**Fingerprinting Vehicles With CAN Bus Data Samples**

David R. Crow, Scott R. Graham, Brett J. Borghetti

Air Force Institute of Technology, Wright-Patterson Air Force Base, USA

[david.crow@afit.edu](mailto:david.crow@afit.edu)

[scott.graham@afit.edu](mailto:scott.graham@afit.edu)

[brett.borghetti@afit.edu](mailto:brett.borghetti@afit.edu)

**Abstract:** Today’s vehicle manufacturers do not tend to publish proprietary controller area network (CAN) packet formats. This is a form of *security through obscurity* – it makes reverse engineering efforts more difficult for would-be intruders – but obfuscating the CAN data in this way does not adequately hide the vehicle’s unique signature. Specifically, modern methods can identify a vehicle’s signature in a segment of its CAN data, even if these data are unprocessed or limited in scope. In deep learning, this is a multiclass classification problem which asks the following question: given a sample of CAN data, can we determine which vehicle generated the sample? To answer this question, we employ a dataset from Stone et al. (2018). The corpus contains over 230 megabytes of data from one capture on each of 11 different vehicles. Each capture contains the CAN messages generated while driving a predetermined, ten-minute route in the local area. The dataset contains over four million messages, each of which contains up to eight data bytes. In this research, 1,024 bytes of sequential CAN data constitute one data sample, so formatting, concatenating, and partitioning the messages gives over twenty-eight thousand individual vehicle-labeled samples. Two distinct deep learning models are trained using supervised machine learning in an iterative search over various model configurations. The test set performance indicates that a standard multilayer perceptron (MLP) can classify the CAN data samples with 73% accuracy. The same results also indicate that a deep convolutional neural network (CNN) can classify the samples at a much greater performance level (up to 100% accuracy for some individual vehicles). Clearly, one can determine which vehicle generated a given sample of CAN data. This erodes consumer safety: a sophisticated attacker who establishes a presence on an unknown vehicle can use similar techniques to identify the vehicle and better format attacks tailored for that vehicle.

**Keywords:** classification, deep learning, controller area network, CAN security, fingerprinting vehicles

**1. Introduction**

The CAN bus, which connects a large number of the devices in a modern vehicle (including vital systems like the brakes, steering wheel, and transmission), is vulnerable. It is vulnerable to cyberattacks capable of altering, preventing, or otherwise modifying the operator’s desired behavior. This research presents a new vulnerability: the packets that broadcast on a vehicle’s CAN bus uniquely identify the vehicle. This means that an attacker can use one of the tools presented here to construct a database of known CAN packet formats. Such a database allows the attacker to strengthen an attack on a desired vehicle and thus risks the passengers’ privacy and safety.

We illustrate this vulnerability by building, tuning, and evaluating two deep learning models. We train the models by giving as input CAN data samples and each sample’s generating vehicle, and we test these models by giving new samples as input and comparing the predicted vehicle to the actual vehicle. This research seeks to determine which vehicle generated each test sample.

Data comes from one primary source and consists of 230 megabytes of raw CAN data captured from 11 distinct vehicles. By formatting and partitioning the available data, we generate two disparate datasets. The first is composed of all available 1,024-byte CAN samples; the second contains samples from the first such that the classes are evenly represented. Each sample’s label is the ID of the vehicle that generated it.

We then feed these samples into the deep learning models. The first, an MLP, is simple enough that an attacker only somewhat familiar with deep learning tools can easily implement it and use it to classify CAN samples with some level of success. The second, a more complex CNN, achieves much better classification performance than does the MLP.

Overall, results indicate that one can use one of these deep learning methods to determine which vehicle generated a given CAN data sample. This risks vehicle operator and passenger safety because it gives bad cyber actors another tool to correctly structure malicious CAN packets.

The remainder of this report presents the research in detail. Section 2 examines some of the related work in current literature and explains why this work is insufficient for the research at hand. Section 3 describes the data, the deep learning model, the model-fitting process, and the model analysis and evaluation tools. Section 4 presents and discusses the results obtained. Section 5 explains the implication of these results and suggests a possible opportunity for future research.

**2. Background**

To better understand this research, we present basic information about the CAN protocol, deep learning, and related research. Section 2.1 discusses the CAN protocol, including its message structure and contents. Section 2.2 discusses time series classification approaches in deep learning and how they relate to this research. Section 2.3 examines related work in CAN analysis and in fingerprinting vehicles, both with and without deep learning.

