can-sleuth: Investigating and Evaluating Automotive Intrusion Detection Datasets

Kidmose, Brooke; Meng, Weizhi

Published in:
Proceedings of the EICC 2024: European Interdisciplinary Cybersecurity Conference

Link to article, DOI:
10.1145/3655693.3655696

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

can-sleuth: Investigating and Evaluating Automotive Intrusion Detection Datasets

Brooke Kidmose
blam@dtu.dk
Technical University of Denmark
Kongens Lyngby, Denmark

Weizhi Meng
weme@dtu.dk
Technical University of Denmark
Kongens Lyngby, Denmark

ABSTRACT

The modern automobile is a network—specifically, a controller area network (CAN)—of computers. Automotive computers manage the engine (e.g., fuel injection), the transmission (e.g., automatic shifting), the vehicle speed (e.g., cruise control), and many, many more systems. Therefore, a vehicle's CAN bus is safety critical; by design, it is robust, reliable, and error tolerant. Unfortunately, it is not secure; it was developed in the 1980s, and, at that time, it was a closed system—no Internet access. The modern automobile is *not* a closed system, yet the CAN bus remains insecure. Automotive researchers are gravitating toward intrusion detection as one possible solution to the problem of automotive [in]security. To build and evaluate an intrusion detection system (IDS), however, researchers need adequate training and testing data.

In this paper, we investigate and evaluate the following automotive intrusion detection datasets: (1) the HCRL Car Hacking dataset, (2) the HCRL Survival Analysis dataset, and (3) the `can-train-and-test` dataset. The HCRL Car Hacking dataset (`hcrl-ch`) and the HCRL Survival Analysis dataset (`hcrl-sa`) are well-established in the literature, whereas the `can-train-and-test` dataset is a promising new dataset. First, we investigate the `can-train-and-test` dataset—in particular, we evaluate the impacts of various features on the performance of sixteen machine learning IDSs. Second, we compare `can-train-and-test` to `hcrl-ch` and `hcrl-sa`. We find that, compared to the two established datasets, `can-train-and-test` provides new and greater insights to researchers interested in automotive intrusion detection, automotive firewalls & filtering, and more. With an order of magnitude more training and testing data, `can-train-and-test` enables the data-intensive machine learning models to demonstrate their full potential, with eight of the sixteen models achieving an average F1-score above 0.9. Moreover, `can-train-and-test` maintains ample differentiation power; the standard deviation of the models' average F1-scores was 0.2247, which exceeds the standard deviations of `hcrl-ch` (0.2202) and `hcrl-sa` (0.2243).

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; *Embedded systems security*; *Network security*; • **Computing methodologies → Machine learning**.

EICC 2024, June 05–06, 2024, Xanthi, Greece

ACM ISBN 979-8-4007-1651-5/24/06
https://doi.org/10.1145/3655693.3655696

KEYWORDS

automotive, vehicle, controller area network, in-vehicle network, intrusion detection system, can-train-and-test dataset, HCRL Car Hacking dataset, HCRL Survival Analysis dataset, machine learning

ACM Reference Format:
Brooke Kidmose and Weizhi Meng. 2024. `can-sleuth`: Investigating and Evaluating Automotive Intrusion Detection Datasets. In *European Interdisciplinary Cybersecurity Conference (EICC 2024), June 05–06, 2024, Xanthi, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3655693.3655696

1 INTRODUCTION

To secure automotive networks—both *intra*-vehicle networks and *inter*-vehicle networks—we look to the controller area network (CAN) bus. If a vehicle's CAN bus is vulnerable, then any vehicle (and any infrastructure) connected to that vehicle is vulnerable. Therefore, to secure the automotive CAN bus (and the automotive technologies it touches), researchers have explored strategies such as intrusion detection [11, 12], firewalls & filtering [15, 16], honeypots [16, 22], dynamic arbitration [5, 11], and cryptography [1, 11, 21]—to name a few. Developing and implementing automotive security technologies generally requires intimate knowledge of the CAN bus and millions CAN frames that cross the bus every few minutes [10]. Moreover, automotive security technologies that involve machine learning generally require copious amounts of CAN traffic data for training and testing.

In 2018, the Hacking and Countermeasure Research Lab (HCRL) published the HCRL Car Hacking dataset and the Survival Analysis dataset [3, 19, 20], which became extremely popular with automotive security researchers. Lampe and Meng identified several limitations of existing open access datasets; as such, they curated the `can-train-and-test` dataset to better meet the needs of automotive security researchers [6, 8, 13, 14]. All of the above datasets are freely available online (see details and links in Section 2).

This paper constitutes a deep dive into the HCRL Car Hacking dataset (`hcrl-ch`), the HCRL Survival Analysis dataset (`hcrl-sa`), and `can-train-and-test` dataset. By conducting experiments, we "sleuth out" important characteristics of our impact data—e.g., the presence (or absence) of the `timestamp` feature, the subdivision (or lack thereof) of the `data field` feature. Moreover, we show that, compared to the HCRL datasets, `can-train-and-test` provides much more training and testing data (as well as specialized scenarios) while maintaining equivalent (if not greater) differentiation power.

Research Questions:

- **RQ1:** How does the `timestamp` feature impact the performance of different machine learning models?

- **RQ2:** How does subdividing the `data field` feature into eight bytes impact the performance of different machine learning models?
- **RQ3:** How does the newly-published `can-train-and-test` dataset compare to the well-established `hcrl-ch` and `hcrl-sa` datasets?

We developed our research questions in an effort to address existing gaps in the automotive intrusion detection literature. Existing works have experimented with different combinations of features and different machine learning models; however, few studies analyze a *large* number of *diverse* machine learning models. We evaluated sixteen different machine learning models, including both traditional machine learning and deep learning as well as both supervised and unsupervised approaches. In addition, we sought to analyze the newly-published `can-train-and-test` through the lens of comparison—an analysis which had not yet been conducted.

To answer our research questions, we designed and executed a number of experiments. To address **RQ1**, we conducted a series of experiments *with* the `timestamp` feature, followed by a series of experiments *without* the `timestamp` feature. Then, we collected, aggregated, and analyzed the results. Similarly, to assess the impact of subdividing the `data field` feature—**RQ2**—we conducted a series of experiments *with* subdivision, followed by a series of experiments *without* subdivision. Finally, we evaluated all of our machine learning models against `hcrl-ch`, `hcrl-sa`, and `can-train-and-test`.

Contributions: Our contributions are as follows:

(1) We tailor and apply sixteen distinct machine learning models to answer research questions **RQ1**, **RQ2**, and **RQ3**.
(2) Using our machine learning models, we evaluate the impacts of (1) including or excluding the `timestamp` feature and (2) subdividing or not subdividing the `data field` feature. By analyzing the performance of each machine learning model during each scenario, we determine the ideal scenario for each model.
(3) We compare the newly-published `can-train-and-test` dataset to two well-established open-access datasets—the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset. We demonstrate that the `can-train-and-test` dataset contains novel features not available in either of the HCRL datasets, and we highlight a number of unique insights the `can-train-and-test` can provide.

