Using internal context to detect automotive controller area network attacks

Tomlinson, A., Bryans, J. & Shaikh, S. A.

DOI 10.1016/j.compeleceng.2021.107048
ISSN 0045-7906

Publisher: Elsevier

Using Internal Context To Detect Automotive Controller Area Network Attacks

Andrew Tomlinson, Jeremy Bryans, Siraj Ahmed Shaikh

Systems Security Group, Institute for Future Transport and Cities, Coventry University, UK, CV1 5FB

Abstract

The rise in data use within cars has led to concerns about their cybersecurity. The Controller Area Network (CAN) enables communication between components core to the car's safety and performance, and has been demonstrated to be particularly vulnerable to hacking and malicious cyber-intrusion. CAN intrusion detection systems have been envisaged. Signatures of known attacks might be used for detection, but this method holds many limitations. Although some attacks might change packet broadcast rates or add unknown packets onto the network, attacks that have little or no effect on these, yet can alter the packet data, have also been devised. We therefore test three novelty detection methods (Local Outlier Factor, Compound Classifier and One-Class Support Vector Machine) that might identify an attack based solely on anomalies in CAN packet field data-values. The methods compare values across a cluster of CAN packets broadcast from different control units, so potentially could identify an attacked control unit even when its subsequent fabricated payload data-values remain plausible. We test the methods on data from two different makes of car across a range of manipulation magnitudes, reflecting the unpredictability of attacks. Different training regimes are tested, enabling us to assess validity across journeys. We also consider the processes needed to determine the CAN fields that might be included in the intrusion detection cluster, and present algorithms for automating those processes.

Keywords: intrusion detection, controller area network, automotive cybersecurity, machine learning

1. Introduction

As cars become more reliant on communication and information processing, concerns are growing about their cybersecurity. In addition to the risk of data being intercepted, illicit cyber intrusion onto the vehicle's internal networks is now a concern [1]. Motives for such vehicle attacks include sabotage, theft, extortion, mischief, espionage and terrorism [2].

One important in-vehicle network in particular, the Controller Area Network (CAN), has been shown to be vulnerable to intrusion attacks that could

compromise the safety of passengers or pedestrians, or cause damage to the car [3, 4]. The CAN is used by electronic control units (ECUs) that govern the car's performance and safety. The CAN protocol specifies mechanisms to ensure robust, error-free messaging, but it lacks security features (such as encryption and authentication) expected in a modern computer network. A high-value car might have 100 or more ECUs, though cheaper cars are likely to have fewer. ECUs increasingly need to communicate with each other and with external devices as vehicles become more automated and autonomous. ECUs broadcast packets onto the CAN to communicate with each other and with external systems via CAN gateways. They typically broadcast many packets per second, either at fixed rates or in response to received requests, so CAN traffic is likely to constantly maintain a thousand or more packet-broadcasts per second. Attacks have been demonstrated that inject fabricated packets onto the CAN, thus transmitting spoofed data (e.g. [3, 4, 5]).

Developing a CAN attack detection system is difficult because attack manifestations are difficult to predict, especially across different makes and models of car. In addition, the configuration of CAN data is kept secret by the car manufacturers, presenting a challenge to research and the independent development of generic attack detection systems.

Our contributions:

1. We evaluate three methods that might be used to detect cyber-attack resulting anomalies in CAN packet data: Compound Classifier (CC); One-Class Support Vector Machine (OCSVM), and Local Outlier Factor (LOF). These use one-class outlier detection, so offer potential where attacks are novel. They are also chosen for their capacity to cope with the specification of the CAN data being secret and differing across car models.
2. We test these on data that simulates representative attack characteristics on cars from different manufacturers, across differing magnitudes of attack manipulation, and across training regimes.
3. We present processes for parsing the CAN data and transforming it into structures suitable for payload anomaly detection, even where the car's data schema is undisclosed.

This paper is structured as follows: Section 2 provides background to CAN attacks and intrusion detection. Our detection model is outlined in Section 3. Section 4 considers CAN intrusion detection as a one class problem, and outlines our detection methods. Our CAN data capture and methods for determining sensor data and field clusters are explained in Section 5. Section 6 describes our evaluation experiment and test data generation, and results and discussion are presented in Section 7. Finally, conclusions and future work are discussed in Section 8.

2. Background

CAN data packets comprise an ID field, up to eight one-byte data fields, and additional fields used for verification and error handling. The packet IDs are

unique to each ECU, and are used by other ECUs to decide whether the packet contains data relevant to them. Depending on their packet's specification, each data field might contain continuous sensor data, category data, constant data (just one value), or cyclical counter data [6].

In some packets, the data fields are composite - the data value being derived by joining the two fields together. Although it is possible to read CAN packets, such as via a device connected to the car's On-Board Diagnostics (OBD) port, it is difficult for the layman to determine which packet IDs stem from which ECUs, what functions the packets describe, what their data means, or how it is derived. Such information is not published by the manufactures. As discussed later, this is one of the research challenges, and it makes it difficult to devise and evaluate methods that might be deployed in CAN intrusion detection systems.

Protecting the CAN from cyber attacks will require the development of robust security mechanisms; though car life-cycles, industry standards formulation, and production schedules hinder their development and deployment [7]. Security developments for in-vehicle networks are thus likely to lag behind the hacker's capabilities. Systems that detect an attack taking place will therefore be important. Proposed attacks typically involve injecting fake packets with phoney data values; or, preventing the broadcast of packets containing information required elsewhere.

Checkoway *et al.* [3] and Koscher *et al.* [4] published frequently cited attacks where malicious packets were broadcast onto the CAN, disrupting or illicitly controlling operations. CAN attacks might entail subtle and convoluted fabrications. Miller and Valasek [5] invoked a Park Assist System to manipulate steering. Their attack had to first inject messages to fool the ECUs into believing the car was in reverse and travelling at low speed, enabling activation of the speed-restricted Park Assist Mode, which in turn responded to fake steering command messages injected onto the CAN. The forged messages injected during this attack were legitimate and plausible, containing values that would be expected in another situation.

There are two strands typically investigated for CAN attack detection. One strand is to detect frequency and timing packet-broadcasts changes that might indicate an attack (e.g. [8, 9]), while other authors have considered frequency characteristics in combination with other characteristics, such as payload values (e.g. [10, 11]). However, not all attacks can be expected to affect timings and broadcast rates [12]. Masquerade attacks have been proposed in which a legitimate ECU is incapacitated, and a masquerading ECU broadcasts fake packets in its place at the same frequency [5, 13], or takes other action to cover any broadcast time-shifts [14]. Therefore, another detection option, considered in our paper, would be to detect anomalies in the data payload of the packets.

CAN payload anomaly detectors can be tailored specifically to one data field, using assumed knowledge of how that single data field is derived and calculated. Such tailoring can be effective (e.g. [15]), but it presents two problems. First, it implies requiring many classifiers to cover the array of data used by the ECUs. Second, it assumes data derivation knowledge is available. A frequently cited barrier to automotive network research is that the car-model data dictionaries,

which contain such information, are not published and are kept secret by the car manufacturers; although the behaviour of ECUs might not be fully determined in any case [10]. Thus, building CAN intrusion detection systems that employ knowledge of the ECUs' data construction might constrict the scope of the detector and present costly tailoring.

