Towards Reverse Engineering Controller Area Network Messages Using Machine Learning

Clinton Young
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 50011
Email: cwyoung@iastate.edu

Jordan Svoboda
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 50011
Email: jordans2@iastate.edu

Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 50011
Email: zambreno@iastate.edu

***Abstract*—The automotive Controller Area Network (CAN) allows Electronic Control Units (ECUs) to communicate with each other and control various vehicular functions such as engine and braking control. Consequently CAN and ECUs are high priority targets for hackers. As CAN implementation details are held as proprietary information by vehicle manufacturers, it can be challenging to decode and correlate CAN messages to specific vehicle operations. To understand the precise meanings of CAN messages, reverse engineering techniques that are time-consuming, manually intensive, and require a physical vehicle are typically used. This work aims to address the process of reverse engineering CAN messages for their functionality by creating a machine learning classifier that analyzes messages and determines their relationship to other messages and vehicular functions. Our work examines CAN traffic of different vehicles and standards to show that it can be applied to a wide arrangement of vehicles. The results show that the function of CAN messages can be determined without the need to manually reverse engineer a physical vehicle.**

***Index Terms*—Controller area network, automotive security, machine learning, classification, reverse engineering**

I. Introduction

The modern day vehicle has become increasingly technologically advanced with the integration of embedded systems. With the adoption of the Controller Area Network (CAN) in vehicles to enable communication between Electronic Control Units (ECUs), vehicles now have more features but are also more vulnerable to cyber attacks. Koscher and Checkoway et al. [1], [2] were the first to demonstrate physical and remote attacks on real vehicles. Additionally, there are numerous other works that have demonstrated the vulnerabilities in modern day vehicles due to the integration with CAN [3]–[6].

In this paper, we present our efforts that have focused on the reverse engineering and classification of CAN messages. The problem is that even though CAN is standardized, the implementation may vary for different manufacturers and vehicle models. These implementations are kept secret, and consequently CAN messages for every vehicle need to be analyzed and reverse engineered in order to obtain information. Due to the lack of publicly available CAN specifications, attackers and researchers need to reverse engineer messages to pinpoint which messages will have the desired impact. The reverse engineering process is needed by researchers and hackers for all manufacturers and their respective vehicles to understand what the vehicle is doing and what each CAN message means. The knowledge of the specifications of CAN messages can improve the effectiveness of security mechanisms applied to CAN.

The rest of this paper is organized as follows. Section 2 surveys related work in automotive security and machine learning. Section 3 discusses our approach in classifying CAN messages. We present our results in Section 4. Finally, in Section 5 we summarize and conclude our efforts.

II. Related Works

Cyber attacks towards modern vehicles executed by injecting spoofed messages to the CAN bus has spawned numerous research efforts towards improving the security of modern vehicles. Koscher and Checkoway et al. [1], [2] were the first to demonstrate physical and remote security breaches in automotive systems. Both research groups injected malicious messages to take control of various vehicle systems such as brakes and engines. Valasek and Miller [7] demonstrated real-world attacks on multiple vehicles via remote hacking to cause a Jeep to brake and crash into a ditch. Their work led to a Chrysler recall of 1.4 million vehicles.

To detect attacks, some approaches use machine learning approaches. Kang et al. [8] trained a deep neural network structure to classify normal vs attack messages using probability-based features of the message bits. Using normal and attack data, the system was able to be trained to recognize specific attacks. Kang et al. [9] proposed an automated process for correlating CAN messages with its ECU by creating a machine learning classifier trained on multiple vehicles. Their approached utilized the time stamp, CAN ID, and data fields and determined that a nearest neighbor approach would have the best success as a classifier. They evaluated multiple machine learning approaches using the same training data to draw their conclusion. Lestyan et al. [10] proposed a method of identifying and extracting vehicle sensors from raw CAN data to infer personal driving behavior. Their approach examined each message data bit for the probability that it was 1. Using a random forest based approach, their algorithm attempted to classify individual messages.

