Concept of a V2X Application-Level Gateway with Context-sensitive Semantic Analysis of Application Data

Sebastian Szancer
Hamburg University of Applied Sciences
Berliner Tor 7
20099 Hamburg, Germany
Email: sebastian.szancer@haw-hamburg.de

***Abstract*— Modern cars 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 enables the realisation of innovative functionality such as "over the air" ECU software updates, optimised navigation and route planning or coordinated autonomous driving. 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. It also realises the proxy-functionality and ensures cryptographic security. This paper presents a concept and prototype implementation of such a V2X Application-Level Gateway for IP-based traffic. 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, the scenario of remotely controlling the vehicle trunk was simulated.**

I. INTRODUCTION

Modern cars are part of various networks, from VANETs (Vehicular ad-hoc networks) to the Internet, making them a part of the "Internet of Things" (IoT). They communicate with a variety of entities ranging from other vehicles (Vehicle-to-Vehicle: V2V) and infrastructure (Vehicle-to-Infrastructure: V2I), such as traffic lights, to Internet-based services running on remote servers. Most of this V2X communication will probably be IP-based, although in case of V2V or V2I communication not based on IP is also possible. It is realised via a Connectivity-Gateway [38] using different technologies such as Wi-Fi (IEEE 802.11), Bluetooth, LTE or 5G. For the modern vehicle V2X communication is essential. Some innovative functionality such as optimised navigation and route planning (which in case of electric vehicles may depend on charging infrastructure), automated coordinated driving or vehicle maintenance via "over the air" ECU software updates cannot be realised without V2X. With V2X conditions such as road traffic or weather can be considered for navigation and route planning in live-time, while "over the air" ECU software updates allow the fast maintenance of a great number of vehicles without them having to go to a car service station. V2X also plays an important role for the realisation of autonomous vehicles.

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) [34]. 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 [7], [26], [44]. 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 [40] which is part of the Connectivity-Gateway, see figure 1. It consists of 3

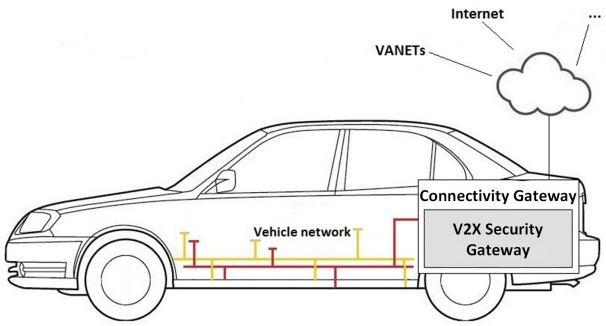

Fig. 1. Schematic: Modern car with V2X Security Gateway

components, combined in a common PAP-structure (packet filter - application-level gateway - packet filter - structure) [10]: 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, page 2. An application-level gateway (ALG) is a proxy which only forwards packets after controlling them on the application-layer [10]. If the data are encrypted, the ALG

should have the necessary cryptographic functionality to decrypt them and encrypt them again before forwarding the data. Such security gateway solutions, while novel in the

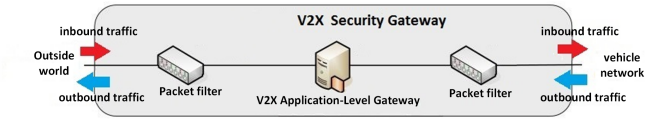

Fig. 2. Overview: V2X Security Gateway architecture

domain of automotive security, are established concepts in the classical IT security domain [10]. The stateful packet filters offer security on the internet- and transport layer. The V2X Application-Level Gateway ensures application-level security including a context-sensitive semantic analysis of application data. It also realises the proxy-functionality and ensures cryptographic security. Additionally it allows role-based access to in-vehicle resources and bandwidth control of V2X traffic. The aim of this paper is the development of a concept and prototype implementation of such a V2X Application-Level Gateway. Also the transfer of some of the functionality to the cloud, where more resources like computing power and memory are available, is discussed. This paper is organised as follows: section II gives an overview of related work and emphasises the contribution of this paper, in section III the concept of a V2X Application-Level Gateway is presented, preceded by a requirements analysis, in section IV a prototype implementation is presented and evaluated and section V concludes this paper and discusses future work.

II. RELATED WORK

This section 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, it remains a distinctive sub-domain and thus is treated separately in this work.

A. 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. [32]. In [32] 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. 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.

Of the work covering V2X security most focus on the realisation of cryptographic security of V2X communication [42], e.g. in the context of ECU software updates [21], or on security issues in VANETs like authentication [12], [20], [17], Denial-of-Service (DoS) attacks [6], [45] or misbehaviour detection [36], [18] 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. [8] 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 [16], with the ECUs, only passing on messages from authenticated CE devices to them. In [45] 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 contribution of this work is the proposed V2X Application-Level Gateway for securing V2X communication. Unlike [8] it is not limited to CE devices and offers more functionality than V2X security gateways

or proxies presented to date. In addition to ensuring cryptographic security and offering proxy functionality decoupling in-vehicle ECUs from the outside world, it also secures V2X communication on the application layer, including a semantic analysis of application data, 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. Application-level security functionality from IDS solutions like [45] could be incorporated into the V2X Application-Level Gateway. The V2X Application-Level Gateway is complementary to packet filter solutions like [32] securing the internet- and transport layer. Together they offer comprehensive security of V2X communication.

B. 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 [22] are covered. Increasingly however, the issue of security is also being addressed. An example is the IoT gateway presented in [29] 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 [11] 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 [41] 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 [43]. 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 [27] is using Docker (https://www.docker.com/) 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 [27] 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 [2]. The definitions of ”context” and ”context-awareness” used by [2] 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 [28], 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 [19]. 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, in addition to the proxy-functionality, cryptographic security and application layer security including the semantic analysis of application data that is: 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.

C. 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 [24], [9], which in this case is limited to Web protocols like HTTP [25], bandwidth control [5] and the semantic analysis of application data [39]. The web security proxy presented in [39] 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 [25]. 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.

III. CONCEPT

In this section the concept of the V2X Application-Level Gateway is presented, beginning with a requirements analysis followed by the architecture derived from it and a description of the aspect of semantic analysis. A discussion of a partially cloud-based approach to the V2X Application-Level Gateway concludes this section.

A. Requirements

The requirements can be divided into functional- and performance requirements. For the V2X Application-Level Gateway the requirements result from its task to secure the V2X communication. The following requirements have been identified:

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 data on the application layer according to predefined rules, including a context-sensitive semantic analysis (see section III-C), which 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.

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.

4) Realising role-based access to resources via ACLs to prevent unauthorised access.

5) Allowing bandwidth control of V2X traffic: dividing bandwidth among applications (like [43]) to optimise vehicle performance in every situation, e.g. by prioritising critical services if necessary.

6) Support of IP-based application layer protocols (e.g. HTTP), since most, if not all V2X traffic will probably be IP-based.

7) 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.

8) Logging functionality to facilitate maintenance.

Performance requirements:

1) Hard real-time capability with deadlines in the milliseconds ($<$10 ms) in case of hard real-time V2X communication such as collision avoidance. For non real-time V2X traffic the end-to-end delay is in the range of 100 ms to $>$1s [7], [26], [44].

2) Sufficient throughput for all V2X traffic (throughput for V2X applications is in the range of 5 Kbps to 700 Mbps and ranges from 10 to 80 Mbps for most applications [7], [26]).

