This paper has been presented on the ICISSP 2019. http://www.icissp.org

Probabilistic Modeling and Simulation of Vehicular Cyber Attacks: An Application of the Meta Attack Language

Sotirios Katsikeas[1], Pontus Johnson[1], Simon Hacks[2], and Robert Lagerström[1]
[1]*Department of Network and Systems Engineering, KTH Royal Institute of Technology, Stockholm, Sweden*
[2]*Research Group Software Construction, RWTH Aachen University, Aachen, Germany*
{*sotkat, pontusj, robertl*}*@kth.se, hacks@swc.rwth-aachen.de*

Keywords: Domain Specific Language, Cyber Security, Threat Modeling, Attack Graphs, Vehicular Security

Abstract: Attack simulations are a feasible means to assess the cyber security of systems. The simulations trace the steps taken by an attacker to compromise sensitive system assets. Moreover, they allow to estimate the time conducted by the intruder from the initial step to the compromise of assets of interest. One commonly accepted approach for such simulations are attack graphs, which model the attack steps and their dependencies in a formal way.
To reduce the effort of creating new attack graphs for each system of a given type, domain-specific attack languages may be employed. They codify common attack logics of the considered domain. Consequently, they ease the reuse of models and, thus, facilitate the modeling of a specific system in the domain. Previously, MAL (the Meta Attack Language) was proposed, which serves as a framework to develop domain specific attack languages.
In this article, we present vehicleLang, a Domain Specific Language (DSL) which can be used to model vehicles with respect to their IT infrastructure and to analyze their weaknesses related to known attacks. To model domain specifics in our language, we rely on existing literature and verify the language using an interview with a domain expert from the automotive industry. To evaluate our results, we perform a Systematic Literature Review (SLR) to identify possible attacks against vehicles. Those attacks serve as a blueprint for test cases checked against the vehicleLang specification.

1 INTRODUCTION

Vehicles, especially passenger cars, are ubiquitous in European countries and their amount is still rising. For example, from January 2009 to January 2018 the number of registered passenger cars was increased by nearly 18% in Sweden (Statistiska centralbyrån, 2018) and 12.5% in Germany (Kraftfahrt-Bundesamt, 2018). This means that 62% of Swedes and 56% of Germans owned a car in average in January 2018. Simultaneously, IT-systems pervade vehicles more and more to increase driving security and user experience. Contemporaneously, the number of IT related security issues grows as well (Symantec, 2017). This raises the need to assess the cyber security of vehicles.

However, assessing the cyber security of computing systems in general and of vehicles in special is difficult. In order to identify vulnerabilities, the security-relevant parts of the system must be understood, and all potential attacks have to be identified. There are three challenges related to these needs: Firstly, it is challenging to identify all relevant security properties of a system. Secondly, it might be difficult to collect this information. Lastly, the collected information needs to be processed to uncover all weaknesses that can be exploited by an attacker.

Hitherto, we have proposed the use of attack simulations based on system architecture models (e.g., (Ekstedt et al., 2015; Holm et al., 2015)) to support these challenging tasks. Our approaches facilitate a model of the system and simulate cyber attacks in order to identify the greatest weaknesses. This can be imagined as the execution of a great number of parallel virtual penetration tests. Such an attack simulation tool enables the security assessor to focus on the collection of the information about the system required for the simulations, since the first and third challenges are tackled by the simulation.

As the previous approaches rely on a static implementation, we propose MAL (the Meta Attack Language) (Johnson et al., 2018). This domain-specific language defines which information about a system is required and specifies the generic attack logic. As MAL is a meta language, no certain domain of interest, like vehicle security, is represented. Therefore, this work aims to create and evaluate a domain-specific, probabilistic modeling language for simulation of known cyber attacks on modern connected vehicles, so called vehicleLang.

To create vehicleLang, we follow the means of DSR (Design Science Research) as presented by Peffers et al. (Peffers et al., 2007). Therefore, we conduct a domain survey to extract a feature matrix containing all the security assets, the possible attacks and the corresponding defenses. This feature matrix serves as input to compile a detailed list of all the possible attacks following some widely used attack classifications and taxonomy frameworks. Subsequent, vehicleLang is constructed and evaluated by the results of a SLR (Systematic Literature Review). A bottom-up approach is followed and, therefore, the created artifact is limited to modeling the internal networks of the vehicles (i.e., CAN bus (Controller Area Network), FlexRay, and LIN bus (Local Interconnect Network)), the connected components to them (i.e., ECUs (Electronic Control Units), Gateway ECUs) and the attacks that are related. However, it is scheduled to extend vehicleLang to other parts of the vehicle as well in the near future.