3. Detection Model

Our proposal is that CAN data packets might be clustered into functional groups, such that an attack altering the data in one of the data fields in the cluster might leave it detectably anomalous to the data in the remaining fields in the cluster.

A suitable anomaly classifier would be integrated into a CAN intrusion detection system that could be attached to the CAN network, either directly or via the OBD port in the driver's compartment. The trained classifier would assess the CAN packet fields in real-time by comparing the packet's values against those most recently broadcast by other fields in the same cluster. An anomaly would be flagged when a broadcast is observed that contains sensor data values that are inconsistent with the most recent values from other data fields in the cluster. Thus, the system would process snapshots of the CAN traffic, with each snapshot comprising data payload values for the currently broadcast packet, and the most recently broadcast payload values for packets assigned to the same cluster (Algorithm 1). This method could be implemented on a live system since it requires just the trained classifier plus a one dimensional array holding the most recent values from the desired CAN data fields, with only the changed data fields being updated as each new record is detected.

Algorithm 1 Classification Process

```
ℂ ← cluster comprising associated sensor data fields
for each broadcast of packet ℙ do
  if ℙ fields correspond to ℂ fields then
    ℂ ← updated data for fields corresponding to ℙ
    Apply anomaly test to ℂ
  end if
end for
```

The process requires the identification of CAN packets that might be associated into functional clusters; and the data fields of the packets in the cluster need to be parsed into a structure suitable for the anomaly detection. CAN data fields might hold category data or counting sequences, but for our classifier we are restricting to only fields that hold sensor data. The process also requires suitable anomaly classifiers.

We are interested in solutions that need no knowledge of the specific CAN dictionary, so could be readily adaptable across car makes and models. Also, we seek a solution that makes as few assumptions as possible about the likely

attacks, particularly not anticipating which data fields might be manipulated. As demonstrated in [5], attack fabrications might be convoluted and difficult to predict. We therefore consider classifiers that might be able to detect an anomaly in any one of a cluster of data fields without specifying which data field will be the compromised one.

4. CAN Traffic and One Class Novelty Detection

Our evaluated classifiers allow one class novelty detection. Cybersecurity intrusion detection methods often rely on comparing an instance against a set of known, predefined limits, or against characteristics from known attack classes [16]. Such methods can offer accurate detection for known attacks [17], but they assume that the attack signature is known and that the detection system can be easily updated as new attacks emerge.

One-class classification methods, in contrast, can be used where anomalies (attack data in our case) are difficult to produce, or cannot be assumed to be fully known. Such methods develop a decision surface using only normal data. Subsequent decisions depend on whether the new instance falls within the decision surface (hence the instance deemed normal), or falls outside the decision surface (hence the instance is deemed an anomaly).

Anomaly detection can use outlier detection or novelty detection. Outlier detection assumes the training data contains anomaly (i.e. outlier) instances. The goal of the classifier is therefore to first determine which training instances might be classified as outliers; these are then used to formulate the decision surface. Outlier classifiers assume the outliers are known and can be included in the training data, and that their approximate rates within the population are also known. On the other hand, in novelty detection only normal instances are assumed to be available for training; the decision surface is thus determined using only these.

One-class novelty detection methods seem suitable for automotive CAN intrusion detection because of their non-reliance on anomalous classes (e.g. attack signatures) during classifier training. Attacks on the CAN network are still emerging and cannot be assumed to be known or comprehensively represented by training data. Moreover, it is difficult to derive attack data covering the wide range of makes, model and configurations of car. A one class approach has been demonstrated by Longari *et al.* [18], using Long Short-Term Memory autoencoders to model legitimate data sequences on individual CAN IDs, although the authors acknowledge that their detection process is currently relatively slow.

4.1. Our Evaluated Methods

The Compound Classifier (CC) was devised for rapid classification using a reduced instruction set processor [19]. Primarily used in rapid image processing applications (.e.g. [20]), it has also been used in medical diagnosis [21], and was considered by us as a potential CAN data anomaly detector [22]. Its authors state that the CC: can generalise, learn and make rapid decisions; lends itself

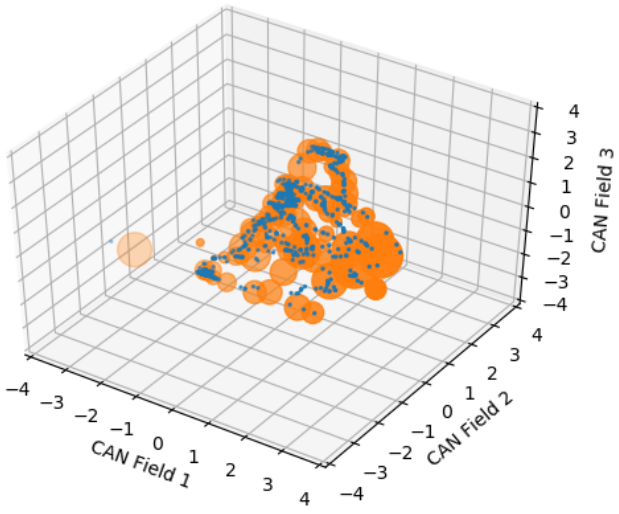

Figure 1: Example Compound Classifier (CC). The classifier's trained hyperspheres are the larger, lighter coloured spheres. The training data set is shown as the dark points. This CC was trained using only three CAN data fields (dimensions) for illustration.

to visualisation; and, can be used for multi-class and one-class problems [19]. It uses nearest neighbour calculations and probability density functions to replace the training data points with a smaller number of hyperspheres. Hyperspheres are constructed as the training data is observed, and are changed in size and location according to whether each training instance is inside them or not. During training iterations, the hyperspheres thus evolve to encompass the shape of the training data set (see Fig. 1).

The Local Outlier Factor (LOF) also relies on distance measurements and density calculations, and has been shown to be resilient to data sets with varying cluster densities [23]. Its calculation entails determining the local density of data points as defined by the distance to data points in the immediate surrounding area [24]. The LOF score is derived by comparing the density score of a point with the density scores of its K nearest neighbours (KNN). Each instance acquires a score which is higher in instances that lie away from their own K nearest neighbours. Thus, the score can be used as a threshold to classify outliers.

Multi-class Support Vector Machines construct a hyperplane that separates the classes of data by the largest margin. In contrast, the One Class Support Vector Machine (OCSVM) attempts to find the largest margin between the projected training data and its origin, thus demarking data-dense regions in input space. Target instances are classified according to their position relative to the margin. The OCSVM is deemed an especially suitable choice where data has a multimodal distribution and is high-dimensional [25], or where training data sets might be small [23].

