Master Thesis

Using Machine Learning for Intrusion Detection in a Combined Car and Train Monitoring System

Intrusionsdetektion in einem kombinierten Monitoringsystem für Automobil und Bahn mit Hilfe von maschinellem Lernen

by
Mario Freund

Supervisors
Prof. Dr. Andreas Polze, Prof. Dr. Anja Lehmann
Professur für Betriebssysteme und Middleware, Professur für Cybersecurity - Identity Management
Dominik Spychalski
Industrial Cyber Defense GmbH

Hasso Plattner Institute at University of Potsdam

November 3, 2023

Abstract

Automotive and railway vehicles are developing towards an increased digitilization for an improved user experience, optimized maintenance, autonomous driving, and more efficient operation. This also increases the surface for cyber attacks. This thesis deals with a comparison of Controller Area Network (CAN) and Multifunction Vehicle Bus (MVB) and the proposal of a joint intrusion detection system for automotive and railway vehicles based on machine learning. To do so, concrete approaches are investigated for their suitability. A semi-supervised by Hoang and Kim [18] and an unsupervised approach by Jeong et al. [21] are analyzed deeply under the usage of available datasets. It is shown that the approach by Hoang and Kim is not suitable for the objective of this thesis. The unsupervised method X-CANIDS [21] is more effective and is applied to CAN and MVB networks to assess the performance. Masquerade and unstealthy fabrication attacks can be detected with high recall and precision. Other attacks, like suspension attakcs and stealthy fabrication attacks, are not detected well. Furthermore, the popular HCRL dataset [47] is proven to be an inappropriate benchmark for intrusion detection methods and should be replaced by other ones like SynCAN [17] or ROAD [55]. Finally, a joint intrusion detection system is described based on well-defined use cases. It is discussed that X-CANIDS can be well installed in combination with other intrusion detection methods like timing-based or rule-based methods.

Zusammenfassung

Automobil- und Schienenfahrzeuge entwickeln sich in Richtung einer zunehmenden Digitalisierung für ein verbessertes Benutzererlebnis, eine optimierte Wartung, autonomes Fahren und einen effizienteren Betrieb. Dies erhöht auch die Angriffsfläche für Cyberangriffe. Diese Arbeit beschäftigt sich mit einem Vergleich von Controller Area Network (CAN) und Multifunction Vehicle Bus (MVB) und dem Vorschlag eines gemeinsamen Intrusionsdetektionssystems für Automobil- und Schienenfahrzeuge auf Basis von maschinellem Lernen. Dazu werden konkrete Ansätze auf ihre Eignung hin untersucht. Ein teil-überwachter Ansatz von Hoang und Kim [18] und ein unüberwachter Ansatz von Jeong et al. [21] werden unter Verwendung von verfügbaren Datensätzen eingehend analysiert. Es zeigt sich, dass der Ansatz von Hoang und Kim für das Ziel dieser Arbeit nicht geeignet ist und generell für Intrusionsdetektion nur eingeschränkt geeignet ist. Die unüberwachte Methode X-CANIDS [21] ist effektiver und wird auf CAN- und MVB-Netzwerke angewendet, um die Leistung zu bewerten. Masquerade- und offensichtliche Fabrication-Angriffe können mit hoher Wiedererkennung und Präzision erkannt werden. Andere Angriffe, wie Suspension-Angriffe und subtile Fabrication-Angriffe, werden nicht gut erkannt. Darüber hinaus hat sich der beliebte HCRL-Datensatz [47] als ungeeigneter Benchmark für Intrusionsdetektionsmethoden erwiesen und sollte durch andere wie SynCAN [17] oder ROAD [55] ersetzt werden. Schließlich wird ein gemeinsames System zur Erkennung von Intrusionen auf der Grundlage genau definierter Anwendungsfälle beschrieben. Es wird erörtert, dass X-CANIDS gut in Kombination mit anderen Intrusionsdetektionen wie zeit- oder regelbasierten Methoden eingesetzt werden kann.

Acknowledgments

The FINESSE collaborative research project "Vehicle intrusion detection and prevention in a uniform structure for road and rail" on which this thesis is based was funded by the German Federal Ministry of Education and Research under grant number 16KIS1584K.

I want to thank Robert, Kordian, and Katja from the HPI OSM chair for the constructive feedback throughout this work. Thank you as well to Dominik, Markus, and Christoph from INCYDE for insights into the FINESSE project and the support.

Thank you to my computer science teacher in high school for encouraging me to study computer science.

Finally, I owe a huge thank you to all my family and friends who have always supported me, especially my parents and my partner.

Contents

1 Introduction

In the modern world, mobility infrastructure is more and more connected via networks. This trend can be mainly observed within the two main sectors, railway and automotive. Both have several use cases for increased connectivity. For instance, both sectors are developing in the direction of autonomous driving, the optimization of traffic through the processing of real-time data, optimized maintenance, or a more convenient user experience. This development leads to the occurrence of more external and internal interfaces to services [50]. From a security point-of-view, this also implies a bigger attack surface, which threatens security as well as safety.

Security flaws in vehicles were already demonstrated by various publications. One of the most famous works is about the remote hack of a Jeep Cherokee [31]. The authors showed that they were able to establish a remote connection to any vehicle of a certain type in the USA, and with some effort could inject arbitrary Controller Area Network (CAN) payloads. Thus, physical actions could be performed that could potentially harm passengers.

To address those security issues, there are domain-specific standards that regulate and describe the methods of establishing an information security management system and thus a risk management process. The most important standard for Operational Technology (OT) security is IEC 62443 [46]. For railway, there is the derivation TS50701 [36], and for automotive ISO/SAE 21434 [41]. The risk assessment processes of those standards lead to the implementation of security measures depending on the risk that was assessed and the remaining risk that is accepted. For industrial automation and control systems (IACS), usually a low to medium risk can be accepted. However, to evaluate the effectiveness of those security measures, security monitoring detecting possible intrusions is necessary. Successful detection is also a requirement to identify common attacks and attack patterns.

On the other hand, security events need to be detected to be able to react quickly and apply incident response measures. IEC 62443-2-1 directly asks to report events that are relevant for security [46].

One way to generate security-relevant events is the implementation and usage of an intrusion detection system (IDS). According to Lazarevic et al. [25], IDSs consist of several components. There are one or more devices collecting data (sensor), a detector that analyzes data to generate alarms, a knowledge base that provides collected information in a preprocessed form, a configuration device, and a response component to initiate actions from detected intrusions [25]. Lazarevic et al. also formulate some characteristics an intrusion detection system should fulfill. A very central requirement for an IDS is that it should deliver a small false alarm rate while detecting intrusions reliably [25]. Chapter 2 goes more into detail about how to measure the performance of an IDS.

The goal of this work is to provide an approach for a joint intrusion detection system in the context of a combined security monitoring of railway and automotive. Recent publications such as [50] show that from abstract point-of-view railway and automotive have

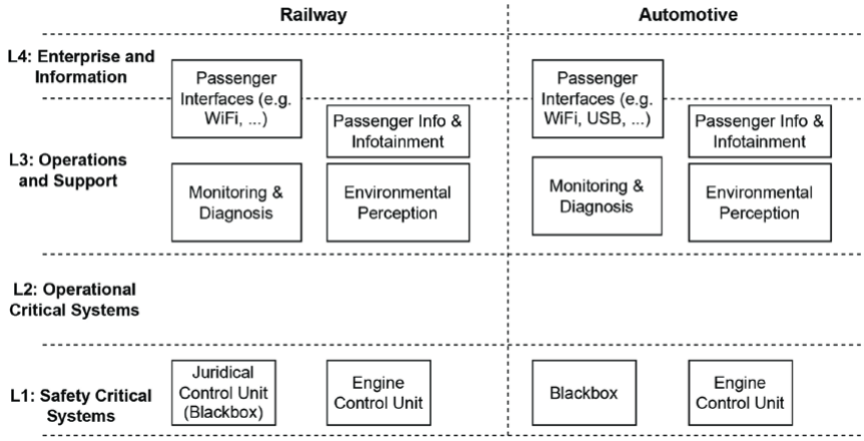


Figure 1.1: This is a layer model comparing rail and automotive vehicles based on IEC62264. Especially on layers three and four, railway and automotive vehicles have many similarities like the implementation of passenger interfaces. Source: [50]

several similarities despite railway being a non-individual, guided way of transportation and automotive being an individual way of transportation.

Figure 1.1 shows a schematical model based on IEC62264. As can be seen, railway and automotive vehicles share the underlying basis of having Engine Control Units respectively Electronical Control Units (ECUs) that are crucial for operating the vehicle. On higher layers, there are systems for monitoring and diagnoses alongside systems for environmental perception (L3) and interfaces for passengers (L3/L4). Recent developments also led to the installation of passenger infotainment systems with increased functionality.

Internally, different technologies are used for communication. In the railway sector, the combination of Wired Train Bus (WTB) and Multifunction Vehicle Bus (MVB), which were originally introduced in the 1990s, dominate [50]. Next to that, also CAN, Profibus, or Train Communication Network (TCN) are installed on trains [50]. Communication technologies for internal train communication are standardized in IEC 61375 [12]. Nowadays, other technologies like Real-Time Ethernet or Ethernet Train Backbone (ETB) are supported by this standard as well. In the automotive sector, the most important bus technology is CAN. However, new vehicles use Automotive Ethernet as well. Those different technologies have different characteristics but generally serve the same purpose within the vehicle.

Externally, both domains share the property of being connected to other vehicles or operational infrastructure. Those remote connections mainly rely on mobile communication systems like GSM-R or soon the new standard Future Railway Mobile Communication System (FRMCS), which is based on 5G. For internet connections of automotive vehicles, usually Long Term Evolution (LTE) is used [50]. Finally, as Spychalski et al. claim, the risk management processes of both domains have similarities taking their corresponding standards TS50701 and ISO/SAE 21434 into account [50].

Based on these facts, it seems valuable to have a joint security understanding of both sectors. This is why a project consortium around Escrypt, Fraunhofer SIT, INCYDE, University of Passau, and Yekta IT was formed in 2021. Their goal is to realize “Intrusion

Detection and Prevention in a Uniform Structure for Road and Railway" [45]. So far, this project dealt mostly with the question of how to aggregate and exchange data from and between multiple vehicles and how to define a uniform architecture.

The goal of this thesis is to start from the other side and evaluate concrete intrusion detection approaches for in-vehicle networks. In scope are the two field buses CAN and MVB representing the automotive respectively railway sectors. Since the mid-1990s, almost every train of Bombardier and Siemens has been equipped with MVB [44]. That is why this protocol is featured in this thesis. CAN is mainly inspected in this thesis because it is still widely used today. For instance, many cars are still in use having a CAN implementation on board. Although automotive ethernet is used as well, CAN still finds its place in manufacturing today as can be observed by the development of CAN XL starting in 2018 [2]. Furthermore, machine learning is leveraged for this work as one way of conducting intrusion detection. Machine learning is a subdomain of artificial intelligence (AI) and has broad usage in multiple fields. The general idea is that machine learning models can learn the communication patterns of the used network protocols and by that can separate between benign network traffic and intrusions. The central research question of this work is which machine learning approach of intrusion detection is suitable to realize a joint IDS for railway and automotive. Besides, the goal is to develop concrete ideas on how to initialize, deploy, update, and maintain the system. It is also necessary to discuss weaknesses of the chosen intrusion detection approach and find solutions how to mitigate these respectively to discuss what to add to overcome these issues. The structure of this thesis follows all the necessary considerations to answer the research questions.

1.1 Structure of the Thesis

After the introduction, Chapter 2 introduces all the necessary foundations for this work. These include intrusion detection and collaborative intrusion detection, machine learning and special types of machine learning models, and the two bus protocols CAN and MVB. For the structured discussion of the joint IDS, use cases for such a system are defined that are filled with concrete ideas later. Related work is presented in chapter 3. It contains a brief overview of recent publications about intrusion detection on CAN and other relevant in-vehicle networks. Moreover, the project FINESSE and further topics like distributed intrusion detection systems and intrusion detection with machine learning are presented. Chapter 4 starts with a comparison of CAN and MVB to justify joint security monitoring. Furthermore, it explains why CAN and MVB are not protected against attacks on confidentiality, integrity, and availability. An attacker model is described and justified based on recent publications before concrete attacks are derived from this attacker model. Finally, the chapter introduces the datasets that were used for the evaluation of the approaches. Chapter 5 presents the overall method for choosing the IDS approach based on considered challenges. After that, the evaluated intrusion detection approaches are explained. Finally, the implementations and the experimental setup are introduced. Chapter 6 contains the evaluation of the approaches respecting the available datasets while chapter 7 discusses the achieved results critically regarding the introduced challenges and the context of a joint IDS. After that, concrete ideas for the use cases of the joint IDS are

introduced next to an overall architecture of it. The thesis is summarized in chapter 8 and conclusions are made considering the research questions.

1.2 Contributions

This thesis delivers several contributions. First, an attacker model for attacks on in-vehicle networks of railway and automotive vehicles is proposed. Already existing attack classes from CAN are applied to MVB to derive a joint security understanding based on a prior comparison of both field buses. Second, a semi-supervised intrusion detection method by Hoang and Kim [18] and an unsupervised detection method (X-CANIDS) by Jeong et al. [21] are evaluated using public CAN datasets and a proprietary MVB dataset. The evaluation results are used to discuss the feasibility of the approaches for a joint intrusion detection system. Third, this thesis provides an implementation of X-CANIDS and an adaption to MVB (X-MVBIDS) for experiments. The implementation is available on GitHub [14]. Finally, use cases for a joint intrusion detection system are formulated and leveraged to propose an architecture. The evaluation analysis is used to identify the strengths and weaknesses of the chosen approach and to name possible mitigations.

2 Foundations

In this chapter, some important foundations are introduced. It starts with a taxonomy of intrusion detection and collaborative intrusion detection. Next, important basic concepts for machine learning, the relation between machine learning and intrusion detection, and special machine learning concepts are presented. The chapter ends with a description of CAN respectively MVB, and use cases for a joint IDS.

2.1 Intrusion Detection and Collaborative Intrusion Detection

According to the National Institute of Standards and Technology (NIST), intrusion detection "is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network" [3]. Using this definition, Lazarevic et al. [25] define three core demands for IDSs. First, the prediction method should classify intrusions as malicious and benign behavior as not malicious. This implies an as high as possible number of true positives and as high as possible number of true negatives. Table 2.1 shows the evaluation of intrusions that is commonly used. Most importantly, intrusion detection can be seen as a binary classification.

Second, the time performance is determined by the processing time and the propagation time, each of which should be low. Processing time means the time until the IDS can generate a security event out of incoming data and propagation time means the time until the security event is available for further inspection and reaction [25]. Third, Lazarevic et al. demand fault tolerance. For instance, the IDS should be able to recover quickly from attacks on itself or a huge number of misleading alarms [25].

Figure 2.1 shows the taxonomy used by Lazarevic et al.. Five aspects are relevant to describe an IDS. The information source is self-explaining. Here in this work, the focus is on network-based intrusion detection. As for the analysis strategy, they define two major methods. Anomaly detection aims to identify a pattern deviation from the normal behavior of the system, the network in this case. Misuse detection tries to match the behavior with certain malicious patterns that are known prior. So, those two methods

		Predicted label	
		Normal	Intrusions (Attacks)
Actual label	Normal connections	True Negative (TN)	False Positive (FP)
	Intrusions (Attacks)	False Negative (FN)	True Positive (TP)

Table 2.1: The table shows evaluation classifications of IDSs. Source: [25]

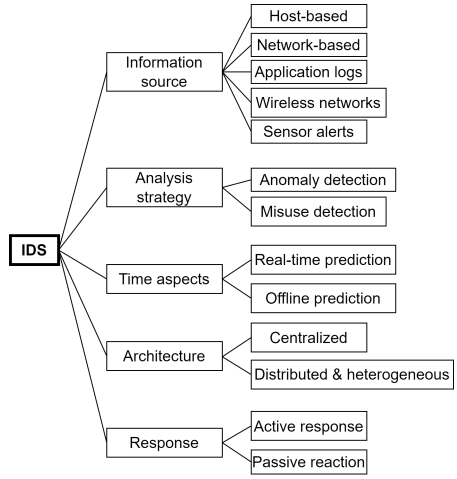


Figure 2.1: The figure shows a taxonomy of IDSs. Source: Replicated diagram from [25]

are inverse to one another on an abstract level. Time aspects differ between real-time and offline detection. Offline detection means that logs are stored first and then analyzed at a later point in time [25]. Architectural considerations include a centralized and a distributed respectively heterogeneous approach in the taxonomy. Finally, the response can include active responses and passive reactions [25]. The response is not considered in this thesis. Going more into detail about the collaborative aspects of intrusion detection, Vasilomanolakis et al. [54] differ between a centralized, decentralized, and distributed architecture. The centralized architecture has one central analysis unit, to which several monitors are connected to. Following their definition, the central unit can either receive logs from the monitors or already raised alarms. Additionally, the architecture does not scale with more monitors. The one central analysis unit stays and eventually results in a performance bottleneck [54]. A decentralized architecture splits up the central analysis unit into multiple parts, which can result in a hierarchical structure. Additional analysis units share the workload and by that mitigate the bottleneck problem [54]. A distributed collaborative IDS (CIDS) unifies monitoring and analysis and hence, each entity is monitoring as well as analyzing. Intrusion detection becomes a peer-to-peer task in this case [54].

2.2 Machine Learning

The following section gives the most important foundations for machine learning. This thesis leverages machine learning and there are experiments in order to choose a suitable machine learning approach for the given problem. However, to fully understand machine learning and its concepts, a book, e.g., by Zhang et al. [59] is recommended.

2.2.1 Basic Considerations

Starting, the most important prerequisite for machine learning is data. Usually, to describe this data, there is a certain number of features. For instance, to describe a pixel on a screen,

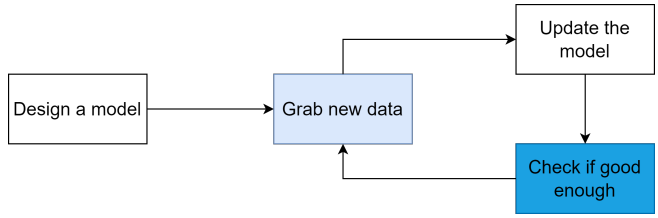


Figure 2.2: The figure shows the basic circle of training a machine learning model. Source: Replicated diagram from [59]

the red, green, and blue values of that pixel can be used. In this case, the data has three features [59]. Furthermore, a dataset might consist of more than one instance of those data points. These examples can be called samples. In order to train a performant machine learning model, it is important to have an appropriate feature set and enough samples [59]. Both requirements pose a challenge in real life.

A machine learning model can be seen as the computing part between input data and output prediction that are related to a certain prediction task. Normally, the input gets passed as some kind of numerical vector. The output of a model depends heavily on the machine learning task and the type of the model. Machine learning has two modes of application: training and prediction. During training, the model gets fed with samples. Based on the computed output, the model is adjusted. In the next round, more samples get fed and the model gets optimized again. This circulation represents an incremental process, in which the model gets better and better at fulfilling the task. Figure 2.2 shows this basic circle.

During prediction, the model is used to compute outputs from prior unknown data. Depending on the success of the training phase, the performance is better or worse. To serve this separation of training and prediction, the data is split accordingly. The minimum split is a split into training and test data. Additionally, validation data can be split to monitor the performance of the model on independent data during training.

In order to check for the correct model output respecting a certain task and updating the model accordingly, labels are required. Label means the expected output of a model for a certain input in this context. In real life, the presence of labeled data is rare respectively lots of resources need to be used to obtain labeled data [59]. This is why there exist three different relevant types of learning: supervised learning, semi-supervised learning, and unsupervised learning.

Supervised learning assumes that there is a completely labeled dataset that can be learned with. Zhang et al. differ between six different kinds of supervised machine learning tasks [59]: The first type is the classical regression problem. There, a fixed set of features is used to estimate a certain attribute of data. An example can be to estimate the price of a car depending on the extra features added to the basic car. The second type is the classification. In this case, a fixed number of classes are given, and the input sample is matched with one of those classes based on the features. A very classic example is to classify photos as cats or dogs. The third type of supervised machine learning task is tagging which is a special case of classification that can match an input with more than one class. Thus, the input gets labeled with potentially more than one tag. The

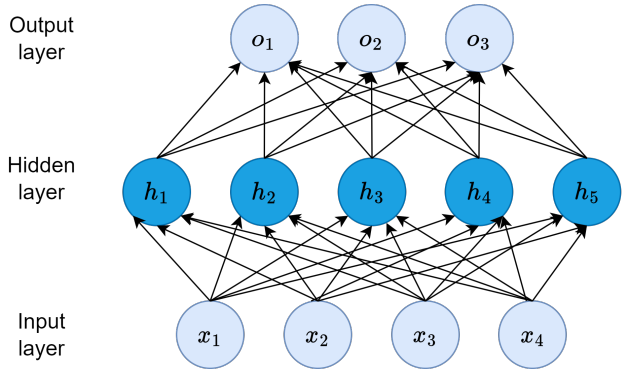


Figure 2.3: The figure shows a basic multilayer perceptron. An arbitrary number of hidden layers can be used. The sizes of the input layer, hidden layers, and output layer can differ from each other. Source: Replicated diagram from [59]

fourth type is the search. Particularly, the task is about assigning a relevance score to result items based on a search item. The most prominent example might be the page rank algorithm by Google. As the fifth type, the recommendation task is like the search task, the only difference being that based on the user personal preferences are considered for the ranking. For instance, this task applies to movie recommendations. The last type is sequence learning. It is a special type of learning where the input and output sequence can be of variable length. An example could be machine translation [59].

Unsupervised and semi-supervised approaches apply for situations where there is no or not enough labeled data available. Unsupervised training can for example be reconstruction learning. In this case, the model tries to reconstruct the input. The challenge here is not to copy the input into the output but to learn the patterns of the data to reconstruct unknown data well. Another example could be word embeddings into a vector space [59]. Semi-supervised training is not explicitly mentioned by Zhang et al. [59]. However, van Engelen and Hoos describe semi-supervised learning as a way of using unlabeled as well as labeled data for training [53]. Furthermore, they present a taxonomy including the two main types of semi-supervised learning [53]. Inductive semi-supervised methods usually use the few existing labeled samples to train a supervised model and include the unsupervised data in some form. Often, pseudo-labels are created for the unlabeled data by obtaining the labels from the supervised model. In contrast, transductive learning does not deliver a model, but a label for the unlabeled data. This could for example be done by clustering the data and looking at which other labeled data points the unlabeled data points are close to [53].

Machine learning approaches featured in this thesis are deep learning approaches. This means that a model is used which has one or more hidden layers. In general, such a multilayer perceptron can be understood as in figure 2.3. Other model architectures, inputs, and outputs exist. However, this is the easiest architecture to understand concepts behind deep learning.

Input features of a sample get fed into the input layer. The input can be understood as a matrix $X \in R^{n \times d}$ for n samples with d features. The hidden layers have several units that are connected to the input layer or previous hidden layers. In the easiest case, the model

is fully connected, i.e., each neuron is connected to each of the previous neurons in the previous layer. The neurons have weights w and biases b. In the example from [59] there is one hidden layer and one output layer. The matrices get multiplied with the corresponding weights and biases get added on every layer. Finally, an output matrix is computed in the output layer [59]. In order to add a non-linearity to the model, each neuron is or is not activated by an activation function. The most famous activation function is probably the rectified linear unit (ReLU) with $ReLU(x) = max(x, 0)$. There are also many other activation functions, and this has been a research object until today [59].

A common way to begin training is to initialize all weights and biases randomly. During a training round, also called an epoch, a batch of training data is used as input for the model. The output of the model would most likely be different from the expected output. This difference can be determined by a loss function. One example of a common loss function is the mean squared error function (MSE), which calculates the squared distance between data points: $MSE = \frac{1}{N} * \sum (y_i - \hat{y_i})^2$. In this case, y_i is the predicted value and $\hat{y_i}$ the real value, while N is the number of samples.

The essence of every model is to update the weights and biases towards a lowered loss function as the goal is to lower the loss function. In order to do so, the derivative of the average loss is calculated. The derivative with respect to the weights and biases returns the gradient. Thus, in order to lower the loss, the parameters must be updated in the direction of the negative gradient [59]. This is achieved by multiplying the gradient by a small value η, also called the learning rate, and subtracting it from the current parameters [59]. A higher learning rate implies bigger steps and faster learning. However, learning too aggressively can lead to the miss of the local or global minimum of the loss function and finally to a high loss. The learning rate is a hyperparameter that needs to be tuned [59]. The just-described optimization algorithm is called "Minibatch Stochastic Gradient Descent" and is one of the most popular ones. Numerous other algorithms exist. However, the general principle of training stays the same:

1. Take a batch of data and calculate the inference of the model.
2. Calculate the loss with respect to a chosen loss function.
3. Calculate the gradient using the so-called backpropagation algorithm.
4. Use an optimization algorithm and calculate a parameter update towards a smaller loss function.
5. Update the weights and biases accordingly.
6. Take the next batch and start again.

Machine learning concepts that are under inspection in this thesis are explained later in this chapter.

2.2.2 Special Considerations for Intrusion Detection

The previous subsection introduced some fundamental concepts of machine learning. Sticking to this terminology, intrusion detection can be understood as a binary classification task. Furthermore, supervised, semi-supervised, and unsupervised approaches are possible in order to train the classifier. Throughout this work, certain metrics are

Name	Calculation	Comment
Accuracy (ACC)	$\frac{TP+TN}{FP+FN+TP+TN}$	This metric can be misleading as usually intrusions are underrepresented in the data. A classifier classifying everything as benign might therefore have a high accuracy despite not performing.
Error (ERR)	$\frac{FP+FN}{FP+FN+TP+TN}$	This can suffer from the same problem as above.
True Positive Rate (TPR) or Recall (REC)	$\frac{TP}{FN+TP}$	A high recall is important for intrusion detection systems. It states how many attacks can be detected.
False Positive Rate (FPR)	$\frac{FP}{FP+TN}$	Too many false positives might lead to the infeasibility of the intrusion detection approach.
True Negative Rate (TNR)	$1 - FPR$	This metric is complementary to FPR.
False Negative Rate (FNR)	$1 - TPR$	This metric is complementary to TPR.
Precision (PRE)	$\frac{TP}{TP+FP}$	A high precision means a relatively low number of false positives.
F1 score (F1)	$2 * \frac{PRE*REC}{PRE+REC}$	This is a combination of precision and recall. In practice though, precision respectively recall could be more important than the other one.

Table 2.2: The table shows important performance metrics for binary classification. Source: [38]

required to measure the performance of classifiers. Earlier in this chapter, the concept of true negatives, false negatives, true positives, and false positives was already introduced. Every prediction that is conducted by an intrusion detection system can be put in one of those categories. The resulting confusion matrix is the basis for the following metrics in table 2.2. More metrics exist [38], but usually those metrics are enough to inspect the performance.

2.2.3 Autoencoder

After the general introduction of machine learning and its application in the field of intrusion detection, some more detailed concepts are introduced briefly. The detailed mathematical background of every concept would exceed the scope of this thesis but can be studied in the referenced literature. These sections are meant to explain the functionality and the sense of each concept.

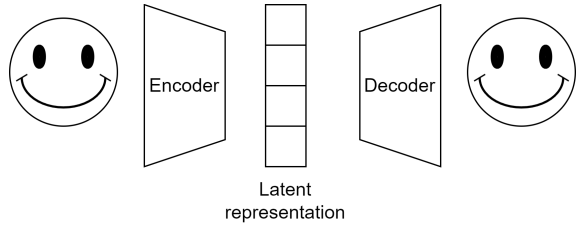


Figure 2.4: The figure shows an abstract representation of an autoencoder.

The first important one is called an autoencoder and is generally used for reconstruction learning. Koenigstein et al. define autoencoders as "[...] type of algorithm with the primary purpose of learning an 'informative' representation of the data that can be used for different applications by learning to reconstruct a set of input observations well enough" [4]. Concretely, autoencoders get a set of input features and compress them into a latent representation of these. The first part doing exactly that is called an encoder. The second part called the decoder, tries to decompress the latent representation to the original features [30].

Figure 2.4 shows an example of an autoencoder. An image, the smiley, in this case, is the input of the autoencoder. It can be represented by its pixel values. The autoencoder encodes the input to a latent representation and decodes it again to the original size. The goal is to reconstruct the smiley. The autoencoder generalizes well when it learns the general patterns of a smiley and reconstructs prior unknown smileys. In order to achieve a meaningful representation, the dimensions of latent spaces are usually much lower than the original dimensions [30].

