TOWARDS ENHANCING SECURITY AND SAFETY IN MODERN VEHICLES

PENGFEI JING

PhD

The Hong Kong Polytechnic University

2025

The Hong Kong Polytechnic University

Department of Computing

Towards Enhancing Security and Safety in Modern Vehicles

Pengfei Jing

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

Jun 2024

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Pengfei Jing
_____ (Name of student)

Dedication

To my grandmother, Xuebi Yang: Your unconditional love nurtured me. Your virtues of diligence, perseverance, and selflessness will be passed down, generation by generation.

Abstract

The advent of modern vehicles has seen a paradigm shift from purely mechanical systems to highly sophisticated entities, underpinned by advanced Electronic Control Units (ECUs) and intricate In-Vehicle Networks (IVN). These advancements facilitate a host of new functionalities, including remote control and autonomous driving, yet concurrently raise significant security and safety concerns. This thesis endeavors to tackle these issues, focusing on enhancing the security of modern vehicular systems and ensuring the safety of autonomous driving mechanisms.

Revisiting Automotive Attack Surfaces. The complexity of modern vehicles, characterized by their extensive external attack surfaces and complex internal IVN topology, poses a substantial challenge to cybersecurity. Despite efforts by existing standards such as WP29 R155e and ISO 21434 to provide a baseline, their effectiveness against evolving threats remains questionable. Through an in-depth interview with 15 industry experts, we uncovered significant limitations in current security practices and regulatory frameworks. We propose CarVal, a novel datalog-based methodology that leverages an enhanced threat database to infer multi-stage attack paths, assess risks more efficiently in IVNs, and uncover new attack surfaces by analyzing five real-world vehicles. This approach not only identifies the inadequacies in existing regulations but also introduces a more effective

mechanism for threat analysis and risk assessment in automotive systems.

Enhancing Autonomous Driving Safety. From the autonomous driving standpoint, we focus on the perception and control modules. Our first investigation reveals vulnerabilities in the lane detection module of a real vehicle, highlighting its susceptibility to misdirection through minimal, strategically placed road markings. We developed a two-stage approach to automatically generate these markings, significantly impacting steering decisions without detection by human drivers, as demonstrated through experiments on a real vehicle equipped with Autonomous Driving Systems (ADS). Concurrently, we turn our attention to the control module of ADS, where we pinpoint a critical oversight in existing safety research. By proposing new metrics and enhancing fuzzing methodologies, we conducted comprehensive evaluations on Apollo's Model Predictive Controller (MPC). The findings unearthed significant defects, underscoring the inability of Apollo's controller to perform basic maneuvers and identifying 14 new bugs, subsequently acknowledged and addressed by the development team. This dual-focused inquiry not only sheds light on previously overlooked vulnerabilities but also sets the groundwork for more robust autonomous driving systems.

In conclusion, this thesis identifies critical security and safety vulnerabilities in modern vehicles and autonomous driving systems, and proposes innovative methodologies for their mitigation. Through the application of CarVal, we demonstrate the potential for automated threat analysis and risk assessment in improving automotive cybersecurity. Furthermore, our investigations into the lane detection and control modules of ADS highlight the need for robust testing mechanisms to uncover and address subtle yet significant vulnerabilities. Looking forward, the ongoing evolution of vehicle technologies and attack vectors necessitates contin-

uous refinement of security and safety measures.

Keywords: Cyber Physical System, Vehicular Security, Autonomous Driving Safety

Publications Arising from the Thesis

1. **Pengfei, Jing**, Zhiqiang Cai, Yingjie Cao, Le Yu, Yuefeng Du, Wenkai Zhang, Chenxiong Qian, Xiapu Luo, Sen Nie, and Shi Wu (2023). "Revisiting Automotive Attack Surfaces: a Practitioners' Perspective". In: 2024 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, pp. 80–80.

2. **Pengfei, Jing**, Qiyi Tang, Yuefeng Du, Lei Xue, Xiapu Luo, Ting Wang, Sen Nie, and Shi Wu (2021). "Too good to be safe: Tricking lane detection in autonomous driving with crafted perturbations". In: 30th USENIX Security Symposium (USENIX Security 21), pp. 3237–3254.

3. **Pengfei, Jing**, Sen Nie and Shi Wu, Zijiang Yang, Xiapu Luo (2025). "Expectations aren't Guarantees: Identifying the Inconsistency Between Planning and Control in Autonomous Driving". In: Proceedings of the 2025 on ACM SIGSAC Conference on Computer and Communications Security. 2025 (Under Review).