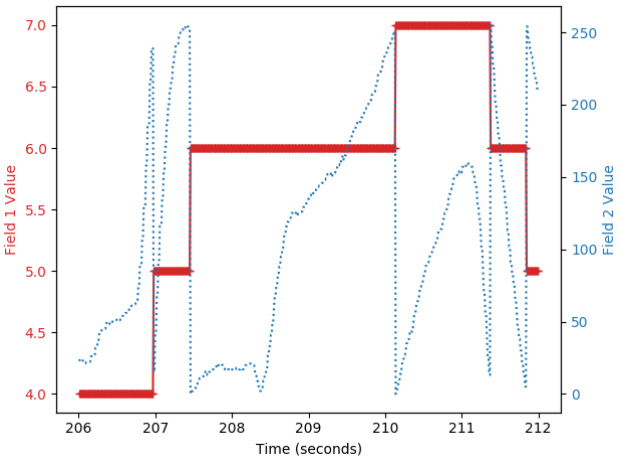

Figure 2: Two data fields in a CAN packet show a pattern that suggests they are meant to be combined to represent a single reading from a sensor. When field 2 reaches the top or bottom of the value capacity suggested by its bit size, a corresponding unitary change is seen in field 1. Field 2 shows also a propensity to jump between its peak and near zero corresponding to the changes in Field 1.

5. CAN Cluster Identification

To determine and evaluate algorithms for identifying the CAN packet clusters, we used data captured from two unmodified popular UK small family cars of different manufacturers, henceforth referred to as Car1 and Car2. CAN logs were captured onto a laptop using a Kvaser Leaf Pro HS v2 OBD-II reader attached to the car's OBD port. We selected journeys that each spanned many minutes and included a variety of routes, roads and driving situations (urban, dual carriageway, parking, etc). The log files were split into pre-analysis and evaluation sets, and processed using Python 3.6. Early analysis therefore involved profiling the data to determine: a) the sensor data fields, including those that represent combined, two-byte values; and, b) the suitable grouping of packets into functional clusters.

5.1. Identifying Concatenated Sensor Fields

The CAN logs revealed concatenated sensor fields that presented a pattern showing a field with a high number of unique values, adjacent to a field with only a few unique values. However, this pattern might be coincidental, so could not be used as the sole indication that fields should be concatenated. More reliably, such data fields could be determined by observing whether the values in a field incremented or decremented by one when the next data field in the packet was a sensor field that incremented gradually towards 255 and then plummeted towards 0, or fell gradually towards 0 and then jumped towards 255 (Fig. 2). Whilst such fields were determined initially by visual inspections, an algorithm was produced to automate this process (Algorithm 2).

Algorithm 2 Determine concatenated fields for a CAN ID

```
𝕏 ← all records in time order for CAN ID(x)
jump ← 100
for all data fields in 𝕏 do
  fieldA, fieldB ← next adjacent data field pair
  for all rows in 𝕏 do
    if fieldA^row(value) = fieldA^row−1(value) + 1 then
      if fieldB^row−1(value) - fieldB^row(value) >jump
      and fieldB^row−1(value) >= fieldB^row−2(value)
       then
        likely ← likely + 1
      else if fieldB^row(value) <= fieldB^row−1(value) then
        unlikely ← unlikely + 1
      end if
    else if fieldA^row(value) = fieldA^row−1(value) - 1 then
      if fieldB^row(value) - fieldB^row−1(value) >jump
      and fieldB^row−1(value) <= fieldB^row−2(value)
       then
        likely ← likely + 1
      else if fieldB^row−1(value) >= fieldB^row(value) then
        unlikely ← unlikely + 1
      end if
    end if
  end for
  if likely - unlikely >threshold then {our threshold = 1}
    fieldAB ← concatenate(fieldA(hex),fieldB(hex))
  end if
end for
```

```
CAN ID: 80
DATA1     24
DATA2    256
DATA3     54
DATA4    256
DATA5      1
DATA6      2
DATA7    256
DATA8      1
dtype: int64
```

```
CAN_ID: 80
Fields: DATA1 DATA2
Likely2: 249
Unlikely2: 0

CAN_ID: 80
Fields: DATA3 DATA4
Likely2: 518
Unlikely2: 0
```

Figure 3: Example output from the composite data field detection algorithm. The algorithm shows both the number of unique values for each field (left of image), and the likelihood score of adjacent fields being connected (right of image). The pattern for the two identified composite fields (DATA1 and DATA2, and DATA3 and DATA4) was typical for all composite fields - a very high number of unique values (e.g. DATA2 or DATA4), following a field with a few unique values (e.g. DATA1 or DATA3).

The algorithm is repeated for all combinations of adjacent data fields for a CAN ID (i.e. Field1 and Field2, Field2 and Field3, Field3 and Field2, Field2 and Field1, etc), and counts instances where the unit change of FieldA corresponds to previous, same-direction, changes in FieldB followed by FieldB suddenly changing in the other direction by an amount ("jump" in the Algorithm) large enough to suggest it has entered the next value-cycle. Such occurrences increase the likelihood that FieldA and FieldB are composite. Conversely, changes in FieldA that do not correspond to such a cyclical pattern in FieldB, decreases the likelihood. Occurrences are counted, generating likelihood and unlikelihood scores.

The algorithm was implemented in Python and tested against both cars. Fields were accepted as being potentially concatenated when the likelihood score was higher than the unlikelihood score. For validation, visual inspection of the CAN log was also carried out. The observed pattern for fields identified as composite was typical of that shown in Figure 3, i.e. two fields next to each other, the second with a high number of unique values. Visual inspection of the remaining fields did not suggest any missed concatenations, although it is acknowledged that the absence of evidence does not necessarily represent evidence of absence.

The algorithm does not pick up single fields that might be sensor readings. However, from studying the number of unique values and their pattern, some of these were suggested from Car 1, and included in the subsequent clustering process. Also, the algorithm does not compare non-adjacent fields, or check if sensor fields might be composite from three or more fields. These checks could be added to the algorithm, however the visual inspection of the data revealed no observed instances that suggested such fields, and we have not found any reported cases.