The remainder of the paper is structured as follows. Section 2 considers related work. In Section 3 we present the MAL and its syntax to provide the framework for vehicleLang, which is presented with the core language in Section 4. The language itself is evaluated by a number of test cases depicted in Section 5. Finally, the paper is concluded in Section 6.

2 RELATED WORK

Our work relates to three domains of previous work: model-driven security engineering, attack/defense graphs, and vehicle security. First, there are domain-specific languages for security analysis of software and system models defined in the domain of model-driven security engineering as vehicleLang is one. Second, attack/defense graphs are applied as formalism for its analysis. Last, besides the fact that vehicleLang is in the domain of vehicle security, the results of existing vehicle security research are utilized for evaluation.

Model-driven security engineering induced a large number of domain-specific languages (Jürjens, 2005; Basin et al., 2011; Alam et al., 2007; Paja et al., 2015). These languages facilitate the capability to model a system's design according to components and their interaction. Furthermore, they also enable to model security properties such as constraints, requirements, or threats. They are built upon different formalisms and logics like the Unified Modeling Language and the Object Constraint Language. Model checking and searches for constraint violations are applied to conduct security analysis in these languages.

The concept of attack trees is commonly based on the work of Bruce Schneier (Schneier, 1999; Schneier, 2000). They were formalized by Mauw & Oostdijk (Mauw and Oostdijk, 2005) and extended to include defenses by Kordy et al. (Kordy et al., 2010). As summarized in (Kordy et al., 2014), there are several approaches elaborating on attack graphs, e.g. (Ingols et al., 2009; Williams et al., 2008). Elaborating on the theoretical achievements of the beforehand presented papers, different tools using attack graphs were developed. These tools mostly build up on collecting information about existing system or infrastructure and automatically create attack graphs based on this information. For example, the TVA tool (Noel et al., 2009) models security conditions in networks and uses a database of exploits as transitions between these security conditions.

The approaches of attack graphs and system modeling are united in our previous works: e.g., P2CySeMoL (Holm et al., 2015), and securiCAD (Ekstedt et al., 2015). The central idea of these works is to automatically generate probabilistic attack graphs from a given system specification. The attack graph serves as inference engine that produces predictive security analysis results from the system model.

To the best of our knowledge, there exists no research in the direction of probabilistic attack graph modeling within the domain of vehicle security. Therefore, we relate following to work elaborating on vehicle hacking. A first impression on the domain of vehicle hacking can be received by the work of Wolf (Wolf, 2009) and Smith (Smith, 2016). On the one hand, the authors give insights into vehicles' IT architectures. We use these insights as input for the general construction of our instance of MAL. On the other hand, they depict different ways to compromise various components of the vehicle under investigation.

More detailed attack scenarios are given by different authors e.g. Wolf et al. (Wolf et al., 2004), Takahashi et al. (Takahashi et al., 2017), or Cho and Shin (Cho and Shin, 2016). The first two work concentrate on attacking the bus of the vehicle. In (Wolf et al., 2004) the authors attack different vehicular bus networks and show their weaknesses as well as the weaknesses of future bus networks. To strengthen vehicle's security, they also propose a secured communication along the various buses. In contrast to Wolf et al., Takahashi et al. (Takahashi et al., 2017) focuses on LIN bus. They show possible attacks on this elementary vehicle communication system through different case studies. Furthermore, they contribute by proposing different countermeasures to raise vehicle's security. The last work by Cho and Shin (Cho and Shin, 2016) demonstrates attacks on ECUs using DoS (Denial-of-Service). For this purpose, they facilitate the error-handling scheme of the in-vehicle network to shut down or disconnect the ECU. Additionally, they suggest and evaluate approaches to detect and prevent their attacks.

3 THE META ATTACK LANGUAGE

As already presented, this domain specific language relies on the MAL (the Meta Attack Language) (Johnson et al., 2018). Due to reasons of brevity, we refer the reader to the the original paper regarding the core parts of the grammar (syntax), the formalism and more details behind the MAL language.