These detection approaches require the analysis of raw CAN messages by manually inspecting high volumes of data to reverse engineer message syntax and semantics to reconstruct the vehicle operations and contextualize the messages. Literature on network traffic analysis and automatic recognition of the nature of given network packets already exists. However, these works are inapplicable to the automotive network as CAN has less structure in its protocols. For example, CAN messages do not include conventional networking protocol features such as source or destination addresses or port numbers. Conventional networks also have a clear separation between network and application layers which is not the case with CAN.

There exists separate research on the reverse engineering and translation of CAN messages. Marchetti and Stabili [11] proposed READ, a novel algorithm for automatic reverse engineering of CAN data frames. READ analyzes traffic traces containing unknown CAN messages to identify and label the signal type based on their data frames. Their method isolates counters and cyclic redundancy checks (CRCs), among other values, to label the signals. Verma et al. [12], proposed a simple algorithm to extract CAN message signals and label them using OBD-II PIDs. Their algorithm, ACTT: Automotive CAN Tokenization and Translation, leverages diagnostic information to parse CAN by breaking messages into tokens and then learning the translation from bits to vehicle function. Their signal extraction only identifies signals that do not have a contiguous set of bits. Pese et al. [13], developed a tool called LibreCAN, which translates CAN messages. This tool captures the bit-flip rate of messages and uses them along with body data from a phone to classify messages.

The main limitation of these works is the few features that can be extracted and analyzed primarily from CAN traffic. This is due to the fact that access to an operating vehicle is typically required to manually reverse engineer the CAN IDs and to validate results. This issue can be mitigated if access to the complete specifications of CAN messages were available. However, as previously mentioned, CAN IDs and their functions are manufacturer trade secrets. Additionally, manufacturers have an incentive to prevent reverse engineering of their CAN messages, as it is the main deterrent for hackers. This paper proposes the novel contribution of reverse engineering CAN messages without the need for a physical vehicle.

A. Ground Truth Data

1) DBC file: A .dbc file (database for CAN) contains the translation for all CAN messages in a particular vehicle. This file is typically held secret by manufacturers and varies per make and model. DBCs contain the signal definitions, segment position (start and end bits), binary to decimal encoding scheme, and the conversion information for translating the decimal to meaningful physical value. The DBC also includes message timing information such as frequency and whether the message timing is constant or triggered by an event, and the corresponding ECU(s) responsible for a message. Nearly all research requires reverse engineering some of the information of DBCs.

2) J1939: CAN data can follow the Society of Automotive Engineers standard SAE J1939 [14], which is the vehicle bus recommended practice for vehicle component communication and diagnostics. SAE J1939 has been widely adopted by diesel engine manufacturers for use in large tractors and trucks. This standard defines that all J1939 packets, except for the request packet, should contain eight bytes of data and a standard header which contains a Parameter Group Number (PGN), that is embedded in the message's 29-bit identifier. A PGN identifies a message's function and associated data. J1939 attempts to define standard PGNs to encompass a wide range of automotive and other vehicle purposes.

III. Proposed Approach

In this paper we propose to reverse engineer CAN messages without manually testing on a physical vehicle and without the use of a DBC file. We describe the process of extracting CAN message features and their utilization in machine learning techniques. We utilize machine learning techniques to reduce the obstacle of reverse engineering CAN messages by identifying important CAN messages and their functions. The contribution of our work can be broken down into:

- A supervised learning approach that extracts and identifies CAN data changes and labels specific IDs corresponding to vehicular functions.
- An unsupervised clustering approach that classifies unknown CAN messages using the labels learned with our supervised learning approach. Unknown CAN functions are extrapolated from known CAN messages.

Our work provides an early demonstration that by utilizing machine learning techniques, it is possible to reverse engineer CAN message functions without needing to manually test against a physical test vehicle. Our work indicates that the absence of a DBC file does not necessarily prevent a researcher or attacker from deriving the meaning of individual CAN messages.