5.2. Identifying Associated Fields

Having identified sensor fields, we need to group these into clusters, each of which contain the sensor fields pertaining to associated functions. For the anomaly detection process, the car function that the fields pertain to does not

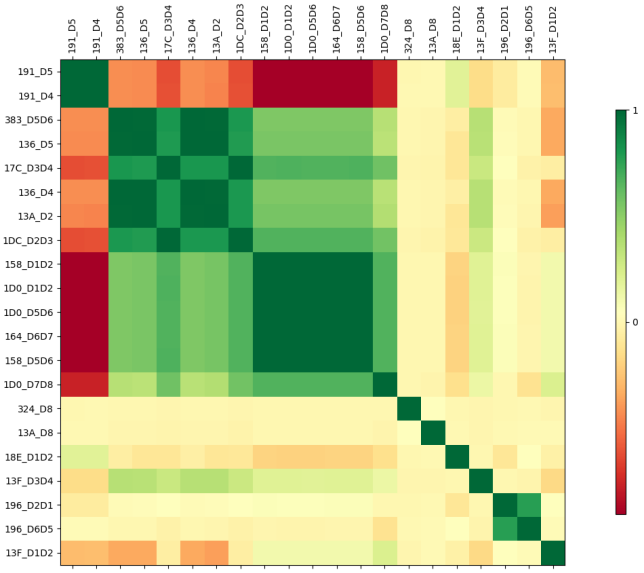

Figure 4: Hierarchically clustered correlation matrix plot for sensor fields for Car 1. The two dark cluster blocks on the diagonal seems to correspond to the individual aspects (accelerator and speed) identified whilst monitoring the CAN data during driving.

need to be known. However, determining the function would provide a validation that the cluster detecting algorithm has worked, as well as help to determine which clusters might be targets for specific attacks. Although the lack of an available dictionary means that the mapping of CAN packet IDs to ECUs, or to car functions, is not available, estimation can be made from analysis of CAN traffic combined with journey records captured on video. Car 2 was not available long enough for a detailed functional analysis, but in Car 1 it was possible to determine fields that change with the application of the accelerator, or with changes in speed regardless of the accelerator position, or as brakes are applied.

Hierarchical cluster analysis was undertaken of the fields identified as sensor data. Pearson's correlation coefficient was used as a distance measurement to cluster the data fields according to the data field correlations. Implementation used the Python pandas .corr() function for the correlations [26], and SciPy .distance, .linkage and .fcluster functions to generate the hierarchical clustering [27]. Figures 4 and 5 show the resulting cluster matrices. The CAN packet ID is shown by the field's prefix, and the data field positions are shown by the D suffix. Composite fields have two D suffix elements. Thus, for example, data field 17C_D3D4 is a composite of the D3 and D4 fields from a packet with CAN ID 17C.

For this study, fields with a correlation ≥ 0.75 were considered as being in a related cluster. The two large darker clusters towards the top left and centre of Fig. 4 comprise fields that the manual analysis suggested were concerned with

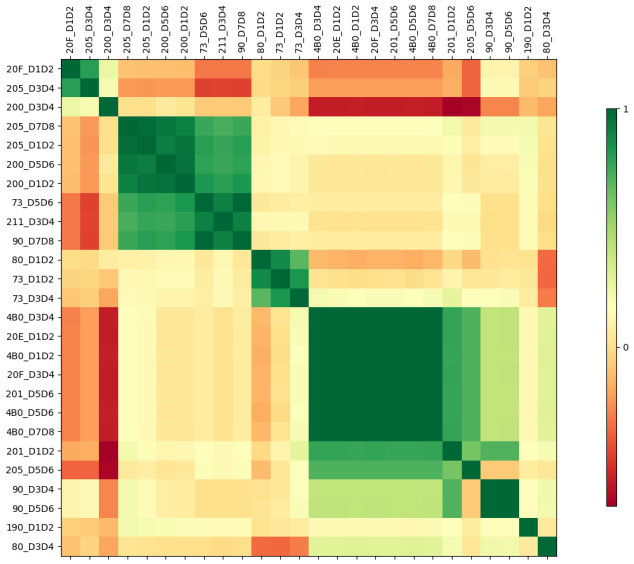

Figure 5: Hierarchically clustered correlation matrix plot for sensor fields for Car 2.

the accelerator and speed respectively. Of course, other than from conducting detailed and comprehensive reverse engineering, or from acquiring the CAN dictionary, there is no way of determining what precisely the fields are measuring. They might be capturing something only very tentatively linked, such as fuel efficiency or a component pressure reading. Even so, the potential use of such correlated fields for determining contextual anomalies holds, no matter the nature of the link. The clusters from both cars that were used for the subsequent experiments are shown in Table 1.

Table 1: Clusters used in the anomaly detection experiments. In Car 1 the cluster names indicate the function the cluster seemed to be associated with. A detailed analysis of car functions was not carried out in Car 2, so the cluster names are not given functional associations.

Car	Cluster	Data Fields
Car 1	Accelerator	1DC_D2D3, 136_D5, 13A_D2, 17C_D3D4, 136_D4, 383_D5D6
	Speed	1D0_D7D8, 164_D6D7, 158_D1D2, 158_D5D, 1D0_D1D2, 1D0_D5D6, 191_D4, 191_D5
Car 2	Cluster A	4B0_D1D2, 4B0_D3D4, 4B0_D5D6, 4B0_D7D8, 20E_D1D2, 201_D5D6, 20F_D3D4
	Cluster B	73_D5D6, 90_D7D8, 200_D1D2, 200_D5D6, 205_D1D2, 205_D7D8, 211_D3D4

6. Experiment Data Generation and Evaluation Criteria

Having identified sensor fields, including those with values derived from composite fields, and determined clusters of associated fields, the CAN log flow can now be parsed into a format suitable for the anomaly detection. Cluster snapshot records were created, with each snapshot record being created upon the

broadcast of one of the CAN packets in the cluster and comprising the most recently broadcasted values for the data fields in the cluster. Snapshot records were created for each of the main clusters identified by the clustering algorithm (i.e. Acceleration and Speed clusters for Car 1, and Cluster A and Cluster B for Car 2).

Fig. 6 shows a subset of fields to illustrate the updating of the snapshot array. 6a shows CAN broadcasts between 27.08628 and 27.18628 seconds for two CAN IDs. Fig. 6b shows the snapshot array state after the broadcast of the CAN packets. The array is updated as each new packet within its CAN ID scope is broadcast. For example, the broadcast of CAN ID 324 at 27.18604 seconds, results in an array update at that time for element 324_D8. Likewise, the broadcast of CAN ID 383 at 27.18628 seconds, results in an update of array element 383_D5D6. Fig. 6b also shows that the first few snapshot rows did not change when packets were broadcast since these packet broadcasts would have contained the same field value as the previous broadcast for the relevant CAN ID.

In a production version, the amalgamated data would not need to be retained after each classification has taken place, so a single, continuously updated array could be used. However, to facilitate test replication, and allow attack data to be simulated, each change to the array was saved as a separate snapshot record, forming a snapshot record file.

TIME	CAN_ID	D8	D5D6
27.08628	324	165	0
27.08653	383	34	561
27.18604	324	180	0
27.18628	383	53	903

(a) Sample from the CAN packet flow. For illustration only two IDs and and two data fields are shown.

TIME	CAN_ID	17C_D3D4	158_D1D2	191_D5	18E_D1D2	1DC_D2D3	324_D8	383_D5D6
27.18481	17C	32	0	55	0	32	165	561
27.18496	18E	32	0	55	0	32	165	561
27.18520	158	32	0	55	0	32	165	561
27.18544	191	32	0	57	0	32	165	561
27.18561	1DC	32	0	57	0	32	165	561
27.18604	324	32	0	57	0	32	180	561
27.18628	383	32	0	57	0	32	180	903

(b) Section of the snapshot array of amalgamated CAN data. Each row shows the change in the data elements over time as each CAN ID is broadcast. The Time and CAN_ID columns are not included in the actual array.

