HAW HAMBURG

Masterarbeit

Sebastian Szancer

Traffic Analysis in V2X Application-Level Gateways

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Sebastian Szancer

Traffic Analysis in V2X Application-Level Gateways

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Franz Korf
Zweitgutachter: Prof. Dr. Martin Becke

Eingereicht am: 14. April 2021

Sebastian Szancer

Thema der Arbeit

Traffic Analysis in V2X Application-Level Gateways

Stichworte

V2X, Security, Application-Level Gateway, Semantische Analyse, Kontext-sensitiv

Kurzzusammenfassung

Zukünftig werden Fahrzeuge mit einer Vielzahl von Teilnehmern in diversen Netzwerken, von VANETs (Vehicular ad-hoc Networks) bis hin zum Internet kommunizieren. Das werden andere Fahrzeuge, Infrastruktur, wie z.B. Ampeln, oder Services in der Cloud sein. Diese V2X Kommunikation ist von zentraler Bedeutung, da sie die Verkehrssicherheit und Verkehrseffizienz erhöht, zur leichteren Wartung von Fahrzeugen beiträgt und eine wichtige Rolle für die Realisierung autonomer Fahrzeuge spielt. Es ist zwingend notwendig, dass die V2X Kommunikation entsprechend abgesichert wird, da sie sicherheitskritische Funktionen umfasst. Die Absicherung geschieht durch ein V2X Security Gateway im Fahrzeug, welches den Fahrzeug-internen Diensten, die mit der Außenwelt kommunizieren, als Proxy dient und sowohl die kryptografische Sicherheit, als auch die Sicherheit auf dem Internet-, Transport- und Application-Layer gewährleistet. Eine zentrale Komponente eines solchen V2X Security Gateways ist das V2X Application-Level Gateway, welches die Proxy-Funktion, die kryptografische Sicherheit und Sicherheit auf dem Application-Layer realisiert. Die Sicherheit auf dem Application-Layer umfasst die kontext-sensitive semantische Analyse von Anwendungsdaten, die Erkennung von Application-Layer Protokoll-Verletzungen und die Erkennung von Application-Layer DoS-Angriffen. Diese Arbeit stellt das Konzept und eine Prototyp-Implementierung eines solchen V2X Application-Level Gateways vor. Die Implementierung wurde in einem Test-Netzwerk, welches das interne Fahrzeug-Netzwerk repräsentiert, evaluiert. In dem Netzwerk, bestehend aus einem *Edgecore SDN Switch*, *Intel NUCs* und *Raspberry Pis*, welche Fahrzeug ECUs repräsentieren, wurden mehrere V2X Szenarios simuliert: die Steuerung des Kofferraums über HTTPS, das Erhalten von Verkehrs-Updates über MQTT und ein einfacher V2V Traffic Safety Service der ETSI CAM nutzt. Jedes Szenario beinhaltete Angriffe, welche für die Evaluierung des V2X Application-Level Gateways entworfen wurden. Es wurde gezeigt, dass das V2X Application-Level Gateway alle Angriffe erkennen und darauf reagieren konnte.

Sebastian Szancer

Title of Thesis

Traffic Analysis in V2X Application-Level Gateways

Keywords

V2X, Security, Application-Level Gateway, Semantic Analysis, Context-sensitive

Abstract

Future cars will communicate with a variety of entities ranging from other vehicles and infrastructure, such as traffic lights, to Internet-based services running on remote servers. This V2X communication is essential for future vehicles, since it increases traffic safety and traffic efficiency, contributes to easier vehicle maintenance and also plays an important role for the realisation of autonomous vehicles. It is necessary that V2X communication is appropriately secured, especially since it includes safety-critical communication. This can be done with a V2X Security Gateway in the vehicle, which serves as a proxy for vehicle-internal services communicating with the outside world and ensures cryptographic security as well as security on the internet-, transport- and application layer. A central component of such a V2X Security Gateway is the V2X Application-Level Gateway, which ensures security on the application layer, including a context-sensitive semantic analysis of application data, detection of application layer protocol violations and detection of application layer DoS attacks. It also realises the proxy-functionality and ensures cryptographic security. This work presents a concept and prototype implementation of such a V2X Application-Level Gateway. The implementation was evaluated with the V2X Application-Level Gateway software run on an *Intel NUC* integrated in a test network representing an internal vehicle network. In this network, consisting of an *Edgecore SDN switch* and *Intel NUCs* and *Raspberry Pis* representing vehicle ECUs, several V2X scenarios like remotely controlling the vehicle trunk via HTTP, receiving traffic updates via MQTT and a basic V2V traffic safety service using the ETSI CAM were simulated. Each scenario included realistic attacks devised for evaluating the V2X Application-Level Gateway. It was shown that with the traffic analysis in the V2X Application-Level Gateway all attacks could be detected and handled.

Contents

List of Figures

List of Tables

1 Introduction

Future cars will communicate with a variety of entities ranging from other vehicles (V2V: vehicle to vehicle) and infrastructure such as traffic lights (V2I: vehicle to infrastructure), to Internet-based services running on remote servers. All this external vehicle communication is called V2X communication. While most of the V2X communication will be IP-based, real-time V2V or V2I communication will be realised over network- and transport protocols developed for that purpose, such as WSMP [113, 64]. V2X communication is realised via a Connectivity-Gateway [103] using different technologies such as DSRC (Dedicated Short Range Communication), Wi-Fi (IEEE 802.11), LTE [26], 5G or Bluetooth.

For future vehicles V2X communication is essential. It increases traffic safety and traffic efficiency, contributes to easier vehicle maintenance and also plays an important role for the realisation of autonomous vehicles. E.g. "over the air" ECU software updates allow the fast maintenance of a great number of vehicles without the need to go to a car service station. Traffic efficiency is enhanced by optimised navigation and route planning (which in case of electric vehicles may depend on charging infrastructure) considering conditions such as road traffic or weather in live-time. Safety applications like collision avoidance enhance traffic safety. And some innovative functionality such as automated coordinated driving cannot be realised without V2X.

In general communication in the automotive context is divided into 5 domains: engine control, infotainment, maintenance, safety electronics (e.g. ABS, airbag, seat-belt pretensioner) and comfort (e.g. power windows) [97]. V2X communication encompasses the infotainment, maintenance and engine control domain, ranging from music streams to ECU-software updates and inter-vehicle collision avoidance. Single use cases from the comfort domain, such as setting the car heating, could be realised as well. Although most of the V2X communication is soft real-time, in some instances, like the above mentioned collision avoidance, it is hard real-time with deadlines in the milliseconds [12, 78, 118]. It is mandatory that the V2X communication is appropriately secured, since it encompasses safety-critical domains. This can be done with a V2X Security Gateway (see chapter 2,

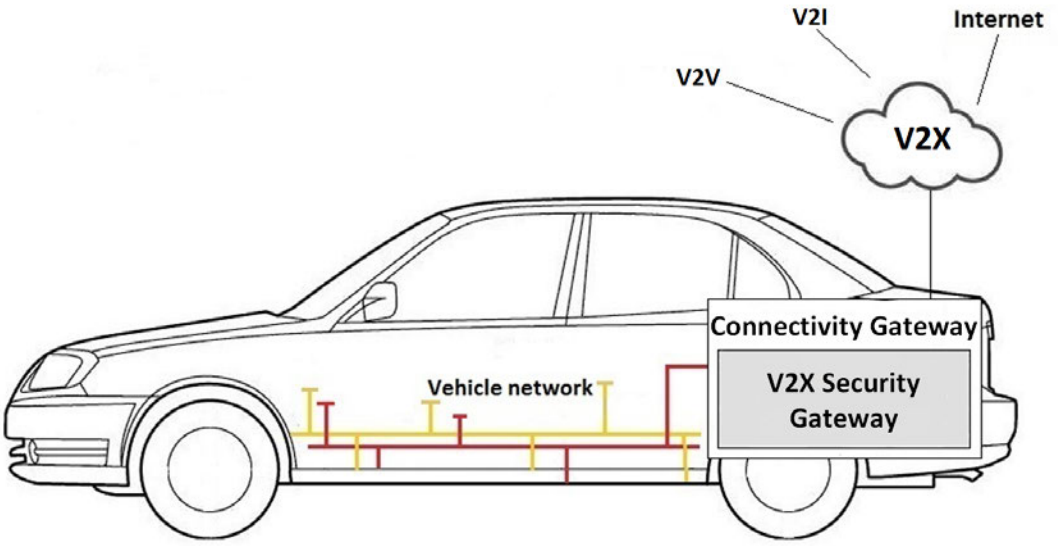

Figure 1.1: Schematic: Modern car with V2X Security Gateway

section 2.2.4), which is part of the Connectivity-Gateway, see figure 1.1.A central component of such a V2X Security Gateway is the V2X Application-Level Gateway. The V2X Application-Level Gateway serves as a proxy decoupling the in-vehicle network from external communication partners. It ensures cryptographic security and application-level security including a context-sensitive semantic analysis of application data, detection of application layer protocol violations and detection of application layer DoS attacks. Additionally it allows role-based access to in-vehicle resources and bandwidth control of V2X traffic. The aim of this work is the development of a concept and prototype implementation of such a V2X Application-Level Gateway.

This work is organised as follows: chapter 2 gives the necessary background on automotive IT, IT-security concepts and semantic analysis, chapter 3 gives an overview of related work, emphasising the contribution of this work and proceeds with a requirements analysis. Chapter 4 discusses the concept of the V2X Application-Level Gateway emphasising context-sensitive semantic analysis and DoS detection based on the preceding requirements analysis. In chapter 5 a prototype implementation is presented, which is evaluated in chapter 6. Chapter 7 concludes this work and discusses future work.

2 Background

This chapter gives the necessary background on automotive IT focusing on V2X communication, relevant IT-security concepts like security gateways and gives a definition of the term ”semantic analysis” used in this work. It also gives an overview of application layer protocols used in V2X communication and of known attacks related to these protocols as well as their countermeasures. The application layer protocols range from automotive protocols like the ETSI (European Telecommunications Standards Institute) ”Cooperative Awareness Basic Service” or ”Diagnostics over IP” to traditional Web protocols like HTTP(S).

2.1 IT-Security Concepts

2.1.1 Cryptographic Security

A key concept in IT-security is cryptographic security. In this work cryptographic security is defined as confidentiality, integrity and authenticity. Confidentiality means that a message sent by a sender to a receiver cannot be read by a third party. Integrity means that a message sent by a sender to a receiver cannot be modified by a third party without the receiver noticing it. Authenticity means that for every message the receiver can check if the message originated from the sender or a third party. Cryptographic security can be achieved with cryptographic algorithms. Cryptographic algorithms are classified into symmetric algorithms, e.g. AES or DES and asymmetric algorithms, e.g. RSA or elliptic curves [93]. Symmetric algorithms are generally faster than asymmetric algorithms, but the number of cryptographic keys increases exponentially with a growing number of users, while with asymmetric algorithms this increase is linear. The secure storage of the cryptographic keys is critical for ensuring the cryptographic security of a system.

2.1.2 Security Gateways, Application-Level Gateways and Intrusion Detection Systems

Another important IT-security concept are security gateways. A security gateway divides a network into an external network and an internal network and all communication between the external and internal network runs exclusively over the security gateway. E.g. the V2X security gateway separates the in-vehicle network from the internet or VANETs and controls all traffic going through it, thus protecting the in-vehicle network from threats in the external network. Depending on the OSI layer a component controlling traffic operates on, it is either a packet filter, operating on the link-, internet- and transport layer, or an application-level gateway (ALG) operating on the application layer [86]. An ALG is a proxy which only forwards packets after controlling them on the application-layer [15]. If the data are encrypted, the ALG should have the necessary cryptographic functionality to decrypt them and encrypt them again before forwarding the data. The definition of "security gateway" used in this work is a component combining packet filters with an ALG [86, 15]. A related term is "firewall". In literature, both the separate packet filter and ALG, as well as a component combining packet filters with an ALG are called "firewall" [86, 71]. So the terms "firewall" and "security gateway" can be used as synonyms, but for clarity, in this work only the term "security gateway" is used.
Another related term is "intrusion detection system" (IDS). An IDS detects attacks by analysing network traffic [86]. Generally an IDS is classified as either "anomaly-based" or "specification-based" [72]. An anomaly-based IDS detects attacks by identifying deviations from a defined norm. A simple example of such a deviation, or anomaly, is a significantly increased packet rate. Known attacks with specific patterns can be described with specifications. A specification-based IDS detects attacks by checking if messages correspond to a defined specification, describing an attack. Instead of "specification-based", also the terms "signature-based" or "rule-based" are used [60]. A problem with specification-based intrusion detection systems is, that only known attacks can be detected and the IDS has to be constantly updated. Also, to cover as many attacks as possible, many specifications are needed [60]. With anomaly-based intrusion detection systems false positive rates (i.e. a legitimate message is falsely classified as anomalous) have to be minimised, especially in the automotive context, since false alarms can have more severe consequences in vehicles than in e.g. a PC system [109, 60]. Detecting anomalies is not limited to network metrics, such as e.g. packet rates, but can encompass the semantics of messages. The IDS described in [60] e.g. checks if messages sent to

remote peers contain sensitive data. Anomaly detection can also be context-sensitive, i.e. dependent on the current state of the system. E.g. for classifying a message as normal or anomalous in a vehicle the vehicle's current state (e.g. *parked*, *driving*, *crashed*) is considered [109]. Increasingly, machine learning is used for anomaly detection [49]. The IDS is trained to learn the "normal behaviour" based on which it detects anomalies. An IDS can either be network-based, i.e. located in the network or host-based, i.e. located in the host it is to protect. An IDS that does not only detect attacks, but also reacts to them, e.g. by dropping packets, is called an "intrusion prevention system" (IPS) or "intrusion detection and prevention system" (IDPS) [50].

2.1.3 Denial-of-Service Attacks

Denial-of-Service (DoS) attacks target the availability of a system, so that service to legitimate users is denied. By overwhelming a target with traffic, its performance is degraded, possibly to the extent of rendering it completely useless. DoS attacks can take different forms and occur on different OSI layers but the general concept is always the same: exhaust a resource of the target system so that legitimate users cannot use it. An example of a transport layer DoS attack is TCP SYN flooding, where an attacker generates a large number of packets with random source addresses and the TCP SYN flag set, requesting allocation of a buffer at the receiving node and after the entire buffer space is exhausted, legitimate users cannot connect with the victim machine [18]. DoS attacks can also take place on the application layer: an attacker could try to overwhelm a service with HTTP(S) requests or MQTT messages. DoS attacks where the attacker uses a large number of sources to flood the target with traffic are called Distributed Denial-of-Serice (DDoS) attacks [31].
Identifying (D)DoS traffic and (D)DoS protection are no trivial tasks. When it comes to DDoS protection, the action a single network node, e.g. a single vehicle, can take to protect itself from DDoS attacks is limited. Effective DDoS countermeasures are network-based services aiming at filtering and dropping malicious traffic before it reaches the intended target [68, 27]. For a single network node to protect itself from DoS attacks, or at least mitigate these attacks, a mechanism known as *Hashcash* [7] has been proposed. The basic principle behind this mechanism is to assign a cost to each participation in a protocol with the protected network node, with a participation being e.g. making a request. The requester has to compute a token (the computation is based on finding partial hash-collisions) before the node proceeds with the protocol, e.g. processes the

request. Thus a DoS attack to overwhelm the node with requests becomes unfeasible since every request requires a significant amount of resources, in this case, computational power. This solution comes with the cost of reduced performance due to the computation of the tokens and thus may not be applicable when performance is a critical factor. Also, it is only suitable for DoS attacks, but not DDoS attacks, since in DDoS attacks the attacker has a large number of processors at his disposal and thus can easily compute a large number of tokens. Apart from applying the *Hashcash* mechanism a single network node can protect itself from DoS attacks by deploying a host-based IDS detecting DoS attacks and dropping messages so that the node is not overwhelmed by traffic. Ideally only DoS traffic would be dropped and not legitimate traffic. However, identifying DoS traffic and distinguishing DoS attacks from legitimately increased traffic is no easy task. In many cases the traffic rate alone is not sufficient to define rules for identifying DoS traffic. Instead, a more thorough analysis of the traffic is required, e.g. examining the proportion of connection requests in all messages [53]. A high proportion of connection requests is not typical for legitimate traffic and could indicate a DoS attack. For certain use cases a (configurable) static rule set may be sufficient to identify DoS traffic, but generally intrusion detection systems using machine learning are proposed as host-based DoS countermeasures [53, 117, 76]. The advantage of a configurable static rule set is being a lightweight solution. The rules do not have to be limited to the traffic itself, but can also include the source of traffic, e.g. "did the source of the packet appear before or after the detection of the attack" [76]. While such a rule does not contribute to detection, it can help to distinguish DoS traffic from legitimate traffic. The more distributed a DoS attack is, the harder it becomes to distinguish DoS traffic from legitimate traffic, since with an increasing number of sources, the DoS traffic per source decreases, more resembling legitimate traffic in that respect. With DDoS an attacker does not have to rely on fewer sources generating an abnormal amount of traffic (per source) but can instead attack with a large number of sources generating normal amount of traffic (per source).

2.2 IT in Modern Vehicles

2.2.1 The modern Vehicle Network

Since the V2X Application-Level Gateway is to protect the vehicle, or more specifically the in-vehicle network, from attacks over V2X, this section gives a brief overview of

the future in-vehicle network. Traditionally the in-vehicle network consisted of multiple ECUs (Electronic Control Unit) interconnected over different bus systems like CAN or FlexRay. Modern in-vehicle networks consist of 50 to 100 ECUs. In the future the ECUs will be interconnected over a Real-time Ethernet backbone with peripheral bus systems, e.g. CAN [99], see figure 2.1. Increasing demand for bandwidth of vehicle

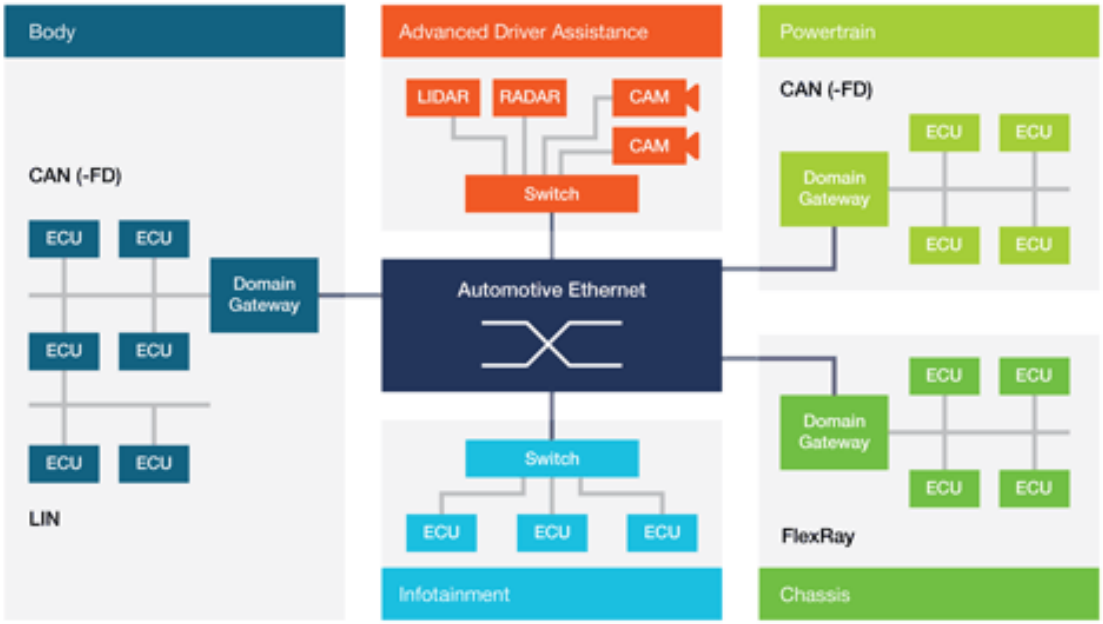

Figure 2.1: Modern in-vehicle network [111]

applications, e.g. from the infotainment domain or Advanced Driver Assistance Systems (ADAS), makes the shift to a Real-time Ethernet backbone necessary. The traditional bus systems like CAN will be used at the periphery of the network connecting sensors and actuators requiring only low bandwidths. Also, the use of Ethernet technology in the vehicle network allows complex topologies and the use of Software-Defined Networking (SDN) facilitating security [47].

For future automotive networks a reduction of the number of ECUs and a centralisation of computing power, so that multiple software components run on the same ECU, is proposed [16]. Instead of adding new ECUs to offer new functionality, new software components would be integrated into existing ECUs. For the software architecture of future automotive networks a service-oriented architecture (SOA) is proposed [16]. In a SOA service providers offer services to service consumers. Lose coupling of providers and consumers is achieved with a middle-ware. An important aspect in the context of services is moving functionality from the vehicle to the cloud: resource-intensive services

can be moved to powerful external servers, which only increases the significance of V2X communication.

2.2.2 V2X Communication

V2X communication can be divided into communication with cloud-based services on the one hand and V2V and V2I communication on the other hand. The communication with cloud-based services is traditional Web communication based on the internet infrastructure and the IP protocol stack using established application layer protocols like HTTP(S) or MQTT. V2V and V2I communication, realising applications like safety-critical collision avoidance, have hard real-time requirements and rely on direct short-range communication in Vehicular Ad Hoc Networks (VANETs). While V2I solutions exploit Road Side Units (RSUs) and require a large deployment investment, especially when coupled with an RSU-to-RSU communication infrastructure, V2V solutions avoid costly RSU installations and opportunistically exploit the VANET for the delivery of messages [79]. The main challenge for the communication in VANETs are the highly dynamic topologies, which come with frequent link breakages, network fragmentation, and a high number of packet collisions and interferences [79]. An additional challenge for a subset of safety-critical V2V applications requiring the exact (relative) positions of vehicles, e.g. for collision avoidance, is the reliability of positioning information from different sources like GPS and sensors, e.g. LIDAR or radar, under varying conditions like bad weather or driving in a tunnel [52]. This aspect however, is beyond the scope of this work.
The technology to realise VANET communication is Dedicated Short Range Communication (DSRC), which provides local-area, low-latency network connectivity and is based on IEEE 802.11 and standardised as IEEE 802.11p [63] covering the physical and medium access control (MAC) layers [112, 113, 52]. The spectrum allocated for V2X communication is 5.850 GHz to 5.925 GHz in the US and 5.855 to 5.925 GHz in Europe [65]. It is divided into several channels, see figure 2.2, page 9, over which entities exchange V2X application data. One channel (Ch.178) is the control channel, which is used to manage the exchange of application data between entities (e.g. for service providers to advertise their service, including the information on which channel it operates and for service consumers to receive this information about services of interest to them). The remaining channels are used to exchange V2X application data, with one channel designated for time-critical safety-applications like collision avoidance (Ch.172 in the US) [65].

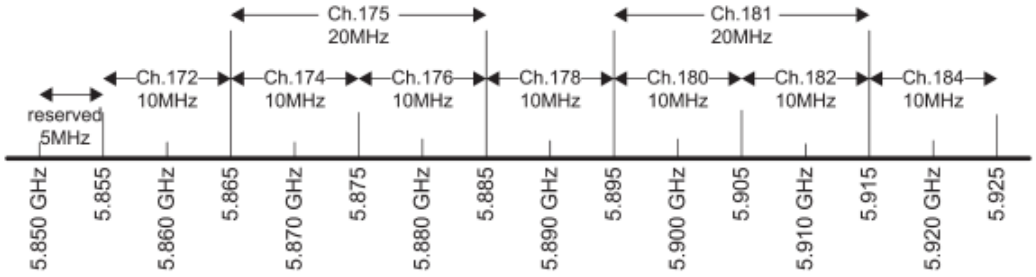

Figure 2.2: The 5.9 GHz spectrum allocation [65]

All devices communicating over the channels are synchronised over UTC (Coordinated Universal Time). On the link layer the devices exchange Ethernet frames via unicast or multicast/broadcast. In the US the protocol stack for realising V2X communication is specified in IEEE standards, in Europe it is specified in ETSI (European Telecommunications Standards Institute) standards. Initially, both the IEEE and ETSI V2X communication stacks were based on IEEE 802.11p, however currently an alternative technology to IEEE 802.11p is considered for V2X (especially V2V/V2I) communication in Europe, namely the LTE-based LTE-V2X PC5, specified in ETSI EN 303 613 [39]. Both the ETSI and IEEE standards specify that above the link layer two protocol stacks shall be used: the traditional IP-based stack on the one hand and a V2X-specific stack with network layer and transport layer protocols designed for V2X communication on the other [38, 113]. The V2X-specific stack is for high-priority, time-sensitive communication, while the IP-based stack is for supporting less demanding traditional Internet applications [113]. Since the high-priority, time-sensitive communication, e.g. for collision avoidance, is limited to local peers and thus a global addressability, as offered by IP, is unnecessary and the cost of the IP stack overhead in a dynamic environment with hard real-time requirements is detrimental, this approach makes sense. The reason for IP connectivity in VANETs [83] is facilitating compatibility and interoperability with traditional internet or IoT applications by making a VANET node globally addressable and enabling the use of the IP protocol stack in V2V and V2I communication.
The IEEE 1609 set of standards covers the layers based on IEEE 802.11p. It not only covers the OSI network- and transport layers, but also a security layer designed to ensure the cryptographic security of V2X communication. Collectively, IEEE 802.11p and IEEE 1609 are called wireless access in vehicular environments (WAVE) [113]. On the network and transport layer IEEE 1609.3 [64] specifies the use of the WAVE Short Message Protocol (WSMP) for a fast and efficient message exchange in VANETs, both for safety-critical and non safety-critical applications [113, 112], see figure 2.3, page 10. WSMP

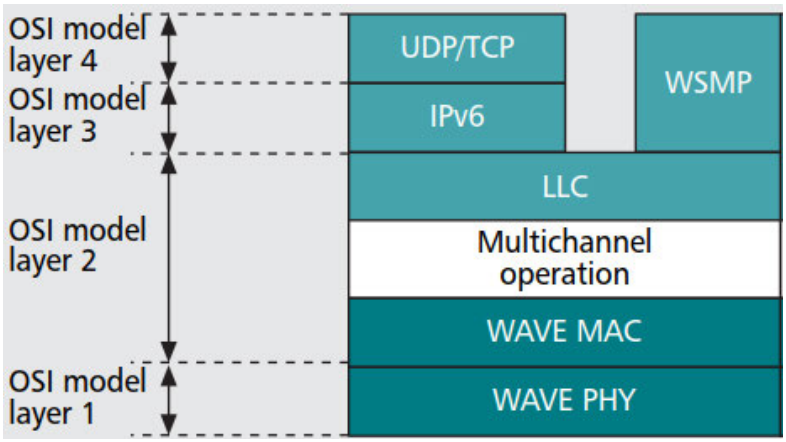

Figure 2.3: Overview: WAVE OSI layers [113]

is used to transport application layer protocols. In WSMP an application is identified by the Provider Service Identifier (PSID). The PSID is the equivalent to a port number, e.g. the ETSI Cooperative Awareness Basic Service, see below, is identified by the PSID ”0x24”. A message in WSMP is called a WAVE Short Message (WSM). It consists of a network header (WSMP-N-Header) providing network protocol functions, a transport header (WSMP-T-Header), which contains the PSID, providing transport protocol functions and a payload containing the application data, see figure 2.4. The WSM is encapsulated in an Ethernet frame (Ether-Type: 0x88DC).

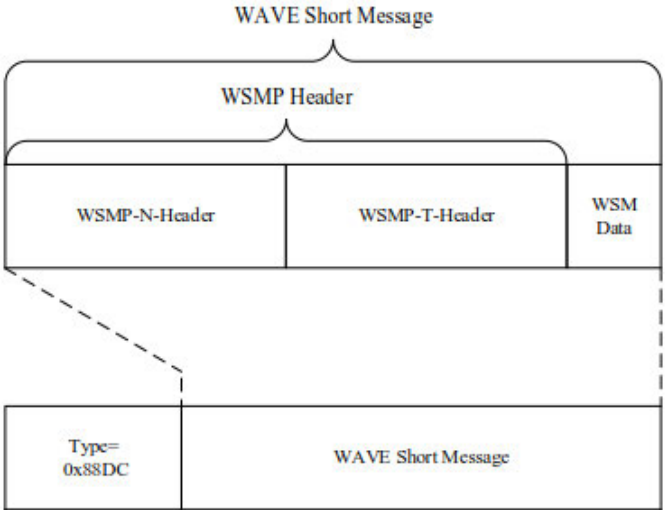

Figure 2.4: Structure of a WSM [64]

A source application sends the WSM, encapsulated in an Ethernet frame, with a destination MAC address and a PSID. It is delivered to the receiving application based on the PSID and the destination MAC address. When it comes to quality of service (QoS) WSMP reverts to IEEE 802.11p (offering either "QoSAck" or "QoSNoAck" as per IEEE Std 802.11) and offers no additional QoS.
V2V applications such as e.g. collision avoidance, which are transported over WSMP (or other V2X network/transport protocols) have to be standardised so that vehicles from different manufacturers can interact correctly. The idea is a common base message standard for e.g. safety applications, which specifies the message structure and the rate at which the messages are broadcasted. Such a common base message containing all relevant information such as vehicle position, vehicle speed, acceleration, driving direction etc. would ensure the compatibility of V2V safety applications, like collision avoidance, from different manufacturers.
One such standard is the ETSI Cooperative Awareness Basic Service specified in ETSI EN 302 637-2 [37]. It defines the structure of the base safety message, called Cooperative Awareness Message (CAM), containing all relevant information regarding the vehicle (e.g. its width and length) and its dynamics (e.g. position, speed, acceleration, heading etc.). Figure 2.5 shows the general structure of an ETSI CAM as specified in [37]. The ITS (*Intelligent Transport System*) PDU (*Packet Data Unit*) header, Basic Container and HF (*High Frequency*) Container are mandatory fields, while the rest is optional. Every CAM also contains a timestamp, i.e. the time of the generation of the CAM. The ITS

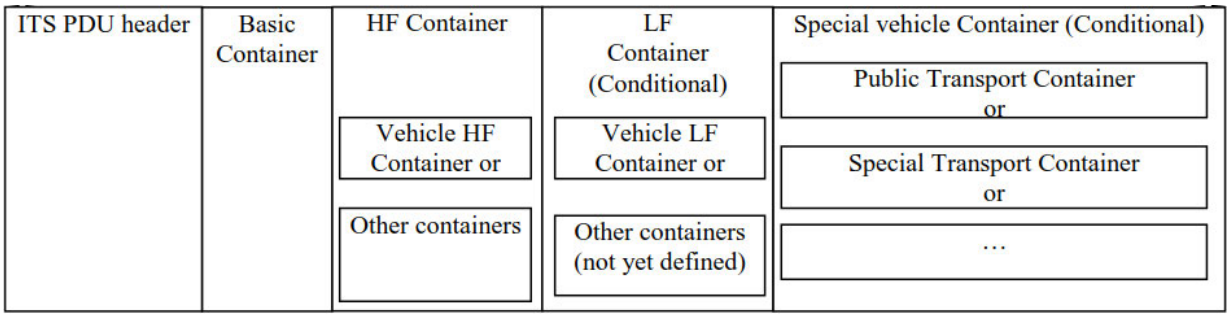

Figure 2.5: General structure of an ETSI CAM [37]

PDU header contains the protocol version and a unique identifier of the sender (*station ID*), as well as an ID identifying the message as a message of the type ETSI CAM. The Basic Container contains both the type of the sending system, e.g. passenger car or truck and the latest geographic position of the sending system. The HF Container contains all fast-changing information of the system, such as e.g. speed, see figure 2.6, page 12.

```
BasicVehicleContainerHighFrequency ::= SEQUENCE {
    heading Heading,
    speed Speed,
    driveDirection DriveDirection,
    vehicleLength VehicleLength,
    vehicleWidth VehicleWidth,
    longitudinalAcceleration LongitudinalAcceleration,
    curvature Curvature,
    curvatureCalculationMode CurvatureCalculationMode,
    yawRate YawRate,
    accelerationControl AccelerationControl OPTIONAL,
    lanePosition LanePosition OPTIONAL,
    steeringWheelAngle SteeringWheelAngle OPTIONAL,
    lateralAcceleration LateralAcceleration OPTIONAL,
    verticalAcceleration VerticalAcceleration OPTIONAL,
    performanceClass PerformanceClass OPTIONAL,
    cenDsrcTollingZone CenDsrcTollingZone OPTIONAL
}
```

Figure 2.6: ETSI CAM HF Container fields [37]

Most information in a CAM, e.g. the vehicle position, come with a confidence, since often a 100% accuracy cannot be guaranteed. In case of the vehicle position its confidence is defined as an ellipse (with the centre being being the vehicle position) with a predefined confidence level (e.g. 95%).
It is specified that CAMs are broadcasted without acknowledgements or retransmissions. Instead of a fixed broadcasting rate, adaptive rates depending on the sending vehicle's behaviour were proposed. E.g. an accelerating vehicle broadcasts CAMs more frequently than a vehicle at a low constant speed. The minimum time between sending two CAMs is 100 ms and the maximum time is 1000 ms. Within this minimum and maximum the CAM frequency depends on the dynamics of the sending vehicle and the channel congestion status. The default interval between two CAMs is the maximum time of 1000 ms. This interval decreases when the vehicle's position changes more than 4 m, or its speed changes by more than 5 m/s, or its heading changes by more than 4 degrees within a certain time after the last CAM. The interval is set back to the maximum time of 1000 ms when the changes in position, speed and direction remain under the defined thresholds for a certain time, which is configurable.

2.2.3 Digital Maps for Automotive Applications

For numerous V2X applications, ranging from navigation to safety critical real-time applications like collision avoidance, digital maps are essential. A digital map allows a vehicle to be aware of its own position in the world, the positions of other objects like nearby vehicles or trees and the geography of the surrounding environment, e.g. the structure of the road network. A digital map can contain different types of objects: static objects with small spatial expansion, such as road signs or trees, static continuous structures, such as roads and dynamic objects, such as vehicles or pedestrians. Depending on the use case, both the mapping process and the types of objects contained in the map and thus the requirements regarding the map vary [70]:

- Navigation: the whole environment passed by the mapping vehicle should be mapped. The map building can be done offline. In the automotive domain traditionally the focus is on static continues structures like roads, but depending on the context a navigational digital map can also contain static objects like e.g. trees. Since navigational maps cover larger areas, efficient storage is important.
- Real-time applications: the near environment of the vehicle needs to be mapped, including both static and dynamic objects. Only up to date information is of interest and the map is built online in real-time. This kind of maps is constructed with information from sensors, such as e.g. laser scanners, and used for real-time applications like collision avoidance. With the development of V2X communication not only on-board sensors contribute to the construction of such maps, but also information obtained from V2X messages from other traffic participants [106, 34].

In the automotive domain a concept to combine both the navigational and real-time application maps has emerged: the Local Dynamic Map (LDM) [106, 34]. The LDM combines static digital maps (i.e. navigational maps) with dynamic objects. It consists of four layers, see figure 2.7, page page 14, with each layer containing different types of objects [106, 34]:

1. Permanent Static: contains the information obtained from static digital maps, i.e. static continuous structures such as roads and intersections.
2. Transient Static: extends the first layer by adding static objects with small spatial extension, such as roadside infrastructure and landmarks.

3. Transient Dynamic: adds temporary local information like weather or traffic conditions, e.g. traffic jams, and traffic lights signal phases.

4. Highly Dynamic: extends the first three layers by adding dynamic objects such as vehicles and pedestrians. The information about those objects is not limited to their positions, but also includes other highly dynamic data like e.g. their velocity, heading, etc. obtained from V2X messages and sensor data.

