ADSecLang: A Domain-Specific Language for Cybersecurity Testing of Autonomous Vehicles

Andrew Roberts[1], Jingyue Cheng[2], Olaf Maennel[3], Mohammad Hamad[2], Sebastian Steinhorst[2]
[1]FinEst Centre for Smart Cities, Estonia
[2]Technical University of Munich,Germany
[3]University of Adelaide, Australia

***Abstract*—Domain-specific languages for safety validation testing have reduced the complexity of safety scenario generation and enhanced the adoption of Autonomous Driving (AD) safety testing. Yet, there is a lack of comparable solutions for cybersecurity testing. In this work, we present *ADSecLang*, a domain-specific language for cybersecurity testing of AD systems. *ADSecLang* provides a concise syntax that enables the tester to construct scenarios for AD cybersecurity straightforwardly that can be implemented in an AD test simulation platform. The proposed language is validated within the CARLA AD simulation platform in a use-case scenario of two diverse AV camera sensor manipulation attacks on a state-of-the-art trajectory-guided AD solution. The results show that the language is able to support the translation of threats from an abstract description to the technical implementation of the attack test cases and that these test cases could identify vulnerabilities in the target AD solution.**

***Index Terms*—Autonomous Driving, Cybersecurity testing**

I. Introduction

Vulnerability testing of autonomous driving (AD) to cyber attacks is a burgeoning field of research. Initial contributions to this field have focused on novel vulnerability discovery utilising penetration testing methods [1] [2] and fuzzing [3], [4]. However, there exists a gap between this novel, experimental work and the practical implementation of testing to validate the operational readiness of real-world AD systems. Real-world, operational AV testing requires a more rigid approach centred on a structured testing methodology aligned to composite vehicle development and test validation processes. For safety validation testing, domain-specific languages for safety scenario generation, such as SCENIC [5] and ASAM OpenSCENARIO [6], provide a systematic expression that enables a common taxonomy, traceability of testing processes and repeatability and automation of testing for scalability. Yet, there exists a sparsity of research on the development of a domain-specific language for cybersecurity testing of AD systems. One of the primary benefits of the development of a domain-specific language for cybersecurity is that it can simplify the task of writing scenarios for security by providing a concise syntax. In addition, the lack of a domain-specific taxonomy for cybersecurity testing of AD systems further challenges the development and evaluation of AD security testing tools, processes, and methods.

This paper aims to develop such a language, which we call ADSecLang. ADSecLang acts as an intermediary layer in the testing process, which constructs scenarios for cybersecurity through the translation of functional threat descriptions to concrete test cases. Figure 1 depicts the scenario-based abstraction of ADSecLang, which represents the incremental and iterative definition of the threat scenario. Firstly, the abstract description of the threat scenario originates from adversarial analysis, which can leverage sources such as threat libraries, system data, and other knowledge-base repositories. Secondly, a logical, syntactical expression of the threat scenario is created using a taxonomy. Finally, the technical description of the threat scenario is integrated within the AD simulation testing platform. ADSecLang aims to contribute greater intuition through readable, concise syntax for the development of adversarial agents in simulation testing that would otherwise require complex expressions and constraints. ADSecLang requires the tester to consider all elements of the threat model from attacker tools to desired attack outcomes at both an abstract and parameterised level of abstraction. To demonstrate the utility of ADSecLang, we initially focus on semantic AI security and we evaluate the language to support two use-case scenarios of a camera manipulation attack. In summary, the main contributions of our work are:

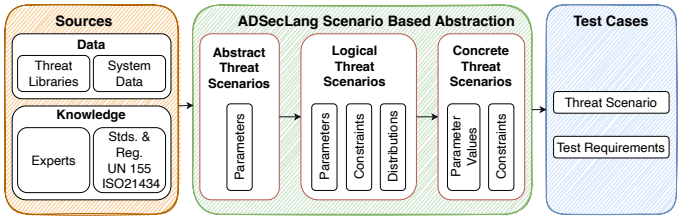

Fig. 1: ADSecLang: scenario-based abstraction view.

- We provide an *Attack Taxonomy* which provides common knowledge for the construction of cybersecurity test scenarios (Sec. II-A).
- We present *ADSecLang*, a domain-specific language for autonomous vehicle cybersecurity testing (Sec. II).
- We use *ADSecLang* to identify vulnerabilities in a state-of-the-art trajectory-guided end-to-end AD solution (Sec. III).