B. Architecture

The architecture of the V2X Application-Level Gateway was developed based on the requirements identified in section III-A, the concept of service-oriented communication and the general best-practice application-level gateway software architecture described in [37]. The architecture of [37], see figure 3 (page 5), 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 application-level gateway. The *Connector* is used to proactively establish connections from the gateway 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 Channels* and *Output Channels*, e.g. one for each connection. The *Input Handler* and *Output Handler* provide proxy functionality to services communicating via the application-level gateway and thus the architecture meets the functional requirement *2)*. This base architecture was extended by several components to meet all requirements defined in section III-A, see figure 4 (page 5). The additional components provide the entire security functionality, whereas the base architecture components provide the basic communication functionality. The *Encryption/Decryption Components* for inbound and outbound traffic provide cryptographic security, meeting requirement *3)*. For ensuring cryptographic security EVITA HSMs[3] can be used in the V2X Application-Level Gateway. The *Access Control*

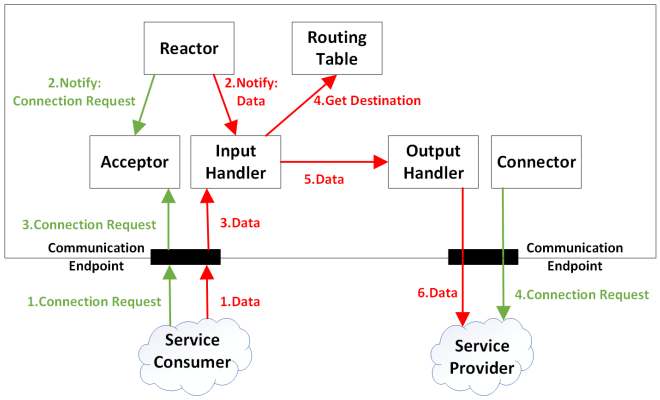

Fig. 3. Overview: Best practice Application-Level Gateway Software Architecture

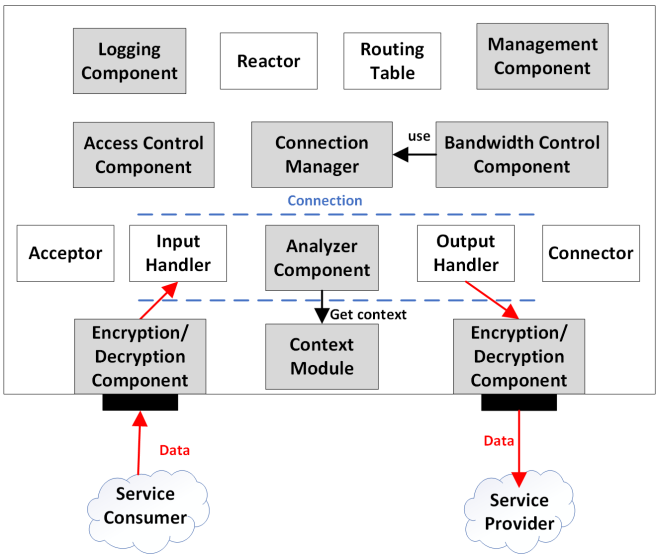

Fig. 4. Overview: V2X Application-Level Gateway architecture

Component manages the ACLs realising role-based access to resources, meeting requirement *5)*. 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* controlling the application data according to predefined rules, including a context-sensitive semantic analysis which meets requirement *4)*. A message is only forwarded if it is semantically correct. The *Context Module* holds the context required for the analysis, see section III-C. The *Analyzer Components* have to support all relevant IP-based protocols, meeting requirement *1)*. This can be achieved with exchangeable protocol-specific sub-components for the *Analyzer Component*, each supporting an IP-based application protocol, e.g. HTTP. The *Connection Manager* manages all (active) connections. The *Bandwidth Control Component* allows to manage the bandwidth of V2X traffic via the Connection Manager, according to certain distribution techniques, meeting requirement *6)*. With the *Logging Component*, which logs the activity of the other components, requirement *7)* 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 *8)*. So the developed architecture meets all functional requirements identified in section III-A. 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 section III-A.

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 5. Conversely,

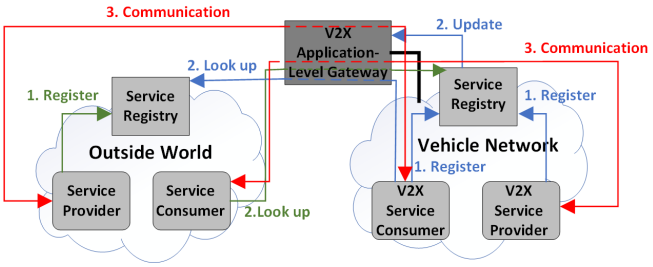

Fig. 5. Communication between vehicle services and external services

an external service provider registers with a service registry 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.

C. Semantic Analysis

In this work the semantic analysis of data is defined as checking the conformity of syntactically correct data with 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 *context-sensitive* or *context-insensitive* and either stateful or stateless. A general definition of context is any information that can be used to characterise the situation 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. traffic, weather). In the examples above, the semantic correctness is independent of the context and therefore the rules are classified as context-insensitive. The context can be modelled with different degrees of complexity and levels of abstraction depending on the use-cases requiring context. For example for this paper the modelling of the vehicle’s engine with a simple state machine consisting of only the two states ”DRIVING” and ”STOPPED” is sufficient, while other use-cases might 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). Generally, there are enough cases where the semantic correctness depends on the context and thus context awareness is necessary for a semantic analysis. The following is an example for the use of context-sensitive rules. Figure 6 shows state machines describing the correct behaviour of a vehicle’s trunk and engine. For better clarity, only the valid state transitions, excluding self-transitions, are depicted. The

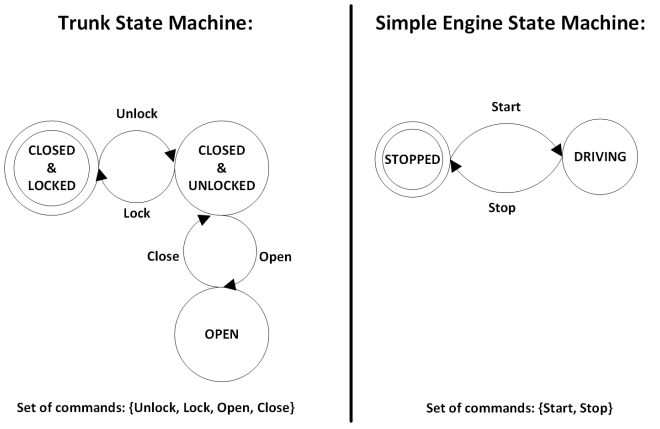

Fig. 6. Simple state machines describing the correct behaviour of a vehicle trunk and engine