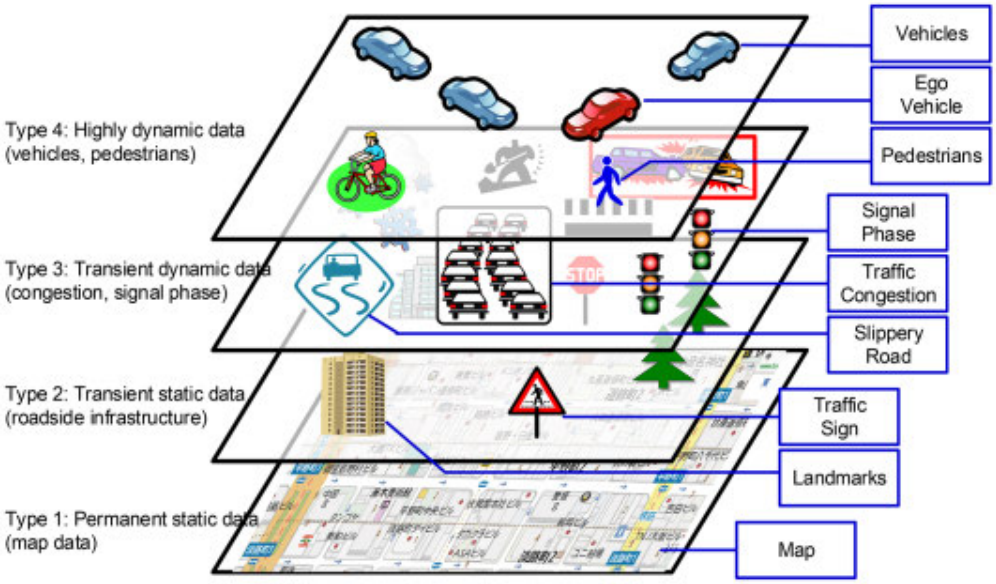

Figure 2.7: Layers of the Local Dynamic Map (LDM) [106]

The LDM stores the information in a database and has SQL as a query language. It is standardised by ETSI and currently specified in ETSI EN 302 895 [36] and already mature implementations both commercial and from the scientific community exist [106, 34]. However, they have certain shortcomings [34]: Most of the current approaches have a database-centric model of the LDM using a static schema with the LDM objects being mapped directly to relational tables. So new types of objects require a modification of the schema, which makes it harder to add new domains, e.g. traffic regulations. Also, the database schema cannot simply capture and query class hierarchies and the dependencies between the different objects as it is not graph-based. Working on top of a static database ignores the streaming nature of the dynamic LDM data, which advocates for real-time queries over large amounts of data "in-stream", i.e. without storing. Thus [34] propose the use of an additional stream database for the stream data suited for that purpose. For

expressing relations between objects and modelling them as class hierarchies an ontology is used. As the recent work on LDMs [34] shows, their development is still ongoing.

2.2.4 V2X Security Gateway

V2X communication is secured with a V2X Security Gateway, the general concept of which was developed in [110]. It generally consists of 3 components, combined in a common PAP-structure (packet filter - application-level gateway - packet filter - structure) [15]: 2 stateful packet filters, the first one for inbound-traffic, the second one for outbound-traffic and an application-level gateway, the V2X Application-Level Gateway, see figure 2.8.

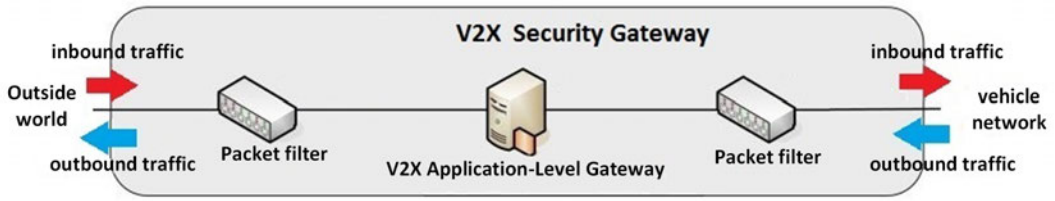

Figure 2.8: Overview: V2X Security Gateway architecture

The stateful packet filters offer security on the internet- and transport layer protecting against attacks like TCP SYN flooding. The V2X Application-Level Gateway offers security on the application layer. It serves as a proxy decoupling the in-vehicle network from external communication partners and ensures cryptographic security. Additionally it allows role-based access to in-vehicle resources and bandwidth control of V2X traffic. The protection offered by the packet filters and the V2X Application-Level Gateway covers DoS detection on the respective OSI layers. Such security gateway solutions, while novel in the domain of automotive security, are established concepts in the classical IT security domain [15].

2.3 Application Layer Protocols used in V2X Communication and related Attacks

With the integration of modern vehicles into the IoT the set of application layer protocols used for V2X communication significantly increases, encompassing not only protocols

from the automotive domain, but also typical IoT protocols like MQTT and traditional Web protocols like HTTP(S) or DNS. The use of these protocols in V2X communication not only allows a new level of interconnection of vehicles offering new functionality, but also comes with new security risks, since vehicles become potential targets for a number of known attacks like cross-site scripting or SQL injections.
This section gives an overview of (mostly IP-based) application layer protocols that can be used in V2X communication, including representatives of the automotive domain, typical IoT protocols and the traditional Web and attacks associated with these protocols. While the focus in this work is on application layer protocols, it has to be kept in mind, that for a complete security analysis also the lower layers of the OSI model have to be considered. E.g. most application layer protocols presented here are based on either TCP or UDP, or both, on the transport layer. Thus any service running on one of these application layer protocols can be the target of a transport layer attack like TCP SYN flooding or TCP session reset, TCP session hijacking, UDP datagram injection or UDP bomb [75]. Also, an application layer protocol may be affected by or partly reliant on another application layer protocol, so that an attack on the other protocol can affect the use of the first application layer protocol. An example is the Dynamic Host Configuration Protocol (DHCP) starvation attack [18, 75]: By flooding the local network with DHCP requests form randomly generated MAC addresses, an attacker can deplete the available pool of IP addresses in the DHCP server. Then, this attack will prevent a node from obtaining an IP address and subsequently e.g. contacting any Voice over IP (VoIP) server over the Session Initiation Protocol (SIP) [18] or performing configuration necessary for communicating over Diagnostics over IP (DoIP) [75]. This way the DHCP starvation attack can affect the use of the SIP or DoIP protocol. THE IEEE 802.1x specification provides a mechanism where a node attached to a network port must authenticate its MAC address prior to being able to transmit or receive traffic on the network. This includes the DHCP request, thus preventing the attack [18].
The following application layer protocols have been identified for direct use in V2X communication:

- ETSI Cooperative Awareness Basic Service
- DoIP (Diagnostics over IP) [30]
- SOME/IP (Scalable service-Oriented MiddlEware over IP) [107]
- MQTT (Message Queuing Telemetry Transport) [84]

- AMQP (Advanced Message Queuing Protocol) [4]
- RTP (Real-time Transport Protocol)
- SIP (Session Initiation Protocol)
- HTTP(S) (Hypertext Transfer Protocol (Secure))
- DNS (Domain Name System)
- SSH (Secure Shell)

DoIP and SOME/IP are protocols from the automotive domain. While DoIP focuses on the purpose of vehicle diagnostics, SOME/IP aims to provide a general scalable mechanism for remote procedure calls and event notifications in a service-oriented context, fulfilling the requirements regarding resource consumption in embedded systems. SOME/IP supports both request-response and publish-subscribe messaging. A service has a unique ID and every one of its methods, that can be called via remote procedure call, has a method ID. Combined, the IDs identify a remote procedure call to a method of a service. A client using a service also has an ID. SOME/IP messages consist of a header, containing the IDs and the length of the message among others and the payload.
MQTT and AMQP are both broker-based public-subscribe messaging protocols used in the IoT. In MQTT a publisher can publish messages with "topics" via a broker. A topic is a hierarchical structured string (e.g. "*home/room1/temperature*" to refer to the temperature in the room). A publisher connects to the broker and then sends its messages to the broker. A subscriber can connect to the broker and subscribe to a topic and the broker then delivers the published messages of this topic to the subscriber. Any client can publish and subscribe to any topic [54]. There are 3 quality of service classes: 1) "at most once" (*QoS 0*), where the message is sent only once and the client and broker take no additional steps to acknowledge delivery ("fire and forget"), 2) "at least once" (*QoS 1*), where the message is re-tried by the sender multiple times until acknowledgement is received (acknowledged delivery) and 3) "exactly once" (*QoS 2*), where the sender and receiver engage in a two-level handshake to ensure only one copy of the message is received (assured delivery). Similarly in AMQP a client can send a message via a publisher to a broker, which stores the message in a queue. A subscriber receives the message from the queue and delivers it to the receiving client. With routing keys it is possible to send a message to exactly one receiver.

RTP and SIP are protocols used in the context of real-time multimedia transmission like e.g. ”Voice over IP” or video streaming. The data are transported over RTP, while SIP performs all steps necessary before the exchange of data via RTP between two (or more) communication partners can take place, like establishing the connection. E.g. in the case of using RTP for ”Voice over IP” the call setup is done with SIP [18].
HTTP, DNS and SSH are well known application protocols from the traditional Web. Simply put, HTTP is the protocol used for application data transfer in the Web, DNS translates domain names to IP addresses and SSH realises secure connections to remote network peers.
Various attacks on these protocols exist. These attacks differ in their aim, sophistication and feasibility and thus have different potentials. There are attacks on the cryptographic security, i.e. confidentiality, integrity and authenticity, like spoofing and eavesdropping, DoS attacks, side channel attacks, which are based on information that is gained from the physical implementation of a system, e.g. timing analysis, acoustic analysis or power consumption analysis [89] and attacks generally based on malicious messages, e.g. buffer overflow attacks, malformed messages deviating from the structure defined by the protocol, messages violating the protocol sequence, or messages with malicious content, e.g. SQL injection. Less obvious examples of messages with malicious content are semantically incorrect, but otherwise normal messages, e.g. a message containing credit card data with a value of ”13” for the field ”month”.
The semantic correctness of a message can also depend on the current context, e.g. the message of a remote vehicle trunk control to open the trunk is semantically incorrect when the vehicle is driving. A malicious message can either be created by an attacker who is communicating with the target directly, or be a legitimate message intercepted by the attacker and reused either without modification (replay attacks) or modified by him. The aim of attacks based on malicious messages ranges from triggering bugs in the targeted application to gaining unauthorised access to resources or taking control of the target device. Attacks on integrity and authenticity have a similar aim, while the immediate goal of eavesdropping and side channel attacks is illegitimately gaining information. With DoS attacks the availability of a system is targeted.
The potential of a concrete attack is never defined by the attack itself alone, but also by the ”value” of a target, e.g. a DoS attack targeting an ECU of the infotainment system has less potential than a DoS attack targeting a safety-critical component. Also, the effort of implementing countermeasures differs: e.g. for buffer overflow attacks it is sufficient to check the size of the message and drop messages with an inappropriate size, while e.g. detecting messages with malicious content is a more complex task. Generally,

all these attacks have to be considered when using the application protocols listed above in V2X communication.
Table 2.1 gives an overview of the protocols and the related attacks:

Table 2.1: Overview of V2X application-layer protocols and related attacks

Protocol	**Environment**	**Protocol-Stack**	**Attacks**
HTTPS	Web	TCP/IP	malicious messages, DoS attacks
DNS	Web	TCP or UDP/IP	spoofing, malicious messages, DoS attack via DNS replies, DNS re-binding
SSH	Web	TCP/IP	brute force attacks, side channel attacks
RTP	Multimedia	UDP/IP	eavesdropping, malicious messages (DoS)
SIP	Multimedia	TCP or UDP/IP	spoofing, eavesdropping, malicious messages, DoS attacks
MQTT	IoT	TCP/IP	spoofing, eavesdropping, malicious messages, access control attacks via topics, DoS attacks
AMQP	IoT	TCP (or UDP)/IP	malicious messages, DoS attacks
Some/IP	Automotive	TCP or UDP/IP	spoofing, eavesdropping, malicious messages (except buffer overflow attacks), DoS attacks
DoIP	Automotive	TCP or UDP/IP	spoofing, malicious messages (except buffer overflow attacks), DoS attacks
Cooperative Awareness Basic Service	Automotive	WSMP/ETSI	malicious messages, DoS attacks

Since the ETSI Cooperative Awareness Basic Service is transported over WSMP, which specifies means ensuring cryptographic security, attacks on cryptographic security practically do not apply to the ETSI Cooperative Awareness Basic Service. However attacks based on malicious messages and DoS attacks are possible. While malformed messages or buffer overflow attacks can be easily detected, due to the V2V context in which ETSI Cooperative Awareness Basic Service is used, it is particularly vulnerable to messages semantically incorrect in the current context, i.e. messages with false information regarding a vehicle, like a false vehicle position or acceleration. To detect such messages a context-sensitive semantic analysis is required.
For DoIP an extensive security analysis can be found in [75]. To summarise, a lack of authentication allows spoofing and a weak data integrity check allows the unauthorised

modification of messages by an attacker. The proposed solution is to use cryptography to ensure integrity and authenticity. Also, the DoIP specification prescribes a check of the payload size to protect against buffer overflow attacks.
For SOME/IP [57] lists the following threats: spoofing due to a lack of authentication, protocol violations, e.g. sending multiple responses to a request which should be answered with only one response and packets modified by an attacker due to a lack of integrity e.g. triggering bugs. Additionally, for use cases with hard real-time traffic (e.g. cyclical safety messages every 100 ms) timing issues (e.g. deviations from defined cycle times) have to be considered, however, this is relevant more for communication in the in-vehicle network and less for V2X communication. Due to a lack of confidentiality, eavesdropping is also possible. The proposed solutions is the use of cryptography to ensure confidentiality, integrity and authenticity and a network IDS able to monitor all traffic in the network to detect protocol violations, timing issues and malformed packets [57]. For the detection of malformed messages and many protocol violations and timing issues, a host IDS would be sufficient. The SOME/IP specification also prescribes a check of the payload size to protect against buffer overflow attacks [107]. Also, for both DoIP and SOME/IP DoS attacks are imaginable, though neither protocol facilitates DoS attacks and for any attacker communicating with a target over DoIP or SOME/IP it is possible to create and send malicious messages.
Since in MQTT any client can publish and subscribe to any topic, access control attacks via topics, where an attacker gains unauthorised access to resources by subscribing or publishing to a certain topic are possible [54, 41, 23]. In the absence of authentication, attackers can pose as the MQTT broker realising man-in-the-middle attacks [55]. Eavesdropping, attacks based on malicious messages and buffer overflow attacks are also possible. The proposed solutions are the authentication of publishers, subscribers and the broker and Access Control Lists (ACLs) specifying which users can publish/subscribe to which topics [54, 28, 23]. To prevent eavesdropping and message modification, MQTT would also have to offer confidentiality and integrity. The specification of MQTT version 5.0 states that MQTT publishers and subscribers should offer authentication, authorisation, integrity and confidentiality [9]. One recommended option for offering confidentiality, integrity and authentication is TLS. With AMQP, buffer overflow attacks and attacks based on malicious messages are possible. As for cryptographic security, encryption and authentication is provided by TLS [80]. Also, for both MQTT and AMQP DoS attacks are possible. DoS attacks could target either the broker or one or more clients, with the broker probably being the preferred target due to its central role. In the case of MQTT, messages from the quality of service class ”exactly once” are preferred for

DoS attacks due to the involved four-part handshake on the application layer [23, 41]. Such DoS attacks can be detected with an IDS [53]. In the context of DoS attacks, the automatic recover-ability from the DoS attack, the time taken for recovering and the impact of broker failure (if it was the target of the DoS attack) are also significant security concerns [53].
With RTP, due to a lack of cryptographic security, eavesdropping and the modification and injection of packets is possible. Since RTP is used for the transport of audio or video data, the most feasible attack with inserting or modifying packets is a DoS attack, where the insertion or modification of packets causes a significant degradation of the quality of the multimedia transmission [42, 2]. Buffer overflow attacks are also imaginable. One proposed solution is the Secure Real-time Transport Protocol (SRTP), which provides confidentiality, message authentication, and replay protection for RTP traffic [42]. However, SRTP incurs an additional overhead to verify the HMAC-SHA1 message authentication code for each packet. Therefore [42] propose SRTP+, which significantly decreases the verification overhead compared to SRTP and thereby increases the number of faked packets required to mount a successful DoS attack.
For SIP [18, 35] list the following threats: due to a lack of cryptographic security, spoofing is possible, allowing SIP registration hijacking. A user has to register itself with a SIP proxy, which allows the proxy to direct inbound calls to the user. An attacker can impersonate a user to a SIP proxy and replace the legitimate registration with its own address, causing the inbound calls to be sent to the attacker. Also, eavesdropping, malicious messages, replay attacks and buffer overflow attacks are possible. Known attacks based on malicious messages are the Cancel- or Bye-Attack, where an attacker sends the ”Cancel” or ”Bye” command in the payload of a SIP message to a user to terminate an ongoing conversation. Traditional DoS attacks by overwhelming the target with messages are also imaginable. The proposed solution is the use of cryptography (e.g. TLS) to ensure confidentiality, integrity and authenticity [18, 35].
Most attacks on HTTP(S) are based on malicious messages. Prominent examples are SQL injection or Cross-site scripting (XSS) [8, 102, 55]. Malicious content, e.g. an SQL command in the case of SQL injection or a script in the case of XSS, is injected into the HTTP(S) packet, e.g. into the URL or payload. Another known attack is HTTP request smuggling [56], where HTTP header fields declaring the payload length are manipulated and the payload contains a smuggled malicious HTTP request. Depending on the application attacked via HTTP(S) malicious HTTP(S) messages can have various effects. If e.g. access control is realised via the URL (e.g. resources of *user1* are accessed via ”*.../user1/...*”), a modification of the URL allows access control attacks. Deviations from

designated data types or patterns in the payload can possibly trigger bugs, e.g. if an application handling credit card data receives a payload where the field "month" is not an integer in the range of 1 to 12. Buffer overflow [71] and DoS attacks are also possible in HTTP(S).
With DNS, due to a lack of authentication and integrity, spoofing and packet manipulation are possible, which is why DNSSEC was introduced [6]. DNSSEC secures DNS data by providing data origin authentication and integrity by using digital signatures [6]. Still, DNS servers can be used as amplifying reflectors in DoS attacks [61]. An attacker sends DNS queries with a spoofed address, i.e. the address of the victim, which is then overwhelmed by DNS replies. Another known attack is DNS rebinding [66]. An attacker owning a domain name, e.g. "*attacker.com*", first needs to attract traffic to this domain. The DNS queries for "*attacker.com*" are answered with the IP address of the attackers own server and a short Time-To-Live (TTL). The users' browsers issue HTTP requests to "*attacker.com*" and receive a malicious HTML document. This HTML document then issues a second HTTP request to "*attacker.com*". The user's DNS cache has expired (because of the short TTL), causing the browser to issue another DNS query for "*attacker.com*". This time,the attacker's DNS server responds with the IP address of a target server. The browser allows the attacker's script to read HTTP responses from the target server because the two connections share a single host name and therefore belong to the same browser security origin [66]. Countermeasures are preventing malicious Web sites from obtaining socket-level access to arbitrary IP addresses by appropriate socket access policies of browser plug-ins and preventing the resolving of external host names to internal IP addresses, so that "*attacker.com*" cannot be resolved to internal IP addresses of target servers, e.g. protected by a firewall [66]. Attacks based on malicious messages are also possible with DNS.
Known attacks against SSH are brute force attacks on SSH passwords, trying different character combinations, e.g. dictionary attacks trying every word from a word-list or dictionary as password [92, 43] and side channel attacks [89]. In the case of SSH a known side channel attack is a timing attack where the attacker is deducing information like e.g. password length from the statistical analysis of timing behaviour of encrypted packets in SSH [108, 89]. However, the feasibility of such attacks is disputed [59, 89].

2.4 Semantic Analysis

The term ”semantic analysis” in literature is used in different contexts. These contexts can generally be classified as video or image analysis, e.g. sports video analysis [33] or cartographic image analysis [74], or text-based analysis, e.g. Latent Semantic Analysis [25, 40, 58].
The semantic analysis of videos or images is generally a process beginning with pre-processing to reduce the dimensionality and perform a segmentation of the input data [33, 74], in the case of a single image called ”tresholding”, e.g. reducing a coloured image to black and white. Next, single features are extracted, e.g. in case of a cartographic image this can be a coastline or main road or in case of a sports video (frame) e.g. an active region or dominant colour with which the frame can be classified. Based on these features the semantic analysis can be performed.
The general idea behind text-based semantic analysis, like Latent Semantic Analysis, is a statistical analysis based on a so called co-occurrence table or term-document matrix. For n given documents and m given terms a co-occurrence table, or term-document matrix, is created where every element $s(m_i, n_j)$ denotes how often the term m_i occurs in document n_j. Based on this, the semantic analysis of the terms is performed. An example using this idea is the auto-debugging method presented in [87]. In general, to debug a program consisting of n lines of code, the program is executed e.g. 10 times and e.g. 4 times the result is not correct. Then for every line of code i it is checked in how many of the 4 faulty runs it was called. If e.g. line 5 was called in each of the 4 faulty runs and line 8 was called in only 2 of the 4 faulty runs, line 5 is classified as more suspicious than line 8. This way the lines of code that most probably are connected to a software bug causing an error are automatically identified.
In this work the semantic analysis of data is generally defined as checking the conformity of syntactically correct data by applying a set of semantic rules, see chapter 4, section 4.2. The data are classified as semantically correct if they are in conformity with the given semantic rule set. This work defines 3 types of semantic analysis:

- Stateless semantic analysis: the result of semantic analysis, i.e. the semantic correctness, is independent of the state of the analysed system, e.g. the value of ”13” for the field ”month” is invalid regardless of the state of the system.
- Stateful semantic analysis: the result, i.e. semantic correctness, depends on the state of the analysed system, e.g. the ”open” command of a remote trunk control is invalid when the trunk is locked, but valid when the trunk is unlocked.

- Context-sensitive semantic analysis: the result, i.e. semantic correctness, depends on the context the analysed system operates in, e.g. in case of the remote trunk control the ”open” command is invalid regardless of the state of the trunk when the vehicle is driving. Here, the vehicle is the context in which the analysed system, the trunk, operates. Technically the context-sensitive semantic analysis can be seen as equivalent to the stateful semantic analysis, since in the end both the analysed system's state and the state of the context amount to ”a state” on which the semantic correctness depends. But to emphasise the fact of the external dependencies when a system is dependent on the context it operates in, which in the automotive domain can be e.g. another component or multiple other components of the vehicle, the abstract state of the vehicle (e.g. ”driving” or ”stopped”), the conditions on the road (like the weather or traffic), etc., this work distinguishes between ”stateful”- and ”context-sensitive (stateful)” semantic analysis.

The aim of semantic analysis in this work is to increase the safety of a remotely controlled system by limiting the possible behaviour of the system to safe behaviour. So even when cryptographic security is compromised and an attacker can influence the system's behaviour by sending commands, the potential damage is limited. E.g. with a semantic analysis of commands to remotely control a vehicle trunk, it becomes impossible to open the trunk via remote command while the vehicle is driving. V2V messages, e.g. containing the positions of nearby vehicles, could also be viewed as a form of ”imlicit remote control”, since they are messages from a remote sender affecting the receiving vehicle's behaviour, or at least they contribute to the vehicle's decision on how to react in a situation. E.g. messages containing the position of the preceding vehicle will contribute to the receiving vehicle's decision on how to behave (slow down, stop, etc.), which will also be influenced by data from a range of sensors. The semantic analysis of a system's behaviour can possibly enhance safety in any scenario where machines interact with humans, or with each other, e.g. in industrial plants, etc. It does not have to be limited to a system's external communication, but can also be realised for internal system control, in case of vehicles, in the in-vehicle network.

3 Analysis

This chapter gives an overview of related work covering security gateways and security proxies. With the increasing interconnection modern vehicles are undergoing the same development as manufacturing is with its concept of "Industry 4.0", which results in an integration of both in the "Internet of Things" (IoT). Thus not only work from the automotive domain is presented, but also work from the industrial domain, the IoT and the traditional Web. With modern vehicles being integrated into the IoT, the traditional distinction between the automotive and IoT domain may become less reasonable, with the automotive domain becoming a sub-domain of the IoT. But since special properties and constraints apply to the automotive domain compared with other IoT devices, e.g. safety- and hard real-time requirements and a wider range of functional domains encompassing infotainment, engine control and safety electronics, it remains a distinctive sub-domain and thus is treated separately in this work. Also, this chapter presents the requirements analysis considering the application protocols and related attacks from chapter 2, from which the concept of the V2X Application-Level Gateway is derived.

3.1 Related Work

3.1.1 Automotive Security Gateways

Most of the work covering automotive security gateways, focuses exclusively on securing the communication of the in-vehicle network, like the work of Pese et al. [95]. In [95] Pese et al. present the concept and prototype implementation of an automotive firewall to prevent inter-domain attacks in an Ethernet-based in-vehicle network with a domain architecture. In a domain architecture the network is divided into several domains like infotainment, engine control etc. and each domain is connected to the rest of the network via a domain controller, which separates its domain from the rest of the network. The domain controllers are connected via an Ethernet-backbone. The automotive firewall is

to prevent attacks from one domain against devices of another domain. It consists of a stateless packet filter which filters the Ethernet-traffic according to configured rules and a stateful packet filter which filters the IP/TCP- and UDP-traffic according to configured rules. The stateless packet filter is implemented in hardware for performance reasons, while the more complex stateful packet filter is implemented in software. The stateless packet filter is located between the network domain segments controlling all inter-domain Ethernet-traffic. The stateful packet filter is implemented on every domain controller filtering incoming traffic that passed the stateless packet filter. The automotive firewall offers security on the link- and transport layer. To additionally offer security on the internet-layer the filtering of IP-traffic according to configured rules could also be implemented in a stateless packet filter. Whereas [95] can offer security on the link-, network- and transport layer, the V2X Application-Level Gateway proposed by this work secures the application layer and thus is complementary to packet filter solutions like [95]. Combined they offer comprehensive security of V2X communication covering all relevant OSI layers. In [60] an automotive IDS primarily securing the in-vehicle network is proposed, but [60] also state that the IDS should detect the leakage of sensitive information via V2X. This requires the IDS to check if V2X messages contain sensitive data, which can be done with a semantic analysis of application data to determine if they are sensitive. In [109] propose context-awareness or context-sensitivity as a useful feature of future automotive intrusion detection systems. In such an IDS anomaly detection would be context-sensitive, i.e. dependent on the current state of the vehicle. E.g. for classifying a message as normal or anomalous the vehicle's current state (e.g. *parked*, *driving*, *crashed*) would be considered. The functionality described by [60] and [109], i.e. semantic analysis and context-awareness, are key aspects and functional requirements for the V2X Application-Level Gateway proposed by this work.

Of the work covering V2X security most focus on the realisation of cryptographic security of V2X communication [115], e.g. in the context of ECU software updates [62], or on security issues in VANETs like authentication [19, 51, 46], Denial-of-Service (DoS) attacks [11, 117] or misbehaviour detection [100, 44] in VANETs. Misbehaviour detection here means the detection of network nodes spreading false information in the network due to either malfunction or malicious intent, in the context of VANETs e.g. falsely reporting a traffic accident. Few papers focus on securing V2X communication from the perspective of a vehicle built-in security gateway, which offers more functionality than decrypting and encrypting V2X traffic and checking certificates and signatures. One of them is the work of Bouard et al. [13] presenting a proxy securing the communication between CE (Consumer Electronics) devices and ECUs of the in-vehicle network, which is realised

via the proxy. The purpose of this security proxy is decoupling the CE devices from the ECUs and ensuring that only authorised devices can communicate with the ECUs. The security proxy communicates via a secure IP-based middleware, e.g. SEIS [45], with the ECUs, only passing on messages from authenticated CE devices to them. Like [13] the V2X Application-Level Gateway proposed by this work is a proxy decoupling ECUs of the in-vehicle network from external devices. Unlike [13] it is not focusing on CE devices specifically and provides extended functionality like context-sensitive semantic analysis or DoS detection. In [117] Yang et al. describe an intrusion detection system (IDS) based on machine learning for V2X communication to detect DoS attacks, port scans, brute force attacks (presumably on cryptographic security) and some web-based attacks (SQL injection, cross-site scripting (XSS)). The use of different machine learning algorithms is evaluated. It was shown that generally with tree-based algorithms a higher accuracy and detection rate was achieved than with K-nearest neighbour and support vector machine approaches. The application layer security functionality from [117] targets some attacks identified to be prevented by the V2X Application-Level Gateway proposed by this work, e.g. web-based attacks, and thus could be incorporated into the V2X Application-Level Gateway.
The contribution of this work is a V2X Application-Level Gateway for securing V2X communication. In addition to ensuring cryptographic security and offering proxy functionality decoupling in-vehicle ECUs from the outside world, it secures V2X communication on the application layer, including a context-sensitive semantic analysis of application data and detecting application-layer DoS attacks. It also uses a role-based access approach with ACLs to deny unauthorised access to in-vehicle resources via V2X and allows bandwidth control of V2X traffic.

3.1.2 IoT and Industrial Security Gateways and Proxies

When it comes to gateways or proxies in the IoT or the industrial domain the focus has often been on protocol translation ensuring the interoperability of heterogeneous devices or systems. Also, gateways for coordination or data integration, like a gateway for making vehicle sensor data available for processing in the cloud [67] are covered. Increasingly however, the issue of security is also being addressed. An example is the IoT gateway presented in [90] using TLS for cryptographic security. With the use of TLS the data is encrypted and an authentication of peers is possible. The IoT security proxy presented in [17] uses symmetric-key algorithms for the encryption of data exchanged between the

proxy and other devices and for the authentication of these devices thereby addressing cryptographic security. Symmetric-key algorithms are used for simplicity and performance reasons. Additionally the proxy uses Access Control Lists (ACLs), which specify the set of operations that each group is allowed to perform. To perform an operation protected by an ACL, a requester must include a certificate in his request, proving he is a member of a group allowed to perform that operation. The industrial security proxy described in [114] also restricts access to resources, e.g. an operation, using a role-based approach. It also implements a rudimentary form of semantic analysis of application data: if a command is sent to a device via the security proxy, it checks if that command is in the set of commands that the device is able to execute at all and if not, drops the invalid command.

Another useful feature is bandwidth control, e.g. implemented in the proxy described in [116]. In this case bandwidth is managed by arranging all network streams in a so called stream hierarchy, which is represented by a graph. Network streams are represented by the leaf nodes of this graph. The internal nodes implement a certain bandwidth distribution technique, e.g. "Mutex", which ensures that at all times at most one of its children is assigned with bandwidth. Other distribution techniques are "Priority", assigning bandwidth according to the priorities of its children and "Weight", assigning bandwidth by distributing the available bandwidth between all its children according to defined weights of streams. Another feature that can enhance security is virtualisation. The IoT gateway "LEGIoT" presented in [82] is using Docker [29] for the virtualisation of all its components via Docker containers. This allows a fast building process, instantiation, easy management and isolation of components giving the system flexibility. Virtualisation comes at the cost of increased resource demand, e.g. memory or CPU performance, in the case of [82] to run the Docker Engine realising the virtualisation on the system.

Yet another interesting feature is context-awareness like in the IoT-eHealth gateway proposed by [3]. The definitions of "context" and "context-awareness" used by [3] are from [1], with context being any information that can be used to characterise the situation of an entity (e.g. an object) and context-awareness being the use of context (e.g. by an application) to provide relevant information or service. In case of the V2X Application-Level Gateway the vehicle's state and the environment could be context.

An example of using learning-based anomaly-detection in an IoT security gateway is [88], where the gateway is part of a distributed system utilising federated learning. Each security gateway acts as a local access gateway to the Internet for a number of IoT devices. It monitors the communication of the IoT devices and detects anomalies based on anomaly detection models it trains locally. The local models are aggregated to a global

detection model. Due to the diversity of IoT devices the models are device-type-specific (e.g. camera, smart plug, smart coffee machine). A general criterion for classifying IoT devices is the complexity and variance of their network traffic [48].
The above mentioned security measures from the IoT and industrial domain are applicable in the context of an automotive V2X Application-Level Gateway. Summing up, that is: proxy-functionality, cryptographic security and application layer security including the semantic analysis of application data, ACLs realising role-based access to resources, a context-awareness of the system, which is useful for the semantic analysis of application data, bandwidth control, anomaly-detection and virtualisation.

3.1.3 Web Security Gateways and Proxies

In general, web security gateways and web security proxies offer similar functionality to that of IoT or industrial security gateways and proxies: the proxy-functionality, encryption, application layer security [69, 14], which in this case is limited to Web protocols like HTTP [77], bandwidth control [10] and the semantic analysis of application data [104]. The web security proxy presented in [104] analyses the application data and classifies it as either valid or invalid according to predefined rules, e.g. "the data-type must be *int*". The set of rules can easily be extended. Only if the data is valid the HTTP request or response is forwarded. An additional useful feature of web security gateways and proxies, is logging [77]. Since modern vehicles are both consumers and providers of web services, the above mentioned functionality is applicable in the context of an automotive V2X Application-Level Gateway.

3.2 Requirements

For the V2X Application-Level Gateway the requirements result from its task to secure the V2X communication. A central aspect is securing applications from the threats presented in chapter 2, section 2.3. The first step is to identify which of these threats can be addressed in a V2X Application-Level Gateway and which should to be addressed elsewhere. E.g. transport layer attacks such as TCP SYN flooding or UDP datagram injection are addressed in packet filters securing the communication on the network- and transport layer and not in ALGs securing communication on the application layer complementary to such packet filters. Since side channel attacks are based on information that

is gained from the physical implementation of a system, e.g. timing analysis, acoustic analysis or power consumption analysis, it makes little sense to address them in an ALG. Also, application layer attacks that affect V2X communication by targeting resources not protected by the V2X Application-Level Gateway can hardly be addressed by it, e.g. DHCP starvation attacks exhausting available IP addresses to prevent IP-based V2X communication. Since DDoS attacks are countered or mitigated in the network and not by a single network node, they cannot be addressed effectively in the V2X Application-Level Gateway. When it comes to DoS attacks, the V2X Application-Level Gateway could protect the in-vehicle network from application layer DoS attacks via V2X. DoS attacks on the lower OSI layers, e.g. TCP SYN flooding, are addressed in components protecting the link-, internet- and transport layer.

Protection from application layer DoS attacks can be realised by identifying and dropping DoS traffic. The identification of DoS traffic can either be based on machine learning or configurable static rules. In both cases the basic principle is the same: when one (or more) characteristic (e.g. requests per second) of the traffic from a source surpasses a certain threshold it is identified as DoS traffic and dropped. To effectively protect from DoS attacks on the application layer the following questions have to be answered first: Which of the in-vehicle services are threatened? And what is their specification with respect to DoS detection, i.e. how regular or irregular is the communication? For services with uniform communication patterns configurable static rules would be sufficient for DoS detection, while services with diverse communication patterns require the use of learning-based DoS detection. Although simple threshold-based techniques are prone to incorrectly classifying normal traffic as anomalous traffic and are unable to adapt to the evolving nature of attacks [22] and more sophisticated anomaly detection algorithms, particularly those using machine learning, can help minimise false positives [32], in the automotive domain simple uses cases exist, for which simple threshold-based techniques are sufficient. An example would be locking and unlocking the doors or opening and closing the vehicle trunk via V2X. In both cases, where a panicked (or very impatient) user pressing the button multiple times in a second is the most extreme valid scenario, any number of requests beyond a few per second can generally safely be classified as a DoS attack (or a malfunction).