A. Feature Extraction

1) Message Feature Extraction: We first needed to extract the CAN fields and features required for our approach, as machine learning approaches require specific input feature formatting to properly generate classification. To do so we developed a simple a CAN message parser that extracts CAN IDs, time stamps, and data fields for every message. These CAN fields are used to create ordered dictionaries and lists of CAN messages with the same ID. For our feature engineering process, we determined it was pertinent to choose features that are standard to all vehicles. The CAN fields our parser extracts are in all CAN messages regardless of vehicle's make or model. Additionally, these CAN fields are used to extract other features such as message timings and distances.

2) CAN Function Extraction: Without the full CAN specification, researchers need to reverse engineer CAN messages to determine their functions and respective data details. The typical method of reverse engineering this information is to manually test individual CAN messages on a physical vehicle. We propose an alternative method to reverse engineer CAN messages with the use of labeled time frames that correspond to specific vehicular functions and a data change algorithm that detects CAN messages with changing data bits. We hypothesize that there are specific CAN messages that correspond to specific vehicular functions. For example, when a vehicle is braking, the brake controller is active, therefore a certain ECU is transmitting braking signal to the brake from the brake pedal. The data field of this certain CAN ID is changing as the brake pedal pressure is increasing and the brake is being engaged. We utilize a CAN data bit change detector in combination with labeled CAN logs to reverse engineer a few CAN messages.

Our supervised learning approach takes as input a labeled CAN data set that detailed what the vehicle was doing in certain time frames. Our approach monitors all the CAN messages in the time frame for data bit changes. The contribution here is that given minimal knowledge on what the vehicle is doing in certain time frames and by detecting CAN messages with changing data bits, it is possible to reverse engineer some CAN message functions. We describe our data change detection approach in Algorithm 1.

Algorithm 1: Obtaining data values for CAN data frame and checking for data bit changes

```
Data: CAN Data Raw
Result: List of CAN IDs and training features
training features = [”CAN ID”:[], ”Time Stamp”:[], ”Data”:[], ”Vehicle Function:[]]
for Every message in CAN Log do
    Parser(message): To obtain individual CAN fields
    for Every CAN ID do
        Create dictionary of CAN fields and data
        Continuously monitor data field bits for changes
    if message data bit changes then
        Record CAN ID and vehicular function
    else
        pass
Return CAN data fields and training features
```

Our algorithm associates labels with CAN messages with respect to their vehicular function. CAN function extraction is important in the next step of our approach as it provides CAN message labels to help classify unknown CAN messages. Given that our data sets are unlabeled, this approach provides the labeling needed for our machine learning approaches.

B. CAN Clustering

We utilize hierarchical clustering towards reverse engineering CAN message functions. A basic understanding of how K-means clustering works is needed to explain hierarchical clustering. K-means can be broken down into these sections:

1) Decide number of clusters (k)
2) Select random points from the data as centroids
3) Assign all points nearest cluster centroid
4) Calculate centroid of new clusters
5) Repeat steps 3 and 4

This is an iterative process; it keeps running until the centroids of newly formed clusters do not change or the max number of iterations is reached. The issue with K-means clustering is that it always tries to make the clusters the same size. Additionally, the number of clusters needs to be defined at the beginning of the algorithm. Realistically, the number of clusters is unknown. Hierarchical clustering addresses these issues. We implement agglomerative clustering, a type of hierarchical clustering, to cluster CAN messages together by related vehicular functions. Each CAN ID is assigned as a single cluster and as we iterate through the IDs and their respective distance measurements, then CAN ID clusters are merged starting with the nearest neighbor. By grouping CAN IDs together, related CAN IDs are clustered together by related function. Therefore, if certain CAN IDs have a known function, the functions of unknown CAN IDs can be determined.

We combine a supervised learning approach that reverse engineers some CAN ID message functions and then using these extracted labels towards classifying other unknown CAN message functions with the use of agglomerative clustering.

C. Reverse Engineering CAN Messages

The core of our work is to reverse engineer CAN messages with respect to their vehicular function. Raw CAN is unlabeled and requires proprietary information from the manufacturer to reverse engineer. This is a major impediment to researchers and attackers, and we address this obstacle. Our work utilizes raw CAN data, captured from a variety of vehicles to derive CAN clusters. The goal is to circumvent the need for a physical test vehicle and manufacturer specific data towards reverse engineering CAN messages.