The remainder of this paper is organized as follows: Section 2 summarizes pertinent background information and related work. In Section 3, we detail our methodology; then, in Section 4, we present and discuss our results. In Section 5, we address limitations of our research as well as opportunities for future work. Section 6 concludes our work.

2 BACKGROUND & RELATED WORK

HCRL Car-Hacking dataset (`hcrl-ch`). Collected and curated by the Hacking and Countermeasure Research Lab (HCRL), the `hcrl-ch` dataset [19, 20] contains attack-free CAN traffic as well as four types of attacks:

(1) Denial of Service
(2) Fuzzing
(3) RPM Spoofing
(4) Gear Spoofing

All traffic was collected from a Hyundai YF Sonata, and all attacks were real, not simulated. According to Rajapaksha et al. [18], when it comes to automotive intrusion detection—and, in particular, the CAN bus—the HCRL Car-Hacking dataset is the most widely used. Researchers leverage `hcrl-ch` to develop, optimize, and evaluate automotive IDSs.

Link: ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

HCRL Survival Analysis dataset (`hcrl-sa`). The `hcrl-sa` dataset [3], also developed by the Hacking and Countermeasure Research Lab, contains data from three different vehicles—a Chevrolet Spark, a Hyundai YF Sonata, and a Kia Soul. In addition to attack-free CAN traffic, the dataset provides three types of attacks:

(1) Flooding (i.e., Denial of Service)
(2) Fuzzing
(3) Malfunction (i.e., Spoofing)

Link: ocslab.hksecurity.net/Datasets/survival-ids

`can-train-and-test`. Collected and curated by Lampe and Meng, the `can-train-and-test` dataset [6, 8, 13, 14] contains attack-free CAN traffic as well as nine unique attacks:

(1) Denial of Service
(2) Fuzzing
(3) Systematic
(4) Gear Spoofing
(5) RPM Spoofing
(6) Speed Spoofing
(7) Combined Spoofing
(8) Standstill
(9) Interval

Traffic was collected from four different vehicles—a Chevrolet Impala, a Chevrolet Traverse, a Chevrolet Silverado, and a Subaru Forester. Many attacks were conducted against a live vehicle traveling at speed; due to safety considerations, some attacks were simulated.

According to Lampe and Meng, the `can-train-and-test` dataset was designed to help researchers evaluate a proposed IDS under specific conditions (e.g., evaluate the IDS against a known vehicle and an unknown attack) [8, 13]. Within each sub-dataset (#1 - #4), there are two training subsets and four testing subsets, as follows:

- `train_01_attack_free`: Train the IDS (attack-free traffic)
- `train_02_with_attacks`: Train the IDS (attack-free and attack traffic)
- `test_01_known_vehicle_known_attack`: Evaluate the IDS against a known vehicle type and a known attack.
- `test_02_unknown_vehicle_known_attack`: Evaluate the IDS against an unknown vehicle type and a known attack.
- `test_03_known_vehicle_unknown_attack`: Evaluate the IDS against a known vehicle type and an unknown attack.
- `test_04_unknown_vehicle_unknown_attack`: Evaluate the IDS against an unknown vehicle type and an unknown attack.

Table 1: HCRL Car Hacking Dataset (hcrl-ch)[1]

Set	# of Training Samples	# of Testing Samples	Total # of Samples
Attack-free	N/A	N/A	988,872
DoS	2,665,774	1,000,000	3,665,772
Fuzzing	2,838,863	1,000,000	3,838,861
Gear Spoofing	3,443,145	1,000,000	4,443,143
RPM Spoofing	3,621,705	1,000,000	4,621,703

[1]We partitioned each attack capture into a training subset and a testing subset. We saved 1,000,000 CAN frames for the evaluation—approximately 25%.

Table 2: can-train-and-test[1]

Set	# of Training Samples	# of Testing Samples	Total # of Samples
set_01	10,653,152	34,006,457	44,659,609
set_02	17,340,826	38,605,982	55,946,808
set_03	12,025,781	31,774,043	43,799,824
set_04	9,492,819	39,342,021	48,834,840

[1]We exclude two testing subsets—test_05_suppress and test_06_masquerade—from the values in the table. No equivalent testing subsets are available in hcrl-ch and hcrl-sa; therefore, we omit them from our comparative evaluation in Section 4.

Table 3: HCRL Survival Analysis Dataset (hcrl-sa)[1]

train	test_01	test_02	test_03	test_04
627,264	399,539	186,237	306,089	79,788

[1]We curated hcrl-sa to match can-train-and-test. hcrl-sa is partitioned into one training subset and four testing subsets (e.g., test_01_known_vehicle_known_attack), similar to can-train-and-test. We were able to include all of the files in the original HCRL Survival Analysis Dataset except the attack-free file for the Chevrolet Spark. We selected the Chevrolet Spark to serve as the "unknown" vehicle; therefore, we could not include its attack-free traffic in the training data. The unused file contains 136,935 lines.

Table 4: can-train-and-test (sub-dataset #1)

train	test_01	test_02	test_03	test_04
10,653,152	5,702,678	6,447,925	8,635,287	13,220,567

The can-train-and-test dataset has since been extended with two additional testing subsets, as follows:

- test_05_suppress: Test the IDS against a suppress attack.
- test_06_masquerade: Test the IDS against a masquerade attack.

The extended dataset, which includes the four original testing subsets as well as the suppress and masquerade subsets, is available in the can-train-and-test-v1.5 repository [7].

Tables 1 and 2 provide numerical breakdowns of hcrl-ch and can-train-and-test, respectively. Tables 3 and 4 enumerate the sizes of hcrl-sa and sub-dataset #1 of 2. In terms of CAN traffic samples, we can see that can-train-and-test is about an order of magnitude larger than either of the HCRL datasets; as such, it is better equipped to showcase the capabilities of machine learning—especially deep learning—IDSs, which notoriously require a lot of training data.

Tables 2 and 4 also reveal that the can-train-and-test dataset contains a high volume of testing data relative to training data—and relative to the traditional 70/30 to 80/20 range of train/test splits [2]. In part, the relatively high volume of testing data arises from the four independent testing subsets contained in each sub-dataset. However, Table 4 calls attention to the fact that some testing subsets are nearly as large—or even larger—than the training subset. Given an abundance of data, it is beneficial to evaluate a would-be intrusion detection system against a high volume of data. In the real world, an IDS would be continuously challenged by an unending stream of data. A large testing subset is more realistic than a small testing subset but is still more practicable than a never-ending real-world test.