II. ADSecLang: The Proposed Solution

This section introduces the attack taxonomy used to support the development of ADSecLang and presents the cybersecurity testing framework where ADSecLang can be adopted.

A. Attack Taxonomy

The attack taxonomy of ADSecLang (as shown in Fig. 2) categorizes cyber attacks into two domains: **Action** and **Impact**.

1) Action: represents the execution of an attack method. The success of an action depends on the fulfillment of one or more preconditions. As a result, we subdivide the Action domain into two sub-domains: **Method** and **Preconditions**. The **Method**

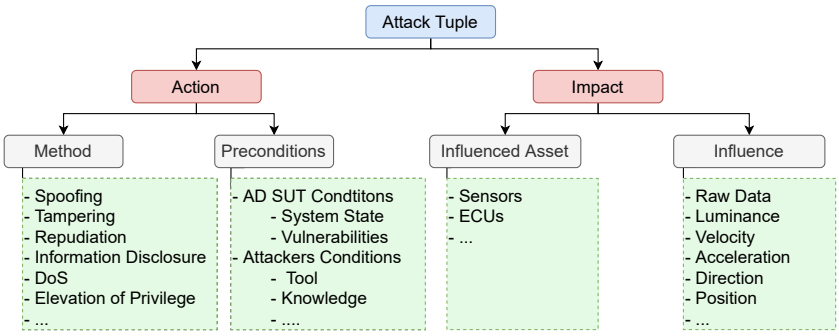

Fig. 2: Attack Taxonomy - Detailed Description.

is defined as the threat technique. This threat description can be derived from a functional description such as STRIDE, Attack Trees, or a textual interpretation. **Preconditions** are a set of conditions that must be met to execute an attack. These preconditions must be inherent attributes that already exist and are not generated by the execution of the attack. The preconditions can be further divided into two categories: conditions on the AD System-Under-Test (SUT) and conditions on the attacker.

- *AD SUT Conditions* are categorized into requirements for the state of the tested system and vulnerabilities within the system. System state conditions refer to the requirement that the target system must be in a specific state (such as a particular version of an operating system, system software/application, or a specific hardware/software state, such as firmware update status) for the attack to be executed. System vulnerabilities refer to exploitable weaknesses in the system's design and operation.
- *Attacker Conditions* can be further segmented into three types: attack tools, attacker knowledge (capabilities, skills), and the level of privileges that an attacker can obtain. The tools and knowledge of the attacker help to profile the type of threat actor capable of conducting the attack. The level of privileges refers to the permissions needed to access or manipulate target system assets. An example would be permission to run processes on the target or existing access to the target asset to manipulate data.

Some attack methods can only be executed successfully when multiple preconditions are met simultaneously. Such conditions will be grouped within braces {}. For example, the precondition `[{A, B}, C]` can be interpreted as 'A and B must be met simultaneously, or C must be met'. To encompass the requirement for multiple preconditions, we define an **Action Group**:

```
action: [method, preconditions]
  method: [category, description]
  preconditions: [precond1, precond2, ...]
    precond1: [category, description]
    precond2: [category, description]
```

2) Impact: Executing an *Action* will introduce one or more **Impacts** into the system. In other studies [7], these impacts are also denoted as consequence and effect. Although *Impacts* represent the outcomes and effects of attacks, they can also serve as preconditions for subsequent attacks. Consequently, some researchers [8] have alternatively referred to them as post-conditions. In our taxonomy, the utilization of *Impact* aims to identify the direct consequences of an *Action*, which may additionally serve as preconditions for further attacks. The term 'goal' in the attack model represents the ultimate impact. The dimension of *Impact* can be subdivided into two sub-dimensions, namely **Influenced Assets** and **Influence**, which serve to identify the assets directly affected by the *Action* and ascertain the direct impact incurred on these assets. **Influenced Assets** can be characterized by their respective category and name. For example, the sensor category can include cameras, radars, LiDAR, GPS, or any other AD sensor. The electronic control unit (ECU) category comprises brake control ECUs, engine control ECUs, tire pressure monitoring ECUs, or any other vehicular ECU. **Influence** can be specified as its *Parameter* and *Value*, denoting the specific parameter influenced by the attack and the corresponding altered value, respectively. For instance, if we aim to adjust the brightness of an image captured by a camera, we should specify the parameter as luminance and set its value to 300% (indicating that the brightness has been increased to 300% of the original brightness). To achieve the scalability of ADSecLang, users can add new parameters and a value range in the property configuration file. The **Impact Group** is defined as follows:

```
impact: [influenced_asset, influence]
  influenced_asset: [category, name]
  influence: [parameter, value]
```

