Context-aware Intrusion Detection in Automotive Control Systems

Armin R. Wasicek
University of California, Berkeley

Mert D. Pesé
University of Michigan

André Weimerskirch
Lear Corporation

Yelizaveta Burakova
University of Michigan

Karan Singh
University of Michigan

Abstract

This paper describes a method and framework to detect manipulations in automotive control systems. As the automotive industry is shifting towards employing software-based solutions, the incentives for attackers to manipulate automotive systems increase. At the boundary where the cyber-physical world interface is particularly sensitive for security and safety, manipulations in the computer system might have an uncontrollable impact in the physical environment and could lead to potentially dangerous situations. This paper presents a context-aware intrusion detection system (CAID) framework capable to recognize manipulations of the physical system using cyber means. CAID uses sensor information to establish reference models of the physical system and then checks correctness of current sensor data against the reference models. Thereby, it establishes the notion of plausibility of a controller's operation. CAID augments today's cyber physical intrusion detection systems (IDS) by adding a physical model to the detection engine. The CAID framework has been evaluated in a vehicular setup using a test vehicle. Telemetry data has been collected from a test vehicle that has then been chip-tuned with a commercially available chip-tuning tool to obtain manipulated data. CAID was able to recognize the chip tuning with a very high probability using an unsupervised Artificial Neural Network (ANN). This proof-of-concept could be the starting point to enhance current automotive IDS using the CAID framework in order to detect future automotive attacks to safety-critical systems.

1 Introduction

The automotive industry is on the verge of transitioning from a mechanical to a software industry [6]. Initially, mechanical subsystems were augmented and replaced with electrical components to increase reliability and to improve active safety systems. Infotainment and connected services were added to provide comfort features to drivers and passengers. The coming wave of new vehicle technologies includes advanced safety functions, automation features, and interconnection with road and traffic infrastructures to optimize traffic flow and to enable cooperation between vehicles. At the same time, technologies for vehicle customization and personalization are gaining momentum to provide an individualized driving experience. Software is a major innovation factor behind these new developments emerging in the automotive space [2].

This paradigm change in the automotive industry might entail a change in current and future vehicle use models. Traditional car ownership and maintenance are expected to persist, but particularly in urban environments, new forms of mobility start to emerge and will eventually prevail. The focus will shift from a vehicle-centric ownership model towards a mobility-on-demand and mobility-as-a-service model. Single vehicles might be shared among many users. Vehicles will allow increasing levels of customization for each user. In order to adapt to a new user or a new service session, vehicles must be able to quickly reconfigure their configurations. This does not only relate to comfort features like preset radio stations, setting individual dashboard layouts, or adjusting seat position, but might extend to cyber-physical properties that determine the vehicle dynamics. An engine controller, for instance, could be reconfigured to optimize power and maximize acceleration. This kind of flexibility requires vehicle software to activate or deactivate functions, and/or to configure parameters for control loops.

The flexibility of software is a strength and weakness at the same time. Engineers can model processes and procedures in computer programs in a quick and cost-efficient manner. Changing software configurations is one of the main reasons behind software's success story. Updating software and distributing program code to ve-

hicles can be executed quickly at little cost, assuming that a communication infrastructure is available. The ease for updating software does not only allow for well intended change, but this approach is vulnerable to unintentional and malicious manipulation. The flexibility of software can potentially be unintentionally or intentionally lead to vehicle damage and it might even endanger passengers' safety. Safety and security measures are put in place to limit the possible executions of a computer program to a range of safe states. In systems that interact with the physical environment safety and security are crucial, because manipulating their behavior might have a negative impact on physical systems. Security mechanisms of automotive computer systems should therefore prioritize mitigation against cyber-physical attacks and limit the impact of such attacks in case of a security breach.

This paper presents the Context-aware Intrusion Detection (CAID) framework. CAID is specifically targeted at protecting automotive control systems. It taps into various control systems of the vehicle and monitors the state of the controllers by checking their input/output behavior. By capturing benign execution traces in a reference model [30], CAID is capable of recognizing malign executions. CAID is generic and makes no assumptions on the implementation of a controller. It is adaptive and leverages a learning system to integrate new information. CAID is non-intrusive and monitors on-board vehicle communication passively. It can easily be integrated in automotive firewalls/filters, e.g. Controller Area Network (CAN) gateways to discard detected malicious on-board communication messages and prevent attacks, or to relay detected anomalies to remote forensics services that analyze the detected anomalies in a central server. While there are already automotive Intrusion Detection Systems (IDSs) available today, none of them leverages physical models in the way CAID does. We expect that CAID will be part of all future automotive IDSs in order to stay competitive with advanced attacker strategies.

The paper is organized as follows: Section 2 defines the scope of an automotive intrusion detection system and potential threats to automotive control systems. Section 3 contains a technical description of the CAID framework. The case study in Section 4 uses chip-tuning as an attack vector and elaborates how CAID recognizes this attack. Section 6 discusses the results and limitations, and Section 7 concludes this work.

2 Automotive Systems and Threats

Several research teams have demonstrated impressively during the last years that it is possible to compromise automotive systems [15, 16, 13, 7, 10]. Attacks using each available interface have been shown, such as the On-board Diagnostics II (OBD-II) port, USB, CD/DVD, Bluetooth, and cellular connections. It is widely agreed that a defense-in-depth approach is required to protect vehicles. While hardening all interfaces is a good starting point, further security mechanisms are required in a vehicle. A major aspect is a secure software development process. This is out of scope of this paper, and the interested reader is referred to SAE J3061 [20]. There is a wide body available of further research about automotive threats and countermeasures, e.g. [24] and [28].

2.1 Automotive Process Control

The automotive context includes all processes related to a vehicle's main objective of moving people and goods in a safe, secure, and efficient manner. Many of these processes are part of automatic control systems, i.e., systems that manage, command, direct, or regulate the behavior of processes according to a *control law*. Control systems are the heart and soul of almost all major automotive subsystems such as engine, power-train, battery management, electrical system, steering, etc. They provide the intelligence to take timely decisions and actions in the vehicle. Table 1 summarizes some of these systems.

Control systems and processes collaborate to achieve the overall control objective. For that purpose, control loops can be layered or nested. A supervisory control mechanism might provide input to several low level controllers. For example, in a Hybrid Electrical Vehicle (HEV) architecture, a hybrid controller regulates the power that flows from electric motor and combustion engine to the transmission. Each motor has its own controller that deals with the propulsion specific variables.

The rate of change of a process is another important aspect for a controller. Certain process variables in an engine will change faster than others. This translates into different execution times for a control cycle. For instance, the spark advance control governs the firing of the spark plugs in a combustion engine. Timing is critical as it has a major impact on efficiency of the combustion, its emissions, and ultimately on driveability. A sensor will report the position of the piston to the electronic control unit (Electronic Control Unit (ECU)) that computes the optimal timing for the spark event. Depending on RPM and type of engine, the spark plugs are activated every 10 to 16 ms. The fuel injector regulates the air/fuel mixture and requires feedback about the composition of the exhaust fumes. This control process is slower than the spark advance control.