Encoders and Decoders can have any structure. However, in the scope of this thesis neural networks, specifically deep learning networks, are inspected. Autoencoders can be trained as any other neural network. The central question is how to calculate the reconstruction error of the autoencoder. Recall that the mean squared error was introduced earlier in this chapter. An obvious choice is to use this loss function to measure the distance between the input and output representation of the autoencoder and use optimization algorithms based on that loss function. The two approaches inspected in this thesis both leverage the mean squared error.

Regarding the training of autoencoders, there is no significant difference between them and other neural networks. However, the size of the latent space is an interesting hyperparameter that can have a huge impact. The larger the latent space is, the better the reconstruction of the original input works. On the other side, a too-large latent space can also lead to overfitting to the training examples and thus the model would generalize poorly.

2.2.4 Generative Adversarial Networks

Generative adversarial networks (GANs) were originally introduced in 2014 by Goodfellow et al. [15]. They consist of two parts: the generator, which tries to synthesize data following a certain distribution, and the discriminator, which tries to differentiate between the synthesized data and real data. Both parts work in an adversarial way. This means that

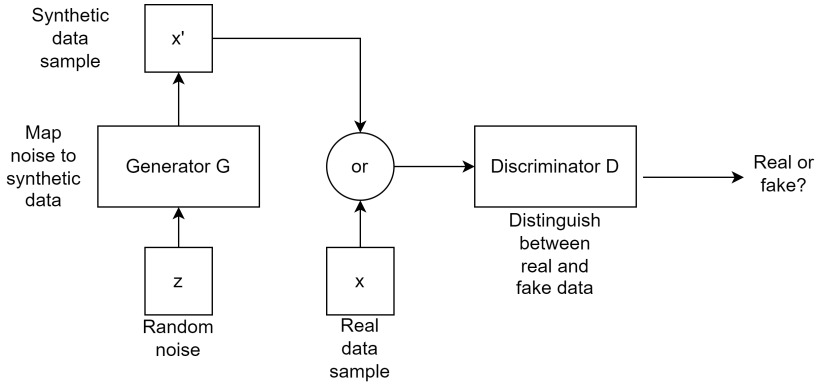


Figure 2.5: The figure shows an abstract representation of a GAN. The generator tries to create synthetic data samples according to a certain distribution. The discriminator tries to differentiate between the synthetic data samples and real data samples. Source: Replicated diagram from [10]

the discriminator (D) is supposed to get better and better at distinguishing real data from synthetic data and the generator (G) is supposed to get better and better at generating synthetic data that matches the distribution of the real data [10].

The concept from above is represented by figure 2.5. In practice, the networks D and G are usually realized by fully connected or convolutional networks [10]. The training follows the structure of having two alternating phases: First, the discriminator is trained to put out the probability of its input to be synthetic data or real data (represented usually by zero and one). The weights are updated via backpropagation and optimization to learn to differentiate between these two classes. Second, the weights of the discriminator get fixed, and the weights of the generator are connected to the discriminator. Backpropagation and optimization against the output of one (real data) are conducted. By that, the generator learns to output data following the distribution of the real data. The goal is to optimize the generator, so the discriminator is not able to differ between real and synthetic data anymore. In practice, this optimal state is usually not achieved [10]. After that, the generator can be used to create artificial data samples that match a real distribution.

2.2.5 Sequence Learning with LSTMs

Certain machine learning tasks might pose the challenge of learning the sequential or temporal relations between data points. Such sequence learning tasks as introduced before require special types of neural networks. Instead of single data points, sequences of data points are passed to the input layer. These neural networks are called recurrent neural networks (RNNs) and are different from normal neural networks in a way that they are connected to each other within one layer and can have an internal state that can be understood as a memory [43]. Salehinejad et al. provide a useful overview of the different types of RNN [43].

As can be identified in figure 2.6, the simplest form of an RNN is a sequential composition of an input layer, a hidden layer, and an output layer. For each timestep, the input contains N features. The hidden layer has M units per timestep and the output layer has P units per timestep. It is important to mention that these are not necessarily the same,

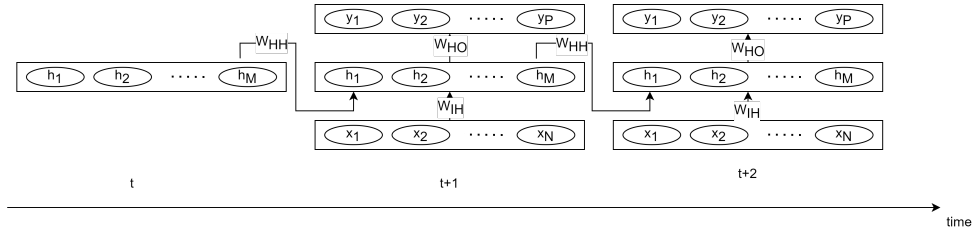


Figure 2.6: The diagram shows an abstract representation of an unfolded RNN. The first hidden state on the left indicates previous timesteps that are not included in the diagram. Each hidden state has a forward connection to the next hidden state. Other characteristics are similar to multilayer perceptrons. Source: Replicated diagram from[43]

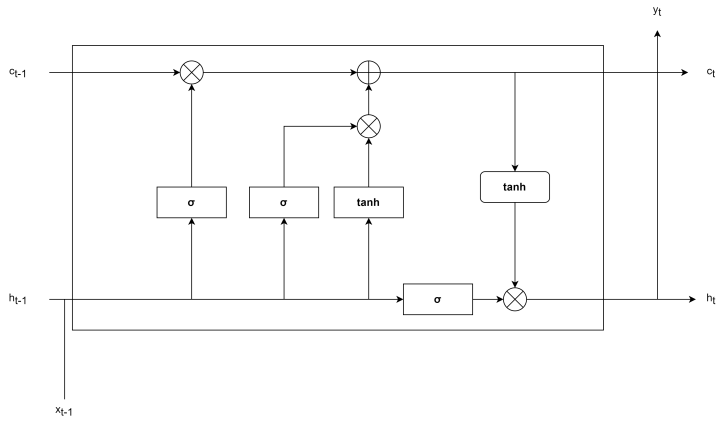


Figure 2.7: The diagram shows an abstract representation of an LSTM cell. Rectangles denote layers and round forms element-wise operations. The "x" indicates a product. The "+" indicates a sum. The top line marks the cell state. The bottom line marks the hidden state. From left to right there is the forget gate, the input gate, and the output gate. Source: Own diagram based on [28, 43, 34]

thus N, M, and P can be different. Similarly to other fully connected networks, weight matrices are applied to the connections between the layers, and biases are added. The hidden layer is connected across the timesteps in order to fulfill the memory function [43]. This architecture implies that the hidden states are calculated with respect to the previous hidden state and the current input. The backpropagation that is used for neural networks is executed as backpropagation through time in this case, where error signals get propagated backward through time [43].

As Bengio et al. state, it is a common problem for standard, simple RNNs to suffer under vanishing gradients [5], which leads to the problem that the RNN ignores long-term dependencies [43]. This is also caused by using standard nonlinear activation functions like sigmoid [43]. On the other hand, exploding gradients can occur that increase the training loss drastically [5].

Due to the above difficulties, new forms of RNNs have been developed over time. One of the most popular ideas is using Long Short-Term Memories (LSTMs) as hidden layer units.

In addition to the hidden state of the unit, a cell state is added. Figure 2.7 shows these states coming in from the previous LSTM cell and going out to the next LSTM cell. The hidden state h_t can also serve as output y_t if desired. This depends on the task. From left to right, the first part is the forget gate [43]. In the LSTM cell, the cell state can be seen as a long-term memory. By using the sigmoid function there (σ), certain parts of the cell state can be erased as the sigmoid function returns values between zero and one and thus can give weight to the components [34]. The middle part of the cell is the input gate which regulates what parts of the information are put into the cell state to update it. The sigmoid function determines again which parts of the information will be passed to the cell state. The tanh function calculates the new values that are put through to the cell state [34]. The last part is the output. It regulates what is put from the cell state to the hidden state by calculating the update values. The mechanism is the same as for the input gate, just the other way around. [34]. All value updates happen concerning the weights and biases of the connections which are trainable parameters.

In total, these enhancements of the hidden unit reduce the problem of exploding and vanishing gradients and lead to the longer preservation of "memories". On the contrary, they have many more parameters than simple RNNs and thus take more resources for training and inference [43].

The most important machine learning concepts are introduced now. More considerations respecting the concrete intrusion detection approaches follow in chapter 5.

2.3 Protocols

The following section provides the necessary foundations for CAN and MVB. To do so, the most important aspects of both protocols get summarized from their standards. Here, the focus is more on the data formats and the way the protocol manages data exchange than on the physical bus access. Notably, in the intrusion detection literature, there is a disbalance between the amount of content for CAN and the amount of content for MVB. There is way more content available for CAN than for MVB. This mainly results from the fact that CAN experiments and attacks have been conducted mainly on cars so far. Cars can be bought and maintained individually, and it is therefore easier to inspect these and do experiments. There is also a public repository where users share reverse-engineered CAN signal translations of their models [9]. Railway vehicles on the other hand are usually maintained by companies for public or goods transport. Experiments cannot be conducted easily but need to be organized together with those companies. Also, MVB datasets are rarely publicly available as the vehicle configurations are proprietary.

2.3.1 Controller Area Network

CAN was originally introduced in 1986 by Bosch GmbH [56]. The version 2.0 specification was released in 1991 [49]. It was originally designed to be used for networks of "automotive electronics, engine control units, sensors, anti-kid-systems" [49], and more. Precisely, the idea to develop this bus came up for the internal connection of Mercedes-Benz cars [56]. It can reach a data transmission rate up to 1Mbit/s [49] and ensures fast, reliable connections. Also, embedded devices can make use of CAN as it is lightweight and

provides little overhead [56]. CAN is standardized in the international standard ISO 11898 [40].

CAN is located on the physical respectively link layer of the OSI reference model. It has some basic properties [49]. First, it guarantees the prioritization of messages. This is important for safety-critical applications as some bus member's data might be more important than the data of others. Second, it guarantees certain latency times. This serves the reliability as well. Third, it has error detection and handling. Fourth, it has automatic retransmissions after collisions or errors. Additionally, there are some other properties like configuration flexibility or system-wide data consistency [49]. CAN follows the multi-master principle. That means that any node connected to the bus can send data if the bus is free [49]. In case there is a collision on the bus, CAN has an arbitration mechanism, which takes the identifier of each of the colliding messages and compares them bitwise. The one that has the lower bit first prevails. Using that mechanism, a priority hierarchy can be established. Furthermore, the identifier is used to identify the content of the sent message. CAN can be operated in different topologies like line topology, ring topology, or star topology.

2.3.1.1 Message Transfer

Messages are referred to as frames. The most important frame is a data frame.

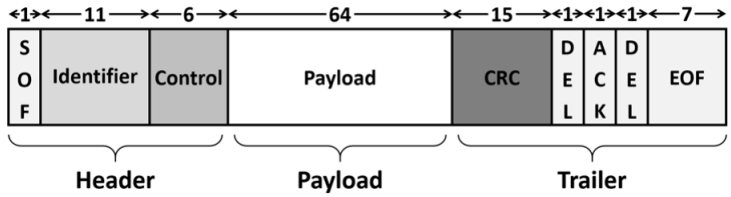


Figure 2.8: The diagram shows a simplified structure of a CAN frame. Source: [24]

Figure 2.8 shows the structure of a data frame in simplified form. Its purpose is to ship data from a transmitter to several receivers. Data frames can be received by multiple receivers, which aligns with the multicast characteristic of CAN [49]. The data frame starts with a leading dominant bit, the start of frame (SOF). The transmission system of CAN understands a logical zero bit as dominant and a logical one bit as recessive. After that, the identifier (ID) follows. In the CAN 2.0A specification, the length is 11 bits. However, in CAN 2.0B, an extended identifier with 29 bits is possible. This allows for more bus participants [49]. Not shown here in figure 2.8 is the Remote Transmission Request Bit (RTR) that must be set to 'dominant' for a data frame [49]. Subsequently, there is the control field with six bits. It consists of the data length code (DLC) and two reserved bits for future extensions. The data length code determines the byte length of the payload. The payload or data field can contain up to eight bytes. Finally, there is the trailer of the frame which contains a Cyclic Redundancy Code (CRC) for error detection, an acknowledgment field, and the end of frame (EOF). The EOF is defined by seven recessive bits. There are also delimiters between some fields. The acknowledgement field is used by a receiver to confirm that it has received the message correctly. It does so by overwriting the recessive ACK bit by a dominant ACK bit.

Next, there is also the possibility of actively asking for data via a remote frame. The remote frame has a similar structure compared to the data frame with the difference that there is no data field. The control field contains the data length of the requested data [49].

2.3.1.2 Error Handling

Finally, there are two more types of frames which are the error frame and the overload frame. Error frames are used to indicate the detection of one or several errors like a bit error, stuff error, CRC error, form error, or acknowledgment error [49]. Error frames are necessary to trigger a retransmission of faulty frames, and by that ensure the system-wide data consistency [49]. Overload frames can be used to indicate internal conditions of a bus member that cause a delay of the next data or remote frame [49].

In total, CAN is a reliable, lightweight, and performant bus system that can be used in many domains. In chapter 4, examples of datasets are discussed. It also makes clear which kind of data is sent in the case of cars.

CAN was developed at a time when automotive vehicles were closed systems. However, as pointed out later, CAN does not serve any IT security objective like confidentiality, integrity, or availability. Conditions of the 1990s do not hold in today's world anymore, especially regarding interfaces to the outside world.

2.3.2 Multifunction Vehicle Bus

MVB is a field bus for data communication that was originally developed in the 1990s by a project consortium of Siemens and other manufacturers. It is specified in IEC 61375-3-1 [12] as part of the Train Communication Network (TCN). TCN specifies a general communication system network for railway vehicles with one or more parts. MVB serves as a field bus on one vehicle, whereas Wire Train Bus (WTB) is used to connect multiple coaches or vehicles [22].

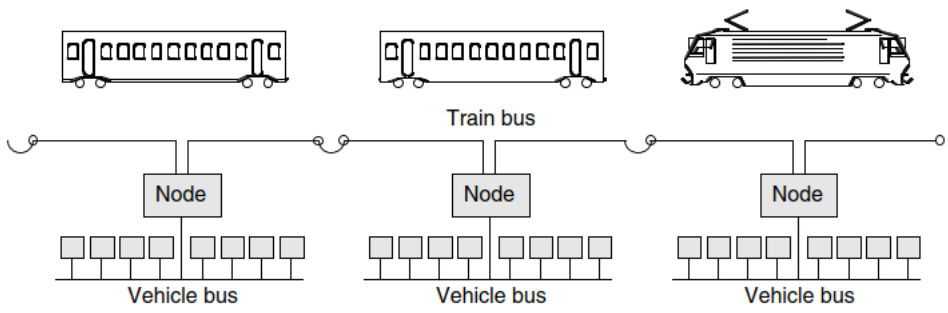


Figure 2.9: The figure shows the basic structure of TCN. Multiple vehicle buses are connected via a train bus. Source: [22]

Figure 2.9 shows a typical topology of TCN with multiple parts that are connected via the train bus. One vehicle might also contain more than one vehicle bus [22]. One instance of such a vehicle bus is MVB. According to IEC 61375-3-1, MVB can consist of multiple

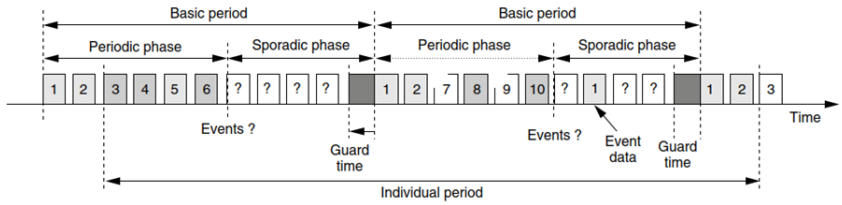


Figure 2.10: The diagram shows the transmission periods of MVB. During the periodic phase, process data is transmitted in fixed time slots. During the sporadic phase, message data can be sent but there are no fixed time slots for the message data. Source: [22]

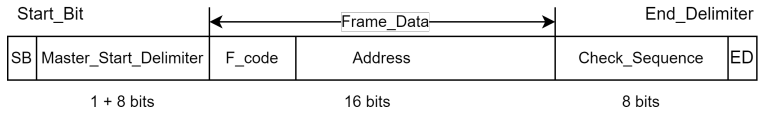


Figure 2.11: The diagram shows the structure of a master frame. The F code signalizes the type of address, e.g., a port address for process data. Source: Replicated diagram from [12]

bus segments of Electrical Short Distance media (ESD), Electrical Medium Distance media (EMD), and Optical Glass Fibre media (OGF) [12]. Multiple segments can be combined in star or line topologies. Additionally, double-line segments can be used to increase availability [12]. Manchester encoding is used for bit encoding. Next to the bit values zero and one, there are also the non-data symbols NH and NL [12].

Like CAN, MVB is in the physical respectively link layer of the OSI reference model. It assures a throughput of 1.5 Mbit/s. In railway vehicles, it is supposed to transport real-time data from the upper layers. This data can have two forms, either process variables or messages [12]. Generally, process variables are safety-critical and must be delivered within certain time periods. On the contrary, messages are not safety-critical and can be sent sporadically [22].

The delivery of messages and process variables is orchestrated by a bus master. However, the bus mastership can also be transferred to other nodes.

As can be seen in figure 2.10, MVB is based on the concept of basic periods, which are split up into periodic phases and sporadic phases [22]. During the periodic phases, the bus master asks bus nodes (slaves) for their process variables. Process variables do not necessarily have the same period. Typical periods can be for example 128ms, 256ms, or 512ms [12]. Process variables can be consumed by multiple data sinks. After the periodic phase, there is the sporadic phase. Here, the bus master asks for events and potentially for the status of certain devices. Message data can be sent or not be sent in return. Again, the message data can be consumed by multiple data sinks.

2.3.2.1 Message Transfer

The two most important frames are the master frame and the slave frame. Combined, they form a telegram.

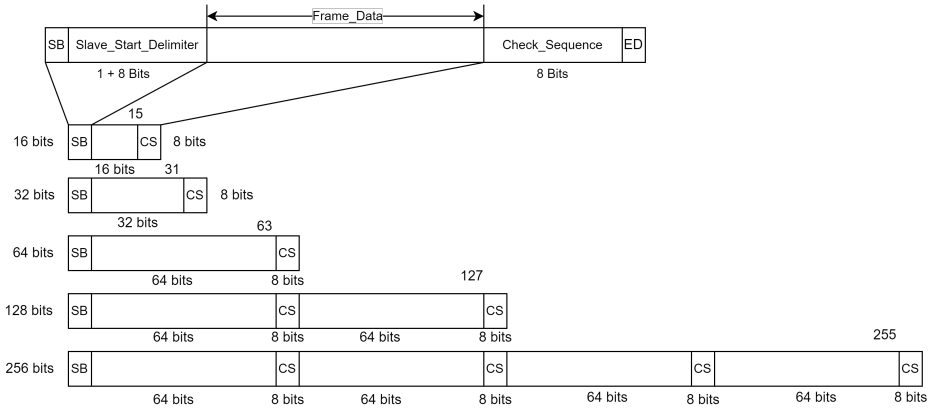


Figure 2.12: The diagram shows the structure of a slave frame. The frame data length is defined by the F code. A CRC check sequence must be included at least once or every 64 bits. Source: Replicated diagram from [12]

The structure of the master frame can be identified in figure 2.11. It starts with a start bit. After that, the master start delimiter follows, which consists of a specific sequence of bit values and non-data values. The actual frame data consists of a so-called F code and an address. The frame ends with a check sequence and the end delimiter. CRC is also used in the case of MVB. According to the norm, an F code indicates the type of the address that is expected after and indicates the size of the expected answer [22]. F codes with values zero to four stand for requested process data with a size of $2^{(Fcode+1)}$ bytes. F codes with values of five to seven and ten to eleven are reserved for future use. F codes with code eight announce mastership transfer to a different device. The rest of the F codes are meant for event announcements or for asking a device about its status.

As can be seen in figure 2.12, the slave frame structure follows the same pattern as the master frame, the only difference being that the frame data can be of variable length depending on the data size. For every 64 bits, there needs to be a check sequence for the proceeding bits. In total, up to 256 bits of data can be sent in one frame.

2.3.2.2 Telegrams

To share process data, the master frame contains an F code with a value between zero and four as stated above. The composition of the master frame and the corresponding slave is called a process data telegram, in this case [12], and the 12-bit address is a logical address addressing a process data port. A collection of multiple process variables that are provided by the same bus device and have the same period is sent in the corresponding process data response as a “dataset”. The idea behind that is to save bus capacity by sending the process data together. The data that is sent in the process data response is only identified by the predecessing master frame. This is also why the time between a master frame and the corresponding slave frame must be below 42.7us [12], which also plays a role in later attack considerations. After the process data response is sent, each data sink puts the response data into its memory buffer and by that overwrites the data it received before.

For message data, there is the so-called message data telegram. Again, it is a sequence of a master frame, the message data request, and a slave frame, the message data response. A message data response can be in single-cast addressing mode or broadcast addressing mode. For a single-cast, the slave frame needs to contain the address of the destination device.

Following the same pattern as the other two telegram types, there is also a supervisory telegram for mastership transfer requests, event requests, or device status requests. A corresponding response follows on each request.

2.3.2.3 Events

Events take place in the event phase, which is a sub-period of the sporadic period. General events, group events, and single events exist. The idea of the event phase is to enable bus devices to share the information that they have message data available. As already mentioned before, message data does not necessarily have a fixed period. During the event phase, the bus master asks for events. This can be done generally, for groups, or single devices. Usually, the bus master asks generally and then decreases the size of asked devices if there are bus collisions.

2.3.2.4 Error Handling

According to IEC 61375-3-1, a correct frame has a suitable start delimiter, correct dimensions, and a correct CRC [12]. For the actual reaction to errors, especially in case of faulty frames and incorrect timing, there is not much information in the standard. Point 5.1.10 (Receiver behavior in case of error) states that if a collision is detected "the decoder shall ignore all frames on that line until the next Master Frame_Delimiter is received" [12]. However, it can be suspected that this behavior applies to most of the error situations because an immediate retransmission would lead to a disturbed timing of other process variables or messages. Particularly in the periodic phase, this would not be feasible. It suits the protocol characteristics to wait for the next period.

2.4 Joint Intrusion Detection System

As the final part of the foundations, this section introduces the use cases of a joint IDS in the context of a security monitoring system for automotive and railway vehicles. Following the taxonomy of Lazarevic et al. [25], the goal is to describe a network-based IDS. In terms of the analysis strategy, this work aims at anomaly detection. The intrusions shall be detected in real-time or at least close to real-time. For CAN intrusion detection, this requirement is very common. The architecture can be either distributed, centralized, or decentralized according to Vasilomanolakis et al. [54]. Which architecture is finally chosen, is left open for chapter 7 when all results can be considered. The response of the IDS is out of scope. After the introduction of the use cases, the thesis proceeds with related work.

Use Case 1 – Training

The system shall provide the opportunity to train individual models for intrusion detection on CAN and MVB. The training process should be well-defined and reproducible for further iterations. Furthermore, the training parameters should be configurable to adjust the models.

Use Case 2 – Deployment

The system shall provide a well-defined process of deployment of the machine learning models. The models can be deployed on a central unit, on middle nodes, or the vehicles themselves.

Use Case 3 – Detection

The machine learning model shall allow a high recall and precision for intrusion detection. Furthermore, it should allow to determine the fragments of the communication data in which the intrusion occurs for further analysis. These fragments need to be reported to an analyzing unit.

Use Case 4 – Update

The system shall provide the capability to update the models respecting the analysis of the reported fragments. Adjustments should be made for example in reaction to many false positives or if new training data is available.

3 Related Work

This chapter gives an overview of relevant related work considering the fields of joint security monitoring for railway and automotive, intrusion detection on in-vehicle networks, and collaborative intrusion detection.

3.1 Project FINESSE

A consortium around Escrypt, Fraunhofer SIT, INCYDE, University of Passau, and Yekta IT was formed in 2021 to work on a project called FINESSE[1] (German "Fahrzeug-Intrusions-Detektion und -Prävention in einheitlicher Struktur für Straße und Schiene", English "Vehicle Intrusion Detection and Prevention in a Uniform Structure for Road and Railway") [45]. It is funded by the German Ministry for Education and Research. The consortium wants to develop a fleet-based security monitoring system that combines railway as well as automotive vehicles. A central component of this project is a Vehicle Security Operation Center (VSOC). The VSOC collects data streams from system buses and internal communication systems of the vehicles as well as system logs, aggregates and analyzes the data to then execute appropriate reactions such as creating alerts. Intrusion detection has two main parts in this project: classic, rule-based detection and detection based on machine learning. The detection of attacks also results in the deployment of security parameters back to the vehicles to make local intrusion detection possible [45]. The combined security monitoring of railway and automotive vehicles aims towards the creation of a "mobile threat intelligence" [50]. Attacks on railway and automotive vehicles could follow similar patterns and a joint monitoring of attacks could increase the knowledge in both sectors.

Figure 3.1 shows an initial architecture idea from the project proposal. An innovative approach is that alerts can also be created locally on the vehicles. This reduces the possible number of logs that need to be sent to the central entity. Components marked with a "C" are part of the VSOC [45]. As can be seen in the figure, logs and alerts are collected in the "enrichment" component. Results are collected and analyzed in the Security Incident and Event Management (SIEM). From there, three important actions are initiated [45]: First, incoming logs get analyzed further based on AI and rules. Eventually, this leads to new alerts. Second, new rules and AI model updates are derived and deployed on the Smart Sensors of the vehicles. Third, the alerts are used to initiate incident response.

The FINESSE project's objectives differ to some extent from this thesis' objectives. The central part of this thesis is to find a machine learning intrusion detection approach that can be used for automotive as well as railway vehicles and that can be deployed in a joint architecture. Throughout this work, intrusion detection usually means network-based intrusion detection. Also, certain aspects of deployment, ideas about adding rules

[1] https://www.forschung-it-sicherheit-kommunikationssysteme.de/projekte/finesse, accessed: October 26, 2023

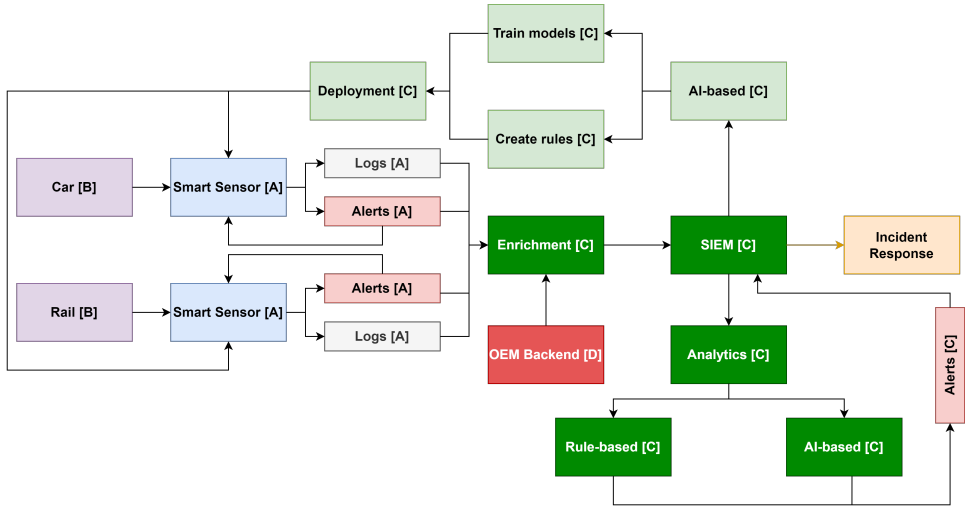


Figure 3.1: The reference architecture of FINESSE is shown. The components with a "C" are part of the VSOC. The VSOC collects alerts and logs ("A") in a SIEM, analyzes them, and eventually derives more alerts based on rules and AI models. New models get trained and new rules are formulated for deployment back to the Smart Sensors ("A") of the vehicles ("B"). Incident response measures can be launched from the SIEM. Source: [45]

for intrusion detection based on what can be hard to detect with the chosen machine learning approach, and updating procedures are proposed. The thesis does not include any content about the incident response or additional SIEM functionalities. Furthermore, the FINESSE project also has host-based intrusion detection in scope [45] but this thesis has not. Most importantly, this thesis is independent of FINESSE content except for the initial considerations about the similarities between automotive and railway vehicles and the general context.