B. Semantics of ADSecLang

The safety scenario domain-specific languages are based on scenario abstraction methodologies such as the Pegasus method [6], which segments three levels of abstraction of the scenario: 1) abstract, 2) logistic, and 3) concrete. For example, an abstract scenario could be described as: '*A malicious actor motivated to cause a safety violation using a laser beam device targeted at a car*'. The logical scenario might be: '*A malicious actor using a laser beam device projecting a luminance of approximately 100 to 300% with a pulse width of 0 to 1*'. Finally, the concrete scenario would specify: '*A malicious actor using a laser beam device projecting 300% luminance with a pulse width of 1*'. Within ADSecLang, the abstract describes the cyber threat scenario according to local parameters. The logical cyber threat scenario extends this description by adding parameter value ranges. Finally, the concrete scenario description contains the set parameter values, which will be utilized as the scenario implementation within the AD simulation testing platform. ADSecLang is designed as an extension of the safety scenario

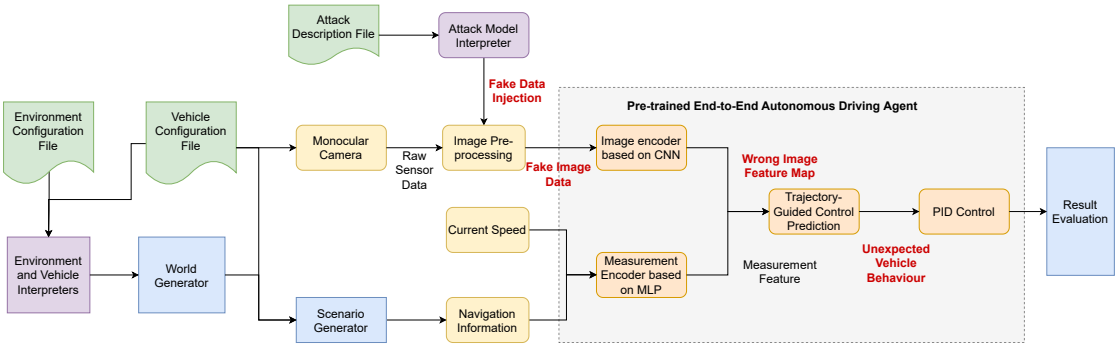

Fig. 3: ADSecLang Cybersecurity Testing Workflow - Camera Attack.

languages [5], [6], using the same abstraction method, language semantics and syntax. ADSecLang provides an extension to these areas for cybersecurity.

C. Compilation of ADSecLang

Compilation of ADSecLang involves three configuration files. Each file contains various user-defined parameters:

- *Environment Configuration File*: This file provides adjustable parameters for scene generation, including town, weather, and traffic density. It also allows users to define constraints on these parameters for scene sampling.
- *Vehicle Configuration File*: This file allows the user to define the parameters of the autonomous vehicle; these include the sensors required, the location of the sensors in the vehicle, the type of sensors, the data to be recorded, and the frequency of recording.
- *Attack Description File*: This file is formatted in the YAML syntax and allows users to define the attack model.

The first two configuration files are relatively simple: the *Environment Configuration File* and the *Vehicle Configuration File*. The environment and vehicle configurations stored in their respective configuration files are read as parameters for generating the driving simulation world and transferred to the world generator. The *Attack Description File* is a more complicated design which has two functions:

- The attack description file is utilized to extract the parameters, which are then translated into concrete code implementation for data processing based on the corresponding attack parameters.
- It is also responsible for connecting the simulation environment, attack code, and autonomous driving system. The attack description file defines the input and output interfaces of the attack code. The input interface connects real-time data captured by sensors in the rendering engine in a simulation environment, such as images captured by camera sensors or status information of ECUs. The output interface sends malicious data generated by attacks to the target AD solution.