Nowadays, a control law will be implemented in a digital process controller. Typically, control engineers devise the control algorithm and its parameters in a model-based design environment. The parameters are usually

Table 1: Examples of automotive, closed-loop control systems

Control System	Indirectly controlled variable	Directly controlled variable	Manipulated variable	Sensor	Actuator
Fuel injection system	Air-fuel ratio	Exhaust oxygen content	Quality of injection fuel	Zirconia or Titania based electrochemical	Fuel injector
Knock control	Knock	Knock sensor output	Ignition timing	Piezo-electric accelerometer	Ignition coil switch. Transistor
Anti-lock braking system	Wheelslip limit	Wheelspeed	Brake time pressure	Magnetic reluctance	ABS solenoid valve
Spark advance control	Efficiency, driveability, emissions		Ingition timing		Spark plug

kept in a large number of look-up tables. The algorithm is then split in functions and tasks that form the logical system view. In the next step, the functions are mapped to the vehicle's electrical and electronics (E/E) architecture consisting of networked micro-controllers in an ECUs. A typical automotive E/E architecture will use a serial bus, e.g., Controller Area Network (CAN) or FlexRay as a network. Finally, binary images encompassing code and parameters are created and flashed onto the ECUs. This work assumes that the automotive process control system is the point of attack.

Securing control systems against attacks is an active research area (e.g., [4]). This work focuses on recognition of attacks after the fact.

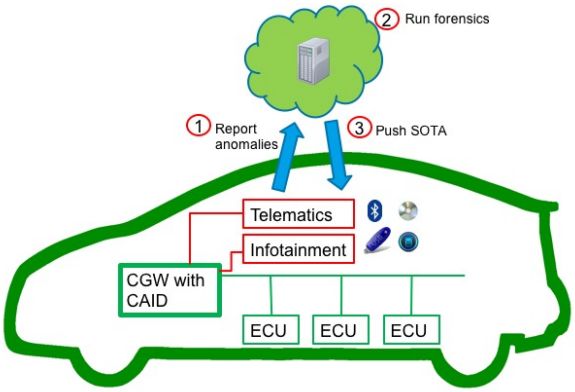


Figure 1: Central gateway deploying CAID Framework

2.2 Secure Vehicle Architecture

An abstract modern vehicle electronics architecture is displayed in Figure 1. Current efforts to secure vehicles are ECU-based security mechanisms such as secure boot, secure on-board communication protocols, access control, and IDS.

An essential aspect of a secure vehicle is the separation of network segments, connected by gateways. Ideally, there is a Central Gateway (CGW) that separates the infotainment and telematics unit, the OBD-II port, and further wireless interfaces, from all safety-critical systems like the power-train bus. The CGW is then able to route and filter all on-board messages using a routing table or a white-list. The white-list defines the messages that are allowed to pass from a defined origin port to a defined destination port. For instance, a white-list implemented in a CGW could define that a message to display information about the current radio station in the vehicle's dash board is allowed to pass, but a diagnostics message to manipulate the brakes is not allowed to pass from the infotainment network segment to another segment. Experts believe that such a CGW filter would have avoided that any of the attacks reported in the last years would have advanced beyond the infotainment and telematics systems, and none of those successful remote attacks would have had any impact to the vehicle's physical driving behavior.

2.3 IDS and Filter

IDS are deployed in vehicles to monitor the in-vehicle communication bus and to detect anomalous on-board communication messages. While there is no clear terminology in the automotive domain, we define IDS as a system that monitors the on-board communication to detect anomalous messages using heuristic methods (e.g. see [3]). We define a filter as a module that is located between two networks, that analyzes on-board messages, and forwards or discards them based on a white-list. CGWs usually implement a filter. Furthermore, it is possible to introduce filters between exposed interfaces and the on-board communication bus, e.g. between Tire

Pressure Monitoring System (TPMS) and CAN. Note that implementations usually use combinations and definitions slightly overlap, e.g., white-list based CGWs use basic heuristics to measure typical message frequency, and heuristics-based IDS use white-lists to discard obviously malicious messages in order to reduce the amount of computations. It is a common approach to integrate IDS in a CGW to utilize both white-list and heuristics in the decision process of relaying or discarding messages. Note that it is also possible to corrupt a CAN message while it is transmitted, e.g. by corrupting the check-sum in real-time, for instance, as described in [26]. Filters effectively raise the threshold for attacks to propagate to safety-critical systems.

Today a wide variety of automotive IDS are available and research has been performed in the last decade [5], [12], [17], [30]. Several commercial products are available, and those products use combinations of machine learning, white-listing, and heuristics.

2.4 Close the circle: IDS, Forensics, SOTA

A proper future vision to prevent attacks is to use filters and IDS to detect on-going attacks (see Figure 1). While the filter prevents that an on-going attack can advance to a critical system, the IDS detects the attacks to the non-critical system (e.g. on an infotainment system). The IDS then reports the attack to a central service such as a cloud, e.g., via telematics or during the next maintenance service. The central service runs forensics on the data reported by several vehicles and the reports are leveraged to fix software vulnerabilities. A Software over-the-Air (SOTA) mechanism is then used to update software in vehicles and remove vulnerabilities. This cycle will be used to gradually improve cybersecurity of vehicles in the field.

This paper aims to extend the detection of attacks to the critical system domain.

2.5 Cyber-Physical Attacks

Automotive systems are Cyber-Physical System (CPS). CPS are integrations of computation, networking, and physical processes. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa [14]. Cyber-physical attacks target this integration, thus, attacks in cyberspace impact the physical environment and changes of the physical parameters impact the embedded system.

Common CPS defense mechanisms focus on cyber properties for example by protecting the network communication against replay or injection attacks. Whereas countermeasures against this kind of attacks are currently being realized, not all attack vectors are covered. Attackers who exploit the semantics of messages can remain undetected, because only cyber properties are checked. The integrity of the communication pattern does not change, but the actions that follow from the communication. For instance, in a control system, an unintended control action might result from a deliberately tampered sensor input.

As a result, a future IDS for CPS also has to involve the physical part of a system by, for example, analyzing and interpreting the payload of messages. Current approaches in the automotive domain only check if the CAN payload is within the allowed range for the physical quantity. It is, however, not sufficient to only analyze the range of values in the message payload to detect attackers deliberately manipulating physical variables.

Restricting physical variables to a certain range does not prevent attacks since the current value of a physical variable depends on many factors, including context and correlation. Preceding and succeeding messages are decisive to monitor the change of a physical variable and therefore to analyze the semantics correctly. Working towards a a fully semantic-based analysis, the proposed solution overcomes this problem by using a behavior-based approach. The idea is to compare the actual behavior to a reference model that is unique for every control system. This approach extends from syntax to semantics and thus enables reliable misbehavior detection in the physical domain.

2.6 Automotive IDS

Today's automotive IDS are mainly leveraging cyber parameters of the on-board communication bus messages, such as CAN message frequencies and entropy. They combine data from several CAN lines, check payload content to a certain degree (e.g. check for range), and work fairly reliable. However, IDS are based on statistical methods which means that there will always be false alarms and also missed attacks.

This work refines current IDS approaches by integrating a physical model of the vehicle to establish a context-awareness in the IDS. The proposed approach is driver-independent and only focuses on physical parameters which can be obtained by the existing sensors in the vehicle. It can be used to increase the precision of current automotive IDS and to detect software modifications that have a physical impact. A proof-of-concept is made with respect to chip tuning, a modification of the engine control unit that has a physical impact. The approach can be refined to establish a physical model of other vehicle aspects, such as manipulation of brakes or manipulation of emissions control, hence allowing a comprehensive detection of any software manipulation that leads to a

Figure 2: Possible configuration for a recognition unit using the IDS Framework

change of the vehicle's behavior and of the control mechanism.