Link (original): bitbucket.org/brooke-lampe/can-train-and-test/

Link (extended): bitbucket.org/brooke-lampe/can-train-and-test-v1.5/

2.1 Class Imbalance

(1) **hcrl-ch**
- train_02_with_attacks **7:1**
- test_01_DoS **3:1**
- test_02_fuzzing **4:1**
- test_03_gear_spoofing **4:1**
- test_04_rpm_spoofing **4:1**

(2) **hcrl-sa**
- train_02_with_attacks **11:1**
- test_01_known_vehicle_known_attack **4:1**
- test_02_unknown_vehicle_known_attack **5:1**
- test_03_known_vehicle_unknown_attack **12:1**
- test_04_unknown_vehicle_unknown_attack **8:1**

(3) **can-train-and-test** (sub-dataset #1)
- train_02_with_attacks **213:1**
- test_01_known_vehicle_known_attack **89:1**

- `test_02_unknown_vehicle_known_attack` **38:1**
- `test_03_known_vehicle_unknown_attack` **413:1**
- `test_04_unknown_vehicle_unknown_attack` **927:1**

As outlined above, `can-train-and-test` demonstrates extreme class imbalance—the ratio of attack-free samples to attack samples can be as high as 927:1. Extreme class imbalance is associated with a number of challenges, particularly when it comes to capturing performance metrics. However, extreme class imbalance is also realistic to the problem at hand; a million messages will cross the CAN bus within about five to ten minutes [10]. Since vehicles routinely travel the world's roads for months, years, and even decades without incident, we can see that class imbalance in the real world is even steeper. Therefore, `can-train-and-test` enables researchers to assess the capabilities of a proposed IDS in a more authentic scenario. If the proposed IDS has a problematic false positive rate, it will almost certainly be revealed when the ratio of attack-free samples to attack samples is 927:1.

3 METHODOLOGY

Our investigation centered around the `can-train-and-test-v1.5` dataset [7, 8, 13]. We extracted features from the dataset and evaluated the impact of said features on the performance of machine learning IDSs. Next, we pitted `can-train-and-test` against two well-established open-access intrusion detection datasets, namely, the HCRL Car-Hacking dataset [19, 20] and the HCRL Survival Analysis dataset [3].

For the `can-train-and-test` feature evaluation, we selected two features: (1) the `timestamp` feature and (2) the `data field` feature. With the `timestamp` feature, we sought to assess the value of the feature itself; therefore, we conducted one series of experiments that *in*cluded the timestamp, followed by one series of experiments that *ex*cluded the timestamp. The results are shown in Tables 5 and 6, respectively. With the `data field` feature, we sought to evaluate the impact of subdividing the data field into eight data bytes—each data byte treated as a separate feature. If subdividing the data field did not improve performance, then it was a needless preprocessing step. As such, we conducted one set of experiments in which the data field was subdivided into eight distinct feature bytes, followed by one set of experiments in which the data field remained intact as a single feature. The results are shown in Tables 5 and 7, respectively.

To compare `can-train-and-test` to `hcrl-ch` and `hcrl-sa`, we opted to evaluate all sixteen of our machine learning models against the three datasets, then we analyzed the results. To leverage our machine learning models against `hcrl-ch` and `hcrl-sa`, we had to conduct a few preprocessing steps. Essentially, we reformatted `hcrl-ch` and `hcrl-sa` as comma-separated values (CSV) files containing only the expected features in the expected format. For `hcrl-ch`, we removed the leading zero from the `arbitration identifier` feature and added a leading zero to the first byte of the `data field` feature. Then, we concatenated the eight data field bytes into a single feature (the machine learning models are responsible for subdividing—or not subdividing—the data field). Lastly, we removed the `data length code (DLC)` feature, and we converted the R/T flag to a 0/1 flag to match `can-train-and-test`. For `hcrl-sa`, we followed the same preprocessing steps, and then we changed the file extension from `.txt` to `.csv`.

The `can-train-and-test` dataset was curated to facilitate scenario-specific evaluation. As mentioned in Section 2, one scenario available in `can-train-and-test` is the known vehicle, unknown attack pairing. In this scenario, the vehicle has been seen in training, but the attack has not been seen in training. Because `can-train-and-test` was designed to fill this specialized role, it is not directly comparable to `hcrl-ch` and `hcrl-sa`. As such, we opted to conduct two different comparisons: (1) a conventional comparison to a conventionally-partitioned dataset, and a scenario-based comparison to a dataset that partitioned according to scenarios, similar to `can-train-and-test` itself. The data in the `hcrl-ch` dataset comes from exactly one vehicle, so it is not suited to our scenario-based comparison. However, three different vehicles—a Chevrolet Spark, a Hyundai YF Sonata, and a Kia Soul—are represented in the `hcrl-sa` dataset. In addition, `hcrl-sa` contains three different attacks: DoS, fuzzing, and malfunction. Thus, `hcrl-sa` can be configured as a scenario-based dataset—akin to `can-train-and-test`.

- For `hcrl-ch`, we divide each attack into traditional train/test splits (75% for training, %25 for testing). All of the attack-free data is used as training data. We use the train splits for training and the test splits for testing.
- For `hcrl-sa`, we select two of the three vehicles and two of the three attacks for training. These vehicles and attacks will be the *known* vehicles and *known* attacks. We withhold one vehicle and one attack; these will be the *unknown* vehicle and *unknown* attack. All of the data in the Survival Analysis dataset was included in the scenario-based version of the dataset—except the attack-free file for the *unknown* vehicle. In keeping with the `can-train-and-test`, we include the attack-free files in the training data only. Since the *unknown* vehicle cannot be seen in training, its attack-free file was not used.

4 DISCUSSION OF RESULTS

In Table 5, we provide the results of experiments run against the `can-train-and-test` dataset with default parameters—the `data field` feature is subdivided, and the `timestamp` feature is included. The results are averaged per model. Table 6 excludes the `timestamp` feature, while Table 7 eliminates the subdivision of the `data field` feature.