Acknowledgements

First and foremost, I would like to express my gratitude to my advisor, Daniel Xiapu Luo. He has the most pure pursuit of research. His serious attitude towards research and consistently high standards have profoundly influenced all of his students and colleagues, including myself. Daniel possesses a broad and profound knowledge across many fields and knows precisely how to conduct meaningful research. He taught me the importance of being down-to-earth and getting my hands dirty to produce quality research. I feel incredibly fortunate to have completed my Ph.D. under Daniel's guidance.

I wish to thank Keen Lab for their continuous support throughout my Ph.D., which significantly broadened my perspective with an industrial viewpoint. I am grateful to Sen Nie, Zhiqiang Cai, Yuefeng Du, Wenkai Zhang, Qiyi Tang, Huanwei Liu, Shi Wu, and other outstanding researchers and engineers at Keen Lab, whose extreme dedication to research has deeply inspired me.

I am thankful to all the teachers and classmates who have helped me on my research journey. My gratitude goes to Prof. Ting Wang, Prof. Chenxiong Qian, Dr. Lei Xue, Dr. Le Yu, Dr. Hao Zhou, Dr. Yutian Tang for their guidance. I also appreciate the selfless support from Kaifa Zhao, Yangyang Liu, Shiyao Zhou, Wenying Wei, Zihao Li, Weimin Chen, Yingjie Cao, Zhihao Wang, Xinyu Ji, Tianqi Li,

Zuchao Ma, Kunsong Zhao, Junjie Ma, whose companionship and exchanges have been incredibly beneficial in both my personal and professional life.

Thanks to my band members from the band *Nuts*, *After Dinner*, and *Seafood-Cake*. Although we have all moved on to bigger stages in life, I sincerely wish everyone shines with their unique brilliance. Music connects us, and I hope you always feel the pure joy it brings, just like the first time we were touched by the power of music. I also believe we will reunite one day. As Yongchao said, "*SeafoodCake* will definitely perform on stage again". Yes, definitely.

I am grateful for all my close friends. Thanks to Qi Zhou for the invaluable advice at crucial moments in my life. My life is filled with joy thanks to the sharing and exchanges with Yongchao, Guoxin, Yiming, Jiayu. Thanks to my old pals: Yang Du, Rui Gao, DongAo, Junyu, Guoyang, Han Gao, Hao Qin, Xing Zhou, Yiyan, Kailai, who possess qualities I forever admire. I am lucky to have met you all and to share the bits and pieces of life to this day.

Thanks to Jiejie, the best dog in the world.

Thanks to my family for their unconditional support in my life and career, allowing me to smoothly complete my Ph.D. studies. I am grateful to my grandmother for her meticulous care and unconditional love from childhood. My family's support has been pivotal in reaching this stage of my life.

Lastly, to my best-friend-forever, Xuanjin. You have the world's best eyes for discovering beauty, and I hope you will always find the beauty in the mundane daily life. May you find beauty and happiness in every trivial matter because they make up our life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Automotive Attack Surfaces

Recent advancements in automotive technology have led to more complex vehicles, both in terms of vulnerability to external attacks and the intricacy of their internal networks, known as the in-vehicle network (IVN). As manufacturers integrate increasingly sophisticated functionalities into vehicles, such as remote controls and Over-The-Air (OTA) updates, the potential for cyber-attacks has expanded, surpassing that of earlier models with less connectivity [68, 21]. Furthermore, the IVN itself is becoming more complex. The quantity of Electronic Control Units (ECUs) has surged to accommodate new features, such as Advanced Driver-Assistance Systems (ADAS), while the IVN's structure has evolved to support more efficient data exchange within the vehicle, including newer architectures like the gateway-segmented or zonal designs [66, 53, 7]. These developments mean that contemporary vehicles bear little resemblance to those in past studies [68, 21], presenting ongoing challenges in securing them against cyber threats.

Facing these challenges, regulatory agencies have implemented various standards and guidelines, such as WP29 R155e [140] and ISO 21434 [58], aiming to set a cybersecurity baseline for the automotive sector. These initiatives seek to establish a regulatory framework that ensures the security and safety of automotive technologies. Yet, it is uncertain if these regulatory measures provide a robust enough foundation to counteract the dynamic cybersecurity risks confronting today's vehicles.