Figure 6: Parsed CAN data field broadcasts (a) transcribed into a rolling data snapshot array (b).

6.1. Experimental Data Sets

Each experimental data set comprised 8000 randomly selected snapshot records for one of the clusters. For each experimental data set, 6000 snapshots

were assigned to a training set, 1000 for an unmanipulated testing set, and 1000 for an attack testing set. The testing sets determined the false positives and true negatives, whilst the attack sets, which were manipulated to mimic attacks (as discussed below), determined the true positives and false negatives.

All sets were scaled using Scikit-learn's unit variance StandardScaler function [28], retaining the scaling parameters obtained from processing the corresponding training set. The StandardScaler function standardises the data by first subtracting the mean value (giving the values a zero mean), and then dividing by the variance, thus giving the distribution unit variance. Standardisation retains the distribution shape of the data, unlike scalers such as Min-Max which can crush values in data sets with outliers [29].

6.2. Attack Manipulation

For each test iteration, one field in the attack set was manipulated to replicate the effects of injection attacks that fabricate data values. Testing was repeated so that each data field was manipulated. One option would be to apply multipliers to each field in the test data set, and then apply the scaling after the manipulation. However, provisional analysis of each car showed that some fields had a range with an origin that did not start at zero. In cases where the range is narrow, yet there is a high range-origin, the outliers produced by this manipulation method would be extreme; so they would be simple to detect, and their resulting manipulated data would probably be rejected by any reading node in any case.

Therefore, to provide a more controlled testing, we first scaled the attack data set using the scaling parameters fitted from the training data set, and then applied the data manipulation to the attacked field by adding an offset after the scaling. The offsets were -1, -0.5, -0.3, -0.2, -0.1, 0.1, 0.2, 0.3, 0.5 and 1, thus increasing or decreasing the scaled data by specific magnitudes. This method produces attack data that, across fields, is more uniform in its magnitude, irrespective of the permitted range-origin for individual fields. Such a manipulation would be more representative of attacks in which the attacker had more knowledge of the data construction and meaning. The manipulation offset range enables classifier performance to be assessed across a broad range of manipulations, including subtle changes retaining plausibility such as suggested by [5].

6.3. Evaluation Definitions

During evaluation, the snapshot records in the test and attack data sets were fed individually to the classifiers, which had been trained using the corresponding training data sets. Each classifier decided whether the snapshot was an anomaly (i.e. a snapshot containing an attacked field) or normal (i.e. a snapshot that contained no attacked fields). For the CC an anomaly corresponded to instances that fell outside all hyperspheres. For the other LOF and OCSVM classifiers, the decision was as reported by the software implementation function.

The analysis employed the following definitions: *True Positive (TP)* – an attacked snapshot identified as such by the classifier; *False Negative (FN)* –

an attacked snapshot that was not identified as such by the classifier; *False Positive (FP)* – a normal snapshot classified as having been attacked; *True Negative (TN)* – a normal snapshot classified as such. F_β scores [30] were used to provide a summarisation of these, taking β=0.5 so as to attach more weight to precision over recall (Equation 1).

$$F_\beta = \frac{(1+\beta^2) \times TP}{(1+\beta^2) \times TP + \beta^2 \times FN + FP} \quad (1)$$

In the results tables shown later only the FP and FN values are presented. TN and TP values can be calculated by subtracting, respectively, FP and FN values from 1000 (the sample sizes).

7. Results and Discussion

Initial testing (Section 7.1) was conducted using the data sets from Car 1 to determine classifier parameter configurations that gave optimal all-round performance. Because of the large volume of results, these initial evaluations were restricted to examining only the averaged $F_\beta(0.5)$ scores for each training parameter permutation. For each parameter permutation, the average $F_\beta(0.5)$ score was taken across all the attack multipliers for all the fields in the clusters. The optimal parameter values were then fixed for subsequent evaluations across of both cars.

7.1. Optimal Parameter Values

The LOF was tested with KNNs between 3 and 100, and contamination between 0.01 and 0.12. The highest average $F_\beta(0.5)$ scores across the training data sets were achieved with KNN 9 and contamination 0.05. The OCSVM was tested with the RBF (Radial Basis Function) kernel and Nu between 0.001 and 0.7 and gamma between 1e-07 and 0.9, with the highest average $F_\beta(0.5)$ scores obtained with Nu 0.015 and gamma 0.5. These optimal LOF and OCSVM values were used for subsequent testing.

The CC was tested with parameter values that were recommended by its author [19], scaled for the standardisation process (i.e. hypersphere size adjustment of 0.0001 * *the number of dimensions*; and hypersphere movement of 0.05). Other parameter values were tried, but slowed the training, and produced no improvement in classification. The CCs were trained until they captured 97% of the training instances for the Car 1 Speed cluster, 96% for the Accelerator cluster, 97% for Car 2 Cluster A, and 95% for Cluster B. The proportion of training instances captured by the hyperspheres plateaued at these rates. The trained CCs replaced the 6000 training data points with 21 hyperspheres for the Accelerator cluster, 8 for the Speed cluster, and 13 each for Car 2 Clusters A and B.

7.2. Baseline Results

The initial testing of the optimised classifiers was conducted with the training data set, test data set, and attack data set all from within the same journeys. For Car 1, the selected journey was approximately 11 minutes long, and for Car 2 it was approximately 8 minutes. This testing presented a gauge of the potential ability of the classifiers. The $F_\beta(0.5)$ score averages by cluster and data field tested for both cars are summarised in Tables 2 to 5. The highest $F_\beta(0.5)$ scores overall, and averaged within fields, were obtained by the LOF, followed by the OCSVM; with the CC being the least successful. Tables 6 to 8 show the average $F_\beta(0.5)$ scores within each cluster for each magnitude of attack manipulation. Table 9 shows the false positive counts for each classifier, while Tables 10 to 12 list the average number of false negatives for each magnitude of manipulation.

Whilst the number of FPs might initially seem low (e.g. less than 50 per 1000 records for each cluster for the LOF, Table 9) they would imply many FPs per second in a car implementation, which would be problematical. Although the rate of FNs is dependent on the magnitude of the data manipulation for the attack, the high rates for the manipulation levels used here would also be of concern, implying a high rate of missed attacks.

As might be expected, $F_\beta(0.5)$ scores are highest at the larger manipulations (the upper and lower rows in each table), and the number of false negatives lowest at these manipulations, reflecting the easier detection resulting from these manipulations. The OCSVM showed the lower rate of false positives; although tended to generate more false negatives compared with the LOF, hence achieved a lower $F_\beta(0.5)$ score.

The LOF achieved fewer false negatives and a higher $F_\beta(0.5)$ score at lower magnitudes of manipulation compared to the OCSVM, but the lower false positive score of the OCSVM gained it a higher $F_\beta(0.5)$ score at the higher manipulations; even though there does not appear to be a large reduction in its false negative scores, compared to the LOF, at these magnitudes.