command to lock the trunk (”Lock”) is in the set of known commands, but is semantically incorrect if the trunk is open (state ”OPEN”). 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, e.g. the engine and the trunk. The command to open the trunk (”Open”) for instance is semantically incorrect, regardless of the state of the trunk, if the vehicle is driving (engine state ”DRIVING”). A context-sensitive semantic analysis can be realised using finite state machines (FSMs). A system (e.g. a vehicle trunk) can be modelled as an FSM consisting of states and transitions, which are triggered by events from an alphabet. In the case of the vehicle trunk the events represent commands such as ”Open” or ”Close” from a defined set of commands. The general idea of a context-sensitive semantic analysis is to check if an event triggers a transition in the given state of the system or not (with not triggering a transition being equivalent to transitioning to an error state). Events triggering a transition are classified as valid, i.e. semantically correct and events not triggering a transition are classified as invalid, i.e. semantically incorrect. For a system modelled with a single FSM this can be done by extending it, adding a transition for every element in the alphabet in each state and a ”*true*” (T) and ”*false*” (F) state, see figure 7, where a transition for event *a* (*check a*) and a transition for event *b* (*check b*) is added in both state *A* and state *B*. The alphabet is also extended by one additional event for each event in the original alphabet, with every new event being equivalent to an event from the original alphabet (e.g. ”*check a*” and ”*a*”). In every state each of the new events (here: *check a* and *check b*) triggers exactly one transition t_i from the new transitions added to the state. The transition t_i either transitions to the ”*true*” state or ”*false*” state, depending on whether the original event (here: *a* or *b*) equivalent to the new event triggering the transition t_i (here: *check a* or *check b*), triggers a transition in that state itself. E.g. if event *b* triggers a transition in state *A*, then the new transition in state *A* triggered by *check b* transitions to ”*true*” state. The transition in state *A* triggered by *check a* transitions to ”*false*” state if event *a* triggers no transition in state *A*. This way for every event it can be checked whether

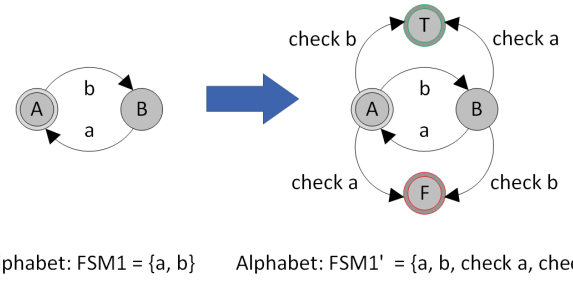

Fig. 7. Extending an FSM to perform a context-sensitive semantic analysis

it triggers a transition to ”*true*” state or ”*false*” state in any state, i.e. whether it is semantically correct or incorrect in that state. Naturally, this check operates only on a subset of the extended alphabet, i.e. the events added to the original alphabet (here: *check a* or *check b*). For an FSM where *s* is the number of states, *t* the number of transitions and *n* the number of events in the alphabet, after the extension there

are $s + 2$ states, $2 * n$ events in the alphabet and $t + s * n$ transitions. The increase in transitions is the most significant, but with "$s * n$" it depends only on the number of states and the size of the alphabet. So in general, in a system modelled with a single FSM, an event e is semantically correct, when the system FSM is in a certain state X (e.g. A or B) in which event e triggers a transition, which can be expressed as a Boolean expression: *S(e): (FSM == A) OR (FSM == B)*. The semantic correctness of events in a system (e.g. a vehicle trunk) can depend on the state of one or more other systems (e.g. the engine). In this case the first is called the dependent system and the latter the affecting system(s). E.g. the event e in a dependent system FSM_1 with an affecting system FSM_2 is valid when the dependent system FSM_1 is in a state A and the affecting system FSM_2 is in a state C. The Boolean expressions for the two separate systems are:
$S_1(e)$: $FSM_1 == A$ and $S_2(e)$: $FSM_2 == C$
which combined results in the Boolean expression for the semantic correctness of event e:
$S(e)$: $S_1(e)$ AND $S_2(e)$ = $(FSM_1 == A)$ AND $(FSM_2 == C)$.
The affecting systems can either affect the dependent system independently of each other, or interdependently. Affecting the dependent system independently of each other means that for evaluating the semantic correctness of an event in the dependent system the state of every affecting system can be considered separately. The semantic correctness of an event e in a dependent system FSM_1 with the affecting systems FSM_2, ... FSM_N can be evaluated with Boolean expressions of the form:
$S(e)$: $S_1(e)$ AND / OR $S_2(e)$ ... AND / OR $S_N(e)$ where S_i is a Boolean expression referring exclusively to FSM_i. In contrast to this, with affecting the dependent system interdependently, at least one sub-expression S_i does not exclusively refer to FSM_i, but refers to at least 2 systems FSM_i and FSM_j with $i \neq j$. An example would be the following expression:
$S(e)$: ($(FSM_1 == A)$ AND $(FSM_2 == C)$) OR
($(FSM_1 == B)$ AND $(FSM_2 == D)$)
Also, with interdependently affecting systems the number of sub-expressions S_i does not have to equal the number of systems FSM_i. In case of a dependent system with independently affecting systems the context-sensitive semantic analysis can be realised analogously to the single system case using extended FSMs. The dependent system FSM is extended as described above. The FSM of an affecting system is extended analogously, see figure 8. The alphabet is extended by the alphabet of the dependent system and a transition for every element in the alphabet of the dependent system is added in each state, as well as a "*true*" state and "*false*" state. This is done for all independently affecting systems. In the example in figure 8 events a and b in the dependent system FSM1 are valid when the affecting system FSM2 is in state C and when FSM2 is in state D, only event a is valid, while event b is invalid. This way for every event in the dependent system it can be checked whether it triggers a transition to "*true*" state or "*false*" state in any state of any affecting system, i.e. whether it is semantically correct or incorrect in the context of the affecting systems. Checking

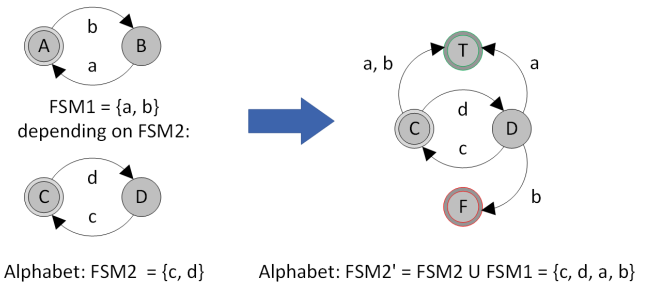

Fig. 8. Extending an FSM for the context-sensitive semantic analysis of a dependent system

the context-sensitive semantic correctness of an event with multiple extended FSMs (one for each affecting system), instead of one big FSM combining all affecting systems, facilitates maintainability and extensibility. Instead of having to use the same one big FSM representing the entire context, the context can be split into modules and different dependent systems can use different affecting system FSMs, namely only the ones they need. It is still possible for different dependent systems to share the same affecting system FSM, as long as it is extended for each of the dependent systems and multiple extension is possible. In case of interdependently affecting systems it is not possible to use separate FSMs for the affecting systems and instead the cross-product has to be used. In the example in figure 9 events e and f in the dependent system FSM3 are valid when the affecting system FSM1 is in state A and FSM2 is in state C or when FSM1 is in state B and FSM2 is in state D:
$S(e)$: ($(FSM_1 == A)$ AND $(FSM_2 == C)$) OR
($(FSM_1 == B)$ AND $(FSM_2 == D)$)
This interdependence on the semantic correctness of events e and f requires the use of the cross-product of the affecting systems FSM1 and FSM2. This cross-product can now be extended analogously to the previous cases: The alphabet is extended by the alphabet of the dependent system and a transition for every element in the alphabet of the dependent system is added in each state, as well as a "*true*" state and "*false*" state. This way for every event in the dependent