1.2 Attacks on ADS Perception

Autonomous vehicles (AVs) have seen significant advancements, relying on a variety of sensors and machine learning algorithms to perceive and interpret their surroundings, thereby performing numerous tasks autonomously. Lane detection is crucial among these tasks, as its accuracy directly influences the vehicle's steering decisions. Consequently, compromising the lane detection system can have dire repercussions. For instance, if adversaries manage to deceive the system into recognizing false road markings as legitimate lanes, the vehicle could be directed into opposing traffic, as illustrated in Fig.1.1. Although recent research has shown that it's possible to manipulate the camera-based perception systems of autonomous vehicles [155, 112, 87], these studies face significant limitations. Firstly, many rely on white-box analysis, presupposing complete access to the vehicle's perception model [155, 112], a challenging scenario with real-world vehicles. Secondly, experiments conducted directly on actual vehicles are scarce. To date, only Nassi et al. have successfully demonstrated a phantom attack on a Tesla's camera-based system [87], which, however, is only effective in low-light conditions and could

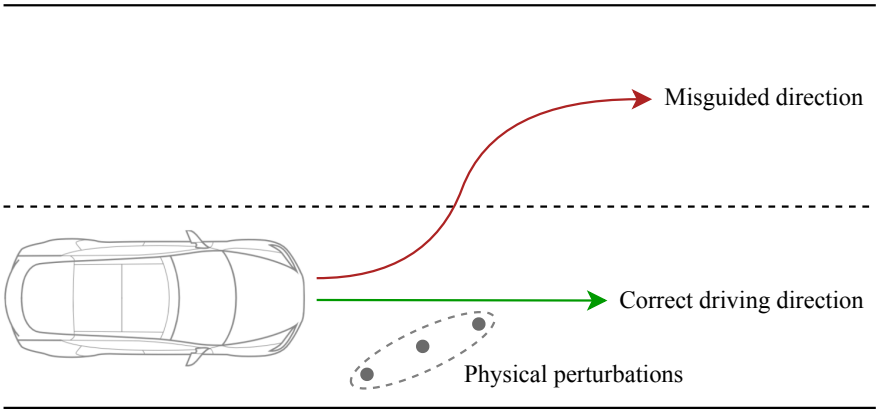


Figure 1.1: Malicious pertubations can mislead the autonomous vehicle into the reverse traffic lane.

be easily detected by alert drivers.

1.3 Testing on ADS Safety

The field of autonomous driving has seen significant advancements, with increasing research focus on its safety aspects. A significant portion of this research targets the *perception module*, exploring how external disturbances can mislead sensors like cameras [62, 88, 111, 18, 49, 13] and LiDAR [18, 49, 17, 157]. Additionally, there is considerable interest in the planning module, focusing on identifying scenarios that may cause the autonomous vehicle to make incorrect planning decisions, such as initiating a collision with a non-player character (NPC) vehicle [73, 67, 131, 133, 143, 156, 54, 116]. Yet, the control module, vital for the vehicle's overall operation and safety, has received less attention in research.

In leading autonomous driving system (ADS) designs [2, 8, 95], the *control module* follows the *planning module*, which devises a *planned trajectory* for the vehicle's short-term path. The control module then calculates the necessary control

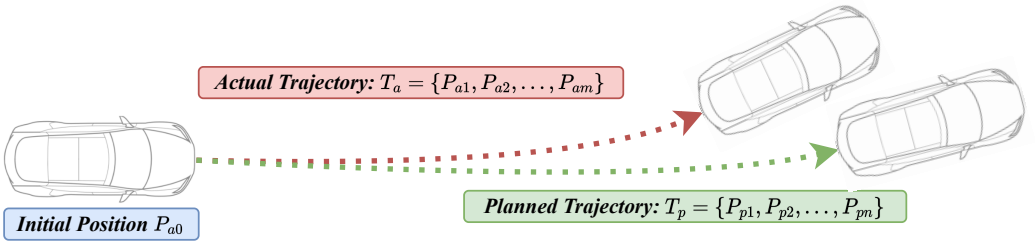


Figure 1.2: **Planning-control inconsistency:** Actual trajectory can be deviated from planned trajectory due to the imperfection of the *control module*.

signals, including the acceleration and steering adjustments needed to follow this trajectory. Previous scenario-based testing approaches [73, 131, 133, 143, 156, 54] often operate under the assumption of a perfect control module, which flawlessly executes the planned movements. This assumption, however, overlooks the practical limitations and imperfections inherent in the control module, including the balance between precision and operational smoothness. As depicted in Fig.1.2, the planning module might suggest a trajectory T_p, but due to potential control inaccuracies, such as over-steering, the vehicle's actual path T_a might deviate from T_p. This gap and the complexities surrounding the control module have not been fully addressed in existing research.