Cluster B in Car 2 seems to have been especially problematical. Although, similar to the other clusters, the false negatives peak at the lower magnitude manipulations, they remain higher across the manipulation range for this cluster. The reasons for this remain unclear and require further investigation.

Table 2: Mean $F_\beta(0.5)$ scores by field: Car 1 Accelerator cluster (1000 record test data set).

	1DC_D2D3	136_D5	13A_D2	17C_D3D4	136_D4	383_D5D6	All
LOF	0.8772	0.8779	0.8633	0.8793	0.8580	0.8375	0.8655
CC	0.5790	0.5811	0.5821	0.5794	0.5751	0.5745	0.5785
OCSVM	0.6435	0.6679	0.6684	0.6451	0.6804	0.6888	0.6657

Table 3: Mean $F_\beta(0.5)$ scores by field: Car 1 Speed cluster (1000 record test data set).

	1D0_D7D8	164_D6D7	158_D1D2	158_D5D6	1D0_D1D2	1D0_D5D6	191_D4	191_D5	All
LOF	0.9606	0.9619	0.9619	0.9620	0.9620	0.9621	0.9581	0.9571	0.9606
CC	0.4848	0.4971	0.4970	0.4971	0.4967	0.4975	0.5122	0.5137	0.4995
OCSVM	0.7626	0.8192	0.8194	0.8194	0.8205	0.8203	0.8857	0.8459	0.8241

Table 4: Mean $F_\beta(0.5)$ scores by field: Car 2 Cluster A (1000 record test data set).

	4B0_D1D2	4B0_D3D4	4B0_D5D6	4B0_D7D8	20E_D1D2	201_D5D6	20F_D3D4	All
LOF	0.9635	0.9629	0.9632	0.9635	0.9635	0.9635	0.9635	0.9634
CC	0.6006	0.6003	0.6006	0.5995	0.5998	0.5984	0.6000	0.5999
OCSVM	0.9300	0.9176	0.9315	0.9304	0.9303	0.9310	0.9304	0.9288

Table 5: Mean $F_\beta(0.5)$ scores by field: Car 2 Cluster B (1000 record test data set).

	73_D5D6	90_D7D8	200_D1D2	200_D5D6	205_D1D2	205_D7D8	211_D3D4	All
LOF	0.7255	0.7356	0.7554	0.7397	0.7352	0.7222	0.3747	0.6840
CC	0.2787	0.2794	0.2834	0.2801	0.2851	0.2828	0.2772	0.2810
OCSVM	0.3375	0.3342	0.3494	0.3653	0.3871	0.3881	0.2220	0.3405

Table 6: LOF Average $F_\beta(0.5)$ scores by cluster.

Offset	Car 1		Car 2	
	Accelerator Cluster	Speed Cluster	Cluster A	Cluster B
-1.0	0.9472	0.9630	0.9638	0.8762
-0.5	0.9146	0.9628	0.9638	0.7525
-0.3	0.8638	0.9625	0.9638	0.6632
-0.2	0.8452	0.9614	0.9636	0.5860
-0.1	0.7946	0.9542	0.9620	0.5221
0.1	0.7811	0.9540	0.9626	0.5498
0.2	0.8152	0.9609	0.9635	0.5844
0.3	0.8544	0.9626	0.9634	0.6642
0.5	0.8996	0.9629	0.9638	0.7557
1.0	0.9399	0.9630	0.9638	0.8863

7.3. Classification Across Journeys

The capability of the trained classifiers to detect across journeys was explored using journeys captured from Car 1. The rows in Tables 13 and 14 show the journey used for the training data sets. The columns show the two journeys used for the testing data sets (Journeys A and D). Journey A was 11 minutes, C was 82 minutes, and D was 8 minutes. B is a composite training data set, made by combining the data sets from journey A with the data set from an

Table 7: CC Average $F_\beta(0.5)$ scores by cluster.

Offset	Car 1		Car 2	
	Accelerator Cluster	Speed Cluster	Cluster A	Cluster B
-1.0	0.9113	0.8183	0.7982	0.4464
-0.5	0.7899	0.6332	0.7047	0.2653
-0.3	0.5652	0.4186	0.5553	0.2393
-0.2	0.3313	0.3435	0.4908	0.2326
-0.1	0.2922	0.3173	0.4490	0.2256
0.1	0.2878	0.3163	0.4386	0.2159
0.2	0.3657	0.3442	0.4997	0.2259
0.3	0.5609	0.4073	0.5663	0.2358
0.5	0.7568	0.6425	0.6980	0.2640
1.0	0.9245	0.7541	0.7982	0.4588

Table 8: OCSVM Average $F_\beta(0.5)$ scores by cluster.

Offset	Car 1		Car 2	
	Accelerator Cluster	Speed Cluster	Cluster A	Cluster B
-1.0	0.9827	0.9889	0.9944	0.9023
-0.5	0.9619	0.9797	0.9944	0.3949
-0.3	0.7936	0.9198	0.9777	0.1949
-0.2	0.6555	0.8135	0.9341	0.1282
-0.1	0.4526	0.6600	0.8245	0.0954
0.1	0.1314	0.4019	0.6595	0.0932
0.2	0.2345	0.6373	0.9306	0.1288
0.3	0.5315	0.8827	0.9835	0.1906
0.5	0.9302	0.9678	0.9944	0.3927
1.0	0.9827	0.9896	0.9944	0.8843

Table 9: False positives recorded for Car 1 and Car 2 (1000 record test data sets).

	Car 1		Car 2	
	Accelerator Cluster	Speed Cluster	Cluster A	Cluster B
LOF	43	48	47	42
CC	86	185	316	86
OCSVM	22	10	7	16

additional 8 minute journey, E. Because of the length of journey C, it was not possible to generate snapshot records for the complete journey due to the time taken to process such a large log file into a text-output snapshot file. Therefore six 100 second samples were extracted at equal intervals across the journey, including the start and the end of the journey. All journeys included a mixture

Table 10: LOF Average number of False Negatives by cluster.

Offset	Car 1		Car 2	
	Accelerator Cluster	Speed Cluster	Cluster A	Cluster B
-1.0	80	0	0	214
-0.5	192	1	0	415
-0.3	341	2	0	649
-0.2	387	7	1	734
-0.1	488	38	8	787
0.1	507	39	5	768
0.2	445	9	1	732
0.3	366	9	2	645
0.5	236	0	0	456
1.0	98	0	0	195

Table 11: CC Average number of False Negatives by cluster.

Offset	Car 1		Car 2	
	Accelerator Cluster	Speed Cluster	Cluster A	Cluster B
-1.0	96	175	0	813
-0.5	423	553	269	909
-0.3	722	781	548	920
-0.2	879	835	548	923
-0.1	898	852	683	926
0.1	900	853	694	930
0.2	861	835	623	926
0.3	725	790	531	922
0.5	484	540	284	910
1.0	46	540	0	910

Table 12: OCSVM Average number of False Negatives by cluster.