We would just like to mention that MAL mainly consists of classes (e.g., `Car`), that can be instantiated (i.e., `myCar`), attack steps on classes (e.g., `Car.hijack`), and defenses on classes (e.g., `Car.immobilizer`). Furthermore, the entities of MAL can be related to each other. Classes, and respectively their instances, can be linked to each other (e.g., `Garage.parked` = {`Car`}). Attack steps are also connected to each other, thus, the successful compromise of one step leads to the second step (e.g., $e = (\texttt{Garage.open}, \texttt{Garage.parked.accessible})$). Additionally, attack steps can be either of the type OR or AND, signifying that either one of its parental steps is needed to elaborate on this step (OR) or all steps are needed (AND). Lastly, defenses are parenting attack steps which they hinder to be performed if they are TRUE (e.g., $(\texttt{Car.immobilizer}, \texttt{Car.drive}) \in E$).

Some attack steps may be accomplished without effort. However, sometimes attack steps require a certain amount of effort, expressed in time. Those attack steps can be associated with a probability distribution describing the expected time to perform this step, the so called *local time to compromise*. Simulating different attackers behaviour on the modeled MAL attack graph allows to calculate the *global time to compromise*. This value provides a measure of how secure various points of the modeled system are in respect to attack resilience. Further, it facilitates a quantitative way of comparing different system designs.

4 THE VEHICULAR CYBER ATTACK LANGUAGE

Beforehand, we briefly presented the foundations underlying the MAL. Following, we will present, first, a core language containing common IT entities already included in the presentation of MAL (Johnson et al., 2018), and, second, vehicleLang covering all the basic assets that comprise the internal vehicle's network, from ECUs and vehicle's networks, up to applications and services running on top of them. The core (light blue) and the extensions made by vehicleLang (dark blue) are depicted in Figure 1. The elements in teal color are also part of the core language, but have been slightly modified within vehicleLang.

4.1 Core Language

The core language consists mainly of entities representing `Machine`, `Vulnerability`, `Account`, `Data`, `Dataflow`, and `Network`. There are still more entities, but in the following, we will only focus on the previously mentioned. A `Machine` represents attackable entities (i.e., hardware or software related) of a system under investigation and, thus, are exposed by a `Vulnerability`. A `Vulnerability` can be exploited by a user represented by an `Account`. An `Account` can also be capable to read, write, and delete `Data` which are stored on a `Machine`. Two `Machines` can exchange `Data` via a `Dataflow` on a `Network`.

Following we will explore those entities with more detail and depict how they and their interrelations are modeled. If one connects to a `Machine`, one can then try to *authenticate* to get *access* to the `Machine`. Another way to get *access* to a `Machine` is to *bypassAccessControl*. If an attacker achieved *access* to a `Machine`, she can: *connect* to all executees (i.e., executed `Software` on this `Machine`), *requestAccess* to all stored `Data`, start *denialOfService* attacks, which can be executed against the executees or to *denyAccess* of certain `Data`. Without being *authenticated* the attacker is, first, able to compromise all privileges related to the connection and, second, exploit the vulnerability related to an account in order to gain its privileges. As far as the attacker attained the `Vulnerability`, she can try to exploit it. A successful *exploited* `Vulnerability` leads to *compromised* privileges of all related `Accounts`.

An `Account` can be *compromised* by either *exploiting* a `Vulnerability` or *authenticating* (e.g., *stealing* the `Credentials`). A *compromised* `Account` allows to overtake other `Accounts` (e.g., a super user can authenticate as each other account on a machine) and use their privileges. Moreover, an attacker can