**2.1 The Controller Area Network protocol**

The CAN protocol is a message broadcast system developed for automobile applications by Bosch in the early 1980s. CAN broadcasts short messages detailing the vehicle’s functionality (e.g., wheel speed, engine temperature, velocity); this ensures data consistency throughout the vehicle. As Figure 1 shows, this also reduces required wiring (National Instruments, 2019).



**Figure 1:** Vehicle network wiring with and without CAN

CAN messages are limited in capacity. Each one contains message overhead and up to 64 bits of data (Robert Bosch GmbH, 1991; Corrigan and Texas Instruments, 2008). Additionally, CAN components may not possess significant computing power. For this reason, automobile manufacturers typically employ *security through obscurity*, in which the exact data contents are obfuscated to prevent significant analysis. For example, it is difficult for a non-insider to learn what information is contained in messages with a specific arbitration ID, even after significant reverse engineering efforts (Buttigieg, Farrugia, and Meli, 2017; Stone et al., 2018).

These obfuscation processes do not mask the vehicle itself because CAN messages from one vehicle are uniquely identifiable. As this research shows, an attacker can devise a system capable of distinguishing CAN messages from different vehicles, even with limited information

**2.2 Deep learning**

The MLP is a feedforward neural network, which means that information flows from the input layer, through each hidden layer, and to the output layer (Goodfellow, Bengio, and Courville, 2016). MLPs are often called vanilla neural networks because they are simple, no-frills networks; one can use an MLP to approximate a nonlinear function using few hidden layers (Burkov, 2019).

In deep learning, CNNs “are a specialized kind of neural network for processing data that has a known grid-like topology”; thus, these networks are well-suited for processing spatially related data such as images or time-series, which are effectively one-dimensional grids (Goodfellow, Bengio, and Courville, 2016; Box, Jenkins, and Reinsel, 1994). By concatenating all sequential CAN messages with the same arbitration ID, we obtain multiple time series. We then tune a CNN to demonstrate its superior performance over the MLP.

**2.3 Related work**

As discussed in Box, Jenkins, and Reinsel (1994), most researchers concerned with time series data attempt to predict future values, especially those researchers focused on CAN security. For example, Marchetti and Stabili (2017) devise an intrusion detection system (IDS) by analyzing sequential CAN IDs, and Tyree et al. (2018) create an IDS that learns relationships in different time series signals. However, some researchers in recent years have used deep learning to classify CAN data. For example, Kang and Kang (2016) train a deep neural network to classify CAN data packets as either *normal* or *abnormal*. The literature survey in Kwon et al. (2017) details more examples of deep learning as applied to IDSs and to CAN security.

Recent research shows that one can certainly distinguish the time series found in reverse-engineered CAN data. Enev et al. (2016), for example, demonstrate that machine learning can discriminate between different drivers using just these time series identified in a single vehicle, but this research does not claim to distinguish the raw CAN data generated by different vehicles.

Previous research efforts have demonstrated the ability to a) classify CAN messages as real or fake, b) manually determine the vehicle by analyzing time series in reverse-engineered CAN data, or c) predict future values of those time series. None of these efforts demonstrate an ability to classify vehicles using raw CAN messages. This is an important gap in current research, and our research seeks to address this gap.

In Section 3, we detail the experimental methodology, including the data’s origins and structure and the data formatting process. Additionally, we describe the deep learning models (i.e., their architectures and hyperparameters) and the model evaluation strategy.

**3. Methodology**

We train two different deep learning models – one an MLP and the other a CNN – to determine how well a model can distinguish between CAN data samples from different vehicles. Every sample in the dataset contains a vehicle ID and 1,024 sequential bytes of CAN data. Input to the models consists of the data bytes; the model then predicts the vehicle ID and compares its output to the true vehicle ID.

The remainder of this section discusses the data in detail: its nature and origins, the cleanup and formatting process, and the classes and their distributions. Additionally, this section describes the deep learning models employed, including the architectures and the model-fitting and evaluation processes.

**3.1 Data**

This research uses data 230 megabytes of CAN data from Stone et al. (2018). The researchers conducted one capture on each of 11 different vehicles. Table 1 displays metadata for the vehicles in the dataset.