D. Cybersecurity Testing Framework

1) Architecture: The proposed cybersecurity testing framework has diverse modules for environmental, vehicle, and attack configuration, simulation test, and result evaluation (Fig. 3). The functions and roles of these modules are as follows:

- *Environment and Vehicle Interpreter:* Reads the environment and vehicle configuration stored in their respective configuration files as a parameter for generating the world.
- *Attack Model Interpreter:* We read the attack description file as attack parameters. We have defined input and output interfaces for the attack model. The input interface obtains images captured by sensors in the real-time rendering engine and completes the data processing corresponding to the attack parameters read by the interpreter in the specific implementation code of the interface. The specific implementation of the output interface is to send the output of the attack model to the user's chosen autonomous driving solution.
- *World Generator:* Initialise the world based on the environment and entity parameters read by the environment interpreter, including object properties and attribute distribution functions. The world generator randomly samples from the distribution function whenever it is called. By reading the sampling results of the world initializer, a specific world is generated in the real-time rendering engine according to certain steps. The generated world contains at least one vehicle and one camera sensor and exposes the calling interfaces of the vehicle and sensors to the attack model.
- *Scenario generation and result evaluation:* We use a CARLA plugin called CARLA Leaderboard [9] to provide us with scenario generation and evaluation of driving violations. Violation testing includes route completion testing, collision testing, red light running testing, stop running testing, lane crossing testing, proxy blocking testing, and timeout testing.

2) Cybersecurity Testing Workflow: The overall workflow of the system is shown in Fig. 3. The attack target system illustrated here is an end-to-end autonomous driving system based on a monocular camera. The target asset in the vehicle of the attack is the monocular front RGB camera.

The workflow is initiated by storing the predefined environment configuration, object properties, and attack description in configuration files. Execution of the *World Generator* uses the *Environment and Vehicle Interpreter* to read the environment information. Subsequently, each time the scenario is generated, sampling is carried out according to the predetermined process, and the sampling results are converted into the parameterized form we designed and then handed over to the *World Generator*. The *World Generator* first initializes the basic configuration of the real-time rendering engine and creates a specific world in

the it, step by step, based on the obtained parameters. Once the world is created, the *Scenario Generator* starts generating test scenarios based on the preset parameters. Subsequently, the *Attack Model Interpreter* retrieves the attack information from the *Attack Description File* and injects the manipulated data to the end-to-end AD system based on the parameters specified by the attack model. Finally, the *Results Evaluation* checks conformity of the AV to safety metrics, as aforementioned, as part of the CARLA Leaderboard [9]. Through conducting multiple iterations of the testing workflow it is possible to evaluate the effectiveness of the attack model.

III. Evaluation Case Studies

This section examines the use of ADSecLang for supporting the security testing of AV systems. It includes a description of the experimental setup (Sec. III-A) and an analysis of results derived from two attack scenarios (Sec. III-B and III-C). The goal of the experiments is to assess the ability of ADSecLang to generate attack test cases capable of identifying vulnerabilities in AD systems.

A. Experimental Setup

The experiments were run on a desktop computer with 12th Gen Intel(R) Core(TM) i3-12100F 4-Core Processor, NVIDIA GeForce GTX 1070Ti GPU, and 16 GB RAM. The use-case scenario testing is conducted on the simulator CARLA 0.9.10. The AD solution tested in the following experiments is a trajectory-guided end-to-end AD solution [10]. This AD solution achieves a new state-of-the-art performance on the CARLA AD Leaderboard [9], in which they rank first in terms of the Driving Score and Infraction Penalty using only a single camera as input. The image captured by the camera has a resolution of 900×256 pixels, and the field of view is maximized at 100 degree.

B. Attack Case I - Strong Light Exposure Attack

1) Attack Design: State-of-the-art camera attacks [11] have shown that strong white LED light directed at the camera sensor will result in significantly higher hue values and cause the entire image to be completely white. This results in the camera being unable to capture any visual information. This attack is based on the fact that CMOS/CCD sensors can be interfered with by malicious optical inputs and will produce unrecognizable images. The broken image will further affect the victim AV's decision control. As a result, it will cause uncertainties, which may lead the victim's AV to deviate or emergency brake, both of which can lead to a collision and/or other safety violations. Common methods of attacking camera devices are lasers or LEDs. The Strong Light Exposure Attack interferes with the camera's automatic exposure control. Under laser irradiation, the surface temperature will rise rapidly due to the non-uniform temperature field. Avalanche breakdown of semiconductor materials will cause irreversible damage to optoelectronic devices. Whilst we cannot achieve the physical effects of a targeted light on the camera sensor in a virtual simulator, we can modify the data to simulate the profile of the cyber-physical attack.