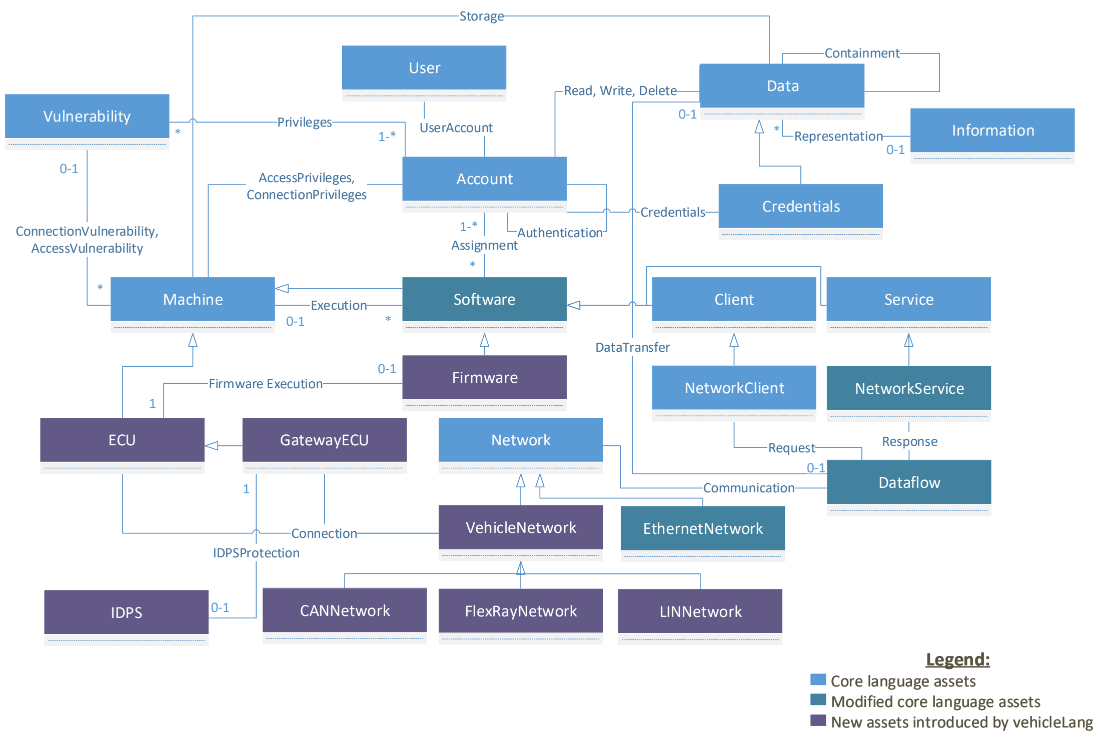

Figure 1: Model of vehicleLang

authenticate at other Machines, which use the same Account. Last, an attacker is capable of *reading*, *writing*, and *deleting* Data on a Machine.

Data is a recursive entity as it can store other Data inside itself. Considering the basic attack steps of Data, there are three: *read*, *write*, and *delete*. To get to these attack steps, the attacker needs to *requestAccess* and the used Account must have granted the corresponding privileges (e.g., *anyAccountRead*). The attack steps *read*, and *delete* behave intuitively. In contrast, *write* allows also to *delete* Data and to *tamper* Data on the Dataflow. The attack step *denyAccess* is the result of a *denialOfService* and prevents the accessibility of the data.

A Dataflow is a logical communication between two Software applications. It allows several different attack steps. First, an attacker can *eavesdrop* on the Dataflow and, therefore, she can *read* the contained data. Second, an attacker can try to carry out a man-in-the-middle attack on the data flow. This leads to the control of the contained Data. In both cases, that Data may be encrypted and authenticated, thus preventing a breach of confidentiality and integrity. Besides *reading*, *writing*, and *deleting*, a *manInTheMiddle* allows also *maliciousRequests* and *maliciousResponds*.

A Network is a physical connection between Machines like Ethernet LANs, the Internet, and Wi-Fi networks. Basically, it provides access to its Dataflows and enables the attacker to perform *denialOfService*, *manInTheMiddle*, and *eavesdrop* attacks.

4.2 vehicleLang

vehicleLang extends the core language by adding the Firmware, ECU, and VehicleNetwork assets. The Firmware, which is some kind of Software, runs on the ECU, which is derived from a Machine. Besides the “simple” ECU, there is also a GatewayECU which acts as gateway to the Network. Moreover, a GatewayECU offers the possibility to activate a firewall and an IDPS (Intrusion Detection and Prevention System) to prevent certain attack steps. The ECUs are connected via the VehicleNetwork to exchange data with each other.

An ECU specifies any ECU, or embedded controller in a vehicle. An ECU is an embedded system in a vehicle controlling one or more electrical system or subsystems like the anti-lock braking system (ABS). Modern vehicles can contain up to 80 ECUs (Ebert and Jones, 2009). We model it as an extension of the existing concept of a `Machine`, since an `ECU` offers many attacks that are not related to the abstract type of a `Machine`. If an attacker is *connected* to an `ECU` she can try to get *access* to the `ECU`, *attemptChangeOperationMode*, or to *modify* the `Firmware`. After compromising `ECU`'s `Firmware`, *bypassingAccessControl*, or properly *authenticating*, the intruder has *access* to the `ECU`. The *access* to an `ECU` can be utilized to *changeOperationMode*, *gainLINAccessFromCAN* (Takahashi et al., 2017), *modify* the `Firmware`, or execute a *serviceMessageInjection* on the related `Services` running on this ECU.