**Table 1:** Make, model, and year for the eleven vehicles

|  |  |  |  |
| --- | --- | --- | --- |
| Vehicle | Make | Model | Year |
| 1 | Chevrolet | Cobalt | 2009 |
| 2 | Chevrolet | Silverado | 2011 |
| 3 | Dodge | 1500 | 2014 |
| 4 | Ford | F-150 | 2017 |
| 5 | Ford | Focus | 2010 |
| 6 | Honda | Accord | 2012 |
| 7 | Honda | Accord | 2015 |
| 8 | Nissan | 370Z | 2015 |
| 9 | Nissan | XTERRA | 2010 |
| 10 | Saab | 9-7X | 2009 |
| 11 | Toyota | Corolla | 2009 |

A large comma-separated values (CSV) file stores the data. Each of the 4,161,755 rows in this file represents one CAN message, and each row contains:

* A capture\_id and a vehicle\_id, which identify the message’s origin;
* A timestamp relative to the start of the capture;
* An arbitration\_id, which serves as the vehicle’s identifier of the message’s contents;
* A dlc, or *data length code*, which indicates how many bytes of data the message contains;
* The hexadecimal data itself; and
* The vehicle’s make, model, and year.

The messages in this CSV file are not immediately suitable for deep learning. Each message contains no more than eight data bytes (and many contain fewer), and it is extremely unlikely that so little information can uniquely identify a vehicle. To address this issue, we build a set of data samples, where each sample contains 1,024 bytes of sequential CAN data from one capture. These samples contain solely the message data – we strip all other information (e.g., vehicle\_id) to ensure sufficient problem difficulty. Specifically, for each capture, we sort the messages by timestamp before splitting all hex data into a list of hex bytes, converting the hex bytes into three-digit integers, and dividing the list into samples of 1,024 integers. Because each integer represents one byte of CAN data, each sample contains individual data bits. We then write each sample and the associated vehicle ID to a new CSV file. This makes it more likely that each sample contains sufficient information and that the underlying CAN data structure is preserved. Table 2 presents a few examples of these data samples.

**Table 2:** Examples of CAN data samples

|  |  |
| --- | --- |
| Vehicle | Data |
| 3 | 074 166 111 254 255 240 254 … |
| 7 | 003 212 003 209 003 199 003 … |
| 8 | 255 248 000 128 015 254 030 … |

The new CSV file contains 28,178 samples from the 11 vehicles. Table 3 illustrates the distributions of samples over all vehicles (the number of samples depends primarily on the length of the data capture). In a balanced dataset, each vehicle would contain about nine percent of all samples. Clearly, this dataset is not balanced. As described in Section 3.3, we address this class imbalance during model training.

**Table 3:** Number of arbitration IDs and samples per vehicle

|  |  |  |
| --- | --- | --- |
| Vehicle | Samples | Proportion |
| 1 | 4102 | 14.56 % |
| 2 | 1824 | 6.47 % |
| 3 | 1756 | 6.23 % |
| 4 | 2182 | 7.74 % |
| 5 | 3791 | 13.45 % |
| 6 | 1695 | 6.02 % |
| 7 | 2198 | 7.80 % |
| 8 | 3020 | 10.72 % |
| 9 | 2553 | 9.06 % |
| 10 | 2974 | 10.55 % |
| 11 | 2083 | 7.39 % |

**3.2 Model architecture**

To demonstrate that a given vehicle’s CAN data uniquely identifies the vehicle, we train and compare the deep learning models on multiple data samples. As described in Section 3.1, each sample contains sequential CAN data from one data capture, and each is labeled with one of 11 vehicles.

Using an MLP to classify vehicles is a naive approach, but it is useful to present the naive approach’s performance to demonstrate that an attacker does not need complex methods to adequately classify vehicles. Each CAN sample contains 1,024 data bytes, so the MLP receives 8,192 bits as input and outputs one of the 11 classes. Section 3.2.1 describes the MLP in detail. We also present a more complex approach – a CNN – to demonstrate that a sophisticated attacker can achieve superior performance over the naive approach. The CNN utilizes one-dimensional convolutional layers because each sample contains sequential, one-dimensional data. In testing, the CNN receives one sample as input and outputs one of the 11 classes. Section 3.2.2 presents the details of the model.