(a) Before (b) After

Fig. 4: Camera view of attack case 1: before (a) and after (b) the implementation of the Strong Light Exposure Attack.

2) ADSecLang Attack Configuration: The concrete scenario using the ADSecLang attack interpreter file is provided below.

```
attack_name: strong light exposure attack
attack_target: monocular camera-based end-to
    -end autonomous driving system
attack_goal: safety hazard
     action: [method, preconditions]
           method: [tampering, modifying the
             data captured in the asset]
           preconditions: [{precond1 AND
             precond2 AND precond3}]
            precond1: [attacker's knowledge,
             the attacker knows the basic
             information about the cameras on
             the victim's autonomous driving
             vehicle]
            precond2: [attack tool, strong LED
              light]
            precond3: [attacker's capability,
             attackers can shine LED light at
             AV camera sensor]
     impact: [influenced_asset, influence]
           influenced_asset: [sensor,
             rgb_camera_front]
           influence: [luminance, 300%]
```

The attack description YAML file is translated using the attack interpreter within the simulation platform.

```
    if(config['attack name']=="Strong light
        exposure attack"):
    percentage = config['impact']['influence
        ']['luminance']
    file.write('    data = cv2.cvtColor(data
        , cv2.COLOR_RGB2YUV)\n')
    file.write('    h = data.shape[0]\n')
    file.write('    w = data.shape[1]\n')
    file.write('    for i in range(h):\n')
    file.write('        for j in range(w):\n
        ')
    file.write('            y = data[i][j
        ][0]*'+str(float(percentage[:-1]) /
        100.0)+'\n')
    file.write('            if y > 255:\n')
    file.write('                y = 255\n')
    file.write('            data[i][j][0] =
        int(y)\n')
    file.write('    data = cv2.cvtColor(data
        , cv2.COLOR_YUV2RGB)\n')
```

3) Results: From the comparison of Fig. 4a and Fig. 4b, we can see that the Strong Light Exposure Attack was successfully implemented. On initiation of the malicious change to the luminance, the monocular camera perception fails to identify the lane lines in the field of view. As a result, the victim AV veered off the lane onto the sidewalk, entering the off-road section of the driving environment. It lost perception and traversed the oncoming lane after being subjected to the Strong Light Exposure Attack. This immediately triggered the failure

TABLE I: Evaluation result of attack case 1.

Criterion	Result	Value
RouteCompletionTest	FAILURE	8.06 %
OutsideRouteLanesTest	FAILURE	11.79 %
CollisionTest	SUCCESS	0 times
RunningRedLightTest	SUCCESS	1 times
RunningStopTest	SUCCESS	0 times
InRouteTest	SUCCESS	
AgentBlockedTest	SUCCESS	
Timeout	SUCCESS	

TABLE II: Evaluation result of attack case 2.

Criterion	Result	Value
RouteCompletionTest	FAILURE	71.3 %
OutsideRouteLanesTest	SUCCESS	0 %
CollisionTest	SUCCESS	0 times
RunningRedLightTest	FAILURE	1 times
RunningStopTest	SUCCESS	0 times
InRouteTest	SUCCESS	
AgentBlockedTest	SUCCESS	
Timeout	FAILURE	

of the *Outside Route Test* and the *Route Completion Test*, terminating the simulation, as presented in Table I.

C. Attack Case II - Laser Beam

1) Attack Design: Adversarial machine learning (ML), as a form of cyber attack, involves designing a targeted numerical vector to make ML models misjudge and cause system failures and crashes. In this attack test case, the laser construction process is determined by several local-parameters: *intercept*, *injection Angle*, *wavelength*, and *laser width*. This laser attack is executed by randomly selecting a parameter and generating adversarial samples. If the confidence level of the classification is reduced, the current parameter settings are retained, which is often similar to the greedy strategy. After adding a laser beam projection to an image, the image pixels change, which in turn affected the results of the classifier. This adversarial attack, when applied to AD, can target the recognition of traffic lights, speed limit signs, and stop signs. Shining a laser on a stop sign can cause the AD system to fail to identify it correctly, leading to a violation of the required safety condition to stop the vehicle. Also, shining a laser on a traffic light can also create color spoofing attacks. Experimentation with laser beam attacks has shown that if the laser covers the entire traffic light, regardless of its color, the accuracy of detecting red or green lights is hardly affected. However, if the laser only shines on one traffic light, there will be a significant decrease in the recognition of the traffic light [12]. However, in our testing, we found that if we use this greedy strategy to search for the optimal parameters for 4000 cycles, the generation of adversarial samples is very slow, and it is impossible to inject adversarial samples into the AD test in real time. Therefore, we generate a laser that can make target recognition ineffective and recognise it as another object, by inputting images captured by the camera into an adversarial sample generation program. We then inject this laser in real-time in the AD test scenario. As in the previous case, we assume that the attacker can find appropriate attack scenarios and not be detected by others in advance. For example, the attacker can deploy multiple infrared light sources next to the road where the attacker's vehicle must pass or on a drone.