First similar approaches have been made in [27] and [30]. This work extends these approaches from simulation to implementation and operation in a real vehicle. It is the first research to leverage machine learning to devise physical models that are capable to detect attacks in a vehicle's control system in an efficient and comprehensive manner. As a result, this work introduces a framework for cyber-physical IDSs called CAID.

3 Context-aware Intrusion Detection (CAID)

CAID is designed to perform intrusion detection in an automotive environment. Its innovation is to detect deviations of the physical processes from the nominal behavior in the automotive system. This is achieved by monitoring streams of sensor and actuator values flowing between the ECUss. CAID is context-aware and leverages physical indicators such as speed, acceleration, and torque. Conventional computer network IDSs monitor indicators such as CPU load, memory consumption and network activity. CAID complements traditional IDSs to leverage the features and satisfy the requirements of a cyber-physical system.

3.1 General Overview

The CAID framework consists of three general modules that are integrated to pursue one or several detection goals. They communicate with each other through events.

- *Monitors* read and aggregate information. A Monitor reads raw automotive on-board communication data streams, extracts and processes features, and produces events. Events are deviations, anomalies, or outliers being detected. A Monitor functions similar to a filter, but does not discard messages.
- *Detectors* perform analysis and determine if an anomaly occurred. The Detector fuses event information from Monitors with high-level operating conditions that impact the interpretation of events. For instance, the car's current gear position might impact the interpretation of an event. Each single Detector has a well-defined detection goal.
- *Reporters* interface with the user. These modules communicate or store the result of the recognition process. 'User' is defined broadly and can be any party involved in the vehicle's life cycle such as car owner, manufacturer, maintenance engineer or a processing system.

Figure 2 depicts the relationship between instances of Monitors, Detectors, and Reporters. All three modules are related in N-to-M relations. A Recognition Unit (RU) refers to an instance of the CAID and it is a configuration of the three module types to target a single recognition goal. For instance, several monitors can connect to one detector which is in turn connected to one or more reporters. Thus, a Detector is at the core of an RU. Monitors, Detectors, and Reporters can be used multiple times in different RUs. For example, a Monitor reporting anomalies of the engine control unit could be additionally used in RUs to recognize other attacks, such as recognizing chip-tuning as well as power-boxing attacks. The three modules can be implemented in a single unit or in a distributed fashion.

An RU must be confined inside the same security perimeter. A security perimeter provides a Trusted Computing Base (TCB) and segregates a secure partition in the system. There is a trade-off when implementing an RU in a single unit or as a distributed system. If the three modules' instances are hosted in a single unit, the perimeter can be enabled by the physical boundaries of the unit or by a virtualization thereof. This design provides an easy way of establishing the security perimeter, however, at the cost of availability since this design introduces a single point of failure. Distributing the instances increases availability, but setting up a security perimeter is more challenging. Current efforts aim to secure the on-board communication in automotive networks to provide authenticity guarantees on the messages. Having authentic information is paramount for the RU. A good example is an IDS that is integrated in a vehicle gateway. Even if the Detector is able to successfully detect an attack, it might not be able to report that attack to server since the Reporter is embedded in the infotainment/telematics unit that has been compromised in the first place.

This general system model is capable to describe most currently used IDSs. Partitioning an IDS in this manner contributes to facilitate the integration of the CAID with a general intrusion detection strategy and into a larger automotive software stack like, for instance Automotive Open System Architecture (AUTOSAR) [21].

3.2 Data Collection

Monitors read input data of system information such as CPU load, temperature or resource consumption provided by the underlying OS in a traditional IDS. CAID

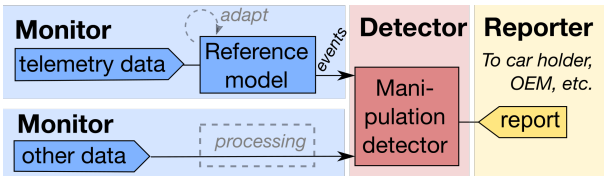


Figure 3: Recognition Unit (RU) data flow example

leverages remote sensor (telemetry) data to build a reference model, as depicted in Figure 3. This data is collected from a set of ECUs of the vehicle. These devices are usually interconnected with each other by an on-board communication bus, or In-vehicle Network (IVN). Although E/E architectures differ between cars, an IVN usually consists of multiple bus systems using different protocols and physical layers which all converge in a node. This node is called CGW and can be regarded as a router between the different bus segments. Bus systems are segmented with ECUs according to the ECU's purpose, mostly due to timing constraints. The most common bus found in vehicles is CAN. For instance, ECUs for engine, transmission and braking functions are located on the power-train CAN. Other commonly used CAN bus lines are the Comfort CAN, Body CAN, and Infotainment CAN.

We set up our monitors of the CAID to collect data from the on-board diagnostics (OBD-II) interface. This connector is mandatory for all vehicles sold in the US since 1996, and it is used for emissions measurements and diagnostic features. Although it was originally designed for emission-relevant information only, it now often provides extensive engine control, chassis, body, and comfort function diagnostic information. A Monitor can be easily connected to the OBD-II port of the vehicle and information about several standardized parameter ID (PID) called OBD Parameter IDs (OBD-II PIDs) [31] can be obtained without prior knowledge of the vehicle architecture or its message format.

The procedure for obtaining information about OBD-II PIDs is similar to a publish-subscribe mechanism. Depending on the sampling frequency constraints of the interface, the monitor entity can subscribe to the relevant ECUs by sending periodical requests for its desired set of OBD-II PIDs. The respective ECUs will then respond within a certain time. The obtained data can then be processed to build a reference model or inside the detection module.

3.3 Detection Module

A detection algorithm in CAID encompasses a reference model that summarizes the typical behavior of the monitored control system. During operation, data is used to verify whether the actual behavior lies within the 'normal' behavior pattern. Thus, the detection algorithm performs a *plausibility check* using the reference model. The reference model implements the *first level of context* to interpret the incoming data. It is dedicated to capture the dynamics of the control system. The complexity of the reference model can vary greatly. It can be as simple as a look-up table [27] or as complex as a Neural Network [30]. It might integrate new data during operation or it can remain static. There are several ways to devise a reference model:

Handcraft A developer can implement a model that is specifically designed to recognize a particular manipulation.

Generate In the automotive domain, controllers are usually developed using a model-based design [25] approach. In model-based design, models are used at all stages during the design for validation and verification. Some models can be used to actually generate code that implements the controller. The very same models could be re-purposed to generate as reference models.

Learn Operational data can be used to build a variety of machine learning models to capture typical behavior. Data-driven modeling and deriving models from data has gained some attention recently [29]. There are many appropriate modeling formalism that are able to abstract the data. For our case study we leverage artificial neural networks (Artificial Neural Network (ANN)). The data-driven approach for building the reference model has also the advantage that these models are usually adaptive, thus, they can be updated during operation to integrate slight changes such as wear-out of the engine in the reference model.