Offset	Car 1		Car 2	
	Accelerator Cluster	Speed Cluster	Cluster A	Cluster B
-1.0	0	14	0	224
-0.5	89	52	0	870
-0.3	525	271	77	950
-0.2	693	515	238	969
-0.1	842	692	501	978
0.1	968	854	712	979
0.2	936	711	236	969
0.3	785	356	52	952
0.5	196	84	0	874
1.0	0	11	0	228

of road types and traffic situations. As with the previous testing, the training data comprised 8000 instances from a random sample from the entire data file. For these experiments the 1000 record test and attack files were fixed, so all classifiers were tested against the same records.

Perhaps unsurprisingly, testing using data from same journey as training often gave the better detection scores. This can be seen in the highest $F_\beta(0.5)$ scores in nine of the twelve columns of data across the Tables. However a deviation to this is seen in the OCSVM for the Speed cluster (Table 14), where the training using journey A gave a higher score for detection on journey D than the OCSVM classifier trained using journey D. Likewise, for both the Accelerator cluster and the Speed cluster, the CC achieved a higher $F_\beta(0.5)$ score for journey A using training data from other journeys (Tables 13 and 14).

Expanding the training data range by combining data from two journeys (A + E), produced mixed results. For the LOF, it resulted in a higher score when detecting from the non-included journey D, although for the OCSVM the performance fell in this situation.

Whilst the sample of results is small, it does highlight that training based on a particular journey cannot be assumed to produce similar detection results across other journeys. Moreover, increasing the range of training data by adding additional journeys, or choosing a longer journey, need not improve the classifier's effectiveness.

Table 13: LOF, CC and OCSVM performance across journeys for the Accelerator cluster. Cells show False Positives, Average False Negatives and $F_\beta(0.5)$.

		Journey used for Test/Attack Data					
		A			D		
		LOF	CC	OCSVM	LOF	CC	OCSVM
Journey used for Training Data	A	35	195	17	423	477	524
		297	555	458	275	384	211
		0.8465	0.5536	0.7502	0.6296	0.5539	0.6219
	B (A+E)	40	193	18	412	597	538
		297	457	465	200	284	214
		0.8393	0.6421	0.7343	0.6709	0.5653	0.6147
	C	257	402	184	446	330	157
		288	421	374	263	524	417
		0.7066	0.5726	0.6985	0.6292	0.5305	0.6761
	D	152	505	496	57	175	17
		639	255	195	562	486	380
		0.5415	0.6104	0.6405	0.6838	0.6067	0.7891

Table 14: LOF, CC and OCSVM performance across journeys for the Speed cluster. Cells show False Positives, Average False Negatives and $F_\beta(0.5)$.

		Journey used for Test/Attack Data					
		A			D		
		LOF	CC	OCSVM	LOF	CC	OCSVM
Journey used for Training Data	A	56	175	14	336	295	34
		208	574	462	258	466	435
		0.8977	0.5792	0.7772	0.6924	0.5948	0.7856
	B (A+E)	58	231	15	177	335	23
		212	674	486	252	533	465
		0.8940	0.4701	0.7482	0.7910	0.5356	0.7610
	C	322	426	98	404	612	133
		260	282	457	298	178	405
		0.6970	0.6298	0.7148	0.6395	0.6027	0.7227
	D	700	907	590	69	465	355
		138	12	150	223	58	216
		0.5932	0.5756	0.6241	0.8801	0.7080	0.6829

7.4. Method Overheads

The maximum times taken by the trained classifiers to assess any of the 1000 snapshot test-runs are shown in Table 15. Added to these times should be the time needed to apply the scaling, which was found to be approximately 0.001 seconds per test run. The maximum observed times for the OCSVM and the LOF to process the 1000 record snapshots, suggests the potential for them to process data faster than the packet broadcast rates. For example, Car 1 broadcast at an average rate of 1432 packets per second, Car 2 at 1652 packets per second, whereas the detection processed the 1000 record files in just a few hundredths of seconds. However, this does not include the time taken to update the snapshot record, and caveats regarding the hardware used here compared with an in-vehicle implementation, need to be considered.

Table 15: Maximum time (seconds) for the trained classifiers to assess the 1000 snapshot test data sets.

	Car 1		Car 2	
	Accelerator Cluster	Speed Cluster	Cluster A	Cluster B
LOF	0.01199	0.02398	0.01799	0.020989
OCSVM	0.00301	0.00400	0.00400	0.004011
CC	2.02783	2.35264	3.31107	3.877810

The methods used here would require the persistent storage of data for the scaling and for the decision surface. Using the Scikit-learn *joblib* functions [28]

for the scaling information and the decision surface for both the LOF and the OCSVM, produced a scaler file of 1KB for each classifier, with a maximum 1850KB file for the LOF decision surface, and 12KB file for the OCSVM. For the CC the decision surface requires the storage of the coordinates and size of each hypersphere, resulting in a 1KB file.

The training time would not impact the detection performance, since training would likely be done at system deployment and not repeated. Even so, times for training using the 6000 snapshot records were recorded. For the LOF, training times for the complete 6000 record file ranged from 0.07 to 0.13 seconds, while for the OCSVM they ranged from 0.05 to 0.08 seconds. Training for the CC took far longer; for most instances between 40 minutes and one hour. Training the CC required resource-intensive, nested, nearest neighbour operations, and was coded entirely by ourselves, so might not have been tuned to its maximum programmable efficiency.

8. Conclusion

This paper presented methods to process CAN packet broadcasts to detect attack-indicative anomalies in the data values, and evaluated one-class anomaly detection methods that might then be applied. This offers the potential detection of attacks that might not alter packet timings, including where the detector lacks knowledge of the CAN dictionary or the attack profile. Once trained, the classifiers have low overheads, giving the potential to run *in situ*.

Automating the identification of composite data fields, and subsequently using correlation coefficients, suggested functional data-field clusters that can be used in anomaly classification. These tallied with clusters determined from manual analysis. We plan to test these algorithms on more cars, and adapt them to determine if data is composed of non-adjacent packet fields.

So far, the results from the classifiers are below what would be acceptable in an actual detector, though the OCSVM and LOF showed superior performance to the CC. However, whilst full accuracy is desirable, it is probably unrealistic, especially in any detection system built without the car manufacturer's CAN dictionary.

Accuracy might be improved by conducting transformations on the data, though this would add processing complexity. Raising the correlation threshold for the clusters might improve classification, but would mean monitoring fewer fields for each decision. Although we attempted to optimise the classifier configurations and kernels, further exploration might determine more optimal settings.

Our study has shown the importance of testing using a broad range of outlier possibilities. Whilst some attacks might produce extreme anomalies, others might be subtle, and might not be assessed if only extreme cases are tested. Our results also show that achieving a trained classifier that can cope across journeys is difficult, and might not be resolved simply by broadening the training scheme.