1.4 Our Work

1.4.1 Outline

This thesis is composed of three works, collectively enhancing the security of connected vehicles (Work 1) and the safety of ADS (Works 2 and 3), as shown in Fig.1.3. Specifically, Work 1 aims to enhance the security of connected vehicles. We first conducted a large-scale interview study with experts working in

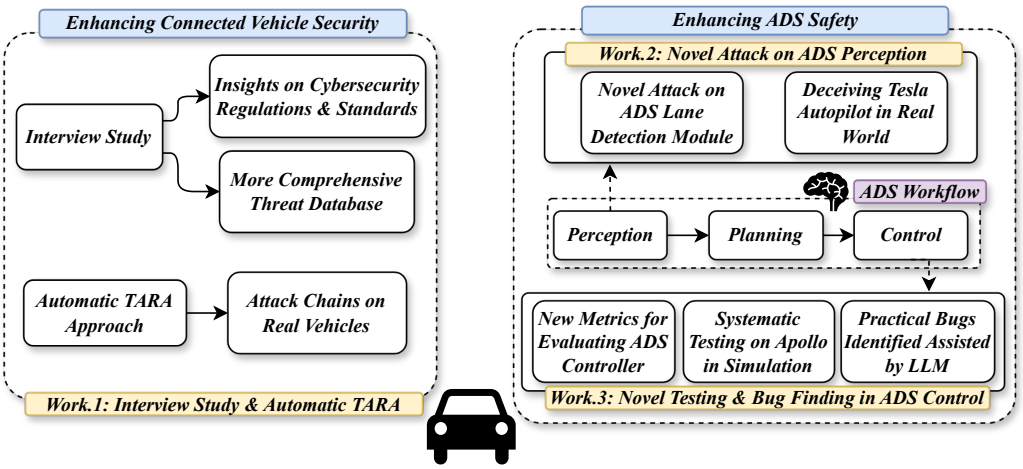


Figure 1.3: **Thesis Outline:** This thesis is composed of three works, collectively enhancing the security of connected vehicles (Work 1) and the safety of ADS (Works 2 and 3).

automotive cybersecurity from the industry, gathering insights on how to improve current regulations and standards, as well as constructing a more comprehensive threat database with data collected from the interviews. As for Works 2 and 3, they focus on enhancing the safety of the ADS system. In Work 2, with a focus on the perception module, we present a novel attack approach to deceive the lane detection module in the perception of the ADS workflow and demonstrate the effectiveness of our attack on the Tesla Autopilot system. In Work 3, we focus on the control module of the ADS. We present new metrics for quantitatively evaluating the performance of the ADS controller, conduct systematic testing on the state-of-the-art open-source ADS - Apollo, and identify practical and new bugs in its codebase with the help of Large Language Models (LLMs).

1.4.2 Work.1: Improving Regulations and Automotive TARA

We conducted a comprehensive semi-structured interview study involving 15 experts in automotive cybersecurity, aiming to understand their views on cybersecurity regulations. From these interviews, we derived 20 significant insights, covering the challenges within the automotive security sector and the shortcomings and suggestions for current regulations. Our analysis led to the identification of two primary limitations. First, we observed that the threat scenarios outlined by existing regulations are too narrow and fail to provide adequate guidance. Second, we noted that the current standard [58] only offers a high-level framework for Threat Analysis and Risk Assessment (TARA), with the practical application of TARA in the field being hampered by the absence of automated tools. Based on these insights, we embarked on the following two contributions to address the identified deficiencies:

An enhanced automotive threat database. To overcome the deficiencies of the current threat database, we developed a more comprehensive and detailed automotive threat database based on our interview findings. This database is structured hierarchically, consisting of 7 themes, 28 codes, and 119 specific threat descriptions. Additionally, we explored the connections between these threats and introduced a Knowledge Graph (KG) for a more integrated database representation.

An automated tool for TARA. To improve the efficiency of TARA, we developed CarVal, a pioneering Datalog-based system for automating the analysis of attack paths in IVNs and the calculation of risk levels. CarVal efficiently identifies multi-stage attack paths in complex IVNs and produces logical paths for further analysis, such as security assessments. Applying CarVal to five actual vehicles,

we mapped out realistic attack scenarios. Following these scenarios, we conducted in-depth security evaluations on five vehicles, exploiting various vulnerabilities within the gateway-segmented IVN architecture. Notably, we discovered *new* potential attack vectors, including vulnerabilities in the In-Vehicle Infotainment (IVI) browser, the official mobile application, the backend server, and IVI-based malware.