The reference model will generate events if the actual behavior and the reference behavior differ, thus indicating a potential intrusion. Anomaly detection algorithms [5] are a useful class of algorithms to build adaptive reference models. Reference models, however, are not restricted to these and could also utilize other types of physical models such as Linear time-invariant (LTI) and differential equations.

3.4 Intrusion Recognition

The events generated by the reference model in the Monitors are aggregated with other relevant information in the Detector module. A Detector implements the *second level of context*. It complements the first level of context that is dedicated to the dynamics of the control system

with global information on operational states. For example, the interpretation of engine events recognized by a Monitor might be different if the car is idling or if it is driving on a highway. Moreover, a Detector can aggregate information from different sources.

A Detector can be implemented using state machines and decision trees. Decision trees are an effective way of matching a series of conditions. According to the context, a subset of the reference model can be extracted and used for the *plausibility check* in order to increase the efficiency of the Detector.

3.5 Reporting

Reporting includes the actual reporting of the event to a party, and the reaction. Both aspects are open challenges today. First, it is unclear what data about the event is actually reported. More information will enable better forensics at the cost of additional cost to transmit the data. Second, it is unclear how to react to the event. It is widely agreed today that there will be no real-time reaction/prevention but that the data will first be further analyzed by a server. It appears reasonable that the forensics process is executed by the vehicle manufacturer. The manufacturer then informs the current vehicle owner or driver about a potential manipulation, attack, or anomaly. The manufacturer will also use the results to fix security vulnerabilities and, if available, distribute the fixed software via SOTA updates.

4 Recognizing Chip Tuning

As a proof-of-concept for a CAID, an IDS implementation to recognize chip tuning has been developed. The RU for chip tuning (RUfCT) uses the IDS framework described in section 3. A test vehicle has been chip tuned using a commercially available, off-the-shelf chip tuning tool. The car's telemetry data is recorded using the OBD-II interface. A standalone ECU implements the Monitors, Detectors, and Reporters required for RUfCT. This case study demonstrates the feasibility of the CAID framework as well as its capability to detect manipulations in the physical domain.

4.1 Manipulating Engine Control

Tools to manipulate an engine's control are widely available and are relatively easy to use. The common goal of such tools is to increase horse powers or to improve fuel efficiency.

The ECU uses a formula and a number of look-up tables to determine the control actions for given operating conditions. A tuned ECUs may, for instance, supply more fuel at full throttle. It may also change the spark timing. Such manipulation can be realized in at least the following ways:

- *Chip tuning* aims to modify the control algorithm parameters in the ECU. These parameters are usually stored in a persistent memory in the ECU's micro controller as a look-up table. The attacker changes the behavior of the controller by overwriting the parameters in the look-up tables with new values. Memory access might be protected, but research shows that many of the flash protection mechanisms implemented in common micro controllers are not adequate [22].

- *Power boxing* intercepts and modifies control messages between ECU and actuator. The power box is installed on a CAN line, for example between the ECU and the fuel injector. Incoming messages are relayed or modified, effectively changing the control parameters and thereby the behavior of the control loop. Installing a power box is easy and usually as simple as disconnecting and reconnecting a few electric cables.

Manipulations of the engine controller raises a number of concerns. Besides the desired, benign effect, negative effects might occur and damage the engine. Moreover, dependencies between control systems might be disturbed and side effects could occur.

Detecting manipulation is valuable for several stakeholders in the automotive environment. The manufacturer can void the warranty in case of illegal manipulation, the insurance company can adjust the driver's insurance policy and authorities could change the registration information for the vehicle. The car owner can be informed, in case the manipulation was not made with her approval.

4.2 Building an Engine Control Reference Model

There are several ways to establish a reference model to capture the specified behavior of the engine control unit. The approach followed in this case study uses telemetry data and leverages machine learning to build a reference model (see Section 3.3 for other options).

4.2.1 Engine Telemetry Data

The set of data points determines the first level of context and it is defined by subject matter experts. The authors selected a set of relevant data points (see Table 2) for the engine control case study. This data is obtained from the vehicle via its OBD-II interface. Each data point has to

Table 2: Data points and their respective entropy in the case study

Data point	Units	Entropy
Vehicle speed	MPH	7.91
Calculated load value	percent	8.10
Engine RPM	RPM	8.13
Absolute throttle position	percent	8.17
Fuel rate	gal/hr	7.91
O2 sensor lambda wide range		8.18
Fuel/Air commanded equivalence	ratio	8.16
Absolute throttle position B	percent	8.19
Accelerator pedal position D	percent	8.22
Catalyst temperature	F	8.19

be queried individually via the chip's request/reply protocol. Using direct access to the CAN bus could increase the sampling frequency for the data points, but requires knowledge of the CAN ID numbers of the individual messages. This is proprietary information that was not available to the authors, so standard OBD-II PIDs were used.

4.2.2 Feature Extraction

The raw data collected from the telemetry unit of the vehicle is processed to generate a feature vector $X = \{x_{mean},\ x_{var},\ x_{std},\ x_{skewness},\ x_{kurtosis}\}$. The time series data is segregated in intervals using a a sliding window technique. Therefore, a window W has n elements t such that $W = \{t_i | i = 1 \ldots n\}$. In the case study an interval has the duration of $2s$ and a period of $1s$. Each interval is mapped to a feature vector consisting of five statistical moments and the first m real-valued coefficients of a power spectrum. The statistical moments used are mean (Eq. 1), variance (Eq. 2), standard deviation (Eq. 3), skewness (Eq. 4), and kurtosis (Eq. 5) as suggested in [18]. The power spectrum is computed using fast Fourier transform (Fast Fourier Transform (FFT)).

$$x_{mean} = \mu = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{1}$$

$$x_{var} = \sigma = \frac{1}{n}\sum_{i=1}^{n} (t_i - \mu)^2 \tag{2}$$

$$x_{std} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (t_i - \mu)^2} \tag{3}$$

$$x_{skewness} = \frac{\frac{1}{n}\sum_{i=1}^{n} (t_i - \mu)^3}{\sigma^3} \tag{4}$$

$$x_{kurtosis} = \frac{\frac{1}{n}\sum_{i=1}^{n} (t_i - \mu)^4}{\sigma^4} \tag{5}$$

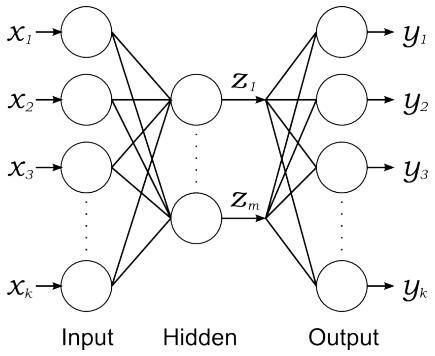


Figure 4: Architecture of a Bottleneck ANN

After generation, the features were normalized using Gaussian normalization (Eq. 6). This is not strictly required, but recommended practice for neural networks.

$$x_{norm} = \frac{x_i - \mu}{\sigma} \qquad \forall x_i \in X, i = 1..k \tag{6}$$

The case study made use of Matlab's `mean`, `var`, `std`, `skewness`, and `kurtosis` functions from the Statistics toolbox.