3.2 AI-based Intrusion Detection on CAN

There exist numerous publications and approaches for the research field of AI-based intrusion detection on CAN. Not all of them can be covered by this thesis. However, Rajapaksha et al. deliver a wide overview of works over the last years [37].

In their paper [18], Hoang and Kim present a semi-supervised approach to detect in-vehicle intrusions on CAN. They take the raw bytes as input and convert the ID of each message into a bit string. Then, they stack 29 of these bit strings on top of each other and receive a 29×29 input window for their neural network. The neural network is a convolutional adversarial autoencoder (CAAE) consisting of an autoencoder reconstructing input and a generative adversarial part that forces the output in the latent space to a categorical distribution that represents the two classifications normal and abnormal. On top of the unsupervised learning phase, there is a supervised learning phase with few labeled data. Using the HCRL dataset [47], the authors show that their method achieves an F1 score of 0.9984 and an error rate of 0.1%.

Dongxian et al. [48] developed the so-called Temporal Convolutional Network-Based Intrusion Detection System (TCNIDS) which is an unsupervised method. It serializes the ID of CAN messages into a word embedding. After that, the input is fed to a temporal convolutional network that is trained to predict the next sequence of CAN messages. The intrusion detection is done by predicting the next sequence of messages and when the "predicted message is in the message set with the top g probability, it is detected as normal, otherwise[,] it is detected as abnormal" [48]. Under the usage of the HCRL dataset [47], they achieve a false positive rate of 0.001 with normal data and an accuracy of at least 0.94626 depending on the concrete attack.

Hanselmann et. al [17] present an unsupervised learning process as well. The presented method consists of an independent LSTM model for each ID that occurs in the dataset [17]. The outputs of those LSTM models are combined in an autoencoder that reconstructs the signal-translated input of the CAN payload. With the resulting loss, an anomaly score is calculated and used to differentiate between normal and abnormal traffic. Next to their intrusion detection approach, the authors also introduce and provide a synthetic dataset (SynCAN). It is used to evaluate their method. For synthetic data, they can achieve an accuracy of up to 0.996, and for real data an accuracy of up to 0.999.

Seo et al. published their research about a generative adversarial network (GAN) based Intrusion Detection System for In-Vehicle Network (GIDS) [47]. Similarly to Hoang and Kim [18], they chose to use a generative adversarial network. Initially, they convert CAN data to images using only the ID of the CAN messages. The conversion to images is realized via one-hot-encoding, which creates 16×3 matrices of zeroes and ones. Intrusion detection is realized with a two-stage process. First, there is a model for known attacks, which is trained on normal and abnormal CAN images and outputs a value between zero and one classifying the CAN images. If the output is below a certain threshold, the image will be classified as abnormal. Otherwise, the image gets passed to a second neural network, a GAN. It is trained by creating fake CAN images by a generator. A discriminator is trained to differentiate between those fake images and the real images. By backpropagation, both get more accurate. Thus, the discriminator can detect abnormal CAN images, which do not suit the normal distribution. In order to evaluate their IDS, Seo et al. used their own dataset which is the common HCRL dataset. They achieve a detection rate of up to 99.9% with the first-stage model (detecting the trained attack) and a detection rate of up to 99.6% with a second-stage model that is only trained on benign data.

Ma et al. [29] show the idea of multi-classifying intrusions using a supervised method. Thus, they also predict the type of attack. The model is based on a detailed feature extraction containing the payload sum of the message data, the time variance within the selected window, the time difference to the previous message, the time difference median absolute error, the CAN ID, the payload, and the payload length. Those features from a sliding window of size w get fed into a Gated Recurrent Unit network (GRU) that outputs a multi-class label. In total, their method achieves an F1-score between 0.9950 and 0.9992 depending on the attack. The HCRL dataset [47] was used. Additionally, the real-time ability of their approach is proven and an architecture for detecting intrusions on multiple vehicles is proposed.

A new publication from 2023 called "Signal-Aware Explainable Intrusion Detection System for Controller Area Network-Based In-Vehicle Network" by Jeong et al. [21]

examines the usage of time series learning using LSTMs. The method takes all signals at a certain point in time and stacks them up until a certain window size is reached. Like CANet [17], those matrices are fed into an LSTM autoencoder that tries to reconstruct the time series. An intrusion threshold is calculated based on validation data. If the loss exceeds the threshold, there will be an alarm. As the highest loss can be localized in the reconstructed data, it can be determined which signal was probably attacked. It is an unsupervised method, which only needs normal data. The authors test the method on their own dataset, which was recorded by driving in everyday situations. Results prove that fabrication and masquerade attacks can be detected with an F1-score of usually 0.99. Suspension attacks do not get detected very well. The performance swings between an F1-Score of 0.933607 and 0.308751 depending on the attacked signal.

3.3 AI-based Intrusion Detection on other In-Vehicle Networks

Despite CAN being the most inspected protocol for in-vehicle IDSs, there are also other in-vehicle network technologies that are used more frequently now. Recent trends show that communication based on Ethernet is used more often than before for railway as well as automotive. This trend is manifested with the establishment of automotive Ethernet and train Ethernet Consist Network (ECN).

Yue et al. [58] published a paper about intrusion detection based on AI for train ECN in 2021. An ensemble method leveraging different classifiers is used combined with a voting system. By that, they hope to leverage the strengths of each of the classifiers and mitigate their weaknesses. Additionally, they built a testbed for training ECN and generated a dataset with several attacks for their evaluation. The proposed method performs well and produces a macro-F-score of 0.972 over all attacks and classifiers that were used.

Jeong et al. [20] came up with an intrusion detection method for Automotive Ethernet in 2021. The contributions of their work include the development of attacks on automotive Ethernet and Audio Video Transport Protocol (AVTP), the proposal of their method and test under real test data, and the demonstration of real-time detection ability. As the model, a 2D Convolutional Neural Network (CNN) was used. On the real data, they achieved F1-scores between 0.9790 and 0.9885.

3.4 Collaborative Intrusion Detection

As pointed out before, this work deals also with the fact that intrusion detection is supposed to be conducted in a collaborative way meaning multiple vehicles of a certain category (e.g., a fleet or the same vehicle type) are monitored together. Chapter2 delivers concrete definitions of the different types of collaborative intrusion detection. This section gives an overview of relevant publications.

In their publication, Li et al. [26] present a disagreement-based semi-supervised learning approach for intrusion detection in IoT networks. Starting their publication, the central observation is made that it can be challenging to obtain labeled training data in real life, especially the necessary quantity. Therefore, it is described how a semi-supervised algorithm can be leveraged, where unlabeled data can be automatically labeled without

human interaction. A tri-training algorithm is used to train three classifiers at the same time. For unlabeled data, a majority voting system is used to teach the third classifier and label accordingly. Additionally, a false alarm filter is created based on the voting system. All components are put together in a CIDS environment. Experiments leveraging a decision tree algorithm and a real IoT environment with 65 nodes show that the approach delivered an error rate of 7.3% and a hit rate of 94.23% and more importantly, that their approach is supreme to comparable stand-alone classifiers like a random forest or a support vector machine (SVM).

Nguyen et al. [32] propose another approach for IoT intrusion detection. They worked on a federated self-learning anomaly detection system. Based on attack data generated by malware, a realistic setting was created. Furthermore, they extract certain features from the network packets like direction, local port type, remote port type, packet length, and more that were concatenated to a tuple. The resulting symbols can then be used for classifier training. GRU models are trained to predict the likelihood of a symbol under the inspection of the k preceding symbols. If the likelihood is under a certain threshold, an alarm will be triggered. The whole training process is conducted in a federated, unsupervised way. The devices just learn with their normal traffic. Initially, random weights are distributed and in each training round a consensus is calculated leading to the update of weights on each device. The method appears to be effective with a true positive rate of up to 95.43% and no false positives respecting different evaluation methods.

Rey et al. [39] inspect a federated learning approach as well. They split a network dataset generated by IoT malware into multiple parts based on the source IP address of network packets. Thus, they obtain eight parts and leave one part on the side to test the detection of unknown attack patterns. 115 features are generated from the dataset to try two different machine learning approaches. First, they try unsupervised training with autoencoders. Second, they try supervised training with multi-layer perceptrons. The evaluation contains different scenarios considering the weight aggregation of the federated learning, the relation of benign and attack data, and the comparison with centralized learning and naive learning, meaning every node would learn for itself. Among others, some central learnings are that federated learning is superior to naive learning and competes on a similar level as centralized learning. Also, regarding the comparison of supervised and unsupervised learning, they found out that the true positive rate of supervised and unsupervised learning is on a comparable level whereas the true negative rate is lower for unsupervised learning.

Lastly, Agrawal et al. [1] generally talk about the usage of federated learning for intrusion detection in distributed networks. Accordingly, they identify some challenges coming with intrusion detection in those networks. For instance, private communication data usually needs to be sent to a central entity for intrusion detection. Furthermore, the central entity represents a single point of failure, and a centralized process is time-consuming. On the contrary, federated learning approaches of intrusion detection face some challenges. Communication overhead in the network is necessary in order to send weights, aggregate them, and distribute them every round. Additionally, malicious clients can poison weights and sabotage the training and finally the intrusion detection. Moreover, data that is not equally distributed can cause the underfitting of models. Finally, edge devices usually have low computation power to train their model locally. In order to face

those challenges, they propose some mitigations like using lightweight machine learning approaches, fast communication protocols (e.g., 5G), compression, and more.

4 Security and Attack Considerations

Starting this chapter, the two protocols CAN and MVB are compared to justify common security considerations. After that, an attacker model is introduced based on available literature and past experiments. The attacker model is used to derive attack classes. Less research about the security of railway vehicles has been conducted so far compared to automotive vehicles. However, what becomes clear from the previous chapters is that modern trains are in theory also prone to attacks from the outside world as well as from internal points of connection. Finally, concrete (attack) datasets are presented respecting the introduced attacker model.

4.1 Comparison of CAN and MVB

This part is about a brief comparison of both protocols. This is necessary to motivate common security considerations like possibly similar attack patterns.

Domains

As stated in earlier sections, MVB was originally designed for railway vehicles and CAN for automotive vehicles. In today's projects, CAN also plays a role in internal train networks. MVB on the other side, is mostly limited to the use in trains. Although automotive and railway vehicles have different requirements for their general architecture, their internal communication mechanisms have some similarities. For example, CAN and MVB as field buses, connect multiple ECUs, sensors, and other devices with each other. Furthermore, both serve a certain safety criticality. Data of several nodes are safety-critical (e.g., speed measures, motor rotations per minute (RPM), light signals) in vehicles. Both CAN and MVB address those safety aspects. The two major concerns in this case are the prioritization of messages, and the error detection resulting in a retransmission. CAN ensures the prioritization of messages by the identifier of a frame and the arbitration mechanism. A retransmission can be issued after the detection of faulty frames. To detect bit errors, a CRC is used. MVB declares safety-critical data as process variables and assigns a fixed timeslot to them during the periodic transmission phase. CRC is used as well to ensure correct bit sequences. In case of an error, a retransmission is issued in the next period. As a result, the data of the faulty telegram is not passed to the application [12]. As can be seen, both protocols address safety requirements differently. However, the challenges they have to face are comparable.

Topology

In terms of bus topology, the biggest difference between both field buses is that MVB was standardized as a part of TCN with respect to a multi-modular architecture of railway

vehicles. Usually, multiple traction units are put together for a combined railway vehicle. However, the topologies are quite similar locally. MVB supports a total of 4095 physical devices and 4095 logical ports. Bus or star topologies are supported [12]. For CAN, the number of identifiers is logically limited to 2048 with the 11-bit identifier configuration. In real vehicles, both field buses show a similar number of participants on the bus. Datasets that were used in the context of this thesis have up to around 100 addresses in the case of CAN and up to around 200 addresses in the case of MVB. Still, this is highly dependent on the concrete vehicle.

Bus usage

The bus access is realized differently by CAN and MVB. CAN follows the multi-master principle while MVB has only one master at a time, which can change if requested. CAN is flexible with adding new bus nodes as the identifier of the nodes' data frame can be chosen based on its priority. If there is supposed to be a periodicity in sending the data, this can be configured accordingly. MVB is less flexible with that. The periodicity of telegrams is defined in the poll lists of the bus master [22]. Once installed, the bus cannot be changed easily by just adding one node, at least for process variables. Furthermore, the process data is requested by the bus master and therefore requires a master frame and slave frame to form a telegram. This is different compared to CAN. From datasets that are introduced later in this thesis, it can be observed that the remote frame request is usually not used. Instead, the nodes send their data individually in fixed periods. Fixed periods start from 10ms and can reach up to around 2s. Some nodes just send data based on certain events. Such an event could be for instance that indicator lights are activated. In the provided MVB datasets, the number of sporadic telegrams (message data) is small. The periods of process data ranged from 16ms up to 1024ms. Thus, it can be concluded that timings are also quite similar in both bus implementations.

Error handling

Error handling is mostly achieved by checking for the correct frame format and calculating the CRC over the sent data in both cases. The fixed periods of MVB require a waiting time until the next period to send correct data. CAN also uses acknowledgments to indicate the correct delivery of frames.

Security measures

Neither CAN nor MVB implement security measures that would support integrity, availability, or confidentiality. The CRC used to check for bit errors cannot be seen as integrity protection as an intruder could simply calculate a CRC sequence for spoofed data. Furthermore, even if intended, the implementation of security measures for those field buses would pose various challenges. For instance, the data format of CAN 2.0 A only has space for 8 bytes of data. This is hardly enough to calculate modern message authentication codes (MACs). Even if a newer CAN specification was available that encapsulates security measures, new questions would come up, like about the key management of the keys that are used for cryptographic operations. More details about security considerations follow

in the next section. For now, without a deeper analysis, it is clearly visible that both field buses did not consider attack possibilities in their design. They were designed for closed systems that only face safety issues.

4.2 Attacker model

The attacker model is described based on the three important categories attacker profile, attack vectors, and attack techniques. To prove the relevance of each assumption, concrete examples from the literature are given. Most research regarding this has been conducted on automotive vehicles so far. It is therefore discussed which assumptions are also valid for railway vehicles.

Tippenhauer et al. published a paper about attacker models and profiles for cyber-physical systems [42]. They analyzed relevant literature to come up with six types of attackers:

1. Basic user/ script kiddie
2. Insider
3. Hacktivist
4. Terrorist
5. Cybercriminal
6. Nation-state

The attack types introduced below highly depend on the skill of an attacker. As recent literature has shown [31, 23], the compromise of ECUs placed in internal vehicle networks at least needs reasonable resources and proficient skills of the attacker. Considering this, the attacker profiles that are most relevant here are the insider, the nation-state attacker, and well-skilled cybercriminals. Hacktivists and terrorists could have vehicles as targets as well. However, according to Tippenhauer et al., these two attacker types are not as skilled regarding technical aspects [42]. Instead of nation-state attackers, other well-funded groups can be considered as well. Basic users or script kiddies can be considered if they have access to publicly available exploits.

Regarding attack vectors, there are many opportunities for an adversary to access the targeted systems. Very important for this thesis and the inspected attacks is first the physical access. On automotive vehicles, there is usually a so-called OBD-2 port to access vehicle buses. OBD stands for "onboard diagnostics" and is used for diagnostic tools but can also be used to inject malicious payload when intended. This was already demonstrated on multiple occasions [13, 23, 7]. There are also other physical ports that can be leveraged in theory. Checkoway et al. demonstrated how they can craft a malicious CD that contains an audio file that exploits a buffer overflow in the media player firmware to inject prepared CAN messages on the internal vehicle bus [7]. Trains also have physical access points installed. For example, it is common to enable maintenance by serial ports, or network cable ports. Second, automotive vehicles as well as railway vehicles offer different types of wireless connectivity. Trains primarily have medium-range and long-range interfaces, via e.g., GSM-R, FRMCS for train control, but also Wi-Fi for

passenger convenience [50]. Cars can have, among other fields of usage, cellular long-range connections for live traffic updates or local short-range networks like Bluetooth for connecting with a mobile phone. Third, there is the danger of supply chain attacks. In case there are security flaws in vehicle components, especially ECUs, it can have severe consequences. In total, the most important vectors for this work are physical access and wireless access because these vectors were already used in the past and do not require proficient insider knowledge [23, 7, 31] to gain initial access.

Category	Assumptions
Profile	• Basic user/ script kiddie with public exploit • Insider • Hacktivist/ terrorist • Cybercriminal • Nation-state/ well-funded organization

Attack Vectors	• Physical access
	  – Service ports (e.g., OBD-2)
	  – Sensors
	  – Devices
	• Wireless access
	  – Short-range (e.g., Bluetooth)
	  – Medium-range (e.g., Wi-Fi)
	  – Long-range (e.g., Cellular)
	• Supply chain

Attack Techniques	• Exploitation of software flaws (e.g., buffer overflows) • Message injection • Malware

Table 4.1: The table shows the attacker model used for this thesis.

Several attack techniques are relevant for the attacker model. As the literature shows, an important technique is the exploitation of software flaws. This was shown in multiple ways [7, 31]. For example, both the cited publications used buffer overflows for their attacks. Checkoway et al. could also profit from security flaws in the remote cellular implementation to avoid proper authentication [7]. These flaws can be abused for the deployment of malware, with which the authors could establish a permanent remote connection [7, 31]. Another important aspect is the injection of malicious messages as an attack on the internal or external communication systems of the vehicle. Again, this was already demonstrated by injecting malicious CAN payloads [13, 23, 7, 31] and the remote

attack via short-, medium-, and long-range wireless connections [7, 31]. The OBD-2 port can be used as well for malicious injections.

Most of the assumptions for the attacker model were made based on previous attacks on automotive vehicles. However, the examined attack vectors and techniques could also apply to trains. The attacker profile might be different. It can be assumed that more insider knowledge is needed and access to certain facilities within railway infrastructure is probably crucial as well. Table 4.1 summarizes the attacker model.

Using these basic assumptions for the attacker model, the thesis now continues with concrete attack classes for CAN and MVB.

4.3 Attacks

The attack classes here were originally introduced for CAN. In the first step, those three attack classes are summarized and justified considering the characteristics of CAN. In the next step, special characteristics of MVB are analyzed to transfer the attack classes from CAN to MVB.

4.3.1 Attacks on CAN

Cho et al. [8] defined some generic attacks on CAN in 2016. Since then, these attack classes have become the basis for the important ROAD dataset [55]. Other datasets are not explicitly based on these attack classes but can be described well with these too [47, 11, 17].

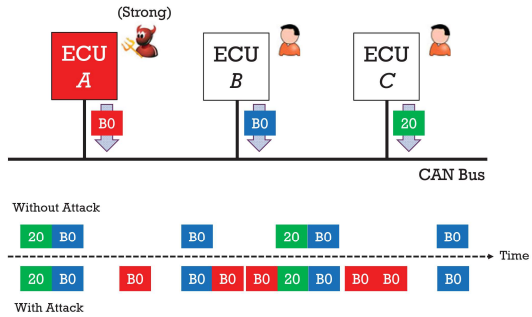


Figure 4.1: The figure shows a fabrication attack. The CAN frames with ID $0xB0$ get spoofed by injecting packets on the bus from a strongly compromised ECU that overwrites the payload of $0xB0$. The message timing changes. Source: [8]

Figure 4.1 shows the first type of attack, the fabrication attack [8]. This attack requires the attacker to fully compromise an ECU. Thus, the attacker can send any CAN frame on the bus that is desired. Cho et al. define such an attacker as a "strong" attacker [8]. The scenario can be seen as realistic as several publications already achieved such a

compromise [31, 23, 7]. The figure shows a typical example of a fabrication attack. There, the attacker's goal is to overwrite the message payload of a CAN frame with ID $0xB0$. So, CAN frames with this ID get injected on the bus. As a result, consuming ECUs update their local data with the malicious CAN payload. Section 4.4 is about the datasets and explains concrete attacks using this scheme. Regarding the timing, it is enough for the attacker to inject one malicious frame every period after the valid frame to overwrite the valid data. In general, and this is important for intrusion detection considerations, this injection method changes the timing of the normal message flow.

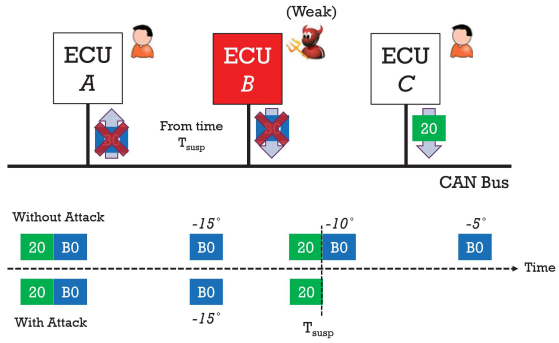


Figure 4.2: The figure shows a suspension attack on ID $0xBO$. The targeted, weakly compromised ECU gets prevented from sending the according frames. The message timing changes. Source: [8]

The second attack is called a suspension attack [8] and can be identified in figure 4.2. In this case, the attacked ECU is considered as "weakly" compromised [8]. Cho et al. justify this with a publication by Foster et al. [13] that showed some scenarios in which attackers could prevent an ECU from sending messages but not injecting messages. This assumption is also supported by observations from Checkoway et. al who noticed that ECUs can be cut off while driving by using a device control service via the OBD-2 port [23]. In general, a standalone suspension attack is not necessarily valuable for an attacker. On the other hand, when being able to cut off multiple ECUs or combining with a fabrication attack (see below), a suspension attack can be useful for an attacker. As a result, the attacked ECU is not able to send its payload, and consuming ECUs cannot update the required data. The suspension attack implies a timing change.

The last attack type is shown in figure 4.3. It combines a fabrication and suspension attack to form a so-called masquerade attack [8]. The attacked ECU gets prevented from sending messages using a suspension attack and a different ECU is used to inject messages via a fabrication attack. By this, the attacker's goal is to entirely replace the valid payload with the malicious payload. The physical effect is the same in the end as for a fabrication attack. However, this type of attack is way stealthier than a fabrication attack. Notably, the message timing does not change.

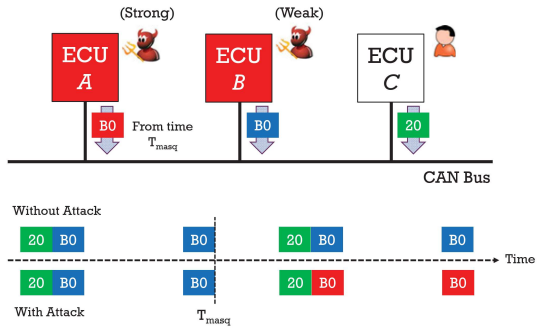


Figure 4.3: The figure shows a masquerade attack on ID $0xB0$. A suspension attack is launched on the weakly compromised ECU B while a fabrication attack is launched on the strongly compromised ECU A. The message timing does not change. Source: [8]

4.3.2 Attacks on MVB

As mentioned before, there is no publication available about attacks on MVB. Therefore, it seems appropriate to look at the CAN attack classes and transfer them to MVB respecting the protocol-specific characteristics. In contrast to CAN, MVB does not follow the multi-master principle. Instead, there is always a bus master and several bus slaves at a time. Following this principle, it makes sense to consider attacks on the bus master and attacks on a slave.

Assuming an attacker compromises or influences the bus master, some attacks are realistic. Sticking with the terminology of fabrication attacks, suspension attacks, and masquerade attacks, an attacker could most importantly mount a suspension attack on the bus master. This would lead to a state where neither process data nor message data is requested from the slave devices. This state can be seen as Denial-of-Service (DoS). In the case of a strongly compromised ECU, an attacker could also possibly stop the master frames just for certain slaves. Moreover, a fabrication attack could be mounted. Such a fabrication attack could have different targets. Assuming an attacker could access and manipulate the message logic of MVB as was already shown for CAN [31, 23, 7], an attacker could inject error frames to make slave responses invalid. Furthermore, the attacker could inject malicious payloads on the bus to overwrite payloads from slaves. In this case, specific timing constraints need to be considered. For CAN, an attacker can overwrite frames by sending the malicious message just after the valid message. For MVB, the standard has an interesting section about that, which states that multiple slave answers can arrive within the defined reply time window after a master frame (42.7 us). In this situation, the first slave frame would be accepted, and the other ones dropped [12]. This means that an attacker could overwrite a slave frame by replying first. However, a window of 42.7 us is small, and the success of that attack could depend on factors that the attacker cannot necessarily control, e.g., the position of the compromised ECU on the bus. If the attacker can cut off a bus slave from communicating, he will be also able

to conduct masquerade attacks targeting the payload of the cut-off slave. Assuming an attacker compromises or influences a bus slave, pure suspension attacks are possible. The compromised ECU does not send any frames anymore. Next, a fabrication attack can be mounted from the slave as described above. And lastly, also a masquerade attack would be possible by making use of two slaves.

To put it into a nutshell, it can be said that an attacker could only achieve a physical effect by either cutting off the bus master respectively slaves or by overwriting process data and message data from the slaves. In the end, all this would influence the slave frames. It becomes obvious as well that despite CAN and MVB following different communication schemes, they are similar on an abstract level as important data gets requested periodically, and less important data can be sent sporadically. The port addresses of MVB can be well compared to the IDs of CAN and the payload of a data frame of CAN can be well compared to a process respectively message data response of MVB. The sent data is subscribed by sinks in both cases. All those similarities justify joint attack considerations as done above.

4.4 Datasets

The presented datasets used for this work were mostly obtained from scientific publications. There are four CAN datasets and one MVB dataset. Of course, more datasets exist, but those seem to be the most popular ones taking the literature into account. The MVB dataset was obtained from a source that is not named here. Due to the protection of the source, some information about the data needs to be omitted throughout the next sections. Notable as well is that the CAN datasets also contained attacks while the MVB dataset consists of benign captures from a stationary train.

Each dataset used for assessment during this work is mentioned in this section. For CAN, there are some categories in which the datasets differ. CAN frames can have two forms. Raw bytes on the bus or the deserialized form according to the model-specific database CAN file (DBC) can be used. The difference in scope of this work is mainly that the deserialized data has the potential of a better machine learning performance according to Jeong et al. [21].

```
BO_ 170 AccPedal: 8 DME
 SG_ KickDownPressed : 53|1@0+ (1,0) [0|3] "" XXX
 SG_ CruisePedalActive : 54|1@0+ (1,0) [0|1] "" XXX
 SG_ CruisePedalInactive : 55|1@0+ (1,0) [0|1] "" XXX
 SG_ ThrottlelPressed : 50|1@0+ (1,0) [0|1] "" XXX
 SG_ AcceleratorPedalPressed : 52|1@0+ (1,0) [0|7] "" XXX
 SG_ AcceleratorPedalPercentage : 16|16@1+ (0.01,0) [0|100] "" XXX
 SG_ Counter_170 : 8|4@1+ (1,0) [0|15] "" XXX
 SG_ EngineSpeed : 32|16@1- (0.25,0) [0|65535] "U/min" XXX
 SG_ Checksum_170 : 0|8@1- (1,0) [0|65535] "" XXX
```

Figure 4.4: The figure shows a DBC file with signal extractions from the E9x series. Source: Screenshot of DBC file from [9]

Figure 4.4 shows a DBC file from OpenDBC [9] concerning the BMW E9x series. It was reverse-engineered by a private user. Usually, those DBC files are proprietary and cannot be obtained publicly. As can be seen, it contains specific information like the offset, signedness, and ranges of data as well as the specific meaning of each ID. This information gets used to interpret the raw bus data. Furthermore, the datasets can be different in terms of capturing conditions. For example, a stationary car has less diverse data than a car that was driven during the capture. This was already shown by Jeong et al. who compared the hamming distances of the payloads from two different captures, one stationary and one moving [21]. CAN datasets can also differ in the number of IDs respectively ECUs in the dataset.

4.4.1 HCRL

The HCRL dataset [47] is probably the most used attack dataset until today. It was used for the evaluation of numerous intrusion detection approaches [18, 48, 47, 29]. Table 4.2 summarizes the most important characteristics.

Vehicle	Hyundai Sonata
Number of signal streams	26
Data format	CSV with timestamp, CAN ID, DLC, Data0, ..., Data7, label
Attacks	DoS, fuzzing, spoofing driving gear, spoofing RPM
Driving situation	Ambient capture: Driving in town Attack captures: Stationary, live injection of the frames
Signal extractions	No
Notes	According to Bridges et al. [55], the attacks were conducted stationary while the car was moved for the ambient data capture. There are no masquerade or suspension attacks.

Table 4.2: The table shows the characteristics of the HCRL dataset.

