. The associated endianness from the byte-level labelling processor is considered to convert the value of the block signal from binary into decimal format. $|v_{mn}|$ is the

number of values in v_{mn}. (11) defines the distinct bit value rate u_{mn} of the signal block Bit_{mn}.

$$u_{mn} = \frac{|v_{mn}|}{T_C}, \quad 1 \leq |v_{mn}| \leq T_C. \tag{11}$$

p_{mn} contains all the differences of Bit_{mn} between adjacent messages in m_C. v_{mn} and p_{mn} indicate the DBC file specification of each bit-level signal block such as the min and max value. v_{mn} and p_{mn} are the same as the V_i and P_i if only the *Unused* signal block or one signal block except the *Unused* signals locates within the analysed byte or bytes.

The bit-level labelling parameter θ_{mn} of the signal block Bit_{mn} is denoted in (12). Note that $\exists \theta_{mn} \propto \Theta_i, \forall Bit_{mn} \in Byte_i$. The bit-level labelling function g uses θ_{mn} to associate the minimised signal block with the new label as in (13).

$$\theta_{mn} = \frac{\sum_{k=m}^{n} b_k}{n-m+1} \times u_{mn}, \quad m \leq n. \tag{12}$$

$$g(\theta_{mn}, a_{mn}) = \begin{cases} Unused & \text{if } \theta_{mn} = 0 \quad and \quad a_{mn} = 0; \\ Constant & \text{if } \theta_{mn} = 0 \quad and \quad a_{mn} \neq 0; \\ Switch & \text{if } 0 < \theta_{mn} \leq \epsilon_0; \\ Counter & \text{if } \epsilon_0 < \theta_{mn} \leq \epsilon_1; \\ Checksum & \text{if } \epsilon_1 < \theta_{mn} \leq \epsilon_2; \\ Dynamic & \text{if } \epsilon_2 < \theta_{mn} \leq 1. \end{cases} \tag{13}$$

IV. Experiment

A. Experiment environment setting

The experiment was performed on a 2006 petrol vehicle with PEAK PCAN-USB Pro and a Macchina P1 board as shown in Fig. 5. The PEAK PCAN-USB Pro can listen and log CAN messages on CAN Bus. Customised python scripts run on Macchina P1 board to log CAN messages as well. The P1 board can further be the station for web application to analyse CAN message streaming.

The CAN Bus Y Splitter Cable is connected to OBD-II port of the test vehicle for accessing CAN Bus with multiple devices. Then, we connect the PCAN-USB Pro and P1 to the other end of Y cable as demonstrated in Fig. 6. We use PCAN-View from PEAK-System to work with PCAN-USB Pro for sniffing and logging CAN messages. We use PEAK Converter to convert the original *.trc* format of CAN log into the *.csv* format file. Laptop's memory and processor are 8GB and i5-7200U CPU@2.50Ghz, respectively.

We conduct the experiment in a vacant road at midnight to fully trigger all vehicle functions (e.g., acceleration and changing lanes) to and to avoid accident. The collected CAN logs are further input into a python script version of the proposed reverse engineering system to slice and label CAN signals.

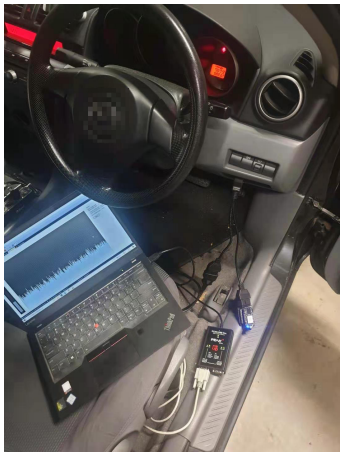


Fig. 5. Data collection experiment setup on test vehicle.

Fig. 6. PEAK PCAN-USB Pro and Macchina P1 board.

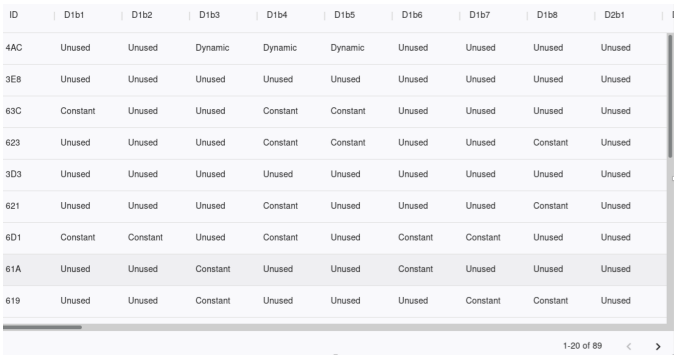
B. Experiment results

We summarise the experiment results on the test vehicle in Table II. The experiment CAN log is 100000 entries long with 26 CAN IDs. The proposed system split the log by CAN ID and analysed each CAN ID's sub-logs at byte-level and bit-level for slicing and labelling signals. The multi-layer design of reverse engineering produces more labels, from 156 signal labelling blocks at byte-level to 299 blocks at bit-level.