4.2.3 Bottleneck Artificial Neural Network (ANN)

Figure 4 depicts the general architecture of a Bottleneck ANN that is a special variant of ANNs suitable for anomaly detection [5]. In a Bottleneck ANN[1] the number k of neurons in the output layer Y is same as the dimension of input vectors X and the hidden layer Z utilizes significantly less neurons m (i.e., $k > m$). They have been successfully used for image compression [11], dimensionality reduction [19], etc. The case study employs as transfer function for each hidden layer neuron a sigmoid function and the linear function for the output neurons.

During training the same feature vector is used for input and output. The hidden layer then generalizes the *ratio between features*. Thus, the ANN stores the typical behavior of an engine. Training has been performed using Levenberg-Marquardt back-propagation by virtue of MATLAB's `trainlm` function.

During operation, the bottleneck ANN checks how well it can reconstruct an input or how similar the given input is to the ones stored in the hidden layer. This similarity is captured as a real valued anomaly score and computed as the error between input and output. The case study uses a mean square error. Note that all inputs are normalized, thus, each one contributes to the same extent to the mean.

$$score = \sum_{i=1}^{k} (x_i - predict(x_i))^2 \tag{7}$$

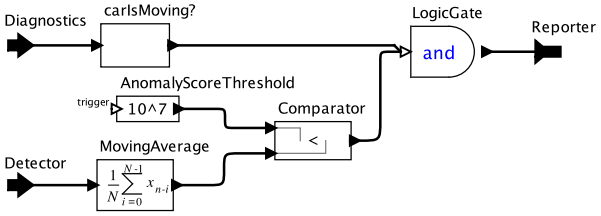


Figure 5: Exemplary manipulation detector data flow

The advantage of a learning approach is that the ANN is able to recognize any deviation from the specified behavior. This is particularly important in the intrusion detection setting, since new attacks are hard to anticipate. A drawback here is that the training data set has to completely cover the specified behavior. Certain data instances might be hard to obtain. A mitigation strategy is to aggregate data from similar vehicles to complete the training data set.

4.2.4 Manipulation detector

The task of the manipulation detector is to integrate first and second level context. It fuses the results from the ANN with environmental information and the operational modes of the vehicle. Figure 5 depicts a model for a manipulation detector. The `carIsMoving` block checks, if the car is in motion. The `MovingAverage` block filters the anomaly scores from the Detector and the `AnomalyScoreThreshold` block checks against a threshold value that has been selected to be 10^7 for the ANN with 43 hidden units. If both conditions hold, a manipulation is reported.

5 Experimental Setup

For the experiments, a 2015 passenger vehicle was used. In the first part of this setup description, the used hardware is presented. Next, the tracks for collecting the data and the selection of the OBD-II PIDs are discussed. Two datasets were collected, one to represent original behavior and one for modified behavior. During the experiments with the vehicle, an OBD-II scan tool and a proprietary off-the-shelf chip tuning device were deployed. The logging of OBD-II PIDs was conducted by plugging the OBD-II scan tool *OBDLink MX Bluetooth* to the OBD-II port of the vehicle. The OBDLink MX uses a STN1110 micro-controller [23] to serialize OBD-II communication. Once connected to an Android device or the PC via Bluetooth, it was able to transmit a maximum of 100 PIDs/second. The Android app for this tool was able to log user-defined PIDs with any sampling rate and save it as an *csv* file. The chip tuning has been realized with the *SKL Motorworks Performance Chip KL-PRO1* that sends a continuous signal to the factory Electronic Control Module (ECM) to reprogram itself for a more optimum fuel mixture and timing curve. According to the vendor, this chip tuner increases the power and torque by up to 20%.

A standard driving pattern was defined to simulate a generic usage pattern and to cover common traffic scenarios. The standardized drive includes highway and urban roads as well as driving during rush hour traffic jams. For both unmodified and modified configurations of the vehicle, three different drivers were dispatched, each with the standardized driving routine in order to eliminate driver-specific influences. It should be noted that all three drivers were driving naturally to represent the average driver.

To demonstrate a proof-of-concept, the detection approach was fully implemented on MATLAB 2015a using the statistics and neural network toolboxes. Pandas 0.17.1 has been used for preprocessing the raw data like unifying time stamps and selecting columns. To simulate a potential implementation as on an ECU, the *Renesas YRDKRX63N* Demonstration Kit using the *Micrium uC/OS-III* that is part of the Basic Software Modules (BSWs) in the AUTOSAR architecture was used. This powerful microcontroller has a CAN interface and supports Digital Signal Processor (DSP) instructions which are beneficial for the signal pre-processing done during the feature extraction. Furthermore, it has wireless connections such as WiFi and Bluetooth.

5.1 Data exploration

The feature extraction process described is Section 4.2.2 was applied to the data collected from the vehicle and results in a feature vector of 54 elements.

Figure 6 depicts the spread of the data points, after the raw data has been normalized using Gaussian normalization (Eq. 6). Each point on the X-axis relates to a data point and contains two bars: a blue one for the original version of the data and an orange bar for the modified data. The height of the bars are the maximum minus the minimum value in the entire data-set collected. The blue and orange line plots over the bars show the median values to indicate the distribution of a data point's values. The figure shows that most of the medians are close to the minimum, thus, the entities are very stable and have just slight variations. The peaks indicate that there are some outliers present. Basically, an ANN can deal with outliers to a certain degree, since the weights of the neurons can amplify and smoothen out.

The scatter-plot matrix in Figure 7 depicts the pairwise relationships between the selected data points (see 4.2.1). The figure shows a subset of the input feature vector, the