1.4.3 Work.2: Attacking Autonomous Driving Perception Module

We embark on the *first* study examining the safety of the lane detection modules in actual vehicles. Using the Tesla Autopilot system [125] as a case study, we demonstrate that it is indeed possible to deceive the lane detection module with strategically placed physical disruptions, leading to dangerous outcomes like collisions with curbs or veering into oncoming traffic. Interestingly, the susceptibility of the system stems not from a deficiency in its deep learning algorithm for lane detection but from its over-sensitivity, where even subtle stickers on the road can be misinterpreted as valid lanes, thus misleading the vehicle. Investigating the lane detection module within a real vehicle presents several challenges. Firstly, due to the proprietary nature of the vehicle's systems, gaining access to and understanding the operation of the lane detection system, especially the deep learning algorithms running on the GPU, is difficult. Secondly, identifying the most effective disturbances that can fool the lane detection system without attracting the driver's attention is a complex task. Thirdly, devising a practical method to implement these disturbances in the real world, such as adding inconspicuous road markings,

poses its own set of challenges. A straightforward method of trial-and-error with ground stickers to misguide the vehicle is highly laborious and prone to mistakes.

We introduce a novel two-phase method to autonomously identify the road markings required to compromise the lane detection module. Initially, we reverse-engineer the firmware of Tesla Autopilot to understand the inputs and outputs of its lane detection module, specifically the camera images and the processed lane images. Armed with this information, we employ black-box attacks against the lane detection module by applying crafted disturbances to the camera image and observing the manipulated lane image. We develop metrics to assess the disturbance's visibility and the detectability of the manipulated lane, setting up an optimization challenge to discover the most effective yet inconspicuous disturbance. Utilizing 5 heuristic algorithms, we determine Particle Swarm Optimization (PSO) as the most effective strategy. In the subsequent phase, we apply the identified optimal disturbances as physical markings and test their impact. Notably, by using physical parameters to describe the digital disturbances, we can easily translate these optimal disturbances into real-world markings. Our comprehensive testing on a Tesla Model S proves that the lane detection module can indeed be fooled by these subtle disturbances, leading to misguidance of the vehicle in auto-steer mode.

1.4.4 Work.3: Identifying ADS Controller Involved Bugs

Evaluating control modules in autonomous driving systems poses significant challenges for two main reasons. First, the absence of definitive benchmarks or metrics complicates the evaluation of control module quality. This is particularly true for

ADS controllers, where multifaceted metrics such as smoothness, often not a concern in other Cyber-Physical System controllers, must be considered [51]. Second, existing scenario-based testing primarily targets the planning module, leading to a scarcity of effective methods for creating control module-specific test scenarios. To overcome these obstacles, we introduce 4 novel metrics tailored for evaluating autonomous driving control systems. Beyond comparing the planned and actual trajectories (i.e., the error between T_p and T_a), our metrics also assess responsiveness, stability, and smoothness, offering a comprehensive set of metrics for control module evaluation. Furthermore, we incorporate these metrics into the existing scenario-based fuzzing framework [67], enabling it to efficiently generate scenarios that challenge the control module's performance.

Leveraging our newly developed metrics and the improved fuzzing method, we perform the first detailed evaluation of the control module in the industrial-level ADS, Apollo. We create two scenario categories: *basic scenarios* to test the control module's fundamental capabilities, and *critical scenarios* to assess its performance under potentially hazardous conditions. These scenarios are then executed in a co-simulation environment, and the control module's performance is quantitatively assessed using our proposed metrics. Unexpectedly, our analysis revealed considerable performance flaws in Apollo's control system, with the system failing to complete basic tasks, such as executing a full turn.