The attacks can all be seen as fabrication attacks as they get injected on the bus in a non-stealthy way. The DoS attack leverages the priority mechanism of CAN and injects messages with ID $0x000$, which blocks the entire bus. The fuzzing attack floods the bus with messages of random ID and payload. Both the spoofing attacks target a certain ID respectively signal stream that is supposed to be overwritten. As Bridges et al. also note, it can be observed that all attacks, especially the spoofing attacks, are conducted with high frequency [55]. Thus, the timing changes in the communication patterns are immense when the bus is under attack. They also note that there are discontinuities in the data captures where timestamps suddenly jump [55].

4.4.2 TU Eindhoven

Dupont et al. published a CAN dataset in 2019, which used an Opel Astra, a Renault Clio, and a self-built testbed [11]. Table 4.3 summarizes their work. As for HCRL, they do

not provide signal extractions. For the two cars they used during the data capture, they provided several attacks. Following their documentation, it can be seen in their readme files that the attacks on the cars were not conducted in real life but manually added to the data trace [11]. This is a potential problem because the vehicle's behavior did not get physically verified.

Vehicle	Opel Astra	Renault Clio	Testbed
Number of signal streams	89	55	17
Data format	Log file with timestamp, CAN ID as hex, and payload as hex string		
Attacks	Diagnostic, DoS, fuzzing, replay, suspension		Diagnostic, DoS, fuzzing, spoofing, suspension
Driving situation	Driving in urban environment		No driving since testbed
Signal extractions	No		
Notes	The spoofing attack is the only masquerade attack.		

Table 4.3: The table shows the characteristics of the TU Eindhoven dataset.

The diagnostic, DoS, fuzzing, and replay attacks are fabrication attacks. The diagnostic attack injects ten messages in a small time window. The concrete attack comes from Woo et al. [57]. For the DoS attack, they also send CAN frames with ID $0x000$ in a high frequency (500 kbps). Fuzzing is done in two ways: In one case, they apply fuzzing on the CAN ID by injecting messages with unknown IDs. In the other case, they apply fuzzing on the payload by injecting messages with previously unknown payload. The replay attack leverages CAN frames with ID $0x1A1$ and replays them from previous occurrences. They also increase the frequency of the frame by a factor of ten. For the suspension attack, frames with ID $0x1A1$ get muted during a period of 10 seconds.

They constructed their own testbed with VW components [11]. Using this, they implemented a more complex masquerade attack on the speedometer, which puts the speed needle to 220 km/h.

4.4.3 SynCAN

The SynCAN dataset was created by Hanselmann et al. to provide a common basis for the test of signal-aware IDSs and was used by them to evaluate their IDS CANet [17]. This dataset is unique because it only provides signal-based data, and it is completely synthetic. Table 4.4 summarizes the characteristics of it.

The attacks provided in this dataset are more advanced, especially compared to HCRL. In contrast to the previous dataset, the attacks have certain signals as targets and therefore do not replace the whole payload of CAN frames. The continuous attack takes a certain signal and overwrites it slowly over a certain period, so it drifts away slowly from the normal values [17]. On the contrary, the plateau attack replaces a signal in one moment and overwrites it with a fixed value during a period. The rest of the attacks in table 4.4 are self-explaining. The flooding attack can be seen as a fabrication attack, whereas the con-

Vehicle	No vehicle as it is a synthetic dataset.
Number of signal streams	10
Data format	CSV with label, time, CAN ID, signal 1 of ID, signal 2 of ID, signal 3 of ID, signal 4 of ID
Attacks	Continuous, plateau, flooding, playback, suppress
Driving situation	-
Signal extractions	Yes, and only signal extractions.
Notes	-

Table 4.4: The table shows the characteristics of the SynCAN dataset.

tinuous, plateau, and playback attacks can be seen as masquerade attacks. Logically, the suppress attack counts as a suspension attack. Compared to real vehicles like the Renault Clio or the Opel Astra from before, the number of signal streams is low. Considering the complexity of the attacks, this dataset still seems to be valuable for this work.

4.4.4 ROAD

In 2022, Bridges et al. released a paper about current CAN datasets and addressed their pros and cons [55]. They concluded that the current datasets might be not enough in terms of attack complexity and comparability among each other. Therefore, they created a new dataset called "Real ORNL Automotive Dynamometer" (ROAD). The dataset was mainly created by putting an anonymized car from the 2010s on a dynamometer and injecting attacks live via the OBD-2 port [55]. Characteristics about the dataset can be found in table 4.5.

Vehicle	Anonymized car from the 2010s
Number of signal streams	106
Data format	Signal-based: CSV with label, time, CAN ID, signal 1 of ID to signal 22 of ID Byte-based: log file with timestamp, CAN ID as hex, and payload as hex string
Attacks	Accelerator, correlated signal, fuzzing, max engine coolant temp, max speedometer, reverse light off, reverse light on
Driving situation	On the dynamometer for attacks and ambient captures, on the road for ambient captures
Signal extractions	Yes, for certain captures.
Notes	Some of the attacks are only available in byte-based form.

Table 4.5: The table shows the characteristics of the ROAD dataset.

ROAD is (to the best knowledge) the only publicly available dataset that provides the signal-based and the byte-based versions of attacks. This makes the dataset valuable

to compare the difference between the detection of attacks on raw bytes and attacks on the signal translations. The following attacks were implemented: The fuzzing attack corresponds to a fabrication attack with cycling IDs between $0x000$ and $0x255$ and the maximum payload [55]. This attack is only available in byte-based form, probably because a signal translation was not possible in this case. The next attacks are also fabrication attacks where certain IDs or signals were targeted. One of these attacks is the correlated signal attack where the four wheel speed values of the car are attacked and replaced with false values. The max speedometer attack sets the speedometer to maximum while the max engine coolant temperature attack does the same for the engine coolant temperature. Lastly, the reverse light attacks target the reverse lights and switch them off or on. All the fabrication attacks were conducted in the stealthiest possible way, namely by overwriting the valid frame by sending the malicious frame just after. All those four attacks are also available as a masquerade attack. To do so, the authors manually changed the dataset after the capture. They entirely replaced the valid frame with the malicious frame, so no timing change occurs in the data [55].

In addition to the planned attacks, the authors discovered more vulnerabilities that could be exploited to conduct the so-called accelerator attack. The attack leads to a state where the speed of the car is locked to a certain level and neither the accelerator pedal nor the cruise control can be used to alter the speed. After the brakes were released, the car would accelerate to this speed. To preserve the anonymity of the car and the manufacturer, the actual injection was not provided, just the state in which the car was after the attack.

All attacks were provided in the byte-based form. However, only the masquerade attacks were provided as signal translations. The authors justified that with the argument that fabrication attacks do not need signal translations to be detected as they cause timing changes. Masquerade attacks on the other side need a careful inspection of the payload. Although the argument makes sense, it would still have been helpful to have more signal translations to compare more attacks in their byte-based and signal-based form.

After inspection and analysis of the presented datasets, it was decided to not consider the dataset from TU Eindhoven for experiments. The reason behind this is that the attacks of this dataset are mostly covered by other datasets. Besides, the dataset is less realistic than other datasets like ROAD and does not contain any live injections of data onto the CAN buses of the used cars. It is still worth mentioning here.

4.4.5 MVB

In contrast to the CAN datasets, the MVB dataset was not publicly available. Furthermore, the data capture was done on a stationary train and contains no attacks. Table 4.6 summarizes the characteristics.

The original data capture was available as a TXT export from Wireshark containing network data from a tool that translates the MVB data to UDP network packets. In several steps, the data was transferred from this format to a CSV format with a continuous timestamp, a control sequence, the type of data (event or frame), the type of frame (master or slave), and the payload. The goal of this transfer was to create a form that is similar to the CAN datasets. In total, there exist four datasets from this train where, judging by the file names, two datasets were captured on one coach and two datasets on another coach.

Vehicle	Anonymous
Number of telegrams	Around 200, process variables and message data combined
Data format	TXT export from wireshark
Attacks	-
Driving situation	Stationary
Data extractions	No
Notes	-

Table 4.6: The table shows the characteristics of the MVB dataset.

After the preprocessing of the data, it was examined which attacks could be realized on the existing data. A central requirement was that after the injection of attack data, the dataset should still be as realistic as possible. This is why the focus was mainly on slave ECUs on the bus, which is not a big constraint because the physical effects result from attacking the slave payloads. For the slaves, masquerade attacks, fabrication attacks, and suspension attacks were implemented. The fabrication attack leverages the mechanism described before. Thus, the attacker injects a malicious process data response before the valid process data response. Only single data points of a data set were attacked based on the process data response observed before. The masquerade attack replaces the valid slave payload with a spoofed payload. The suspension attack mutes the corresponding ECU during a chosen time period.

The only attack conducted on the master was the suspension of a certain master frame which has about the same consequences as muting the slave directly. Other attacks were not conducted as this would change the data stream entirely (altering master frame timings, changing master frame contents).

In total, the MVB dataset used is not optimal compared to available CAN datasets. The attacks conducted, especially compared to ROAD, are not as realistic. Also, the train was stationary while capturing and the total capture time is just around eleven minutes. However, it still assures a start towards intrusion detection on MVB and allows testing of some intrusion detection approaches in the laboratory.

5 Methodology

In this chapter, the empiric methods of this work are described. Starting, some challenges that are crucial for the goals of this thesis are named. After that, concrete consequences for the chosen methods and experiments are derived from these challenges. Next, two intrusion detection approaches are introduced with their concepts, and it is explained how they are used for this work. Finally, the implementations and the experimental setup are explained.

5.1 Challenges

This section is about some challenges that need to be solved when designing a combined intrusion detection system for railway and automotive vehicles based on machine learning.

Challenge 1 – Labeled data (C1)

If the intrusion detection approach based on machine learning leverages supervised learning, huge amounts of labeled data will be necessary for the training process. Furthermore, the relation between attack data and benign data needs to have a good balance to prevent the overfitting of a model to one or the other class. With supervised and certain types of semi-supervised learning, only the attacks can be detected that were really observed and trained before. For unknown attacks, it would be unclear if the model would classify the data correctly. In case new attacks occur, the model must be adjusted constantly to match the threats.

Challenge 2 – Datasets (C2)

CAN and MVB, although they have certain similarities, are different, especially regarding the bus access and the data format of frames. Road vehicles and railway vehicles are different as well comparing their technical design. Even road vehicles and railway vehicles among each other can have differences in their internal communication systems and technical design. This makes the integration into one monitoring system challenging. Next, existing datasets are different from vehicle to vehicle, and have different complexity. In the case of attack datasets, also the attacks might have different complexity.

Challenge 3 – Edge devices (C3)

In the FINESSE project, it is stated that intrusion detection might also be conducted on the vehicles themselves with so-called Smart Sensors [45]. However, it can be challenging to deploy machine learning models on those vehicles as computation resources might be

limited. Additionally, especially on road vehicles, edge devices might be the only option to deploy an intrusion detection component. Moreover, the literature today mostly talks about real-time detection abilities of intrusion detection on vehicles. In total, there is the problematic challenge of having fast intrusion detection, which should classify well on the one hand, and can be deployed on an edge device on the other hand.

Challenge 4 – Signal extractions of CAN (C4)

On CAN, but also MVB, data is shipped as raw bytes. However, the data sinks need to have instructions to interpret the byte values. For CAN, this is achieved by the usage of DBC files. They encode the concrete meaning of each ID by showing the single signals. They also contain offsets, signedness, and ranges of the data. As is shown in chapter 4, most datasets do not have signal extractions. In practice, these DBC files are usually proprietary and not available to third parties. This fact must be considered for this work as it most likely has an impact on intrusion detection performance if these extractions are not available. This depends on who would operate the IDS.

Challenge 5 – Availability of MVB data (C5)

As already mentioned, CAN datasets, and also attack datasets are publicly available. On the other side, public MVB datasets could not be found so far during the project work for this thesis. This poses a challenge for the evaluation of approaches because the MVB data provided for the experiments is proprietary, results from a stationary capture, and only contains simulated attacks that do not represent the complete attack range possible.

5.2 Consequences and General Method

To overcome those challenges, some general restrictions are made. C1 is a challenge that is very common for practical machine learning applications. In practice, it is barely or not feasible to conduct supervised learning due to the mentioned reasons. This is why this work puts the focus on semi-supervised respectively unsupervised learning approaches.

C2 and C5 combined are addressed in several ways. First, this work starts with finding a suitable machine learning approach for CAN datasets and then transfers this approach to the available MVB data respecting the protocol-specific aspects of MVB. Second, the different types of vehicles and the corresponding data distributions of their communication data are addressed by training individual models for each vehicle respectively protocol. Within groups of vehicles, e.g., fleets of the same type of vehicle, other paradigms can be leveraged as well, like federated learning. Third, during evaluations, the approaches get judged by the most complex attacks. This means it is assumed that a certain approach works better if it detects complex masquerade or fabrication attacks. If only attacks of lower complexity (DoS, fuzzing) are detected, the approach cannot be seen as advantageous anymore.

C3 implies that at least the inference of the machine learning model should be lightweight and work on an edge device. The training process can be executed on a central entity.

Addressing C4, the approaches are evaluated with signal extractions and raw bytes if possible.

Having those challenges addressed, two intrusion detection approaches are introduced in this chapter.

5.3 Detecting In-Vehicle Intrusion via Semi-supervised Learning-Based CAAEs

The first evaluated approach was proposed by Hoang and Kim in 2022 [18]. They leveraged a convolutional adversarial autoencoder for semi-supervised learning and evaluated their idea under the usage of the HCRL dataset. The following sections explain their method and state how the approach is evaluated in this work. The results of the evaluation can be found in chapter 6. Their code repository can be found on GitHub [19].

The approach was chosen because the detection performance is promising (error rate of 0.1%, F1-score of 0.9984 on HCRL dataset) and it needs very few labeled data according to the evaluation results [18]. Additionally, it fulfills real-time requirements as described by C3 [18]. The authors claim that also previously unknown attacks can be detected by the model [18].

Hoang and Kim propose a so-called Convolutional Adversarial Autoencoder (CAAE) to conduct intrusion detection on the CAN data. A CAAE combines the two elements that were already introduced in chapter 2, an autoencoder, and a generative adversarial network. Thus, an autoencoder is used for reconstruction learning of the input. Additionally, the latent space of the autoencoder is forced into the categorical distribution to classify the input as attack or benign by a categorical discriminator.

For preprocessing, they assume that the message flow of CAN can be classified by the order of the frames. The idea behind that is that the normal sequence of messages would change in the case of injected messages. That is why they extract the CAN ID of each frame in the used dataset for the input. They convert the 29-bit ID (extended identifier) into a bit string of 29 zeroes respectively ones. During preprocessing, they convert 29 subsequent CAN messages to a matrix of 29×29 bits, which serves as the input for the CAAE. Notably, there is no sliding window when preprocessing the data, or in other words, the input frames do not overlap. An input frame is labeled as malicious if it contains at least one malicious CAN frame. Figure 5.1 shows their CAAE model. The training follows the following scheme: For training, labeled and unlabeled input frames exist that were extracted from a dataset. The number of labeled frames is a lot smaller than the number of unlabeled frames. This serves the semi-supervised setting of this approach. For each epoch of training the model, there are three phases. In the first phase, the reconstruction phase, the autoencoder (the middle part of figure 5.1) is trained. The latent space of the autoencoder is split up into two vectors of size two respectively size ten. $\hat{Y}$ represents the output of the model with probabilities of whether the input frame is an attack frame or a benign frame. $\hat{Z}$ represents additional latent features that are used for the reconstruction of data. In this phase, the learning process is similar to any other autoencoder. The next phase is the regularization phase, which corresponds to the training of a GAN. For the categorical GAN, the real samples are drawn from a categorical distribution with size two. The synthetic samples are represented by the output $\hat{Y}$ of the

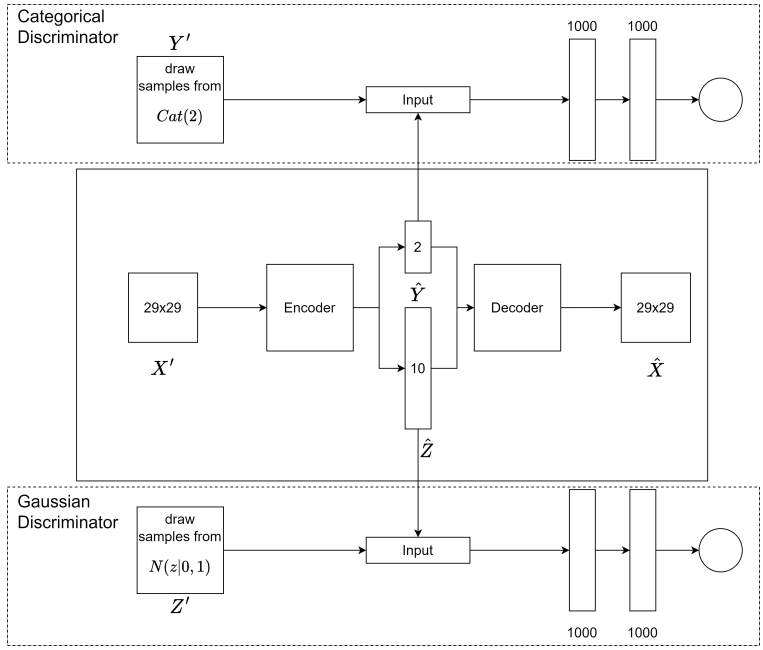


Figure 5.1: The proposed Convolutional Adversarial Autoencoder is shown in the figure. The autoencoder in the middle maps the input to a latent space of size two ($\hat{Y}$) respectively ten ($\hat{Z}$). $\hat{Y}$ is forced into the categorical distribution for output classification. $\hat{Z}$ is forced into the Gaussian distribution to support a better reconstruction. Source: Replicated diagram from [18]

latent space. As described in chapter 2, the discriminator is now trained to differentiate between real and synthetic samples. The same principle is also applied to the Gaussian discriminator. After that, the parameters of the discriminators are fixed, and the encoder is trained towards the output values of the Gaussian distribution respectively the categorical distribution. Finally, during the supervised phase, which is the last phase, the encoder is trained in a supervised way via binary cross-entropy loss to classify the input frames as attack or benign.

Regarding the concrete neural networks, the encoder and decoder were implemented as a convolutional network. These networks are generally well-suited for images. The 29×29 inputs can be seen as images. Convolution works via the alternating application of filtering and max pooling, also called downsampling. The details are not relevant to this work but can be studied here for example [27]. CNNs have some advantages, e.g., they do not have fully connected neurons and by that save parameters. Also, weights can be shared between multiple connections which saves parameters as well[27]. Hoang and Kim use a 3×3 kernel for the filtering and apply four combinations of filtering and max pooling for both the encoder and decoder. They receive a $2 \times 2 \times 64$ output which they flatten and fully connect to the latent space. The decoder works the other way around and reconstructs the original 29×29 size in the end. The discriminators were realized as fully connected networks. The important hyperparameters of training are listed in table 5.1.

Hyper parameter	Value
Batch size	64
Learning rates	10^{-4}
Latent space dimension	$2 + 10$
Optimizer	Adam
Learning rate decay	10^{-1}
Activation function	ReLU
Reconstruction loss function	MSE loss
Discriminator loss function	Wasserstein GAN loss
Supervised loss function	Binary cross-entropy loss

Table 5.1: The table shows the hyperparameters of training the CAAE. Source: [18]

For online intrusion detection, the latent space output is used as the classification output. The categorical distribution indicates a probability for the input frame to be malicious or benign.

5.4 X-CANIDS

The second approach for CAN intrusion detection was published by Jeong et al. in 2023 [21]. In contrast to the semi-supervised model that was introduced before, this is an unsupervised method. No labeled data is needed for training. A further difference is that this approach is based on the signal extractions of the CAN payload. However, the authors also conduct some experiments on the raw bytes as input.

The approach was mainly chosen because it has several advantages over other approaches, particularly over the method by Hoang and Kim that was introduced before. For instance, the method by Jeong et al. was tested using attack classes by Cho et al. [8]. The authors used a dataset with fabrication, masquerade, and suspension attacks. Besides, unsupervised learning helps with embedding this intrusion detection method into a bigger security monitoring architecture. Technically, there is just a certain amount of benign traffic necessary to train a vehicle-specific respectively communication bus-specific model. Another advantage is that, as explained later, the model outputs can be used to find the exact signal that was attacked. This would help security analysts monitor the intrusion detection and evaluate the results. In total, they achieve average F1 scores of 0.9313 for fabrication attacks, 0.9276 for masquerade attacks, and 0.546 for suspension attacks (while the average precision is 0.9765 for suspension attacks) [21].

Figure 5.2 visualizes the training and inference phase of the proposed method. For training and inference, Jeong et al. propose a pipeline with four steps.

First, data from the CAN bus is captured by the so-called message receiver. It caches the payload of the latest signal streams as a matrix $P \in \{\emptyset, 0, 1\}^{|N \times M|}$ where N is the number of signal streams (number of CAN IDs) and M the bit length of the payload. With the CAN specification 2.0, M is 64.

Second, every time interval t the payload sampler which is a part of the feature generator takes an instance of P as P_{copy}. The time interval t is a configurable parameter. The

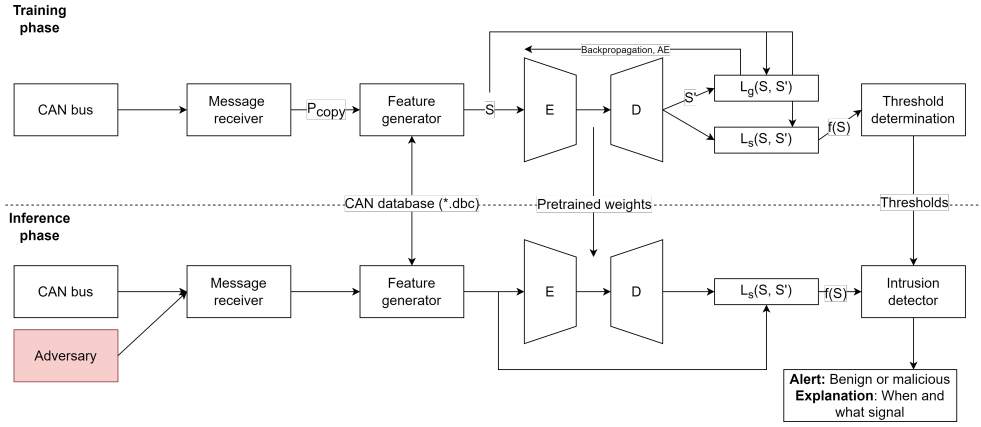


Figure 5.2: The figure shows the architecture of X-CANIDS. During training, the autoencoder is trained to reconstruct benign signal sequences S. S contains w vectors $\hat{s}$. A vector $\hat{s}$ contains all monitored signals at a certain point in time t. An intrusion threshold is determined. During inference, the signal-wise loss is used to check if the threshold gets exceeded. An alarm is raised eventually together with an explanation. Source: Replicated diagram from [21]

first P_{copy} is taken as soon as every signal stream has been observed at least once. The deserializer is the second part of the feature generator and uses the CAN database to extract the signals out of the CAN payload. Thus, every row is converted into a vector of different sizes depending on the number of signals in the signal stream. All signals are concatenated to a vector s. This vector contains all the latest signals of the CAN bus at a certain point in time. The vector s is scaled by the feature scaler, the third component of the feature generator. Jeong et al. apply min-max-scaling with $\hat{s}_i = \frac{s_i - min_i}{max_i - min_i}$. This normalizes each value of vector s into the dimensions $[0, 1]$. The minima and maxima are taken out of the CAN database file. Lastly, the time series feature generator takes the latest w input vectors and calculates a matrix $S \in [0, 1]^{w \times x}$. This matrix S gets returned every t seconds. In consequence, all samples result from a sliding window overlapping by one timestep.

Third, the samples are used as the input to an autoencoder. During the training phase, the autoencoder is trained to reconstruct the input S by reducing the MSE via back-propagation and optimization. Thus, the global loss function is defined as $L_g(S, S') = \frac{1}{wx} \sum_{j=1}^{w} \sum_{i=1}^{x} (S_{ji} - S'_{ji})$. After the learning phase, the special characteristics of an autoencoder are used for an intrusion detection method. The autoencoder should be performant at reconstructing the benign input signals. However, malicious inputs, which do not match the benign signal patterns, are supposed to not reconstruct well. The inference function of the autoencoder is defined as $f(S) = L_s(S, S') = \frac{1}{w} \sum_{j=1}^{w} (S_j - S'_j)^2 = l = \{l_1, l_2, ..., l_x\}$, which corresponds to the signal-wise loss of the autoencoder. It is used to calculate a set of loss vectors on the complete training set. Then, θ_i is defined as $\theta_i = \overline{l_i} + 3\delta_i$ with $\overline{l_i}$ as mean and δ_i as standard deviation of the i-th signal. Error rates of a validation set are calculated as $r_i = l_i / \theta_i$ for $i = 1...x$. With the q-th percentile ($q >= 0.95 <= 1$) of $max(r)$, the threshold Θ is determined. As Jeong et al. state, q can be seen as a hyperparameter for sensitivity [21].

Hyper parameter	Value
Batch size	Not named in the publication.
Learning rate	10^{-4}
Latent space dimension	250 with an input size of 200×150
Compression rate	$\sim 1.17\%$
Optimizer	Adam
Activation function	Tanh
Reconstruction loss function	MSE loss

Table 5.2: The table shows the hyperparameters of training the BiLSTM autoencoder. Source: [21]

For intrusion detection, the inference function is used to determine the signal-wise losses of each input. Then, the error rates r are calculated and an intrusion is detected when $max(r) > \Theta$. The signal that was probably attacked can be determined by checking which of the signals resulted in the maximum error.

The autoencoder was implemented as a BiLSTM network. These networks have LSTMs that feed information forward in time, but also LSTMs that feed information backward in time. This can help with learning time patterns. The input shapes are the ones of the matrix S. Then, two BiLSTM layers follow with size depending on the input shape. The second one of these layers outputs the latent space vector of size h. Then, this vector is repeated w times before the decoder part reconstructs the input size again. Table 5.2 shows the hyperparameters of their network and training. The concrete model layers are defined in section 5.6.

5.5 X-MVBIDS

A goal of this thesis is to propose an intrusion detection method that can be used for CAN and MVB with only slight adaptions. X-CANIDS is a suitable method to convert to MVB because it works on the timing characteristics of CAN. Working with the provided data, it became clear that the timing characteristics of MVB are comparable. The signal streams of CAN that are sent in a fixed period can be mapped to process data ports of MVB. These follow a fixed period as well, and as later analysis shows (section 5.6), the timings are similar. The message data can be compared to signal streams like the one of ID 1649 in the case of ROAD that are not sent in a fixed period. The process data and message data consist of multiple data points that can be compared to all the single signals a signal stream consists of. Finally, the IDs of CAN can be well mapped to the port addresses respectively device addresses of MVB.

Based on these considerations, a training method for intrusion detection on the provided MVB dataset can be derived. The baseline data is the one described in chapter 4. This means there are four benign datasets as CSV files with timestamp, control, type, frame type, and payload. In the first step, the datasets are converted to the "master slave form". There, the F codes of the master frames are analyzed and the frames which request process or message data are used to extract the addresses. After that, the slave frames are joined with this address. All event frames get excluded from the dataset as the focus is on

slave and master frames for physical consequences. With that form, it is now possible to conduct further analysis for whole constant data payloads and constant single data points. The data is checked for whole constant data payloads first, which can be excluded from further data processing. Next, the information about constant data payloads and the bus description is used to split the payload of the slave frames into single data points. The slave frames are now in a comparable form to a CAN frame, with an identifier (the port address that was joined) and the signals (the single data points of each data port). As a result, the original method of X-CANIDS can be applied by checking for constant data points, excluding them, deriving the parameters w, t, x, and applying the proposed feature extraction as before. Two points significantly differ from the original method regarding MVB. On the one hand, the MVB data that was provided for this thesis contains only the raw, byte-based form of the payloads. To overcome this, the data points were converted to decimal numbers during the feature extraction before the scaling was applied. Additionally, the payload contained sequences of data points that consisted of single bits. These bit sequences were combined into one data point to reduce the model dimensions later on. On the other hand, the feature extraction had to respect two types of frames, the master frame, and the slave frame. This challenge was overcome by applying the feature extraction to the slave frames to generate the feature matrix S. This decision is based on the observation that the physical actions are only caused by the payload of the slave frames. The master orchestrates the bus access of the slaves but does not put any message or process data on the bus. Furthermore, manipulations of the master (e.g., suspension, fabrication, and masquerade attacks) would have consequences on the timing of the slave frames.