*3.2.1 The multilayer perceptron*

We test multiple hyperparameter configurations to identify the best MLP; the left panel of Figure 2 displays this model. It has one hidden layer of 512 nodes, and it uses a relu activation function. The dropout layer sets 20% of its inputs to zero. The output layer contains 11 nodes – one for each vehicle in the dataset – and uses a softmax activation function. We compile the model with an Adam optimizer of learning rate and a categorical cross-entropy loss function. The model has 530,443 parameters, all of which are trainable. Every MLP implementation utilizes this basic architecture. In total, we test 27 MLP architectures; Table 4 lists the various options for each hyperparameter used during model construction.

A screenshot of text

Description automatically generated

**Figure 2:** The best architectures identified in iterative model evaluation processes. *Left:* MLP; *Right:* CNN.

**Table 4:** Hyperparameter options for MLP construction

|  |  |
| --- | --- |
| Hyperparameter | Options |
| Number of hidden layers | 1, 2, 3 |
| Number of nodes per hidden layer | 128, 256, 512 |
| Optimizer learning rate | 0.01, 0.001, 0.0001 |

*3.2.2 The convolutional neural network*

As with the MLP, we test multiple hyperparameter configurations before selecting the best CNN. The best model utilizes a typical CNN structure consisting of convolutional, pooling, and dropout layers. The right panel of Figure 2 displays the specific architecture. The convolutional layers each use 32 filters with a kernel size of four, the pooling layer pools over four elements at each step, and the dropout layer sets 20% of its inputs to zero. The convolutional layers and the first dense layer all use a relu activation function; the final dense layer uses softmax. We again compile the model with an Adam optimizer of learning rate and a categorical cross-entropy loss function. In total, the model has 818,427 parameters, and all but 64 are trainable. Every CNN implementation utilizes this basic architecture. In total, we test 18 CNN architectures; Table 5 lists the various options for each hyperparameter used during model construction.

**Table 5:** Hyperparameter options for CNN construction

|  |  |
| --- | --- |
| Hyperparameter | Options |
| Filters per convolutional layer | 32, 64 |
| Kernel size per filter | 2, 3, 4 |
| Pooling size | 2, 3, 4 |

**3.3 Model fitting**

Table 3 shows that the classes in the full dataset are somewhat imbalanced. For example, nearly 15% of all samples come from Vehicle 1; one should expect each vehicle to contain about 9% of the samples. To address this imbalance, we use scikit-learn’s class\_weight module during model training; this tool adjusts weights to account for the frequency of each class, and thus it ensures that both models can use all training data without unnecessarily suffering from class imbalance. Still, we also create a second, balanced dataset by randomly sampling samples from every vehicle.

Another important tool during model training is Keras’s EarlyStopping callback, which limits overfitting. This function monitors validation loss during training and terminates the training if the loss does not improve in 10 epochs. The callback then restores the model to its best version. This tool also limits model training time.

To properly train and evaluate the models, we split each dataset into three sets: 60% of the samples are used in training, 20% in validation, and 20% in testing. Scikit-learn’s train\_test\_split enables this split by randomly sampling from the dataset. We train each model on the training set, terminate training with the early stopping callback (which monitors the validation set’s loss), and evaluate the trained model on the testing set. However, these tools do not guarantee optimality, so we also iteratively improve the models through repeated testing and hyperparameter tuning. A search over a set of possible hyperparameter configurations ensures we can identify the best classifier. In total, we evaluate 27 different MLPs and 18 different CNNs. Tables 4 and 5 depict the hyperparameter options for these models.

Figure 3 shows that the final models possess sufficient capacity for this task. The validation loss and accuracy for each model are nearly as good as the training loss and accuracy, and all four losses and accuracies appear to reach an asymptote. This means that additional training will not benefit the models, and it also means that the models are nearly optimal for this task. Note that the MLP struggles to generalize to new data.

A close up of a map

Description automatically generated

**Figure 3:** Training and validation losses and accuracies over time for the best models. *Top left:* the MLP’s loss; *Top right:* the MLP’s accuracy; *Bottom left:* the CNN’s loss; *Bottom right:* the CNN’s accuracy.

**3.4 Model evaluation and analysis**

We present results for the MLP to demonstrate that an intruder with a low-level understanding of deep learning could use this field to better format CAN attacks. We compare the MLP’s performance to that of a CNN to show that a sophisticated intruder can use more advanced techniques to achieve much better results.