Based on the proposed metrics, we have pointed out *how* the controller is bad, while it is yet unknown *why* so. However, identifying such correspondence between *how* and *why* is challenging due to the complexity of the bug behavior and the control code logic. To identify the specific bugs in the controller code (i.e., answering why), we proposed a semi-automatic bug analysis approach assisted by

the Vision Language Model (VLM) integrated in a Chain-of-Thought reasoning process. Compared with traditional LLMs that could only take text input, VLM can take the visual and text input at the same time, and output content based on both the visual and text input. In this case, VLM becomes a better solution as we can construct both the *bug behavior* and the *controller code* into visual representations that VLM can effectively comprehend. Specifically, in the proposed CoT process, the VLM will first comprehend the bug behavior (from our previous metric-based testing) and the control code logic (from the source code), via the effective and easy-to-understand visual input, and then reason the specific bugs in the source code. Assisted by a subsequent reliable dynamic analysis, we determined 14 previously undiscovered bugs responsible for the controller inadequacies. All identified bugs were acknowledged and promptly addressed by the official team with our assistance. After we fixed most of these bugs, we re-evaluated the controller and found that the controller can follow the planned trajectory more smoothly and accurately, validating our bug findings. All discovered bugs were reported to the official team and were promptly addressed [120].

1.5 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 presents the literature review. Chapter 3 presents our work on revisiting automotive attack surfaces, including the interview study, the automatic TARA tool CarVal, and the experimental analysis on real vehicles. Chapter 4 presents our novel attack on the ADS perception system. Chapter 5 presents our systematic testing on the ADS control system. Chapter 6 concludes this thesis.

The primary research outputs emerged from the thesis are as follows:

- Pengfei, Jing, Zhiqiang Cai, Yingjie Cao, Le Yu, Yuefeng Du, Wenkai Zhang, Chenxiong Qian, Xiapu Luo, Sen Nie, and Shi Wu (2023). "Revisiting Automotive Attack Surfaces: a Practitioners' Perspective". In: 2024 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, pp. 80–80.

- Pengfei, Jing, Qiyi Tang, Yuefeng Du, Lei Xue, Xiapu Luo, Ting Wang, Sen Nie, and Shi Wu (2021). "Too good to be safe: Tricking lane detection in autonomous driving with crafted perturbations". In: 30th USENIX Security Symposium (USENIX Security 21), pp. 3237–3254.

- Pengfei Jing, Xiapu Luo, Sen Nie, and Shi Wu. "Expectations aren't Guarantees: Identifying the Inconsistency Between Planning and Control in Autonomous Driving". *(Under review of ICSE 2025)*

Chapter 2

Literature Review

2.1 Revisiting Automotive Attack Surfaces

Research studies related to the cybersecurity of modern vehicles are prospering these years, and there are a series of related surveys as the milestones [82, 106, 102, 55, 107, 32, 35]. After going through the surveys and investigating the related works, we compare our research with previously discovered attack surfaces in Tab.2.1. Note that there are also many studies focusing on attacking the sensors to affect the behaviors of the autonomous driving systems [61, 17, 118, 89, 110, 18]. As shown in Table.2.1, various attack surfaces have been explored in previous research. However, they suffer from the following limitations. First, they failed to consider emerging attack surfaces due to automotive user interfaces (e.g., mobile app, in-vehicle browser, server and IVI malware as we exploited). In addition. *none* of them have taken into account the in-vehicle network (IVN) topology, leading to two drawbacks: 1). the old attack may not function on the new IVN (e.g., when there is a gateway protection), and 2). potential attack paths

could be neglected, due to the lack of a comprehensive understanding of the IVN topology.

Table 2.1: Comparison with previously discovered cyberattacks on modern vehicles. - *Attack capability*: ○: Affect trivial functions; ◐: Perform limited car controls; ●: Perform safety-critical car controls. - *Real car?*: ○: Simulation; ◐: Testbed; ●: Real cars.

Ref	Attack Surface	Attack capability	Real car?	Bypass gateway?
[52]	OBD-II	●	●	✗
[68]	OBD-II	●	●	✗
[83]	OBD-II	●	●	✗
[24]	OBD-II	◐	●	✗
[114]	OBD-II	◐	●	✗
[21]	CD, Bluetooth, Cellular	●	●	✗
[84]	USB, Wi-Fi, Cellular	●	●	✗
[108]	TPMS	◐	●	✗
[14, 42, 141, 142, 122]	Immobilizer	◐	●	✗
[1, 44]	PKES	◐	●	✗
[19, 152]	Speech recognition system	○	●	✗
[80]	Mobile APP (OBD-II dongle)	◐	◐	✗
[150]	Mobile APP (OBD-II dongle)	◐	●	✗
[149]	Mobile APP (OBD-II dongle)	◐	○	✗
[64]	Telematics	◐	●	✗
[41]	OBD-II dongle	●	○	✗
[69]	A compromised ECU	◐	●	✗
Ours	Mobile APP (Official), IVI browser, backend server, IVI Malware	●	●	✓

2.2 Attacking ADS Perceptions

Adversarial Attacks. Deep neural networks (DNNs) have demonstrated remarkable performance across various domains. Nevertheless, research has revealed that these models exhibit vulnerability when confronted with carefully crafted inputs [121, 103, 75, 154]. Such malicious inputs can induce incorrect decisions while remaining imperceptible to human observers. Additionally, models may be susceptible to other forms of attacks, such as poisoning and backdoor attacks [45, 60, 104].