After the feature extraction and the data split, the autoencoder can be trained. Finally, the intrusion detection threshold is determined under the usage of the validation data. For the inference phase, the schema is analog to X-CANIDS (see figure 5.2) except for the fact that there is no message deserialization. The message receiver collects the MVB frames from the MVB bus and passes the payload to the feature generator every time interval t. To do so, it joins the slave frames with the corresponding master frames and receives the payload for each address. Each payload is cached until there is an update on the bus. The feature generator splits the payload for each address with the help of the bus description and constructs the vector s with length x. It normalizes the vector s by min-max-scaling and receives the scaled vector $\hat{s}$. A feature matrix $S \in [0,1]^{w \times x}$ gets fed to the autoencoder every time interval t starting from the moment when w vectors $\hat{s}$ have been stacked up. The inference function $f(S)$ gets executed and the result is passed to the intrusion detector, which checks if $max(r) > \Theta$ and raises an alarm if so. Section 5.6 introduces the concrete model parameters and discusses the generation of attack datasets for evaluation.

5.6 Implementation and Experimental Setup

In total, there are three intrusion detection approaches to evaluate. The semi-supervised method by Hoang and Kim already comes with a code repository as mentioned before. The unsupervised methods X-CANIDS (signal-translated and byte-based) and X-MVBIDS were implemented from scratch using TensorFlow [51] 2.10. Dataset operations were

conducted with Pandas [35] while statistical calculations and matrix operations were conducted with NumPy [33]. The whole pipeline of preprocessing, data splitting, training, threshold detecting, and evaluating was implemented based on the description by Jeong et al. [21]. The implementations are available on GitHub [14].

The HPI Future Service-Oriented Computing (FSOC) Lab[2] was leveraged for most of the conducted experiments. It provides eight NVIDIA Tesla V100-SXM2 GPUs with 32GB VRAM, an Intel Xeon E5-2698 v4 processor, and about 0.5 TB of RAM. Thus, enough computation power was available. The operating system is Ubuntu 20.04.5 LTS. All machine learning tasks on the FSOC Lab were executed within docker containers.

Additionally, an NVIDIA Jetson AGX Xavier was used. It is an embedded device with a 512-core NVIDIA Volta GPU, a 64-bit-8-core NVIDIA Carmel CPU, 32 GB 256-bit wide LPDDR4X RAM, and 32GB eMMC 5.1 storage. This device is specifically designed for machine learning tasks and was already used by other publications for the matter of measuring the inference times of their models on an embedded device[21, 29]. The used operating system is Ubuntu 20.04.6 LTS. As the machine learning framework, TensorFlow 2.10 was installed natively.

It is now explained which experiments were executed to give answers to the research questions. The semi-supervised approach was tested using two datasets. First, the HCRL dataset was used to conduct some more experiments with other parameters than originally. A low label ratio and train ratio were in focus to check the feasibility in terms of a real IDS. The next section explains label ratio and train ratio as they were declared ambiguously by Hoang and Kim [18]. After that, the ROAD dataset was used to test more complex attacks. This choice was made because ROAD has the most complex attacks of the byte-based datasets. Especially compared to HCRL, it has more stealthy fabrication attacks and masquerade attacks which HCRL does not have at all. Judging by the feature extraction of the method, already an interesting observation can be made. Hoang and Kim assume that the sequence of CAN frames, thus, CAN IDs change when attacks are conducted. This assumption does not hold for masquerade attacks as they do not alter the sequence of frames or the timing. Thus, there would be some changes necessary for the feature extraction to proceed further and detect masquerade attacks as well. Still, the method can be useful under practical conditions because of the characteristics that were introduced above.

X-CANIDS was tested in two bigger scenarios. The first scenario is about the original, signal-translated version of the approach. Jeong et al. [21] used their own dataset for their evaluations. They searched a CAN database file on OpenDBC [9] and a suitable vehicle model to capture CAN data from the OBD-2 port. A Hyundai LF Sonata from 2017 was chosen. In total, seven datasets were obtained from the vehicle. Four were taken for training, one for validation, and one for testing. The last dataset was a stationary capture and was used to compare the model performance on stationary and driving data. All datasets have a length between around 22 minutes and 37 minutes. The focus of their work was mainly the capture of the benign data and not the generation of attacks. In their publication, it is not explicitly named how exactly they executed attacks. However, it is assumed that they created attack datasets by manually changing the payload after, similarly to the dataset of TU Eindhoven [11]. Interestingly, this publication came out

[2] `https://hpi.de/forschung/infrastruktur/future-soc-lab.html`, accessed: October 26, 2023

when the ROAD dataset was already public, which provides fabrication and masquerade attacks. Still, Jeong et al. stated they could not consider the ROAD dataset because it was captured on a dynamometer [21]. Instead, they implemented their own fuzzing, fabrication, and masquerade attacks as well as suspension attacks. That is why their method needs to be reevaluated based on public datasets. The best one for this case is still the ROAD dataset considering the realistic attacks and the number of captures. Furthermore, evaluations on SynCAN were also feasible and therefore used as a baseline. Recall that SynCAN poses a good benchmark for the laboratory because of its synthetic nature and the complexity of the attacks. The combination of evaluation on ROAD and SynCAN gives a good impression of the total performance of the approach. By finding out the performance for each of the three attack classes, fabrication, masquerade, and suspension, strengths and weaknesses can be identified and addressed in a joint architecture.

The second scenario of X-CANIDS considered the evaluation of the byte-based version of the approach. Jeong et al. claimed a worse performance for a model that learns only the byte-based payload of the CAN data. However, they justified their assumption with the comparison based on a stationary capture. It could therefore be a difference to use a non-stationary dataset for that matter. To do so, the X-CANIDS method was applied to the byte-based road dataset. This allows a more realistic evaluation of the byte-based version. Furthermore, challenge C4 addressed this aspect as an important question of this thesis. The road dataset allows a direct comparison of masquerade attack detections in the signal-translated and the byte-based settings. Additionally, fabrication attacks can be tested that are not available in the case of the signal-translated version. For both scenarios, signal-translated and byte-based, suspension attacks were implemented as well which the standard ROAD dataset lacks.

Attack	Target	Period	Attack type
Fabrication attack 1	Slave	1024ms	Set data point to zero
Fabrication attack 2	Slave	128ms	Set data point to maximum
Fabrication attack 3	Slave	1024ms	Set data point to maximum
Masquerade attack 1	Slave	1024ms	Set data point to zero
Masquerade attack 2	Slave	128ms	Set data point to maximum
Masquerade attack 3	Slave	1024ms	Set data point to maximum
Suspension attack 1	Slave/ Master	64ms	Suppress slave/ master frame
Suspension attack 2	Slave/ Master	128ms	Suppress slave/ master frame
Suspension attack 3	Slave/ Master	1024ms	Suppress slave/ master frame

Table 5.3: The table summarizes the implemented attacks on MVB datasets.

Finally, X-MVBIDS was evaluated using the proprietary dataset. The four datasets from two different coaches result from a stationary, attack-free capture. This makes the evaluation harder than for the other datasets, and less transparent. As already mentioned in chapter 4, fabrication attacks, masquerade attacks, and suspension attacks were implemented. All three attack types were applied to bus slaves while only a special type of suspension attack, where certain master frames are dropped to mute a bus slave, was conducted on the master frame. The reason behind this is that only attacks can get tested, which would not result in a completely different data flow. In this case, the datasets would

become unrealistic. A further constraint is the amount of data. The total capture time of all datasets combined is just around 11 minutes.

The attacks can be looked up in table 5.3. The spoofed payloads from the fabrication and masquerade attacks were based on similar attacks from the road dataset like max speedometer attack or max engine coolant temperature attack. Furthermore, it was made sure that frames with different periods were attacked.

The general setup and experiments are now explained. To prepare the experiments, it is now necessary to derive the training parameters for each model that was used. Besides, the label and train ratios of the semi-supervised CAAE approach are clarified.

5.6.1 Clarifying Training Parameters of the CAAE

Hoang and Kim trained their model respecting the hyperparameters named in section 5.3. However, they did not name the number of epochs that were used for training [18]. Judging by their code [19], they had 100 epochs as standard value for training.

For training and evaluation, they used Tensorflow-1.15. With several Python scripts, a pipeline was established consisting of the following steps: Preprocessing, data splitting, training, and performance testing. The corresponding scripts were also leveraged for this thesis and adjusted for analysis if necessary. Two more hyperparameters were introduced by them. One is the "train ratio" and the other one is the "label ratio". It is clear what is meant by the label ratio in this case, namely the relation of unlabeled and labeled samples during training. The meaning of "train ratio" is not obvious in the first place. As Hoang and Kim say, it means "the number of training samples over the total samples in the dataset" [18]. However, in the paragraph below they state "We used 10%, 30%, 50%, and 70% of total attack data for training" [18] referring to this train ratio. This makes the train ratio ambiguous. Some code analysis of their scripts was necessary to finally figure out what was done. First, in the preprocessing part, the attack CSV files DoS, Fuzzy, Gear, and RPM were preprocessed by taking 29 subsequent CAN frames and converting their IDs to a bit string. The resulting input images were saved as TFRecords, a data format that serializes samples to the disk to conduct a training process in multiple stages [52]. The output scheme splits up the input frames into benign and attack (the attack datasets produce also benign frames). Notably, the attack-free data capture of HCRL was not used. Consequently, eight files exist:

1. DoS / Normal DoS
2. Fuzzy / Normal Fuzzy
3. Gear / Normal Gear
4. RPM / Normal RPM

So far, the data was only split up into benign and attack frames. After that, the actual data split was conducted. To do so, the two parts, benign and attack data, were split separately. The normal data was split into training (0.7), validation (0.15), and testing (0.15). Judging by the code, these relations were fixed for the normal data. The train ratio only comes into play for the attacks. The parameter can be adjusted in this case. This means that the authors meant the amount of attack data with the train ratio. For example, a train ratio of 1 would mean they include 100% of the attack data, and a train ratio of 0.1

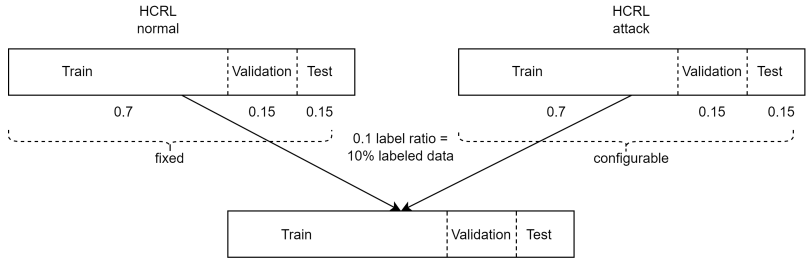


Figure 5.3: The diagram shows the data split strategy used in [18]. The HCRL normal proportions are fixed to the shown split. The HCRL attack proportions are configurable. The standard label ratio is fixed to 0.1. It can also be adjusted to any other value ≤ 1.

would mean they include 10% of the attack data. After the data was split into training, validation, and testing for normal respectively attack data, the training data was split up into the unlabeled and labeled parts. The standard label ratio is 0.1 for both normal and attack data. They are both configured by the same parameter.

The data split is visualized in figure 5.3. Note that different configurations of the train ratio result in a different relation of attack and benign data in the training data.

Having the label and train ratio clarified, the method can be evaluated and analyzed.

5.6.2 Deriving Training Parameters for X-CANIDS and X-MVBIDS

Before the evaluation, the concrete training parameters x, w, and t had to be derived independently for each dataset and scenario. The compression rate of the autoencoder is an important hyperparameter as well. This paragraph summarizes the original paper parameters by Jeong et al. to give a general idea and the opportunity to compare the parameters [21]. During their preparations of the datasets, Jeong et al. made some important observations about the data. Initially, they had 688 signals from 62 signal streams. However, it was shown that during the driving some signals would remain constant throughout the whole capture. As a reaction, they excluded these from the feature extraction with the justification that a simple rule could be added that checks if these signals stay constant [21]. Furthermore, they excluded signals which are checksums, counters, or parity bits. In the end, they reduced the signals to 107 by that strategy. The second parameter that is necessary to determine the input size of the network is the window size w. To figure out w, they analyzed the timings of the signal streams and figured out that except for a few streams, all streams follow a fixed period between 10ms and 2s. Thus, w was set to 200 and the time interval t to 10ms. This allows to capture all changes of the most high-frequent signal streams but also includes at least one potential change of the most low-frequent signal streams. In total, the neural network gets an input of 200×107, a middle layer output of 200×214, and a latent space output of 250, which corresponds to a compression rate of around 1.17%. After the repetition layer, two layers follow which output matrices of size 200×214. For the output, a time-distributed dense layer is applied which reduces the dimension to the original size 200×107. For the actual training, the authors leveraged the already declared training datasets. No additional data split was applied to these. They trained the BiLSTM autoencoder for 2000 epochs with an

early stopping patience of 50 epochs. A loss of around $4.686 * 10^{-4}$ was achieved as the model stopped after 948 epochs. For the intrusion detection threshold, the value $\Theta = 40$ with $q = 0.9869$ was used. Except for SynCAN, the strategy of training 2000 epochs with an early stopping patience of 50 was used for all experiments with X-CANIDS and X-MVBIDS.

After the explanation of the original parameters, it is explained how the parameters, the data split, and the according ratios for each of the experiments with X-CANIDS respectively X-MVBIDS were derived.

Training Parameters for X-CANIDS on SynCAN (signal-translated)

Starting with SynCAN, the parameters x, w, and t had to be determined. The dataset consists of four training datasets that represent the whole training data after they are put together.

```
Label,Time,ID,Signal1_of_ID,Signal2_of_ID,Signal3_of_ID,Signal4_of_ID
0,2088.41338746,id5,0.0,0.9586862512831164
0,2089.55410634,id8,0.24683876529406004
0,2090.88561837,id3,0.2,1.0
0,2091.65840611,id7,0.06388818948282671,0.0
0,2100.36445933,id9,0.4495114006514658
0,2100.47555051,id1,0.0,0.0
0,2103.41338746,id5,0.0,0.9586862512831164
0,2103.51200647,id2,0.0,0.0,0.24188486056440817
0,2104.27893648,id6,0.9473684210526315,0.0
0,2104.55410634,id8,0.24718017437468803
0,2105.88561837,id3,0.4,1.0
0,2106.65840611,id7,0.06397655489013361,0.25
0,2115.47555051,id1,0.0,0.25
0,2118.41338746,id5,0.0,0.9586862512831164
0,2119.05278128,id10,0.45454790160652747,0.1111111111111111,0.9477939303961085,0.1703945623987118
0,2119.55410634,id8,0.24752158345531608
0,2120.88561837,id3,0.6,1.0
0,2121.65840611,id7,0.06406492029744001,0.5
```

Figure 5.4: The figure shows the beginning of the file train1.csv. There are 10 signal streams with 20 signals. Source: Screenshot of train1.csv from [16]

Figure 5.4 shows a part of the file train1.csv [16]. In total, there are 10 signal streams with 20 signals. Thus, parameter x can be set to 20. After this, the timing of the dataset was analyzed to determine the other parameters.

The timings shown in table 5.4 indicate a stable periodicity of the signal streams. Especially these with a mean Δt of 45ms have a low standard deviation of their timing. The lowest mean Δt could be assumed as 15ms, and the highest one as 45ms. This implies a value $t = 15ms$ and $w = 3$. However, the window size was increased to 20 because of the reason that a window size of 3 might be too small to represent attacks over time. By that, the input of the neural network has a dimension of 20×20. Additionally, a latent space size of 40 was chosen, which corresponds to a compression rate of 10%. This is higher by some degree than the original paper suggested. However, the total number of parameters is low for this model with around 37,000 parameters. This allows a bigger latent space. With the original compression rate, a latent space of around seven would be applied, which appears to be small in this case. In general, the size of the latent space is a

ID	Mean Δt in ms	Std. Δt in ms
ID1	15.001743	0.042624
ID2	30.000025	0.005932
ID3	15.001668	0.041410
ID4	45.000000	0.000000
ID5	15.001756	0.042635
ID6	30.000024	0.004748
ID7	15.001712	0.042225
ID8	15.001680	0.042152
ID9	30.000039	0.008895
ID10	45.000000	0.000000

Table 5.4: The table shows the mean time difference and standard deviation between two subsequent frames of a certain ID of the SynCAN dataset. For all IDs, a more or less fixed period can be identified.

hyperparameter that highly depends on the concrete machine learning task and needs to be adjusted based on experiments.

Layer	Parameters	Output shape
Input	0	20×20
BiLSTM	6,560	20×40
BiLSTM	9,760	40
Repeat input 20 times	0	20×40
BiLSTM	9,760	20×40
BiLSTM	9,760	20×40
Time-distributed dense	820	20×20

Table 5.5: The table shows the model parameters for X-CANIDS on SynCAN.

Table 5.5 summarizes the layers including the output shapes of the BiLSTM autoencoder for the SynCAN dataset. The training was executed using all training samples. The data was split using 80% for training, 15% for validation, and 5% for testing. Only 5% were kept for testing as Hanselmann et al. [17] also provided a whole attack free dataset for evaluation. Regarding the preprocessing, it is notable that the data was already normalized between zero and one. Thus, the scaling of data could be omitted. In the first step, all scaled vectors of $\hat{s}$ were extracted from the ambient datasets respecting the parameter t. The extracted vectors were saved as TFRecords. During the data split, the vectors $\hat{s}$ were read in from every TFRecord file and split into training, validation, and testing data. After that, the sliding window was applied to generate the matrices S. By that, information bleeding was prevented. Additionally, training, validation, and testing samples include data from every ambient dataset. This strategy was used for every experiment with the ROAD dataset. 4,140,051 samples were obtained in total for SynCAN.

Training Parameters for X-CANIDS on ROAD (signal-translated)

The ROAD dataset provides signal-translated as well as raw, byte-based captures. It is comparable to the dataset used by Jeong et al. in the way that the number of signals is relatively similar in both cases. ROAD has 664 signals out of 105 signal streams while

the dataset from the Hyundai Sonata has 688 signals out of 62 signal streams. It must be noted that the byte-based version of ROAD has 106 signal streams. Apparently, one signal stream was omitted during the translation by Bridges et al. [55]. Before starting with the determination of all necessary parameters, it is worth noting that all statistical analysis, scaling, and derivation of parameters were only conducted on the benign datasets to serve a fully unsupervised scenario.

A fundamental requirement is the determination of constant signals in the dataset as these do not need to be trained. To do so, the ambient datasets of ROAD were analyzed in that regard. Interestingly, the number of constant signals varies a lot between different data captures. The capture "ambient dyno drive exercise all bits" has only 125 constant signals while the capture "ambient dyno drive winter" has 465 constant signals. This makes sense considering that the first was captured while trying to use all buttons, pedals, and switches in the car, and driving for more than half an hour [55]. The second capture is only around 47 seconds long and is about driving in colder conditions [55]. It can be assumed that the ambient drives from the paper [21] were also just conducted to capture the normal driving in town and not to experiment with all functionalities of the car. That is probably why the number of constant signals is so high for their datasets. Because the signal extractions of ROAD were anonymized, signals that contain checksums, parity bits, etc. cannot be identified without a major effort. It was decided to work based on the least complex dataset "ambient dyno drive winter" regarding the constant signals. In practice, this would mean that a major part of the traffic would be unmonitored. However, it is assumed that the monitored signal streams are the ones that are most critical for driving and thus, the ones that are most likely targeted for an attack. It remains a tradeoff how many signals are monitored. The method proposed here already monitors a lot more signals compared to the original literature.

Next, the parameters x, t, and w needed to be determined. Again, the timing was analyzed to derive these parameters. All ambient captures were analyzed for timing. Not all statistics can be listed here, but the timings were analyzed using the longest dataset "ambient highway street driving long" that was captured on the street to have the most realistic timing statistics.

As can be seen in table 5.6, most of the signal streams follow a fixed periodicity, like SynCAN. The standard deviation is generally higher than for SynCAN. This is probably where real datasets differ from synthetic ones. Generally, every standard deviation is below one except the one of ID 1649. This signal stream has a standard deviation of almost 13. Also, the mean timing difference between two frames of this ID is over 10s. This is why it was decided to exclude this signal stream as the time-series model would probably reconstruct the payload of this signal poorly. The minimum mean Δt is around 10ms, and the maximum mean Δt is around 1500ms. Consequently, the parameter t was set to 10ms, and the parameter w to 150. The question remains how many signals are included finally to determine the parameter x. ID 1649 was excluded already. Furthermore, it was explained before that the constant signals based on the dynamometer winter drive are used. Lastly, it was ensured that no signals got excluded that are part of the test attacks. This violates the assumption that data preparation is only conducted on benign data but is necessary to make a proper evaluation possible. Notably, just parts of the corresponding signal streams had to be fixed. In total, $x = 202$ signal streams remain.

ID	Mean Δt in s	Std. Δt in s	ID	Mean Δt in s	Std. Δt in s
6	1.023104	0.371811	727	0.102150	0.099128
14	0.010206	0.022573	737	0.020402	0.034469
37	0.910228	0.411611	738	0.102070	0.096522
51	0.010206	0.024137	778	0.508787	0.360588
58	0.510431	0.237134	813	0.051016	0.067344
60	0.101970	0.095966	837	0.102087	0.096531
61	0.102233	0.100049	852	0.010202	0.024105
65	1.022354	0.409890	870	0.020405	0.034978
117	0.510154	0.245239	881	1.020691	0.358034
167	0.010211	0.024378	930	1.527569	0.436219
186	0.040794	0.059725	953	1.018881	0.342621
192	0.020428	0.042351	961	0.025502	0.038522
204	1.021245	0.360373	996	0.102320	0.102822
208	0.010204	0.024090	1031	0.102116	0.082560
215	0.102119	0.098205	1049	0.514828	0.279417
241	0.509929	0.235652	1072	0.102101	0.097790
244	0.102172	0.098403	1076	0.020407	0.035096
248	1.020056	0.358429	1124	0.101991	0.099112
253	1.019292	0.347017	1175	0.102155	0.098412
263	0.020398	0.034423	1176	0.020404	0.042300
293	0.010201	0.023600	1225	0.101976	0.096557
300	1.020684	0.350724	1227	0.203931	0.137261
304	0.102134	0.098250	1255	0.101998	0.097923
339	0.020393	0.041882	1262	1.024063	0.430925
354	0.020418	0.034797	1277	0.051035	0.066394
403	0.020401	0.034951	1307	0.203960	0.139877
412	0.020410	0.042005	1314	0.020404	0.034971
420	1.019016	0.356791	1331	0.545104	0.266434
426	1.020604	0.359956	1372	0.153153	0.122611
452	0.255016	0.155788	1398	1.020330	0.379428
458	0.102138	0.099606	1399	0.051121	0.069431
470	0.101686	0.096638	1408	0.020406	0.035027
485	1.021239	0.358835	1413	0.101812	0.097553
519	0.101642	0.096471	1455	0.923077	0.422396
526	0.020403	0.034965	1459	0.101870	0.097151
541	1.020262	0.366774	1505	0.010203	0.024111
560	0.020405	0.042314	1512	1.020729	0.385115
569	0.102163	0.102378	1533	0.916295	0.421587
622	0.102555	0.097677	1560	1.021352	0.396564
627	1.020137	0.355682	1590	0.102125	0.099057
628	0.020424	0.042316	1621	1.019308	0.352915
631	1.526951	0.422334	1628	0.102347	0.103249
640	0.202908	0.137156	1634	0.020401	0.041949
651	0.020410	0.034936	1644	0.101993	0.095661
661	0.030596	0.051410	1649	10.056932	12.978408
663	0.462382	0.254565	1661	0.861361	0.456294
675	0.102116	0.097295	1668	0.102097	0.098726
676	0.255085	0.155844	1693	1.021560	0.396722
683	1.019854	0.336712	1694	0.020402	0.034149
692	0.093491	0.096235	1751	1.020961	0.354235
695	1.018171	0.372762	1760	0.010204	0.024580
705	0.107097	0.098992	1788	0.102111	0.102245
722	0.102069	0.097833	-	-	-

Table 5.6: The table shows the mean time difference and standard deviation between two subsequent frames of a certain ID from "ambient highway street driving long" (ROAD). The periods are less stable compared to SynCAN.

After the determination of the parameters, possible data splits were inspected. Table 5.7 shows all available benign captures.

Capture	Duration (min)	Comment
Ambient dyno drive basic long	21	Basic driving activities
Ambient dyno drive basic short	7	Basic driving activities
Ambient dyno drive benign anomaly	8	Benign anomalies (e.g., open door while driving)
Ambient dyno drive extended long	11	More complex driving activities (e.g., cruise control)
Ambient dyno drive extended short	6	More complex driving activities (e.g., cruise control)
Ambient dyno drive radio infotainment	7	Radio/ infotainment while idling/ driving
Ambient dyno drive exercise all bits	36	Trying to exercise the full range of signals
Ambient dyno idle radio infotainment	11	Radio/ infotainment while idling
Ambient dyno drive winter	1	Driving in colder conditions
Ambient dyno reverse	1	Basic reverse activities
Ambient highway street driving diagnostics	8	Driving in parking lots, streets, highways
Ambient highway street driving long	63	Driving in parking lots, streets, highways

Table 5.7: The table shows all ambient captures of the signal-translated ROAD dataset. Durations were rounded to full minutes. The dataset consists of very different captures. Source: [55]

In terms of the total amount of datasets and the duration, the dynamometer datasets dominate. Judging by the descriptions, the dataset "ambient dyno idle radio infotainment" is stationary. Furthermore, the datasets "ambient dyno reverse", "ambient dyno drive benign anomaly", and "ambient dyno exercise all bits" might differ from the normal driving activities. During this work, many experiments have been executed to find an optimal data split. The problem here is that unlike in the original publication [21], ROAD does not provide multiple, equally distributed datasets. Additionally, the attack datasets were captured on a dynamometer. On the other hand, this makes the evaluation more realistic because driving situations in real life do not always follow the same driving pattern.

The data split applies the approach that was also used on the SynCAN data. It contains every available capture and applies a split of $\frac{4}{6}$ training, $\frac{1}{6}$ validation, and $\frac{1}{6}$ testing. In total, 1,040,030 samples were available. The preprocessing was executed in the same way as for the SynCAN data. The min-max scaling of the data posed a challenge for the ROAD data because, unlike SynCAN, the data was not normalized yet. Besides, the CAN database file was anonymized and does not contain the minimum and maximum for each signal. This is why the ranges of the signals were determined a priori on all benign datasets. In consequence, benign data from the attack datasets might be reconstructed worse because their signal values might be outside the range of the benign datasets.

However, a bad reconstruction might also occur if the benign test data were scaled based on all available data. For the model, the parameters w and x were used as introduced before. Based on multiple experiments, a compression rate of about 1.56% was chosen. This corresponds to a latent space size of 472. Table 5.8 summarizes the model.

Layer	Parameters	Output shape
Input	0	150×202
BiLSTM	654,480	150×404
BiLSTM	1,210,208	472
Repeat input 150 times	0	150×472
BiLSTM	1,090,800	150×404
BiLSTM	980,912	150×404
Time-distributed dense	81,810	150×202

Table 5.8: The table shows the model parameters for X-CANIDS on ROAD.

Training Parameters for X-CANIDS on ROAD (byte-based)

The X-CANIDS method was applied to the byte-based road dataset with the same parameters as before. This means that the same captures were used for training, the same data split was applied, and the same compression rate of the autoencoder was used. The byte fields of the CAN payload were interpreted as signals. Thus, each CAN ID is associated with eight signals because the ROAD datasets only have payloads with eight bytes. The normalization of the data was again conducted under the usage of min-max scaling based on the benign datasets. The parameters w and t were kept as 150 respectively 10ms while x was set to 244. x resulted from the exclusion of constant bytes using the dynamometer winter capture as a baseline. With a compression rate of about 1.56%, a latent space size of 570 was determined. Table 5.9 shows the parameters per layer.

Layer	Parameters	Output shape
Input	0	150×244
BiLSTM	954,528	150×488
BiLSTM	1,764,720	570
Repeat input 150 times	0	150×570
BiLSTM	1,590,880	150×488
BiLSTM	1,430,816	150×488
Time-distributed dense	119,316	150×244

Table 5.9: The table shows the model parameters for X-CANIDS on ROAD byte-based. Each layer has more parameters now because of the bigger input dimensions.