Our process involves reading CAN data, calculating the distances between all CAN IDs, merging nearest IDs together to form clusters, and using these clusters to classify unknown CAN IDs. We utilize Euclidean distance as our distance metric between CAN messages.

$$Message1 = m1 = (x_1, y_1)$$
$$Message2 = m2 = (x_2, y_2)$$
$$Euclideandistance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The CAN IDs with the smallest distance are merged and this process iteratively continues until there are k clusters left. The clusters left are grouping the different CAN messages together by their respective functions.

Our work is two-fold, the labeling of the clusters and the visualization of clusters. A tool utilized for our clustering is a dendrogram. This is used to visualize hierarchical clustering and determines the optimal number of k clusters. The tree-like structure records the sequences of merges of the leaves, in this case the CAN messages. In addition to depicting how the CAN IDs merge, the dendrogram also details the distances between IDs and clusters.

Algorithm 2: Pseudocode for creating a dendrogram and returning predicted data clusters

```
Data: Distance values
Result: Dendrogram and cluster labels
Create distance matrix;
matrix = DataFrame(squareform(distance values));
Create dendrogram;
dendrogram = scripy.dendrogram(linkage(distance
  values));
Create clusters;
cluster = AgglomerativeClustering();
cluster.predict(matrix);
Return dendrogram and clusterings
```

Algorithm 3: Clustering CAN IDs and labeling of messages

```
Data: CAN Data Raw
Result: CAN Message Clusters
[”Cluster”: ”Label”, ”CAN IDs”:[]]
for Every message in CAN Log do
    Parser(message): To obtain individual CAN fields
    for Every CAN ID do
        Calculate the distance of this message from
          every other message
    for Every Message do
        Merge closest CAN IDs
        Continue merges til K clusters are created
Return CAN data fields and training features
```

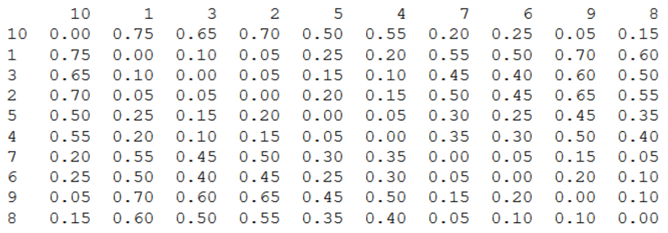
Our clustering approach creates clusters of CAN messages that are related by their vehicular function. Unfortunately, without labels for these CAN IDs, it is not possible to determine what these clusters represent, as they are just groupings of unknown CAN messages. To solve this issue, we utilize the CAN message labels we derived previously in our supervised learning approach. Using these labels, which relate specific vehicular functions to certain CAN IDs, we applied labels to unknown CAN messages.

In the following section, we analyze the CAN labels and the clusters that our approaches generated. We detail our process of verification and validation of the reverse engineering process. We demonstrate that our algorithms are correlating CAN messages to specific vehicular functions, and that these correlations can create labels for the CAN messages. We are also able to cluster CAN messages by their relative and related functions. Combining the clusters and labels enable reverse engineering of CAN message functions.

IV. Results and Evaluation

To evaluate our approach, we utilized three separate CAN data sets. Our initial work utilized a simulated CAN data log with ten different CAN IDs, simulating a driving mode and a braking mode. This data log had messages with different frequencies and message intervals to simulate real data. There were five IDs specifically related to driving and another four IDs related to braking, with one message that was transmitted during both modes. This data was used to verify and validate our algorithms as a control group. Given that raw CAN data is dependent on the vehicle it was captured from, our simulated data gives us a baseline to understand the workings of our algorithms when applied towards raw captured CAN data.