Reviewing Tables 5, 6, and 7, we can see that the impact of (1) excluding the `timestamp` feature or (2) subdividing the `data field` feature differed depending upon the model. The average F1-score for the multi-layer perceptron (MLP) model, *with* the timestamp and subdivision, was 0.9555. *Without* the timestamp, the average F1-score was 0.9115, and *without* the subdivision, the average F1-score was 0.9320. Looking at the Gaussian naive Bayes (NB) model, we see F1-scores of 0.9626, 0.9538, and 0.9775—*with* the timestamp and subdivision, *without* the timestamp, and *without* the subdivision, respectively. For the linear support vector machine (SVM), the F1-scores were 0.4756, 0.4772, and 0.3484—*with* the timestamp and subdivision, *without* the timestamp, and *without* the subdivision, respectively. For MLP, the highest average F1-score was achieved *with* the timestamp and subdivision. For Gaussian (NB), the highest

Table 5: `can-train-and-test`: Averages with default parameters [include subdivision, include timestamp, include testing subset #6].[1]

Model	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9498	0.9787	0.9498	0.9626	0.9550	0.0397
K-Nearest Neighbors	0.9320	0.9830	0.9320	0.9531	0.9361	0.0597
Linear Regression	0.3360	0.9810	0.3360	0.4756	0.3330	0.6700
Logistic Regression	0.9856	0.9827	0.9856	0.9825	0.9908	0.0039
Linear Support Vector Machine	0.9866	0.9785	0.9866	0.9823	0.9921	0.0023
Decision Tree	0.7027	0.9853	0.7027	0.7930	0.7033	0.2962
Extra Trees	0.8783	0.9837	0.8783	0.9181	0.8819	0.1144
Gradient Boosting	0.7986	0.9839	0.7986	0.8683	0.8006	0.1974
Isolation Forest	0.5448	0.9766	0.5448	0.6919	0.5457	0.4534
Random Forest	0.8936	0.9842	0.8936	0.9313	0.8971	0.0993
K-Means	0.4118	0.9796	0.4118	0.5523	0.4092	0.5933
Mini-Batch K-Means	0.5186	0.9778	0.5186	0.6504	0.5195	0.4796
BIRCH	0.4823	0.9783	0.4823	0.5430	0.4831	0.5160
Local Outlier Factor	0.3621	0.9722	0.3621	0.4835	0.3609	0.6403
Multi-Layer Perceptron	0.9343	0.9839	0.9343	0.9555	0.9383	0.0576
Bernoulli Restricted Boltzmann Machine	0.9904	0.9811	0.9904	0.9857	0.9952	0.0000

[1]Testing subset #5 was not averaged into the performance metrics in this table, as the nature of the suppress attack (an attack with no labeled attack CAN frames) results in uninformative performance metrics.

Table 6: `can-train-and-test`: Average performance metrics when the timestamp is excluded [include subdivision, EXCLUDE timestamp, include testing subset #6].[1]

Model	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9346	0.9809	0.9346	0.9538	0.9397	0.0552
K-Nearest Neighbors	0.8753	0.9862	0.8753	0.9158	0.8780	0.1194
Linear Regression	0.3387	0.9809	0.3387	0.4772	0.3357	0.6673
Logistic Regression	0.9849	0.9835	0.9849	0.9820	0.9901	0.0046
Linear Support Vector Machine	0.9861	0.9784	0.9861	0.9820	0.9916	0.0028
Decision Tree	0.5864	0.9844	0.5864	0.6762	0.5857	0.4149
Extra Trees	0.7785	0.9856	0.7785	0.8313	0.7804	0.2176
Gradient Boosting	0.7814	0.9837	0.7814	0.8573	0.7833	0.2148
Isolation Forest	0.5598	0.9771	0.5598	0.7046	0.5607	0.4384
Random Forest	0.7613	0.9859	0.7613	0.8270	0.7627	0.2358
K-Means	0.4750	0.9795	0.4750	0.6136	0.4749	0.5252
Mini-Batch K-Means	0.4789	0.9796	0.4789	0.6139	0.4789	0.5211
BIRCH	0.5678	0.9807	0.5678	0.6508	0.5675	0.4328
Local Outlier Factor	0.3830	0.9700	0.3830	0.4966	0.3826	0.6176
Multi-Layer Perceptron	0.8647	0.9846	0.8647	0.9115	0.8675	0.1297
Bernoulli Restricted Boltzmann Machine	0.9904	0.9811	0.9904	0.9857	0.9952	0.0000

[1]Testing subset #5 was not averaged into the performance metrics in this table, as the nature of the suppress attack (an attack with no labeled attack CAN frames) results in uninformative performance metrics.

average F1-score was achieved *without* the subdivision. For linear SVM, the highest average F1-score was achieved *without* the timestamp. However, if we look at models which achieved an F1-score greater than 0.5 for at least one condition (with or without timestamp, with or without data field subdivision), then we can see that removing the timestamp *never* improved the average F1-score.

When it came to subdividing the `data field` feature, the results were too mixed to generalize; however, when it came to including—or excluding—the `timestamp` feature, the results were clearer cut. Anecdotally, we can see that excluding the timestamp had a catastrophic impact on the performance of the decision tree model but enhanced the performance of the isolation forest model. In general, though, excluding the timestamp is detrimental. Many of the high-performing machine learning models (e.g., logistic regression, linear support vector machine, and Bernoulli restricted Boltzmann machine) exhibited very slight drops in performance when the timestamp was removed. With the timestamp, the F1-scores were 0.9822, 0.9818, 0.9858, respectively; without the timestamp, the F1-scores were 0.9820, 0.9820, and 0.9857, respectively. The *k*-nearest neighbors model saw a more significant drop in performance—from 0.9467 to 0.9158. Many of the models that *improved* without the timestamp were medium- or poor-performing models; the BIRCH model's performance increased from 0.5476 to 0.6508.

Table 7: `can-train-and-test`: Average performance metrics when the data field is not subdivided into eight individual features (one feature per byte) [EXCLUDE subdivision, include timestamp, include testing subset #6][1].

Model	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9782	0.9798	0.9782	0.9775	0.9840	0.0102
K-Nearest Neighbors	0.8769	0.9804	0.8769	0.9181	0.8813	0.1143
Linear Regression	0.2416	0.9828	0.2416	0.3484	0.2372	0.7669
Logistic Regression	0.9666	0.9794	0.9666	0.9710	0.9717	0.0232
Linear Support Vector Machine	0.9883	0.9768	0.9883	0.9825	0.9941	0.0000
Decision Tree	0.7634	0.9843	0.7634	0.8384	0.7653	0.2328
Extra Trees	0.8687	0.9811	0.8687	0.9128	0.8728	0.1230
Gradient Boosting	0.6590	0.9844	0.6590	0.7532	0.6596	0.3397
Isolation Forest	0.6396	0.9780	0.6396	0.7679	0.6408	0.3579
Random Forest	0.8920	0.9808	0.8920	0.9276	0.8964	0.0991
K-Means	0.6028	0.9747	0.6028	0.7190	0.6060	0.3908
Mini-Batch K-Means	0.6124	0.9773	0.6124	0.7266	0.6139	0.3845
BIRCH	0.9883	0.9768	0.9883	0.9825	0.9941	0.0000
Local Outlier Factor	0.2813	0.9818	0.2813	0.3509	0.2784	0.7242
Multi-Layer Perceptron	0.8970	0.9805	0.8970	0.9320	0.9011	0.0947
Bernoulli Restricted Boltzmann Machine	0.9883	0.9768	0.9883	0.9825	0.9941	0.0000

[1]Testing subset #5 was not averaged into the performance metrics in this table, as the nature of the suppress attack (an attack with no labeled attack CAN frames) results in uninformative performance metrics.