2) ADSecLang Configuration: The cyber threat scenario description using the ADSecLang is provided below.

```
attack_name: laser beam attack
attack_target: monocular camera-based end-to
    -end autonomous driving system
attack_goal: safety hazard
   action: [method, preconditions]
    method: [spoofing, shooting laser on the
     camera]
    preconditions: [{precond1, precond2,
     precond3}]
     precond1: [attack tool, laser pointer]
     precond2: [attacker's knowledge, machine
     learning adversarial sample generation
     technology]
     precond3: [attacker's capability,
     attackers can aim lasers at camera
     sensors on the roadside]
   impact: [influenced asset, influence]
    influenced_asset: [sensor,
     rgb_camera_front]
    influence: [raw_data, spoofed data]
```

The attack description YAML file is translated using the attack interpreter within the simulation platform.

```
if(config['attack name']=="Laser beam attack
    "):
file.write('    laser_pattern = cv2.imread("
    laser_for_carriage.png")\n')
file.write('    if laser_pattern is None:\n'
    )
file.write('        print("read image fail
    !!")\n')
file.write('        return 0\n')
file.write('    laser_pattern = cv2.cvtColor
    (laser_pattern, cv2.COLOR_BGR2RGB)\n')
file.write('    data = data.astype(np.
    float32)\n')
file.write('    laser_pattern =
    laser_pattern.astype(np.float32)\n')
file.write('    data = cv2.addWeighted(data,
     1.0 , laser_pattern, 1.0 , 0)\n')
file.write('    data = np.clip(data, 0.0,
    255.0).astype("uint8")\n')
```

3) Results: From the comparison of Fig. 5, we can see that the laser beam attack was successfully implemented in the AD simulation.

The attack achieved its objective of inducing AV behaviour to violate a safety condition. As shown in Table II, the vehicle completed approx. 70% of the route (*Route Completion Test*) and violated a safety condition by driving through a red light (*Running Red Light Test*). The result of the laser attack demonstrated that the laser beam was able to perturb the AD solutions perception of the traffic light, thus causing the victim AV to run a red light.

IV. Related Work

ADSecLang distinguishes itself from the state-of-the-art as it is the only domain-specific language, to our knowledge, for AD cybersecurity testing and it is designed to integrate within a software simulation testing environment for AD systems. Furthermore, the language has been designed to be agnostic to AD solutions or sensor technology and adaptable to accommodate diverse threat scenarios. SCENIC has been utilized to develop driving scenarios for cybersecurity testing. Salgado et al. [13]

(a) Before (b) Generated Laser (c) After

Fig. 5: Camera view of test case 2: (a) before the attack, (b) the generated laser beam, and (c) after applying the attack.

used the abstract and concrete scenario composition of SCENIC to create a scenario of a malicious leading vehicle in a convoy to test the robustness of cruise control and collision avoidance. This scenario demonstrates the effect if an attack had already succeeded, whereas the aim of ADSecLang is to incorporate the technical method of attack to assess the performance.

For more conventional threat modeling, *VehicleLang* [14] and *ALLIA (Agnostic Domain Specific Language for Implementing Attacks in an Automotive Use Case)* [15] are the two most prominent studies for legacy automotive architectures. Both of these solutions are focused on modeling cyber threats to connected vehicular systems and focus their case study evaluations on vehicular communication protocols and connected components. *VehicleLang* provides a conceptual contribution, which is the generation of text-based test cases whose feasibility can be validated by expert opinion. *ALLIA* extends this work by providing a technical implementation, which transforms the text-based test case generation into a technical test case implementation.