Our clustering algorithm and dendrogram required a distance matrix to be created. This matrix represents all the distance metrics for the CAN IDs (see Fig. 1). The matrix was passed to our dendrogram creator and clustering algorithm to cluster the CAN messages. The dendrogram, shown in Fig. 2, is created from the leaves first to create the cluster tree. The dendrogram shows that there are two clusters of the CAN IDs, which is expected since we simulated two separate driving modes of braking and accelerating. Our clustering algorithm confirmed the dendrogram results. CAN IDs 1-6 were designated as accelerating and were clustered into one group, and CAN IDs 7-10 were designated as braking and clustered into another group. The simulated data was used to confirm our hypothesis and approach before applying them to real CAN data.



```
      10     1     3     2     5     4     7     6     9     8
10  0.00  0.75  0.65  0.70  0.50  0.55  0.20  0.25  0.05  0.15
1   0.75  0.00  0.10  0.05  0.25  0.20  0.55  0.50  0.70  0.60
3   0.65  0.10  0.00  0.05  0.15  0.10  0.45  0.40  0.60  0.50
2   0.70  0.05  0.05  0.00  0.20  0.15  0.50  0.45  0.65  0.55
5   0.50  0.25  0.15  0.20  0.00  0.05  0.30  0.25  0.45  0.35
4   0.55  0.20  0.10  0.15  0.05  0.00  0.35  0.30  0.50  0.40
7   0.20  0.55  0.45  0.50  0.30  0.35  0.00  0.05  0.15  0.05
6   0.25  0.50  0.40  0.45  0.25  0.30  0.05  0.00  0.20  0.10
9   0.05  0.70  0.60  0.65  0.45  0.50  0.15  0.20  0.00  0.10
8   0.15  0.60  0.50  0.55  0.35  0.40  0.05  0.10  0.10  0.00
```

Fig. 1: Simulated distance matrix representing all distances between each CAN ID.

We also obtained CAN traffic logs from Oak Ridge National Labs [15]. Their configuration consisted of a vehicle set up on a dynamometer, a treadmill system that simulates real road operation. The CAN traffic was captured by connecting through the On-Board Diagnostic port under the dash of the vehicle. The data was not reverse engineered, as the CAN messages are unknown, but it captured the vehicle driving in different modes, from accelerating to braking. This data was labeled based on the vehicle operations. Typically CAN data captured is not reverse engineered, and to reverse engineer this data requires manual injection. Each captured CAN message is injected into the vehicle to see how the vehicle responds. Our supervised learning approach simulated this process of reverse engineering through manual injection by detecting CAN messages with changing data bits due to different vehicular driving modes. Upon examining our data set, we determined there were 133 unique CAN IDs. Applying our approach, we identify the significant CAN IDs and correlate them to specific vehicular functions.

We identified these CAN IDs, Table I, and reasoned that

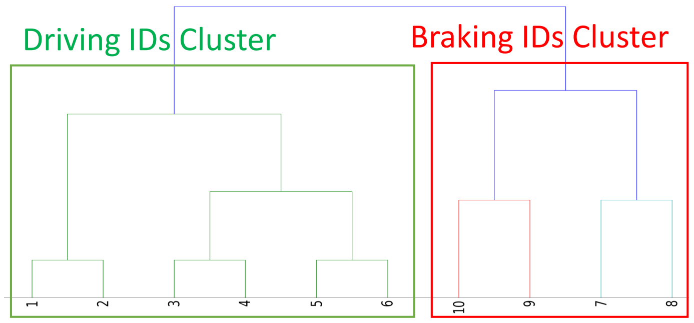


Fig. 2: Dendrogram for simulated data. This figure shows how each CAN ID clustered with the other IDs. It also shows the two major groupings for the CAN IDs.

Vehicular Function	CAN IDs
Drive	17C
Brake	467, 3C3
Shifter	171, 230
Windows and door lock	331, 332, 333

TABLE I: Reverse engineered CAN IDs and their corresponding vehicular function.

they have the specified functions. This conclusion was derived from a combination of knowing what the vehicle was doing at specific times in the log and utilizing the data change algorithm to detect which IDs were changing during these times.

By identifying these CAN IDs, it enabled assignment of labels for the clusters generated with our clustering algorithm. With more data, our algorithms would improve and be able to identify more CAN ID labels. The methodology applied here to derive CAN message functions can be applied to various different vehicles as it does not require any vehicle specific information on its messages.