To evaluate model performance, we compute balanced accuracy for both the balanced and the imbalanced dataset. This metric takes into consideration the various class distributions when computing the classification accuracy. Even for the balanced dataset, this is important. Because we randomly sample from the dataset to build the training, validation, and testing sets, each of those three sets could be imbalanced.

Additionally, we present confusion matrices to allow for an in-depth analysis of where each model fails. For example, Manufacturer X could use the exact same CAN configuration for different vehicles, so even a well-trained model could fail to distinguish between Manufacturer X’s vehicles. These analyses may even imply a similar CAN structure across different manufacturers, so the results of these investigations guide further model modifications and tuning.

The remainder of this paper details the results of the experiment. Section 4 presents and discusses the results and the implied performance of the two models. Section 5 describes the implications of these results and suggests a few areas for future research.

**4. Test set results**

This section presents the test set classification results for the two model types on the two datasets. Results indicate that both the MLP and the CNN can adequately classify CAN data segments, but the CNN is significantly better than the MLP on both the imbalanced dataset and the balanced dataset. Results also indicate that both models perform better on the larger, imbalanced dataset; this is simply due to the availability of training data in each set. Table 6 presents the overall balanced accuracies.

**Table 6:** Balanced accuracy for each model on each dataset

|  |  |  |
| --- | --- | --- |
| Model | Dataset | Accuracy |
| Multilayer perceptron | Imbalanced | 73.03 % |
| Multilayer perceptron | Balanced | 67.90 % |
| Convolutional neural network | Imbalanced | 99.79 % |
| Convolutional neural network | Balanced | 98.23 % |

Table 7 presents class-specific accuracy values for the four trained models. Clearly, the CNN trained on the imbalanced dataset performs better than the other models on every class (except Vehicle 5, for which both the CNN and the MLP achieve 100% accuracy). Additionally, the CNN trained on the balanced dataset outperforms either version of the MLP on most (10/11) of the classes. This is further evidence of the superior performance of the CNN, and it also indicates that more training data is better, even if the classes in the data are imbalanced.

**Table 7:** Class accuracy for each model on each dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vehicle | MLP, Imbalanced | MLP, Balanced | CNN, Imbalanced | CNN, Balanced |
| 1 | 85.87 % | 79.46 % | 100.00 % | 98.32 % |
| 2 | 42.37 % | 31.21 % | 99.74 % | 97.02 % |
| 3 | 61.65 % | 52.35 % | 100.00 % | 98.96 % |
| 4 | 91.44 % | 78.85 % | 100.00 % | 98.81 % |
| 5 | 100.00 % | 90.41 % | 100.00 % | 98.47 % |
| 6 | 62.50 % | 75.63 % | 99.14 % | 95.77 % |
| 7 | 57.48 % | 39.91 % | 99.34 % | 97.27 % |
| 8 | 57.67 % | 63.81 % | 99.80 % | 98.45 % |
| 9 | 69.12 % | 79.44 % | 99.83 % | 99.10 % |
| 10 | 98.97 % | 93.84 % | 100.00 % | 99.42 % |
| 11 | 80.74 % | 69.48 % | 99.76 % | 98.33 % |

Figure 4 presents the multiclass confusion matrices for both model types on both datasets. Generally, one can see that the models tend to correctly classify CAN segments, but some specific misclassifications are certainly common. For example, all four trained models misclassify Vehicle 6 as Vehicle 7 (and vice versa) to some extent; these are different-year Honda Accords. One can also see that the models usually perform similarly on the same vehicle. For example, all models demonstrate strong performance on Vehicle 10 – perhaps Saab employed a distinct data structure when designing its 2009 9-7X. Conversely, neither of the MLP models achieve strong performance on Vehicle 2. One can easily discern other trends by further analyzing Figure 4.

A screenshot of a cell phone

Description automatically generated

**Figure 4:** Confusion matrices. *Top left:* MLP on the imbalanced dataset; *Top right:* MLP on the balanced dataset; *Bottom left:* CNN on the imbalanced dataset; *Bottom right:* CNN on the balanced dataset.

Section 5 discusses the real-world implications of these results. Additionally, the section offers areas for future research in CAN analysis and security.

**5. Conclusion**

This research has two primary conclusions:

* A vehicle’s CAN data uniquely identifies said vehicle; and
* An attacker can use one of the tools presented to improve attacks on vehicle-based CANs.