Lane Detection. Lane detection is a critical task in the environmental perception of autonomous vehicles, as it provides positional information and ensures vehicles remain within lane boundaries. Traditional lane detection methods rely on selected features to identify lane markings [15, 65, 123], with performance heavily dependent on these features. In recent years, DNNs have been widely adopted for lane detection due to their potent feature extraction capabilities [56, 72, 76, 90].

Emerging Attacks on Perception. Numerous autonomous driving systems now employ DNNs to process data, particularly vision data [125, 138, 92, 12]. These vision-based models utilize camera data as input and produce steering angles as output. While these models generally perform well, they can still make erroneous decisions in certain instances, which can have severe repercussions [128, 137]. Eykholt et al. demonstrated the ability to misclassify a stop sign using a physical adversarial example on a DNN model [39]. Although both our method and that in [39] are "two-stage," they serve different purposes. The "two-stage" approach in [39] is for evaluating an attack post-deployment, whereas our "two-stage" method is for executing the attack. Zhou et al. introduced DeepBillboard, a method to generate physical adversarial examples that cause DNN-based autonomous driving systems to steer incorrectly [155]. Shen et al.[115] employed GPS spoofing to misdirect vehicles. Ben Nassi et al.[87] utilized projection to deceive vehicles into perceiving a projection as a genuine object (phantom attack), and they also tested the lane detection module of Tesla Autopilot. However, the phantom attack is limited to nighttime conditions and is easily detectable. In contrast, our attack can be deployed during daylight hours and is more covert.

2.3 Testing ADS Safety

ADS Workflow: Planning to Control. Existing Autonomous Driving System (ADS) architecture can be divided into two main design philosophies: end-to-end design and modular design [48, 22]. In the modular ADS architecture (e.g., Apollo [2], Openpilot [95], and Autoware [8]), the *planning* and *control* modules work in close collaboration to ensure smooth and safe vehicle operation. The planning module is responsible for generating a future trajectory that the vehicle should follow, taking into account various factors like road conditions, obstacles, and traffic rules. Once this trajectory is planned, the control module will calculate the optimal control commands needed to adhere to this trajectory, using control algorithms including Proportional-Integral-Derivative (PID) controllers or Model Predictive Control (MPC). While much of the previous research efforts [144, 67, 74, 79, 132] put into optimizing the quality of the planned trajectory, it is equally crucial to ensure that the control module is capable of accurately following this path.

Scenario-based Testing. There is a series of related works focus on exploring the scenarios that will make the autonomous driving system go wrong [73, 67, 131, 133, 143, 156, 54, 116], and they are summarized in Tab.2.2. These previous works share a similar processing of using *fuzzing* to find the violations. Specifically, starting from randomly generated scenarios, the system will mutate them from various settings (e.g., the trajectory of NPC vehicle), and calculate the fitness score of the mutated scenarios (e.g, the distance to collision), then select the scenarios with higher fitness scores for the next-round mutation. However, almost all previous works failed to include the *control module* (i.e., assuming perfect lane

Table 2.2: Summary of previous works on discovering safety violations on ADS.

Citation	Tested ADS	Simulator	Approach	Results	Control Module Involved?
AV-Fuzzer [73]	Apollo 3.5	LGSVL	Fuzzing based on feedback of distance to collision	5 types of safety violations in Apollo	✗
DriveFuzz [67]	Autoware	Carla	Fuzzing based on feedback of driving quality	33 bugs in Autoware and Carla	✓
CRISCO [131]	Apollo 6.0	LGSVL	Generate scenarios by mining the *influential patterns* and increasing *criticality*	13 types of safety violations in Apollo	✗
MOSAT [133]	Apollo 6.0	LGSVL	Fuzzing based on multi-objective metrics	11 types of safety violations	✗
PlanFuzz [143]	Apollo 3.0, 5.0, Autoware	LGSVL	Fuzzing based on Planning Invariants (PI)	9 DoS vulnerability which stops the vehicle from moving	✗
AVUnit [156]	Apollo 6.0	LGSVL	Failure-Coverage Fuzzing with customized scenarios description	19 planning bugs in Apollo	✗
DoppelTest [54]	Apollo 7.0	None	Fuzzing with Generating multiple autonomous vehicles in traffic	8 bug types in Apollo	✗
Acero [116]	Openpilot and Autoware	Carla	Search the adversarial trajectory to interrupt the victim vehicle	6 attack cases to change the trajectory of victim ADS	✓
Ours	**Apollo 8.0**	**Carla**	**Fuzzing based on feedback of driving quality & *planning-to-control inconsistency***	**14 new bugs in Apollo code (Tab.5.5)**	✓