In-vehicle services directly threatened by application layer DoS attacks via V2X are all services offering some functionality to clients via V2X communication, i.e. services being a server reachable via the internet (or a VANET). In-vehicle clients using V2X can be targeted by DoS attacks if the external server they communicate with is compromised and used to launch a DoS attack against its clients. If an attacker manages to overwhelm

the entire ECU of the targeted V2X service with his DoS attack or even whole parts of the in-vehicle network, he can also affect other services, even those not using V2X. This would make any service of the in-vehicle network potentially vulnerable to V2X application layer DoS attacks. So by identifying and dropping DoS traffic targeting V2X services, the entire vehicle network is protected.
DoS traffic is identified based on network flow statistics like bandwidth and packet inter-arrival time [32, 88] and message features like packet size and message type (see chapter 2, section 2.1.3) [76, 53, 32, 88, 117]. The most effective application layer DoS attack against an application would be one tailored to the application using simple as possible requests asking the application to perform the most complex and, in terms of resources, costly operation possible. Such attacks could be easier detected if detection too would be tailored to the application. The operations of an application would have to be classified according to their costs in terms of resources. Then, additionally to the above mentioned features like packet size and packet inter-arrival time, the resource cost of the traffic could be used for detection. However such a classification requires that all operations of an application can be assigned to a distinct class, which may not always be possible for an application. Also, for the resource cost to be used as additional feature for DoS detection, it is necessary for an application that normal traffic does follow a certain pattern in respect to resource cost which is distinguishable from DoS traffic. E.g. the legitimate traffic of a service offering access to multimedia may predominantly consist of messages requesting video data, which would be classified as the highest cost operations of the application. In this case an application layer DoS attack would not differ from legitimate traffic in terms of resource cost and the feature resource cost would offer no improvement to a detection based on network flow statistics and simple message features like packet size. Another aspect is scalability: in the context of DoS detection e.g. [32] states that ”in order for an algorithm to scale to high bandwidth application, a given algorithm must rely on network flow statistics (how packets are sent) as opposed to deep packet inspection (what is in a packet)”. Any detection using the resource cost of traffic as a feature, would have to scale to a sufficient bandwidth.
To reduce the feasibility of DoS attacks the *Hashcash* mechanism, see chapter 2, section 2.1.3, is a possible option, however the reduction of performance due to *Hashcash* has to be taken into account. Similarly, the V2X Application-Level Gateway could also detect brute force attacks and reduce their feasibility.
Many threats presented in chapter 2, section 2.3 stem from a lack of cryptographic security, e.g. eavesdropping, spoofing or message modification. Therefore, providing cryptographic security is a key requirement for the V2X Application-Level Gateway. Also,

application layer security beyond protection from application layer DoS attacks has to be provided: attacks based on malicious messages must be addressed, which is essential if cryptographic security has been compromised or in the absence of cryptographic security. Attacks based on malicious messages can be classified as:

- buffer overflow attacks
- malformed messages (deviating from the structure defined by the protocol)
- messages violating the protocol message sequence
- messages with malicious content (e.g. SQL injection)

This work defines 2 classes of malicious content:

- ”explicitly malicious content”: e.g. SQL injection or malware, which is solely used in and specially developed for attacks
- malicious semantics: meaning content ”normal in itself” but malicious in the concrete context, like e.g. the value ”13” for a field ”month” or the ”open” command for the trunk of a driving vehicle

The V2X Application-Level Gateway can react in several ways to a malicious message: it can silently drop it, it can drop it and report the attack e.g. to a security centre, or it can notify another component to take action.
Application layer security can be divided into 2 dimensions: the application layer protocol and the application data. With respect to application layer security the application layer data is independent from the application layer protocol, since it can be transported by any application layer protocol and thus the two can be viewed separately. For the application layer protocol it has to be checked if the header violates the protocol or, if possible, contains malicious content, such as e.g. an SQL injection in an HTTP(S) URL. For deviations from the defined structure of the header, protocol-specific stateless checks are sufficient, while violation of the message sequence, e.g. receiving a ”CONNACK” without a prior ”CONNECT” in MQTT, requires protocol-specific stateful checks. For application data it has to be checked if it follows the defined structure and contains no malicious content, such as SQL injections. In most cases of ”explicitly malicious content” stateless checks using regular expressions are sufficient. Additionally the semantics of the application data have to be checked, since an otherwise correct message can contain semantically incorrect data, like e.g. the value ”13” for the field ”month” of credit card data. This may require either stateless or stateful checks. As the correctness of the

semantics of a payload may depend on the context a system operates in, e.g. in case of a remote vehicle trunk control, the check has to be context-sensitive when necessary. Buffer overflow attacks can be easily addressed by checking the sizes of messages.
The following requirements, divided into functional- and performance requirements, have been identified after researching security gateways and proxies in the automotive-, IoT-, industrial- and web domain and analysing attacks on application protocols suitable for use in V2X communication:

Functional requirements:

1. Providing cryptographic security: ensuring confidentiality, integrity and authenticity of V2X traffic to protect the privacy of vehicle- or user-related data and the vehicle itself from attacks and manipulation. For the communication between the V2X Application-Level Gateway and the outside world stronger encryption can be used, while the communication between V2X Application-Level Gateway and vehicle internal services can be secured by weaker encryption to relieve ECUs.

2. Providing application layer security: controlling V2X data on the application layer according to predefined rules, including a context-sensitive (stateful) semantic analysis (see chapter 2. section 2.4 and chapter 4, section 4.2) and the detection of application layer protocol violations (caused by either malformed messages or message sequence violations). This requires the V2X Application-Level Gateway to be aware of all in-vehicle services using V2X to load the respective security configurations (containing the rules etc.). This enhances the protection of the privacy of vehicle- or user-related data and the vehicle itself from attacks and manipulation. It addresses threats based on malicious messages.

3. Providing proxy-functionality: serving as a proxy to vehicle-internal services communicating with the outside world to decouple the in-vehicle network from external communication partners. A service running in the in-vehicle network is reachable over the V2X Application-Level Gateway over a certain IP and port by outside world peers. The V2X Application-Level Gateway can reach this service in the internal network over another (internal) IP and port. A mapping between external and internal IP addresses and ports is required. For security reasons, to hide information from potential attackers, this mapping should be dynamic.

4. Support of IP-based application layer protocols (such HTTP(S), MQTT or SOME/IP), since most V2X traffic will probably be IP-based.

5. Support of application layer protocols (such as ETSI Cooperative Awareness Basic Service) based on V2X-specific network- and transport layer protocols (such as WSMP), since especially real-time V2V/V2I traffic will be based on a V2X-specific stack instead of the IP stack.

6. Realising role-based access to resources via Access Control Lists to prevent unauthorised access.

7. Allowing bandwidth control of V2X traffic: dividing bandwidth among applications (like [116] in section 3.1.2) to optimise vehicle performance in every situation, e.g. by prioritising critical services if necessary.

8. Detecting application layer DoS attacks via V2X, when possible dropping DoS traffic and when applicable, reducing the feasibility of DoS attacks with the *Hashcash* mechanism.

9. Detecting brute force attacks via V2X, when possible dropping brute force traffic and when applicable, reducing the feasibility of brute force attacks with the *Hashcash* mechanism.

10. Configurability of the system, e.g. updating/adding new rules for semantic analysis or bandwidth control to facilitate maintenance and ensure optimum performance over a long product life cycle.

11. Logging functionality to facilitate maintenance.

Performance requirements:

1. Hard real-time capability to support end-to-end delays in the milliseconds ($<$100 ms and for the most demanding use cases $<$10 ms) for hard real-time V2X communication such as traffic safety. The message frequency for such applications is 10 Hz [12, 78, 118]. For non safety V2X applications such as infotainment or traffic efficiency the end-to-end delay of messages is in the range of 100 ms to $>1s$ [12, 78, 118].

2. The V2X Application-Level Gateway has to handle sufficient throughput for all V2X traffic. Throughput for a V2X application is in the range of 5 Kbps to 700 Mbps, ranging from 10 to 80 Mbps for most applications [12, 78]. So for n V2X applications in a vehicle the throughput D can be described as: $D = \sum_{i=1}^{n} d_i$ with d_i

being the throughput of application i, which is in the range of 5 Kbps to 700 Mbps.

3.3 Application Layer Protocol Vulnerabilities - An Exemplary Comparison

With a list of application protocols suitable for V2X communication and related attacks addressed by the V2X Application-Level Gateway and the countermeasures to these attacks identified, the vulnerability of the different protocols to attacks is compared, to determine which attacks are universal and which are applicable only for certain protocols or under certain conditions. The focus of this work is on universal countermeasures to universal attacks instead of specialised countermeasures for specialised attacks. For the comparison HTTPS, MQTT and the ETSI Cooperative Awareness Basic Service were chosen, each representing a different domain: traditional Web, IoT and the automotive domain. Since numerous work on cryptographic security already exists, this work focuses on attacks based on malicious messages and application layer DoS attacks, while attacks on cryptographic security are not considered in the comparison. The vulnerability to the different classes of malicious messages identified in section 3.2, i.e. buffer overflow attacks, malformed messages (deviating from the structure defined by the protocol), messages violating the protocol message sequence and messages with malicious content (e.g. SQL injection or semantically incorrect payload) and to application layer DoS attacks is compared, see table 3.1, page 36. The 3 protocols are similarly vulnerable to all classes of attacks, with the exception of the ETSI Cooperative Awareness Basic Service not being vulnerable to message sequence violations, since there are no different message types and defined sequences for ETSI CAMs and HTTPS being vulnerable to message sequence violations only if a service specifies a certain sequence of HTTP messages (GET, POST, etc.). So generally all attack classes are widely applicable, with protocol message sequence violations being limited to stateful application layer protocols.
Buffer overflow attacks and application layer DoS attacks are universal attacks following the same pattern regardless of the targeted protocol or application: the first exceeds the allowed size of data transported in a message, the second exceeds the allowed amount of traffic. Thus the countermeasures are also universal: checking the size of the transported data or detecting excess traffic and reacting accordingly, e.g. dropping it. Whereas attacks based on malformed messages or malicious content and protocol message sequence

Table 3.1: Overview of selected application-layer protocols vulnerability

Attack	**HTTPS**	**MQTT**	**ETSI CAM**	**Countermeasure**
Buffer overflow	+	+	+	Message size check
Malformed message	+	+	+	Stateless structural check
Message sequence violation	-	+	- -	Stateful protocol check
Malicious content	+	+	+	Stateful/stateless content check
DoS	+	++	+	Detection, Hashcash

++ protocol facilitates the attack
+ attack is applicable
- attack is possible only under certain conditions
- - attack is not applicable

violations are specific attacks tailored to the targeted application or protocol. However the concepts on which the specific countermeasures to these attacks (e.g. filters for SQL injections) are based, are universal: like stateless regular expressions (applicable for e.g. malformed messages) or the concept of statefulness. Since numerous work on stateless filtering of data using regular expressions exists, this work focuses on the development of a concept for a stateful semantic analysis of application data as a universal countermeasure against attacks based on malicious content requiring a stateful analysis. It is examined if this concept of a stateful analysis is applicable for preventing application layer protocol message sequence violations, using MQTT as an example. Also, the detection of application layer DoS attacks is included, since they are applicable against basically any application and application layer protocol. Additionally buffer overflow attacks are covered.

4 Concept

In this chapter the concept of the V2X Application-Level Gateway based on the requirements analysis given in chapter 3 is presented. First the proposed architecture of the V2X Application-Level Gateway is presented. Next the realisation of a stateful analysis applicable for both (context-sensitive) stateful semantic analysis and application layer protocol message sequence violations is discussed. Also, application layer DoS detection and the transfer of some of the functionality of the V2X Application-Level Gateway to the cloud, where more resources like computing power and memory are available, is discussed.

4.1 Architecture

The architecture of the V2X Application-Level Gateway was developed based on the requirements identified in chapter 3, section 3.2, the concept of service-oriented communication and the general best-practice application-level gateway (ALG) software architecture described in [101]. The definition of an ALG used in [101] focuses solely on the proxy-functionality, while this work uses a definition that additionally encompasses security aspects, see chapter 2, section 2.1.2. The architecture of [101], see figure 4.1 (page 38), reifies several design patterns and is extensible. It decouples input from output (Router pattern), service initialisation from the tasks performed once the service is initialised (Acceptor and Connector patterns) and event demultiplexing and event handler dispatching from services performed in response to events (Reactor pattern). Connection requests or data, in an automotive context from either vehicle-internal services or external services, are received at the communication endpoints. In case of a connection request the *Reactor* notifies the *Acceptor*, which then establishes the connection from the service to the ALG. The *Connector* is used to proactively establish connections from the ALG to services. In case of data the *Reactor* notifies the *Input Handler*, which then receives the data, consults the *Routing Table* and requests the *Output Handler* to forward the data to the

destination. There can be multiple *Input Handlers* and *Output Handlers*, e.g. one for each connection. The *Input Handler* and *Output Handler* provide proxy functionality to services communicating via the ALG and thus the architecture meets the functional requirement *3)*, see chapter 3, section 3.2.
This base architecture was extended by several components to meet all requirements defined in chapter 3, section 3.2, see figure 4.2 (page 39). The additional components

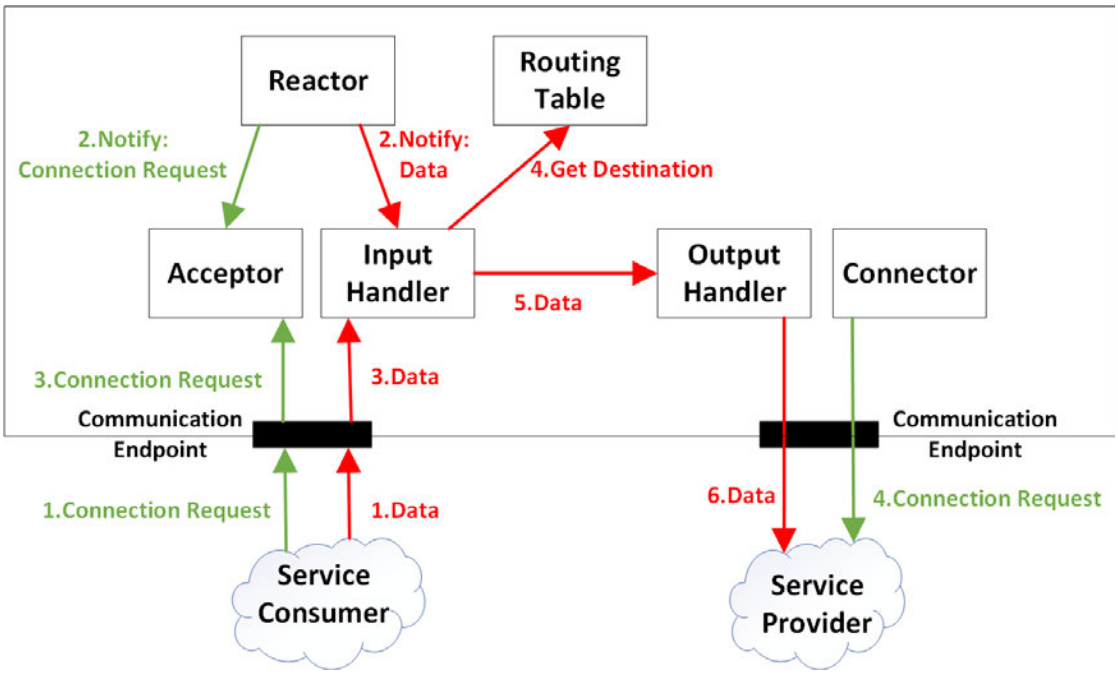

Figure 4.1: Overview: Best-practice ALG software architecture according to [101]

provide the entire security functionality, whereas the base architecture components provide the basic communication functionality. The architecture in figure 4.2 contains all components necessary to meet the requirements defined in chapter 3, but since the focus of this work is the analysis of traffic in V2X Application-Level Gateways, and for better clarity, the relationship of some components to the rest of the architecture (e.g. the *Logging Component* or the *Management Component*) is not detailed here.
The *Encryption/Decryption Components* for inbound and outbound traffic provide cryptographic security, meeting requirement *1)*. For ensuring cryptographic security, Hardware Security Modules (HSMs) [5] can be used in the V2X Application-Level Gateway. The *Access Control Component* manages the ACLs realising role-based access to resources, meeting requirement *6)*. A message is only forwarded if it is valid according to the ACLs.

An *Input Handler* and *Output Handler* form a connection between a service provider and consumer. Each connection has a service-specific *Analyzer Component* consisting of one ore more modules controlling the application data according to predefined rules, including a context-sensitive semantic analysis if necessary, which meets requirement *2)*. A message is only forwarded if it is valid. The modules can also detect application layer DoS and brute force attacks, meeting requirements *8)* and *9)*. The *Context Module* holds the context required for context-sensitive semantic analysis, see section 4.2. The *Analyzer Components* have to support all relevant IP-based application protocols, meeting requirement *4)* and all relevant non IP-based application protocols, meeting requirement *5)*. This can be achieved with exchangeable protocol-specific sub-components for the *Analyzer Component*, each supporting an application protocol, e.g. HTTP(S) or MQTT. The *Connection Manager* manages all (active) connections. The *Bandwidth*

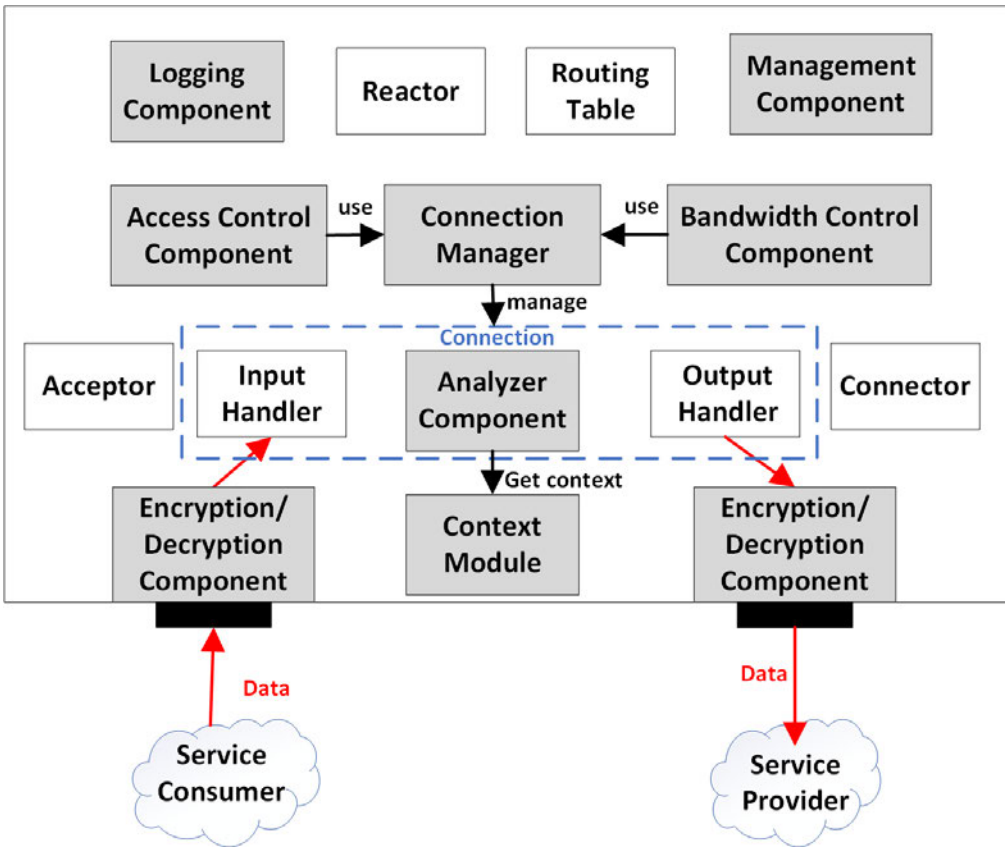

Figure 4.2: Overview: conceptual V2X Application-Level Gateway architecture

Control Component allows to manage the bandwidth of V2X traffic via the Connection Manager, according to certain distribution techniques, meeting requirement *7)*. With

the *Logging Component*, which logs the activity of the other components, requirement *11)* is met. Via the *Management Component* the *Analyzer Components*, *Context Module*, *Access Control Component*, *Bandwidth Control Component* and the *Logging Component* can be configured, meeting requirement *10)*. So the developed architecture meets all functional requirements identified in chapter 3, section 3.2.
Since each message is sequentially processed by a constant number of components and additionally a parallel processing of messages is possible with multiple parallel channels (e.g. for real-time traffic) and the input is decoupled from the output, the architecture can also meet all performance requirements defined in chapter 3, section 3.2.
As already mentioned, the V2X Application-Level Gateway has to be aware of all vehicle-internal services using V2X by having access to a central vehicle service registry containing those services . The general issues of service registration and discovery are beyond the scope of the V2X Application-Level Gateway and thus only briefly described in the context of updating its list of V2X services. Each vehicle internal V2X service provider has to register with a service registry. An external service consumer can then look up this service provider and they can communicate via the V2X Application-Level Gateway, see figure 4.3. Conversely, an external service provider registers with a service registry

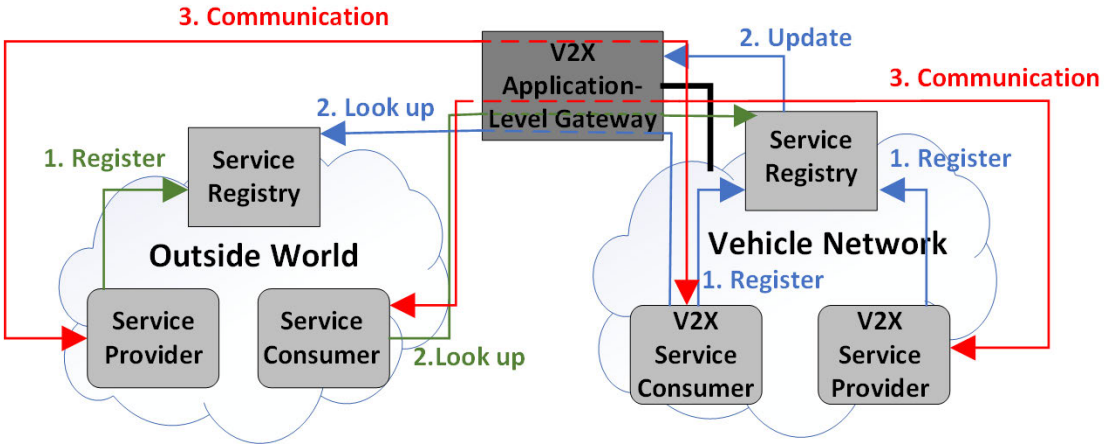

Figure 4.3: Communication between vehicle services and external services

and a vehicle internal V2X service consumer can look it up. Additionally, the vehicle internal V2X service consumer has to register with the vehicle service registry, so that with each registration of either a vehicle internal V2X service consumer or a vehicle internal V2X service provider with the vehicle service registry, the list of V2X services of the V2X Application-Level Gateway is updated so that the respective security configuration (containing the rules for analysis etc.) can be loaded.

4.2 Semantic Analysis of Application Data

4.2.1 Overview

In this work the semantic analysis of data is defined as checking the conformity of syntactically correct data by applying a set of semantic rules. The data are classified as semantically correct if they are in conformity with the given semantic rule set. This paper defines 2 classes of semantic rules: *structural rules* and *content-related rules*. Structural rules refer to properties like payload size or data type, while content-related rules refer to the application-specific meaning of data. E.g. the string "Hamburg" is a semantically correct destination for the navigation system, while the (syntactically correct) string "Asdf" is not. In this case the content-related rule is, that the string has to be in a defined set of known destinations. Another example would be a message, that is expected to contain the part of a software update. One structural rule for such a message is, that its payload size is in a specific range of bytes (for the last part of the update this range can be different, since the last part can possibly be only a few bytes). If checking the payload size finds it is only a few bytes (and it is not the last part of the update), the message will be classified as invalid and the system will react accordingly, probably dropping the message. A semantic rule can be either stateless or stateful. In the first example above, the semantic correctness is independent of the the state and thus the rule is classified as stateless. In the second example, if the specified range for the size of a message is different for the last part of the update, the semantic correctness depends on the state, i.e. if the system is in the state "receiving last part of update" or not and therefore the rule is classified as stateful.
Additionally, semantic analysis can be *context-sensitive*, i.e. depend on the current context. A general definition of context is any information that can be used to characterise the state and the environment of an entity (e.g. an object) [1]. In the case of a vehicle, the context is the vehicle's state and possibly also the state of its environment, e.g. the weather or traffic. This work distinguishes between stateful semantic analysis, where only the state of the analysed system itself is relevant as context and context-sensitive semantic analysis, where not only the analysed system itself is relevant, but also the context it operates in, e.g. in case of a vehicle the traffic or weather. The term "context-sensitive" emphasises the external dependencies of the analysed system. The context can be modelled with different degrees of complexity and levels of abstraction depending on the use-cases requiring context. For example for the modelling of a vehicle's state a simple state machine consisting of only the two states "driving" and "stopped" can be

sufficient in some cases, while others might e.g. require a modelling with distinct speeds (e.g. all speeds in a range from 0 km/h to 120 km/h with a resolution of 2 km/h).

4.2.2 Stateful Semantic Analysis

This section gives a formal definition of a stateful semantic analysis and a concrete example of context-sensitive stateful semantic analysis. Figure 4.4 shows state machines describing the correct behaviour of a vehicle's trunk and a vehicle's state. For both state machines the alphabet corresponds to a set of commands to control the system. E.g. the command to lock the trunk ("lock"), which is in the set of known commands, is semantically incorrect if the trunk is open (state "OPEN"), while the command to close it ("close") would be semantically correct. For better clarity, only the valid state transitions are depicted. All other transitions are invalid and would result in an error state. Self-transitions are not allowed, since redundant commands shall not be processed, e.g. a trunk in open state shall not process the "open" command.

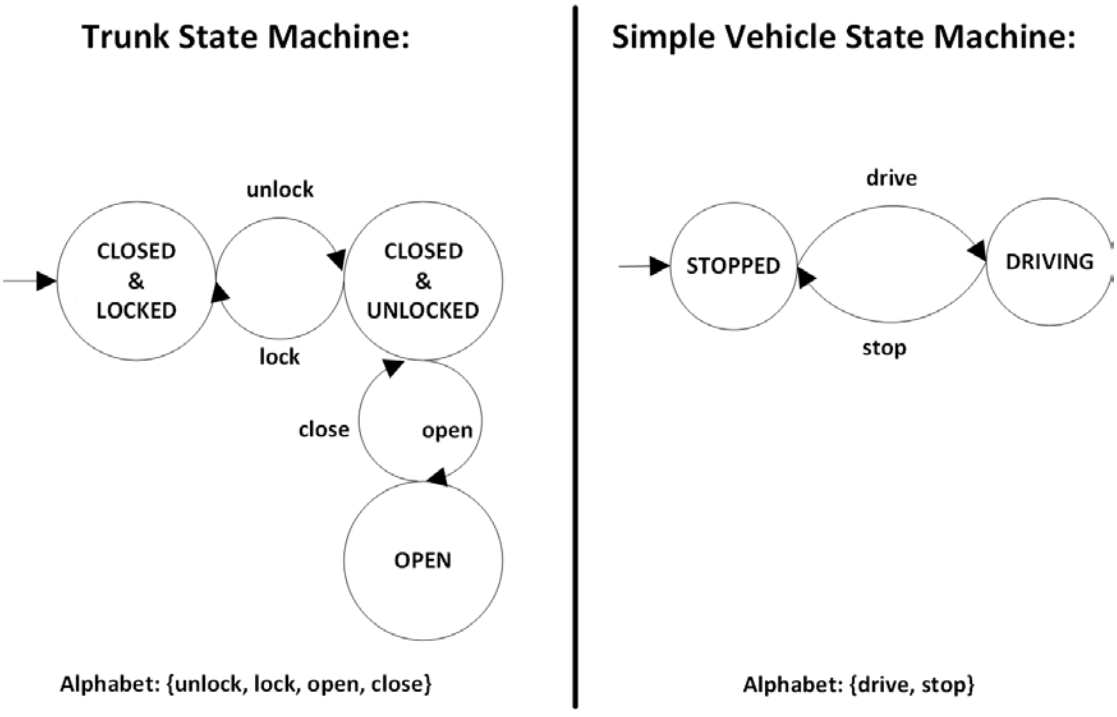

Figure 4.4: Simple state machines describing the correct behaviour of a vehicle trunk and a vehicle's state

For a semantic analysis in some cases it may not be enough to know the state of one component, e.g. the trunk, but rather the states of a composition of components or the state of the entire vehicle are required. The command to open the trunk ("open") for instance is semantically incorrect, regardless of the state of the trunk, if the vehicle is driving (state "driving").
A system (e.g. a vehicle trunk) can be modelled as a finite state machine (FSM) consisting of a set of states S (including an initial state s_0), an alphabet Σ and a transition function δ: $S \times \Sigma \to S$. In the case of the vehicle trunk the alphabet consists of events corresponding to commands such as "open" or "close" to control the trunk, see figure 4.4 (page 42). A stateful semantic analysis of such a system can be realised with a function "*valid*" mapping the system's states S and alphabet Σ on Boolean values *true* and $false$:

$$valid : S \times \Sigma \to \{true, false\} \tag{4.1}$$

For a state $s \in S$ and an event $\sigma \in \Sigma$ the *valid* function is *true* when the event σ is semantically correct in state s and $false$ otherwise, i.e. when the transition function δ returns a state $s' \in S$ for s and σ:

$$valid(s, \sigma) \iff \delta(s, \sigma) \neq \emptyset \tag{4.2}$$

E.g. $valid(open,\ close) = true$, while $valid(closed\ \&\ locked,\ open) = false$. For the vehicle trunk the *valid* function can be realised with table 4.1. For a system with n states and an alphabet size of m the table realising the *valid* function has $n * m$ entries. The time complexity of the *valid* function is $O(1)$, since it amounts to a lookup in a table.

Table 4.1: Table for the *valid* function for a vehicle trunk

State / Command	unlock	lock	open	close
Closed & Locked	*true*	*false*	*false*	*false*
Closed & Unlocked	*false*	*true*	*true*	*false*
Open	*false*	*false*	*false*	*true*

The semantic correctness of events in a system (e.g. a vehicle trunk) can depend on the state of another system (e.g. the state of the vehicle, see figure 4.4, page 42). In this case the first is called the dependent system and the latter the affecting system. The set of states of the dependent system is S_d and the alphabet is Σ_d. The set of states of the affecting system is S_a and the alphabet is Σ_a. The dependent system and the affecting system are a composition of systems. The alphabet Σ of such a composition

is $\Sigma = \Sigma_\mathrm{d} \cup \Sigma_\mathrm{a}$. Analogously to the above *valid* function defined for a single system (see 4.1 and 4.2), for an event $\sigma \in \Sigma$ a *valid* function for a composition of systems shall determine whether σ is semantically correct with the dependent system in a state s_d and the affecting system in state s_a. That is, if σ is semantically correct, the *valid* function shall return *true*, otherwise it shall return $false$. The signature for such a $valid_\mathrm{comp}$ function is:

$$valid_\mathrm{comp} : S_\mathrm{d} \times S_\mathrm{a} \times \Sigma \to \{true, false\} \tag{4.3}$$

Since the dependent system does not affect the affecting system, for a $\sigma \in \Sigma_\mathrm{a}$ it is sufficient to check whether σ is semantically correct in the state $s_\mathrm{a} \in S_\mathrm{a}$, for which the *valid* function defined in 4.2 can be re-used. For a $\sigma \in \Sigma_\mathrm{d}$ the *valid* function defined in 4.2 can also be re-used to check if σ is semantically correct in the state $s_\mathrm{d} \in S_\mathrm{d}$. But additionally for a $\sigma \in \Sigma_\mathrm{d}$ it has to be checked whether it is semantically correct in the state $s_\mathrm{a} \in S_\mathrm{a}$, since the dependent system is affected by the affecting system. Therefore a *context* function has to be defined that for a given $\sigma \in \Sigma_\mathrm{d}$ and $s_\mathrm{a} \in S_\mathrm{a}$ determines whether σ is semantically correct in state s_a. The signature for such a *context* function is:

$$context : S_\mathrm{a} \times \Sigma_\mathrm{d} \to \{true, false\} \tag{4.4}$$

The definition of this *context* function is specific for every pair of a dependent and affecting system. In the example of the vehicle trunk and vehicle state (see figure 4.4, page 42) the *context* function can be realised with table 4.2.

Table 4.2: Table for the *valid* function for a vehicle trunk dependent on the vehicle's state

State / Command	unlock	lock	open	close
Stopped	*true*	*true*	*true*	*true*
Driving	*false*	*true*	*false*	*true*

Using the *valid* function defined in 4.2 and the *context* function from 4.4 the $valid_\mathrm{comp}$ function can be defined as:

$$valid_\mathrm{comp}(s_\mathrm{d}, s_\mathrm{a}, \sigma) = \begin{cases} true, & \text{if } \sigma \in \Sigma_\mathrm{a} \wedge \sigma \notin \Sigma_\mathrm{d} \wedge valid(s_\mathrm{a}, \sigma) \\ true, & \text{if } \sigma \in \Sigma_\mathrm{d} \wedge \sigma \notin \Sigma_\mathrm{a} \wedge valid(s_\mathrm{d}, \sigma) \wedge context(s_\mathrm{a}, \sigma) \\ true, & \text{if } \sigma \in \Sigma_\mathrm{d} \cap \Sigma_\mathrm{a} \wedge valid(s_\mathrm{d}, \sigma) \wedge valid(s_\mathrm{a}, \sigma) \wedge context(s_\mathrm{a}, \sigma) \\ false, & \text{otherwise} \end{cases} \tag{4.5}$$

With the $valid_{\text{comp}}$ function defined in 4.5 for any $\sigma \in \Sigma$, $s_{\text{d}} \in S_{\text{d}}$ and $s_{\text{a}} \in S_{\text{a}}$ it can be checked if σ is semantically correct, e.g. $valid_{\text{comp}}(closed\ \&\ unlocked,\ stopped,\ open) = true$, while $valid_{\text{comp}}(closed\ \&\ unlocked,\ driving,\ open) = false$.
In cases when only the dependent system's alphabet Σ_{d} needs to be validated, it is sufficient to realise the $valid_{\text{comp}}$ function covering Σ_{d} only. This is the case in the above example where the vehicle trunk is the dependent system and the vehicle's state is the affecting system. So the $valid_{\text{comp}}$ function can be realised with two separate tables: one for the trunk and one for the vehicle's state. The table for the trunk remains unchanged, see table 4.1, page 43, and determines for a $\sigma \in \Sigma_{\text{d}}$ and a state $s_{\text{d}} \in S_{\text{d}}$ whether σ is semantically correct in state s_{d}. The table for the vehicle's state, see table 4.2, page 44, i.e. the realisation of the *context* function, determines for a $\sigma \in \Sigma_{\text{d}}$ and a state $s_{\text{a}} \in S_{\text{a}}$ whether σ is semantically correct in state s_{a}. The outcomes of both tables are combined in a conjunction, i.e. the Boolean AND operation, to determine whether $\sigma \in \Sigma_{\text{d}}$ is correct with the trunk being in state s_{d} and the affecting system, i.e. the vehicle, being in state s_{a}.
For a dependent system with n_{d} states and an alphabet size of m and an affecting system with n_{a} states the table for the dependent system has $n_{\text{d}} * m$ entries and the table for the affecting system has $n_{\text{a}} * m$ entries, resulting in a total of $(n_{\text{d}} + n_{\text{a}}) * m$ entries for realising the $valid_{\text{comp}}$ function covering Σ_{d}. The time complexity is again $O(1)$. Separate tables for each system are easier to maintain, e.g. when the affecting system is modified.
A dependent system can depend on more than one affecting system. In case of multiple affecting systems, the affecting systems can affect the dependent system either independently of each other, or interdependently. Figure 4.5 depicts a dependent system *FSM1* with a set of states $S_{\text{d}} = \{E, F\}$ and an alphabet $\Sigma_{\text{d}} = \{e, f\}$ affected by affecting systems *FSM2* (with a set of states $S_{\text{a1}} = \{A, B\}$ and an alphabet Σ_{a1}) and *FSM3* (with a set of states $S_{\text{a2}} = \{C, D\}$ and an alphabet Σ_{a2}).

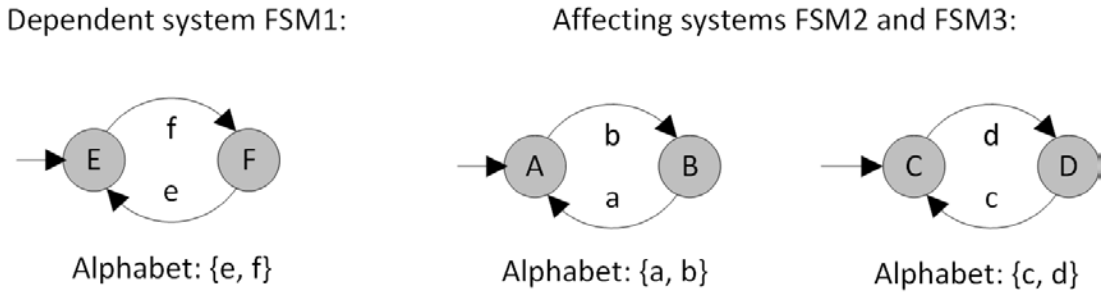

Figure 4.5: Exemplary dependent system with two affecting systems

When the state of affecting system FSM_2 has no effect on how the other affecting system FSM_3 affects the dependent system FSM_1 (and vice versa) the affecting systems affect the dependent system independently of each other. E.g. the event $e \in \Sigma_\mathrm{d}$ triggers the transition of the dependent system from state F to E. Suppose that this transition of the dependent system is only valid when the affecting system FSM_2 is in state A and the affecting system FSM_3 is in state C. Though both affecting systems affect the dependent system, they do so independently of each other. Thus in case of multiple independently affecting systems each affecting system can be considered separately in the semantic analysis. For every affecting system a_i a function $valid_{comp\ i}$ of the form 4.5 can be evaluated. The resulting Boolean values have to be combined in a conjunction to determine whether the given event is valid or not. If the conjunction results in *true* the event is valid, else it is invalid. Again the $valid_{comp}$ functions can be realised with tables. For a dependent system with n_d states, an alphabet Σ_d of the size m and k affecting systems with n_i states each, this would result in a total of $(n_\mathrm{d} + \sum_{i=1}^{k} n_\mathrm{i}) * m$ entries for realising the $valid_\mathrm{comp}$ functions. Separate tables for each system are easier to maintain, e.g. when affecting systems are added or removed.