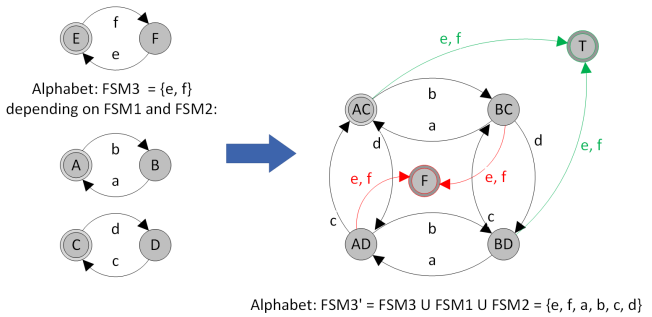

Fig. 9. Extended FSM cross product for the context-sensitive semantic analysis of a dependent system

system it can be checked whether it triggers a transition to "*true*" state or "*false*" state in any state of the cross-product

of affecting systems, i.e. whether it is semantically correct or incorrect in the context of the affecting systems.
For a correct analysis the V2X Application-Level Gateway has to get the current states of all relevant components. This comes with the general problem of temporary inconsistency between the context state as perceived by the gateway and the actual context state, due to the propagation delay of state changes (see figures 10, 11). The context update in the V2X Application-Level Gateway can generally be realised with 2 different methods: either the gateway is notified whenever a context state change occurs (see figure 10), or it proactively requests the current state (see figure 11). 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 3 criteria: their contribution to the (in-vehicle) network load, the effort of an efficient implementation and their potential to alleviate the problem of temporary inconsistency under certain assumptions. Although solving the problem of temporary inconsistency is not a task for the V2X Application-Level Gateway and ultimately every ECU itself decides for every received message if it will be processed or not, the possibility of alleviating the problem under certain realistic assumptions shall be discussed. 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, but the request approaches come with an additional message compared to the notification approach. And 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 significantly higher network load. Whereas with the periodic request approach and the notification approach even if the frequency is high, the network load is a priori known. So when it comes to the contribution to the network load, the notification approach is preferred, 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

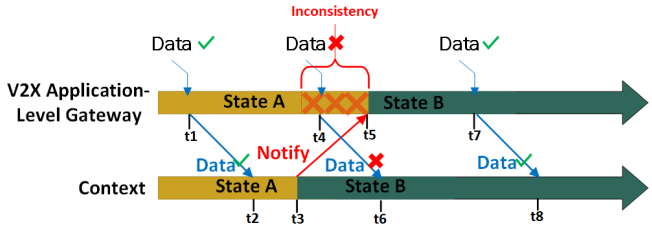

Fig. 10. Inconsistency of context state with notification

implementation again the notification approach is preferred, since it can be implemented directly, while for the periodic request approach optimum cycle times have to be identified first. In practice a state change of the context often does not

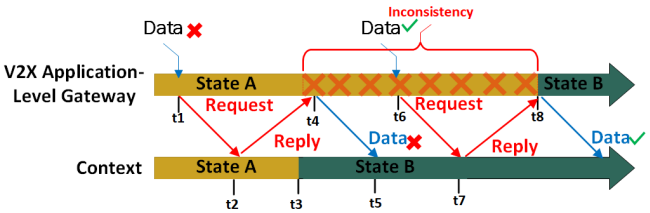

Fig. 11. Inconsistency of context state with request on demand

happen instantly at a single point in time, but takes a certain amount of time (t_{dS}), e.g. the opening or closing of a vehicle trunk. After a state change the state usually stays constant for at least a certain amount of time ($t_{s\ const}$), before further state change is possible. Under these assumptions the problem of temporary inconsistency can be avoided with the notification approach, if the propagation delay of the notification ($t_{\ notify}$) is smaller than the time it takes for the state change (t_{dS}), see figure 12, page 9. At the beginning of a state change (*t1*) a notification with the known time of completion of the state change (*t3*) is sent to the V2X Application-Level Gateway. When it arrives at the V2X Application-Level Gateway (*t2*) with a delay of $t_{\ notify}$, the gateway can synchronise its state at *t3* with the context. In case the successful completion of a state change cannot always be guaranteed, a second notification sent at *t3* to the V2X Application-Level Gateway is needed to avoid inconsistency. This second notification acknowledges the successful completion of the state change at *t4*. The analysis of data arriving after an expected state change (*t3*), but before the acknowledgement of the successful completion of the state change (*t4*), has to be delayed until the acknowledgement arrives and the state of the context is known certainly. This way a temporary inconsistency is avoided and all data can be analysed with the correct context state. For this approach a clock synchronisation of the V2X Application-Level Gateway and the context module is necessary. Whether avoiding temporary inconsistency at the cost of clock synchronisation is worth the effort depends on how critical the effect of allowing temporary inconsistency is on the in-vehicle network and the ECUs directly affected by it. With temporary inconsistency, messages arriving shortly before and after a context state change are forwarded by the V2X Application-Level Gateway to the destination ECUs even if they are invalid. These invalid messages contribute to ECU- and network load. Whether this load is significant depends on the size and frequency of the invalid messages in combination with the frequency of state changes. For many cases, like the remote trunk control described above, it is safe to say that the negative effects of temporary inconsistency are negligible. Since it is impossible to solve the inconsistency problem using the periodic request approach because it would have to send a request at every state change, which is impossible to predict, again the notification approach is

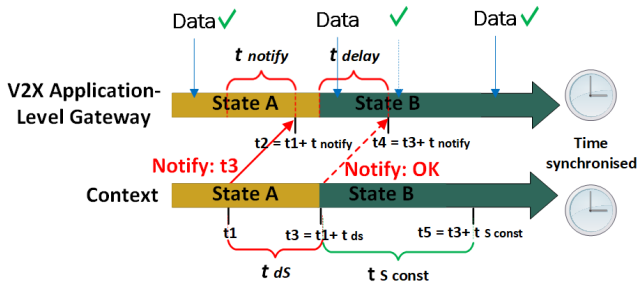

Fig. 12. Inconsistency with non time-discrete state changes

preferred. So under the criteria of contribution to in-vehicle network load, the effort of an efficient implementation and the potential to alleviate the problem of temporary inconsistency the notification approach is preferable for realising context updates and therefore implemented 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 by ECUs of the in-vehicle network e.g. a domain controller, see figure 13. In this case the ECU has to be able to decrypt the data prior to analysis. When a semantic analysis of the same data is performed in multiple components, it should be distributed optimally, where optimally depends on balancing security and performance. For maximum security there should be as much redundancy in the analysis in multiple components as possible. This way the failure of a compromised component can be compensated. 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.

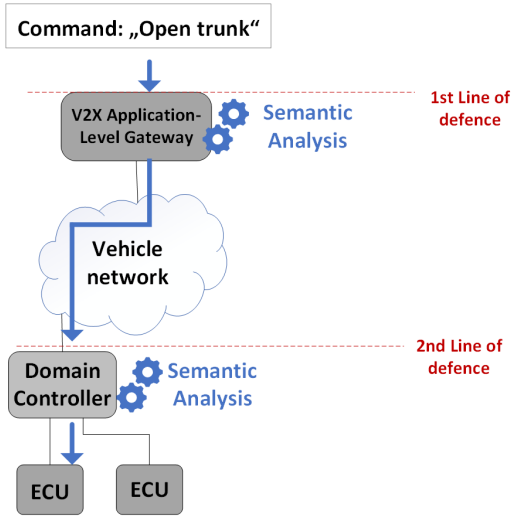

Fig. 13. Simple example of distributed semantic analysis in a vehicle

D. 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 14 (page 10), to relieve the V2X Application-Level Gateway component built into the vehicle. 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.

IV. PROTOTYPE AND EVALUATION

In this section the developed prototype implementation of the V2X Application-Level Gateway is presented and evaluated.