Training Parameters for X-MVBIDS

The evaluation was done based on the proprietary dataset that was provided from an anonymous source. Respecting the established method from X-CANIDS, the parameters for model training were derived. Due to anonymity reasons and the pure number of process data telegrams respectively message data telegrams, the full timing table of the

data cannot be shown here. The four datasets were comparable regarding the number of addresses even though they came from different coaches. The smallest period is 16ms seconds while the largest period is 1024ms. This was also confirmed by the provided bus description. Thus, the parameters $t = 16ms$ and $w = 64$ were set. For x, it was necessary to determine the constant data points and the whole constant payloads of the datasets. Again, as done before with X-CANIDS, the dataset with the most constant payloads and constant data points was chosen as a baseline. Each data field of a telegram was interpreted as a "signal" and the single bits were combined into one data field. Thus, the parameter $x = 274$ could be set.

For the data split, the standard strategy from before was executed. One dataset was put to the side in the beginning to provide benign test data and to serve as a basis for the attacks. The other three datasets were split into training, validation, and test data each in the relations 0.8, 0.15, and 0.05. These relations were chosen to have as many training samples as possible considering the data sparseness. The test data was kept to have a second source of test data next to the dataset that was put aside in the beginning. In total, 37,060 samples were available for the data split.

The resulting model parameters are shown in table 5.10. The latent space size was set to 400 respectively a compression rate of about 2.28%.

Layer	Parameters	Output shape
Input	0	64×274
BiLSTM	$1,203,408$	64×548
BiLSTM	$1,198,400$	400
Repeat input 64 times	0	64×400
BiLSTM	$1,479,600$	64×548
BiLSTM	$1,804,016$	20×548
Time-distributed dense	150,426	64×274

Table 5.10: The table shows the model parameters for intrusion detection on MVB data.

6 Evaluation

The evaluation of approaches respected the available datasets and the characteristics of the intrusion detection approach. The first section of this chapter describes the achieved results. The second section of this chapter analyzes them.

6.1 Evaluation Results

The metrics for evaluation were chosen according to the original literature to allow a direct comparison. This also applies to the rounding accuracy. The result tables of X-CANIDS and X-MVBIDS are too detailed to be presented in the text. This is why they were put into appendix A. The most important results are summarized directly in the text. Interested readers can jump to the tables.

6.1.1 Detecting In-Vehicle Intrusion via Semi-supervised Learning-Based CAAEs

Hoang and Kim originally evaluated their method using the HCRL dataset with different label ratios and attack frame ratios. They used error rate, recall, precision, and F1 score as metrics and achieved good results with an F1 score of up to 0.9984. They also evaluated the detection of unknown attacks which they simulated by omitting one attack type during training and achieved an F1 score of up to 0.9976.

The code by Hoang and Kim was used to conduct experiments. The train ratio was mostly in focus as it seems to be more important for the performance than the label ratio. The label ratio barely influences the results according to Hoang and Kim [18]. The test of more complex attacks was another goal. The first table 6.1 is about the reevaluation of the HCRL dataset with partially the same parameters that were used in the paper. The model was trained using the same hyperparameters.

Train ratio	Error rate	Recall	Precision	F1
0.01	13.9%	0.8423	1	0.9144
0.1	5.1%	0.9430	0.9996	0.9705
0.2	3.2%	0.9635	0.9999	0.9814
0.4	2.9%	0.9667	0.9999	0.9830
0.7	1%	0.9747	0.9994	0.9869

Table 6.1: The table shows the influence of the train ratio on the intrusion detection performance regarding HCRL attacks with other train ratios than originally. The label ratio was fixed to 0.1. The results from [18] could be confirmed.

Train/ label ratio	Error rate	Recall	Precision	F1
0.05	10.39%	0.8824	0.9999	0.9375

Table 6.2: The table shows the influence of a low label and train ratio on detecting HCRL attacks. Even with a label and train ratio of 0.05, a precision of almost one can be achieved. Furthermore, the recall is still high with 0.8824.

The new results confirm the results from the literature. Although the metrics are not the same, the trend regarding the error rate, recall, and precision is similar. Again, the label ratio was fixed to 0.1. Using a train ratio of 0.01, the error rate appears not feasible anymore for any real-world application. However, the precision remains constantly high for any train ratio indicating a low false positive rate. A train ratio of 0.01 means that the training data consists of 0.8% attack data and 99.2% normal data.

For a final test, the train and label ratio were both put to 0.05 to simulate 5% attack data and 5% labeled data (table 6.2).

Again, the precision stays on a high level. Without further interpretation, it can be said that the intrusion detection approach performs well on the HCRL dataset regarding the attacks DoS, Fuzzy, Gear, and RPM and regarding the detection of known attacks.

Next, the ROAD dataset was used for the evaluation of known attacks. During the evaluation, two scenarios were inspected. One included masquerade attacks and the other one did not include masquerade attacks. This results from the observations about the altered CAN ID sequences that were made before. Two experiments were conducted in the first place. The first one used the same conditions for training as were used by Hoang and Kim. All ROAD attack datasets including masquerade attacks were split with a train ratio of 0.7 and a label ratio of 0.1. The model was trained accordingly for 100 epochs and a batch size of 64. The relation of normal and attack data is almost even in that setting, with 56.7% benign data and 43.3% attack data. The second experiment added some more benign data to the split to increase the number of data samples. This was done to make the experiments with HCRL and ROAD more comparable as HCRL has around 400,000 samples in total for the standard data split and ROAD only around 76,000. With the addition of more benign data, ROAD could produce around 289,000 data samples with 88.6% of benign data and 11.4% of attack data. This distribution is not optimal. On the other side, the model could handle this for HCRL data as shown before.

Experiment	Error rate	Recall	Precision	F1
No additional data, all attacks	45.08%	0.4122	0.4799	0.4435
Additional data from the ambient highway street driving long capture, all attacks	52.6%	0.5734	0.1209	0.1997

Table 6.3: The table shows the performance for ROAD attacks with a label ratio of 0.1 and a train ratio of 0.7. Recall and precision are a lot lower than for HCRL attacks. The error rate is around 50% in both cases. Adding more benign data even shrinks the performance.

Table 6.3 shows the results. The model performs badly when classifying the ROAD dataset. Additional normal data does not increase the performance. In contrast, the

performance gets even worse for the precision. This implies an increasing false positive rate.

About half of the attacks of the full ROAD dataset are masquerade attacks, which this approach almost certainly cannot detect without any adjustments. This is why the next experiments were conducted without masquerade attacks. Thus, the attacks now contain the fabrication attacks reverse light on/off, max engine coolant temperature, max speedometer, fuzzing, and correlated signal attack. All of them alter the timing of the frames and the CAN ID sequence consequently. Again, the same two experiments were conducted.

Experiment	Error rate	Recall	Precision	F1
No additional data, without masquerade attacks	44.22%	0.4723	0.5097	0.4903
Additional data from the ambient highway street driving long capture, without masquerade attacks	46.62%	0.5732	0.0841	0.1467

Table 6.4: The table shows the performance for ROAD fabrication attacks with a label ratio of 0.1 and a train ratio of 0.7. Recall and precision are low again. The experiment without additional benign data produces slightly better results than before. Adding more benign data decreases the performance again.

As can be seen in table 6.4, fabrication attacks are not detected well either. In fact, it appears that attack samples from the ROAD dataset cannot be differentiated from benign samples by the model.

More experiments were executed as a reaction, especially regarding the addition of more benign data to have more samples for training. Also, other ambient captures were added to the benign data split like dynamometer captures. Significant improvements could not be achieved; thus, these experiments are not shown here. Section 6.2 contains an analysis of why exactly the proposed method does not work for the ROAD dataset. As the model apparently could not learn the classification, also the evaluation for unknown attacks was omitted.

6.1.2 X-CANIDS

As already mentioned, Jeong et al. used their own datasets for the experiments. For the fabrication attacks, they achieved an average F1 score of 0.931330 with an average precision of 0.990913 and an average recall of 0.910068. For their masquerade attacks, they achieved an average F1 score of 0.927607 with an average precision of 0.990395 and an average recall of 0.912386. The suspension attacks were detected less well with an average recall of 0.424050 and an average precision of 0.976485. The F1 score is 0.545980 in this case. These are the results that can be referred to when evaluating the approach with the new implementation and different models. All the parameters are derived in chapter 5 and can be studied there. The following sections contain the model training and intrusion detection results.

6.1.2.1 SynCAN

For the fitting of the model, a batch size of 64 was used. The model was trained for 400 epochs with an early stopping patience of 50. 400 epochs were chosen because the model seemed to converge fast compared to the literature model. Of course, the model can be trained for 2000 epochs as in the literature. However, the benefit of that should be limited considering that the model loss is already more than four times lower than the literature model after 400 epochs.

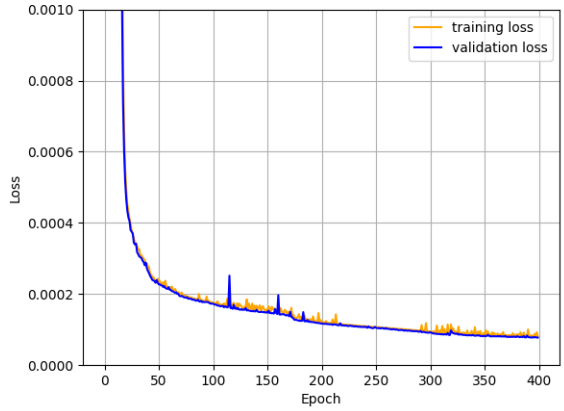


Figure 6.1: The figure shows the loss curve of validation and training over the training epochs. The validation and training loss are close to each other. The training stops with a minimum validation loss of about $7.8 * 10^{-5}$.

Figure 6.1 contains the development of the validation and training loss throughout the training process. In the end, the model has a validation loss of about $7.8 * 10^{-5}$. The trained model was used to determine the $\theta_i s$ on the validation set and training set as described in chapter 5. In contrast to Jeong et al., the intrusion detection threshold Θ was not determined directly and was instead used as an evaluation parameter.

Table A.2 contains all results that were obtained per masquerade attack depending on the parameter q while table A.1 contains all results for the fabrication respectively suspension attacks. Generally, all experiments show that a high intrusion threshold with $q = 0.999$ produces a high precision of more than 0.99 and a low false positive rate. As the threshold decreases, the recall increases. The best F1 scores are scored with $q = 0.99$, or $q = 0.999$. The flooding and plateau attacks are detected best with a recall of 0.941927 respectively 0.892706 for $q = 0.99$ while the continuous attack is detected worst with a recall of 0.715527 for the same q. Although suspension attacks were detected poorly in the original publication [21], the suppress attack is detected relatively well with a recall of 0.745925 for $q = 0.99$. Furthermore, attack-free test data produces a false positive rate of around 0.001127 with $q = 0.999$.

Without further analysis, the SynCAN attacks get detected well considering the complexity of the attacks. Still, SynCAN cannot be seen as a fully realistic benchmark as the number of signals and signal streams is a lot lower compared to real vehicle CAN

buses. Besides, the data was already normalized, and all benign data seems to be equally distributed as the validation loss is close to the training loss (see figure 6.1).

6.1.2.2 ROAD

After having assessed the method in a synthetic setting, it was tested on more realistic data.

Figure 6.2 shows the loss curve of the model training. In contrast to SynCAN, the validation loss is now notably higher than the training loss. The model stopped training after 332 epochs and achieved a minimum validation loss of about $5.243 * 10^{-3}$. After that, the standard method to calculate the intrusion thresholds was used based on the training and validation data. For a percentile $q = 0.98$, an intrusion threshold of about $\Theta = 50.996681$ would apply, which is comparable to the intrusion threshold of Jeong et al. [21].

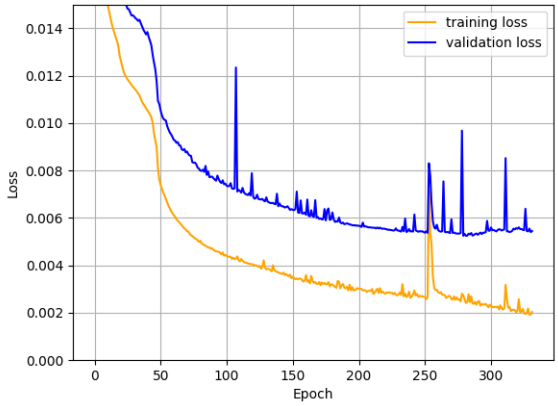


Figure 6.2: The figure shows the loss curve of training and validation loss over the training epochs for the signal-extracted ROAD dataset. The training stops after 332 epochs and a minimum validation loss of about $5.243 * 10^{-3}$.

Tables A.3 and A.4 summarize all intrusion detection performance metrics for the attacks correlated signal attack, max engine coolant temperature attack, max speedometer attack, reverse light on/ off attack, and suspension attacks. As pointed out in chapter 4, the signal-extracted ROAD dataset only provides masquerade attacks as signal extractions. The suspension attacks were created by taking the benign parts of max speedometer attack 1 and dropping the corresponding frames in the attack interval. Three suspension attacks were implemented. One had a signal stream with a period of 10ms as a target, one had a target with a period of 100ms, and the last one had a target with a period of 1s. As can be identified in the tables, the results were different from attack to attack. For the max speedometer attack and the max engine coolant temperature attack, the method worked almost perfectly with a maximum F1 score of about 0.997522 ($q = 1$) respectively 0.987741 ($q = 0.999$) in the best case. The correlated signal attack datasets produce only positive classifications. This is a sign of a bad reconstruction of the entire attack dataset. The

reverse light on/ off attacks deliver a mixed result. For $q \geq 0.99$, no attacks are detected while for $q \leq 0.98$ the F1 score increases step by step. False negatives as well as false positives are produced below this threshold. For the reverse light on attacks, the model predicts only positive classifications below a threshold of $q \leq 0.85$. For all the suspension attacks, the results are relatively similar to the reverse light on/ off results. The crucial threshold is again $q \geq 0.99$. Above that threshold, only negative classifications exist. Below that threshold, false positives and false negatives are produced. For $q \leq 0.9$, only positive classifications are produced. The benign test samples produce a false positive rate of 0.078340 for $q = 0.99$, and 0.001711 for $q = 0.999$.

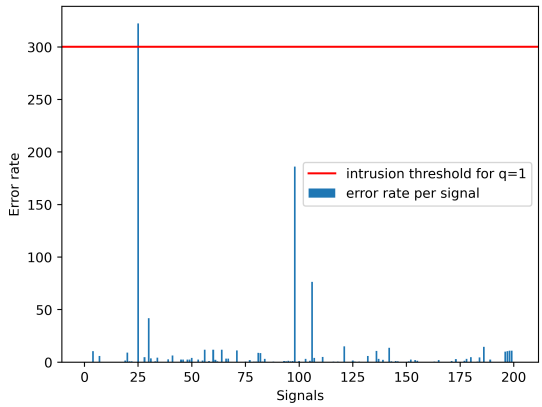


Figure 6.3: The figure shows the error rates of an attack sample. The bar exceeding the intrusion detection threshold Θ is the error rate of the 5th signal of ID 208 which is the speedometer value.

Another aspect pointed out by Jeong et al. is that the intrusion detections are explainable for an analyzing entity. Figure 6.3 shows the resulting error rates for an input sample from the max speedometer attack after the use of the inference function $f(S)$. The error rate of the 5th signal of ID 208 exceeds the example threshold for $q = 1$. Thus, the attacked signal could be identified after analyzing the error rates.

6.1.2.3 ROAD byte-based

An important aspect of this work is the comparison of byte-based intrusion detection and signal-based intrusion detection. Jeong et al. claimed a worse performance for a model that learns only the byte-based payload of the CAN data. However, they justified their assumption with the comparison based on a stationary capture. It could therefore be a difference to use a non-stationary dataset for that matter.

Figure 6.4 shows the loss curve. As can be identified, the model ran into the early stopping later than for both the other, signal-based datasets. The model stopped after 705 epochs with a minimum validation loss of about $1.6004 * 10^{-2}$. The final validation loss is higher than for the signal-based datasets and the learning steps of the model were slower than for the signal-based datasets. The thresholds were determined using the

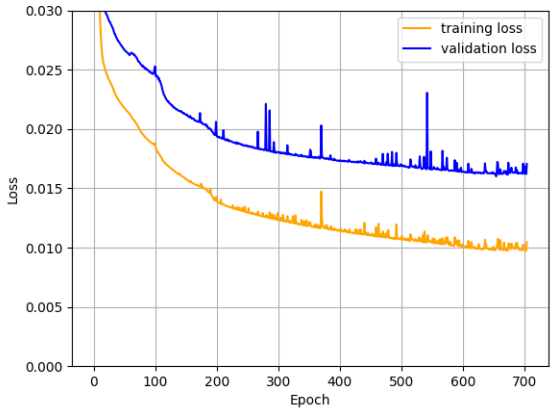


Figure 6.4: The figure shows the loss curve of training and validation loss over the training epochs for the byte-based ROAD dataset. The training stops after 705 epochs with a minimum validation loss of about $1.6004 * 10^{-2}$.

well-established method. A sensitivity parameter $q = 0.98$ would imply a threshold of $\Theta = 51.441104$.

For the evaluation, the focus was on the comparison of the intrusion detection performance under the same conditions. The byte-based ROAD dataset contains fabrication attacks next to the masquerade attacks. Each of the masquerade attacks from previous sections (max speedometer, max engine coolant temperature, correlated signal, reverse light on/ off) is also available as a fabrication attack in the byte-based dataset. Additionally, Bridges et al. [55] provided fuzzing attacks. The evaluation here made use of the masquerade attacks and the corresponding fabrication attacks. First, the performance of masquerade attacks was compared to the signal-translated version of the intrusion detection approach. Second, the fabrication attacks were evaluated to assess the potential of X-CANIDS on these types of attacks. Fuzzing attacks were not included in the evaluation because the fuzzing from the ROAD dataset includes the fuzzing of IDs. This makes feature extraction concerning the established method impossible as the model and feature dimensions are fixed to the benign number of signal streams. Suspension attacks were included as well. They were created the same way as the suspension attacks for the signal-translated dataset. A first look at the results (tables A.5, A.6, A.7) shows that the correlated signal attacks get detected better for the byte-based version of X-CANIDS compared to the signal-based version. This applies to fabrication as well as masquerade attacks. For a sensitivity parameter $q = 1$, an F1 score of 1.0 is reached. Next, the max speedometer attacks and max engine coolant temperature attacks are also detected almost perfectly for $q \geq 0.99$. Judging by the lower false positive rates and higher precision, the benign samples of the attack captures reconstruct better for the byte-based version than for the signal-translated version. The performance for the reverse light off/ on masquerade attacks of the byte-based version is better compared to the signal-translated version. This can be identified by the lower false negative rate and a higher precision. For instance, for $q = 0.9$, the signal-translated version delivers a false negative rate of 0.810569 and a

precision of 0.462680 regarding the reverse light off masquerade attack. The byte-based version delivers a false negative rate of 0.033243 and a precision of 0.745672 in this case. Regarding the reverse light on attack, the false positive rate is far lower in the case of the byte-based version. It produces a false positive rate of 0.083941 for $q = 0.9$ while the signal-translated version produces a false positive rate of 0.988336 in this case. Generally, the byte-based model can divide the benign and attack samples better. Another interesting part is the comparison of fabrication and masquerade attacks for the byte-based model. For the correlated signal attack, max engine coolant temperature attack, and max speedometer attack, the detection performance is the same or almost the same. However, the reverse light off/ on attacks produce a lot more false negatives as fabrication attacks while the false positives stay on a comparable level. The suspension attacks were detected with a performance that was even worse than for the signal-translated version of the IDS. For $q = 0.9$, the suspension attacks targeting a signal stream with a period of 0.1s respectively 1s got detected with a recall of about 0.23 respectively 0.20. The other suspension attack on the high-frequent signal stream was detected with a recall of about 0.003. The attack-free samples produced a false positive rate of 0.279533 for $q = 0.99$, 0.102647 for $q = 0.999$, and 0.025050 for $q = 1$. Interestingly, the false positive rate is higher in the case of the benign test data than in the case of the attack captures. Indeed, the minimum loss at the end of the training is higher in the case of the byte-based version of X-CANIDS than in the case of the signal-translated version of X-CANIDS.

6.1.2.4 Performance on an Embedded Device

An important aspect of the IDS is whether it can be deployed on each entity in the complete system or if data needs to be collected and sent to a central unit for intrusion detection. This refers to the earlier introduction of distributed IDSs. A trend in the publications about intrusion detection on in-vehicle networks is the inclusion of a test of the inference time of the model that was used on an embedded device. By that, a real-time ability or at least close to real-time ability is supposed to be proven. A timing requirement for this work needs to be defined more concretely. In case of an attack on a vehicle that would have safety consequences, real-time detection does not help the passengers of the vehicle. Instead, the goal should be to be able to detect an intrusion within a short time after it occurred. This would still fulfill the criteria of online detection as defined in chapter 2.

To show feasibility in that regard, an NVIDIA Jetson AGX Xavier was leveraged. Technical specifications are defined in chapter 5. The inference times were measured by predicting a set of samples and taking the median computation time of all batches. Table 6.5 summarizes the inference time for different batch sizes per batch and per sample. The time per sample decreases with increasing batch size. The results are comparable to the ones of Jeong et al. [21]. However, their inference times were slightly lower than in this case. For a batch size of 512, an inference time of about 1.4ms can be achieved. Higher batch sizes would make even lower times per sample possible. However, this would make the system wait too long to fill up a batch. For a batch of 512, the system would need to wait at least 512oms with parameter $t = 10ms$. This is why a small batch size would be taken in practice as Jeong et al. already explained [21].

Batch size	Time (ms/batch)	Time (ms/sample)
4	36	9
8	44	5.5
16	62	3.875
32	69	2.15625
64	118	1.84375
128	214	1.671875
256	400	1.5625
512	708	1.382813

Table 6.5: The table shows the rounded inference times of the signal-translated version of X-CANIDS (see table 5.8 for details) per batch and sample (rounded to millionths for the time per sample) depending on the chosen batch size. For an increasing batch size, the inference time per sample decreases. A batch size of 16 is advantageous to fill batches fast and have a good enough inference time per sample [21].

6.1.3 X-MVBIDS

Finally, X-MVBIDS was evaluated. The model training was executed with the same hyperparameters as before regarding the epochs (2000 with an early stopping patience of 50) and the learning rate (10^{-4}). The batch size was reduced to 64 respecting the small number of available samples. In total, 37,060 samples were available. 29,916 samples were used for training respecting the data split relations. Figure 6.5 visualizes the loss curve. The training stops after 60 epochs with a minimum validation loss of $2.4341 * 10^{-2}$.

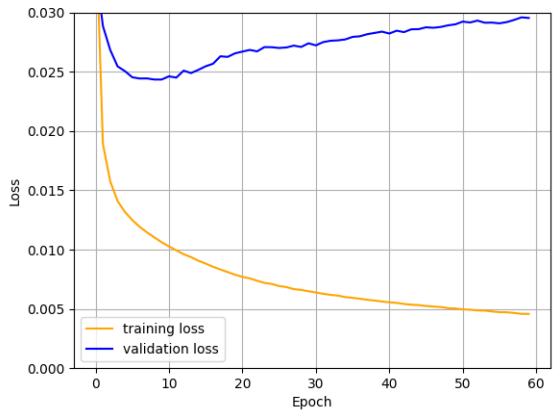


Figure 6.5: The figure shows training and validation loss over the training epochs for the MVB dataset. The training runs into early stopping quickly after 60 epochs with a minimum validation loss of $2.4341 * 10^{-2}$.

After the training, the threshold determination was executed, and the intrusion detection performance was evaluated using the sensitivity parameter q as a hyperparameter.

Table A.8 is about the intrusion detection performance for masquerade attacks. As can be identified, the overall performance is outstanding above $q = 0.99$ with an F1 score

of one or almost one for all three attacks. The precision is one in all three cases. Below $q = 0.99$, the performance shrinks step by step.

Surprisingly, fabrication attacks are not detected for $q \geq 0.85$. None of the attack input matrices get classified as positive above the sensitivity $q = 0.99$. Below $q = 0.99$, some false positives are produced. For suspension attacks, the same results as for the fabrication attacks are achieved. This is an indication that the model cannot separate benign samples from attack samples. The performance results are noted in tables A.9 and A.10.

In summary, the results are split up into the masquerade attacks, which are detected reliably, and into the suspension and fabrication attacks which are not detected at all using a realistic intrusion threshold. Section 6.2 gives the reasoning behind this behavior and explains why the method could still be beneficial. Additionally, it can be said that above a certain threshold ($q = 0.99$) no false positives are generated by the intrusion detection method. This is also confirmed by the attack-free test dataset and the 5% benign data that were part of the data split. No false positives for $q \geq 0.99$ respectively $q \geq 0.96$ were produced.

6.2 Result Analysis

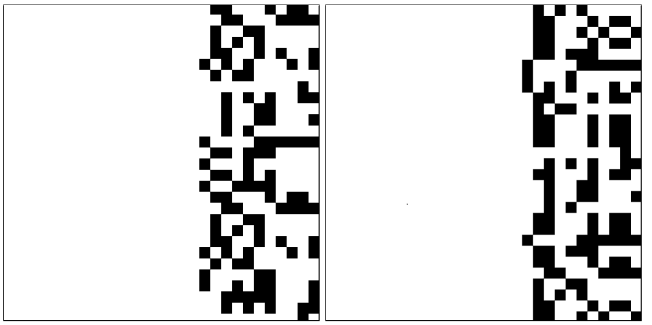
This section analyzes the results that were achieved by the semi-supervised CAAE approach, and the unsupervised approaches X-CANIDS and X-MVBIDS.

6.2.1 Detecting In-Vehicle Intrusion via Semi-supervised Learning-Based CAAEs

The results presented in section 6.1 clearly indicate that the approach does not work for the ROAD dataset. This section shall give reasons. As already introduced before, the HCRL dataset, which is used by most publications so far, has some downsides. It has only fabrication attacks of simple complexity. Although the DoS, Fuzzing, Gear, and RPM attacks are different in their impact, they all follow the same pattern of injecting messages with a high frequency. This high frequency might serve the goal of causing physical behavior but is not stealthy and easy to detect. Technically, no machine learning would be needed to detect such intrusions. A simple component connected to the bus that checks the timing of the messages on the bus against the expected timing (tolerating some delta probably) could detect these attacks. Bridges et al. already mentioned this problem when they introduced the ROAD dataset [55].

Figures 6.6a and 6.6b show a pixel mapping from an attack input frame vs. a normal frame. Although they might look similar in the beginning, a closer look shows significant differences. The attack frame has several identical rows, sometimes even directly after one another whereas the normal frame does not. This visualizes the timing difference mentioned before.

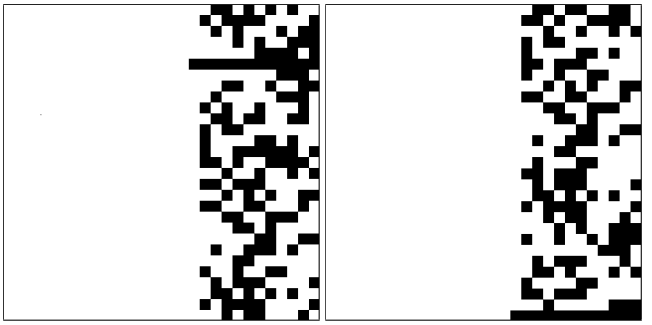
The next two figures 6.7a and 6.7b show an attack frame on the right from the correlated signal attack out of the ROAD dataset next to a normal frame out of the ROAD dataset on the left. Next to the observation that these look more alike, it can be said that there is no case where the same CAN ID appears twice in a row for the attack frame.



(a) The figure shows a benign frame with the dimensions 29×29 from the HCRL RPM dataset after the preprocessing.

(b) The figure shows an attack frame with the dimensions 29×29 from the HCRL RPM dataset after the preprocessing.

Figure 6.6: The two figures show a comparison of HCRL benign and attack input frames after the preprocessing as a pixel mapping. Zero bits correspond to a white pixel while one bits correspond to a black pixel. The attack frame differs significantly from the benign frame. For instance, the malicious CAN message gets injected multiple times in a row.



(a) The figure shows a benign frame with the dimensions 29×29 from the ROAD correlated signal attack dataset after the preprocessing.

(b) The figure shows an attack frame with the dimensions 29×29 from the ROAD correlated signal attack dataset after the preprocessing.

Figure 6.7: The two figures show a comparison of ROAD benign and attack frames after the preprocessing as a pixel mapping. Zero bits correspond to a white pixel while one bits correspond to a black pixel. The attack frame does not contain any ID twice in a row. It can also be identified that the occurring IDs are mostly different in both cases.