When the state of affecting system FSM_2 does have an effect on how the other affecting system FSM_3 affects the dependent system FSM_1 (or vice versa) the affecting systems affect the dependent system interdependently. An example for figure 4.5 would be if the event $e \in \Sigma_\mathrm{d}$, which triggers the transition of the dependent system from state F to E, was only valid either when FSM_2 is in state A and FSM_3 is in state C, or when FSM_2 is in state B and FSM_3 is in state D. With interdependently affecting systems the effect of an affecting system on the validity of an event of the dependent system $\sigma \in \Sigma_\mathrm{d}$ cannot be considered separately in the semantic analysis. For evaluating an event with a $valid_\mathrm{comp}$ function this means that the used *context* function cannot be defined for the dependent system's alphabet Σ_d and the set of states of a single affecting system like in 4.4. Instead the *context* function has to return *true* or *false* for a given event $\sigma \in \Sigma_\mathrm{d}$ and any possible constellation of the interdependently affecting systems' states. So with k interdependently affecting systems it has to be defined for the dependent system's alphabet Σ_d and the cross product of states $S_\mathrm{a1} \times S_\mathrm{a2} \times ... \times S_\mathrm{ak}$. For realising the *valid* function with a table this means that for a dependent system with k interdependently affecting systems the cross product of the dependent and affecting systems' states $S_\mathrm{d} \times S_\mathrm{a1} \times ... \times S_\mathrm{ak}$ is required. For a dependent system with n_d states, an alphabet Σ_d of the size m and k affecting systems with n_i states each, this would result in a total of $m * n_\mathrm{d} * \prod_{i=1}^{k} n_\mathrm{i}$ entries for realising the $valid_\mathrm{comp}$ function. So the interdependence significantly increases the space complexity of the analysis compared to independently affecting systems.

For a correct analysis the V2X Application-Level Gateway has to get the current states of all relevant components. E.g. in the remote trunk control example it needs the current states of the trunk and the vehicle at all time. This context update of the V2X Application-Level Gateway can generally be realised with 2 different methods: either the gateway is notified whenever a context state change occurs or it proactively requests the current state. The request can be sent either periodically, or each time a certain event occurs ("event-driven"). E.g. the arrival of data for a context-sensitive semantic analysis could be such an event. So there are 3 different approaches to compare: context notifications, periodic context requests and event-driven context requests. To identify the best approach, they are compared based on 2 criteria: their contribution to the (in-vehicle) network load and the effort of an efficient implementation.
The contribution of the context update to the network load of the in-vehicle network depends mainly on the size and frequency of the notification- and request/response-messages. The size is constant for all messages and the request approaches come with an additional message compared to the notification approach. With the event-driven context requests approach the frequency of request/reply-messages could be significantly elevated by an attacker by sending a great number of messages to the V2X Application-Level Gateway. Since for every received message it sends a request and receives a reply this would result in unpredictable higher network load. Whereas with the periodic request approach and the notification approach even if the configured frequency is high, the network load is a priori known. So when it comes to the contribution to the network load, the notification approach has a lesser contribution than the periodic request approach, while the event-driven request approach can be ruled out as an alternative due to its vulnerability.
When it comes to the effort of an efficient implementation, for the periodic request approach reasonable cycle times have to be identified first, while the notification approach can be implemented directly. So under the criteria of contribution to in-vehicle network load and the effort of an efficient implementation the notification approach is preferable for realising context updates and therefore used in the V2X Application-Level Gateway.
In any vehicle security architecture the V2X Security Gateway is only the first line of defence. The semantic analysis of data, e.g. the command to open the trunk, can not only be performed by the V2X Application-Level Gateway, but additionally also by ECUs of the in-vehicle network e.g. a domain controller, see figure 4.6, page 48. In this case the ECU has to be able to decrypt the data prior to analysis. It also has to have knowledge of the context required for a context-sensitive semantic analysis if such an analysis is to be performed. Performing a (context-sensitive) semantic analysis of the same data in

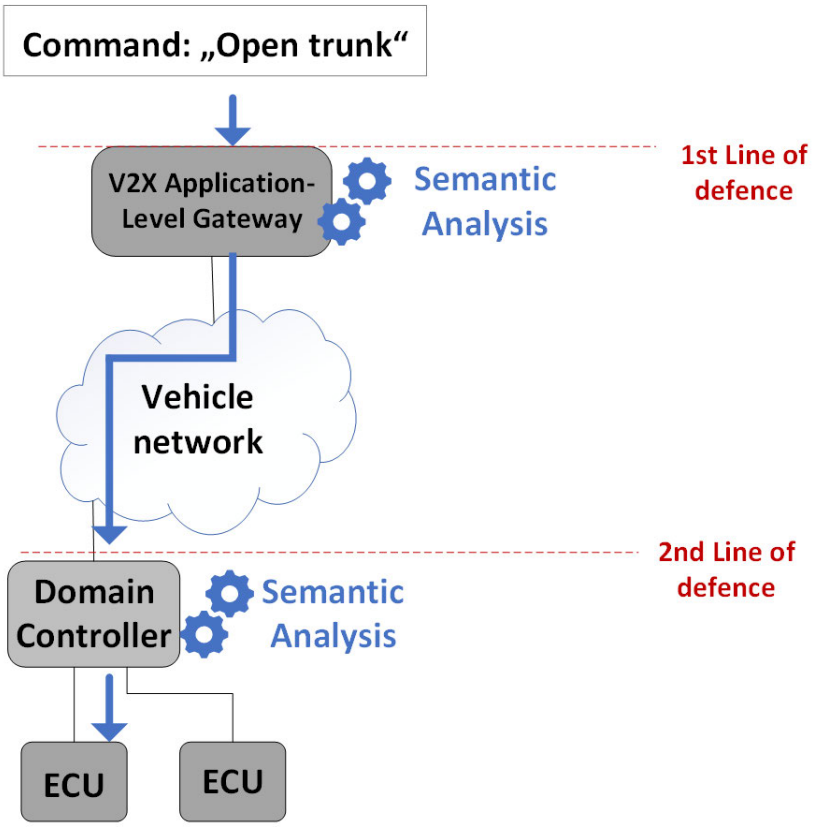

Figure 4.6: Simple example of distributed semantic analysis in a vehicle

multiple components increases security, since if an attacker e.g. manages to surpass the V2X Application-Level Gateway, e.g. by compromising a component of the in-vehicle network, he would still have to surpass a second line of defence. On the other hand the effect of the analysis on a component's performance has to be taken into account. For instance if the V2X Application-Level Gateway is already processing a lot of traffic, for performance reasons the semantic analysis performed by it could be limited to a general analysis on higher levels of abstraction, e.g. checking the data types, while a more specific, e.g. context-sensitive analysis would be performed in an in-vehicle ECU. So for performance reasons an interlocking of the analysis in multiple components, which still covers the entire semantics of the data, but with little or no redundancy, could be preferred depending on the network load and resources available in the vehicle.

4.2.3 Semantic Analysis of ETSI CAMs

For the stateful analysis in section 4.2.2 the dependent and affecting systems could be modelled with limited numbers of distinct states such as ”open” or ”driving”. This makes a *valid* function mapping the states and alphabet on Boolean values, realised with tables, an efficient solution. However, often instead of operating on a set of distinct states and an alphabet with a limited number of elements like in section 4.2.2, the semantic analysis amounts to comparing a discrete value to one or more thresholds. For the semantic analysis in the automotive domain this is the case for e.g. the analysis of V2V messages like ETSI CAMs (chapter 2, section 2.2.2).
When deciding on the extent and complexity of the analysis of ETSI CAMs it has to be kept in mind that ETSI CAMs are not the only source of information on nearby objects a modern vehicle will use to assess the situation around it. Information on nearby objects will also, or even primarily, come from a range of sensors, like e.g. LIDAR (*Light Detection and Ranging*) or cameras. The data from all these sensors will be merged with the data from ETSI CAMs to get the best possible assessment of the situation around the vehicle, see figure 4.7. For nearby vehicles that are not in the range of any sensor, i.e. occluded vehicles, ETSI CAMs will be the only source of information until vehicles start sharing their sensor-based knowledge on nearby objects via V2V. From the merged

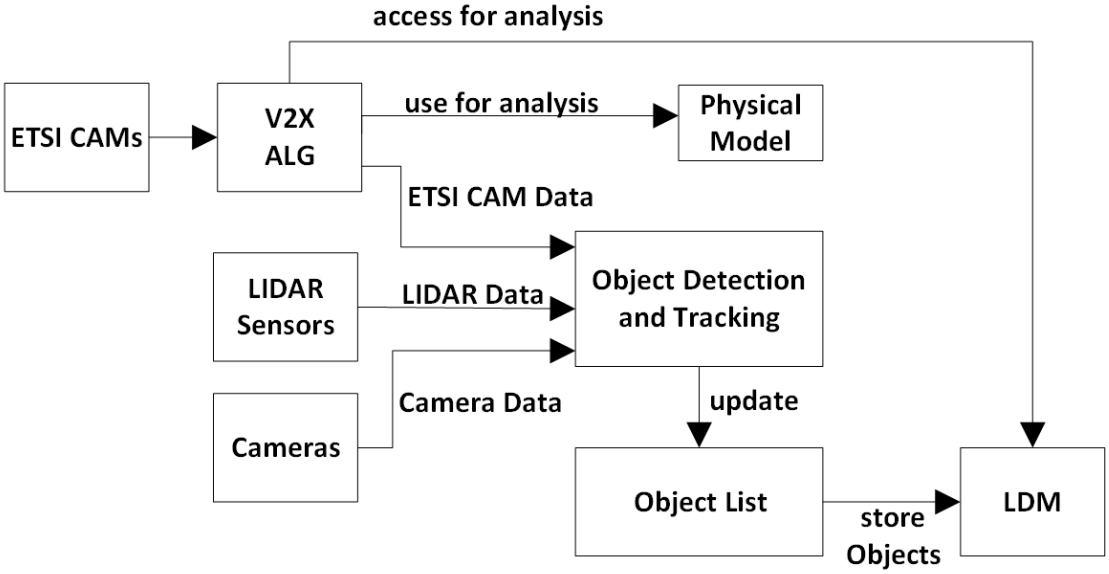

Figure 4.7: Analysis based data from multiple sources: ETSI CAMs and sensors

data the vehicle gets a list of objects, which will be stored in the LDM complementing the map data. The LDM can be accessed by any component requiring information on the vehicle's environment, e.g. the V2X Application-Level Gateway to get information on nearby vehicles for the semantic analysis of ETSI CAMs. Using that information and a physical model the V2X Application-Level Gateway can analyse the data of ETSI CAMs. For such an analysis the LDM data has to come with time-stamps indicating how "old" the information is. Note that due to the sampling rates of the sensors, the update frequency of the LDM is greater than the ETSI CAM frequency, i.e. the information on objects can be updated based on sensor data without the reception of an ETSI CAM. The physical model generally contains all physics-related information necessary for evaluating vehicle data such as speed or position.
So ETSI CAM data can not only be validated by analysing ETSI CAMs only, but it can also be validated using the data from sensors like LIDAR and cameras. If the information from an ETSI CAM deviates from the LIDAR and camera data, especially when the LIDAR and camera data are coherent, this is a strong indication that the ETSI CAM data is invalid. A central aspect when comparing the reliability of ETSI CAMs and sensors such as LIDAR or cameras, is that the sensor data is based on physical measurements, while ETSI CAM data is based on trusting that the sender is neither malicious nor compromised nor malfunctioning.
For the detection of many attacks, especially sophisticated attacks, it makes sense to base the analysis on the merged data from both ETSI CAMs and sensors, as depicted in figure 4.7, instead of analysing ETSI CAM data only. This way, the analysis will not be performed redundantly on each separate set of data, i.e. ETSI CAMs, LIDAR and camera data, but instead it will be performed on the merged data, which offers better information and certainty. A typical attack realised via ETSI CAMs would be sending ETSI CAMs representing vehicles that do not exist to influence the behaviour of receiving vehicles. Such an attack can be detected easier by analysing the merged data, since to insert a fake object which is in the field of view of the vehicle's sensors, also the sensor data would have to be manipulated, which is impossible for an attacker using only ETSI CAMs. Fake occluded vehicles would still be an option for the attacker, but this requires knowledge of the current field of view of all of the vehicle's sensors and besides, inserting a fake occluded vehicle is not as effective as inserting a fake vehicle that could have a direct impact on the attacked target, e.g. collide directly at any moment. In the future vehicles could also share their (sensor-based) knowledge on nearby objects via V2V, which on the one hand would add a new attack vector (i.e. spreading false information about objects), but on the other hand would increase the difficulty of spoofing using

fake objects. When two or more objects from the object list overlap, this could indicate an attack, i.e. inserted fake objects overlapping with real objects, or a collision of real objects. Again, assessing the probability of a collision based on the merged data makes more sense than using the information from ETSI CAMs alone. To correctly assess such a situation, not only the current states (here: positions) of nearby objects are required, but the analysis has to take into account sequences of states (over time). E.g. in the example of the collision, the sequences of the objects' positions up to the overlapping, i.e. before they overlap, determines the probability of the overlapping being a collision: if the sequences of the objects' positions result in a realistic collision course, the overlapping is likely to be a collision. Furthermore, the sequences of the objects' positions after they overlap for the first time, determine the probability of the overlapping being a collision: if the objects' positions do not change, the overlapping is even more likely to be a collision, while e.g. two overlapping objects moving parallel to the analysing vehicle, which is driving on a highway, exclude a collision and indicate an attack.
Yet another example is the analysis of the heading of vehicles. If e.g. all nearby vehicles on a one-way road are heading north (according to their ETSI CAMs) and one vehicle sends an ETSI CAM saying it is heading south, this deviation can be better checked based on the merged data from both sensors and ETSI CAMs instead of just analysing ETSI CAMs.
So for detecting attacks where state information of multiple objects has to be compared or a more complex analysis of a single object's state over a longer period of time is required, it is better to analyse the merged data instead of analysing only ETSI CAMs.
The goal of the semantic analysis of ETSI CAMs is to identify ETSI CAMs which contain information that contradicts the laws of physics or ignores the technical limitations of modern road vehicles, to do a filtering of messages before more complex analyses are performed on the merged data. An example would be an ETSI CAM containing a vehicle speed of *600 km/h*, which exceeds what is possible with current technologies used in road vehicles.
When analysing the data from ETSI CAMs it has to be kept in mind, that most come with a confidence, since often a 100% accuracy of the data cannot be guaranteed. E.g. the vehicle position information usually comes with a predefined confidence level (e.g. 95%) [37]. One way of dealing with inaccuracies is defining tolerances used in the analysis to account for possible deviations of the ETSI CAM data from the real situation. To account for deviations due to inaccuracies in the measurement of a vehicle's state information (e.g. position or speed) a configurable tolerance for the analysis of ETSI CAMs should be specified for each physical quantity like position, speed etc. Another aspect

are atypical situations where the ETSI CAM data deviates from what is expected in a normal situation. An example would be a vehicle driving on a slippery road, where the speed information from the ETSI CAM may not correspond to the real moving speed of the vehicle, depending on how the speed is measured. E.g. a vehicle on an icy road might report a speed of *14 m/s*, while actually driving with a speed of *12 m/s* and covering a distance of *12 m* in one second, which would conflict with the speed information of the ETSI CAM (*14 m/s*). To express different levels of certainty whether an ETSI CAM is valid or not and to correctly assess a wider range of scenarios, i.e. reduce the probability of false positives in case of unusual but possible events like e.g. a vehicle driving on a slippery road or a collision, this work defines 3 classifications of ETSI CAMs, see figure 4.8: 1) typical (and therefore valid), 2) atypical (but possible and therefore also valid, but with a warning) and 3) invalid (due to conflict with the laws of physics or technological boundaries, i.e. a violation of the physical model used for analysis).

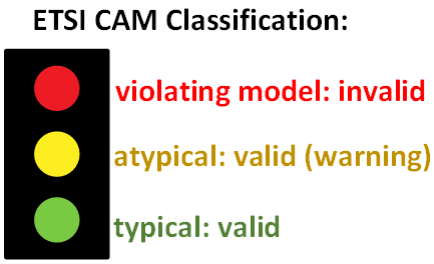

Figure 4.8: Classification of ETSI CAMs

The semantic analysis of ETSI CAMs can be either stateless or stateful. For checking a single value x, e.g. the (current) vehicle speed, a stateless analysis comparing the value x to a threshold can be sufficient. The threshold defines an upper or lower bound for valid values. One threshold is mandatory, while a second one is optional. E.g. for the vehicle speed an upper bound, e.g. *550 km/h*, restricting the vehicle speed to values possible with modern road vehicles is sufficient, since ETSI CAMs use unsigned integers for the vehicle speed and negative speeds are not defined and therefore the lower bound of *0 km/h* is superfluous. The *valid* function for such a stateless semantic analysis of ETSI CAMs can be defined as 4.6, with x being a single value from an ETSI CAM and a and b being thresholds.

$$valid(x) \iff \mathrm{a} \leq \mathrm{x} \leq \mathrm{b} \tag{4.6}$$

A threshold, or a pair of thresholds, divides all possible values into two sets: valid and invalid values. The physical model defines the thresholds for all physical quantities of a

vehicle, i.e. speed, acceleration etc. To additionally classify a valid value as either typical or atypical, another threshold (or pair of thresholds) has to be defined for a function of the form 4.6:

$$typical(x) \iff \mathrm{a} \leq \mathrm{x} \leq \mathrm{b}$$

E.g. for the vehicle speed a possible upper bound for typical values could be *220 km/h* (covering the scenario of fast drivers on a highway). So e.g. while a speed *>550 km/h* would be classified as invalid, a speed *≤550 km/h* but *>220 km/h* would be classified as valid but atypical and a speed *≤220 km/h* would be classified as valid and typical.
The thresholds to classify a valid value as either typical or atypical could be configured dynamically depending on the environment of the vehicle. E.g. the vehicle speed could be classified as typical or atypical depending on whether the vehicle is currently driving in a city or on a highway. Since such an analysis uses state information, i.e. information on the vehicle's environment, it would be classified as stateful analysis. Typical stateful analysis of ETSI CAMs would be comparing the change from the previous vehicle state, e.g. the last known vehicle position (obtained from the LDM), to the current vehicle state, e.g. the current vehicle position (obtained from the received ETSI CAM), to a threshold. The change from the previous state to the current state that is possible in a given time interval $\triangle$t is limited by the physical model. E.g. the position of a vehicle with a maximum driving speed of *153 m/s* cannot change by more than *153 m* in one second. The general form of a *valid* function defined in 4.6 can also be used for such a stateful analysis of ETSI CAMs with $\triangle$x being the state change (that occurred in a time interval $\triangle$t) and a and b being thresholds that need to be defined accordingly, depending on the specifics of the analysis:

$$valid(\triangle \mathrm{x}) \iff \mathrm{a} \leq \triangle \mathrm{x} \leq \mathrm{b}$$

For the position change $\triangle$x of a vehicle with a maximum speed v_{max} (e.g. *550 km/h*) a *valid* function could be defined as:

$$valid(\triangle \mathrm{x}) \iff 0 \leq \triangle \mathrm{x} \leq v_{\max} \cdot \triangle \mathrm{t}$$

Analogously to the above distinction of valid values into typical and atypical values, again a second threshold could be defined to distinguish between typical and atypical position changes.
So far only a single physical quantity, e.g. vehicle speed or position change, was compared to a configurable threshold derived from the technical limitations of modern road vehicles

(e.g. maximum speed). Usually multiple physical quantities are related, with their relationship being defined by the laws of physics. An example is the position and speed of a vehicle. The rest of this section exemplary shows the analysis of related ETSI CAM data, here position and speed data. First the relationship between the data has to be identified. Then equations and inequalities resulting from this relationship, which can be used for the classification of related ETSI CAM data, have to be defined. Also all necessary assumptions that have to hold for the above model to apply have to be formulated.
It is now examined how an ETSI CAM can be classified based on the consistency of its position and speed data with the physical model. When including a vehicle's position in the semantic analysis of ETSI CAMs one has to be aware of the difference between the position change (displacement) $\triangle$x of a vehicle and the distance d travelled by the vehicle, see figure 4.9. The distance d travelled can be equal to, or greater than the

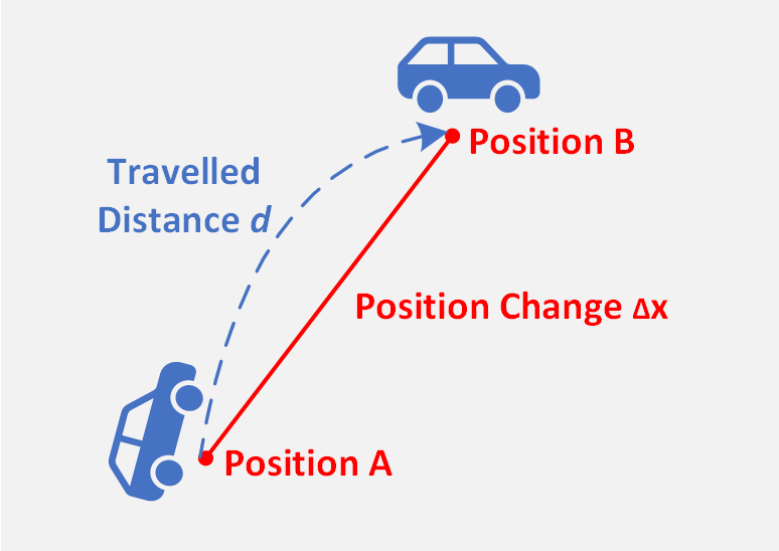

Figure 4.9: Position change (displacement) $\triangle$x and distance d

position change $\triangle$x, depending on whether the route of the vehicle was a straight line or curved, i.e.:

$$\triangle \mathrm{x} \leq d \tag{4.7}$$

The (average) speed s of an object is defined as the distance travelled d in the time $\triangle$t:

$$s = \frac{d}{\triangle \mathrm{t}} \implies d = s \cdot \triangle \mathrm{t} \tag{4.8}$$

The position change $\triangle$x in the time $\triangle$t defines the (average) velocity of an object:

$$v = \frac{\triangle \mathrm{x}}{\triangle \mathrm{t}} \implies \triangle \mathrm{x} = v \cdot \triangle \mathrm{t} \tag{4.9}$$

From an ETSI CAM the current speed s_i of the vehicle and the position change $\triangle$x between the current and the previous state can be determined. The speed $s_{i\text{-}1}$ in the previous state can be obtained from the LDM. The time $\triangle$t between the current and the previous state is in the range of 100 ms to 1 second for occluded vehicles, whose state is only updated upon the reception of an ETSI CAM, since no sensor data on it is available. Due to an increase in ETSI CAM frequency with increasing dynamics (i.e. significant changes in position, speed or heading) $\triangle$t is closer to the minimum period of 100 ms for vehicles in dynamic situations, while during steady driving $\triangle$t is closer to the maximum period of 1 second. Note that the range of 100 ms to 1 second is only true, when there are no significant network losses increasing the time between two consecutive ETSI CAMs. For vehicles that are not occluded and therefore their state is also updated based on sensor data the range of $\triangle$t decreases. For occluded vehicles the range of $\triangle$t can be decreased when sensor data is shared via V2V without significant network losses. With the assumption that for sufficiently small $\triangle$t the speed function of a driving vehicle in the time interval $\triangle$t can be approximated with a monotonous function, the speed $s_{i\text{-}1}$ and s_i is the minimum and maximum speed (or the other way round) in the time interval $\triangle$t. From 4.7 and 4.8 follows:

$$\triangle \mathrm{x} \leq s \cdot \triangle \mathrm{t} \tag{4.10}$$

Let $s_{\max} = max(s_{\mathrm{i\text{-}1}}, s_{\mathrm{i}})$ be the maximum speed in the time interval $\triangle$t. The average speed s is unknown and could only be calculated from $s_{i\text{-}1}$ and s_i under the assumption of linearity of the speed function in $\triangle$t, but s can be substituted with s_{max} without changing the inequality in 4.10:

$$\triangle \mathrm{x} \leq s_{\max} \cdot \triangle \mathrm{t} \tag{4.11}$$

So with 4.11 the upper bound b for the change in position $\triangle$x between two consecutive states with a sufficiently small $\triangle$t can be defined as:

$$b = s_{\max} \cdot \triangle \mathrm{t}$$

This upper bound can be used in a *valid* function of the form 4.6 for a stateful semantic analysis of ETSI CAMs checking if a CAM is in accordance with the physical model in respect to the change in position $\triangle$x.

Of course, if an attacker manipulates data consistently (e.g. vehicle position and speed) and within certain boundaries (e.g. maximum speed), so that it is in accordance with the physical model, such a manipulation cannot be detected with the semantic analysis of ETSI CAMs presented in this section.

4.3 Application Layer Protocol Message Sequence Violation

In this section it is examined whether the stateful analysis presented in section 4.2.2 for the semantic analysis of application data can also be applied to detecting protocol message sequence violations in application layer protocols. Message sequence violations are naturally only applicable to stateful protocols with defined sequences, e.g. MQTT [84] (where e.g. a *PUBLISH* has to come after a *CONNECT* and *CONNACK*) and not applicable to stateless protocols like e.g. the ETSI Cooperative Awareness Basic Service. Therefore, the MQTT protocol is used exemplary in this section to examine whether the stateful analysis presented in section 4.2.2 is applicable for the detection of protocol message sequence violations.
The V2X Application-Level Gateway, which acts as an intermediary between in-vehicle MQTT clients and an MQTT broker, see figure 4.10, has to classify the incoming and outgoing MQTT messages as valid or invalid depending on whether they adhere to the message sequence defined in the protocol or not (for better clarity, the acknowledgements of *PUBLISH* messages ($QoS > 0$) are not depicted).

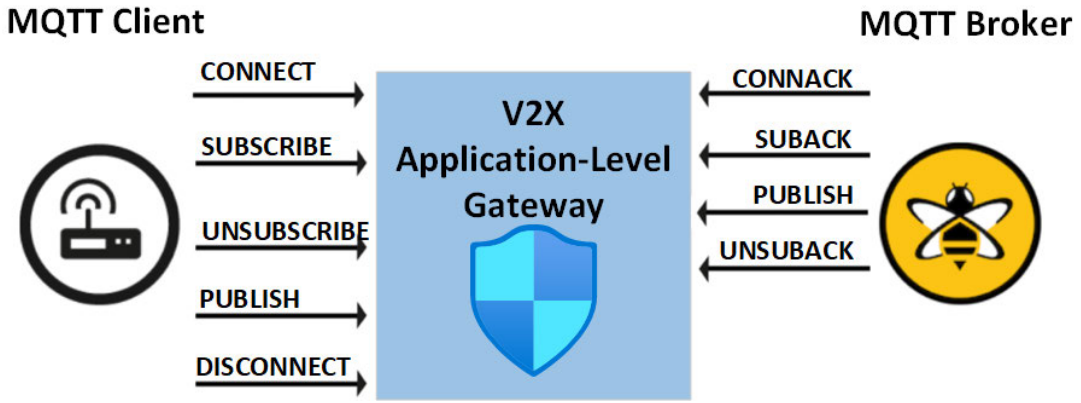

Figure 4.10: V2X Application-Level Gateway as intermediary between MQTT client and broker (used elements from [85])

Since the V2X Application-Level Gateway protects only in-vehicle MQTT clients (and not MQTT brokers), the analysis of the MQTT message sequence is done from the perspective of an MQTT client, i.e. the V2X Application-Level Gateway expects e.g. *CONNECT* messages to be outgoing only and *CONNACK* messages to be incoming only.
When an MQTT client sends a message to the broker, e.g. a *CONNECT* or *SUBSCRIBE*, it expects a certain response, e.g. a *CONNACK* or *SUBACK*. So by deliberately not answering, the broker violates the defined message sequence. However, from the perspective of the client such a violation of the protocol message sequence is impossible to distinguish from messages lost due to problems in the network and thus is not detected by the V2X Application-Level Gateway. The MQTT protocol does not specify an exact re-transmission mechanism or time-outs for lost (unacknowledged) messages, but it does specify a set of rules for handling messages that require acknowledgements (*SUBSCRIBE*, *UNSUBSCRIBE* and *PUBLISH* ($QoS > 0$)) [84], see below. The V2X Application-Level Gateway only detects violations by received messages σ, using a *valid* function. The *valid* function was defined in 4.1 and 4.2 in section 4.2.2 (page 43) as:

$$valid : S \times \Sigma \rightarrow \{true, false\}$$

$$valid(s, \sigma) \iff \delta(s, \sigma) \neq \emptyset$$

To apply this *valid* function for detecting protocol message sequence violations in application layer protocols, the set of states of the protocol S and the set of messages Σ have to be mapped on Boolean values *true* and *false*, using the transition function δ, which defines the protocol message sequence, analogous to the states and events of an analysed system in the semantic analysis in section 4.2.2. So that for a protocol state $s \in S$ and a message $\sigma \in \Sigma$ it can be determined whether the message σ adheres to the message sequence defined by the protocol or violates it.
However, in case of the MQTT protocol the set of states S and the alphabet Σ cannot be defined as easily as in the case of a system with a limited number of discrete states and events, like e.g. a vehicle trunk. In MQTT, once a client has connected with a broker, it can send *SUBSCRIBE* messages, each containing a topic and a packet ID, to the broker to subscribe to topics. For every *SUBSCRIBE* message it has to receive a *SUBACK* message containing the packet ID of the respective *SUBSCRIBE* message. Unsubscribing works analogously to subscribing. *UNSUBSCRIBE* messages containing no matching topic are also acknowledged with an *UNSUBACK*. A client can have only one subscription to the same topic. Therefore it is specified that upon receiving a *SUB-*

SCRIBE message from a client containing a topic to which the client already subscribed, the broker replaces the existing subscription with an (identical) subscription to that received topic.
Publishing messages with the quality of service level *QoS 1* works analogously to subscribing: the receiver acknowledges every *PUBLISH* with a *PUBACK*. Publishing with *QoS 2* features a two-level handshake where the publisher additionally acknowledges the reception of the receivers acknowledgement, which in turn is acknowledged by the receiver, so that both the publisher and receiver are sure that both know that the message was delivered, see figure 4.11. The sender of a *PUBLISH* with *QoS 2* must treat it as unacknowledged until receiving the corresponding *PUBREC* from the receiver, which the sender acknowledges with a *PUBREL*. The sender must not re-send the *PUBLISH* once it has sent the corresponding *PUBREL*. The receiver must acknowledge every *PUBLISH* with a *PUBREC* until it receives the *PUBREL*, but must not further process duplicate *PUBLISH* messages. The receiver must acknowledge every *PUBREL* with a *PUBCOMP*. After sending the *PUBCOMP* the receiver must treat any subsequent *PUBLISH* messages as new messages. While the MQTT protocol does not specify an exact

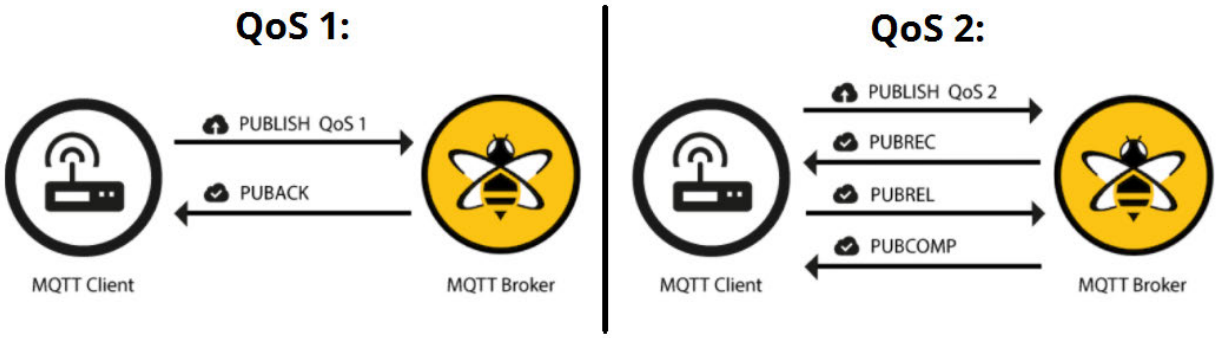

Figure 4.11: MQTT Quality of Service: *QoS 1* and *QoS 2* [85]

re-transmission mechanism or time-outs for publishing with $QoS > 0$, it limits the number of unacknowledged *PUBLISH* messages a client or broker can manage concurrently. For every MQTT client and broker a limit for the number of *PUBLISH* messages it can send without receiving an acknowledgement has to be configured. A counter is initialised with this limit and it is decremented every time a *PUBLISH* ($QoS > 0$) is sent and incremented every time a *PUBACK* (for *QoS 1*) or *PUBCOMP* (for *QoS 2*) is received. If the counter reaches a value of ”0”, no more *PUBLISH* messages ($QoS > 0$) can be sent, until the counter is incremented by receiving *PUBACK* or *PUBCOMP* messages.
So modelling the exact state of the MQTT protocol in detail with one FSM seems im-

practical, since once the client is connected, each *SUBSCRIBE*, *UNSUBSCRIBE* and *PUBLISH* ($QoS > 0$) message (and their respective acknowledgements) adds information to the current state, resulting in complex global states of the modelled system, as a client can e.g. subscribe to a topic and then while waiting for the acknowledgement, publish multiple messages, again expecting an acknowledgement for each. Also the definition of an alphabet Σ, which explicitly includes all messages of the MQTT protocol $\sigma \in \Sigma$, and is required for the *valid* function defined in section 4.2.2, is impractical: for some MQTT messages, like e.g. *CONNECT* or *CONNACK*, only their message type contributes to the state information and thus is sufficient for distinctly identifying them in this context, which is why *CONNECT* and *CONNACK* could be added as elements of such an alphabet Σ. But for others, like e.g. *SUBSCRIBE* or *PUBACK*, in addition to the message type, also the packet ID contributes to the state information and thus is also required for identification, which is why e.g. *SUBSCRIBE* and *PUBACK* cannot be added as elements of such an alphabet Σ. Instead, every possible combination of e.g. *SUBSCRIBE* and a packet ID would have to be added as elements of the alphabet Σ. So instead of using one FSM, multiple (smaller) parallel FSMs can be used. One approach is a hierarchical FSM with a top-level automaton processing MQTT messages according to only their message type, see figure 4.12 and multiple parallel second-level automatons in the "Connected" top-level state processing MQTT messages according to both their message type and packet IDs, see figure 4.13, page 60. The top-level automaton is defined by a set of simple states S, being only a reduced set of abstract states of the MQTT protocol ("Unconnected", "Connecting" and "Connected"), an alphabet Σ consisting of all MQTT message types and a transition function δ: $S \times \Sigma \rightarrow S$ modelling the MQTT message sequence (as defined by the MQTT protocol), given only the reduced set of abstract states and message types of the MQTT protocol.

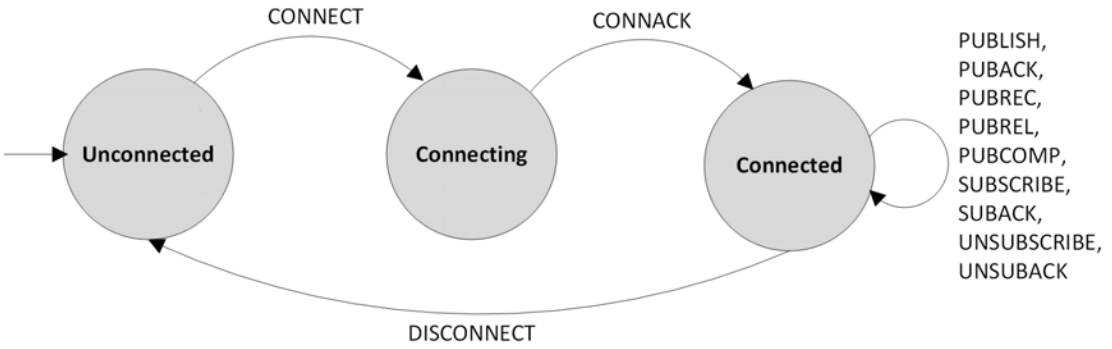

Figure 4.12: MQTT communication from the perspective of an MQTT client

Every MQTT message in the ”Connected”-state that is identified by both its message type and a packet ID x (e.g. *SUBSCRIBE* or *PUBLISH* ($QoS > 0$)) is processed in an instance x of a second-level automaton defined by a set of simple states S (e.g. $S =$ {Connected, Subscribing, Subscribed}), an alphabet Σ consisting of the respective subset of MQTT message types (e.g. $\Sigma = \{SUBSCRIBE, SUBACK\}$) and a transition function δ: $S \times \Sigma \rightarrow S$ modelling the respective MQTT message sequence part, see figure 4.13. So for every packet ID x an instance x of a second-level automaton exists.