A. Implementation

The prototype implementation of the concept presented in section III focuses on providing application layer security and thus offers the following functionality: it serves as a proxy for V2X services, allowing TCP-based communication between vehicle-internal services and external services and performs a context-sensitive semantic analysis of application data according to predefined rules. While being aware of the vulnerabilities of TLS [13], to offer some degree of cryptographic security, SSL (with *openssl* [30]) is used for securing communication over the V2X Application-Level Gateway. To facilitate V2X communication over the internet, the HTTP protocol is supported, which combined with SSL gives HTTPS. Since a thorough context-sensitive semantic analysis is always application-specific, i.e. tailored to a

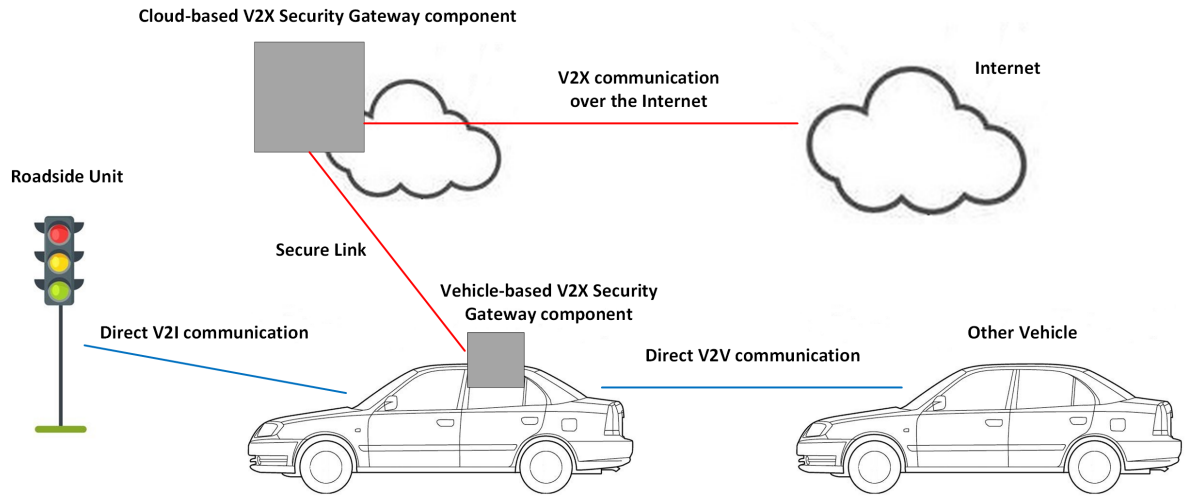

Fig. 14. Cloud-based V2X Application-Level Gateway

specific application, it was implemented exemplary for a service remotely controlling the vehicle trunk via commands such as ”open” or ”lock”. The prototype implementation was written in C++ with the following libraries being used: the TCP socket programming, SSL and HTTP implementation was done with the *POCO* libraries [33], for reading configurations from XML files the *pugixml* library [35] was used and serialization was realised with the *cereal* library [14].

A general overview of the prototype implementation's key elements realising the main functionality is depicted in figure 15. A *CustomSocketAcceptor*, derived from

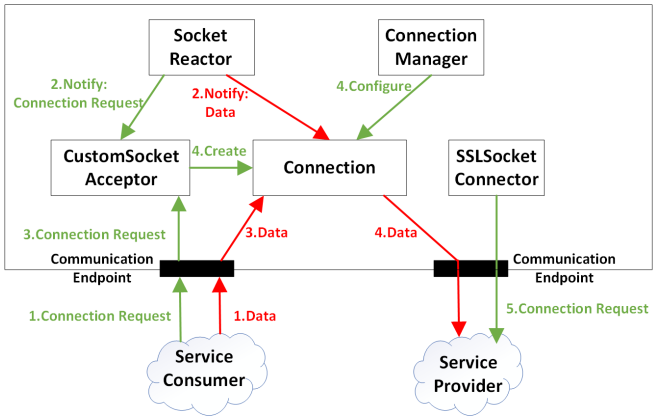

Fig. 15. Prototype implementation overview

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 established (”Connector-Acceptor Pattern”). The *CustomSocketAcceptor* and *Connection* are called by a POCO *SocketReactor* whenever a new connection request or application data arrive to handle it (”Reactor Pattern”). Upon creation, the *Connection* is configured by the *ConnectionManager*, which uses configured factories to create the appropriate objects (”Factory Pattern”). A *Connection* consists of a *CompositeBuffer* for incoming traffic and a *CompositeBuffer* for outgoing traffic, an *SSLConnectionHandler* for incoming traffic and an *SSLConnectionHandler* 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 17 (page 11). A *CompositeBuffer* consists of multiple modules, see figure 16: a simple buffer module containing the data accessible via an interface (*IBuffer*), a module for controlling the read- and write-indices of the buffer module and a module for fill-level control of the buffer. The buffer module can be protocol-specific and is exchangeable depending on the application protocol required for the connection, e.g. for HTTP the *HTTPPacketBuffer* can be used, while the remaining modules of the *CompositeBuffer* can be used independently of the application protocol. Components writing to and reading from a *CompositeBuffer*

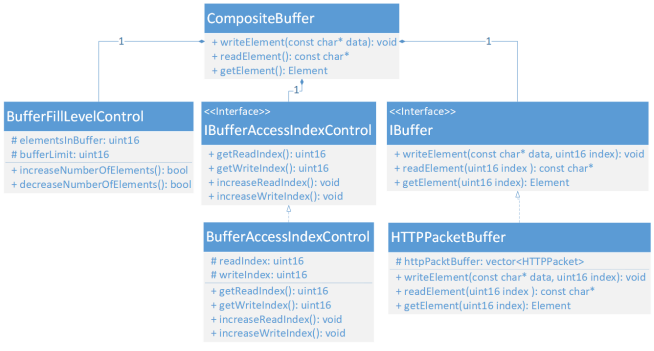

Fig. 16. UML class-diagram: CompositeBuffer

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 *IBuffer*) stores the data. The *CompositeBuffer* accesses this data via the *IBuffer* interface using the indices from the *BufferAccessIndexControl* module. This modularisation facilitates flexibility and code re-use. The *SSLConnectionHandler* 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 performs its analysis and if the data is valid it orders the *SSLConnectionHandler* for outgoing traffic to read the data from the *CompositeBuffer* and send them to the destination. For traffic going the other way round, the *Connection* works analogously. The analyzer

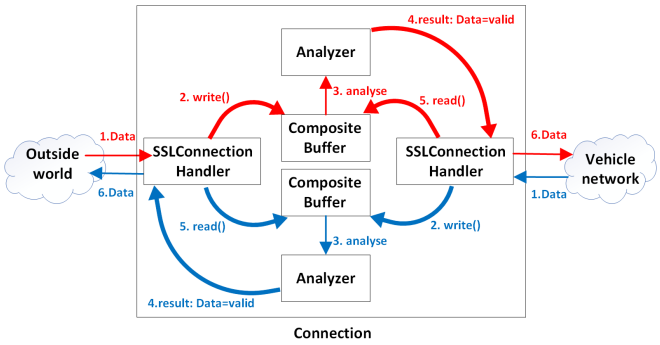

Fig. 17. Connection over V2X Application-Level Gateway

module has to implement the interface *IAnalyzer* with the "*analyse*"-method (signature: "*bool analyse(input)*") allowing the integration into a *Connection* and can be tailored to a specific application. Generally it consists of a set of rules and a check 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. The signature of a rule looks like this: "*bool isValid(input)*" and multiple rules can be combined in an arbitrarily complex Boolean function. For an analysis of a (protocol-specific) header or the (application-specific) payload a protocol- and application-specific extractor module for extracting the relevant information from the packet in the buffer is necessary.