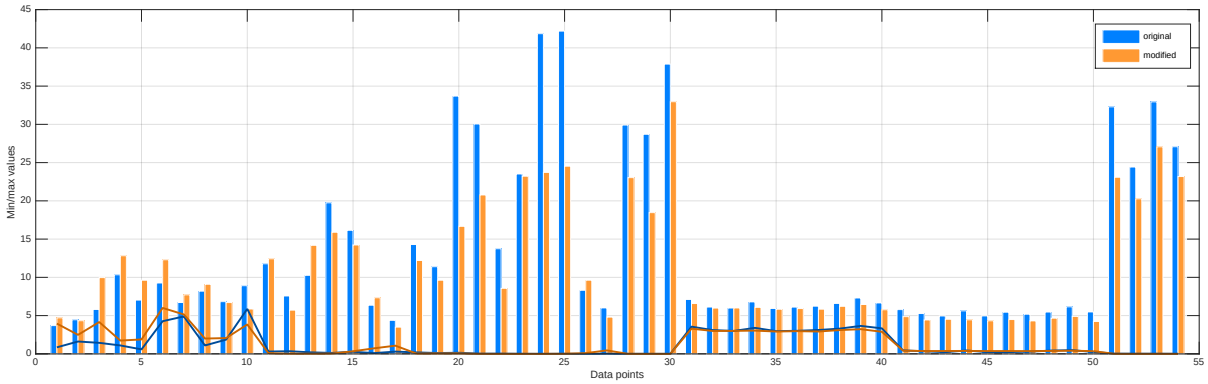


Figure 6: Variation in the feature vector

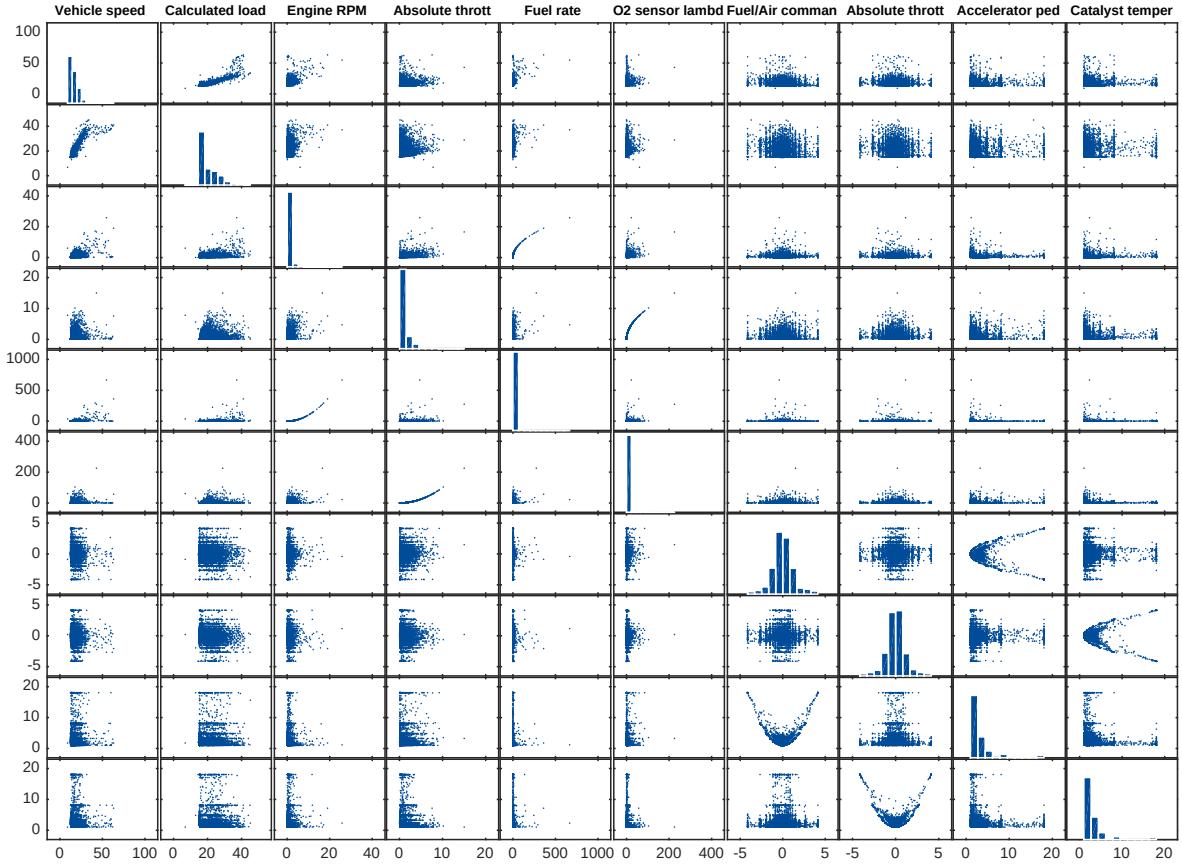


Figure 7: Pairwise relationships between data points

fourth statistical moment has been selected. For instance, it reveals that *Engine RPM* and *Fuel rate* in scatter-plot $(5,3)$ are strongly correlated. Other data points do not correlate well, e.g., *Catalyst temperature* and *Accelerator pedal position* at $(10,9)$. Correlation between data points is due to coupling in the physical environment. For instance, if the Engine RPM increase, more fuel gets burned. The catalyst subsystem works independently from the engine.

The ANN stores the relation between data points and fires an alert, if this relation does not hold for a particular instance of a feature vector. That goal requires cor-

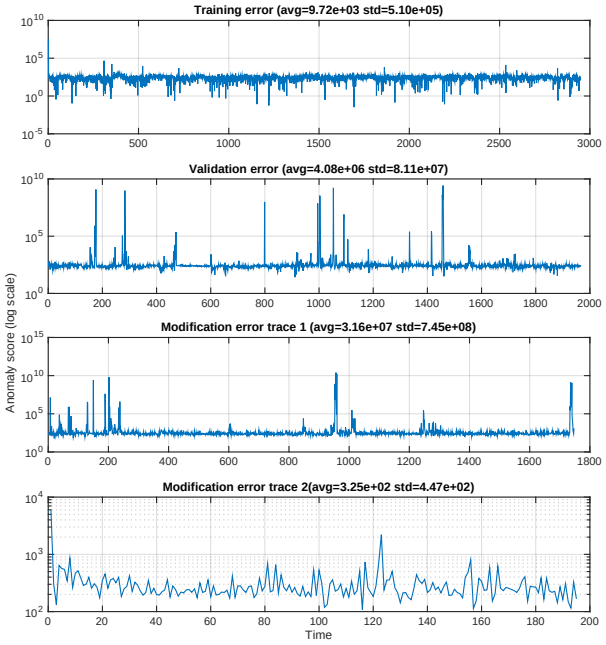


Figure 8: Resulting anomaly scores w.r.t. training set, validation set, modified set 1, and modified set 2

related as well as independent data points. Single data points can serve as indicators that a relation has been violated. Combinations of data points can signify through changes in their degree of correlation that something is odd. Taken together as inputs to a ANN they form an indicator, i.e., the ANN's anomaly score.

5.2 Results

Three different ANN architectures with 16, 32, and 43 hidden neurons were trained for comparison.

Figure 8 depicts the ANN at work: the topmost chart displays the individual anomaly scores for the series of feature vectors used for training. The average score is around $9.72e+03$. The upper middle chart shows the same for the validation data set, averaging at $4.08e+06$. The lower middle chart exemplifies that the scores of a modified vehicle (mean at $3.16e+07$) are substantially higher than those of the other two sets. Finally, the bottom chart uses feature vectors from the modified set as well, but only a small number. The mean score for this trace ($3.25e+02$) stays below the those of the other sets. This indicates two circumstances:

(a) This short period of observation did not reveal any manipulation, thus the vehicle operated in the specified state space despite the manipulation.

(b) A certain number of anomalous feature vectors has to be recognized, before forwarding an alert. How many such feature vectors should be collected is described next.

The left hand side of Figure 9 visualizes that the ANN actually can distinguish behaviors. It shows the anomaly score of the validation data set versus the score of the modified data set for the three ANN architectures. The validation data set is considered to be equal to data obtained from original operation. Each architecture produces scores in a specific band of the logscale chart. After some instances have been evaluated through the ANN, the anomaly score of the modified set remains consistently above the score of the validation set. Therefore, we conclude that the ANN is capable of distinguishing between specified and modified behaviors.

The curves were obtained by sweeping through subsets of the modified dataset:

1. Randomly select a subset of feature vector instances from the modified dataset

2. Compute the mean of the resulting anomaly scores for the subset

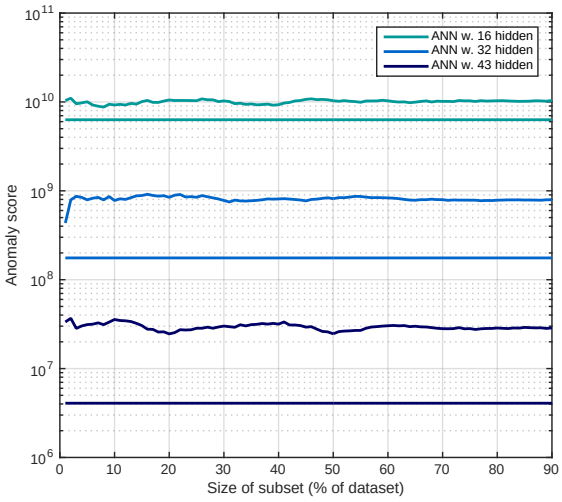
3. Increase the size of the subset, iterate

The resulting trace was smoothened using a moving average filter ($t = 10$). The experiment was repeated for $k = 12$ times and the curves for each ANN architecture were averaged and then displayed.