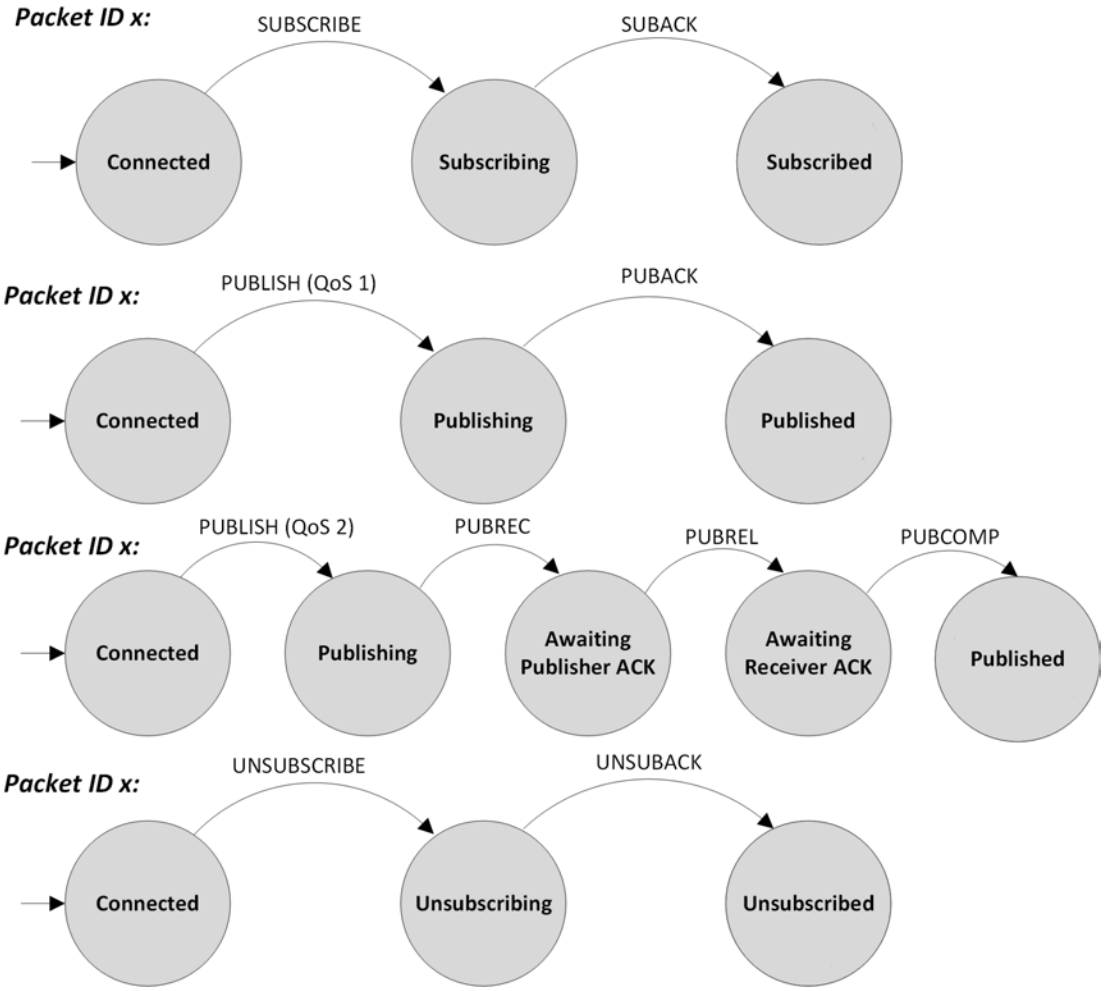

Figure 4.13: MQTT communication from the perspective of an MQTT client

Since it is possible for a message to be sent and delivered multiple times, duplicate instances of automatons are allowed. However one message always triggers only one instance of an automaton, in case of duplicates, the ”oldest” instance is triggered. This

way the stateful analysis defined in section 4.2.2 can be used for the detection of message sequence violations. For every MQTT message σ a *valid* function based on the top-level automaton is applied. If required (in case σ has a packet ID) also a *valid* function based on the respective second-level automaton is applied. In case of a message σ with a packet ID, the conjunction (logical AND) of both Boolean values returned by the *valid* functions has to be used to determine whether the message adheres to the defined protocol sequence or not. The analysis in this work focuses on the following set of MQTT message types:

- CONNECT
- CONNACK
- PUBLISH
- PUBACK (only *QoS 1*)
- PUBREC (only *QoS 2*)
- PUBREL (only *QoS 2*)
- PUBCOMP (only *QoS 2*)
- SUBSCRIBE
- SUBACK
- UNSUBSCRIBE
- UNSUBACK
- DISCONNECT

Analogously to semantic analysis in section 4.2.2 the *valid* function can be realised with a table for the top-level automaton. With such a table the V2X Application-Level Gateway can classify incoming (broker to client) and outgoing (client to broker) MQTT messages as invalid, when their message type indicates a message sequence violation. Incoming and outgoing MQTT traffic differs, see figure 4.10, page 56. E.g. a *CONNECT*, *SUBSCRIBE* or *UNSUBSCRIBE* should only be accepted by the V2X Application-Level Gateway from an MQTT client, while e.g. a *CONNACK*, *SUBACK* or *UNSUBACK* should only be accepted from an MQTT broker. To handle these different conditions for incoming and outgoing messages one solution are two different tables: one for outgoing 'client to broker'

traffic, see table 4.3, and one for incoming 'broker to client' traffic, see table 4.4. This way the compliance with different protocol roles can be efficiently integrated into the control of the protocol message sequence.

Table 4.3: Table for the *valid* function for the MQTT protocol (client-side)

Message / State	Unconnected	Connecting	Connected
CONNECT	*true*	*false*	*false*
CONNACK	***false***	***false***	***false***
PUBLISH	*false*	*false*	*true*
PUBACK	*false*	*false*	*true*
PUBREC	*false*	*false*	*true*
PUBREL	*false*	*false*	*true*
PUBCOMP	*false*	*false*	*true*
SUBSCRIBE	*false*	*false*	*true*
SUBACK	***false***	***false***	***false***
UNSUBSCRIBE	*false*	*false*	*true*
UNSUBACK	***false***	***false***	***false***
DISCONNECT	*false*	*false*	*true*

Table 4.4: Table for the *valid* function for the MQTT protocol (broker-side)

Message / State	Unconnected	Connecting	Connected
CONNECT	***false***	***false***	***false***
CONNACK	*false*	*true*	*false*
PUBLISH	*false*	*false*	*true*
PUBACK	*false*	*false*	*true*
PUBREC	*false*	*false*	*true*
PUBREL	*false*	*false*	*true*
PUBCOMP	*false*	*false*	*true*
SUBSCRIBE	***false***	***false***	***false***
SUBACK	*false*	*false*	*true*
UNSUBSCRIBE	***false***	***false***	***false***
UNSUBACK	*false*	*false*	*true*
DISCONNECT	***false***	***false***	***false***

The second-level automatons (for messages identified by their packet ID) can be implemented efficiently using lists, see figure 4.14. Adding a packet ID from a message (e.g. a *SUBSCRIBE message*) to a list corresponds to creating an instance of the respective automaton, while finding (and deleting) a packet ID upon a list-lookup corresponds to triggering the respective automaton instance (and deleting it if the final state was reached).

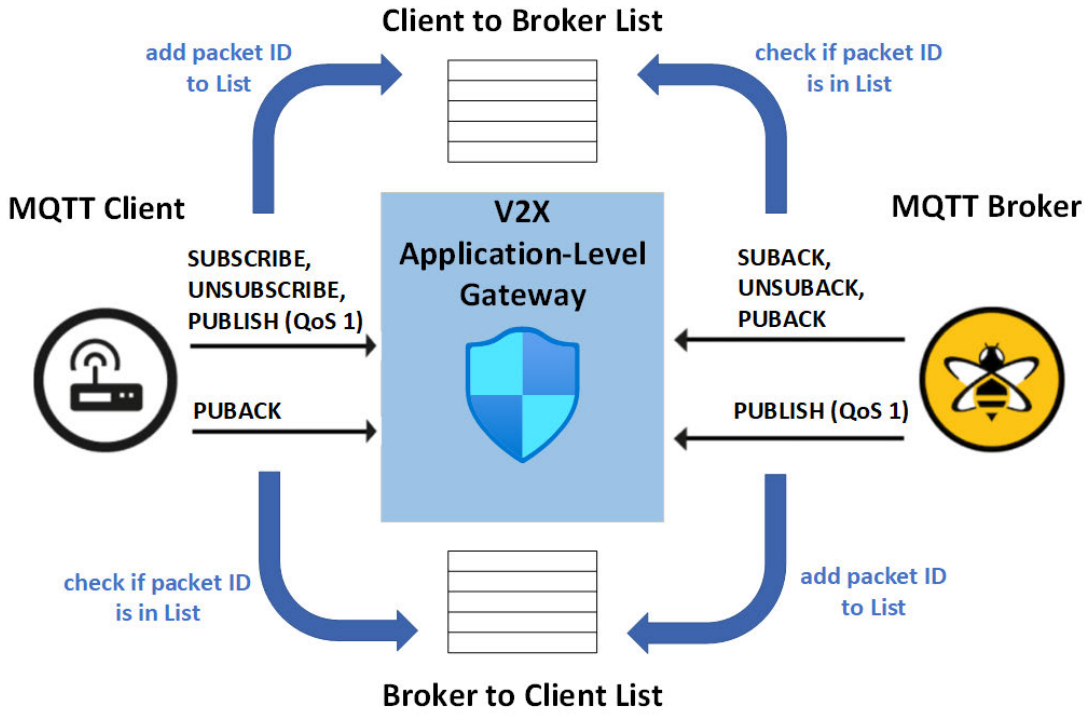

Figure 4.14: Detecting message sequence violations using lists (used elements from [85])

The packet IDs of outgoing (client to broker) messages, except acknowledgements, are stored in the *Client to Broker List* (i.e. packet IDs of *SUBSCRIBE*, *UNSUBSCRIBE* and *PUBLISH QoS 1* messages). For incoming (broker to client) acknowledgements (i.e. *SUBACK*, *UNSUBACK* and *PUBACK*) it is checked whether their packet ID is on the *Client to Broker List* and if it is, the message is classified as valid and the list entry is deleted. Otherwise the message is classified as violating the message sequence.Since it is possible for a message to be sent and delivered multiple times, duplicate entries are allowed.
Analogously, the packet IDs of incoming (broker to client) *PUBLISH QoS 1* messages are stored in the *Broker to Client List*. For outgoing (client to broker) acknowledgements

(i.e. *PUBACK*) it is checked whether their packet ID is on the *Broker to Client List* and if it is, the message is classified as valid and the list entry is deleted. Otherwise the message is classified as violating the message sequence.
Detecting message sequence violations for publishing with *QoS 2* messages works analogously, see figure 4.15. Since publishing with *QoS 2* requires more state information

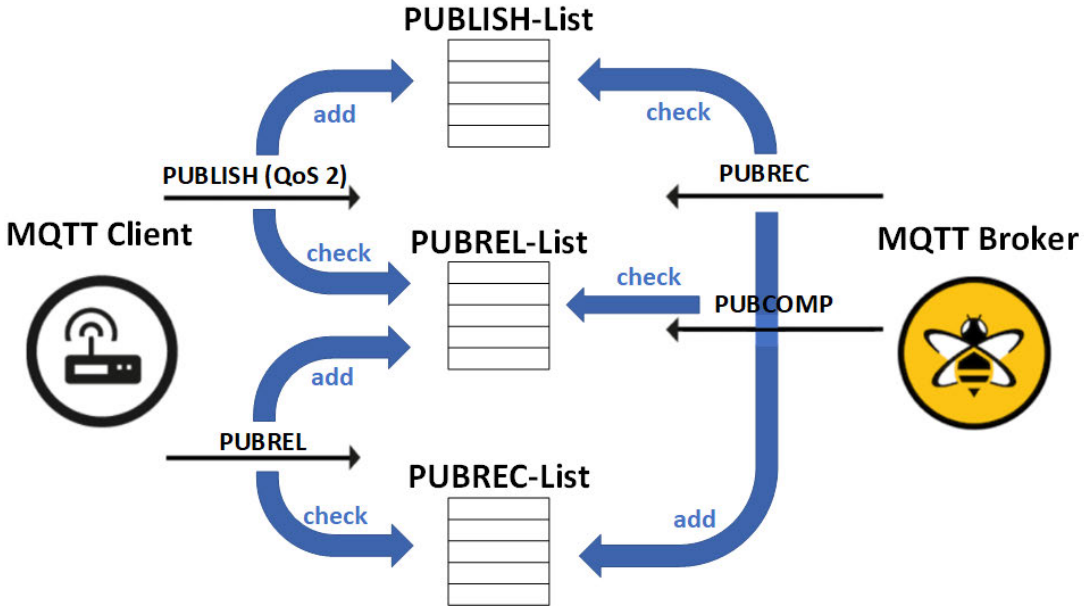

Figure 4.15: Message sequence violations detection for client-to-broker QoS 2 publishing, using lists (used elements from [85])

than communication with $QoS < 2$, multiple lists to account for the different acknowledgements are required. The additional lists can be viewed as extensions of the *Client to Broker List* or *Broker to Client List* respectively. For an outgoing *PUBLISH* it is checked whether its packet identifier is in the *PUBREL-List*, i.e. whether a corresponding *PUBREL* was sent. If none was sent, the *PUBLISH* is valid and its packet identifier is added to the *PUBLISH-List*. For an incoming *PUBREC* it is checked whether its packet identifier is in the *PUBLISH-List*, i.e. a corresponding *PUBLISH* is awaiting acknowledgement. If it is, the *PUBREC* is valid and its packet identifier is added to the *PUBREC-List*. For an outgoing *PUBREL* it is checked whether its packet identifier is in the *PUBREC-List*, i.e. whether a corresponding *PUBREC* was received. If it was received, the *PUBREL* is valid and its packet identifier is added to the *PUBREL-List*. For an incoming *PUBCOMP* it is checked whether its packet identifier is in the *PUBREL-List*, i.e. a corresponding *PUBREL* is awaiting acknowledgement. If it is, the *PUBCOMP*

is valid. Broker to client traffic is handled analogously. List entries for messages that require acknowledgement but are never acknowledged, e.g. a *SUBSCRIBE* lost due to network problems, will never be deleted. To prevent the size of the list from exceeding a configured maximum size, the oldest entries have to be deleted if the size of the list reaches the configured maximum size due to heavy network losses.
The solution proposed in this section ensures the adherence to the defined MQTT protocol message sequence. It was shown that it can be realised using the stateful analysis defined in section 4.2.2. For an application protocol modelled with n states and an alphabet of the size m the table realising the *valid* function has $n*m$ entries. Additionally, for MQTT a list (*Client to Broker List*/*Broker to Client List*) with a dynamically changing size is required. The size of the list depends on the amount of messages sent. The time complexity of the *valid* function for messages classified only based on their message type is $O(1)$, since it amounts to a lookup in a table, while messages classified based on both their message type and packet ID require an additional lookup in a list. Since the list containing n packet IDs can be sorted and in case of duplicate entries each lookup only needs to return a single ID, a binary search, which has a time complexity of $O(log\, n)$, can be used.
Of course, to ensure that an MQTT message is valid and can be forwarded by the V2X Application-Level Gateway to its destination, additional analysis of the structure and content of the header and payload is required, e.g. whether an incoming *PUBLISH* message is valid, also depends on whether its topic is in the list of subscribed topics. The same is true for an outgoing *UNSUBSCRIBE*. This work classifies messages with unmatched topics as messages with malicious content and not protocol message sequence violations, hence they are not covered in the analysis presented in this section. Also, *PUBLISH* and *SUBSCRIBE* messages could be filtered according to a black-list or white-list. But since this section focuses on message sequence violations, this additional analysis is not covered at this point.
In case an MQTT connection between a client and the broker breaks down, it can be re-established and the communication can continue where it broke off, i.e. the client and the broker still hold the state of the protocol before the connection broke down. The client has to reconnect with a *CONNECT* message, which the broker acknowledges with a *CONNACK*. The broker can identify a re-connecting MQTT client by its client ID. The client ID identifies each MQTT client that connects to an MQTT broker and should be unique per client and broker. When an MQTT connection breaks down, also the V2X Application-Level Gateway has to keep the relevant state information of this connection, i.e. the states of all second-level automatons (the lists) and assign it to the

new connection once the disconnected MQTT client has reconnected with the broker and continues communication where it broke off. For this, the V2X Application-Level Gateway also uses the client ID. The state of the top-level automaton is not kept, since a re-connecting MQTT client can only resume communication after it established an MQTT connection with the broker, i.e. reached the "Connected" state after receiving a *CONNACK*. The lists of a client's connection are deleted only after the client disconnects via MQTT *DISCONNECT*.

4.4 Application Layer DoS Detection

To realise the detection of application layer DoS attacks in the V2X Application-Level Gateway the following traffic and packet features identified in chapter 3 are used:

- number of messages per period of time (e.g. requests per second)
- packet inter-arrival time
- packet size
- ratio of packet types (e.g. the ratio of connection requests in MQTT traffic)
- number of connections

To control the traffic with respect to each feature a separate module for each feature is proposed: a *Messages per Period Module*, an *Inter-arrival Time Module*, a *Packet Size Module*, a *Packet Type Ratio Module* and a *Connections Limit Module*. For each of the features a threshold can be configured to distinguish DoS traffic from normal traffic. When the threshold is exceeded, the traffic is identified as DoS traffic and the V2X Application-Level Gateway can react accordingly, e.g. drop the traffic. Once the traffic normalises, i.e. is within the configured threshold, the traffic is identified as normal traffic and again forwarded by the V2X Application-Level Gateway.
The *Connections Limit Module* limits the number of connections allowed for a service. Once the configured limit is reached, no more connections for the service are accepted or established. The *Connections Limit Module* is integrated into the *Connection Manager*, see section 4.1. Since all the other modules are used for controlling the traffic of a single connection, they are integrated into the *Analyzer Component*, see section 4.1, of the respective connection. The *Messages per Period Module* limits the number of messages allowed in a configurable period of time, e.g. 10 messages in 1 second. If the number

of messages received in the configured period exceeds the configured threshold, i.e. the number of messages allowed, the traffic is identified as DoS traffic. E.g. if 20 messages are received in 1 second with the above configuration of 10 messages allowed in 1 second, the messages after the 10-th message are identified as DoS traffic. The *Inter-arrival Time Module* specifies the minimum packet inter-arrival time allowed. When the packet inter-arrival time of received messages drops below this minimum, the traffic is identified as DoS traffic. E.g. if a minimum packet inter-arrival time of 100 ms is configured and the inter-arrival time of received consecutive messages is $<$100 ms the traffic is identified as DoS traffic. Optionally a period and a number of messages with an inter-arrival time less than the configured minimum allowed can be specified. E.g. with a configured period of 1 second, a minimum packet inter-arrival time of 100 ms and a number of messages allowed of 5, 5 messages with an inter-arrival time less than 100 ms are allowed per second.

The *Packet Size Module* specifies an upper bound and lower bound for the size of packets. Packets with a *size out of bounds* are identified as DoS traffic. Like in the above module, optionally a period and a number of packets with a *size out of bounds* allowed in that period can be specified. E.g. with a configured period of 1 second and a number of packets allowed of 10, 10 messages with a size out of bounds are allowed per second.

The *Packet Type Ratio Module* specifies the allowed ratio of configurable message types of an application layer protocol, e.g. the ratio of MQTT CONNECT messages in MQTT traffic. A high proportion of MQTT CONNECT messages is not typical for legitimate traffic and could indicate a DoS attack. In addition to message types and their respective ratios, the period of time for which to calculate the ratios has to be configured. In the MQTT example the proportion of CONNECT messages will naturally be relatively high in the beginning and not reflect the nature of the traffic, while in time it will drop for legitimate traffic and remain relatively high in case of DoS traffic and thus be useful for detecting DoS traffic.

These modules, with the exception of the *Packet Type Ratio Module*, can be used for any application using any application layer protocol. The use of the *Packet Type Ratio Module* only makes sense when the application layer protocol defines different message types, like e.g. MQTT. In this work the thresholds are configured statically and learning-based techniques are not employed, since a static configuration is sufficient for the automotive use cases covered in this work, i.e. the remote control of a vehicle trunk, receiving traffic updates and a simple V2V safety service based on ETSI CAMs.

4.5 Cloud-based Approach

The V2X Application-Level Gateway does not necessarily have to be deployed as one component in the vehicle. Instead, some functionality could be transferred to the cloud, see figure 4.16, to relieve the V2X Application-Level Gateway component built into the vehicle.

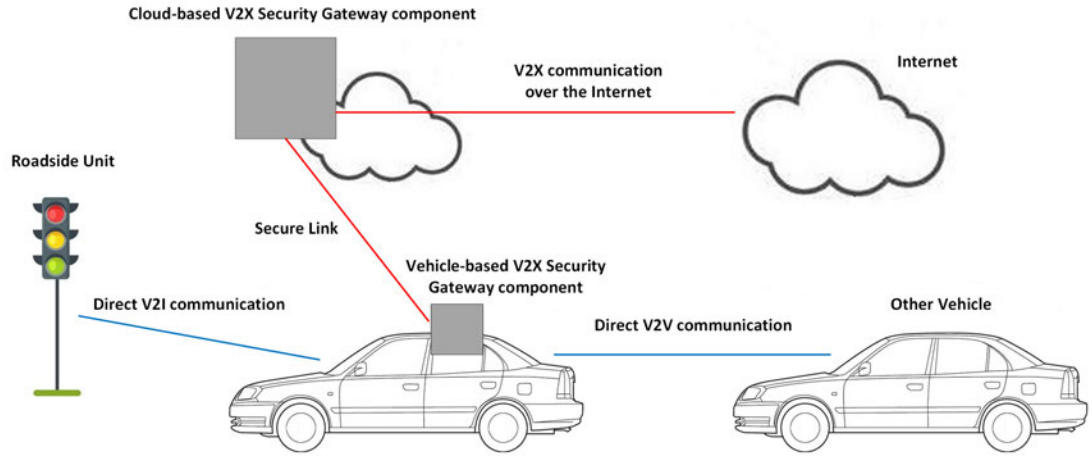

Figure 4.16: Cloud-based V2X Application-Level Gateway

In order to keep the delays of the direct V2V and V2I VANET communication low, it would still be entirely handled by the component built into the vehicle. But all Internet-based V2X communication could be handled by the cloud-based component. The vehicle-based component would only communicate with the cloud-based component over a cryptographically secured link and its only tasks regarding Internet-based traffic would be handling the encryption and the mapping of external addresses to vehicle internal addresses. The remaining functionality of the V2X Application-Level Gateway, like application layer security or access control, would be transferred to the cloud, where a lot more resources like memory and computing power are available than in a vehicle. Apart from being able to devote more resources to the transferred tasks, another benefit is cutting costs by having cheaper devices built in the vehicles and deploying multiple cloud-based V2X Application-Level Gateway components on one server. With one server handling multiple vehicles the data integration, which is a necessary step for e.g. a comprehensive ”big data” analysis, becomes easier. A downside is the dependency of the vehicles on the cloud infrastructure.

5 Prototype Implementation

In this chapter a prototype implementation of the concept of a V2X Application-Level Gateway described in chapter 4 is presented.

5.1 Implementation Overview

The prototype implementation serves as a proxy for V2X services, allowing both TCP/IP-based and WSMP-based communication between vehicle-internal services and external services. It focuses on providing application layer security, allowing the integration of different analysis modules securing V2X communication on the application layer. These analysis modules have to implement a defined interface ("*IAnalysis*", see below). The prototype implementation was written in C++.
It can be divided into 3 layers: a network layer, a buffering layer and an application-level layer, see figure 5.1. The network layer handles the communication of the V2X

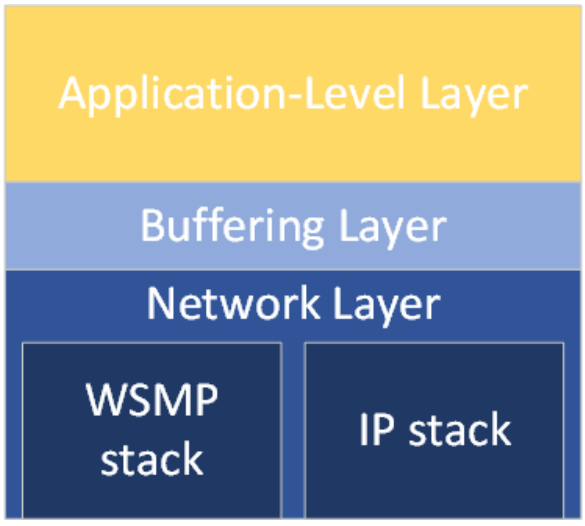

Figure 5.1: Implementation layers

Application-Level Gateway with other devices via sockets. It supports both the IP stack and the WSMP stack. The IP stack was implemented using the *POCO* libraries [96]. Since the *POCO* libraries do not cover the link layer (OSI layer 2) or other network layer protocols besides IP, the WSMP stack was implemented based on raw sockets. The implementation of WSMP for this work only supports WSMs with the mandatory fields and options necessary for associating the data transported in the payload with an application protocol (e.g. ETSI Cooperative Awareness Basic Service).
For performance reasons, to avoid unnecessary copying, data arriving at the V2X Application-Level Gateway is directly received into buffers where it can be analysed. These buffers comprise the buffering layer. The application-level layer encompassing the analysis modules, accesses this buffering layer directly.
For the prototype the following analysis modules have been implemented: an exemplary module for the context-sensitive semantic analysis of application data for a service remotely controlling the vehicle trunk with commands such as "open" or "lock" via HTTPS (see section 5.2.1), a module for detecting MQTT message sequence violations (see section 5.2.3), a module checking the payload size to prevent buffer overflow attacks via too large payloads, a module controlling the messages per period of time, i.e. the data rate of a connection, used for application layer DoS detection (as described in chapter 4, section 4.4) and a module for the context-sensitive semantic analysis of ETSI CAMs (see section 5.2.2). For the application layer DoS detection for connectionless ETSI CAM traffic the data rate of each sending vehicle is analysed.
While being aware of the vulnerabilities of TLS [20], to offer some degree of cryptographic security, SSL (with *openssl* [91]) is used for securing the TCP/IP-based communication over the V2X Application-Level Gateway. For cryptographically securing the TCP/IP-based communication SSL was used, because the *POCO* libraries used for the implementation, offer ready to use SSL functionality. Since cryptographic security is not the focus of this work, the cryptographic layer for the WSMP stack [65] was not implemented.
When the V2X Application-Level Gateway classifies a message as valid, it is forwarded to its destination. When a message is classified as invalid, the policy defined for the prototype is to drop the message and report detected attacks. Since the V2X Application-Level Gateway will be integrated into a vehicle network as part of the CoRE research group's [24] contribution to the SecVI project [105], it reports attacks to an Automotive Cyber Defense Center in the cloud [73]. The Automotive Cyber Defense Center uses a cloud infrastructure to monitor the cyber-security of large vehicle fleets and carry out incidence responses. IoT Edge technologies allow the security management of each vehicle. By monitoring a vehicle fleet, attacks can be detected through the correlations of anomalies

between multiple vehicles that cannot be detected by a single vehicle. The state of a vehicle's IT-infrastructure is monitored with security sensors in the car, including the V2X Application-Level Gateway [73, 81].
Reporting to the Automotive Cyber Defense Center by any component in the vehicle network is done via a central controller. To report an attack, a component, e.g. the V2X Application-Level Gateway, has to broadcast an Ethernet frame with the Ether-Type *0xFFAD* containing the report. The controller then reports the attack to the Automotive Cyber Defense Center. The V2X Application-Level Gateway prototype includes a *LoggingComponent* that handles the reporting of attacks to the Automotive Cyber Defense Center.
Also, the following libraries were used: the *pugixml* library [98] was used for reading configurations from XML files and serialization was realised with the *cereal* library [21].
A general overview of the prototype implementation's key elements realising the main functionality is depicted in figure 5.2 for the IP stack.

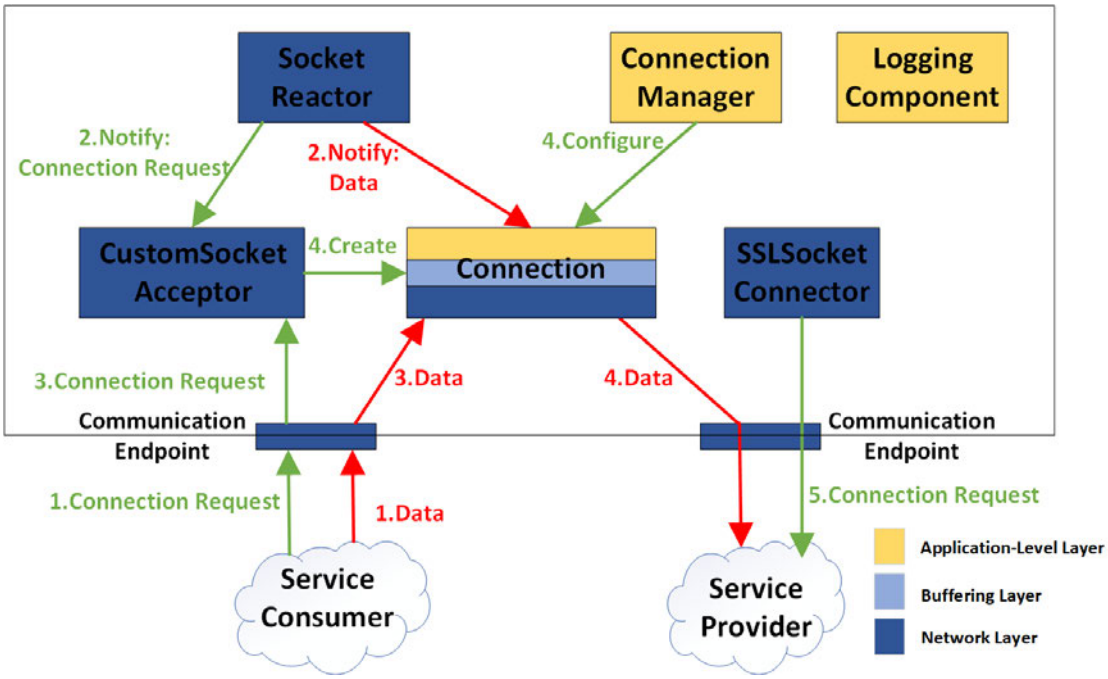

Figure 5.2: Prototype implementation overview (IP stack)

A *CustomSocketAcceptor*, derived from POCOs *SocketAcceptor* class, accepts incoming connection requests, e.g. from a remote trunk control client and creates a *Connection* upon accepting, to handle the accepted connection. Via an *SSLSocketConnector* an SSL connection to the remote peer, e.g. a remote trunk control server, is proactively estab-

lished ("Connector-Acceptor Pattern"). The *CustomSocketAcceptor* and *Connection* are called by a POCO *SocketReactor* whenever a new connection request or application data arrive, to handle the request or data ("Reactor Pattern").
Upon creation, the *Connection* is configured by the *ConnectionManager*, which uses configured factories to create the appropriate objects ("Factory Pattern").
Compared to the IP stack, the implementation for the WSMP stack is simpler, see figure 5.3. A *Connection*, which is configured by the *ConnectionManager*, receives Ethernet frames directly from a network interface via a raw socket and also sends outgoing Ethernet frames via a raw socket. A *Connection* consists of a *CompositeBuffer* for incoming traf-

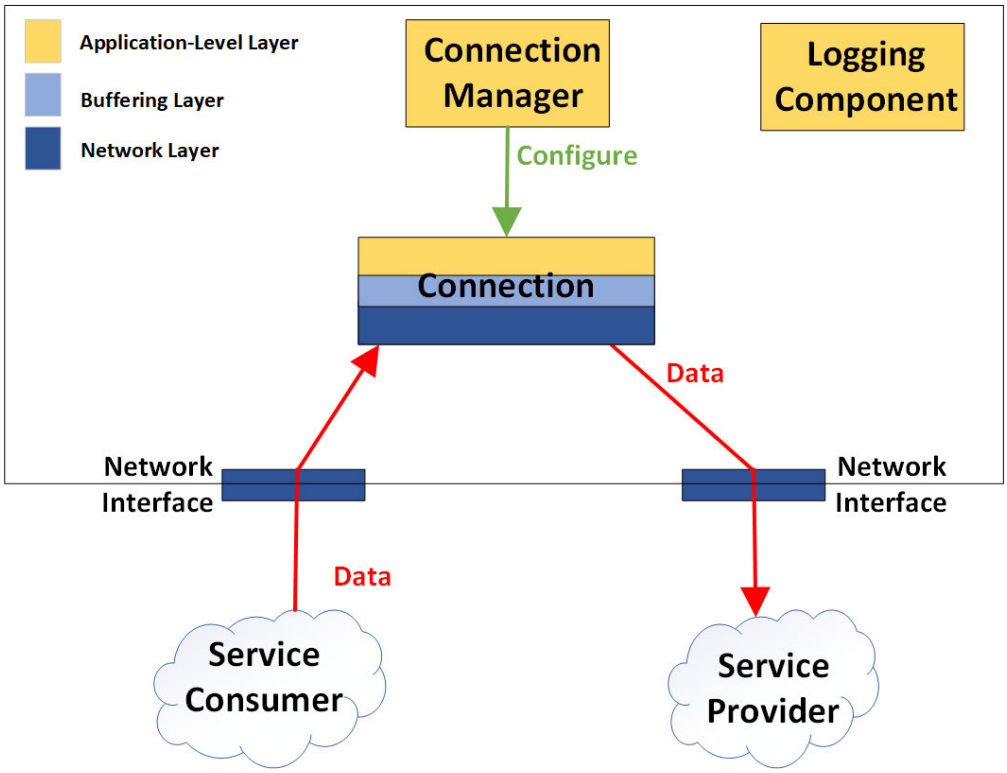

Figure 5.3: Prototype implementation overview (WSMP stack)

fic and a *CompositeBuffer* for outgoing traffic, a *ConnectionHandler* for incoming traffic and a *ConnectionHandler* for outgoing traffic writing to and reading from the buffers and an analyzer module for incoming traffic and an analyzer module for outgoing traffic, see figure 5.4 (page 73). The *ConnectionHandler* can either be an *SSLConnectionHandler* realising a TCP connection with SSL (on the IP stack) or a *WSMPConnectionHandler* based on raw sockets, which extracts the application data from WSMs encapsulated in Ethernet frames. The *ConnectionHandler* for incoming traffic writes the data to the

CompositeBuffer, which notifies the analyzer module whenever it transitions from empty to not-empty and vice versa, or from not-full to full and vice versa. The analyzer module reads the data from the *CompositeBuffer* and performs its analysis and if the data is valid, it orders the *ConnectionHandler* for outgoing traffic to read the data from the *CompositeBuffer* and send them to the destination, see figure 5.4. For traffic going the other way round, the *Connection* works analogously.