Table 8: `hcrl-ch` ONLY: Averages with default parameters [include subdivision, include timestamp].

Model	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.8082	0.7137	0.8082	0.7303	0.8956	0.0073
K-Nearest Neighbors	0.9106	0.9152	0.9106	0.8980	0.9512	0.0059
Linear Regression	0.4587	0.8381	0.4587	0.4870	0.3963	0.6575
Logistic Regression	0.8147	0.7663	0.8147	0.7354	0.9022	0.0005
Linear Support Vector Machine	0.8129	0.7590	0.8129	0.7308	0.9014	0.0002
Decision Tree	0.7143	0.8253	0.7143	0.7339	0.7238	0.2638
Extra Trees	0.8928	0.8860	0.8928	0.8809	0.9249	0.0405
Gradient Boosting	0.9646	0.9701	0.9646	0.9648	0.9684	0.0276
Isolation Forest	0.4273	0.7554	0.4273	0.4680	0.3900	0.6415
Random Forest	0.8604	0.9034	0.8604	0.8713	0.8602	0.1395
K-Means	0.7824	0.8046	0.7824	0.7834	0.8031	0.1730
Mini-Batch K-Means	0.7908	0.8117	0.7908	0.7911	0.8098	0.1680
BIRCH	0.8060	0.7179	0.8060	0.7580	0.8751	0.0485
Local Outlier Factor	0.1752	0.4945	0.1752	0.0594	0.0163	0.9980
Multi-Layer Perceptron	0.8892	0.9037	0.8892	0.8662	0.9428	0.0000
Bernoulli Restricted Boltzmann Machine	0.8114	0.6589	0.8114	0.7271	0.9007	0.0000

For our comparison, in Table 10, we average the results of experiments run against the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset. As such, we can compare the `can-train-and-test` dataset to an aggregated—generalized—perspective of the two HCRL datasets. The results of the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset individually can be found in Tables 8 and 9, respectively.

Zooming in on our comparison, we examine Table 8 and Table 11, in which `hcrl-ch` is juxtaposed against `can-train-and-test`. Similarly, to compare `hcrl-sa` to `can-train-and-test`, we refer to Tables 9 and Table 11. Both Table 5 (referenced in our earlier discussion) and Table 11 contain the average metrics of the `can-train-and-test` dataset with default parameters; however, Table 5 includes testing subset #6 (masquerade) in its averages, whereas Table 11 does not. The `hcrl-ch` and `hcrl-sa` datasets do not contain CAN traffic data equivalent to testing subset #6; as such, in Table 11, we exclude testing subset #6 from the averages to enhance our perspective and our comparison.

When we average our experiments, we can see that many machine learning models perform very well (F1-score > 0.9) against the `can-train-and-test` dataset. A few models perform moderately well (0.9 > F1-score > 0.7), and a few perform poorly (F1-score ≈ 0.5). Looking at the average model performance for the `hcrl-ch` dataset, we see that performance is generally lower: one model performs very well (F1-score > 0.9), many models perform well (0.9 > F1-score > 0.7), a few models perform poorly (F1-score ≈ 0.5), and one model performs very poorly (F1-score < 0.4). Similarly, when averaging model performance for the `hcrl-sa` dataset, we see one

Table 9: `hcrl-sa` ONLY: Averages with default parameters [include subdivision, include timestamp].

Model	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.7739	0.7954	0.7739	0.7839	0.8152	0.1411
K-Nearest Neighbors	0.7959	0.8349	0.7959	0.8109	0.8310	0.1322
Linear Regression	0.2749	0.8452	0.2749	0.2836	0.2213	0.8152
Logistic Regression	0.8440	0.7627	0.8440	0.8003	0.9028	0.0337
Linear Support Vector Machine	0.8550	0.7628	0.8550	0.8053	0.9149	0.0205
Decision Tree	0.6654	0.8758	0.6654	0.7232	0.6561	0.3520
Extra Trees	0.8129	0.8397	0.8129	0.8177	0.8530	0.1047
Gradient Boosting	0.9107	0.9011	0.9107	0.9031	0.9315	0.0467
Isolation Forest	0.4815	0.8013	0.4815	0.5590	0.4718	0.5369
Random Forest	0.8521	0.8833	0.8521	0.8597	0.8710	0.1090
K-Means	0.4396	0.7764	0.4396	0.5207	0.4284	0.5820
Mini-Batch K-Means	0.7287	0.7851	0.7287	0.7511	0.7635	0.2000
BIRCH	0.3274	0.8022	0.3274	0.3830	0.2926	0.7384
Local Outlier Factor	0.1959	0.7267	0.1959	0.1804	0.1382	0.9023
Multi-Layer Perceptron	0.8665	0.8751	0.8665	0.8699	0.8923	0.0805
Bernoulli Restricted Boltzmann Machine	0.8724	0.7628	0.8724	0.8135	0.9337	0.0000

Table 10: `hcrl-ch`, `hcrl-sa`: Averages with default parameters [include subdivision, include timestamp].

Model	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.7910	0.7545	0.7910	0.7571	0.8554	0.0742
K-Nearest Neighbors	0.8532	0.8750	0.8532	0.8544	0.8911	0.0690
Linear Regression	0.3668	0.8417	0.3668	0.3853	0.3088	0.7364
Logistic Regression	0.8293	0.7645	0.8293	0.7679	0.9025	0.0171
Linear Support Vector Machine	0.8339	0.7609	0.8339	0.7680	0.9082	0.0103
Decision Tree	0.6898	0.8506	0.6898	0.7285	0.6899	0.3079
Extra Trees	0.8529	0.8629	0.8529	0.8493	0.8890	0.0726
Gradient Boosting	0.9376	0.9356	0.9376	0.9340	0.9500	0.0371
Isolation Forest	0.4544	0.7783	0.4544	0.5135	0.4309	0.5892
Random Forest	0.8562	0.8933	0.8562	0.8655	0.8656	0.1243
K-Means	0.6110	0.7905	0.6110	0.6520	0.6157	0.3775
Mini-Batch K-Means	0.7597	0.7984	0.7597	0.7711	0.7866	0.1840
BIRCH	0.5667	0.7601	0.5667	0.5705	0.5839	0.3934
Local Outlier Factor	0.1856	0.6106	0.1856	0.1199	0.0773	0.9502
Multi-Layer Perceptron	0.8779	0.8894	0.8779	0.8681	0.9176	0.0402
Bernoulli Restricted Boltzmann Machine	0.8419	0.7108	0.8419	0.7703	0.9172	0.0000

model that performs very well (F1-score > 0.9), many models that perform moderately well (0.9 > F1-score > 0.7), a few models that perform poorly (F1-score ≈ 0.5), and a few models that perform very poorly (F1-score < 0.4). The one model that performed very well against the `hcrl-ch` dataset was also the one model that performed well against the `hcrl-sa` dataset—the gradient boosting model. Interestingly, when pitted against `can-train-and-test`, it achieved an average F1-score of 0.8897, below our cut-off for the "performed very well" classification.