A *changeOperationMode* puts the `ECU` into diagnostics, which is usually used for service mechanics to seek for failures of the system, or into update mode, which is e.g. utilized to install a new `Firmware`. An `ECU` in these modes will no longer send messages and an attacker can imitate it, if she is careful, thus, executing a *bypassMessageConfliction*. Furthermore, the *changeOperationMode* can be protected by the defense *operationModeProtection*. This defense can be implemented in the real world by either preventing diagnostics mode after vehicle starts moving or allowing diagnostics mode only after some physical change is done on the vehicle (e.g., a physical switch is pressed).

bypassMessageConfliction enables the attacker to inject forged service messages that could notify about vehicle's fault or report fake status like speed or operation mode. This can lead to an unresponsive `ECU` (*denialOfService*) or allows to inject messages for the services running on this `ECU` (*serviceMessageInjection*). A possible defense against *serviceMessageInjection* is by using message confliction mechanisms, which act like a host-based IDPS (Kleberger et al., 2011). A host-based IDPS monitors and analyzes the internals of a computing system as well as the network packets on its network interfaces. Therefore, it observes the inbound and outbound packets from the device and alerts if suspicious activities are detected (Newman, 2009).

The `VehicleNetwork` extends the `Network` of the core language and serves as an abstract layer containing the common attack steps and relations of its specializations: `CANNetwork`, `FlexRayNetwork`, and `LINNetwork`. E.g., *accessNetworkLayer* implies the possibility to transmit messages over the network. But, it does not imply the possibility to listen to others' traffic on the network, because the attacker has no access on the `GatewayECU` but is capable to communicate into the network. Therefore, she is able to *connect* to all related `ECU`, perform a *networkSpecificAttack*, or execute a *messageInjection*. Additionally, the *eavesdrop* attack is still inherited from the `Network`.

A *maliciousRequest*/*maliciousRespond* on the `Network` allows to *connect* to the `Machine` on top of which the `Service` or `Client` is running, respectively. For the purpose of protection, an IDPS can be deployed which enforces the attacker in some cases to bypass it. As this takes time, we use an exponential distribution with a rate parameter of 6.13 to estimate the time consumption (Holm et al., 2012; Sommestad et al., 2012).

A `CANNetwork` represents the CAN bus network which is a soft real-time control network used for e.g. anti-lock breaking or engine control (Wolf, 2009). It is subject to two CAN related attacks: *exploitArbitration*, and *busOffAttack*. A *exploitArbitration* for message prioritization in `CANNetwork` can lead to invalidation of legitimate messages and allows *tampering* with the `Dataflow`. *exploitArbitration* is different from the *messageInjection* attack, because it allows direct malicious respond and request.

A *busOffAttack* exploits the error-handling scheme of in-vehicle networks to disconnect or *shutdown* not compromised `ECUs` (Cho and Shin, 2016). The *busOffAttack* is easy to mount if the attacker has access to a `CANNetwork` or specializations of it, because the attacker only needs to discover messages which are sent periodically. Consequently, this removes the need for a costly reverse-engineering procedure for the messages or the network. Additionally, an IDPS cannot distinguish between injected and error messages. As a result, an attacker who has access to a `CANNetwork` can directly apply a *busOffAttack* and a *denialOfService* to all connected `ECU`, no matter if there is an IDPS in place or not.

Cho and Shin (Cho and Shin, 2016) propose a protection against the *busOffAttack* attack, which is modeled with the *busOffProtection* defense. In this case, the *busOffAttack* cannot be performed against the `ECU`. In reality, this defense is two-fold. First, the presence of more than 16 consecutive error frames indicates a *busOffAttack*. However, some errors can also produce so many consecutive error frames. Therefore, second, the protected `ECU` should send a message to the `CANNetwork` and observe if there will be another message with the same ID, something that is by design not allowed on a `CANNetwork`. Consequently, if those two criteria are fulfilled there must be a *busOffAttack* and the `ECUs` can be designed in a way that they do not get shut down.

The `FlexRayNetwork` is a hard real-time control network used for X-by-wire applications e.g.

break-by-wire or emergency braking systems. It allows three special kinds of attacks: *commonTimeBaseAttack*, *exploitBusGuardian*, and *sleepFrameAttack*. The attacker sends for a *commonTimeBaseAttack* more than defined messages within one certain timeframe to make the whole network inoperable (Wolf, 2009, p. 103). Consequently, this leads to a *denialOfService* of the `FlexRayNetwork`.