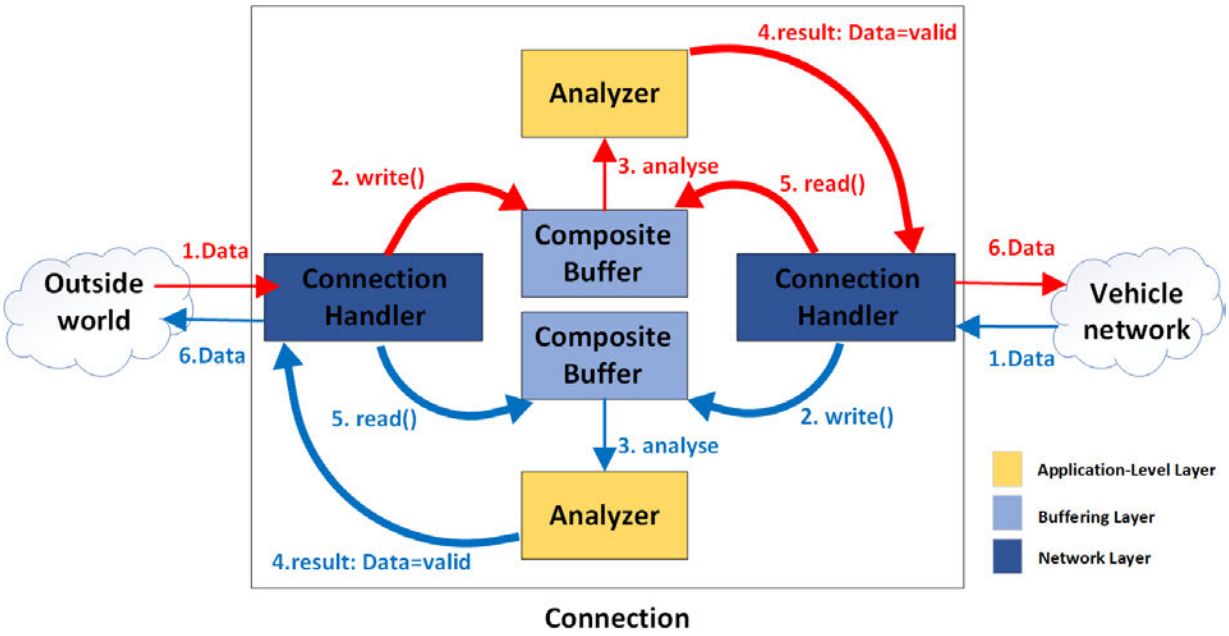

Figure 5.4: Connection over V2X Application-Level Gateway

The analyzer module has to implement the interface *IAnalysis* with an "*analyse*"-method (signature: "*bool analyse(input)*") allowing the integration into a *Connection* and can be tailored to a specific application. The analyzer module itself can be one single module or consist of multiple modules. Generally it consists of a set of rules and checks if the received data is valid or not, according to the defined set of rules. These rules can apply to the packet as a whole (e.g. size of the packet), the header or the payload. A rule, or a set of rules, can be encapsulated in a module implementing the *IAnalysis* interface and multiple modules can be combined in an arbitrarily complex Boolean function within the top-level analyzer module. This modular approach allows the flexible creation of complex analyses from simpler modules and facilitates maintainability and code re-use.
For an analysis of a (protocol-specific) header or the payload transported by an application protocol, e.g. HTTP, a protocol-specific extractor module for extracting the

relevant information from the message stored in the buffer is necessary, see figure 5.5, since the buffer only stores "raw data". For extracting specific parts of the payload, the

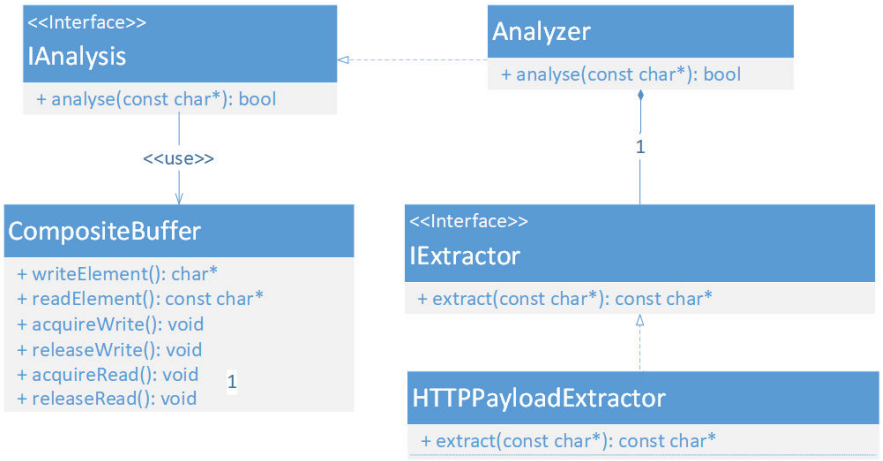

Figure 5.5: UML class-diagram: Extractor used by analyzer modules to extract relevant information

extractor also has to be application-specific. An analyzer module uses an extractor module, implementing the "*IExtractor*" interface, to correctly interpret the data stored in a *CompositeBuffer*. E.g. with the *HTTPPayloadExtractor* the payload of an HTTP message can be extracted (without copying) from a *CompositeBuffer* and then analysed by the analyzer module. For the MQTT protocol an extractor for extracting data from an MQTT message, e.g. the MQTT message type and packet identifier, was implemented, allowing the detection of MQTT message sequence violations by an analyzer module. For ETSI CAMs an extractor to access the data, e.g. the sending vehicle's last position, was implemented.

A *CompositeBuffer* consists of multiple modules, see figure 5.6 (page 75): a simple buffer module containing the buffer data accessible via an interface (*IDataBuffer*), a module for controlling the read- and write-indices of the buffer module and a module for fill-level control of the buffer. Components writing to and reading from a *CompositeBuffer* do not have to manage the read- and write-indices themselves. Instead, the access method (e.g. FIFO) is implemented in the *BufferAccessIndexControl* module (implementing the *IBufferAccessIndexControl* interface), which controls how the buffer module (implementing the interface *IDataBuffer*) stores the data. The *CompositeBuffer* accesses this data via the *IDataBuffer* interface using the indices from the *BufferAccessIndexControl* module. This modularisation facilitates flexibility, maintainability and code re-use.

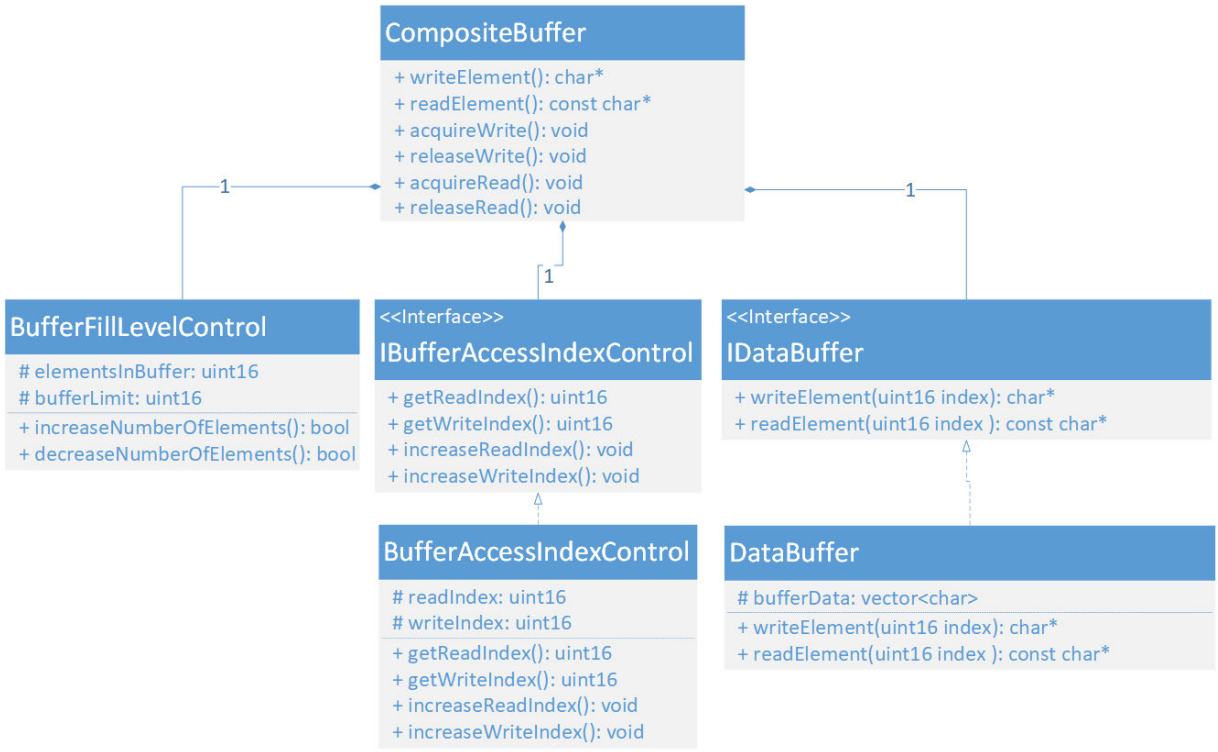

Figure 5.6: UML class-diagram: CompositeBuffer

5.2 Stateful Analysis

5.2.1 Context-sensitive Semantic Analysis

For a context-sensitive semantic analysis as described in chapter 4, section 4.2 the analyzer module has to know the relevant context, e.g. the vehicle's current state, at all time. In this prototype implementation the vehicle state is decomposed into separate subsystem states: an abstract vehicle state describing whether the vehicle is driving or not and the trunk state. The V2X Application-Level Gateway has a *ContextModule* for every subsystem holding the subsystem's current state, see figure 5.7 (page 76). It is notified of every state change in any of the subsystems, so that the *ContextModules* always hold the correct states.
An analyzer module can use the *ContextModules* to check if a given payload is valid in the current state. E.g. the analyzer module for the remote trunk control realising a *valid* function with tables for the trunk and the vehicle state, see tables 5.1 and 5.2 (page 76), maps a command to the column index of each table (e.g. "open" to index "2") and the

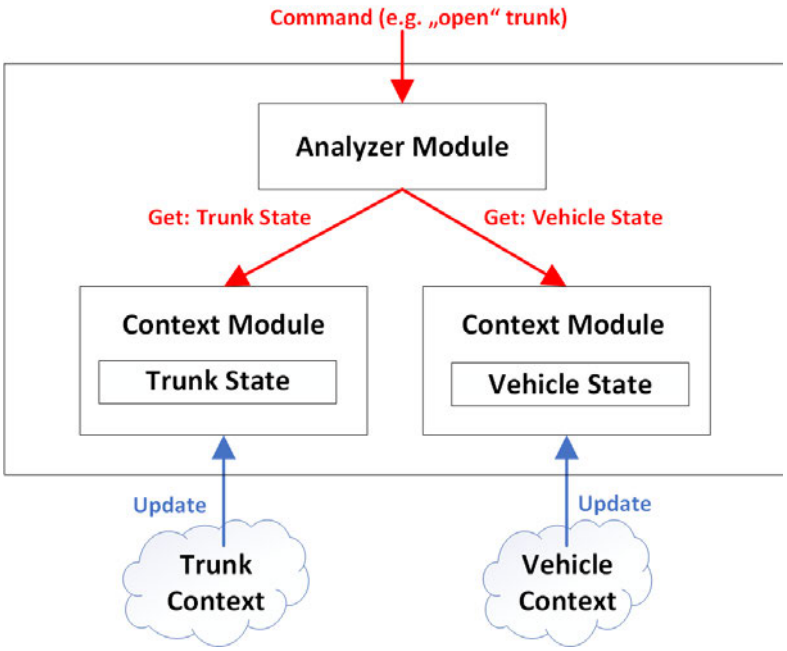

Figure 5.7: V2X ALG context modules

states from the *ContextModules* to the line indexes of the respective tables (e.g. ”Closed & Unlocked” to index ”1” and ”Stopped” to index ”0”) to determine whether the command is valid in the current state. Only when both tables return *true* for the given command it is classified as valid, otherwise it is invalid.

Table 5.1: Table for the *valid* function for a vehicle trunk

State / Command	unlock	lock	open	close
Closed & Locked	*true*	*false*	*false*	*false*
Closed & Unlocked	*false*	*true*	*true*	*false*
Open	*false*	*false*	*false*	*true*

Table 5.2: Table for the *valid* function for a vehicle trunk dependent on the vehicle's state

State / Command	unlock	lock	open	close
Stopped	*true*	*true*	*true*	*true*
Driving	*false*	*true*	*false*	*true*

The *ContextModules* hold the state of the vehicle and it is possible to add more modules, not only describing the vehicle itself, but also the state of the vehicle's environment, e.g. whether it is dark or not (day or night).
Normally, the main component for holding the state of the vehicle's environment is the

LDM, see chapter 2, section 2.2.3, which contains information on nearby objects (e.g. nearby vehicles' positions) and also information on road conditions, such as e.g. "slippery road". Since for this work no information on the vehicle's environment beyond ETSI CAMs is required and for the semantic analysis of ETSI CAMs it is sufficient to store the information from the ETSI CAMs in a local object list, see section 5.2.2, the LDM is not part of the implementation and substituted with the local object list.

5.2.2 Semantic Analysis of ETSI CAMs

The analyzer module for the semantic analysis of ETSI CAMs implemented for the prototype determines whether an ETSI CAM from a vehicle is valid, based on whether its data deviates from the expected values, see chapter 4, section 4.2.3. This is done exemplary for the vehicle's speed and position.
The ETSI CAM (see chapter 2, section 2.2.2) implementation for this work includes only selected elements necessary for exemplary realising the semantic analysis of ETSI CAMs for the prototype. Specifically, the mandatory elements, i.e. the ITS PDU header, the time-stamp, the Basic Container and the HF Container are included, with the HF Container containing the mandatory field "speed" (measured in centimetres per second). The position coordinates were implemented as *x,y-coordinates* of a 2-dimensional plane instead of the WGS 84 (World Geodetic System 1984) to simplify calculations.
For the semantic analysis of ETSI CAMs the analyzer module uses a *Physical Model* and a *Local Object List*, see figure 5.8. A list entry represents the last known state of a vehicle, i.e. its position and speed. Upon receiving a valid ETSI CAM the corresponding vehicle in the list is updated. The time-stamp from the ETSI CAM is also stored in the list entry.

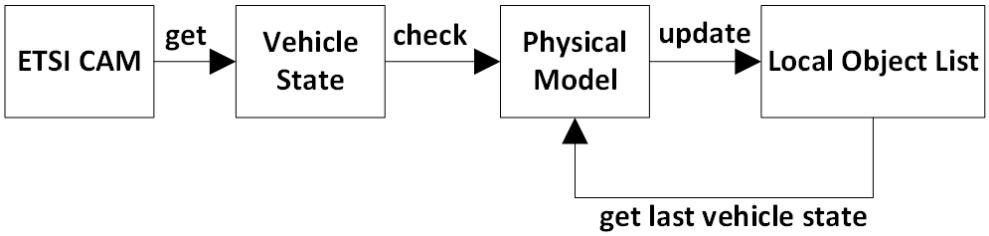

Figure 5.8: ETSI CAM analysis components

The *Physical Model* contains the thresholds resulting from the laws of physics and the technical limitations of road vehicles, which for this implementation corresponds to a

speed limit of *550 km/h*, which no road vehicle is expected to exceed in the near future and a maximum change in position between two consecutive vehicle states derived from that speed limit. To express the different levels of certainty as to whether an ETSI CAM is valid or not and to correctly assess a wider range of scenarios, besides the speed limit of *550 km/h* a threshold of *220 km/h* was added, see figure 5.9.

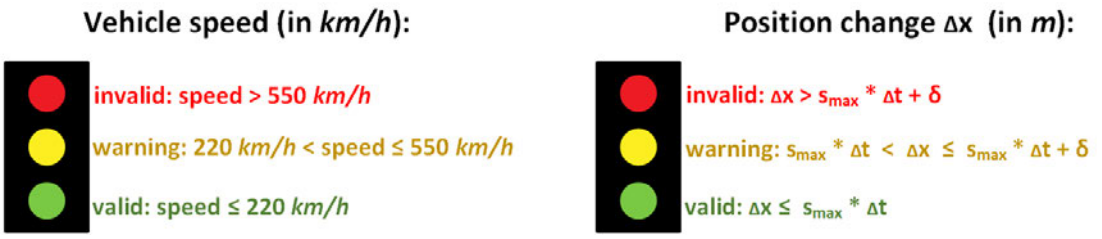

Figure 5.9: ETSI CAM analysis thresholds

While a speed of $>$*550 km/h* is impossible and an ETSI CAM containing such a speed is classified as invalid and dropped (and a malicious semantics attack is reported to the Automotive Cyber Defense Center), a speed of $>$*220 km/h* but $\leq$*550 km/h* is unusual, but possible e.g. when driving on a highway. An ETSI CAM with such a speed information, if otherwise valid, is classified as atypical but valid and forwarded with a warning sent to the Automotive Cyber Defense Center. An ETSI CAM containing a speed of $\leq$*220 km/h* is classified as valid and forwarded normally.
With the reception of every ETSI CAM the change in position $\triangle$x is calculated using the position information from the ETSI CAM and the last known position from the *Local Object List*. Also the time elapsed between the update of the last known position and the current position, $\triangle$t, is calculated using the time-stamps in the *Local Object List* and the received ETSI CAM. The current speed and the speed in the previous state are obtained from the ETSI CAM and the *Local Object List* respectively. With this information it is checked whether the position and speed information of an ETSI CAM are consistent with the physical model defined in chapter 4, section 4.2.3, i.e. "$\triangle x \leq s_{\max} \cdot \triangle t$". ETSI CAMs that are consistent with the physical model are classified as valid and forwarded normally. When an ETSI CAM is found to be inconsistent with the physical model it is classified as invalid and dropped and a malicious semantics attack is reported to the Automotive Cyber Defense Center. To account for a possible inaccuracy of the speed and position data, a configurable tolerance δ is used, see figure 5.9. An ETSI CAM with data inconsistent with the physical model, but not exceeding the threshold including the tolerance, is classified as atypical but valid and forwarded with a warning sent to the Automotive Cyber Defense Center.

5.2.3 Protocol Message Sequence Violation

Exemplary modules, one for incoming and one for outgoing traffic, for detecting protocol message sequence violations have been implemented for the MQTT protocol, following the concept described in chapter 4, section 4.3. The *valid* function for both incoming broker-to-client and outgoing client-to-broker traffic was realised with the respective tables (see table 5.3 and table 5.4 on page 79). The message type of the analysed MQTT message is mapped to a column index (e.g. *CONNECT* to index "0") and the current protocol state is mapped to a line index (e.g. "Unconnected" to index "0") resulting in a Boolean value saying whether messages with the given message type (from either an MQTT client or broker) are valid in the current protocol state or not.

Table 5.3: Table for the *valid* function for the MQTT protocol (client-side)

Message / State	Unconnected	Connecting	Connected
CONNECT	*true*	*false*	*false*
CONNACK	***false***	***false***	***false***
PUBLISH	*false*	*false*	*true*
PUBACK	*false*	*false*	*true*
PUBREC	*false*	*false*	*true*
PUBREL	*false*	*false*	*true*
PUBCOMP	*false*	*false*	*true*
SUBSCRIBE	*false*	*false*	*true*
SUBACK	***false***	***false***	***false***
UNSUBSCRIBE	*false*	*false*	*true*
UNSUBACK	***false***	***false***	***false***
DISCONNECT	*false*	*false*	*true*

For messages that in addition to their message type also need their packet ID checked to determine whether they are valid (e.g. *PUBACK* or *SUBACK*), additionally the *Client to Broker List* or *Broker to Client List* is checked. So for e.g. a *PUBACK* message first the table checks whether a message with the message type *PUBACK* is generally allowed in the current MQTT protocol state (e.g. "Connected") and if it is, the list is used to check whether the *PUBACK* message is valid based on its packet ID.
There is a *Client to Broker List* and *Broker to Client List* (each including the (sub-) lists required for messages with *QoS 2*) for every client ID to allow continuous operation in the event of an MQTT connection breaking down and the client re-connecting with the broker (over the V2X Application-Level Gateway) and resuming communication where it broke off.

Table 5.4: Table for the *valid* function for the MQTT protocol (broker-side)

Message / State	Unconnected	Connecting	Connected
CONNECT	***false***	***false***	***false***
CONNACK	*false*	*true*	*false*
PUBLISH	*false*	*false*	*true*
PUBACK	*false*	*false*	*true*
PUBREC	*false*	*false*	*true*
PUBREL	*false*	*false*	*true*
PUBCOMP	*false*	*false*	*true*
SUBSCRIBE	***false***	***false***	***false***
SUBACK	*false*	*false*	*true*
UNSUBSCRIBE	***false***	***false***	***false***
UNSUBACK	*false*	*false*	*true*
DISCONNECT	***false***	***false***	***false***

The lists and the protocol state are held in a separate component, the MQTT state module, used by both the broker-to-client and client-to-broker modules to analyse the MQTT messages, see figure 5.10.

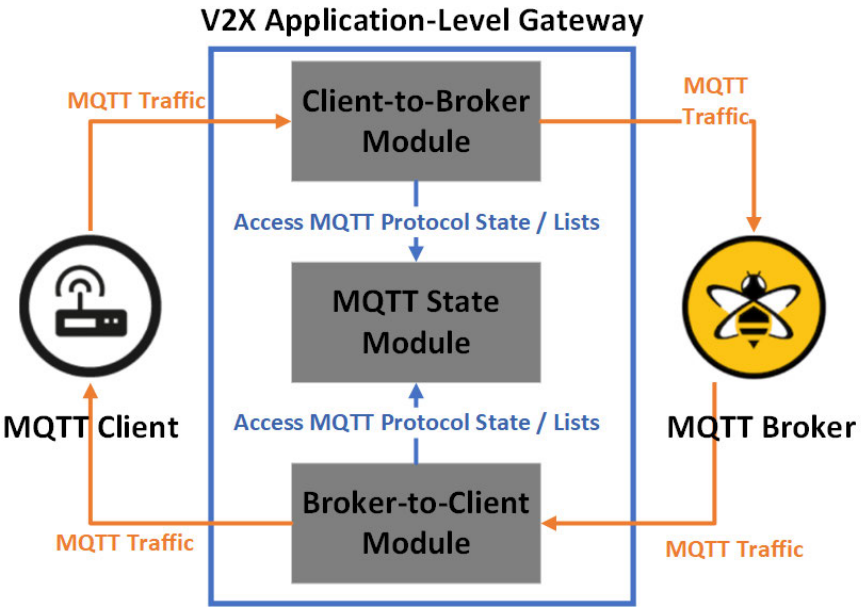

Figure 5.10: MQTT protocol analysis components

The MQTT protocol state is modelled with a finite state machine (FSM) using an efficient implementation of the state pattern [94]. In this state pattern implementation each

state of an FSM is a separate *struct* derived from a common base *struct*, see listing 5.1. Each state is identified by a unique *state code* and has to implement all possible transitions to other states. The change of states is realised with the placement operator *new*. The state pattern was chosen for its clarity, extensibility and maintainability. The state is updated with every valid incoming or outgoing MQTT message.

Listing 5.1: State pattern: MQTT protocol

```
struct IMQTTProtocolState {
  ///Unique MQTT state code for the state (used as valid-table line index).
  uint8_t stateCode;

  //Process the message CONNECT.
  virtual void signalConnect() = 0;

  //Process the message CONNACK.
  virtual void signalConnack() = 0;
  ...
  //Process the message DISCONNECT.
  virtual void signalDisconnect() = 0;
};

class MQTTAbstractFSM{
protected:

  ///MQTT protocol "Unconnected" state. This is the initial state.
  struct Unconnected: public IMQTTProtocolState{

    ///Constructor.
    Unconnected(){
      std::cout<<"Unconnected"<<std::endl;
      stateCode = 0;
    }

    //Transition to "Connecting".
    void signalConnect() override {
      new(this) Connecting;
    }

    //Invalid!
```

```
    void signalConnack() override {}

    ...

    //Self transition: already disconnected.
    void signalDisconnect() override {}
  };


  //Remaining states: ''struct Connecting'', ''struct Connected''.
  ...

  //The current state.
  Unconnected protocolState;

  //Pointer to current state.
  IMQTTProtocolState *statePointer;


public:

  //Constructor.
  MQTTAbstractFSM():statePointer(&protocolState){}

  //Delegates the message CONNECT to the current state.
  void signalConnect(){
    statePointer->signalConnect();
  }

  //Delegates the message CONNACK to the current state.
  void signalConnack(){
    statePointer->signalConnack();
  }

  ...
};
```

5.3 Service Registration with the V2X Application-Level Gateway

To communicate via the V2X Application-Level Gateway services running in the in-vehicle network have to register with it. This is done via Remote Method Invocation (RMI) over a specified interface, see figure 5.11. A remote client, e.g. running on an ECU, can register V2X services with the V2X Application-Level Gateway by calling the "*registerService*"-method of the contract interface "*IServiceRegistration*" implemented by a *V2XServiceRegistration* stub, which takes a service's name, version, role (service provider or consumer), provider address (in case of IP-based services the provider IP), provider port and application layer protocol (e.g. HTTP) as parameters.
A skeleton on the V2X Application-Level Gateway's side then does the unmarshalling and calls the remote implementation registering the V2X service with the V2X Application-Level Gateway. This registration is done via HTTPS to offer some degree of cryptographic security.

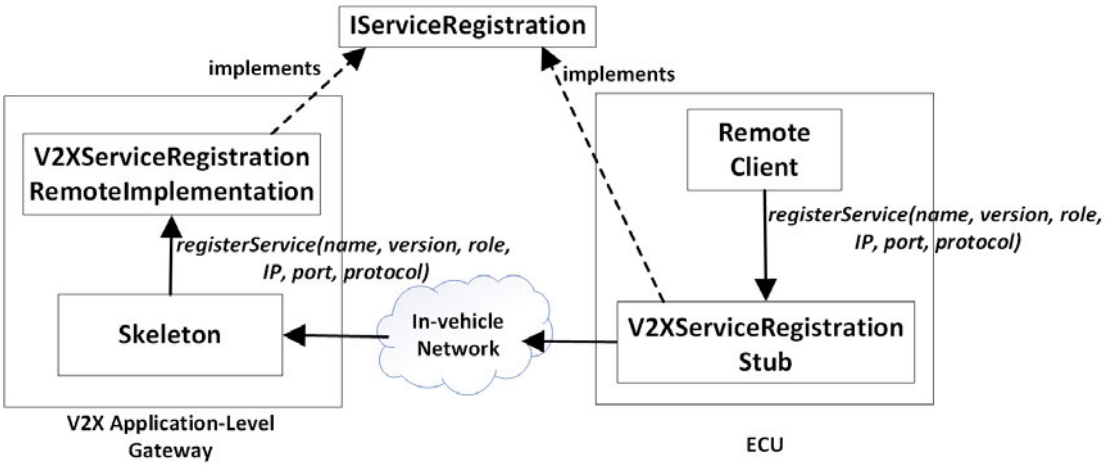

Figure 5.11: V2X service registration via RMI

6 Evaluation

In this chapter the developed prototype implementation of the V2X Application-Level Gateway is evaluated. It is shown that the V2X Application-Level Gateway protects the vehicle from attacks via V2X identified in chapter 3, section 3.2: attacks based on malicious semantics, application layer protocol message sequence violations, application layer DoS attacks and buffer overflow attacks.
The attacks are executed exemplary for the following application layer protocols: HTTP, MQTT and the ETSI Cooperative Awareness Basic Service. The following applications, developed for the evaluation, were attacked: an application for remotely controlling the vehicle trunk via HTTPS, one for receiving and displaying subscribed traffic updates (e.g. congestion warnings) via MQTT and a simple V2V traffic safety service using ETSI CAMs.

6.1 Overview

For the evaluation the prototype implementation of the V2X Application-Level Gateway was deployed in a test network representing an internal vehicle network, see figure 6.1, page 85. The network consists of an *Edgecore SDN switch* which connects multiple *Intel NUCs* and *Raspberry Pis* representing vehicle ECUs. With this network the use of Ethernet technology and Software-defined networking (SDN) in combination with security mechanisms, like e.g. anomaly detection, in future vehicle networks is evaluated by the CoRE research group [24] as part of the SecVI project [105]. The goal is a low-complexity, robust and secure in-vehicle network satisfying the requirements of future vehicles like sufficient bandwidth and connectivity.
The V2X Application-Level Gateway is part of such a network protecting it as a first line of defence against attacks from the outside world. The HTTPS remote trunk control, the MQTT traffic update service and the V2V traffic safety service using ETSI CAMs were also deployed in this network. After successful evaluation, the internal vehicle network

developed by the CoRE research group is deployed in the SecVI demonstration vehicle, see figure 6.2, page 86.
For the evaluation of the V2X Application-Level Gateway, application layer attacks via V2X on these applications were simulated. When the V2X Application-Level Gateway classifies a message as invalid, which indicates an attack, the policy defined for the prototype is to drop the message and report the attack to the Automotive Cyber Defense Center (see chapter 5, section 5.1).

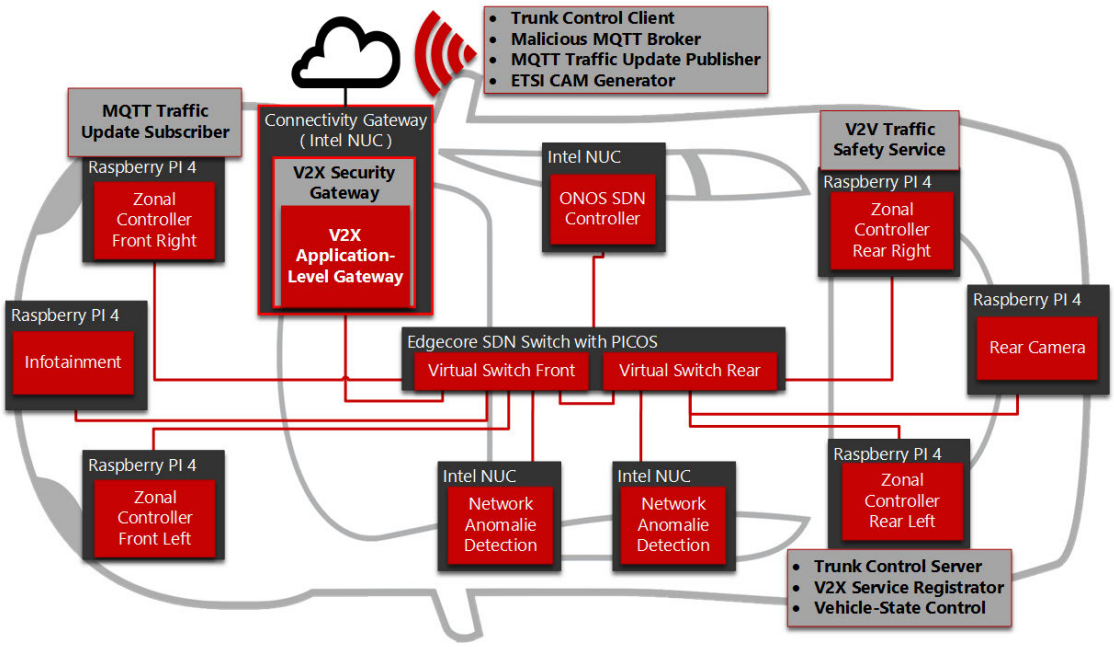

Figure 6.1: In-vehicle test network

Besides the V2X Application-Level Gateway prototype the following 9 components were implemented and deployed in the evaluation:

- **Trunk Control Server**: allows a remote client to control a simulated vehicle trunk via a set of defined commands.
- **Trunk Control Client**: for remotely controlling a vehicle trunk via a set of defined commands.
- **V2X Service Registrator**: allowing to register V2X services via HTTPS with the V2X Application-Level Gateway.

- **Vehicle-State Control**: holds the vehicle state (”driving” or ”stopped”) and notifies registered observers upon any state change.
- **MQTT Traffic Update Publisher**: for publishing traffic updates (e.g. congestion warnings).
- **MQTT Traffic Update Subscriber**: for receiving and displaying traffic updates.
- **Malicious MQTT Broker**: a rudimentary MQTT broker that can be configured to violate the MQTT protocol message sequence.
- **ETSI CAM Generator**: generates and sends ETSI CAMs coming from multiple (simulated) sources: an attacker and several regular vehicles.
- **V2V Traffic Safety Service**: receives ETSI CAMs and displays safety-relevant information on the nearby objects (position and current speed).

Figure 6.2: SecVI project demonstration vehicle

The trunk control server allows a remote client to connect and control the simulated trunk with the following set of simple commands: ”unlock”, ”open”, ”close” and ”lock”. It holds the state of the trunk (possible states: ”closed and locked”, ”closed and unlocked” and ”open”), which changes upon receiving the appropriate command, see figure 6.3 (page 87). Multiple observers can register to be notified via HTTPS by the trunk control

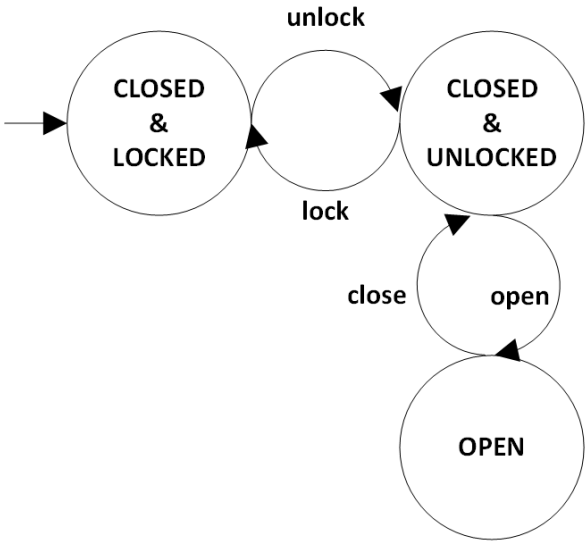

Figure 6.3: Trunk control state machine implemented by the trunk control server

server upon any state change. The trunk control client connects to a trunk control server and sends the commands allowing remote trunk control to it. Additionally the trunk control client can send malformed payloads, i.e. invalid commands. The communication between trunk control server and client is via HTTPS. The vehicle-state control holds the state of the vehicle (possible states: ”driving” and ”stopped”), which can be switched via keyboard. Multiple observers can register to be notified via HTTPS by the vehicle-state control upon any state change.
The MQTT traffic update publisher publishes traffic updates, such as congestion warnings, via the malicious MQTT broker. The MQTT traffic update subscriber can subscribe to traffic updates and displays the received updates. The malicious MQTT broker allows clients to publish and subscribe and can be configured to violate the MQTT protocol, e.g. by sending a *SUBACK* message without a prior *SUBSCRIBE* message from a client.
The ETSI CAM generator can be configured to generate and send ETSI CAMs coming from multiple (simulated) sources: an attacker and several regular vehicles. An ETSI CAM contains basic information from the sending entity such as its ID, the latest position and its current speed. The ETSI CAMs of the regular vehicles are generated based on a configurable initial position and a constant driving speed. For the attacker the ETSI CAM generator can generate ETSI CAMs with either typical data, atypical data (e.g.

very high, but still realistic, speed), or malicious semantic (i.e. invalid data). It can generate 2 types of ETSI CAMs with malicious semantics:

- ETSI CAMs that ignore the technical limitations of modern vehicles containing values that exceed values imaginable for road vehicles in the near future, e.g. a vehicle speed greater than *550 km/h*, which can be detected using stateless analysis.
- ETSI CAMs that contradict the laws of physics or ignore the technical limitations of modern vehicles by containing values that deviate too much from the values possible based on the previous state of the object, e.g. when the distance between the position from the ETSI CAM and the previous position exceeds the distance possible when driving with maximum speed. Detecting these ETSI CAMs requires stateful analysis.

The V2V traffic safety service receives ETSI CAMs from the ETSI CAM generator and displays safety-relevant information on the nearby simulated vehicles (position and current speed).
The trunk control client, the MQTT traffic update publisher and the ETSI CAM generator can be configured to perform both application layer DoS attacks and buffer overflow attacks, i.e. sending a message exceeding the size allowed by a given application. The trunk control server, V2X service registrator and vehicle-state control were run on the *Zonal Controller Rear Left*, the MQTT traffic update subscriber was run on the *Zonal Controller Front Right* and the basic V2V traffic safety service was run on the *Zonal Controller Rear Right*, see figure 6.1, page 85. The trunk control client, MQTT traffic update publisher, malicious MQTT broker and ETSI CAM generator were run outside the in-vehicle network.
Figure 6.4, page 89, shows the general setup of the described components. With this setup the V2X Application-Level Gateway was evaluated by analysing the traffic in the following simulated V2X scenarios including attacks. The general application layer attacks identified in chapter 3, section 3.2 are:

- buffer overflow attacks
- malformed messages
- messages violating the application layer protocol message sequence
- messages with malicious content
- application layer DoS attacks

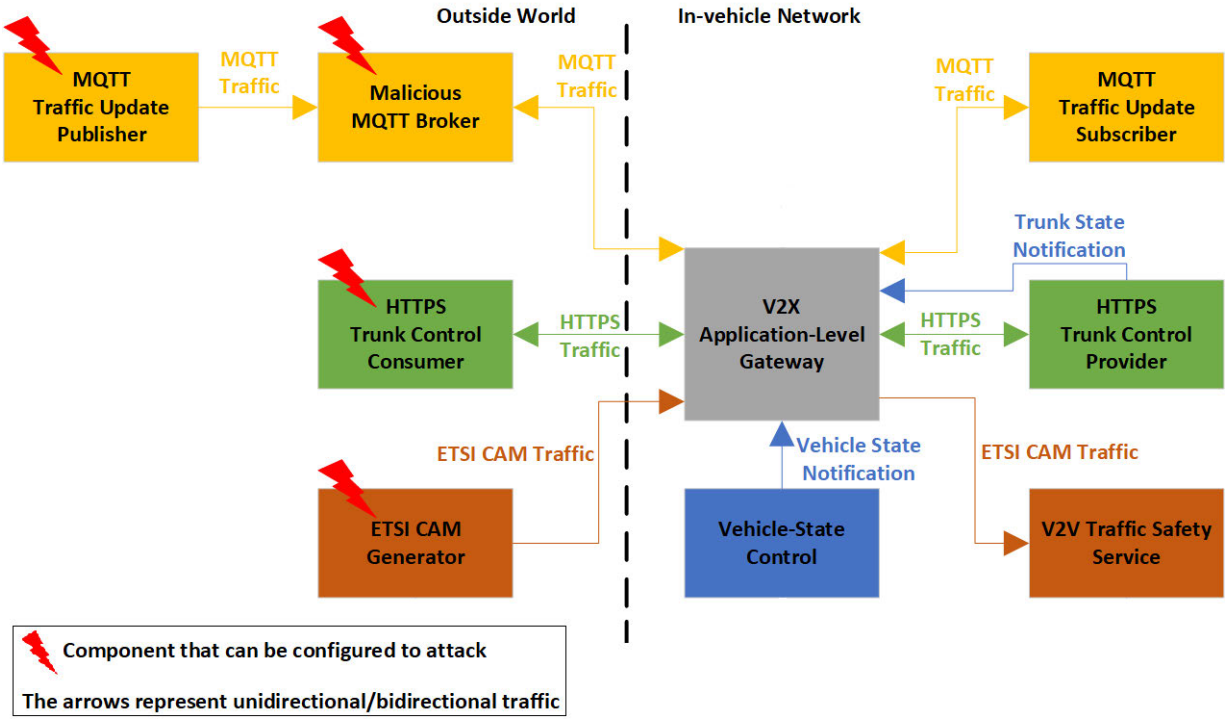

Figure 6.4: Schematic setup for the evaluation of the V2X Application-Level Gateway

Since this work focuses on the semantic analysis of application data, attacks based on malicious semantics were picked as representatives of messages with malicious content. To cover attacks on the application layer protocol itself and not only attacks on application data, a violation against the MQTT protocol message sequence is included, since of the used application layer protocols, MQTT is most suitable for such attacks. Application layer DoS attacks are included due to being an attack class distinct from malicious messages because of their attack pattern, i.e. a flood of messages harmless in themselves as opposed to a single harmful message. For a broader coverage of attacks based on malicious messages, buffer overflow attacks are also included.