We calculated the standard deviation (STD) of the machine learning models' average F1-scores to assess how much the average F1-score varied from model to model. For `can-train-and-test`, the STD was 0.2247. For `hcrl-ch`, the STD was 0.2202. Lastly, for `hcrl-sa`, the STD was 0.2243. Variation in performance from model to model is important because it enables us to differentiate between models—e.g., we can differentiate between "good" models and "great" models, we can rank models by performance, and we can determine which models can best detect which attacks. The `can-train-and-test` dataset yielded a greater standard deviation across all models than either the `hcrl-ch` or `hcrl-sa` datasets.

In Tables 12 (`hcrl-ch` and `hcrl-sa`) and 13 (`can-train-and-test`), we provide all of the KNN metrics—all datasets, all sub-datasets, and all testing subsets. Referring back to Tables 8, 9, and 11, we know that the KNN model performed significantly better when pitted against the `can-train-and-test` dataset than when pitted against the `hcrl-ch` dataset and the `hcrl-sa` dataset. Tables 12 and 13 confirm that the KNN model's performance is significantly higher for `can-train-and-test` than for `hcrl-ch` and `hcrl-sa`: the F1-scores for `can-train-and-test` are generally in the upper 0.8s to the upper 0.9s, while the F1-scores for `hcrl-ch` are in the mid-0.8s to mid-0.9s and the F1-scores for `hcrl-sa` are in the lower 0.7s to lower 0.9s.

Table 11: `can-train-and-test`: Averages with default parameters [include subdivision, include timestamp, EXCLUDE testing subsets #5 and #6].[1]

Model	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9495	0.9789	0.9495	0.9627	0.9548	0.0398
K-Nearest Neighbors	0.9208	0.9836	0.9208	0.9467	0.9248	0.0712
Linear Regression	0.3283	0.9797	0.3283	0.4710	0.3254	0.6774
Logistic Regression	0.9856	0.9830	0.9856	0.9822	0.9909	0.0036
Linear Support Vector Machine	0.9861	0.9779	0.9861	0.9818	0.9917	0.0025
Decision Tree	0.7154	0.9861	0.7154	0.7979	0.7159	0.2835
Extra Trees	0.8667	0.9846	0.8667	0.9102	0.8700	0.1266
Gradient Boosting	0.8270	0.9844	0.8270	0.8897	0.8291	0.1687
Isolation Forest	0.5363	0.9783	0.5363	0.6842	0.5366	0.4630
Random Forest	0.8986	0.9842	0.8986	0.9347	0.9022	0.0941
K-Means	0.4093	0.9787	0.4093	0.5484	0.4066	0.5959
Mini-Batch K-Means	0.5180	0.9774	0.5180	0.6484	0.5189	0.4802
BIRCH	0.4854	0.9779	0.4854	0.5476	0.4861	0.5132
Local Outlier Factor	0.3907	0.9699	0.3907	0.5119	0.3898	0.6110
Multi-Layer Perceptron	0.9355	0.9844	0.9355	0.9561	0.9397	0.0562
Bernoulli Restricted Boltzmann Machine	0.9905	0.9812	0.9905	0.9858	0.9952	0.0000

[1]This evaluation was conducted to compare the `can-train-and-test` dataset with the HCRL Car Hacking (`hcrl-ch`) dataset and the HCRL Survival Analysis (`hcrl-sa`) dataset. Testing subsets #5 and #6 were not averaged into the performance metrics in this table, since there is no equivalent data in the two HCRL datasets.

Table 12: `hcrl-ch`, `hcrl-sa`: Performance metrics for the *k*-nearest neighbors (KNN) model [include subdivision, include timestamp].

Dataset, Testing subset	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
CH, #1	0.8419	0.8553	0.8419	0.8102	0.9142	0.0073
CH, #2	0.9580	0.9592	0.9580	0.9561	0.9775	0.0026
CH, #3	0.9088	0.9114	0.9088	0.8973	0.9500	0.0069
CH, #4	0.9336	0.9348	0.9336	0.9282	0.9630	0.0066
SA, #1	0.8944	0.8908	0.8944	0.8831	0.9365	0.0195
SA, #2	0.6836	0.7355	0.6836	0.7072	0.7255	0.2301
SA, #3	0.9393	0.9322	0.9393	0.9343	0.9593	0.0203
SA, #4	0.6662	0.7809	0.6662	0.7190	0.7026	0.2590

Table 13: `can-train-and-test`: Performance metrics for the *k*-nearest neighbors (KNN) model [include subdivision, include timestamp].

Sub-dataset, Testing subset	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
#1, #1	0.9855	0.9850	0.9855	0.9853	0.9893	0.0069
#1, #2	0.9714	0.9497	0.9714	0.9604	0.9840	0.0033
#1, #3	0.9976	0.9964	0.9976	0.9965	0.9988	0.0000
#1, #4	0.9982	0.9979	0.9982	0.9981	0.9987	0.0008
#2, #1	0.9988	0.9988	0.9988	0.9988	0.9990	0.0007
#2, #2	0.7930	0.9966	0.7930	0.8824	0.7932	0.2065
#2, #3	0.9261	0.9951	0.9261	0.9594	0.9272	0.0717
#2, #4	0.7658	0.9955	0.7658	0.8637	0.7656	0.2345
#3, #1	0.9912	0.9913	0.9912	0.9899	0.9956	0.0000
#3, #2	0.9461	0.9643	0.9461	0.9546	0.9537	0.0387
#3, #3	0.9980	0.9963	0.9980	0.9971	0.9990	0.0000
#3, #4	0.9489	0.9580	0.9489	0.9534	0.9589	0.0310
#4, #1	0.9825	0.9822	0.9825	0.9768	0.9911	0.0002
#4, #2	0.7357	0.9947	0.7357	0.8448	0.7361	0.2635
#4, #3	0.9820	0.9811	0.9820	0.9766	0.9908	0.0004
#4, #4	0.7117	0.9543	0.7117	0.8096	0.7156	0.2805

Table 14: `hcrl-sa, can-train-and-test`: Performance metrics for the multi-layer perceptron (MLP) model when pitted against `hcrl-sa` and `can-train-and-test` (sub-dataset #1).