An additional problem of the proposed feature extraction is that it does not consider that different datasets have also a different number of signal streams. With the 106 signal streams of ROAD, the 29×29 dimension is not enough to capture a representative sequence of CAN frames. The model can simply not learn the necessary patterns with the current feature extraction.

With all these constraints considered, it becomes clear that this semi-supervised approach is not suitable for intrusion detection. To make a more performant detection possible, significant changes to the feature extraction like including the actual CAN payload

and increasing the input dimensions would be necessary. Consequently, the dimensions of the whole network would grow. This is why the decision was made to not follow this approach anymore.

Furthermore, it was shown that the HCRL dataset is not a sufficient benchmark. Bridges et al. already suspected that in their publication [55] and it is now shown with a concrete example. The experiments and evaluation showed as well that an inappropriate benchmark method can lead to wrong conclusions regarding the performance of an intrusion detection method. In fact, Berger et al. already worked out in 2018 that the HCRL dataset has a simple structure and could achieve an accuracy of 99% for intrusion detection on this dataset with a neural network of just five neurons in the hidden layer [6]. Of course, this was supervised learning in this case. Still, it shows the problem this dataset has of being not complex enough to be a reliable benchmark.

6.2.2 X-CANIDS

Since X-CANIDS produced more promising results, more aspects of the performance and practicality of the approach need to be analyzed.

6.2.2.1 Differences between ROAD and SynCAN

The evaluation results of X-CANIDS on the datasets SynCAN and ROAD differ from each other. While the model learns well with the SynCAN data and produces generally good results for precision and recall, the results are mixed for ROAD data.

Starting with SynCAN, every attack that was under test could achieve a precision of at least 0.99 assuming $q = 0.999$. The recall differs between the attacks and is generally higher for less complex attacks and lower for more complex attacks. It is a positive sign that all attacks that were featured by the SynCAN dataset deliver an F1 score of at least 0.73 assuming a $q = 0.99$. This speaks for the potential of the model as it even exceeds the performance of the CANet model regarding certain metrics, for which the SynCAN dataset was originally developed [17]. For instance, CANet delivers a recall of 0.613 for the suppress attack while X-CANIDS delivers a recall of about 0.666 respectively 0.746 for $q = 0.999$ respectively $q = 0.99$. In summary, the method is very feasible for the SynCAN data considering the low false positive rate it achieves. Furthermore, some of the presented attacks of SynCAN are very sophisticated and to the best knowledge have not been conducted in real life so far. On the other hand, the synthetic nature of SynCAN caused an almost perfect learning process where the training data and test data are equally distributed. In real life, this could differ.

A good example of this are the results that were achieved for the ROAD data. The production of only positive classifications for the correlated signal attack indicates a different driving behavior during the attack capture compared to the benign driving. This is something that can happen at any time during daily driving and is one reason why unsupervised methods are prone to generate false positives in case of intrusion detection. A takeaway message from the comparison of both datasets and the intrusion detection on them is that good training data is necessary to achieve a low false positive rate. Good means that the training data covers as much as possible driving situations and that the

volume of data is enough. This allows to have a lower intrusion threshold Θ and thus detecting more attacks without producing too many false positives becomes possible.

Considering that ROAD is the only dataset available so far that provides real captures of an attacked car that is driving at the same time, the outstanding benchmark of Jeong et al. [21] cannot be seen as completely realistic. As mentioned before, it is assumed that they created the intrusions by manually changing the captured datasets.

6.2.2.2 Masquerade Attacks

Masquerade attacks are mostly detected with a high recall. This applies for the SynCAN attacks plateau and playback as well as for the max speedometer attack and the max engine coolant temperature attack of ROAD. The correlated signal attack of ROAD gets detected with high recall for the byte-based implementation. For the signal-translated version of ROAD, all samples get classified as attacks. On the other side, some masquerade attacks are detected less well, with a lower recall. These are the reverse light on/ off attacks from ROAD and the continuous attack from SynCAN.

This exposes a weakness of X-CANIDS. Attacks that set their target signal/ byte field to maximum or at least to a significantly different value (max speedometer attack, max engine coolant temp attack, and plateau attack) get detected well because they result in a relatively high reconstruction error. Stealthier attacks like the continuous attack overwrite the target little by little. Thus, they cause less reconstruction error in total. The reverse light on/ off attacks switch a byte field of $0C$ to 04 respectively the other way around. This could result in a smaller reconstruction error than setting the byte field to FF like the max speedometer attack depending on the scaled values. To detect these attacks, a lower intrusion threshold would be necessary. This would most likely result in a high false positive rate.

6.2.2.3 Fabrication Attacks

A true comparison of fabrication attacks and masquerade attacks was possible by implementing the byte-based version of X-CANIDS. Furthermore, the SynCAN dataset provided the flooding attack. The latter one could be detected with an F1 score of 0.920854 for $q = 0.99$. This performance is expectable considering that flooding the bus with messages of a certain ID is particularly unstealthy. The ROAD fabrication attacks were detected less well than the corresponding masquerade attacks in the case of the reverse light on/ off attacks. An indication of that is the increased false negative rate regarding the fabrication attacks. The question is why exactly this happens. Intuitively, fabrication attacks should be detected even better because they influence the timing of the CAN frames. Looking at the feature extraction of X-CANIDS, it can be noticed that the parameter t is always set to the lowest occurring base period of signal streams. In the case of ROAD, this is $t = 10ms$. The fabrication attacks of ROAD overwrite the corresponding signal stream just after it has been sent on the bus. A look into the captures confirms that this can happen already after 2ms. This means that the payload sampler of x-canids would ignore this payload and directly take the next, benign signal payload and pass it to the deserializer. This could happen for attacks on the most high-frequent signal streams. Since the CAN frames do not arrive perfectly on time every period, it can also happen that the malicious frames

get processed by the payload sampler. Consequently, some attack frames are ignored by the IDS and are not included in the sample matrix S. The malicious samples still create enough loss regarding the max speedometer attack, correlated signal attack, and max engine coolant temperature attack. However, the loss for the reverse light on/ off attacks is already low for masquerade attacks and would now become even lower for the fabrication attacks.

6.2.2.4 Suspension Attacks

Suspension attacks do not get detected well in general. For SynCAN, the suppress attack is detected best out of all suspension attacks with an F1 score of 0.831385 and $q = 0.99$. The suspension attacks of ROAD get detected worse. In both versions, signal-translated and byte-based, the produced loss is not high enough to separate attack and benign samples. There is a particular reason for these performance issues. The problem is that during the feature extraction, the payload sampler always caches the latest signal payload for all signal streams. This means that a sample matrix S always contains the same amount of signal streams, no matter if a suspension attack is launched or not. Following that thought, the autoencoder would reconstruct with high losses only when the attacked signal stream would change its values often under attack-free circumstances. Thus, it highly depends on the concrete attacked signal stream if the intrusion can be detected or not. This is also confirmed by the results of Jeong et al. who could detect just a few suspension attacks well [21]. In the case of this evaluation, the attacked signal stream with a period of 10ms was the one of ID 208. It contains the speedometer value and reverse light on/ off values next to one other signal. The other ones were excluded as constant signals according to the method. In the case of the speedometer, the speed stays at a constant value, and the reverse lights stay on a constant value (probably off) too. This could be normal driving behavior. In summary, the method is not well suited to detect suspension attacks.

6.2.3 X-MVBIDS

The adaption of X-CANIDS to MVB could be successfully conducted. However, it remains a problem that there are no publicly available datasets (challenge C5). The proprietary dataset of this thesis made it possible to get a general idea of the intrusion detection performance. However, the length of the captures and the fact that the captures were obtained from a stationary train made it hard to get a realistic benchmark. Especially, the number of constant byte fields might change for a non-stationary capture. Still, it was shown that the method is indeed applicable to MVB respecting the protocol-specific aspects. The next two subsections contain some discussions about intrusion detection results on masquerade attacks on the one hand and fabrication respectively suspension attacks on the other hand.

6.2.3.1 Masquerade Attacks

Masquerade attacks are detected well for all three implemented attacks. It should be noted that the implemented attacks followed the pattern of some masquerade attacks of the ROAD dataset. This means the attacked byte fields were set to maximum respectively

minimum. More stealthy or continuous attacks could be potentially detected with a lower recall. In practice, this problem can be mitigated by having more training data to make the model fit better and by that reducing the threshold for intrusion detection.

6.2.3.2 Suspension and Fabrication Attacks

Suspension and fabrication attacks do not get detected at all for a realistic threshold $q \geq 0.85$. There are two different explanations for that. Suspension attacks suffer from the general weakness of the method as discussed before. In this case, the detection is even worse because the model could not fit the MVB data well enough, a result of the data sparseness. This effect can probably be mitigated with more data. Regarding the fabrication attacks, there is a different reason for the poor performance. The fabrication attacks that were implemented for MVB use the short time window, in which slave responses are allowed after the master request was sent. As already mentioned in chapter 2, this time window is 42.7us. When two slave frames are received during this time, only the first one will get accepted. Assuming an attacker would be able to conduct such an attack, this would have severe consequences for the feature extraction. For each extracted vector s, the attacked slave frame would be overwritten by the benign one because the parameter $t = 16ms$ is too big, and thus, only the latest slave frame would be processed further in a time window of 16ms. The effect is similar to the fabrication attacks in the ROAD dataset. This effect can be mitigated by extremely decreasing the parameter t. On the contrary, this could exhaust the computational resources and the feasibility of the approach might not be given anymore. Although this makes the approach look nonperforming, it is very unlikely that an attacker can mount this type of attack. As MVB has strict slots for each slave response, other types of fabrication attacks cannot be launched on slaves without causing bus errors. Masquerade attacks seem to be more feasible for an attacker in this case. Other attacks could try to reconfigure the whole bus and mount fabrication attacks. Based on the available datasets, this could not be evaluated.

7 Discussion

This chapter discusses the results of the evaluation of the semi-supervised method and the unsupervised method regarding the feasibility in practice in the case of joint IDS. It is discussed whether it is possible to overcome the introduced challenges from chapter 5. The second section of this chapter contains a proposal for an intrusion detection architecture as part of a joint security monitoring system.

7.1 Feasibility in Practice

The semi-supervised approach by Hoang and Kim cannot be seen as applicable to any real-life application. It performs well on the HCRL dataset but is not able to detect any complicated attack that cannot be detected by a timing-based IDS. It can be finally concluded that feasibility in practice is not given. This also makes the approach unsuitable regarding the research question of how a joint IDS can be realized.

X-CANIDS can be beneficial in practice despite its weaknesses that were explained above. Especially for a distributed system with multiple vehicles, the method is applicable. More details can be found in the according section below. The measurement of inference times shows that the application of the method on ROAD led to a bigger model but still produced good enough inference times to be deployed on an edge device. Thus, it is possible to overcome challenge C3. The model size could be further reduced by trying to use alternative RNN cells like GRUs. Challenge C1 considers the problem of having enough labeled data. This problem is solved by using the unsupervised method X-CANIDS. It is still necessary to have huge amounts of data from diverse driving situations for good model training.

The proposed method should not be used as a standalone in practice. The detection performance of masquerade attacks and unstealthy fabrication attacks is convincing. On the other side, stealthy fabrication attacks and suspension attacks are very hard to detect using this method. It is therefore proposed to combine X-CANIDS with other intrusion detection methods in practice. A good choice would be a timing-based detector like the one proposed by Cho et al. [8]. Timing-based detectors are well suited to detect suspension and fabrication attacks because they can detect deviations from the base period of signal streams. This would be a suitable addition to X-CANIDS, which is more performant on masquerade attacks. Regarding the sensitivity parameter q, it is recommended to set it to a high value $q = 1 - \epsilon$. By that, a low false positive rate of normally below 0.01 is assured. The recall would become lower by that as well. However, it is not crucial to detect every single attack sample of an intrusion. Depending on the concrete attack, multiple attack frames are usually injected to have a permanent effect on the vehicle.

Regarding the exclusion of constant signals and signals that are checksums or depend in another way on other signals, it should be noted that X-CANIDS is only suited to monitor

the most important signal streams. Full inclusion of all signals would result in a big model and would cause unnecessary monitoring of constant signals. This is one more reason to combine X-CANIDS with other methods. In the context of FINESSE [45], AI-based intrusion detection is supposed to be combined with rule-based intrusion detection. This poses another useful addition to X-CANIDS. Rules could be used to check if constant signals change or if checksums, parity bits, etc. are correct.

Challenge C4 was addressed by testing the byte-based version of X-CANIDS. It was shown that signal translations are not necessarily needed for good intrusion detection performance. In the case of ROAD, the byte-based model performed even better than the signal-translated one, e.g., for the correlated signal attack. This contradicts the statements of Jeon et al. [21]. Regarding the research question of how to realize a joint IDS, it shows that a third-party provider could also establish such an IDS without having the proprietary DBC files available.

X-MVBIDS serves challenge C2 as it is an adaption of X-CANIDS to MVB. It is a suitable approach for the detection of masquerade attacks and it was shown that it has comparable model parameters to X-CANIDS. Fabrication attacks as they were introduced and suspension attacks cannot be detected with this approach. This is why a timing-based detector should be added also in this case. Considering the joint monitoring of railway and automotive vehicles, it was shown that the attack classes by Cho et al. [8] can be applied to MVB. However, it can be assumed that more types of attacks are possible. This remains for further research. In a distributed setting, there is a crucial difference between railway and automotive vehicles. Automotive vehicles usually have just a few CAN buses on board. A complete train with multiple coaches could have many more buses. Additionally, the train configurations can change depending on the type. This poses a challenge for a joint monitoring system of multiple vehicles.

7.2 Joint Architecture

Now that the performance of all the intrusion detection approaches is evaluated, chapter 2.4 is picked up again to fill the use cases that are introduced there. As the semi-supervised approach [18] was classified as not suitable for the objectives of this thesis, only X-CANIDS and X-MVBIDS are considered.

Use case 1 considered the training of the model. The idea for the proposed IDS is to train the autoencoders of X-CANIDS and X-MVBIDS on a central entity, e.g., the VSOC. For each bus configuration, an own model needs to be trained respecting the parameters of the method. These parameters need to be determined after an analysis of the training data. After the model training, the threshold determination needs to be conducted. Only benign data is necessary for the training.

Use cases 2 and 3 considered the deployment and intrusion detection. The model weights can be deployed on dedicated embedded devices on the vehicles respectively other types of devices with more computation power if possible. Sticking to the terminology of FINESSE, these are declared as Smart Sensors. They are bus participants and collect the communication data from the bus. They apply feature extraction and intrusion detection according to the method. Next to the model weights, the important sensitivity parameter q is deployed as well as the tolerance values θ_i. q should be picked as high as possible

to avoid false positives. Detected intrusions lead to the generation of alerts. These are accompanied by the corresponding logs and the error rates of the sample that caused an alert. The error rates can be used for explanation in the VSOC as explained in chapter 6.

Finally, use case 4 considered the ability to update the IDS. After the analysis in the VSOC, it might be concluded that the false positive rate is still too high. In this case, q can be adjusted and deployed back on the vehicles. Another example of an update is the availability of more training data. The model could be trained for more epochs or completely trained from scratch. In this case, the model updates can be deployed back to the vehicles.

The described mechanisms are visualized in figure 7.1. It shows an example car fleet of two cars next to an example train fleet of two trains. The introduced architecture can be classified as centralized architecture according to the introduced definitions. The alerts and logs are collected at a central unit. However, the intrusion detection itself, and thus, the pre-filtering of communication data, is conducted on the vehicles.

The benefit of the joint architecture is that security analysts can leverage the joint security considerations of railway and automotive vehicles for a more valuable analysis. This is supported by the similarity of the approaches of X-CANIDS and X-MVBIDS. By that, analysis of attacks creates knowledge for both sectors. This knowledge can be enriched with other external sources. In total, this approach is supreme to the independent monitoring of each single vehicle.

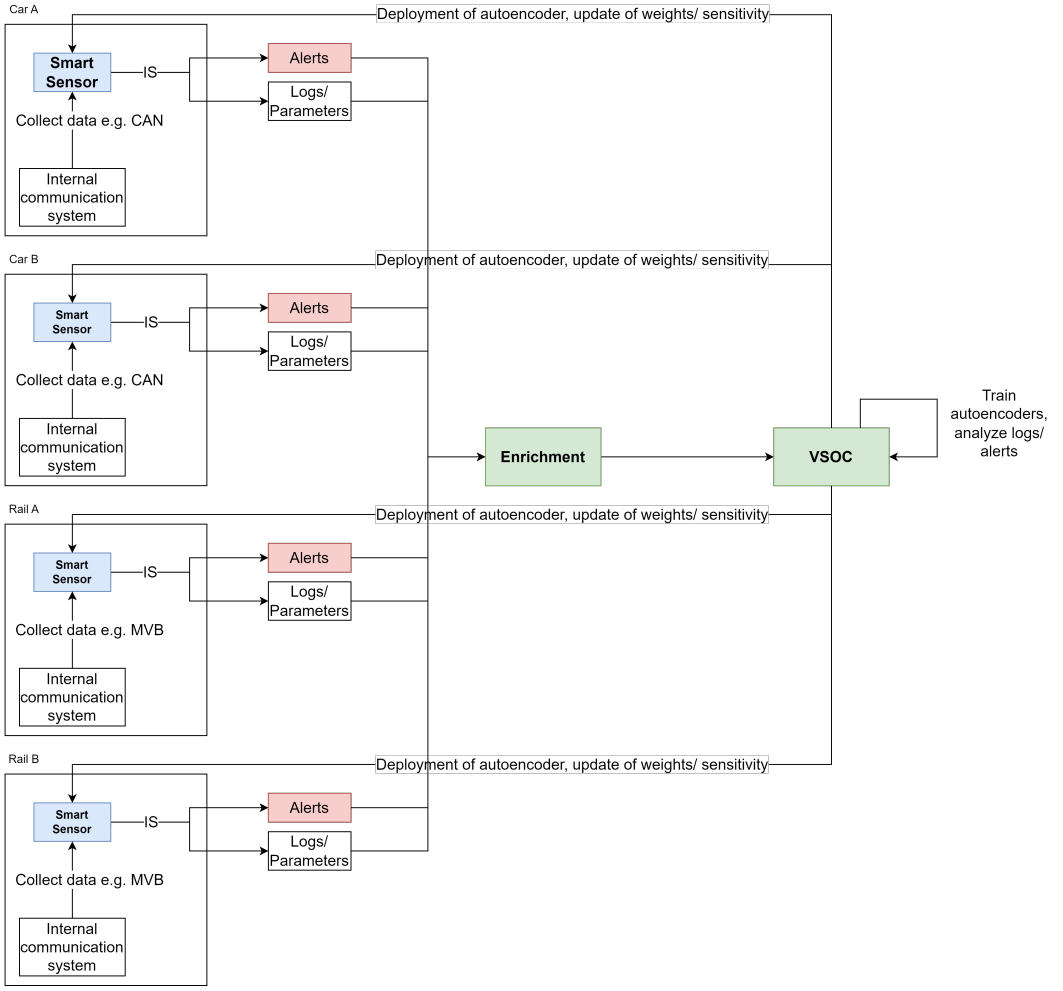


Figure 7.1: This image shows a conceptual architecture of a joint IDS for rail and automotive. The components Enrichment, VSOC, and Smart Sensor were taken from the context of FINESSE [45]. Each vehicle has one or more Smart Sensors, which the model weights get deployed on. The intrusion scores (IS) are calculated for each incoming sample. The IS corresponds to the maximum error rate $max(r)$ and causes an alert if it exceeds Θ. Weights and parameters can be updated as a reaction to further analysis in the VSOC. The Enrichment collects the data and prepares it for analysis in the VSOC. Source: Own figure based on [45]

8 Conclusion

This thesis dealt with the proposal of a joint IDS based on machine learning for a combined car and train monitoring system. After the necessary foundations including the description of the two field buses MVB and CAN, some general use cases for such a joint IDS were introduced. These were kept as generic as possible to allow a wide range of solutions for the given problem. Related work was presented. After this, CAN and MVB were compared to develop a joint security understanding of these. An attacker model was introduced to derive concrete attacks on both in-vehicle networks. With the help of public, widely accepted CAN datasets, and a proprietary MVB dataset, a semi-supervised and an unsupervised machine learning approach were introduced and evaluated. The semi-supervised approach turned out to be unsuitable for the solution of the given problem. In consequence, the focus was put on the unsupervised approach X-CANIDS. It was possible to implement two adaptions of it. One was implemented for the detection of byte-based CAN frames, and the other one for the detection of MVB data.

It was shown that the approach is suitable for certain types of attacks, like masquerade attacks and unstealthy types of fabrication attacks. Other attack types, like stealthy fabrication attacks and suspension attacks, could not be detected well.

Finally, concrete details about the use cases of such a joint IDS were formulated. It was also discussed that X-CANIDS and X-MVBIDS could be combined with other intrusion detection approaches to mitigate the weaknesses and have multiple sources for intrusion detection. Rules and timing-based approaches could be a valuable addition.

The research questions could be answered with a proposal of a concrete approach. The challenges C1 to C5 could be considered and solutions could be found. C1 addressed the problem of the availability of labeled data and attack data. X-CANIDS and X-MVBIDS are unsupervised methods and avoid this problem for the training process. Challenge C2 regarded the differences between CAN and MVB and the differences between datasets. This challenge was solved by adapting X-CANIDS to MVB and training individual models per bus configurations. Quality differences between datasets were named and considered for the evaluations. C3 was about the role of edge devices. An approach could be suggested that allows training at a central entity and detecting (inference) on edge devices such as the NVIDIA Jetson AGX Xavier. The model sizes can potentially be further reduced in the future. C4 regarded the problem of proprietary signal extractions. It was shown that signal translations are not necessarily needed for high precision and recall. C5 posed the biggest challenge of not having publicly available MVB datasets. This was compensated partially by obtaining proprietary datasets. However, the amount of data and the characteristics of it were barely suitable for a realistic benchmark. Thus, the MVB performance assessment of this work can only be understood as a first step.

In conclusion, X-CANIDS and X-MVBIDS have the advantage of unsupervised learning. Only benign data is needed for training. Furthermore, their sensitivity is configurable, and they deliver a mechanism to identify attacked signals respectively byte fields.

8.1 Future Work

Future work could try to realize a proof-of-concept implementation with the live detection of injected frames and could also evaluate the performance outside of a laboratory. The implementation of the live extraction of features differs significantly from the theoretical work with datasets. Moreover, the architecture of a joint IDS towards a real security monitoring system with data exchange, updates, analysis mechanisms, and incident response is an important topic. By that, the results of this work and the current project work of FINESSE could be connected.

It is also crucial to conduct more research on the security and exploitation of railway in-vehicle networks. It is necessary to provide publicly available datasets for the benchmarking and comparison of intrusion detection approaches.

Bibliography

[1] Shaashwat Agrawal, Sagnik Sarkar, Ons Aouedi, Gokul Yenduri, Kandaraj Piamrat, Mamoun Alazab, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, and Thippa Reddy Gadekallu. "Federated Learning for intrusion detection system: Concepts, challenges and future directions". In: *Computer Communications* 195 (2022), pages 346–361. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2022.09.012. URL: https://www.sciencedirect.com/science/article/pii/S0140366422003516.

[2] CAN in Automation. *History of CAN technology*. URL: https://www.can-cia.org/can-knowledge/can/can-history/.

[3] Rebecca Bace and Peter Mell. "NIST special publication on intrusion detection systems". In: *National Institute of Standards and Technology* 16 (2001).

[4] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*. Edited by Lior Rokach, Oded Maimon, and Erez Shmueli. Cham: Springer International Publishing, 2023, pages 353–374. ISBN: 978-3-031-24628-9. DOI: 10.1007/978-3-031-24628-9_16. URL: https://doi.org/10.1007/978-3-031-24628-9_16.

[5] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pages 157–166. DOI: 10.1109/72.279181.

[6] Ivo Berger, Roland Rieke, Maxim Kolomeets, Andrey Chechulin, and Igor Kotenko. "Comparative study of machine learning methods for in-vehicle intrusion detection". In: *International Workshop on Security and Privacy Requirements Engineering*. Springer. 2018, pages 85–101.

[7] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. "Comprehensive experimental analyses of automotive attack surfaces". In: *20th USENIX security symposium (USENIX Security 11)*. 2011.

[8] Kyong-Tak Cho and Kang G. Shin. "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection". In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pages 911–927. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cho.

[9] Cameron Clough and Greg Hogan. *opendbc*. 2023. URL: https://github.com/commaai/opendbc.

[10] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. "Generative Adversarial Networks: An Overview". In: *IEEE Signal Processing Magazine* 35.1 (2018), pages 53–65. DOI: 10.1109/MSP.2017.2765202.

[11] Guillaume Dupont, Alexios Lekidis, J. (Jerry) den Hartog, and S. (Sandro) Etalle. *Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2*. 2019. DOI: 10.4121/uuid:b74b4928-c377-4585-9432-2004dfa20a5d. URL: https://data.4tu.nl/articles/dataset/Automotive_Controller_Area_Network_CAN_Bus_Intrusion_Dataset/12696950/2.

[12] *Electronic railway equipment - Train communication network (TCN) - Part 3-1: Multifunction Vehicle Bus (MVB)*. Standard. IEC 61375-3-1:2012. Rue de Verembé 3, CH-1211 Geneva, CH: International Electrotechnical Commission, June 2012.

[13] Ian Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage. "Fast and Vulnerable: A Story of Telematic Failures". In: *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. Washington, D.C.: USENIX Association, Aug. 2015. URL: https://www.usenix.org/conference/woot15/workshop-program/presentation/foster.

[14] Mario Freund. *can-ids*. 2023. URL: https://github.com/freundma/can-ids (visited on Oct. 16, 2023).

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Edited by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Volume 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[16] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer. *SynCAN*. 2020. URL: https://github.com/etas/SynCAN (visited on Sept. 25, 2023).

[17] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. "CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data". In: *IEEE Access* 8 (2020), pages 58194–58205. DOI: 10.1109/ACCESS.2020.2982544.

[18] Thien-Nu Hoang and Daehee Kim. "Detecting in-vehicle intrusion via semi-supervised learning-based convolutional adversarial autoencoders". In: *Vehicular Communications* 38 (2022), page 100520. ISSN: 2214-2096. DOI: https://doi.org/10.1016/j.vehcom.2022.100520. URL: https://www.sciencedirect.com/science/article/pii/S2214209622000675.

[19] Thien-Nu Hoang and Daehee Kim. *Semi-supervised Deep Learning Based In-vehicle Intrusion Detection System Using Convolutional Adversarial Autoencoder*. 2022. URL: https://github.com/htn274/CanBus-IDS (visited on Sept. 11, 2023).

[20] Seonghoon Jeong, Boosun Jeon, Boheung Chung, and Huy Kang Kim. "Convolutional neural network-based intrusion detection system for AVTP streams in automotive Ethernet-based networks". In: *Vehicular Communications* 29 (2021), page 100338. ISSN: 2214-2096. DOI: https://doi.org/10.1016/j.vehcom.2021.100338. URL: https://www.sciencedirect.com/science/article/pii/S2214209621000073.

[21] Seonghoon Jeong, Sangho Lee, Hwejae Lee, and Huy Kang Kim. *X-CANIDS: Signal-Aware Explainable Intrusion Detection System for Controller Area Network-Based In-Vehicle Network*. 2023. arXiv: 2303.12278 [cs.CR].

[22] Hubert Kirrmann and P.A. Zuber. "The IEC/EEE train communication network". In: *Micro, IEEE* 21 (Apr. 2001), pages 81 –92. DOI: 10.1109/40.918005.

[23] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. "Experimental Security Analysis of a Modern Automobile". In: *2010 IEEE Symposium on Security and Privacy*. 2010, pages 447–462. DOI: 10.1109/SP.2010.34.

[24] Vipin Kumar Kukkala, Sooryaa Vignesh Thiruloga, and Sudeep Pasricha. "INDRA: Intrusion Detection Using Recurrent Autoencoders in Automotive Embedded Systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pages 3698–3710. DOI: 10.1109/TCAD.2020.3012749.

[25] Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. "Intrusion Detection: A Survey". In: *Managing Cyber Threats: Issues, Approaches, and Challenges*. Edited by Vipin Kumar, Jaideep Srivastava, and Aleksandar Lazarevic. Boston, MA: Springer US, 2005, pages 19–78. ISBN: 978-0-387-24230-9. DOI: 10.1007/0-387-24230-9_2. URL: https://doi.org/10.1007/0-387-24230-9_2.