The clustering algorithm was then applied to the Oak Ridge data set to obtain the CAN message clusters. First, a dendrogram was created with the distance matrix for the Oak Ridge data to visualize the clustering of messages. The dendrogram, shown in Fig. 3, indicated that the CAN IDs clustered into two major clusters and two minor clusters. The clustering algorithm returned the respective cluster labels for all the messages. Our algorithm separated out driving and braking CAN IDs into different clusters. CAN ID 17C for driving and IDs 230 and 171 for gear shifter are in the green cluster, whereas braking 467, 3C3 and other uncommon functions such as door lock were clustered in red.

Using these labels, we classify the functions of unknown CAN IDs. To check our clusters, we had a few other CAN IDs reverse engineered manually. CAN ID 430, we identified as shifting to drive. This ID was clustered together with shifter (171, 230) and drive (17C). CAN ID 465 was identified as shifting to reverse. This ID was clustered together with brake (467, 3C3). These results demonstrates that even given raw unlabeled CAN data, it is possible to reverse engineer some CAN messages and, using these known IDs, extrapolate the functions of other unknown IDs.

Cluster Label[Vehicle Function]	CAN IDs
0 (Driving)	216, 217, 214, 213, 37B, 3CC, 92, 326, 336, 40A, 483, 366, 265, 422, 415, 416, 3B6, 410, 411, 596, 312, 43D, 43E, 82, 83, 81, 85, 3B3, 24A, 3B7, 24C, 24B, 3B8, 368, 369, 367, 423, 365, 581, 42C, 440, 42D, 440, 42D, 242, 59E, 2A1, 434, 430, 453, 455, 333, 332, 179, 178, 175, 171, 4B0, 185, 78, 91, 165, 166, 167, 25B, 25C, 25A, 17C, 33B, 35E, 156, 3AB, 3AA, 230, 7D, 7A, 5B3, 47, 42, 41, 20A, 471, 14B, 352, 200, 202, 204, 350, 77, 76, 485, 484, 4B, 4C, 4A, 486, 3A8, 476, 477, 474, 475, 2EC
1 (Brake)	439, 454, 3C7, 3C3, 43C, 3EB, 3EA, 84, 2F1, 447, 446, 386, 431, 331, 482, 465, 467, 466, 488, 38D, 384, 472, 473
2	5A5
3	5B5
4	86, 87, 4BE, 4BF, 600

TABLE II: Cluster labels from Oak Ridge.

While these results are promising, the lack of reverse engineered data made verifying the results challenging. A proprietary manufacturer CAN specification would be needed to completely evaluate the classification of the messages. To address this issue, we obtained CAN data that followed the J1939 standard. This standard means that the PGNs are well-defined and therefore we can validate our clustering results. J1939 messages have an identifier, called a PGN (parameter group number), that defines the message function. Where automotive CAN messages can identify the message and the sending ECU depending on the manufacturer of the vehicle, J1939 messages define the message function and the source of the message.

The 3rd data set was CAN data captured from a John Deere tractor as it performed tillage operations at different sites. This data set is raw, has no labels, and contains over 300 CAN IDs. Using the same algorithms and process, a distance matrix was created for this data. This distance matrix was then used to create a dendrogram to visualize the CAN clustering and determine the number of cluster labels. The algorithm returned four clusters, Table III. The largest cluster contained all the driving critical functions, such as braking and engine controller. The other three clusters were much smaller and contained proprietary IDs, memory accesses, and data transfer