Dataset, Sub-dataset Testing subset	Accuracy	Precision	Recall (TPR)	F1-score	G-mean	FPR
SA, #1	0.9810	0.9819	0.9810	0.9812	0.9811	0.0187
SA, #2	0.7407	0.7517	0.7407	0.7461	0.7875	0.1627
SA, #3	0.9708	0.9712	0.9708	0.9678	0.9848	0.0009
SA, #4	0.7736	0.7954	0.7736	0.7844	0.8158	0.1396
CT&T, #1, #1	0.9873	0.9860	0.9873	0.9866	0.9910	0.0053
CT&T, #1, #2	0.9694	0.9539	0.9694	0.9609	0.9815	0.0062
CT&T, #1, #3	0.9976	0.9955	0.9976	0.9965	0.9988	0.0000
CT&T, #1, #4	0.9983	0.9979	0.9983	0.9981	0.9988	0.0007

Lastly, in Table 14 we assess the performance of the multi-layer perceptron (MLP) model when pitted against the `can-train-and-test` dataset and the `hcrl-sa` dataset. We conduct experiments that showcase the MLP model's capabilities against four testing subsets: `test_01_known_vehicle_known_attack`, `test_02_unknown_vehicle_known_attack`, `test_03_known_vehicle_unknown_attack`, and `test_04_unknown_vehicle_unknown_attack`. For testing subsets #1 and #3, the F1-score are fairly close: 0.9866 and 0.9965 for `can-train-and-test`; 0.9812 and 0.9678 for `hcrl-sa`. However, when we look at testing subsets #2 and #4, the MLP model performs significantly better when pitted against the `can-train-and-test` dataset. The F1-scores are 0.9609 and 0.9981 for `can-train-and-test`, whereas, for `hcrl-sa`, the F1-scores are 0.7461 and 0.7844. In all testing subsets, the MLP performed better against `can-train-and-test` than against `hcrl-sa`.

5 LIMITATIONS & FUTURE WORK

5.1 Limitations

We have identified two main limitations of our work—both of which correspond to opportunities for future work:

(1) *Our feature investigation is not comprehensive.* In our future work, we will select additional features for investigation (e.g., data length code). Moreover, for each machine learning model, we will determine which configuration of features is optimal.
(2) Our dataset comparison is not comprehensive. In our future work, we will target additional CAN intrusion detection datasets for comparison (e.g., signal-based datasets). Furthermore, we will analyze the use cases of the existing open-access CAN intrusion detection datasets, and we will determine, for each use case, which dataset would be best. For example, for signal-based IDSs, a signal-based dataset would be better than `can-train-and-test`.

5.2 Future Work

Feature investigation. We have identified two features for further investigation: (1) the `data length code (DLC)` feature and (2) the `data field` feature.

As mentioned in Section 3, the DLC is included as a feature in both the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset. As such, we have developed the following research question: Would the `DLC` feature enhance the performance of one or more of our machine learning models? Normally, the `DLC` feature should match the actual length of the `data field` feature. A mismatch between the DLC and the size of the data field (in bytes) is an obvious indication of a malfunction or an attack. However, even a moderately-capable attacker should be able to generate attack CAN frames in which the DLC is correct for the attack CAN frame's data field. `cansend`, one of Linux's `can-utils`, automatically computes and sends the correct value for the DLC based on the data field. In fact, depending on the tools and technologies used, it might be more difficult to send a CAN frame with an invalid DLC. As such, the value of the DLC to a machine learning model is difficult to estimate.

In this work, we investigated the `data field` feature—specifically, we investigated the impact of subdividing (or not subdividing) the data field into eight discrete features (one feature per byte). However, we did not explore the impact of removing the `data field` feature in its entirety. Some IDSs (e.g., ID sequence-based IDSs [9, 11, 17]) do not look at the `data field` feature. Instead, they analyze patterns in the `arbitration identifier` feature. Therefore, we have composed the following research question: If we remove the `data field` feature, will some machine learning models perform better—perhaps because noise in the data has been reduced?

Dataset comparison. Though `hcrl-ch` is believed to be the most popular CAN intrusion detection dataset in the literature [13, 18], many others are also wildly popular or gaining traction. For example, the Synthetic CAN Bus dataset (SynCAN) [4] is the most popular CAN intrusion detection dataset when it comes to evaluating unsupervised payload-based IDSs [18]. Moreover, the Real ORNL Automotive Dynamometer CAN intrusion (ROAD) dataset [23, 24] contains 33 attack traffic captures—and ten unique attacks, depending on how the unique attacks are counted [13]—compared to the ≤ 5 attacks available in pre-existing open-access CAN intrusion detection datasets. Even our new dataset, `can-train-and-test`, contains only nine unique attacks. Unfortunately, while the ROAD dataset contains descriptions that can help researchers label the CAN traffic captures, the CAN traffic captures themselves are not labeled. In order to compare the ROAD dataset to the `can-train-and-test` dataset, we would need to label the ROAD dataset. In our future work, we plan to label the ROAD dataset and compare

it to `can-train-and-test`. In addition, we plan to develop "practitioner guidance" indicating which dataset—`can-train-and-test` or ROAD—would be best suited to a given research problem.

As earlier mentioned, we did not compare `can-train-and-test` to a signal-based dataset. In our future work, we plan to conduct a *qualitative* assessment of the benefits of signal-based datasets (e.g., SynCAN, ROAD) compared to raw datasets (e.g., `hcrl-sa`, `can-train-and-test`). In a later step, we would like to conduct a *quantitative* assessment of signal-based datasets vs. raw datasets. We will investigate the possibility of adapting our machine learning models to receive either raw data or signal values as input. Such an adaptation would allow us to quantitatively compare `can-train-and-test` to a signal-based dataset such as SynCAN.

6 CONCLUSION

In this work, we investigated the newly-published `can-train-and-test` dataset, and we pitted that newcomer to the well-established HCRL Car Hacking and Survival Analysis datasets.

In our feature sleuthing, we determined that the presence of the `timestamp` feature enhanced the performance of many of our machine learning models. In particular, if a given model was even moderately successful (i.e., an average F1-score greater than 0.5), then excluding the timestamp either had no impact or had a negative impact. As such, we determined that there is no benefit in removing the `timestamp` feature. When it came to the `data field` feature, we found that some machine learning models benefited from a subdivided data field (eight bytes as distinct features), while others benefited from an unpartitioned data field. Therefore, we can choose to subdivide—or not subdivide—the `data field` feature depending on the specific model selected.