The Bus Guardian is usually a part of a `FlexRayNetwork` connected `ECUs` and regulates the access to the bus for sending messages to certain, defined time periods (Sung et al., 2008). This Bus Guardian is utilized in the *exploitBusGuardian* attack for sending well-directed faked error messages to deactivate `ECUs` (Wolf, 2009). Therefore, on all `FlexRayNetwork` connected `ECUs` a *shutdown* can be performed. However, the Bus Guardian mechanism is hardened and, therefore, an attacker may need to spend much effort on this attack (Mundhenk et al., 2015).

A `LINNetwork` is a low-level non-real-time subnet which is used for e.g. door locking or light sensors. The attacker can easily *gainLINAccessFromCAN* from the `ECU` (Koscher et al., 2010; Rippel, 2008) on the `LINNetwork` and, consequently, perform the attack step *accessNetworkLayer*. Another possible attack on the `LINNetwork` is *injectBogusSyncBytes*. To perform this attack, the attacker sends frames with bogus synchronization bytes within a SYNCH field. This makes the local LIN network inoperative or causes at least serious malfunctions (Wolf, 2009). This leads in our model to a *denialOfService* of the `LINNetwork`.

A specific attack on the `LINNetwork` which exploits the error handling mechanism is the *injectHeaderOrTimedResponse*. If the attack is successful, it allows to *tamper* the `Dataflow` (Takahashi et al., 2017). To overcome this shortcoming of the `LINNetwork` Takahashi et al. (Takahashi et al., 2017) proposes a protection against this attack, which is modeled as the *headerOrTimedResponseProtection* defense in our model. Additionally, to exploit this attack significant effort is needed (Takahashi et al., 2017).

vehicleLang is an open source project and therefore the code behind the language is publicly available on the internet.

5 EVALUATION

According to Hevner et al. (Hevner et al., 2004), five methods are possible to evaluate the output of DSR: observations, analysis, experiments, tests, and descriptions. Since developing vehicleLang is similar to developing source code, we opt for testing as the evaluation method. We ground our decision on the fact that testing is widely spread in application development and commonly accepted as means to ensure that an application behaves as intended.

More concretely, we apply two different kinds of testing. First, we implement unit tests to ensure that vehicleLang behaves like we expect it to. To ensure our results, we apply, additionally, cross checking by another developer, who works also on a realization of MAL. This cross checking includes a revision of vehicleLang as well as the implementation of further unit tests to uncover unintended behavior. Second, we implement integration tests. Those integration tests rely on a compiled attack list created by a SLR.

Our SLR follows the methodology of Webster and Watson (Webster and Watson, 2002). The scope of the SLR is in line with the objectives of this paper, in other words, identifying attacks on the internal networks of vehicles. Therefore, we searched for the terms “vehicle security”, “in-vehicle protocol security”, “CAN bus attacks”, “LIN bus attacks”, “FlexRay attacks”, “connected cars security”, and “automotive communication security” in title and abstract of articles in the Google Scholar[1], IEEE Xplore[2], and Springer Books[3] databases from 2006 to present. We scanned the abstracts of about 100 found results, filtered them with respect to their relevance to our research objective, and ended up with fourteen conference papers, three long papers, five scientific books, and one ENISA (European Network and Information Security Agency) report relevant to our objective. We completed the SLR by conducting a back- and forward search, which resulted in no more additional papers.

We scanned the identified literature and found 22 possible attacks on seven different classes. We relate to each class its possible attacks and give a short description of the attack steps. Furthermore, we refer to defenses which could be put in place to prevent the attack. We also kept track of the literature in which each attack is mentioned, and linked it to the test cases which models the related behavior.