For a context-sensitive semantic analysis as described in section III-C the analyzer module has to know the vehicle's current state at all time. In this prototype implementation the vehicle state is decomposed into separate subsystem states: the engine state and the trunk state. The V2X Application-Level Gateway has a *ContextModule* for every subsystem holding the subsystem's current state, see figure 18 (page 12). It is notified of every state change in any of the subsystems, so that the *ContextModules* always hold the correct states. A *ContextModule* models the state as a finite state machine (FSM) using an efficient implementation of the state pattern [31]. In this state pattern implementation each state of an FSM is a separate *struct* derived from a common base *struct*, see listing 1. The change of states is realised with the placement operator new.

Listing 1. State pattern example: vehicle trunk

```
struct ITrunkState {
  //Process the command "unlock".
  virtual void signalUnlock() = 0;
  //Process the command "lock".
  virtual void signalLock() = 0;
  //Process the command "open".
  virtual void signalOpen() = 0;
  //Process the command "close".
  virtual void signalClose() = 0;
};


class TrunkFSM{

private:

  //Vehicle trunk "closed & locked" state.
  struct TrunkStateLocked: public ITrunkState{

    //Constructor.
    TrunkStateLocked(){
      std::cout<<"closed & locked"<<std::endl;
    }

    //Transition to "closed & unlocked".
    void signalUnlock() override {
      new(this) TrunkStateClosed;
    }

    //Self-transition: already locked.
    void signalLock() override {}

    //Invalid!
    void signalOpen() override {}

    //Self transition: already closed.
    void signalClose() override {}
  };

  //Remaining states...

protected:

  //The current state.
  TrunkStateLocked state;

  //Pointer to current state.
  ITrunkState *statePointer;

public:

  //Constructor.
  TrunkFSM(): statePointer(&state){}

  //Delegates the command "unlock" to the state.
  void signalUnlock(){
    statePointer->signalUnlock();
  }

  ...
};
```

The state pattern was chosen for its clarity, extensibility and maintainability. An analyzer module can use the *ContextModules* to check if a given payload is valid in the current state. It accesses the *ContextModules* via a filter component checking if the payload is in the set of known commands, see figure 18, in this case whether it is a known remote trunk control command (”unlock”, ”open”, ”close” or ”lock”). If

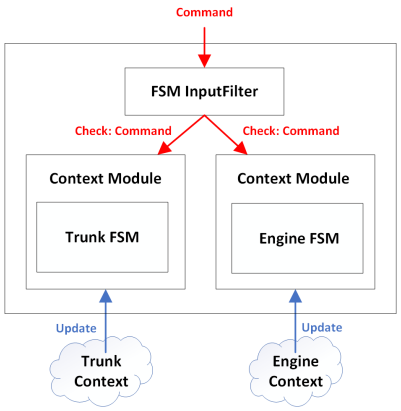

Fig. 18. V2X ALG context modules with input filter

the payload corresponds to a known command, a check is performed if it is valid in the context of the trunk (e.g. the ”open” command would be invalid if the trunk was locked and closed) and if it is valid in the context of the engine (e.g. the ”open” command would be invalid if the vehicle was driving). To perform such a check for a set of commands (e.g. trunk control commands) for a context (e.g. the trunk) using an FSM like in listing 1, it has to be extended as described in section III-C. This can be done by defining an interface for checking for any command if it is valid, see listing 2. Each state has to implement this interface, which corresponds to adding a transition for each possible trunk command to the state. Instead of transitioning to a ”true” or ”false” state, a Boolean is returned. This way it is possible to check for any command in any state if it is valid or not. The implementation for the engine FSM is analogous to that for the trunk FSM.

Listing 2. Extended FSM for context-sensitive analysis of trunk commands

```
struct ICheckTrunk {
  //Check the command "unlock".
  virtual bool checkSignalUnlock() = 0;
  //Check the command "lock".
  virtual bool checkSignalLock() = 0;
  //Check the command "open".
  virtual bool checkSignalOpen() = 0;
  //Check the command "close".
  virtual bool checkSignalClose() = 0;
};


struct CheckedTrunkState: public ITrunkState,
    ICheckTrunk {
    //
};
```

```
class CheckedTrunkFSM{

private:

  //Vehicle trunk "closed & locked" state.
  struct TrunkStateLocked: public CheckedTrunkState{

    //Constructor.
    TrunkStateLocked(){
      std::cout<<"closed & locked"<<std::endl;
    }

    //Transition to "closed & unlocked".
    void signalUnlock() override {
      new(this) TrunkStateClosed;
    }

    //Other transitions as in ''class TrunkFSM''...

    //Check the command "unlock".
    bool checkSignalUnlock() override {
      return true;
    }

    //Check the command "lock".
    bool checkSignalLock() override{
      return false;
    }

    //Check the command "open".
    bool checkSignalOpen() override{
      return false;
    }

    //Check the command "close".
    bool checkSignalClose() override{
      return false;
    }
  };

  //Remaining states...

  ...
};
```

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 19, page 13. A remote client, e.g. running on an ECU, registers 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 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.