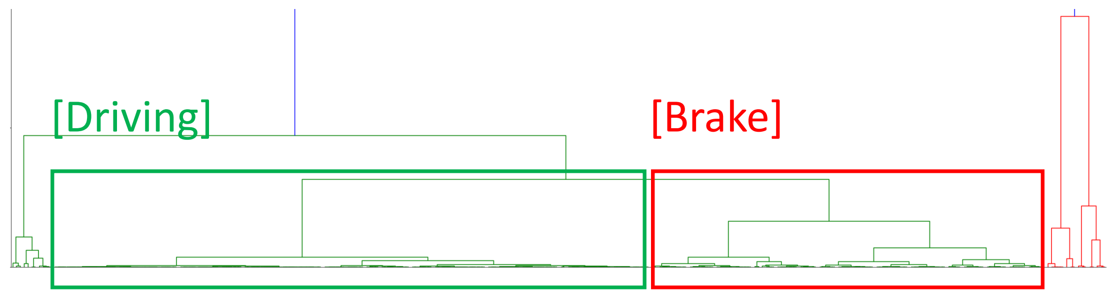


Fig. 3: Captured raw data dendrogram, clustered the CAN IDs into 5 groupings. The figure identifies the 2 major clusters related to Brake and Driving.

Cluster Label	CAN Function
0	Driving Functions (Engine and brake controllers)
1	Proprietary IDs
2	Memory Access IDs
3	Data Transfer IDs

TABLE III: Reverse engineered J1939 CAN IDs and their corresponding vehicular function.

IDs in separate clusters. The results using this labeled data set validates our CAN clustering by confirming that related CAN messages are being clustered together. Given that J1939 CAN data has known functions, it is possible to check and confirm that CAN IDs with related functions cluster together.

These results show that even with CAN logs from different types of vehicles with different standards, our algorithms were able to classify, identify, and label specific CAN IDs by their respective functions. Our algorithms do not require insider knowledge nor reverse engineered data to classify CAN IDs. The only input required was raw CAN data logs. Our results indicate that machine learning algorithms are able to classify CAN messages using raw CAN logs as the input.

V. Conclusion

This paper presents a novel application of machine learning towards the reverse engineering of CAN messages. Our proposed approach does not require prior insider knowledge about the CAN messages. We proposed a supervised method to extract CAN message function by analyzing change in their data fields respective to vehicular function. The results of this method are used to apply labels in our unsupervised learning method.

We propose utilizing the machine learning approach of clustering to cluster CAN messages together based on their function. This approach results in numerous clusters of CAN IDs, and with our previous labels we are able to classify the function for these unknown CAN messages. Our work is an initial step towards automating reverse engineering of CAN message features without the need to physically and manually test messages on a live vehicle.

Acknowledgment

This material is supported by the National Science Foundation under Grant No. CNS 1645987.

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, and H. Shacham, "Experimental security analysis of a modern automobile," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the USENIX Conference on Security*, Aug. 2011.

[3] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network," in *Proceedings of the International Conference on Information Networking (ICOIN)*, Jan. 2016.

[4] U. Larson, D. Nilsson, and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, Jun. 2008.

[5] M. Müter, A. Groll, and F. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *Proceedings of the International Conference on Information Assurance and Security (IAS)*, Aug. 2010.

[6] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, and K. Oishi, "A method of preventing unauthorized data transmission in controller area network," in *Proceedings of the IEEE Vehicular Technology Conference (VTC)*, May 2012.

[7] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," in *Black Hat USA*, 2014.

[8] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PLoS ONE*, vol. 11, no. 6, Jun. 2016.

[9] T. U. Kang, H. M. Song, S. Jeong, and H. K. Kim, "Automated reverse engineering and attack for CAN using OBD-II," in *Proceedings of the IEEE Vehicular Technology Conference (VTC)*, Aug. 2018.

[10] S. Lestyan, G. Ács, G. Biczók, and Z. Szalay, "Extracting vehicle sensor signals from CAN logs for driver re-identification," in *Proceedings of the International Conference on Information Systems Security and Privacy*, 2019.

[11] 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, Apr. 2019.

[12] M. Verma, R. Bridges, and S. Hollifield, "ACTT: automotive CAN tokenization and translation," in *Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec. 2018.

[13] M. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. Shin, "LibreCAN: Automated CAN message translator," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Nov. 2019.

[14] SAE International, "Serial control and communications heavy duty vehicle network," Jun. 2012. [Online]. Available: https://www.sae.org/standards/content/j1939_201206/

[15] Oak Ridge National Labs, "User facilities at the National Transportation Research Center," https://www.ornl.gov/facility/ntrc/research-areas/vehicle-systems, 2020.