The following provides a sample test case as depicted in Figure 2. It is based on a scenario where an attacker aims to perform a *maliciousResponse* on the `OtherDataflow` by using as entry point the *physicalAccess* on `VehicleNetwork vNet1`. It simulates two `VehicleNetworks` with two `ECUs`, each one of those running a `Service`. Each `ECU` is placed in its

[1]https://scholar.google.com
[2]http://ieeexplore.ieee.org/Xplore/home.jsp
[3]http://www.springer.com/gp/products/books

own `Network` and the `Networks` are connected to each other via a `GatewayECU`. On this `GatewayECU` the firewall and the IDPS are unfortunately disabled or not present at all. The attacker initially has *access* to the network layer of the `Network` where the `ECU#1` running the `Service` is connected. Consequently, she can also *compromise* the `Dataflows` in this `Network`. However, she only reaches (is able to *connect* to) the `Service` and the respective `ECU`, but is not able to *compromise* them. Same holds for the other `Network`, since the attacker can *compromise* the `Network`, because the firewall and IDPS are disabled, but she is not able to *compromise* the `ECU#2` and the corresponding `Service#2` connected to the other `Dataflow`.

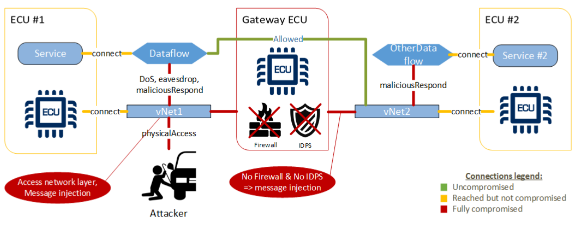

Figure 2: Sample Attack on Vehicle's Network

In total, we conducted more than 50 tests to verify vehicleLang. Those tests confirmed that vehicleLang acts like expected and that the attacks and the possible countermeasures successfully modeled.

When it comes to performance and scalability evaluation, the following notes must be done. Vehicular infrastructure is by many factors smaller compared to an IT infrastructure, therefore, work towards better scalability for vehicleLang was minimal. In more detail, only in two occasions scalability wise decisions were taken. First, the way the dataflows are modeled in vehicleLang, namely, the decision to model a dataflow as a many-to-many or one-to-many connection between the network service and clients. The advantage of this approach is that it significantly decreases the number of needed dataflows in a complete model. Second, another scalability related question was how the firewall allowed dataflows should be modeled. The approach followed was that any firewall allowed dataflow should be connected to all the permitted networks. This decision was taken in favor of easier implementation and higher user friendliness

6 CONCLUSIONS AND FUTURE WORK

Assessing the cyber security of modern vehicles is becoming of paramount importance as more IT and cyber-physical systems pervade them and, simultaneously, the number of IT security issues increases. Within this paper, we presented vehicleLang, a DSL based on the MAL. vehicleLang will foster security analysts in the automotive domain to model vehicles and focus on analyzing possible design weaknesses.

Of course, further work remains. First, we only modeled the internal vehicle networks and the connected components to them but we did not model further components, like the infotainment system, the telematics unit, the connectivity with the internet, etc.

Second, our studies have shown that the expected time consumption of attackers is only researched for one of the included attacks. Consequently, future research should elaborate on uncovering the needed time consumption to create more realistic models. As already mentioned, evaluating an artifact via test cases is feasible, but, third, a complete real world evaluation of vehicleLang is still underway and will be presented in further extensions of this work.

Finally, one more limitation of vehicleLang is that it does not allow to identify zero-day attacks as it relies on known cyber attacks. On the contrary, it allows to uncover unknown combinations of known cyber attacks and, therefore, offers a better foundation for analysis of possible weaknesses.

ACKNOWLEDGEMENTS

This work has received funding from the Swedish Governmental Agency for Innovation Systems (Vinnova).

REFERENCES

Alam, M., Breu, R., and Hafner, M. (2007). Model-driven security engineering for trust management in sectet. *JSW*, 2(1):47–59.

Basin, D., Clavel, M., and Egea, M. (2011). A decade of model-driven security. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 1–10. ACM.

Cho, K.-T. and Shin, K. G. (2016). Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1044–1055, New York, NY, USA. ACM.

Ebert, C. and Jones, C. (2009). Embedded software: Facts, figures, and future. *Computer*, 42(4):42–52.

Ekstedt, M., Johnson, P., Lagerström, R., Gorton, D., Nydrén, J., and Shahzad, K. (2015). securiCAD by foreseeti: A CAD tool for enterprise cyber security management. In *Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International*, pages 152–155. IEEE.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 28(1):75–105.

Holm, H., Shahzad, K., Buschle, M., and Ekstedt, M. (2015). P^{2}CySeMoL: Predictive, probabilistic cyber security modeling language. *IEEE Transactions on Dependable and Secure Computing*, 12(6):626–639.