The bar diagram on the right side of Figure 9 shows the quality of the recognition. The ANN with 43 hidden nodes performs best, the average anomaly score for the modified data-set is $6-8$ times higher than that of the original dataset. The ANN employing 32 hidden nodes has a $4-5$ times higher score and the architecture with 16 nodes was about 1.5 times above.

5.3 Implementing the Recognition Unit

As described in Figure 3 and Section 3.1, the CAID framework consists of three general modules which can be regarded as atomic Software Components (SWCs) in the AUTOSAR architecture. Each AUTOSAR SWC encapsulates part of the functionality of the application and can be implemented on its own dedicated ECU. It is also possible that all SWCs share one physical ECU as in our case. SWCs communicate between each other for exchanging data over the Virtual Function Bus (VFB). The VFB is the sum of all communication mechanisms provided by AUTOSAR on an abstract, target-independent level. During development, the SWCs with their communication interfaces are mapped to the specific target by using the configured BSW. A BSW provides the infrastructure for execution on an ECU and has to



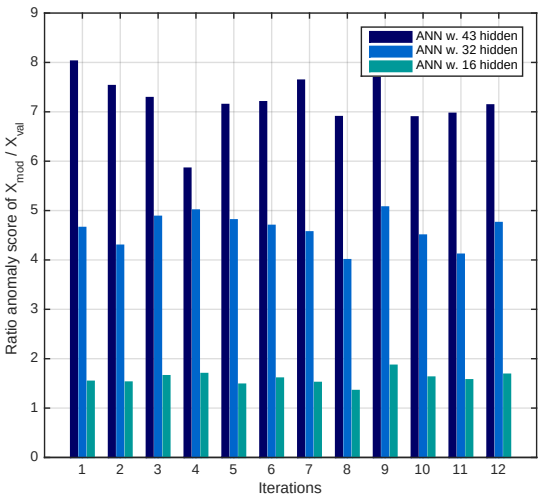


Figure 9: Ratio between anomaly score between validation set and modified set over time for different Bottleneck ANN architectures

be configured according to the specific target architecture. The BSW includes several automotive communication services like CAN or Ethernet, as well as the Runtime Environment (RTE) (i.e., Operating System (OS)) and memory services. The logical architecture of the RU is mapped to the physical hardware by the Microcontroller Abstraction Layer (MCAL) which provides target-specific drivers and pin-maps.

Figure 10 depicts a possible implementation using three ECUs, each one containing the SWC for one component of the RU. The RTE is specifically configured to connect the SWC with its necessary BSWs and implements the VFB functionality on a specific ECU. These ECUs are part of the *Diagnostic CAN* bus which connects the OBD-II port with the CGW. As we do not have direct access to the CAN lines of the vehicle, we plugged the ECU running the RU directly to the OBD-II port which is connected to the Diagnostic CAN bus.

The tasks of the SWCs can be summarized as follows:

- *Monitor:* The monitor sends periodic OBD-II PID requests to obtain the necessary parameters from the ECUs. The payloads of the PID responses are processed by calculating the absolute value of the PID according to the instructions described in [31]. These values are then stored in the internal memory of the ECU for further processing.

- *Detector:* The ANN algorithm running within this SWC checks the plausibility of a sequence of data obtained from the Monitor. The weights of the ANN have been hardcoded in the ECU. Their values have been determined previously, by offline training in MATLAB. If a manipulation is detected, an interrupt flag is set.

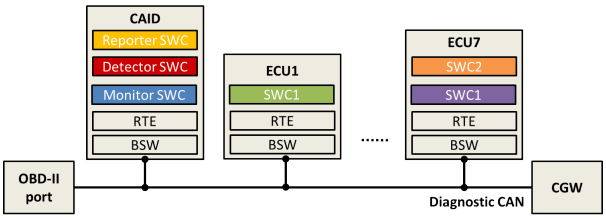


Figure 10: Example approach of RU implementation on one ECU

- *Reporter:* Eventually, this module uses the defined interrupt to perform its routine.

6 Discussion

The results presented in section 5.2 are very promising. They show that our approach was able to recognize the manipulated engine controller in our test vehicle with a high probability. The data, however, was collected in a well-defined environment on urban and highway roads and drivers were driving cautiously. Some circumstances could not be addressed with the data at hand and some challenges remain open:

- *Road conditions:* The impact of road conditions on the recognition is not yet clear. Driving on slippery roads or surfaces with differing degrees of grip has an impact on the engine control. For instance, if the traction control detects a wheel slip, it intervenes in the engine management system to counter this effect by throttling back engine power. This kind of behavior has to be included in the training set.

- *Changing drivers:* Each person has his or her unique personal style when driving a vehicle. It has been shown that recorded vehicle data is sufficient to distinguish between different drivers [8]. Particularly the temporal patterns of events such as braking helps to discriminate drivers. The impact of different drivers on the recognition is most likely low, because of the windowing technique used: Window sizes are quite long, so the impact when an event occurs is mitigated.

- *Different devices/cars:* So far, we have used one test vehicle and one particular tuning device. It would be interesting to see how different devices are recognized and, maybe, if they can be distinguished. Additionally, comparing the results between different cars is future work.

- *Vehicle aging and continued model training:* Our tests were performed over a few months, during which the test vehicle aged only insignificantly. It can be expected that the engine controller features change with increasing age. To avoid false alarms, a proper model needs to include aging and possibly refine the model by continuous training. It is unclear though how to perform such continuous training without opening the system to attacks that manipulate the engine slowly over time.

- *Maintenance and module replacement:* The lifecycle of a vehicle includes regular maintenance and also replacement of components. The manufacturer authorized maintenance, which might include software updates that modify the vehicle's behavior and that might change engine parameters, will require a re-training of CAID. Again, this could open the system to attacks.

- *Training at large scale:* The current approach requires each car to train an individual ANN. This is a clear drawback, because a training set has to be built for each car individually. Splitting the training in two phases could remove that constraint: First a generic ANN is pre-trained for one model series and then adapted using vehicle specific data [9].

- *Security:* Due to the centralized approach of having only one ECU implementing the entire RU, it is easy to deactivate the CAID by removing this ECU from the CAN network. In order to increase availability, a distributed approach can be considered by spreading the modules of the RU on different ECUs which cannot be abstracted from the IVN.

- *Data quality:* Data collection using the OBD-II interface is very convenient, because it is accessible and an entire tooling ecosystem has evolved around wireless OBD-II dongles. The sampling frequency using this interface, however, scales linearly with the number of data points queried, because of the request/reply protocol specified. For many applications this is no limitation, but for checking the engine behaviors, a higher data fidelity would be preferable. Directly connecting to the CAN bus would remove this constraint, but requires knowledge of the vehicles CAN IDs.