6.2 Attacking the HTTPS Remote Trunk Control Service

The remote control of the vehicle trunk via HTTPS was simulated and attacked. The attacks in this scenario were:

1. attacks based on malicious semantics requiring a context-sensitive semantic analysis
2. buffer overflow attacks, here, messages with a payload that exceeds the maximum payload size allowed by the application
3. an application layer DoS attack

The V2X Application-Level Gateway was tested by sending commands to it from the trunk control client. To define the test classes for the evaluation of the context-sensitive semantic analysis of a remote trunk control, the possible input, in the form of a received event σ in an arbitrary (non-error) trunk state, was classified based on the outcome of the *valid* function defined in chapter 4, section 4.2.2. This is done for both the vehicle trunk viewed as an isolated system and as part of a driving vehicle, see figure 6.5. When the vehicle trunk (viewed as an isolated system) is in an arbitrary (non-error) state s and receives an event σ, this event can either be in the set of known trunk commands, i.e. be an element of the trunk alphabet Σ ($\sigma \in \Sigma$), or it can be an unknown command ($\sigma \notin \Sigma$), for which the *valid* function $valid(s, \sigma)$ is always $false$ (test class *4*). An event $\sigma \in \Sigma$ is a valid command when $valid(s, \sigma)$ is $true$, which is the case when σ triggers a transition to a valid trunk state, i.e. $\delta(s, \sigma) \neq \emptyset$. When an event $\sigma \in \Sigma$ does not trigger a transition to a valid trunk state, $valid(s, \sigma)$ is $false$ and σ is an invalid command, regardless of the context the trunk operates in (test class *2*).

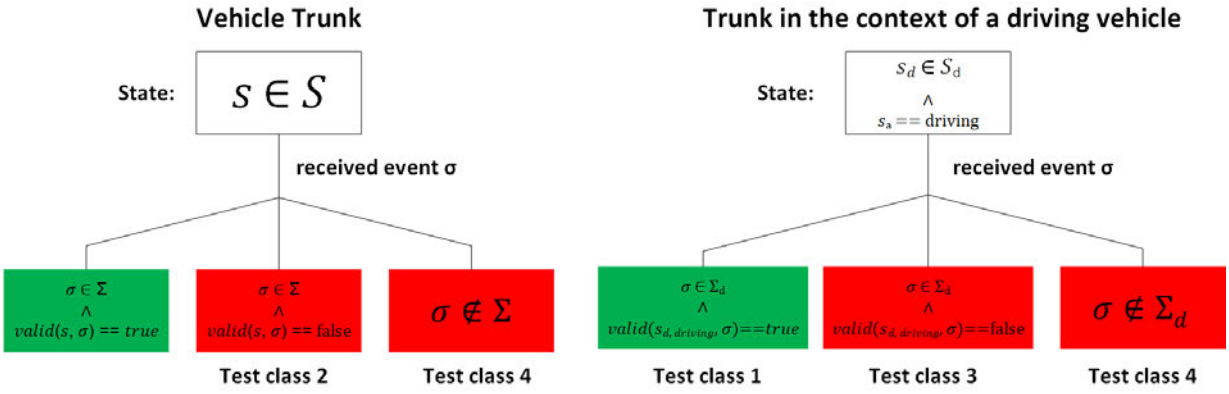

Figure 6.5: Defining test classes for the evaluation of the semantic analysis of a remote trunk control

The trunk, as part of a vehicle, is a dependent system (where the set of states is called S_d and the alphabet Σ_d), which is affected by the vehicle (an affecting system with a set of states S_a and an alphabet Σ_a). I.e. the trunk's behaviour also depends on the

state of the vehicle $s_a \in S_\mathrm{a}$. When the vehicle is in driving state ($s_a ==$ driving), the trunk's behaviour differs from its behaviour as an isolated system, e.g. it cannot be opened. When the vehicle is not in driving state, the trunk behaves in the same way it would behave as an isolated system, which is why for the further definition of test classes only the driving state is considered. When the trunk is in an arbitrary (non-error) state s_d and the vehicle is driving, which corresponds to a system state $s_{\mathrm{d,driving}}$, a received event σ can be an element of the trunk alphabet Σ ($\sigma \in \Sigma_\mathrm{d}$), or it can be an unknown command ($\sigma \notin \Sigma_\mathrm{d}$), for which the *valid* function $valid(s_{\mathrm{d,driving}}, \sigma)$ is always $false$ (test class *4*). An event $\sigma \in \Sigma_\mathrm{d}$ is a valid command when $valid(s_{\mathrm{d,driving}}, \sigma)$ is $true$ (test class *1*) and invalid when $valid(s_{\mathrm{d,driving}}, \sigma)$ is $false$ (test class *2*). Summarising, whether a known command is valid, depends on both the trunk and vehicle state, while invalid commands can be distinguished into commands being invalid depending on both the trunk and vehicle state, or just the trunk state. So for this evaluation 4 test classes of commands were defined:

1. valid commands
2. commands invalid independently from the context, such as the "lock"-command in "open"-state
3. commands invalid only in a certain context, such as the "open"-command in "closed and unlocked"-state with the vehicle in "driving"-state
4. unknown commands, such as "xyz", which are invalid per se

These test classes cover all possible attacks on the semantics of remotely controlling a vehicle trunk. Buffer overflow attacks and an application layer DoS attack are also performed. The V2X Application-Level Gateway is expected to forward all valid commands to the trunk control server and drop all invalid commands. It is also expected to drop the buffer overflow attack messages and all application layer DoS messages during such an attack. Also, all attacks should be reported to the Automotive Cyber Defense Center. In the tests commands from all 4 test classes were sent with the vehicle either stopped or in driving state, see the sequence diagram in figure 6.6, page 92.
First the V2X Application-Level Gateway registered itself with the trunk control server and vehicle-state control to receive notifications upon state changes, so that it holds the vehicle state correctly at all time, which is necessary for the context-sensitive semantic analysis. Then the V2X service registrator registered the remote trunk control service with the V2X Application-Level Gateway, so that it can serve as a proxy for the

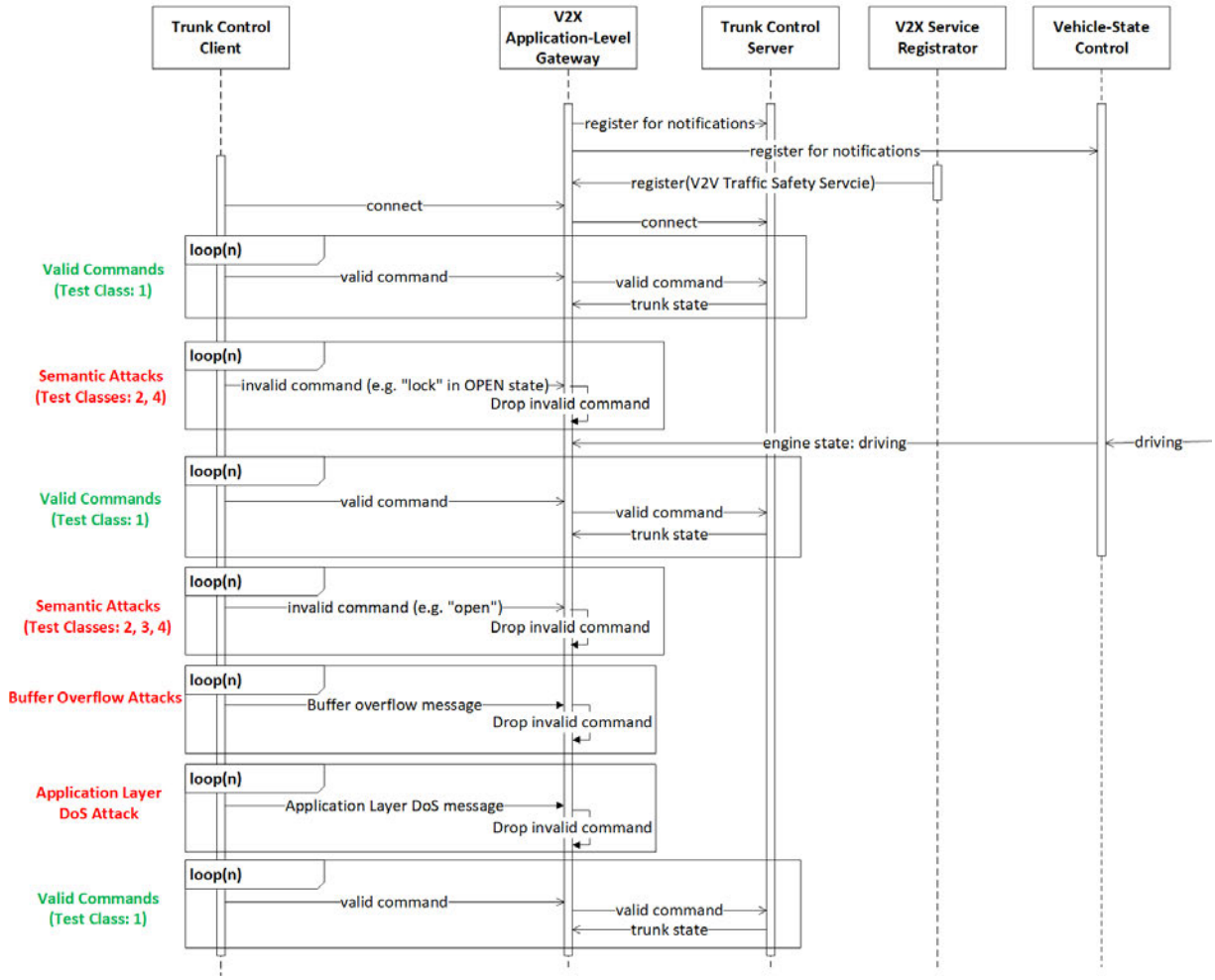

Figure 6.6: Sequence of the evaluation in the remote trunk control scenario

trunk control server and client. Next, the trunk control client connected to the V2X Application-Level Gateway, which proactively established a connection to the trunk control server. Now the trunk control client can remotely control the vehicle trunk via the V2X Application-Level Gateway.

First, valid commands (test class *1*) were sent with the vehicle state being ”stopped”. They were all forwarded, which is the expected correct behaviour. Next, some invalid commands, such as the ”lock”-command in ”open”-state and the ”open”-command in ”closed and locked”-state (test class *2*) and unknown commands (test class *4*), were sent, with the vehicle state being ”stopped”. They were all dropped, which is the expected correct behaviour.

Then the vehicle state was switched to ”driving”. First valid commands (test class *1*) were sent and then invalid commands were sent. Of the invalid commands some were unknown (test class *4*), invalid independently from the context, such as the ”lock”-command in ”open”-state (test class *2*), while others were only invalid with the vehicle in ”driving”-state, such as the ”open”-command in ”closed and unlocked”-state (test class *3*). The valid commands were all forwarded, while all invalid commands were dropped, which is the expected correct behaviour. Also, all attacks were reported to the Automotive Cyber Defense Center.
Next, buffer overflow attacks and an application layer DoS attack were launched, see the sequence diagram in figure 6.6, page 92. First, 400 buffer overflow messages were sent, which were dropped by the V2X Application-Level Gateway, see figure 6.7. Then an application layer DoS attack was launched, sending an amount of commands per second, which exceeds the number of messages per second allowed for the remote trunk control. Here 10 was configured as a maximum threshold, which is seen as a realistic value still covering the scenario of a panicked user pressing the button multiple times, which represents the highest possible valid traffic rate. The DoS rate was 20 messages per second and 1000 DoS messages were sent (resulting in an attack duration of ∼50 seconds). After some time valid messages were sent again. The V2X Application-Level Gateway dropped all DoS messages during the DoS attack, see figure 6.7. It forwarded all valid messages after the application layer DoS attack. Also, the buffer overflow and application layer DoS attacks were reported to the Automotive Cyber Defense Center.

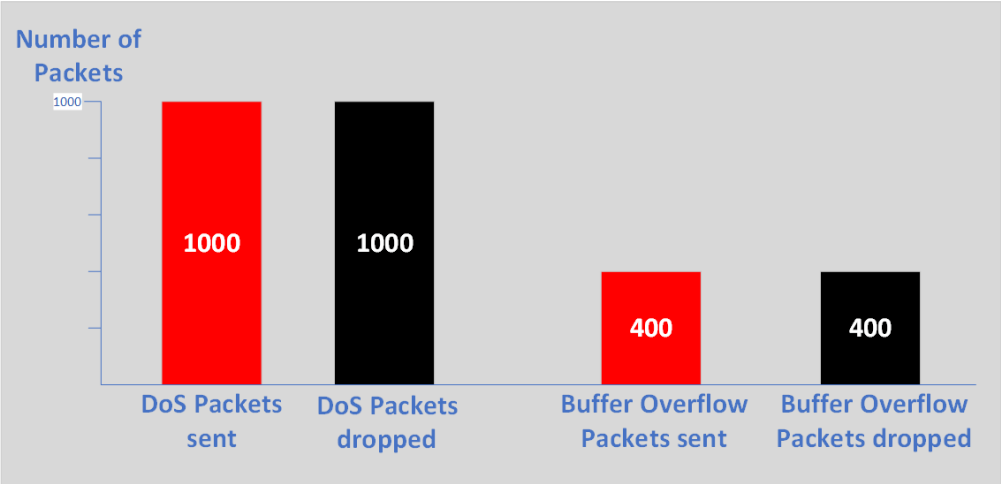

Figure 6.7: Sent and dropped application layer DoS and buffer overflow messages

6.3 Attacking the MQTT Traffic Update Service

A service for traffic updates via MQTT was simulated. The attacks in this scenario were:

1. attacks based on protocol message sequence violation requiring a stateful analysis
2. buffer overflow attacks, here, messages with a payload that exceeds the maximum payload size allowed by the application
3. an application layer DoS attack

A buffer overflow attack does not necessarily have to be realised via the payload. E.g. with MQTT *PUBLISH* messages an attacker could insert a topic with a length exceeding the maximum length the MQTT client (or broker) can handle. The topic is always preceded by a field declaring the length of the topic and any receiver will only process the amount of bytes specified in that topic length field, with most implementations probably checking if the value of the topic length field exceeds the topic length the receiver can handle. For the payload, no such field declaring the length exists, making it more likely for an implementation not to check the payload size than not to check the topic length. This higher probability of a successful buffer overflow attack via the payload is the reason why it was chosen as a representative of buffer overflow attacks.
From the perspective of an attacker the MQTT messages most suited for an application layer DoS attack against an MQTT client are *PUBLISH* messages with *QoS 2*, since additionally to a payload the client might process, it is required to acknowledge the message with a *PUBREC* and then await an acknowledgement for that *PUBREC*, which again has to be acknowledged. But from the perspective of the V2X Application-Level Gateway prototype implementation, which realises an application layer DoS detection based solely on the traffic rate, it makes no difference whether an application layer DoS attack is realised with *PUBLISH* messages with *QoS 2* or *QoS 0*, since incoming messages are classified as DoS traffic based only on their traffic rate. This is why for this evaluation the application layer DoS attack was realised with *PUBLISH* messages with *QoS 0*.
To define the test classes for the evaluation of the detection of message sequence violations, the MQTT protocol message sequence was divided into sub-sequences. Each sub-sequence consists of a number of consecutive MQTT messages (e.g. sub-sequence: $\{CONNECT, CONNACK\}$ consisting of 2 messages) and there is no overlapping of the sub-sequences, i.e. for 2 sub-sequences the intersection is $\emptyset$ (e.g. $\{CONNECT, CONNACK\} \cap \{PUBLISH\ (QoS1), PUBACK\} = \emptyset$). Every sub-sequence covers one of the

automatons to model the MQTT protocol defined in chapter 4, section 4.3, figure 4.13 (page 60). To avoid overlapping, the top-level automaton (see chapter 4, section 4.3, figure 4.12 (page 59) is redefined as a set of smaller automatons, see figure 6.8.

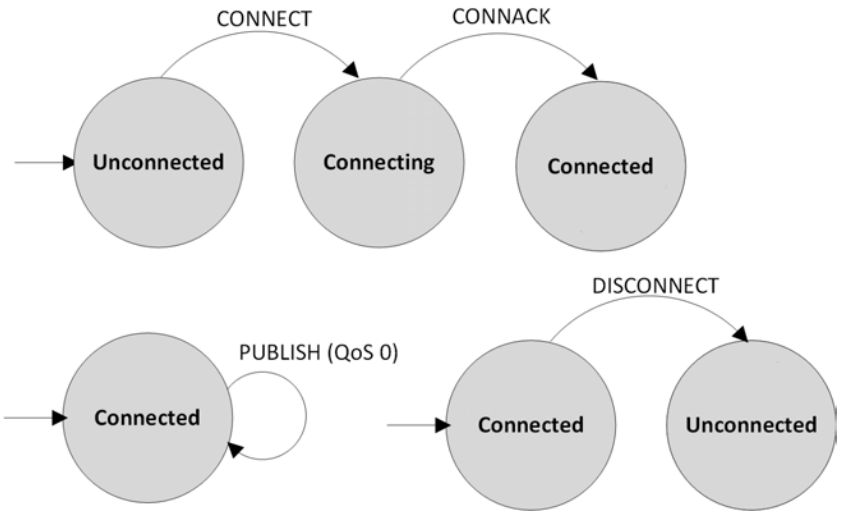

Figure 6.8: MQTT top-level automaton redefinition for sub-sequence automatons

For every MQTT message m_i in a sub-sequence it is checked for every possible subsequent message m_{i+1} whether m_{i+1} violates the defined protocol sequence, see figures 6.9 to 6.11, pages 95 and 96 (the colour red indicates a sequence violation, the colour green indicates adherence to the protocol sequence). A message m_{i+1} violates the defined protocol sequence when it does not correspond to a transition in the respective automaton from chapter 4, section 4.3 (or in case of the top-level automaton its redefinition in figure 6.8). For the *CONNECT* message (m_0) the subsequent message (m_1) can either be *CONNACK* or another message, which would be a sequence violation, because in this case only *CONNACK* corresponds to a transition in the respective automaton, see figure 6.8.

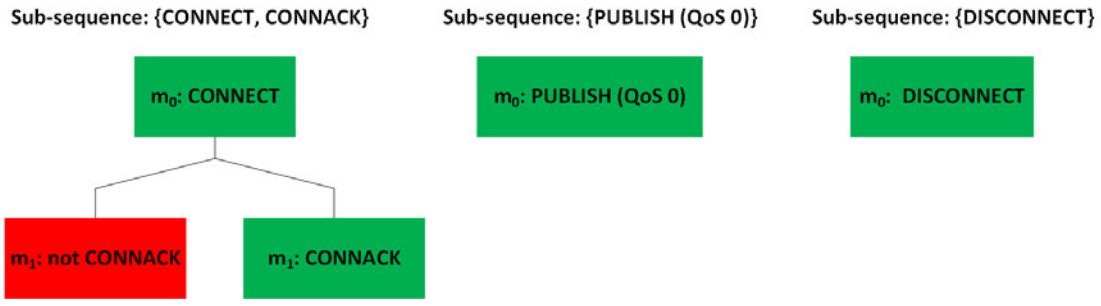

Figure 6.9: Defining test classes for the evaluation of message sequence violation detection

Analogously for e.g. a *PUBLISH* (*QoS1*) message (m_0) with a packet identifier x (ID: x), the subsequent message (m_1) can either be a *PUBACK* with a matching packet identifier (ID $= x$), or an ID $\neq x$, or another message. In this case the latter two are sequence violations, since neither corresponds to a transition in an automaton, with ID $\neq x$ being an ID for which there is no outstanding acknowledgement.

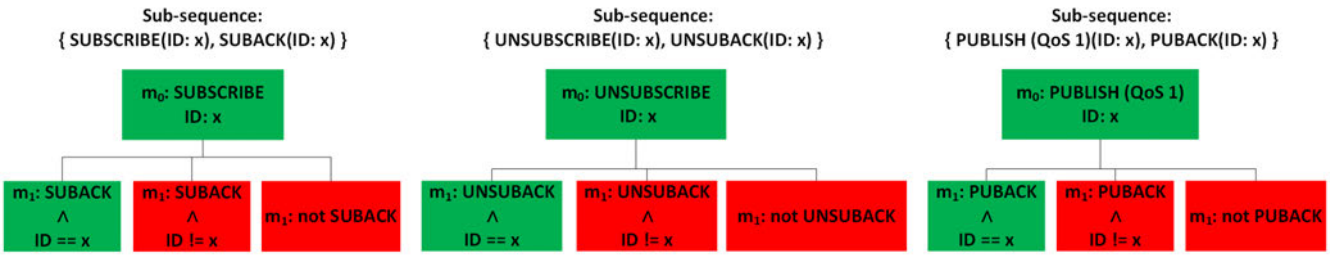

Figure 6.10: Defining test classes for the evaluation of message sequence violation detection

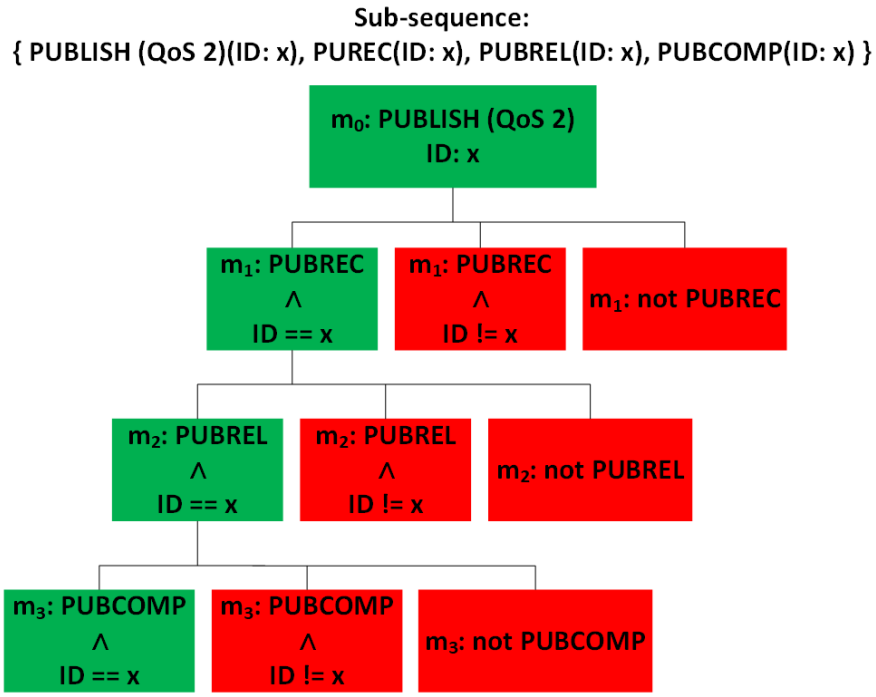

Figure 6.11: Defining test classes for evaluating message sequence violation detection

Note that the goal of the above analysis is to derive test classes for the evaluation of the detection of protocol message sequence violations defined in chapter 4, section 4.3.

Therefore, to derive test classes, every sub-sequence (and the corresponding automaton) is treated separately here. So e.g. the correct MQTT sequence, where after a *SUBSCRIBE* has been sent, a *PUBLISH QoS1* is sent and after that the corresponding *PUBACK* and *SUBACK* is received, is divided into the two parallel sub-sequences here: *SUBSCRIBE* and corresponding *SUBACK* on the one hand and *PUBLISH QoS1* and the corresponding *PUBACK* on the other hand.
Since a violation by the deliberate lack of a reply by the broker cannot be detected by the V2X Application-Level Gateway, it is not considered in this evaluation. Also, since this work classifies messages with unmatched topics as messages with malicious content and not protocol message sequence violations, they are not covered in the evaluation.
For the evaluation of detecting protocol message sequence violations 4 test classes of messages were defined:

1. messages adhering to the defined sequence classified based on their message type only (e.g. *CONNECT*)
2. messages adhering to the defined sequence classified based on both their message type and packet ID (e.g. a *SUBACK* corresponding to a *SUBSCRIBE*)
3. messages violating the defined sequence classified based on their message type only (e.g. a *SUBACK* without a prior *CONNACK*)
4. messages violating the defined sequence classified based on both their message type and packet ID (e.g. a *SUBACK* without a prior matching *SUBSCRIBE*)

To cover the defined test classes, using MQTT with *QoS 0* is sufficient. The V2X Application-Level Gateway is expected to forward all messages adhering to the MQTT protocol sequence and to drop all messages violating the MQTT protocol sequence. It is also expected to drop all buffer overflow attack messages and all application layer DoS messages. Also, all attacks should be reported to the Automotive Cyber Defense Center.
Once the V2X service registrator has registered the MQTT traffic update service with the V2X Application-Level Gateway, so that it can serve as a proxy for the MQTT traffic update subscriber and the malicious MQTT broker, the MQTT traffic update subscriber connected to the V2X Application-Level Gateway which proactively established a connection to the malicious MQTT broker, see the sequence diagram in figure 6.12, page 98. The malicious MQTT broker answered the *CONNECT* with a *SUBACK* first, violating the defined protocol sequence and then correctly acknowledged the connection request with a *CONNACK* (covering test classes *1* and *3*). The MQTT traffic update publisher

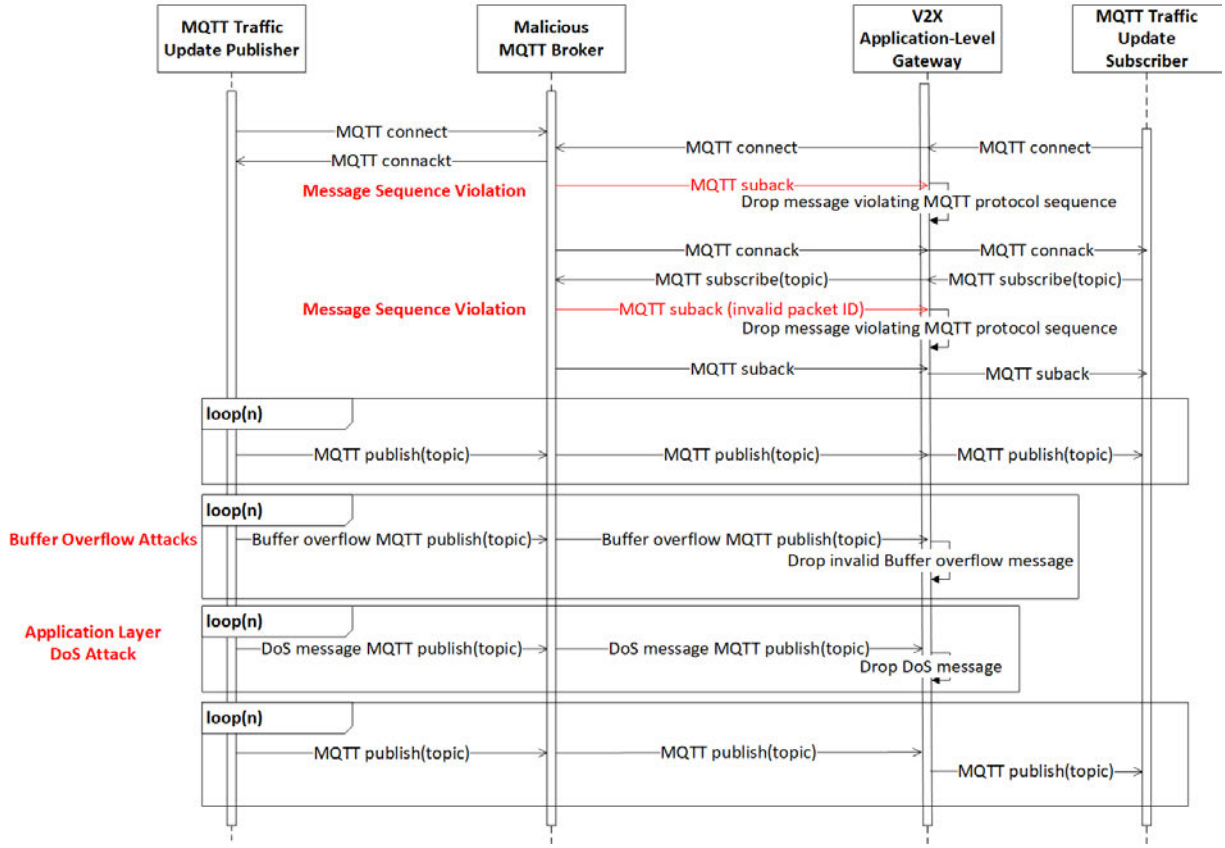

Figure 6.12: Sequence of the evaluation in the traffic update scenario

also connected to the malicious MQTT broker. Now the MQTT traffic update publisher is ready to publish traffic updates via the malicious MQTT broker and the MQTT traffic update subscriber is ready to subscribe to receive traffic updates via the V2X Application-Level Gateway and the malicious MQTT broker. The MQTT traffic update subscriber subscribed to receive traffic updates and the malicious MQTT broker first violated the message sequence of the MQTT protocol by sending a *SUBACK* message with an invalid packet ID to the MQTT traffic update subscriber (covering test class *4*), but then correctly acknowledged the *SUBSCRIBE* with a *SUBACK* with a matching packet ID (covering test class *2*).
Next, the MQTT traffic update publisher published several traffic updates with the topic the subscriber subscribed to. All valid messages, i.e. adhering to the defined protocol sequence, were forwarded by the V2X Application-Level Gateway, while all invalid messages, i.e. violating the defined protocol sequence, were dropped.

Next, buffer overflow attacks and an application layer DoS attack were launched by the MQTT traffic update publisher, see the sequence diagram in figure 6.12, page 98. First, 100 buffer overflow messages were published, which were dropped by the V2X Application-Level Gateway, see figure 6.13. Then the application layer DoS attack was launched by publishing an amount of traffic updates per period, which exceeds the number of messages allowed per period for the traffic update service. Here 1 was configured as a maximum threshold for a period of 1 second, which is seen as realistic for receiving traffic updates. The DoS rate was 2 messages per second and 200 DoS messages were sent (resulting in an attack duration of ∼100 seconds). After some time regular traffic updates were sent again. The V2X Application-Level Gateway dropped all DoS messages during the DoS attack, see figure 6.13. It forwarded all valid messages after the application layer DoS attack. Also, the message sequence violations, the buffer overflow and application layer DoS attacks were reported to the Automotive Cyber Defense Center.

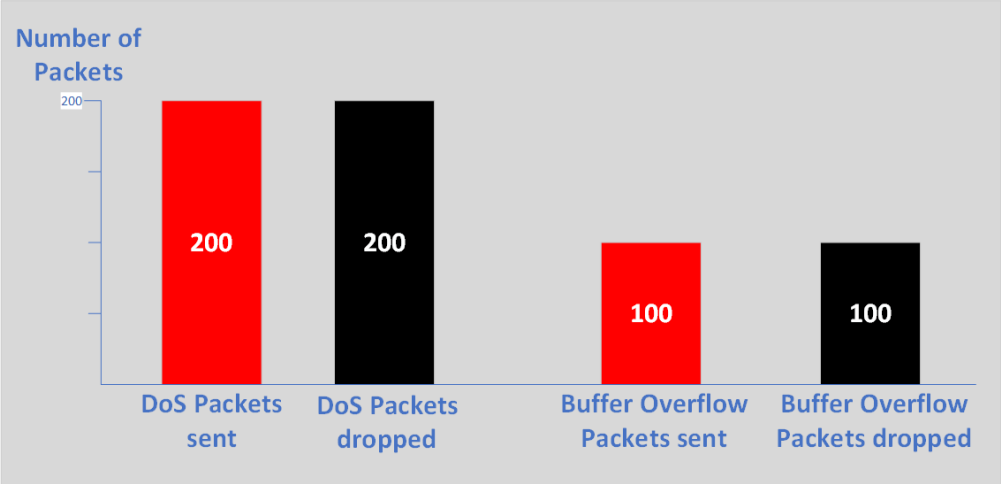

Figure 6.13: Sent and dropped application layer DoS and buffer overflow messages

6.4 Attacking the ETSI CAM V2V Traffic Safety Service

A V2V traffic safety service using ETSI CAMs was simulated. The attacks in this scenario were:

1. attacks based on malicious semantics requiring a semantic analysis

2. buffer overflow attacks, here, messages with a payload that exceeds the maximum payload size allowed by the application
3. an application layer DoS attack

The test classes for the evaluation of the semantic analysis of ETSI CAMs were defined based on the analysis in chapter 4, section 4.2.3. The following 5 test classes were defined:

1. valid ETSI CAMs containing correct data
2. ETSI CAMs that ignore the technical limitations of modern vehicles containing values that exceed values imaginable for road vehicles in the near future, which can be detected using stateless analysis
3. ETSI CAMs containing values that are realistic, but only typical in extreme situations and suspicious in a normal situation
4. ETSI CAMs that contradict the laws of physics or ignore the technical limitations of modern vehicles by containing values that deviate too much from the values possible based on the previous state of the object, which can be detected using stateful analysis
5. ETSI CAMs containing values that deviate from the values expected in a normal situation, based on the previous state of the object, but still are in accordance with the laws of physics and the technical limitations of modern vehicles

Specifically, for representing the second and third test class, ETSI CAMs containing vehicle speeds of more than *550 km/h* and *220 km/h* respectively, were chosen. For representing the fourth test class, ETSI CAMs containing conflicting vehicle position and speed information were chosen, i.e. vehicle positions and speeds that deviate from the expected values. E.g. when the position change is *25 m* for a given time interval, but the speed during this time interval did not exceed *10 m/s*, this contradicts the model defined in chapter 4, section 4.2.3. For the fifth test class also ETSI CAMs with contradicting vehicle positions and speeds were chosen. However, for this test class the deviations are small enough so that with the configured tolerance δ the data is classified as atypical but still valid.
The ETSI CAM generator sends ETSI CAMs of each test class to the V2X Application-Level Gateway, see the sequence diagram in figure 6.14, page 101. The V2X Application-Level Gateway is expected to forward all valid ETSI CAMs and to drop all invalid ETSI

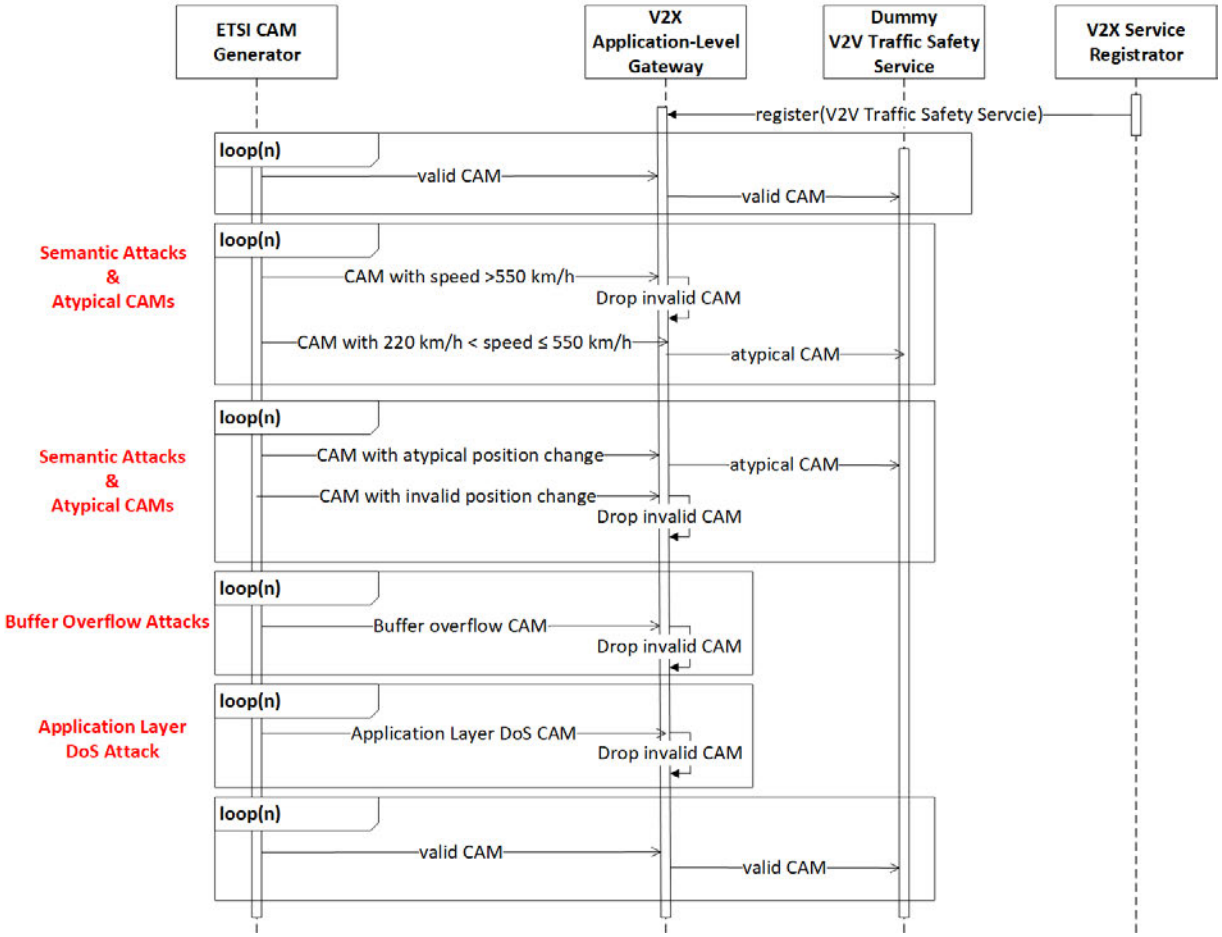

Figure 6.14: Sequence of the evaluation in the traffic update scenario

CAMs. It is also expected to drop all buffer overflow attack messages and all application layer DoS messages. Also, all attacks should be reported to the Automotive Cyber Defense Center.