We simulate the reverse engineering model of READ [7] and compare the proposed system with READ. We noticed that our system is restricted by multiple parameters for slicing *Dynamic* signal but READ largely leverage the bit flip rate which tends to over-labelling *Dynamic* signal. For example, most significant bits of a big-endian *Dynamic* signal owns bigger bit-flip rate. Our system considers both the bit flip rate and unique value rate of signal blocks to avoid over-

TABLE II
Experiment Results and Comparision

Tested System (layer)	Number of CAN ID	Number of Signal Labeling Blocks
Byte-level of proposed system	26	156
Bit-level of proposed system	26	299
READ	26	322



ID	D1b1	D1b2	D1b3	D1b4	D1b5	D1b6	D1b7	D1b8	D2b1
4AC	Unused	Unused	Dynamic	Dynamic	Dynamic	Unused	Unused	Unused	Unused
3E8	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
63C	Constant	Unused	Unused	Constant	Constant	Unused	Unused	Unused	Unused
623	Unused	Unused	Unused	Constant	Constant	Unused	Unused	Constant	Unused
3D3	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
621	Unused	Unused	Unused	Constant	Unused	Unused	Unused	Constant	Unused
6D1	Constant	Constant	Unused	Constant	Unused	Constant	Constant	Unused	Unused
61A	Unused	Unused	Constant	Unused	Unused	Constant	Unused	Unused	Unused
619	Unused	Unused	Constant	Unused	Unused	Unused	Constant	Constant	Unused

1-20 of 89

Fig. 7. Reverse engineering system integrated as module into a web application with experiment on CAN log from a 2019 petrol vehicle.

slicing a two byte long *Dynamic* signal but READ tends to further truncate *Dynamic* signal. Compared with READ, our proposed system reduces error labelling rate and achieves higher accuracy with less labels. The proposed system can be utilized in real-world in-vehicle scenarios due to the low computational consumption and low data collection overhead. The experiments also prove that the proposed system has high scalability to be further deployed in more complex scenarios.

C. System feasibility and adaptability

To feed the demand from our industry partner, we further evaluated the feasibility and adaptability of our reverse engineering system. We integrated the proposed system with a web application which is a vehicle penetration platform from our industry partner. The labelling outcomes at bit-level is shown in Fig. 7 where the uppercase D refers to byte position and lowercase b stands for bit position. The test CAN log is collected from a 2019 petrol vehicle. It costs at most 10 minutes to generate the labelling outcomes by using over 1 million entries of test CAN log. The running time decreases with smaller size of CAN logs. The experiment shows the proposed reverse engineering system can be easily added as module to other application or platform, which benefits the IVN security stakeholders.

V. Conclusions

In this paper, we proposed a multi-layer reverse engineering system to analyze CAN messages at both byte-level and bit-level. By further slicing signal blocks and labelling at bit-level under the sliced byte-level signal blocks, the proposed system outperforms other work in terms of the amount of predicted labels. The proposed reverse engineering system is feasibility and adaptability that can be added into other platform or application as a module. By using the proposed system, IVN security researchers and industry engineers can obtain a predicted decoding specification for CAN frame.

In future work, we plan to improve the proposed system by introducing advanced matching algorithm to associate more descriptive labels with signal blocks with reference from OBD-II diagnostic messages. In this paper, we only evaluate the proposed reverse engineering system on limited cars due to restrictions of COVID-19 lockdown. Thus, we plan to evaluate the proposed system with more vehicle variables (i.e., the amount, models and makers of cars) in the future. We also plan to improve the computation complexity of the proposed system to deal with huge amount of CAN logs.

Acknowledgment

We thank our industry partner for providing access to test vehicles. We thank Thanh Phuoc Nguyen and Jacob Pace for helping with technical issues and experimental evaluation. This work is funded in part by the University of Technology Sydney, in part by the Insurance Australia Group (IAG), and in part by the iMOVE CRC under Grant 5-028. This work is also supported by the Cooperative Research Centres program, an Australian Government initiative.

References

[1] G. Leen and D. Heffernan, "Expanding automotive electronic systems," *Computer*, vol. 35, no. 1, pp. 88–93, 2002.

[2] M. K. Ishak and F. K. Khan, "Unique message authentication security approach based controller area network (can) for anti-lock braking system (abs) in vehicle network," *Procedia Computer Science*, vol. 160, pp. 93–100, 2019.

[3] W. Zeng, M. A. Khalid, and S. Chowdhury, "In-vehicle networks outlook: Achievements and challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1552–1571, 2016.

[4] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *The Ethics of Information Technologies*. Routledge, 2020, pp. 119–134.

[5] W. Wu, R. Li, G. Xie, J. An, Y. Bai, J. Zhou, and K. Li, "A survey of intrusion detection for in-vehicle networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2019.

[6] C. Young, J. Svoboda, and J. Zambreno, "Towards reverse engineering controller area network messages using machine learning," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE, 2020, pp. 1–6.

[7] M. Marchetti and D. Stabili, "Read: Reverse engineering of automotive data frames," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1083–1097, 2018.

[8] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures: Challenges and future directions," *IEEE Network*, vol. 31, no. 5, pp. 50–58, 2017.

[9] M. Markovitz and A. Wool, "Field classification, modeling and anomaly detection in unknown can bus networks," *Vehicular Communications*, vol. 9, pp. 43–52, 2017.

[10] M. D. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. G. Shin, "Librecan: Automated can message translator," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2283–2300.

[11] T. U. Kang, H. M. Song, S. Jeong, and H. K. Kim, "Automated reverse engineering and attack for can using obd-ii," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2018, pp. 1–7.

[12] J. de Hoog, T. Bogaerts, W. Casteels, S. Mercelis, and P. Hellinckx, "Online reverse engineering of can data," *Internet of Things*, vol. 11, p. 100232, 2020.

[13] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[14] H. Wen, Q. Zhao, Q. A. Chen, and Z. Lin, "Automated cross-platform reverse engineering of can bus commands from mobile apps," in *Proceedings 2020 Network and Distributed System Security Symposium (NDSS'20)*, 2020.

[15] M. E. Verma, R. A. Bridges, J. J. Sosnowski, S. C. Hollifield, and M. D. Iannacone, "Can-d: A modular four-step pipeline for comprehensively decoding controller area network data," *arXiv preprint arXiv:2006.05993*, 2020.