Clearly, one can fingerprint vehicles with CAN data samples. The best classifier in our research uses a CNN trained on an imbalanced dataset, and it achieves a balanced classification accuracy of 99.79% on the test set. This performance indicates that deep learning methods can definitively fingerprint a vehicle using only its raw CAN data. This means a malicious intruder can employ deep learning for personal gain. Such an attacker can use a well-tuned CNN (or an MLP if resources or knowledge are limited) to build a database of known vehicles; the attacker can then reference this database when intruding on a new CAN bus to better structure the desired attack. This presents a risk to today’s vehicles.

Devising a Siamese neural network (SNN) capable of learning the difference between CAN segments is the logical next step for this research. Given enough training data, an SNN can distinguish new cars – that is, those not present in the original training set – because it learns during training whether two observations come from the same source or from different sources. Standard classifiers (e.g., MLPs and CNNs), on the other hand, must learn a fixed set of classes; when presented with an observation from a new class, a standard classifier must classify it as one of the classes in the original training set. For this reason, an SNN is extensible: one can train an SNN on a set of vehicles and, because it knows why samples from two vehicles are different, one can introduce many new vehicles and still maintain solid performance. This is a boon for the malicious intruder: such an attacker need not collect CAN segments from every potential vehicle to effectively attack new vehicles.

Overall, our findings indicate a significant CAN vulnerability that an attacker familiar with deep learning can easily exploit. A bad actor who gains access to an unknown vehicle’s CAN bus can employ deep learning to identify the vehicle – or at least determine that the vehicle is not one of the known vehicles – and improve the desired attack. This breach of privacy risks operator and passenger safety, and it warrants action on the part of automobile manufacturers. It also warrants further research into other deep learning applications on the CAN. However, the acts of accessing the increasingly restricted CAN bus, of collecting significant quantities of data, and of building, training, and tuning effective deep learning models serve as nontrivial barriers to the determined cyber attacker.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense, or the United States Government. This work is approved for public release under case number 88ABW-2019-4730.

**References**

Box, G.E., Jenkins, G.M., and Reinsel, G.C. (1994) *Time Series Analysis: Forecasting and Control*. Prentice-Hall, Inc., Upper Saddle River, New Jersey.

Burkov, A., (2019) *The Hundred-Page Machine Learning Book*, Andriy Burkov.

Buttigieg, R., Farrugia, M. and Meli, C. (2017) “Security Issues in Controller Area Networks in Automobiles”, *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering*, pp 93-98.

Corrigan, S. and Texas Instruments, (2008) “Introduction to the Controller Area Network (CAN)”, *SLOA101A*, Dallas, TX.

Enev, M., Takakuwa, A., Koscher, K. and Kohno, T. (2016) “Automobile Driver Fingerprinting”, *Proceedings on Privacy Enhancing Technologies*, Vol. 2016, No. 1, pp 34-50.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*, The MIT Press, Cambridge, Massachussetts.

Kang, M.J. and Kang, J.W. (2016) “Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security”, *PLOS ONE*, Vol. 11, No. 6, pp 1-17.

Kwon, D., Kim, H., Kim, J., Suh, S.C., Kim, I. and Kim, K.J. (2017) “A Survey of Deep Learning-Based Network Anomaly Detection”, *Cluster Computing*, Vol. 22, No. 1, pp 949-961.

Marchetti, M. and Stabili, D. (2017) “Anomaly Detection of CAN Bus Messages Through Analysis of ID Sequences”, *2017 IEEE Intelligent Vehicles Symposium*, pp 1577-1583.

National Instruments (2019) *Controller Area Network (CAN) Overview*, [online], National Instruments, <https://www.ni.com/en-us/innovations/white-papers/06/controller-area-network--can--overview.html>.

Robert Bosch GmbH, (1991) *CAN Specification Version 2.0*, Robert Bosch GmbH, Stuttgart, Germany.

Stone, B., Graham, S., Mullins, B., and Schubert Kabban, C. (2018) *Enabling Auditing and Intrusion Detection for Proprietary Controller Area Networks*, PhD, Dissertation, Air Force Institute of Technology.

Tyree, Z., Bridges, R.A., Combs, F.L. and Moore, M.R. (2018) “Exploiting the Shape of CAN Data for In-Vehicle Intrusion Detection”, *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp 1-5.