First the V2X service registrator registered the V2V traffic safety service with the V2X Application-Level Gateway, so that it can serve as a proxy for the V2V traffic safety service and the ETSI CAM generator simulating several vehicles sending ETSI CAMs. Next, the ETSI CAM generator generated and sent valid ETSI CAMs, The V2X Application-Level Gateway classified them as valid and forwarded them normally. Then the ETSI CAM generator (specifically the attacker component) generated and sent ETSI CAMs containing vehicle speeds of more than *550 km/h* and *220 km/h* respectively, covering test classes *2* and *3*. The V2X Application-Level Gateway classified ETSI CAMs containing speeds *>550 km/h* as invalid, dropped them and reported an attack to the Automo-

tive Cyber Defense Center. ETSI CAMs containing speeds *>220 km/h* but *≤550 km/h* were classified as valid and forwarded, but a warning was sent to the Automotive Cyber Defense Center. Next, ETSI CAMs with a vehicle position where the change from the previous position exceeds the distance possible when driving with the given speed (i.e. $\triangle x > s_{max} \cdot \triangle t + \delta$) were generated and sent, covering test class *4*. They were classified as invalid, dropped and reported to the Automotive Cyber Defense Center. Then ETSI CAMs with a vehicle position where the change from the previous position exceeds the distance possible when driving with the given speed, but still falls below the threshold including the tolerance (i.e. $s_{max} \cdot \triangle t < \triangle x \leq s_{max} \cdot \triangle t + \delta$) were generated and sent, covering test class *5*. They were classified as valid and forwarded, but a warning was sent to the Automotive Cyber Defense Center.
Next, buffer overflow attacks and an application layer DoS attack were launched by the ETSI CAM generator (specifically the attacker component), see the sequence diagram in figure 6.14, page 101. First, 400 buffer overflow messages were sent, which were dropped by the V2X Application-Level Gateway, see figure 6.15. Then the application layer DoS attack was launched by sending an ETSI CAM each 50 *ms* thereby exceeding the 100 *ms* allowed between two consecutive ETSI CAMs. During the attack 1000 DoS messages were sent. After some time valid ETSI CAMs were sent again. The V2X Application-Level Gateway dropped all DoS messages during the DoS attack, see figure 6.15. It forwarded all valid messages after the application layer DoS attack. Also, the buffer overflow- and application layer DoS attacks were each reported to the Automotive Cyber Defense Center.

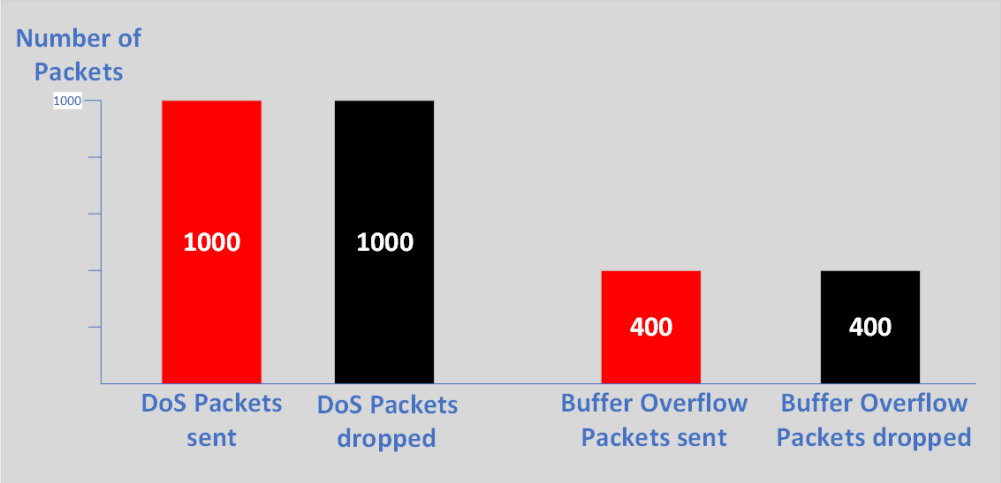

Figure 6.15: Sent and dropped application layer DoS and buffer overflow messages

6.5 Summary

Table 6.1 summarises the evaluation. For the given application protocols HTTPS, MQTT and ETSI CAMs of the ETSI Cooperative Awareness Basic Service all performed attacks, namely buffer overflow attack, protocol message sequence violation, application layer DoS attack and malicious semantics, were detected and the messages dropped by the V2X Application-Level Gateway, while all valid messages were forwarded as expected. This shows that the stateful analysis presented in chapter 4 allows the realisation of both a context-sensitive semantic analysis and the detection of protocol message sequence violations. The developed prototype meets the requirements of providing application layer security, proxy-functionality and supporting both IP-based and not IP-based application layer protocols for V2X communication.

Table 6.1: Evaluation overview: protocols and detected attacks

Attack	**HTTPS**	**MQTT**	**ETSI CAM**
Buffer overflow	✓	✓	✓
Message sequence violation	-	✓	-
Malicious Semantics	✓	-	✓
Application layer DoS	✓	✓	✓

✓performed and detected
- not performed

7 Conclusions and Future Work

In this work the concept of a V2X Application-Level Gateway was developed. The main requirements of such a gateway are that it offers application layer security of the V2X communication, including the context-sensitive semantic analysis of application data, detection of application layer protocol violations and detection of application layer DoS attacks, as well as proxy functionality and cryptographic security. It has to support both IP-based application protocols and application protocols based on V2X-specific network- and transport layer protocols, such as the ETSI Cooperative Awareness Basic Service, which can be based on WSMP. Additionally it enables bandwidth control of V2X traffic and a role-based access to in-vehicle resources via ACLs.
From these requirements the architecture of the V2X Application-Level Gateway, which is based on the general best-practice application-level gateway software architecture from [101], was derived. Based on this concept, a prototype was developed. The prototype implementation offers the following functionality: it serves as a proxy for V2X services, allowing both TCP/IP-based communication and WSMP-based communication between vehicle-internal services and external services. It allows the integration of modules securing V2X communication on the application layer. These modules have to implement a defined interface to be integrated into the V2X Application-Level Gateway. For the prototype the following modules have been implemented: an exemplary module for the context-sensitive semantic analysis of application data for a service remotely controlling the vehicle trunk via commands such as ”open” or ”lock”, a module for detecting MQTT protocol violations, a module checking the payload size to prevent buffer overflow attacks via oversized payloads, a module for application layer DoS detection controlling the data rate of application messages, i.e. messages per period of time and a module for the semantic analysis of ETSI CAMs. To offer some degree of cryptographic security SSL is used for securing TCP/IP-based communication.
The implementation was evaluated with the V2X Application-Level Gateway software run on an *Intel NUC* integrated in a test network representing an internal vehicle network, which was developed by the CoRE research group [24]. In this network, consisting

of an *Edgecore SDN switch* and *Intel NUCs* and *Raspberry Pis* representing vehicle ECUs, several V2X scenarios including attacks were simulated. This test network will be deployed in a test vehicle as part of the SecVI project [105]. For the tests 3 applications were developed: one for remotely controlling the vehicle trunk via HTTPS, one for receiving traffic updates (e.g. congestion warnings) via MQTT and a basic V2V traffic safety service using ETSI CAMs. The vehicle trunk remote control service was attacked with malformed messages, i.e. invalid commands, and malicious semantics, e.g. the "open" command while the vehicle is driving. The traffic update service was attacked with messages violating the MQTT protocol message sequence, e.g. a *SUBACK* without a prior corresponding *SUBSCRIBE*. The basic V2V traffic safety service was attacked with malicious semantics, e.g. speeds exceeding the maximum speed possible with modern road vehicles. All services were also targeted by application layer DoS attacks exceeding normal traffic rates and buffer overflow attacks where the payloads exceeded the maximum payload size specified by each service. It was shown that the V2X Application-Level Gateway detected all attacks and dropped all invalid messages and DoS traffic after detecting an application layer DoS attack and reported every attack.
The goal of future work is further development of the V2X Application-Level Gateway by extending the functionality providing more modules securing communication on the application layer. Protocol-specific modules could be added to prevent protocol violations for given application layer protocols like e.g. HTTP or SOME/IP, or to detect certain explicitly malicious content like e.g. SQL injections. Also, the semantic analysis of ETSI CAMs could be extended, e.g. by extending the physical model used for analysis.
To optimise application layer DoS detection the use of machine learning solutions for identifying DoS traffic could be examined. In the context of DoS attacks it could be analysed, if mechanisms like *Hashcash* are a suitable measure for DoS protection for automotive applications. Also, the remaining features of the V2X Application-Level Gateway like bandwidth control and role-based access to in-vehicle resources via ACLs could be further specified and implemented in the prototype.
Another possible area of research are the packet filter components of the V2X Security Gateway securing the communication on the network- and transport layer complementary to the V2X Application-Level Gateway. The safe storage and protection of cryptographic keys, certificates and configurations against manipulation could be yet another subject of future research, since potential attackers could have physical access to the automotive hardware.

Bibliography

[1] Abowd, Gregory D. ; Dey, Anind K. ; Brown, Peter J. ; Davies, Nigel ; Smith, Mark ; Steggles, Pete: Towards a better understanding of context and context-awareness. In: *International symposium on handheld and ubiquitous computing* Springer (Veranst.), 1999, S. 304–307

[2] Adams, Mike ; Kwon, Minseok: Vulnerabilities of the Real-time transport (RTP) protocol for voice over IP (VoIP) traffic. In: *2009 6th IEEE Consumer Communications and Networking Conference* IEEE (Veranst.), 2009, S. 1–5

[3] Aman, Waqas ; Kausar, Firdous: Towards a Gateway-based Context-Aware and Self-Adaptive Security Management Model for IoT-Based eHealth Systems. In: *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS* 10 (2019), Nr. 1, S. 280–287

[4] *AMQP*. https://www.amqp.org/. – Accessed: 10.08.2020

[5] Apvrille, Ludovic ; El Khayari, Rachid ; Henniger, Olaf ; Roudier, Yves ; Schweppe, Hendrik ; Seudié, Hervé ; Weyl, Benjamin ; Wolf, Marko: Secure automotive on-board electronics network architecture. In: *FISITA 2010 world automotive congress, Budapest, Hungary* Bd. 8, 2010

[6] Ariyapperuma, Suranjith ; Mitchell, Chris J.: Security vulnerabilities in DNS and DNSSEC. In: *The Second International Conference on Availability, Reliability and Security (ARES'07)* IEEE (Veranst.), 2007, S. 335–342

[7] Back, Adam u. a.: Hashcash-a denial of service counter-measure. (2002)

[8] Balduzzi, Marco ; Gimenez, Carmen T. ; Balzarotti, Davide ; Kirda, Engin: Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications. In: *NDSS* Citeseer (Veranst.), 2011

[9] Banks, A ; Briggs, E ; Borgendale, K ; Gupta, R: MQTT Version 5.0. In: *OASIS Standard* (2019)

[10] Bellovin, Steven M. ; Cheswick, William R.: Network firewalls. In: *IEEE communications magazine* 32 (1994), Nr. 9, S. 50–57

[11] Bittl, Sebastian: Efficient Secure Communication in VANETs under the Presence of new Requirements Emerging from Advanced Attacks. (2017)

[12] Boban, Mate ; Kousaridas, Apostolos ; Manolakis, Konstantinos ; Eichinger, Joseph ; Xu, Wen: Use cases, requirements, and design considerations for 5G V2X. In: *arXiv preprint arXiv:1712.01754* (2017)

[13] Bouard, Alexandre ; Schanda, Johannes ; Herrscher, Daniel ; Eckert, Claudia: Automotive proxy-based security architecture for ce device integration. In: *International Conference on Mobile Wireless Middleware, Operating Systems, and Applications* Springer (Veranst.), 2012, S. 62–76

[14] Brose, Gerald: A gateway to web services security–Securing SOAP with proxies. In: *International Conference on Web Services* Springer (Veranst.), 2003, S. 101–108

[15] BSI: *Sichere Anbindung von lokalen Netzen an das Internet.* 8 2014

[16] Buckl, Christian ; Camek, Alexander ; Kainz, Gerd ; Simon, Carsten ; Mercep, Ljubo ; Stähle, Hauke ; Knoll, Alois: The software car: Building ICT architectures for future electric vehicles. In: *2012 IEEE International Electric Vehicle Conference* IEEE (Veranst.), 2012, S. 1–8

[17] Burnside, Matt ; Clarke, Dwaine ; Mills, Todd ; Maywah, Andrew ; Devadas, Srinivas ; Rivest, Ronald: Proxy-based security protocols in networked mobile devices. In: *Proceedings of the 2002 ACM symposium on Applied computing*, 2002, S. 265–272

[18] Butcher, David ; Li, Xiangyang ; Guo, Jinhua: Security challenge and defense in VoIP infrastructures. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37 (2007), Nr. 6, S. 1152–1162

[19] Calandriello, Giorgio ; Papadimitratos, Panos ; Hubaux, Jean-Pierre ; Lioy, Antonio: Efficient and robust pseudonymous authentication in VANET. In: *Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks*, 2007, S. 19–28

[20] Calzavara, Stefano ; Focardi, Riccardo ; Nemec, Matus ; Rabitti, Alvise ; Squarcina, Marco: Postcards from the post-HTTP world: amplification of

HTTPS vulnerabilities in the web ecosystem. In: *2019 IEEE Symposium on Security and Privacy (SP)* IEEE (Veranst.), 2019, S. 281–298

[21] *Cereal C++ library.* https://uscilab.github.io/cereal/. – Accessed: 15.08.2020

[22] Chandola, Varun ; Banerjee, Arindam ; Kumar, Vipin: Anomaly detection: A survey. In: *ACM computing surveys (CSUR)* 41 (2009), Nr. 3, S. 1–58

[23] Chifor, Bogdan-Cosmin ; Bica, Ion ; Patriciu, Victor-Valeriu: Mitigating DoS attacks in publish-subscribe IoT networks. In: *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)* IEEE (Veranst.), 2017, S. 1–6

[24] *CoRE research group.* https://core.informatik.haw-hamburg.de/. – Accessed: 01.10.2020

[25] Deerwester, Scott ; Dumais, Susan T. ; Furnas, George W. ; Landauer, Thomas K. ; Harshman, Richard: Indexing by latent semantic analysis. In: *Journal of the American society for information science* 41 (1990), Nr. 6, S. 391–407

[26] Dey, Kakan C. ; Rayamajhi, Anjan ; Chowdhury, Mashrur ; Bhavsar, Parth ; Martin, James: Vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication in a heterogeneous wireless network–Performance evaluation. In: *Transportation Research Part C: Emerging Technologies* 68 (2016), S. 168–184

[27] Dietzel, Christoph ; Smaragdakis, Georgios ; Wichtlhuber, Matthias ; Feldmann, Anja: Stellar: network attack mitigation using advanced blackholing. In: *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies* ACM (Veranst.), 2018, S. 152–164

[28] Dinculeană, Dan ; Cheng, Xiaochun: Vulnerabilities and limitations of MQTT protocol used between IoT devices. In: *Applied Sciences* 9 (2019), Nr. 5, S. 848

[29] *Docker.* https://www.docker.com/. – Accessed: 10.08.2020

[30] AUTOSAR: *Specification of Diagnostic over IP.* 4.3.1. : , 2017

[31] Doshi, Keval ; Yilmaz, Yasin ; Uludag, Suleyman: Timely Detection and Mitigation of Stealthy DDoS Attacks via IoT Networks. In: *arXiv preprint arXiv:2006.08064* (2020)

[32] Doshi, Rohan ; Apthorpe, Noah ; Feamster, Nick: Machine learning ddos detection for consumer internet of things devices. In: *2018 IEEE Security and Privacy Workshops (SPW)* IEEE (Veranst.), 2018, S. 29–35

[33] Duan, Ling-Yu ; Xu, Min ; Chua, Tat-Seng ; Tian, Qi ; Xu, Chang-Sheng: A mid-level representation framework for semantic sports video analysis. In: *Proceedings of the eleventh ACM international conference on Multimedia*, 2003, S. 33–44

[34] Eiter, Thomas ; Füreder, Herbert ; Kasslatter, Fritz ; Parreira, Josiane X. ; Schneider, Patrik: Towards a semantically enriched local dynamic map. In: *International Journal of Intelligent Transportation Systems Research* 17 (2019), Nr. 1, S. 32–48

[35] El Sawda, Samer ; Urien, Pascal: SIP Security Attacks and Solutions: A state-of-the-art review. In: *2006 2nd International Conference on Information & Communication Technologies* Bd. 2 IEEE (Veranst.), 2006, S. 3187–3191

[36] ETSI, EN: Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM). In: *302 895 V1.1.1 (2014-09) Intelligent Transport Systems (ITS)* (2014)

[37] ETSI, EN: Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service. In: *302 637-2 V1.3.2 (2014-11) Intelligent Transport Systems (ITS)* (2014)

[38] ETSI, EN: Vehicular Communications; GeoNetworking; Part 3: Network Architecture. In: *302 636-3 V1.2.1 (2014-12) Intelligent Transport Systems (ITS)* (2014)

[39] ETSI, EN: LTE-V2X Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band. In: *303 613 V1. 1.0 (2019-05) Intelligent Transport Systems (ITS)* (2019)

[40] Evangelopoulos, Nicholas ; Zhang, Xiaoni ; Prybutok, Victor R.: Latent semantic analysis: five methodological recommendations. In: *European Journal of Information Systems* 21 (2012), Nr. 1, S. 70–86

[41] Firdous, Syed N. ; Baig, Zubair ; Valli, Craig ; Ibrahim, Ahmed: Modelling and evaluation of malicious attacks against the iot mqtt protocol. In: *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green*

Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) IEEE (Veranst.), 2017, S. 748–755

[42] Garg, Sachin ; Singh, Navjot ; Tsai, Timothy: SRTP+, An efficient scheme for RTP packet authentication. In: *Avaya Labs Research Technical Report (ALR-2004-001)* (2004)

[43] Gasser, Oliver ; Holz, Ralph ; Carle, Georg: A deeper understanding of SSH: results from Internet-wide scans. In: *2014 IEEE Network Operations and Management Symposium (NOMS)* IEEE (Veranst.), 2014, S. 1–9

[44] Ghosh, Mainak ; Varghese, Anitha ; Kherani, Arzad A. ; Gupta, Arobinda: Distributed misbehavior detection in VANETs. In: *2009 IEEE Wireless Communications and Networking Conference* IEEE (Veranst.), 2009, S. 1–6

[45] Glass, Michael ; Herrscher, Daniel ; Meier, Herbert ; Schoo, Peter u. a.: "Seis"—security in embedded IP-based systems. In: *ATZelektronik worldwide* 5 (2010), Nr. 1, S. 36–40

[46] Gong, Xuwei: *Security Threats and Countermeasures for Connected Vehicles.* 2019

[47] Hackel, Timo ; Meyer, Philipp ; Korf, Franz ; Schmidt, Thomas C.: Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)* IEEE (Veranst.), 2019, S. 1–5

[48] Haefner, Kyle ; Ray, Indrakshi: ComplexIoT: Behavior-Based Trust For IoT Networks. In: *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)* IEEE (Veranst.), 2019, S. 56–65

[49] Haga, Tomoyuki ; Takahashi, Ryota ; Sasaki, Takamitsu ; Kishikawa, Takeshi ; Tsurumi, Junichi ; Matsushima, Hideki: Automotive SIEM and Anomaly Detection using Sand-sprinkled Isolation Forest. (2017)

[50] Hamada, Yoshihiro ; Inoue, Masayuki ; Ueda, Hiroshi ; Miyashita, Yukihiro ; Hata, Yoichi: Anomaly-Based Intrusion Detection Using the Density Estimation of Reception Cycle Periods for In-Vehicle Networks. In: *SAE International Journal of Transportation Cybersecurity and Privacy* 1 (2018), Nr. 11-01-01-0003, S. 39–56

[51] Hao, Yong ; Cheng, Yu ; Zhou, Chi ; Song, Wei: A distributed key management framework with cooperative message authentication in VANETs. In: *IEEE Journal on selected areas in communications* 29 (2011), Nr. 3, S. 616–629

[52] Harding, John ; Powell, Gregory ; Yoon, Rebecca ; Fikentscher, Joshua ; Doyle, Charlene ; Sade, Dana ; Lukuc, Mike ; Simons, Jim ; Wang, Jing u. a.: Vehicle-to-vehicle communications: readiness of V2V technology for application. / United States. National Highway Traffic Safety Administration. 2014. – Forschungsbericht

[53] Haripriya, AP ; Kulothungan, K: Secure-MQTT: an efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things. In: *EURASIP Journal on Wireless Communications and Networking* 2019 (2019), Nr. 1, S. 1–15

[54] Harsha, MS ; Bhavani, BM ; Kundhavai, KR: Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and ACLs. In: *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* IEEE (Veranst.), 2018, S. 2244–2250

[55] Hassija, Vikas ; Chamola, Vinay ; Saxena, Vikas ; Jain, Divyansh ; Goyal, Pranav ; Sikdar, Biplab: A survey on IoT security: application areas, security threats, and solution architectures. In: *IEEE Access* 7 (2019), S. 82721–82743

[56] Heled, Ronen: HTTP REQUEST SMUGGLING. (2005)

[57] Herold, Nadine ; Posselt, Stephan-A ; Hanka, Oliver ; Carle, Georg: Anomaly detection for SOME/IP using complex event processing. In: *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium* IEEE (Veranst.), 2016, S. 1221–1226

[58] Hofmann, Thomas: Unsupervised learning by probabilistic latent semantic analysis. In: *Machine learning* 42 (2001), Nr. 1-2, S. 177–196

[59] Hogye, Michael A. ; Hughes, Christopher T. ; Sarfaty, Joshua M. ; Wolf, Joseph D.: Analysis of the feasibility of keystroke timing attacks over ssh connections. In: *Research Project at University of Virginia* (2001)

[60] Hoppe, Tobias ; Kiltz, Stefan ; Dittmann, Jana: Applying intrusion detection to automotive it-early insights and remaining challenges. In: *Journal of Information Assurance and Security (JIAS)* 4 (2009), Nr. 6, S. 226–235

[61] Hudaib, Adam Ali Z.: DNS advanced attacks and analysis. In: *International Journal of Computer Science and Security (IJCSS)* 8 (2014), Nr. 2, S. 63

[62] Idrees, Muhammad S. ; Schweppe, Hendrik ; Roudier, Yves ; Wolf, Marko ; Scheuermann, Dirk ; Henniger, Olaf: Secure automotive on-board protocols: a case of over-the-air firmware updates. In: *International Workshop on Communication Technologies for Vehicles* Springer (Veranst.), 2011, S. 224–238

[63] IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments. In: *IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009)* (2010), S. 1–51

[64] IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Networking Services. In: *IEEE Std 1609.3-2016 (Revision of IEEE Std 1609.3-2010)* (2016), S. 1–160

[65] IEEE Guide for Wireless Access in Vehicular Environments (WAVE) Architecture. In: *IEEE Std 1609.0-2019 (Revision of IEEE Std 1609.0-2013)* (2019), S. 1–106

[66] Jackson, Collin ; Barth, Adam ; Bortz, Andrew ; Shao, Weidong ; Boneh, Dan: Protecting browsers from DNS rebinding attacks. In: *ACM Transactions on the Web (TWEB)* 3 (2009), Nr. 1, S. 1–26

[67] Jeong, YiNa ; Son, SuRak ; Lee, ByungKwan: The Lightweight Autonomous Vehicle Self-Diagnosis (LAVS) Using Machine Learning Based on Sensors and Multi-Protocol IoT Gateway. In: *Sensors* 19 (2019), Nr. 11, S. 2534

[68] Jonker, Mattijs ; King, Alistair ; Krupp, Johannes ; Rossow, Christian ; Sperotto, Anna ; Dainotti, Alberto: Millions of targets under attack: a macroscopic characterization of the DoS ecosystem. In: *Proceedings of the 2017 Internet Measurement Conference* ACM (Veranst.), 2017, S. 100–113

[69] Jourdan, Guy-Vincent: Centralized web proxy services: Security and privacy considerations. In: *IEEE Internet Computing* 11 (2007), Nr. 6, S. 46–52

[70] Jungnickel, Ruben ; Köhler, Michael ; Korf, Franz: Efficient automotive grid maps using a sensor ray based refinement process. In: *2016 IEEE Intelligent Vehicles Symposium (IV)* IEEE (Veranst.), 2016, S. 668–675

[71] Kamara, Seny ; Fahmy, Sonia ; Schultz, Eugene ; Kerschbaum, Florian ; Frantzen, Michael: Analysis of vulnerabilities in internet firewalls. In: *Computers & Security* 22 (2003), Nr. 3, S. 214–232

[72] Kleberger, Pierre ; Olovsson, Tomas ; Jonsson, Erland: Security aspects of the in-vehicle network in the connected car. In: *2011 IEEE Intelligent Vehicles Symposium (IV)* IEEE (Veranst.), 2011, S. 528–533

[73] Langer, Falk ; Schüppel, Fabian ; Stahlbock, Lukas: Establishing an Automotive Cyber Defense Center. (2019)

[74] Levachkine, Serguei ; Velázquez, Aurelio ; Alexandrov, Victor ; Kharinov, Mikhail: Semantic analysis and recognition of raster-scanned color cartographic images. In: *International Workshop on Graphics Recognition* Springer (Veranst.), 2001, S. 178–189

[75] Lindberg, Johan: Security analysis of vehicle diagnostics using DoIP. (2011)

[76] Loukas, Georgios ; Öke, Gülay: Protection against denial of service attacks: A survey. In: *The Computer Journal* 53 (2010), Nr. 7, S. 1020–1037

[77] Luotonen, Ari ; Altis, Kevin: World-wide web proxies. In: *Computer Networks and ISDN systems* 27 (1994), Nr. 2, S. 147–154

[78] MacHardy, Zachary ; Khan, Ashiq ; Obana, Kazuaki ; Iwashina, Shigeru: V2X access technologies: Regulation, research, and remaining challenges. In: *IEEE Communications Surveys & Tutorials* 20 (2018), Nr. 3, S. 1858–1877

[79] Martuscelli, Giuseppe ; Boukerche, Azzedine ; Foschini, Luca ; Bellavista, Paolo: V2V protocols for traffic congestion discovery along routes of interest in VANETs: a quantitative study. In: *Wireless Communications and Mobile Computing* 16 (2016), Nr. 17, S. 2907–2923

[80] McAteer, Ian N. ; Malik, Muhammad I. ; Baig, Zubair ; Hannay, Peter: Security vulnerabilities and cyber threat analysis of the AMQP protocol for the internet of things. (2017)

[81] Meyer, Philipp ; Hackel, Timo ; Langer, Falk ; Stahlbock, Lukas ; Decker, Jochen ; Eckhardt, Sebastian A. ; Korf, Franz ; Schmidt, Thomas C. ; Schüppel, Fabian: A Security Infrastructure for Vehicular Information Using SDN, Intrusion Detection, and a Defense Center in the Cloud. In: *2020 IEEE Vehicular Networking Conference (VNC)* IEEE (Veranst.), 2020, S. 1–2

[82] Morabito, Roberto ; Petrolo, Riccardo ; Loscri, Valeria ; Mitton, Nathalie: LEGIoT: A lightweight edge gateway for the Internet of Things. In: *Future Generation Computer Systems* 81 (2018), S. 1–15

[83] Morris, Robert ; Jannotti, John ; Kaashoek, Frans ; Li, Jinyang ; Decouto, Douglas: CarNet: A scalable ad hoc wireless network system. In: *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, 2000, S. 61–65

[84] *MQTT Specification.* https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. – Accessed: 10.08.2020

[85] *MQTT QoS.* https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/. – Accessed: 29.11.2020

[86] Müller, Klaus-Rainer: *IT-Sicherheit mit System: Integratives IT-Sicherheits-, Kontinuitäts-und Risikomanagement-Sichere Anwendungen-Standards und Practices.* Springer-Verlag, 2014

[87] Nguyen, Hoang Duong T. ; Qi, Dawei ; Roychoudhury, Abhik ; Chandra, Satish: Semfix: Program repair via semantic analysis. In: *2013 35th International Conference on Software Engineering (ICSE)* IEEE (Veranst.), 2013, S. 772–781

[88] Nguyen, Thien D. ; Marchal, Samuel ; Miettinen, Markus ; Fereidooni, Hossein ; Asokan, N ; Sadeghi, Ahmad-Reza: DÏoT: A federated self-learning anomaly detection system for IoT. In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)* IEEE (Veranst.), 2019, S. 756–767

[89] Noack, Andreas: Timing analysis of keystrokes and timing attacks on ssh revisited. In: *Horst Gortz Institut fur IT-Sicherheit Ruhr-Universitat Bochum* (2007)

[90] Nugur, Aditya: *Design and Development of an Internet-Of-Things (IoT) Gateway for Smart Building Applications*, Virginia Tech, Dissertation, 2017

[91] *OpenSSL.* https://www.openssl.org/. – Accessed: 21.08.2020

[92] Owens, Jim ; Matthews, Jeanna: A study of passwords and methods used in brute-force SSH attacks. In: *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008

[93] Paar, Christof: Embedded security in automobilanwendungen. In: *Elektronik Automotive* 1 (2004), S. 152

[94] Pareigis, Stephan: *State Machine*. https://autosys.informatik.haw-hamburg.de/codesamples/state-machine/. – Accessed: 15.08.2020

[95] Pesé, Mert D. ; Schmidt, Karsten ; Zweck, Harald: Hardware/software co-design of an automotive embedded firewall / SAE Technical Paper. 2017. – Forschungsbericht

[96] *POCO C++ libraries*. https://pocoproject.org/. – Accessed: 15.08.2020

[97] Pretschner, Alexander ; Broy, Manfred ; Kruger, Ingolf H. ; Stauner, Thomas: Software engineering for automotive systems: A roadmap. In: *Future of Software Engineering (FOSE'07)* IEEE (Veranst.), 2007, S. 55–71

[98] *Pugixml C++ library*. https://pugixml.org/. – Accessed: 15.08.2020

[99] Roermund, Timo van ; Birnie, Andy ; Moran, Robert ; Frank, Juergen: Securing the in-vehicle network of the connected car. In: *14th Escar Europe Conference, München*, 2016

[100] Ruj, Sushmita ; Cavenaghi, Marcos A. ; Huang, Zhen ; Nayak, Amiya ; Stojmenovic, Ivan: On data-centric misbehavior detection in VANETs. In: *2011 IEEE Vehicular Technology Conference (VTC Fall)* IEEE (Veranst.), 2011, S. 1–5

[101] Schmidt, Douglas C.: A family of design patterns for applications-level gateways. In: *TAPOS* 2 (1996), Nr. 1, S. 15–30

[102] Scholte, Theodoor ; Robertson, William ; Balzarotti, Davide ; Kirda, Engin: Preventing input validation vulnerabilities in web applications through automated type analysis. In: *2012 IEEE 36th annual computer software and applications conference* IEEE (Veranst.), 2012, S. 233–243

[103] SCHUNTER, Matthias u. a.: Vehicle to Cloud-Research Challenges for Intelligent Vehicles. In: *15th Escar Europe Conference, Berlin*, 2017

[104] Scott, David ; Sharp, Richard: Abstracting application-level web security. In: *Proceedings of the 11th international conference on World Wide Web*, 2002, S. 396–407

[105] *SecVI Project*. https://secvi.inet.haw-hamburg.de/. – Accessed: 01.10.2020

[106] Shimada, Hideki ; Yamaguchi, Akihiro ; Takada, Hiroaki ; Sato, Kenya u. a.: Implementation and evaluation of local dynamic map in safety driving systems. In: *Journal of Transportation Technologies* 5 (2015), Nr. 02, S. 102

[107] AUTOSAR: *SOME/IP Protocol Specification*. 1.0.0. : , 2016

[108] Song, Dawn X. ; Wagner, David A. ; Tian, Xuqing: Timing analysis of keystrokes and timing attacks on ssh. In: *USENIX Security Symposium* Bd. 2001, 2001

[109] Studnia, Ivan ; Nicomette, Vincent ; Alata, Eric ; Deswarte, Yves ; Kaâniche, Mohamed ; Laarouchi, Youssef: Security of embedded automotive networks: state of the art and a research proposal, 2013

[110] Szancer, Sebastian: Grundprojekt Bericht: Architektur eines V2X Automotive Security Gateways. (2018)

[111] *Automotive Ethernet Testing*. https://sg.tek.com/automotive/automotive-ethernet. – Accessed: 30.08.2020

[112] Tonguz, Ozan K. ; Boban, Mate: Multiplayer games over vehicular ad hoc networks: A new application. In: *Ad Hoc Networks* 8 (2010), Nr. 5, S. 531–543

[113] Uzcátegui, Roberto A. ; De Sucre, Antonio J. ; Acosta-Marum, Guillermo: Wave: A tutorial. In: *IEEE Communications magazine* 47 (2009), Nr. 5, S. 126–133

[114] Wei, Dong ; Darie, Florin ; Shen, Ling: Application layer security proxy for smart Grid substation automation systems. In: *2013 IEEE PES Innovative Smart Grid Technologies Conference (ISGT)* IEEE (Veranst.), 2013, S. 1–6

[115] Weimerskirch, Andre: V2X security & privacy: the current state and its future. In: *ITS World Congress, Orlando, FL*, 2011

[116] Wijnants, Maarten ; Lamotte, Wim: The NIProxy: a Flexible Proxy Server Supporting Client Bandwidth Management and Multimedia Service Provision. In: *2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks* IEEE (Veranst.), 2007, S. 1–9

[117] Yang, Li ; Moubayed, Abdallah ; Hamieh, Ismail ; Shami, Abdallah: Tree-based Intelligent Intrusion Detection System in Internet of Vehicles. In: *arXiv preprint arXiv:1910.08635* (2019)

[118] Yang, Xue ; Liu, Leibo ; Vaidya, Nitin H. ; Zhao, Feng: A vehicle-to-vehicle communication protocol for cooperative collision warning. In: *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.* IEEE (Veranst.), 2004, S. 114–123

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Masterarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Traffic Analysis in V2X Application-Level Gateways

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original