References

[1] U.S. Department of Transportation / National Highway Traffic Safety Administration, Federal Automated Vehicles Policy, Tech. Rep. September, U.S. Department of Transportation (2016).
URL https://www.transportation.gov/AV/federal-automated-vehicles-policy-september-2016

[2] The Institution of Engineering and Technology (IET), The Knowledge Transfer Network (KTN), Automotive Cyber Security: An IET/KTN Thought Leadership Review of risk perspectives for connected vehicles, Tech. rep. (2015).
URL https://www.theiet.org/media/2309/iet-automotive-cyber-security-tlr-lr-1.pdf

[3] S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, Comprehensive Experimental Analyses of Automotive Attack Surfaces, in: 20th USENIX Security Symposium, Vol. August, The USENIX Association, San Francisco, 2011, pp. 77–92.
URL http://dl.acm.org/citation.cfm?id=2028067.2028073

[4] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, Experimental Security Analysis of a Modern Automobile, in: IEEE Symposium on Security and Privacy, Oakland, CA, 2010, pp. 1–16. doi:10.1109/SP.2010.34.

[5] C. Miller, C. Valasek, CAN Message Injection: OG Dynamite Edition, Tech. rep. (2016).
URL http://illmatics.com/canmessageinjection.pdf

[6] M. Markovitz, A. Wool, Field classification, modeling and anomaly detection in unknown CAN bus networks, Vehicular Communications 9 (2017) 43–52. doi:10.1016/j.vehcom.2017.02.005.

[7] T. Hoppe, S. Kiltz, J. Dittmann, Applying Intrusion Detection to Automotive IT Early Insights and Remaining Challenges, Journal of Information Assurance and Security 4 (May) (2009) 226–235.

[8] A. Tomlinson, J. Bryans, S. Shaikh, H. Kalutarage, Detection of Automotive CAN Cyber-Attacks by Identifying Packet Timing Anomalies in Time Windows, in: Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018, 2018. doi:10.1109/DSN-W.2018.00069.

[9] K.-T. Cho, K. G. Shin, Fingerprinting Electronic Control Units for Vehicle Intrusion Detection, in: T. Holz, S. Savage (Eds.), Proceedings of the 25th USENIX Security Symposium, USENIX Association, Austin, TX, 2016, pp. 911–927.

URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cho

[10] A. Taylor, N. Japkowicz, S. Leblanc, Frequency-based anomaly detection for the automotive CAN bus, 2015 World Congress on Industrial Control Systems Security, WCICSS 2015 (2015) 45–49doi:10.1109/WCICSS.2015.7420322.

[11] H. K. Kalutarage, O. Al-Kadri, M. Cheah, G. Madzudzo, Context-aware anomaly detector for monitoring cyber attacks on automotive CAN bus . Context-aware Anomaly Detector for Monitoring Cyber Attacks, in: Proceedings of the 2019 Computer science in cars symposium (CSCS 2019), no. October, 2019. doi:doi.org/10.1145/1122445.1122456.

[12] B. Groza, P. S. Murvay, Efficient Intrusion Detection with Bloom Filtering in Controller Area Networks, IEEE Transactions on Information Forensics and Security 14 (4) (2019) 1037–1051. doi:10.1109/TIFS.2018.2869351.

[13] S. Fröschle, A. Stühring, Analyzing the capabilities of the CAN Attacker, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10492 LNCS (2017) 464–482. arXiv:9780201398298, doi:10.1007/978-3-319-66402-6-27.

[14] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, R. Poovendran, Cloaking the Clock: Emulating Clock Skew in Controller Area Networks, Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018 (2018) 32–42arXiv:1710.02692, doi:10.1109/ICCPS.2018.00012.

[15] I. Studnia, E. Alata, V. Nicomette, M. Ka, I. Studnia, E. Alata, V. Nicomette, M. Ka, Y. L. A, A language-based intrusion detection approach for automotive embedded network, in: IEEE (Ed.), 21st IEEE Pacific Rim International Symposium on Dependable Computing, Zhangjiajie, China, 2015.

[16] E. S. M. El-Alfy, K. A. Al-Utaibi, Learning mechanisms for anomaly-based intruson detection: Updated review, 2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017 2017-Janua (2017) 1273–1281. doi:10.1109/ICACCI.2017.8126017.

[17] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, A. Mouzakitis, Intrusion Detection Systems for Intra-Vehicle Networks: A Review, IEEE Access 7. doi:10.1109/ACCESS.2019.2894183.

[18] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, S. Zanero, CANnolo: An Anomaly Detection System based on LSTM Autoencoders for Controller Area Network, IEEE Transactions on Network and Service Management 4537 (c) (2020) 1–12. doi:10.1109/TNSM.2020.3038991.

[19] B. G. Batchelor, Colour Recognition, in: B. G. Batchelor (Ed.), Machine Vision Handbook, Springer-Verlag, London, 2012, Ch. 16, pp. 665–694. doi:10.1007/978-1-84996-169-1.

[20] A. Menendez, G. Paillet, Fish inspection system using a parallel neural network chip and the image knowledge builder application, Ai Magazine 29 (1) (2008) 21–28.

[21] B. G. Batchelor, Practical Approach to Pattern Classification, Plenum Press, 1974.

[22] A. Tomlinson, J. Bryans, S. A. Shaikh, Using A One-Class Compound Classifier To Detect In-Vehicle Network Attacks, in: GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, ACM, Kyoto, 2018. doi:10.1145/3205651.3208223.

[23] C. Chio, D. Freeman, Machine Learning and Security, 1st Edition, O'Reilly Media Inc., Sebastopol, CA, 2018.

[24] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander, LOF: Identifying Density-Based Local Outliers, Proceedings of the 2000 Acm Sigmod International Conference on Management of Data (2000) 1–12doi:10.1145/335191.335388.
URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.8948`

[25] S. Omar, A. Ngadi, H. H. Jebur, Machine Learning Techniques for Anomaly Detection: An Overview, International Journal of Computer Applications 79 (2) (2013) 975–8887. doi:10.5120/13715-1478.

[26] PyData.org, Pandas 0.23.4 documentation (2018).
URL `https://pandas.pydata.org/pandas-docs/stable/index.html`

[27] SciPy.org, SciPy 1.1.0 Reference (2018).
URL `https://docs.scipy.org/doc/scipy/reference/index.html`

[28] Scikit-learn, Scikit-learn web site (2018).
URL `https://scikit-learn.org/stable/`

[29] A. Geron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, O'Reilly Media Inc., Sebastopol, CA, 2017.

[30] G. Bonaccorso, Machine Learning Algorithms, 2nd Edition, Packt Publishing, Birminghan, UK, 2018.

Andrew Tomlinson recently completed a PhD in automotive cybersecurity at Coventry University, where he is now a Postdoctoral Research Fellow. His research interests include machine learning and data science.

Jeremy Bryans is Assistant Professor at the Institute for Future Transport and Cities (IFTC) at Coventry University. He specialises in formal methods and verification for software and system design.

Siraj Ahmed Shaikh is Professor of Systems Security and currently the Director of Research at the Institute of Future Transport and Cities (IFTC) at Coventry University. His research interests include insider and stealthy attacks, automotive and transport cybersecurity, and software assurance.