V. Conclusion

In this paper, we presented ADSecLang, a domain-specific language for autonomous vehicle cybersecurity testing. As part of the development of the language, we derived a taxonomy for AD cyber attacks and used the taxonomy to translate functional descriptions of threats to a domain-specific language that can be interpreted in the AD simulation testing platform. We demonstrated the feasibility and utility of ADSecLang to support a use-case evaluation of diverse attack scenarios on the camera sensor. The results demonstrated that ADSecLang was successful in generating attacks that could find vulnerabilities in a trajectory-guided end-to-end AD algorithm.

Future work for the development of ADSecLang will be to extend the language to encompass more diverse semantic cybersecurity scenarios and evaluate the utility of the language to support system-level attack scenarios (Buffer Overflow, Denial-of-Service, Network Attacks, etc.). We further aim to improve the results evaluation module. Metrics for AD testing predominantly focus on safety impacts, however, we would consider it necessary to define metrics that assist in directly evaluating the security of the system under test. Whilst this has proven a difficult challenge, the contemporaneous work on benchmarking for machine learning security and cybersecurity assurance levels (CALs) for automotive systems as conducted by the autonomous vehicle cybersecurity standardisation bodies provides some guidance how to achieve this. We further see the importance of integrating the language within a common AD cybersecurity testing evaluation platform, such as Simutack [16], for an open-source release.

Acknowledgment

This work is supported by the European Union-funded project CyberSecDome (Agreement No.: 101120779) and also co-funded by the European Union and Estonian Research Council via project TEM-TA5.

References

[1] R. S. Hallyburton, Y. Liu, Y. Cao, Z. M. Mao, and M. Pajic, "Security analysis of Camera-LiDAR fusion against Black-Box attacks on autonomous vehicles," in *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Aug. 2022.

[2] Y. Cao, S. H. Bhupathiraju, P. Naghavi, T. Sugawara, Z. M. Mao, and S. Rampazzi, "You can't see me: Physical removal attacks on lidar-based autonomous vehicles driving frameworks," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, 2023.

[3] S. Kim, M. Liu, J. J. Rhee, Y. Jeon, Y. Kwon, and C. H. Kim, "Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CSS '22)*. ACM Press, 2022.

[4] Z. Wan, J. Shen, J. Chuang, X. Xia, J. Garcia, J. Ma, and Q. A. Chen, "Too Afraid to Drive: Systematic Discovery of Semantic DoS Vulnerability in Autonomous Driving Planning under Physical-World Attacks," in *Network and Distributed System Security (NDSS) Symposium*, 2022.

[5] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: A language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM Press, 2019.

[6] H. Chen, H. Ren, R. Li, G. Yang, and S. Ma, "Generating autonomous driving test scenarios based on openscenario," in *2022 9th International Conference on Dependable Systems and Their Applications (DSA)*, 2022.

[7] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Conference on Security*, 2011.

[8] H. S. Lallie, K. Debattista, and J. Bal, "A review of attack graph and attack tree visual syntax in cyber security," *Computer Science Review*, 2020.

[9] CARLA, "Carla autonomous driving leaderboard." [Online]. Available: https://leaderboard.carla.org/

[10] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. Qiao, "Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline," in *NeurIPS*, 2022.

[11] C. Yan, W. Xu, and J. Liu, "Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle," *Def Con*, 2016.

[12] R. Duan, X. Mao, A. K. Qin, Y. Chen, S. Ye, Y. He, and Y. Yang, "Adversarial laser beam: Effective physical-world attack to dnns in a blink," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2021, pp. 16 057–16 066. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.01580

[13] I. F. Salgado, N. Quijano, D. J. Fremont, and A. A. Cardenas, "Fuzzing malicious driving behavior to find vulnerabilities in collision avoidance systems," in *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2022, pp. 368–375.

[14] S. Katsikeas, P. Johnsson, S. Hacks, and R. Lagerström, "Vehiclelang: A probabilistic modeling and simulation language for modern vehicle it infrastructures," *Computers & Security*, vol. 117, p. 102705, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822001031

[15] C. Wolschke, S. Marksteiner, T. Braun, and M. Wolf, "An agnostic domain specific language for implementing attacks in an automotive use case," in *Proceedings of the 16th International Conference on Availability, Reliability and Security (ARES)*. ACM Press, 2021.

[16] A. Finkenzeller, A. Mathur, J. Lauinger, M. Hamad, and S. Steinhorst, "Simutack - an attack simulation framework for connected and autonomous vehicles," in *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, 2023.