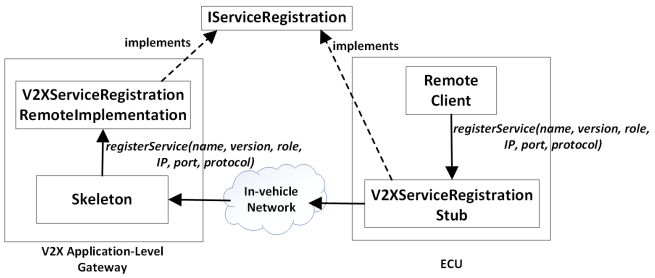

Fig. 19. V2X service registration via RMI

B. Evaluation

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, see figure 20. In this network, consisting of an *Edgecore SDN switch* and *Intel NUCs* and *Raspberry Pis* representing vehicle ECUs, the scenario of remotely controlling the vehicle trunk was simulated. For the evaluation 4 additional programs were implemented and deployed in the tests: a trunk control server allowing a remote client to control a dummy vehicle trunk via a set of defined commands, a trunk control client for remotely controlling a vehicle trunk via a set of defined commands, a V2X service registrator allowing to register V2X services via HTTPS with the V2X Application-Level Gateway and a dummy engine control holding the state of a dummy engine and notifying registered observers upon any state change. The trunk control server allows a remote client to connect and control the dummy 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, e.g. in the ”closed and unlocked” state receiving the ”open” command triggers the transition to the ”open” state. Multiple observers can register to be notified via HTTPS by the trunk control 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. The communication between trunk control server and client is via HTTPS. The dummy engine control holds the state of the engine (possible states: ”driving” and ”stopped”), which can be switched via keyboard. Multiple observers can register to be notified via HTTPS by the dummy engine control upon any state change. The trunk control server, V2X service registrator and dummy engine control were run on the *Zonal Controller Rear Left*. The trunk control client was run outside the network. With this setup the V2X Application-Level Gateway was evaluated by checking if the vehicle trunk could be remotely controlled over it securely. First the V2X Application-Level Gateway registered itself with the trunk control server and dummy engine 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 trunk control server and client. Next, the trunk control client connects to the V2X Application-Level Gateway and it proactively establishes a connection to the trunk control server. Now the trunk control client can remotely control the vehicle trunk via the V2X Application-Level Gateway. The V2X Application-Level Gateway is tested by sending commands to it from the trunk control client. There are 4 test classes: 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 engine in ”driving”-state and 4) unknown commands, such as ”asdf”, which are invalid per se. These test classes cover all possible attacks on the semantics of remotely controlling a vehicle trunk. The V2X Application-Level Gateway is expected to forward all valid commands to the trunk control server and drop all invalid commands. First, valid commands (*test class 1*) were sent with the vehicle engine state being ”stopped”. They were all forwarded, which is the expected correct behaviour. Next, some invalid commands (*test class 2*), such as the ”lock”-command in ”open”-state and the ”open”-command in ”closed and locked”-state and unknown commands (*test class 4*), were sent, with the vehicle engine state being ”stopped”. They were all dropped, which is the expected correct behaviour. Then the engine state was switched to ”driving”. Again, first valid commands (*test class 1*) were sent, then invalid and unknown (*test class 4*) commands were sent as well. Of the invalid commands some were invalid independently from the context (*test class 2*) , such as the ”lock”-command in ”open”-state, while others (*test class 3*) were only invalid with the engine in ”driving”-state, such as the ”open”-command in ”closed and unlocked”-state. The valid commands were all forwarded, while all invalid and unknown commands were dropped, which is the expected correct behaviour. Sending valid, unknown and invalid commands alternately also resulted in all valid commands being forwarded and all invalid and unknown commands being dropped. So the context-sensitive semantic analysis of the V2X Application-Level Gateway worked correctly in all of the tests, which cover all possible attacks on the semantics of remotely controlling a vehicle trunk.

V. 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 semantic analysis of data, which is context-sensitive if necessary, as well as proxy functionality and cryptographic security. Additionally it enables bandwidth control of V2X traffic and a role-based access to in-vehicle resources via

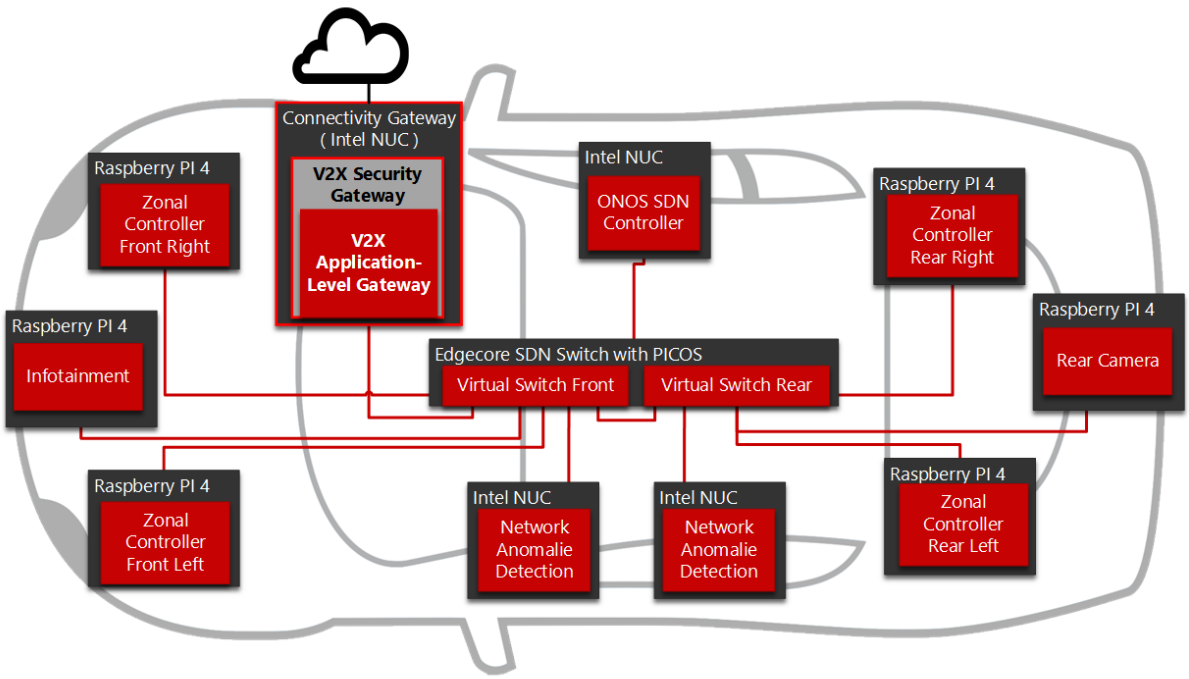

Fig. 20. In-vehicle test network

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 [37], 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 TCP-based communication between vehicle-internal services and external services. With application layer security the focus is on the context-sensitive semantic analysis of data, which is implemented exemplary for a service remotely controlling the vehicle trunk via commands such as ”open” or ”lock”. To offer some degree of cryptographic security HTTPS is used for 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. In this network, consisting of an *Edgecore SDN switch* and *Intel NUCs* and *Raspberry Pis* representing vehicle ECUs, the scenario of remotely controlling the vehicle trunk was simulated. It was shown that the V2X Application-Level Gateway analysed incoming trunk commands correctly and dropped them when they were invalid in the current context, such as the ”open” command while the (simulated) vehicle was driving.

The goal of future work is further development of the V2X Application-Level Gateway by extending the functionality providing application layer security by adding more rule sets for the analysis of application data. The rules could be structural or content-related and they could apply to application headers or payloads. It could be examined if such rules can be transferred from e.g. intrusion detection systems. A possible extension of the context-sensitive semantic analysis would be the analysis of sequences of context states instead of just the current context state. 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.

Apart from that, the analysis of the benefits and costs of applying virtualisation to the V2X Application-Level Gateway is a possible goal of future work. 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.

A security threat not addressed by the V2X Security Gateway are Denial-of-Service (DoS) attacks. Although most effective DoS countermeasures are network-based [23], [15] and the action a single network node, in this context a single vehicle, can take to protect itself from DoS attacks is limited, a mechanism has been proposed that enables single network nodes to contribute to security form DoS attacks. The basic principle behind this mechanism, known as *Hashcash* [4], assigns a cost to participating in

a protocol with a node, e.g. making a request, by having the requester compute a token (the computation is based on finding partial hash-collisions) for the node to proceed with the protocol, e.g. process 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. It could be analysed, if mechanisms like Hashcash are a suitable measure for DoS protection in the automotive context.

REFERENCES

[1] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., & Steggles, P. (1999, September). Towards a better understanding of context and context-awareness. In International symposium on handheld and ubiquitous computing (pp. 304-307). Springer, Berlin, Heidelberg.

[2] Aman, W., & Kausar, F. (2019). Towards a Gateway-based Context-Aware and Self-Adaptive Security Management Model for IoT-Based eHealth Systems. INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, 10(1), 280-287.

[3] Apvrille, L., El Khayari, R., Henniger, O., Roudier, Y., Schweppe, H., Seudi, H., ... & Wolf, M. (2010, May). Secure automotive on-board electronics network architecture. In FISITA 2010 world automotive congress, Budapest, Hungary (Vol. 8).

[4] Back, A. (2002). Hashcash-a denial of service counter-measure.

[5] Bellovin, S. M., & Cheswick, W. R. (1994). Network firewalls. IEEE communications magazine, 32(9), 50-57.