Holm, H., Sommestad, T., Franke, U., and Ekstedt, M. (2012). Success rate of remote code execution attacks - expert assessments and observations. *Journal of Universal Computer Science*, 18(6):732–749.

Ingols, K., Chu, M., Lippmann, R., Webster, S., and Boyer, S. (2009). Modeling modern network attacks and countermeasures using attack graphs. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 117–126. IEEE.

Johnson, P., Lagerström, R., and Ekstedt, M. (2018). A meta language for threat modeling and attack simulations. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, page 38. ACM.

Jürjens, J. (2005). *Secure systems development with UML*. Springer Science & Business Media.

Kleberger, P., Olovsson, T., and Jonsson, E. (2011). Security aspects of the in-vehicle network in the connected car. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 528–533.

Kordy, B., Mauw, S., Radomirović, S., and Schweitzer, P. (2010). Foundations of attack–defense trees. In *International Workshop on Formal Aspects in Security and Trust*, pages 80–95. Springer.

Kordy, B., Piètre-Cambacédès, L., and Schweitzer, P. (2014). Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer science review*, 13:1–38.

Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., and Savage, S. (2010). Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462.

Kraftfahrt-Bundesamt (2018). Bestand in den jahren 1960 bis 2018 nach fahrzeugklassen. `https://www.kba.de/DE/Statistik/Fahrzeuge/Bestand/FahrzeugklassenAufbauarten/b_fzkl_zeitreihe.html`. [Online; accessed 28-March-2018].

Mauw, S. and Oostdijk, M. (2005). Foundations of attack trees. In *International Conference on Information Security and Cryptology*, pages 186–198. Springer.

Mundhenk, P., Steinhorst, S., Lukasiewycz, M., Fahmy, S. A., and Chakraborty, S. (2015). Security analysis of automotive architectures using probabilistic model checking. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 38:1–38:6, New York, NY, USA. ACM.

Newman, R. C. (2009). *Computer security: Protecting digital resources*. Jones & Bartlett Publishers.

Noel, S., Elder, M., Jajodia, S., Kalapa, P., O'Hare, S., and Prole, K. (2009). Advances in topological vulnerability analysis. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, pages 124–129.

Paja, E., Dalpiaz, F., and Giorgini, P. (2015). Modelling and reasoning about security requirements in socio-technical systems. *Data & Knowledge Engineering*, 98:123–143.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77.

Rippel, E. (2008). Embedded security challenges in automotive designs. In *Proc. Workshop on Embedded Security in Cars (escar 2008)*.

Schneier, B. (1999). Attack trees. *Dr. Dobb's journal*, 24(12):21–29.

Schneier, S. (2000). Lies: digital security in a networked world. *New York, John Wiley & Sons*, 21:318–333.

Smith, C. (2016). *The Car Hacker's Handbook: A Guide for the Penetration Tester*. No Starch Press.

Sommestad, T., Holm, H., and Ekstedt, M. (2012). Estimates of success rates of remote arbitrary code execution attacks. *Information Management & Computer Security*, 20(2):107–122.

Statistiska centralbyrån (2018). Fordonsstatistik januari 2006–februari 2018. `http://www.scb.se/hitta-statistik/statistik-efter-amne/transporter-och-kommunikationer/vagtrafik/fordonsstatistik/pong/tabell-och-diagram/fordonsstatistik/`. [Online; accessed 28-March-2018].

Sung, G.-N., Juan, C.-Y., and Wang, C.-C. (2008). Bus guardian design for automobile networking ecu nodes compliant with flexray standards. In *2008 IEEE International Symposium on Consumer Electronics*, pages 1–4.

Symantec (2017). Internet security threat report. `https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf`. [Online; accessed 28-March-2018].

Takahashi, J., Aragane, Y., Miyazawa, T., Fuji, H., Yamashita, H., Hayakawa, K., Ukai, S., and Hayakawa, H. (2017). Automotive attacks and countermeasures on lin-bus. *Journal of Information Processing*, 25:220–228.

Webster, J. and Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, 26(2):xiii–xxiii.

Williams, L., Lippmann, R., and Ingols, K. (2008). *GARNET: A graphical attack graph and reachability network evaluation tool*. Springer.

Wolf, M. (2009). *Security engineering for vehicular IT systems—improving trustworthiness and dependability of automotive IT applications*. Vieweg + Teubner.

Wolf, M., Weimerskirch, A., and Paar, C. (2004). Security in automotive bus systems. In *Proceedings of the Workshop on Embedded Security in Cars*.