As an RU performs a similar function to that of a runtime monitor, the CAID approach could be integrated with the diagnostics system. There is definitely a mutual benefit between monitoring a system for safety and correct operation as well as for security. Detectors using algorithms to discern between accidental and malicious failures could then be implemented [1].

Finally, a limitation is that the recognition process did not exactly reveal what was going wrong, but was only able to raise a red flag.

7 Conclusion

Manipulation of control systems is a real threat for vehicles. An example is chip-tuning in which the engine control parameters are modified to provide more power and/or torque, sometimes with unwanted and potentially safety-critical side-effects. Malicious manipulation to modify a vehicle's driving behavior that endangers passengers' safety would be possible as well. This paper presented the CAID framework to unveil such manipulations. The effectiveness of the framework is demonstrated at the example of chip-tuning of an actual test vehicle. After training on relevant engine parameters, a Bottleneck ANNs was capable to detect deviations in the behavior of the control system. The result gained in the automotive case study is promising and it has the potential to extend to various other control systems. This approach has been proposed to overcome the limitations in today's automotive IDS that do not incorporate physical models. All future automotive IDS should incorporate mechanisms to protect the physical domain and to stay competitive with advanced automotive attacks.

8 Acknowledgments

This research was in part supported by a Marie Curie IOF Action within the 7th Framework Programme under the funding ID PIOF-GA-2012-326604 (MODESEC), and by the Michigan Mobility Transformation Center (MTC). The responsibility for the content rests with the authors.

References

[1] BASILE, C., GUPTA, M., KALBARCZYK, Z., AND IYER, R. K. An approach for detecting and distinguishing errors versus attacks in sensor networks. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on* (2006), IEEE, pp. 473–484.

[2] BROY, M., KRUGER, I., PRETSCHNER, A., AND SALZMANN, C. Engineering automotive software. *Proceedings of the IEEE 95*, 2 (Feb 2007), 356–373.

[3] CAIN, H. Applying machine learning for anomaly detection in can bus networks. In *13th escar Europe* (Cologne, Germany, 2015).

[4] CARDENAS, A. A., AMIN, S., AND SASTRY, S. Secure control: Towards survivable cyber-physical systems. In *The 28th International Conference on Distributed Computing Systems Workshops* (2008), IEEE, pp. 495–500.

[5] CHANDOLA, V., BANERJEE, A., AND KUMAR, V. Anomaly detection: A survey. *ACM Comput. Surv. 41*, 3 (July 2009), 15:1–15:58.

[6] CHARETTE, R. This car runs on code. *IEEE Spectrum* (Feb 2009).

[7] CHECKOWAY, S., MCCOY, D., KANTOR, B., ANDERSON, D., SHACHAM, H., SAVAGE, S., KOSCHER, K., CZESKIS, A., ROESNER, F., KOHNO, T., ET AL. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium* (2011), San Francisco.

[8] ENEV, M., TAKAKUWA, A., KOSCHER, K., AND KOHNO, T. Automobile driver fingerprinting. *Proceedings on Privacy Enhancing Technologies 2016*, 1 (2016), 34–50.

[9] ERHAN, D., BENGIO, Y., COURVILLE, A., MANZAGOL, P.-A., VINCENT, P., AND BENGIO, S. Why does unsupervised pretraining help deep learning? *The Journal of Machine Learning Research 11* (2010), 625–660.

[10] FOSTER, I., PRUDHOMME, A., KOSCHER, K., AND SAVAGE, S. Fast and vulnerable: A story of telematic failures. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)* (Washington, D.C., Aug. 2015), USENIX Association.

[11] GAIDHANE, V. H., SINGH, V., HOTE, Y. V., KUMAR, M., ET AL. New approaches for image compression using neural network. *Journal of Intelligent Learning Systems and Applications 3*, 04 (2011), 220.

[12] HOPPE, T., KILTZ, S., AND DITTMANN, J. Security threats to automotive can networks practical examples and selected short-term countermeasures. In *Computer Safety, Reliability, and Security*, M. Harrison and M.-A. Sujan, Eds., vol. 5219 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 235–248.

[13] KOSCHER, K., CZESKIS, A., ROESNER, F., PATEL, S., KOHNO, T., CHECKOWAY, S., MCCOY, D., KANTOR, B., ANDERSON, D., SHACHAM, H., ET AL. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on* (2010), IEEE, pp. 447–462.

[14] LEE, E. A. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on* (2008), IEEE, pp. 363–369.

[15] MILLER, C., AND VALASEK, C. Adventures in Automotive Networks and Control Units. Tech. rep., 2014.

[16] MILLER, C., AND VALASEK, C. Remote Exploitation of an Unaltered Passenger Vehicle. Tech. rep., 2015.

[17] MUTER, M., AND ASAJ, N. Entropy-based anomaly detection for in-vehicle networks. In *Intelligent Vehicles Symposium (IV), 2011 IEEE* (June 2011), pp. 1110–1115.

[18] NANOPOULOS, A., ALCOCK, R., AND MANOLOPOULOS, Y. Feature-based classification of time-series data. *International Journal of Computer Research*, 10 (2001), 49–61.

[19] PARVIAINEN, E. *Intelligent Data Engineering and Automated Learning – IDEAL 2010: 11th International Conference, Paisley, UK, September 1-3, 2010. Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, ch. Dimension Reduction for Regression with Bottleneck Neural Networks, pp. 37–44.

[20] SAE. Sae j3061 - cybersecurity guidebook for cyber-physical vehicle systems, 2016.

[21] SCHNEID, O. *Autosar Compendium, Part 1: Application & RTE*. CreateSpace, 2015.

[22] SKOROBOGATOV, S. P. Copy protection in modern microcontrollers.

[23] SOLUTIONS, O. Stn1110 - multiprotocol obd to uart interpreter datasheet. Datasheet, 2010.

[24] STUDNIA, I., NICOMETTE, V., ALATA, E., DESWARTE, Y., KAANICHE, M., AND LAAROUCHI, Y. Survey on security threats and protection mechanisms in embedded automotive networks. In *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on* (June 2013), pp. 1–12.

[25] SZTIPANOVITS, J., AND KARSAI, G. Model-integrated computing. *Computer 30*, 4 (Apr 1997), 110–111.

[26] UJIIE, Y., KISHIKAWA, T., HAGA, T., MATSUSHIMA, H., WAKABAYASHI, T., TANABE, M., KITAMURA, Y., AND ANZAI, J. A method for disabling malicious can messages by using a centralized monitoring and interceptor ecu. In *13th escar Europe* (Cologne, Germany, 2015).

[27] WASICEK, A. Copy Protection for Automotive Electric Control Units using Authenticity Heartbeat Signals. In *IEEE 10th International Conference on Industrial Informatics (INDIN)* (2012).

[28] WASICEK, A. Protection of Intellectual Property Rights in Automotive Control Units. In *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* (2014), vol. 7, pp. 201–212.

[29] WASICEK, A., LEE, E., KIM, H., GREENBERG, L., IWAI, A., AND AKKAYA, I. System simulation from operational data. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE* (June 2015), pp. 1–6.

[30] WASICEK, A., AND WEIMERSKIRCH, A. Recognizing manipulated electronic control units. October 2015.

[31] WIKIPEDIA. Obd-ii pids — Wikipedia, the free encyclopedia, 2016. [Online; accessed 10-January-2016].