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

[7] Boban, M., Kousaridas, A., Manolakis, K., Eichinger, J., & Xu, W. (2017). Use cases, requirements, and design considerations for 5G V2X. arXiv preprint arXiv:1712.01754.

[8] Bouard, A., Schanda, J., Herrscher, D., & Eckert, C. (2012, November). Automotive proxy-based security architecture for ce device integration. In International Conference on Mobile Wireless Middleware, Operating Systems, and Applications (pp. 62-76). Springer, Berlin, Heidelberg.

[9] Brose, G. (2003). A gateway to web services securitySecuring SOAP with proxies. In Web Services-ICWS-Europe 2003 (pp. 101-108). Springer, Berlin, Heidelberg.

[10] Bundesamt fr Sicherheit in der Informationstechnik (BSI): Sichere Anbindung von lokalen Netzen an das Internet (ISi-LANA), BSI-Standards zur Internet-Sicherheit (ISi-S), Version 2.1 vom 26.08.2014

[11] Burnside, M., Clarke, D., Mills, T., Maywah, A., Devadas, S., & Rivest, R. (2002, March). Proxy-based security protocols in networked mobile devices. In Proceedings of the 2002 ACM symposium on Applied computing (pp. 265-272). ACM.

[12] Calandriello, G., Papadimitratos, P., Hubaux, J. P., & Lioy, A. (2007, September). Efficient and robust pseudonymous authentication in VANET. In Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks (pp. 19-28). ACM.

[13] Calzavara, S., Focardi, R., Nemec, M., Rabitti, A., & Squarcina, M. (2019, May). Postcards from the post-HTTP world: amplification of HTTPS vulnerabilities in the web ecosystem. In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 281-298). IEEE.

[14] Cereal C++ library. https://uscilab.github.io/cereal/ Accessed: 15.04.2020

[15] Dietzel, C., Smaragdakis, G., Wichtlhuber, M., & Feldmann, A. (2018, December). Stellar: network attack mitigation using advanced blackholing. In Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies (pp. 152-164). ACM.

[16] Glass, M., Herrscher, D., Meier, H., & Schoo, P. (2010). Seissecurity in embedded IP-based systems. ATZelektronik worldwide, 5(1), 36-40.

[17] Gong, X. (2019). Security Threats and Countermeasures for Connected Vehicles.

[18] Ghosh, M., Varghese, A., Kherani, A. A., & Gupta, A. (2009, April). Distributed misbehavior detection in VANETs. In 2009 IEEE Wireless Communications and Networking Conference (pp. 1-6). IEEE.

[19] Haefner, K., & Ray, I. (2019, December). 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) (pp. 56-65). IEEE.

[20] Hao, Y., Cheng, Y., Zhou, C., & Song, W. (2011). A distributed key management framework with cooperative message authentication in VANETs. IEEE Journal on selected areas in communications, 29(3), 616-629.

[21] Idrees, M. S., Schweppe, H., Roudier, Y., Wolf, M., Scheuermann, D., & Henniger, O. (2011, March). Secure automotive on-board protocols: a case of over-the-air firmware updates. In International Workshop on Communication Technologies for Vehicles (pp. 224-238). Springer, Berlin, Heidelberg.

[22] Jeong, Y., Son, S., & Lee, B. (2019). The Lightweight Autonomous Vehicle Self-Diagnosis (LAVS) Using Machine Learning Based on Sensors and Multi-Protocol IoT Gateway. Sensors, 19(11), 2534.

[23] Jonker, M., King, A., Krupp, J., Rossow, C., Sperotto, A., & Dainotti, A. (2017, November). Millions of targets under attack: a macroscopic characterization of the DoS ecosystem. In Proceedings of the 2017 Internet Measurement Conference (pp. 100-113). ACM.

[24] Jourdan, G. V. (2007). Centralized web proxy services: Security and privacy considerations. IEEE Internet Computing, (6), 46-52.

[25] Luotonen, A., & Altis, K. (1994). World-wide web proxies. Computer Networks and ISDN systems, 27(2), 147-154.

[26] MacHardy, Z., Khan, A., Obana, K., & Iwashina, S. (2018). V2X access technologies: Regulation, research, and remaining challenges. IEEE Communications Surveys & Tutorials, 20(3), 1858-1877.

[27] Morabito, R., Petrolo, R., Loscri, V., & Mitton, N. (2018). LEGIoT: a Lightweight Edge Gateway for the Internet of Things. Future Generation Computer Systems, 81, 1-15.

[28] Nguyen, T. D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N., & Sadeghi, A. R. (2019, July). DoT: A federated self-learning anomaly detection system for IoT. In 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS) (pp. 756-767). IEEE.

[29] Nugur, A. (2017). Design and Development of an Internet-Of-Things (IoT) Gateway for Smart Building Applications (Doctoral dissertation, Virginia Tech).

[30] OpenSSL. https://www.openssl.org/ Accessed: 21.04.2020

[31] Pareigis, Stephan. State Machine. https://autosys.informatik.haw-hamburg.de/codesamples/state-machine/ Accessed: 15.04.2020

[32] Pes, M. D., Schmidt, K., & Zweck, H. (2017). Hardware/Software Co-Design of an Automotive Embedded Firewall (No. 2017-01-1659). SAE Technical Paper.

[33] POCO C++ libraries. https://pocoproject.org/ Accessed: 15.04.2020

[34] Pretschner, A., Broy, M., Kruger, I. H., & Stauner, T. (2007, May). Software engineering for automotive systems: A roadmap. In Future of Software Engineering (FOSE'07) (pp. 55-71). IEEE.

[35] Pugixml C++ library. https://pugixml.org/ Accessed: 15.04.2020

[36] Ruj, S., Cavenaghi, M. A., Huang, Z., Nayak, A., & Stojmenovic, I. (2011, September). On data-centric misbehavior detection in VANETs. In 2011 IEEE Vehicular Technology Conference (VTC Fall) (pp. 1-5). IEEE.

[37] Schmidt, D. C. (1996). A family of design patterns for applications-level gateways. TAPOS, 2(1), 15-30.

[38] Schunter, M., et al. (2017, November). Vehicle to Cloud - Research Challenges for Intelligent Vehicles. 15th Escar Europe Conference, Berlin.

[39] Scott, D., & Sharp, R. (2002, May). Abstracting application-level web security. In Proceedings of the 11th international conference on World Wide Web (pp. 396-407). ACM.

[40] Szancer, S. (2018, October). Architektur eines V2X Automotive Security Gateways (Report, Hamburg University of Applied Sciences).

[41] Wei, D., Darie, F., & Shen, L. (2013, February). Application layer security proxy for smart Grid substation automation systems. In Innovative Smart Grid Technologies (ISGT), 2013 IEEE PES (pp. 1-6). IEEE.

[42] Weimerskirch, A. (2011, October). V2X security & privacy: the current state and its future. In ITS World Congress, Orlando, FL.

[43] Wijnants, M., & Lamotte, W. (2007, June). The NIProxy: a Flexible Proxy Server Supporting Client Bandwidth Management and Multimedia Service Provision. In World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a (pp. 1-9). IEEE.

[44] Yang, X., Liu, L., Vaidya, N. H., & Zhao, F. (2004, August). A vehicle-to-vehicle communication protocol for cooperative collision warning. In Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on (pp. 114-123). IEEE.

[45] Yang, L., Moubayed, A., Hamieh, I., & Shami, A. (2019). Tree-based Intelligent Intrusion Detection System in Internet of Vehicles. arXiv preprint arXiv:1910.08635.