Lastly, in our comparison, we concluded that `can-train-and-test` provides novel features and insights when juxtaposed against existing open-access CAN intrusion detection datasets—such as the HCRL datasets. Specifically, we observed that `can-train-and-test` contains much more training and testing data than either of the HCRL datasets (an order of magnitude more), and `can-train-and-test`'s attack traffic captures are more realistic in terms of class imbalance. When pitted against the `can-train-and-test` dataset, half of our machine learning models—eight out of sixteen—achieved an average F1-score above 0.9. When pitted against the HCRL datasets, which contain much less training data, the average F1-scores were noticeably lower. Though average F1-scores were high for the `can-train-and-test` dataset, there was no loss of differentiation power; the standard deviation of the models' average F1-scores was 0.2247, exceeding the standard deviations of the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset—0.2202 and 0.2243, respectively.

ACKNOWLEDGMENTS

We would like to thank our test drivers, especially Christian Lampe, Julie Lampe, and Wayne & Vickie Olsen.

REFERENCES

[1] Tri P. Doan and Subramaniam Ganesan. 2017. CAN Crypto FPGA Chip to Secure Data Transmitted Through CAN FD Bus Using AES-128 and SHA-1 Algorithms with A Symmetric Key. *SAE International* (2017). https://saemobilus.sae.org/content/2017-01-1612/

[2] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. 2018. Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation. *University of Texas at El Paso* (2018). https://scholarworks.utep.edu/cs_techrep/1209/

[3] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. 2018. Anomaly intrusion detection method for vehicular networks based on survival analysis. *Vehicular Communications* 14 (2018), 52 – 63. https://www.sciencedirect.com/science/article/pii/S2214209618301189

[4] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. 2020. CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data. *IEEE Access* 8 (2020), 58194 – 58205. https://ieeexplore.ieee.org/document/9044377

[5] Riadul Islam and Rafi Ud Daula Refat. 2020. Improving CAN bus security by assigning dynamic arbitration IDs. *Journal of Transportation Security* 13 (2020), 19 – 31. https://link.springer.com/article/10.1007/s12198-020-00208-0

[6] Brooke Lampe. 2023. can-train-and-test. *Bitbucket* (2023). https://bitbucket.org/brooke-lampe/can-train-and-test/src/master/

[7] Brooke Lampe. 2023. can-train-and-test-v1.5. *Bitbucket* (2023). https://bitbucket.org/brooke-lampe/can-train-and-test-v1.5/src/master/

[8] Brooke Lampe. 2023-10. can-train-and-test: A New CAN Intrusion Detection Dataset. *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)* (2023-10). https://doi.org/10.1109/VTC2023-Fall60731.2023.10333756

[9] Brooke Lampe and Weizhi Meng. 2022. IDS for CAN: A Practical Intrusion Detection System for CAN Bus Security. *2022 IEEE Global Communications Conference (GLOBECOM 2022)* (2022). https://doi.org/10.1109/GLOBECOM48099.2022.10001536

[10] Brooke Lampe and Weizhi Meng. 2023. can-logic: Automotive Intrusion Detection via Temporal Logic. *IoT '23: Proceedings of the 13th International Conference on the Internet of Things* (2023), 113 – 120. https://doi.org/10.1145/3627050.3627059

[11] Brooke Lampe and Weizhi Meng. 2023. Intrusion Detection in the Automotive Domain: A Comprehensive Review. *IEEE Communications Surveys & Tutorials* 25 (2023), 2356 – 2426. https://doi.org/10.1109/COMST.2023.3309864

[12] Brooke Lampe and Weizhi Meng. 2023. A survey of deep learning-based intrusion detection in automotive applications. *Expert Systems with Applications* 221 (2023), 119771. https://doi.org/10.1016/j.eswa.2023.119771

[13] Brooke Lampe and Weizhi Meng. 2024. can-train-and-test: A Curated CAN Dataset for Automotive Intrusion Detection. *Computers & Security* 140 (2024), 103777. https://doi.org/10.1016/j.cose.2024.103777

[14] Brooke Elizabeth Lampe. 2023. can-train-and-test. *DTU Data* (2023). https://doi.org/10.11583/DTU.24805533

[15] Kerstin Lemke, Christof Paar, and Marko Wolf. 2006. *Embedded Security in Cars.* Springer-Verlag. https://link.springer.com/book/10.1007/3-540-28428-1

[16] Siti-Farhana Lokman, Abu Talib Othman, and Muhammad-Husaini Abu-Bakar. 2019. Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review. *EURASIP Journal on Wireless Communications and Networking* 2019 (2019). https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-019-1484-3

[17] Mirco Marchetti and Dario Stabili. 2017. Anomaly detection of CAN bus messages through analysis of ID sequences. *2017 IEEE Intelligent Vehicles Symposium (IV)* (2017), 1577 – 1583. https://ieeexplore.ieee.org/document/7995934

[18] Sampath Rajapaksha, Harsha Kalutarage, M. Omar Al-Kadri, Andrei Petrovski, Garikayi Madzudzo, and Madeline Cheah. 2023. AI-Based Intrusion Detection Systems for In-Vehicle Networks: A Survey. *Comput. Surveys* 55 (2023), 1 – 40. https://doi.org/10.1145/3570954

[19] Eunbi Seo, Hyun Min Song, and Huy Kang Kim. 2018. GIDS: GAN based Intrusion Detection System for In-Vehicle Network. *2018 16th Annual Conference on Privacy, Security and Trust (PST)* (2018). https://ieeexplore.ieee.org/document/8514157

[20] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. 2020. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications* 21 (2020), 100198. https://www.sciencedirect.com/science/article/pii/S2214209619302451

[21] Hiroshi Ueda, Ryo Kurachi, Hiroaki Takada, Tomohiro Mizutani, Masayuki Inoue, and Satoshi Horihata. 2015. Security Authentication System for In-Vehicle Network. *Sei Technical Review* (2015). https://global-sei.com/technology/tr/bn81/pdf/81-01.pdf

[22] Vilhelm Verendel, Dennis K. Nilsson, Ulf E. Larson, and Erland Jonsson. 2008. An Approach to using Honeypots in In-Vehicle Networks. *2008 IEEE 68th Vehicular Technology Conference* (2008). https://ieeexplore.ieee.org/document/4657092

[23] Miki E. Verma, Michael D. Iannacone, Robert A. Bridges, Samuel C. Hollifield, Bill Kay, and Frank L. Combs. 2020. ROAD: The Real ORNL Automotive Dynamometer Controller Area Network Intrusion Detection Dataset (with a comprehensive CAN IDS dataset survey & guide). *arXiv* (2020). https://arxiv.org/abs/2012.14600v1

[24] Miki E. Verma, Michael D. Iannacone, Robert A. Bridges, Samuel C. Hollifield, Pablo Moriano, Bill Kay, and Frank L. Combs. 2022. Addressing the Lack of Comparability & Testing in CAN Intrusion Detection Research: A Comprehensive Guide to CAN IDS Data & Introduction of the ROAD Dataset. *IEEE Transactions on Vehicular Technology* (2022). https://arxiv.org/pdf/2012.14600v2.pdf