[26] Wenjuan Li, Weizhi Meng, and Man Ho Au. "Enhancing collaborative intrusion detection via disagreement-based semi-supervised learning in IoT environments". In: *Journal of Network and Computer Applications* 161 (2020), page 102631. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102631. URL: https://www.sciencedirect.com/science/article/pii/S1084804520301053.

[27] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. "A survey of convolutional neural networks: analysis, applications, and prospects". In: *IEEE transactions on neural networks and learning systems* (2021).

[28] *Long short-term memory*. 2023. URL: https://en.wikipedia.org/wiki/Long_short-term_memory (visited on Sept. 11, 2023).

[29] Haoyu Ma, Jianqiu Cao, Bo Mi, Darong Huang, Yang Liu, and Shaoqian Li. "A GRU-based lightweight system for CAN intrusion detection in real time". In: *Security and Communication Networks* 2022 (2022).

[30] Umberto Michelucci. "An Introduction to Autoencoders". In: *CoRR* abs/2201.03898 (2022). arXiv: 2201.03898. URL: https://arxiv.org/abs/2201.03898.

[31] Charlie Miller and Chris Valasek. "Remote Exploitation of an Unaltered Passenger Vehicle". In: (Aug. 2015). URL: https://illmatics.com/Remote%20Car%20Hacking.pdf.

[32] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad-Reza Sadeghi. "DÏoT: A Federated Self-learning Anomaly Detection System for IoT". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019, pages 756–767. DOI: 10.1109/ICDCS.2019.00080.

[33] *NumPy*. 2023. URL: https://numpy.org/ (visited on Oct. 30, 2023).

[34] Christopher Olah. "Understanding lstm networks". In: (2015). URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[35] *pandas*. 2023. URL: https://pandas.pydata.org/ (visited on Oct. 30, 2023).

[36] *Railway applications - Cybersecurity*. Standard. CLC/TS 50701:2022 XX. Rue de la Science 23 B-1040 Brussels, BE: European Committee for Electrotechnical Standardization, 2022.

[37] Sampath Rajapaksha, Harsha Kalutarage, M. Omar Al-Kadri, Andrei Petrovski, Garikayi Madzudzo, and Madeline Cheah. "AI-Based Intrusion Detection Systems for In-Vehicle Networks: A Survey". In: *ACM Comput. Surv.* 55.11 (2023). ISSN: 0360-0300. DOI: 10.1145/3570954. URL: https://doi.org/10.1145/3570954.

[38] Sebastian Raschka. "An overview of general performance metrics of binary classifier systems". In: *arXiv preprint arXiv:1410.5330* (2014).

[39] Valerian Rey, Pedro Miguel Sánchez Sánchez, Alberto Huertas Celdrán, and Gérôme Bovet. "Federated learning for malware detection in IoT devices". In: *Computer Networks* 204 (2022), page 108693. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108693. URL: https://www.sciencedirect.com/science/article/pii/S1389128621005582.

[40] *Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling*. Standard. ISO 11898-1:2015. Ch. de Blandonnet 8, CH-1214 Geneva, CH: International Organization for Standardization, Dec. 2015.

[41] *Road vehicles - Cybersecurity engineering*. Standard. ISO/SAE 21434:2021(E). Ch. de Blandonnet 8, CH-1214 Geneva, CH: International Organization for Standardization/ Society of Automotive Engineers, Aug. 2021.

[42] Marco Rocchetto and Nils Ole Tippenhauer. "On attacker models and profiles for cyber-physical systems". In: *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II 21*. Springer. 2016, pages 427–449.

[43] Hojjat Salehinejad, Julianne Baarbe, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. "Recent Advances in Recurrent Neural Networks". In: *CoRR* abs/1801.01078 (2018). arXiv: 1801.01078. URL: http://arxiv.org/abs/1801.01078.

[44] Barabara Schmitz and Thomas Seger. "Informations-und Steuerungstechnik auf Schienenfahrzeugen". In: *Micro, IEEE* 21 (2008), pages 20 –22. DOI: 10.1109/40.918005.

[45] Max Schubert, Jens Gramm, Christoph Krauß, Ali Yekta, Stefan Katzenbeisser, and Jan Eichhorn. "Fahrzeug Intrusion Detektion und Prävention in einheitlicher Struktur für Straße und Schiene". In: (Nov. 2020). Project Proposal.

[46] *Security for Industrial Automation and Control Systems*. Standard. ISA-62443. 67 Alexander Drive, US-27709 North Carolina, US: International Society of Automaton, Apr. 2018.

[47] Eunbi Seo, Hyun Min Song, and Huy Kang Kim. "GIDS: GAN based Intrusion Detection System for In-Vehicle Network". In: *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. 2018, pages 1–6. DOI: 10.1109/PST.2018.8514157.

[48] Dongxian Shi, Ming Xu, Ting Wu, and Liang Kou. "Intrusion detecting system based on temporal convolutional network for in-vehicle CAN networks". In: *Mobile Information Systems* 2021 (2021), pages 1–13.

[49] CAN Specification. "Bosch". In: *Robert Bosch GmbH, Postfach* 50 (1991), page 15.

[50] Dominik Spychalski, Markus Heinrich, and Christoph Krauß. "Rail meets Automotive: Commonalities in vehicle-side IT/OT security considerations". In: volume 114. Signaling + Datacommunication, Nov. 2022, pages 51–57.

[51] *TensorFlow*. 2023. URL: https://www.tensorflow.org/ (visited on Oct. 30, 2023).

[52] *TFRecord and tf.train.Example*. 2023. URL: https://www.tensorflow.org/tutorials/load_data/tfrecord (visited on Sept. 22, 2023).

[53] Jesper E Van Engelen and Holger H Hoos. "A survey on semi-supervised learning". In: *Machine learning* 109.2 (2020), pages 373–440.

[54] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. "Taxonomy and Survey of Collaborative Intrusion Detection". In: *ACM Comput. Surv.* 47.4 (2015). ISSN: 0360-0300. DOI: 10.1145/2716260. URL: https://doi.org/10.1145/2716260.

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

[56] Wilfried Voss. *A comprehensible guide to controller area network*. Copperhill Media, 2008.

[57] Samuel Woo, Hyo Jin Jo, and Dong Hoon Lee. "A practical wireless attack on the connected car and security protocol for in-vehicle CAN". In: *IEEE Transactions on intelligent transportation systems* 16.2 (2014), pages 993–1006.

[58] Chuan Yue, Lide Wang, Dengrui Wang, Ruifeng Duo, and Xiaobo Nie. "An Ensemble Intrusion Detection Method for Train Ethernet Consist Network Based on CNN and RNN". In: *IEEE Access* 9 (2021), pages 59527–59539. DOI: 10.1109/ACCESS.2021.3073413.

[59] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021).

A Appendix– Evaluation Result Tables

Test Suppress					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.812009	0.0	1.0	0.187991	0.316485
0.999	0.334214	0.001158	0.992419	0.665786	0.796932
0.99	0.254075	0.011041	0.938962	0.745925	0.831385
0.98	0.226883	0.021390	0.891655	0.773117	0.828166
0.97	0.206862	0.029738	0.858613	0.793138	0.824578
0.96	0.191262	0.037214	0.831882	0.808738	0.820147
0.95	0.179616	0.045059	0.805659	0.820384	0.812955
0.9	0.158912	0.084520	0.693802	0.841088	0.760378
0.85	0.145630	0.129890	0.599629	0.854370	0.704684
Test Flooding					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.718661	0.0	1.0	0.281339	0.439133
0.999	0.178194	0.001044	0.993955	0.821806	0.899720
0.99	0.107294	0.009638	0.950834	0.892706	0.920854
0.98	0.092339	0.018699	0.910190	0.907661	0.908924
0.97	0.082709	0.028256	0.871432	0.917291	0.893773
0.96	0.076496	0.038132	0.834892	0.923504	0.876965
0.95	0.071864	0.047870	0.801907	0.928136	0.860417
0.9	0.063122	0.106220	0.648080	0.936878	0.766168
0.85	0.057256	0.164248	0.545125	0.942744	0.690804

Table A.1: The table shows the intrusion detection performance of X-CANIDS on SynCAN fabrication (flooding) and suspension (suppress) attacks respecting the sensitivity parameter q with $0.85 \leq q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. The achievable precision is high in both cases. The achievable recall is higher for the flooding attack. With a suitable $q \geq 0.999$, a low false positive rate of below 0.01 can be achieved. False positive rate and false negative rate are counterparts as one rises when the other shrinks.

Test Plateau					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.525436	0.000008	0.999918	0.474564	0.643650
0.999	0.104566	0.001066	0.994273	0.895434	0.942269
0.99	0.058073	0.012501	0.939660	0.941927	0.940792
0.98	0.039682	0.025528	0.886034	0.960318	0.921682
0.97	0.033279	0.036267	0.846365	0.966721	0.902548
0.96	0.028452	0.048349	0.805934	0.971548	0.881026
0.95	0.025864	0.060001	0.770397	0.974136	0.860369
0.9	0.020882	0.125271	0.617639	0.979118	0.757462
0.85	0.018235	0.186524	0.521028	0.981765	0.680769
Test Continuous					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.797070	0.000016	0.999532	0.202930	0.337367
0.999	0.411631	0.000791	0.991834	0.588369	0.738595
0.99	0.284473	0.008340	0.933373	0.715527	0.810060
0.98	0.249044	0.018930	0.866272	0.750956	0.804503
0.97	0.223494	0.029806	0.809671	0.776506	0.792742
0.96	0.206112	0.041039	0.759547	0.793888	0.776338
0.95	0.193551	0.052466	0.715095	0.806449	0.758030
0.9	0.171349	0.114820	0.540963	0.828651	0.654593
0.85	0.155700	0.174332	0.441602	0.844300	0.579895
Test Playback					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.768089	0.0	1.0	0.231911	0.376506
0.999	0.232201	0.000928	0.992522	0.767799	0.865817
0.99	0.129284	0.008595	0.942057	0.870716	0.904983
0.98	0.108267	0.018068	0.887902	0.891733	0.889813
0.97	0.092697	0.026199	0.847513	0.907303	0.876390
0.96	0.080622	0.034380	0.811026	0.919378	0.861810
0.95	0.074476	0.043969	0.771596	0.925524	0.841579
0.9	0.063003	0.090259	0.624916	0.936997	0.749778
0.85	0.056303	0.142893	0.514541	0.943697	0.665969

Table A.2: The table shows the intrusion detection performance of X-CANIDS on SynCAN masquerade attacks (not changing timing) respecting the sensitivity parameter q with $0.85 \leq q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. The achievable precision is high in all three cases. The achievable recall depends on the attack. It is relatively low for the continuous attack compared to the other attacks. With a suitable $q \geq 0.999$, a low false positive rate of below 0.01 can be achieved. False positive rate and false negative rate are counterparts as one rises when the other shrinks.

Correlated Signal Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≤ 1	0.0	1.0	0.799093	1.0	0.888329

Max Engine Coolant Temperature Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.027682	0.0	1.0	0.972318	0.985965
0.999	0.024221	0.0	1.0	0.975779	0.987741
≤ 0.99	0.0	1.0	0.247009	1.0	0.396162

Max Speedometer Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.004944	0.0	1.0	0.995056	0.997522
0.999	0.003543	0.009009	0.992205	0.996457	0.994326
0.99	0.000577	0.529635	0.684695	0.999423	0.812651
0.98	0.0	0.938739	0.550740	1.0	0.710293
0.97	0.0	0.958369	0.545614	1.0	0.706016
0.96	0.0	0.974680	0.541427	1.0	0.702501
0.95	0.0	0.981982	0.539573	1.0	0.700939
≤ 0.9	0.0	1.0	0.535053	1.0	0.697113

Reverse Light Off Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	0.863076	0.105544	0.562024	0.136924	0.220201
0.97	0.857842	0.119071	0.541479	0.142158	0.225194
0.96	0.852271	0.128758	0.531592	0.147729	0.231206
0.95	0.848050	0.142118	0.513992	0.151950	0.234558
0.9	0.810569	0.217602	0.462680	0.189431	0.268807
0.85	0.567955	0.428858	0.499122	0.432045	0.463167
0.8	0.281108	0.588009	0.547371	0.718892	0.621515
0.7	0.123080	0.634937	0.577368	0.876920	0.696293
0.6	0.063650	0.716266	0.563904	0.936350	0.703896
0.5	0.036299	0.788911	0.547163	0.963701	0.698013

Reverse Light On Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	–	0.0	–
0.98	0.264654	0.795618	0.442858	0.735346	0.552797
0.97	0.246255	0.854448	0.431387	0.753745	0.548725
0.96	0.221488	0.884775	0.430762	0.778512	0.554636
0.95	0.194127	0.903844	0.434007	0.805873	0.564175
0.9	0.070291	0.988336	0.447212	0.929709	0.603922
≤ 0.85	0.0	1.0	0.462373	1.0	0.632360

Table A.3: The table shows the intrusion detection performance of X-CANIDS on ROAD masquerade attacks (not changing timing) respecting the sensitivity parameter q with $0.5 \leq q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. For the correlated signal attack, the performance metrics do not change for $q \leq 1$ as it produces only positive classifications. The max engine coolant temperature attack and the max speedometer attack are detected well in contrast to the reverse light attacks.

Suspension Attack 0.01s					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	0.216015	0.822418	0.563211	0.783985	0.655508
0.97	0.141527	0.877834	0.569487	0.858473	0.684738
0.96	0.097300	0.931990	0.567125	0.902700	0.696605
0.95	0.068436	0.953401	0.569275	0.931564	0.706693
≤ 0.9	0.0	1.0	0.574946	1.0	0.730116
Suspension Attack 0.1s					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	0.177501	0.832802	0.577533	0.822499	0.678585
0.97	0.117105	0.880664	0.581184	0.882895	0.700952
0.96	0.074689	0.931078	0.579054	0.925311	0.712334
0.95	0.054403	0.952776	0.578725	0.945597	0.718012
≤ 0.9	0.0	1.0	0.580567	1.0	0.734632
Suspension Attack 1s					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	0.200923	0.830252	0.512660	0.799077	0.624599
0.97	0.146079	0.891317	0.511514	0.853921	0.639785
0.96	0.101999	0.939496	0.510936	0.898001	0.651301
0.95	0.074833	0.958543	0.513367	0.925167	0.660326
≤ 0.9	0.0	1.0	0.522216	1.0	0.686126

Table A.4: The table shows the intrusion detection performance of X-CANIDS on ROAD suspension attacks respecting the sensitivity parameter $q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. For each suspension attack, a signal stream with a different base period was attacked (0.01s, 0.1s, 1s). The performance is worse compared to the max speedometer attack and max engine coolant temperature attack. The separation of benign and attack samples is harder for the model in the case of suspension attacks than for masquerade attacks.

Correlated Signal Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.0	0.0	1.0	1.0	1.0
0.999	0.0	0.759351	0.839809	1.0	0.912931
0.99	0.0	0.932251	0.810256	1.0	0.895184
0.98	0.0	0.955540	0.806433	1.0	0.892846
0.97	0.0	0.966126	0.804708	1.0	0.891787
0.96	0.0	0.977417	0.802875	1.0	0.890661
0.95	0.0	0.985180	0.801620	1.0	0.889888
≤ 0.9	0.0	1.0	0.799235	1.0	0.888416

Max Engine Coolant Temperature Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.538062	0.0	1.0	0.461938	0.631953
0.999	0.025952	0.0	1.0	0.974048	0.986854
0.99	0.006920	0.308740	0.513417	0.993080	0.676887
0.98	0.005190	0.382520	0.460368	0.994810	0.629447
0.97	0.0	0.534620	0.380263	1.0	0.551001
0.96	0.0	0.602724	0.352439	1.0	0.521190
0.95	0.0	0.636776	0.34	1.0	0.507463
0.9	0.0	0.874574	0.272770	1.0	0.428624
0.85	0.0	0.967650	0.253176	1.0	0.404055

Max Speedometer Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.999	0.0	0.0	1.0	1.0	1.0
0.99	0.0	0.024196	0.979429	1.0	0.989608
0.98	0.0	0.025904	0.978009	1.0	0.988882
0.97	0.0	0.026948	0.977143	1.0	0.988439
0.96	0.0	0.028750	0.975651	1.0	0.987675
0.95	0.0	0.029415	0.975102	1.0	0.987394
0.9	0.0	0.241484	0.826706	1.0	0.905133
0.85	0.0	0.547965	0.677662	1.0	0.807865

Reverse Light Off Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.999	1.0	0.0	-	0.0	-
0.99	0.485825	0.012698	0.975664	0.514175	0.673445
0.98	0.419001	0.030409	0.949793	0.580999	0.720972
0.97	0.336652	0.056307	0.921040	0.663348	0.771238
0.96	0.270840	0.074185	0.906821	0.729160	0.808343
0.95	0.233041	0.084879	0.899466	0.766959	0.827944
0.9	0.033243	0.326483	0.745672	0.966757	0.841943
0.85	0.021768	0.427402	0.693836	0.978232	0.811848

Reverse Light On Attack Masquerade					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.999	1.0	-	-	0.0	-
0.99	0.133475	0.012629	0.987371	0.866525	0.923009
0.98	0.116614	0.016411	0.983589	0.883386	0.930799
0.97	0.103408	0.017952	0.982048	0.896592	0.937377
0.96	0.096097	0.019191	0.980809	0.903903	0.940787
0.95	0.092442	0.019990	0.980010	0.907558	0.942394
0.9	0.051645	0.083941	0.916059	0.948355	0.931927
0.85	0.035019	0.198668	0.801332	0.964981	0.875575

Table A.5: The table shows the intrusion detection performance of X-CANIDS on ROAD byte-based masquerade attacks (not changing timing) respecting the sensitivity parameter q with $0.85 \leq q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. The general reconstruction works better in the case of byte-based intrusion detection compared to signal-translated intrusion detection. The correlated signal attack produces perfect results for $q = 1$. The reverse light attacks also produce better results than before with a higher recall and precision.

Correlated Signal Attack Fabrication					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.0	0.0	1.0	1.0	1.0
0.999	0.0	0.759520	0.839661	1.0	0.912843
0.99	0.0	0.932299	0.810112	1.0	0.895096
0.98	0.0	0.955571	0.806290	1.0	0.892758
0.97	0.0	0.966150	0.804565	1.0	0.891700
0.96	0.0	0.977433	0.802733	1.0	0.890573
0.95	0.0	0.985190	0.801478	1.0	0.889800
≤ 0.9	0.0	1.0	0.799093	1.0	0.888329

Max Engine Coolant Temperature Attack Fabrication					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
1	0.051903	0.0	1.0	0.948097	0.973357
0.999	0.025952	0.0	1.0	0.974048	0.986854
0.99	0.006920	0.308740	0.513417	0.993080	0.676887
0.98	0.005190	0.382520	0.460368	0.994810	0.629447
0.97	0.0	0.534620	0.380263	1.0	0.551001
0.96	0.0	0.602724	0.352439	1.0	0.521190
0.95	0.0	0.636776	0.34	1.0	0.507463
0.9	0.0	0.874574	0.272770	1.0	0.428624
0.85	0.0	0.967650	0.253176	1.0	0.404055

Max Speedometer Attack Fabrication					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.999	0.0	0.0	1.0	1.0	1.0
0.99	0.0	0.024196	0.979429	1.0	0.989608
0.98	0.0	0.025904	0.978009	1.0	0.988882
0.97	0.0	0.026948	0.977143	1.0	0.988439
0.96	0.0	0.028750	0.975651	1.0	0.987675
0.95	0.0	0.029415	0.975102	1.0	0.987394
0.9	0.0	0.241484	0.826706	1.0	0.905133
0.85	0.0	0.547965	0.677662	1.0	0.807865

Reverse Light Off Attack Fabrication					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.999	1.0	0.0	-	0.0	-
0.99	0.817813	0.012679	0.934142	0.182187	0.304907
0.98	0.767112	0.030364	0.883333	0.232888	0.368597
0.97	0.695454	0.056223	0.842450	0.304546	0.447368
0.96	0.637147	0.074074	0.828638	0.362853	0.504701
0.95	0.598107	0.084751	0.823978	0.401893	0.540270
0.9	0.062025	0.325993	0.739606	0.937975	0.827062
0.85	0.035660	0.428262	0.689714	0.964340	0.804228

Reverse Light On Attack Fabrication					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.999	1.0	0.0	-	0.0	-
0.99	0.400472	0.009536	0.981846	0.599528	0.744472
0.98	0.387500	0.012681	0.976499	0.612500	0.752808
0.97	0.378302	0.014102	0.974312	0.621698	0.759053
0.96	0.372170	0.015218	0.972598	0.627830	0.763079
0.95	0.367571	0.015928	0.971558	0.632429	0.766143
0.9	0.292807	0.074769	0.890555	0.707193	0.788353
0.85	0.260259	0.205844	0.755510	0.739741	0.747542

Table A.6: The table shows the intrusion detection performance of X-CANIDS on ROAD byte-based fabrication attacks (changing timing) respecting the sensitivity parameter q with $0.85 \leq q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. The fabrication attacks produce comparable results to the masquerade attacks in the case of the correlated signal attack, max engine coolant temperature attack, and max speedometer attack. The performance regarding the reverse light attacks is worse in terms of precision and recall.

Suspension Attack 0.01s					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.95	1.0	0.0	-	0.0	-
0.9	0.997208	0.114779	0.037736	0.002792	0.005199
0.85	0.860400	0.733683	0.234742	0.139600	0.175080
0.8	0.302932	0.979745	0.534237	0.697068	0.604886
≤ 0.7	0.0	1.0	0.617174	1.0	0.763275
Suspension Attack 0.1s					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.95	1.0	0.0	-	0.0	-
0.9	0.769195	0.114779	0.764253	0.230805	0.354539
0.85	0.508609	0.733683	0.519174	0.491391	0.504901
0.8	0.276408	0.979745	0.543516	0.723592	0.620758
0.7	0.010237	1.0	0.614740	0.989763	0.758424
≤ 0.6	0.0	1.0	0.617174	1.0	0.763275
Suspension Attack 1s					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.95	1.0	0.0	-	0.0	-
0.9	0.796184	0.156039	0.678019	0.203816	0.313417
0.85	0.527687	0.772693	0.496333	0.472313	0.484025
0.8	0.266636	0.997749	0.542326	0.733364	0.623541
0.7	0.008376	1.0	0.615185	0.991624	0.759309
≤ 0.6	0.0	1.0	0.617174	1.0	0.763275

Table A.7: The table shows the intrusion detection performance of X-CANIDS on ROAD byte-based suspension respecting the sensitivity parameter $q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. The overall performance of detecting suspension attacks is worse than for the signal-translated version of X-CANIDS.

Masquerade attack 1					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.999	0.008112	0.0	1.0	0.991888	0.995927
0.99	0.007375	0.0	1.0	0.992625	0.996299
0.98	0.005900	0.006459	0.986823	0.994100	0.990448
0.97	0.004425	0.174740	0.734894	0.995575	0.845600
0.96	0.004425	0.244708	0.664370	0.995575	0.796930
0.95	0.004425	0.282741	0.631431	0.995575	0.772753
0.9	0.004425	0.391819	0.552826	0.995575	0.710900
0.85	0.002950	0.486186	0.499446	0.997050	0.665518
0.8	0.0	0.995335	0.328329	1.0	0.494349
≤ 0.7	0.0	1.0	0.327299	1.0	0.493181
Masquerade attack 2					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	0.0	0.0	1.0	1.0	1.0
0.98	0.0	0.006365	0.986497	1.0	0.993202
0.97	0.0	0.172207	0.729745	1.0	0.843760
0.96	0.0	0.241160	0.658488	1.0	0.794082
0.95	0.0	0.278642	0.625297	1.0	0.769456
0.9	0.0	0.386139	0.546323	1.0	0.706609
0.85	0.0	0.479137	0.492509	1.0	0.659975
0.8	0.0	0.995403	0.318402	1.0	0.483012
≤ 0.7	0.0	1.0	0.317403	1.0	0.481861
Masquerade attack 3					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	0.0	0.0	1.0	1.0	1.0
0.98	0.0	0.006459	0.986900	1.0	0.993407
0.97	0.0	0.174740	0.735757	1.0	0.847765
0.96	0.0	0.244708	0.665358	1.0	0.799057
0.95	0.0	0.282741	0.632463	1.0	0.774857
0.9	0.0	0.391819	0.553922	1.0	0.712934
0.85	0.0	0.486186	0.500184	1.0	0.666831
0.8	0.0	0.995335	0.328329	1.0	0.494349
≤ 0.7	0.0	1.0	0.327299	1.0	0.493181

Table A.8: The table shows the intrusion detection performance of X-MVBIDS on MVB masquerade attacks respecting the sensitivity parameter $q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. All masquerade attacks were detected with high recall and precision for a suitable q.

Fabrication attack 1					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	1.0	0.006311	0.0	0.0	-
0.97	1.0	0.170757	0.0	0.0	-
0.96	1.0	0.239130	0.0	0.0	-
0.95	1.0	0.276297	0.0	0.0	-
0.9	1.0	0.382889	0.0	0.0	-
0.85	1.0	0.475105	0.0	0.0	-
0.8	0.021689	0.995442	0.307899	0.978311	0.468385
≤ 0.7	0.0	1.0	0.311610	1.0	0.475156
Fabrication attack 2					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	1.0	0.006347	0.0	0.0	-
0.97	1.0	0.171721	0.0	0.0	-
0.96	1.0	0.240480	0.0	0.0	-
0.95	1.0	0.277856	0.0	0.0	-
0.9	1.0	0.385049	0.0	0.0	-
0.85	1.0	0.477786	0.0	0.0	-
0.8	0.021423	0.995416	0.311799	0.978577	0.472916
≤ 0.7	0.0	1.0	0.315472	1.0	0.479633
Fabrication attack 3					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	1.0	0.006311	0.0	0.0	-
0.97	1.0	0.170757	0.0	0.0	-
0.96	1.0	0.239130	0.0	0.0	-
0.95	1.0	0.276297	0.0	0.0	-
0.9	1.0	0.382889	0.0	0.0	-
0.85	1.0	0.475105	0.0	0.0	-
0.8	0.021689	0.995442	0.307899	0.978311	0.468385
≤ 0.7	0.0	1.0	0.311610	1.0	0.475156

Table A.9: The table shows the intrusion detection performance of X-MVBIDS on MVB fabrication attacks respecting the sensitivity parameter $q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. For $q \geq 0.85$, no intrusions are detected. For $0.85 \leq q \leq 0.98$, only false positives are produced. For $q \geq 0.99$, only negative classifications are produced.

Suspension attack 1					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	1.0	0.006365	0.0	0.0	-
0.97	1.0	0.172207	0.0	0.0	-
0.96	1.0	0.241160	0.0	0.0	-
0.95	1.0	0.278642	0.0	0.0	-
0.9	1.0	0.386139	0.0	0.0	-
0.85	1.0	0.479137	0.0	0.0	-
0.8	0.022053	0.995403	0.313582	0.977947	0.474889
≤ 0.7	0.0	1.0	0.317403	1.0	0.481861
Suspension attack 2					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	1.0	0.006365	0.0	0.0	-
0.97	1.0	0.172207	0.0	0.0	-
0.96	1.0	0.241160	0.0	0.0	-
0.95	1.0	0.278642	0.0	0.0	-
0.9	1.0	0.386139	0.0	0.0	-
0.85	1.0	0.479137	0.0	0.0	-
0.8	0.013688	0.995403	0.315418	0.986312	0.477980
≤ 0.7	0.0	1.0	0.317403	1.0	0.481861
Suspension attack 3					
q	**FNR**	**FPR**	**Precision**	**Recall**	**F1**
≥ 0.99	1.0	0.0	-	0.0	-
0.98	1.0	0.006459	0.0	0.0	-
0.97	1.0	0.174740	0.0	0.0	-
0.96	1.0	0.244708	0.0	0.0	-
0.95	1.0	0.282741	0.0	0.0	-
0.9	1.0	0.391819	0.0	0.0	-
0.85	1.0	0.486186	0.0	0.0	-
0.8	0.020649	0.995335	0.323745	0.979351	0.486625
≤ 0.7	0.0	1.0	0.327299	1.0	0.493181

Table A.10: The table shows the intrusion detection performance of X-MVBIDS on MVB suspension attacks respecting the sensitivity parameter $q \leq 1$. The table shows discrete choices of q with the aim of presenting high F1 scores. For $q \geq 0.85$, no intrusions are detected. For $0.85 \leq q \leq 0.98$, only false positives are produced. For $q \geq 0.99$, only negative classifications are produced.