following without evaluating the performance of the controller), while controller is the key module to ensure the vehicle can follow the planned trajectory. Particularly, previous works only evaluate the correctness of the planned trajectory [73, 131, 133, 143, 156, 54, 116], without considering whether the subsequent control module can follow the trajectory. DriveFuzz [67] and Acero [116] are the only two works that involved the control module (i.e., sending the throttle and steering command to the vehicle instead of simply teleporting vehicles to the planned points). However, they still failed to evaluate the inconsistency between the planning and control module.

Chapter 3

Revisiting Automotive Attack Surfaces

3.1 Overview

As modern vehicles become increasingly complex in terms of both external attack surfaces and internal in-vehicle network (IVN) topology, ensuring their cybersecurity remains a challenge. Existing standards and regulations, such as WP29 R155e and ISO 21434, attempt to establish a baseline for automotive cybersecurity, but their sufficiency in addressing the evolving threats is unclear. To fill in this gap, we first carried out an in-depth interview study with 15 experts in automotive cybersecurity, uncovering the particular challenges encountered during security activities and the limitations of current regulations. We identified 20 key insights from the interview data, ranging from the challenges and gaps in the existing automotive security industry to the limitations and recommendations for current regulations. Notably, we discovered that the quality of threat cases provided by existing reg-

ulations is unsatisfactory, and the Threat Analysis and Risk Assessment (TARA) process is often highly inefficient due to the lack of automatic tools. In response to the above limitations, we first built an improved threat database for automotive systems using the collected interview data, which enhanced the existing database both quantitatively and qualitatively. Additionally, we present CarVal, a datalog-based approach designed to infer multi-stage attack paths in IVNs and calculate risk values, thereby making TARA more efficient for automotive systems. By applying CarVal to five real vehicles, we performed extensive security analysis based on the generated attack paths and successfully exploited the corresponding attack chains in the newly gateway-segmented IVN, uncovering new automotive attack surfaces that previous research failed to cover, including the in-vehicle browser, official mobile app, backend server, and in-vehicle malware.

In summary, aiming to secure the connected vehicles from the cybersecurity perspective, we make the following contributions:

- An in-depth interview study with 15 automotive security experts, identifying 20 key points ranging from challenges in conducting security activities to specific limitations of existing regulations.
- An improved threat database for automotive cybersecurity, developed using the data collected from the interviews, which enhances the existing database both qualitatively and quantitatively.
- The design and development of CarVal, a novel Datalog-based approach to infer attack paths and assess corresponding risk values in modern IVNs. CarVal is capable of inferring multi-stage attacks and prioritizing attack paths based on the calculated risk values.

- Extensive security analysis on five real cars based on attack paths discovered by CarVal, which led to the identification of new attack chains that previous works failed to cover, from new attack surfaces to the ECUs behind the gateway.

3.2 Background

Threat Analysis and Risk Assessment for Automotive Systems. The increasing computerization and complexity of modern vehicles have led to the emergence of new attack surfaces and corresponding cyberattacks [68, 150, 21, 108]. This necessitates conducting TARA on contemporary vehicles to identify potential threats, vulnerabilities, and associated risks within the system. By comprehending these risks, vehicle manufacturers can implement appropriate mitigation strategies. However, security assessment guidelines provided by current regulations, such as WP29 R155e [140] and ISO 21434 [58], exhibit limitations in delivering comprehensive security assessments. These guidelines are often too generic and fail to provide specific guidance on addressing security risks related to a particular system [26]. Furthermore, existing regulations [140, 23] only enumerate discrete threats that manufacturers should consider, leaving an efficiency gap in automated risk assessment for modern vehicles. Automotive ISAC's Risk Assessment and Management [6] provides best practices for risk assessment and management in the automotive industry, offering a comprehensive approach to identifying and mitigating risks. Additionally, risk assessment for autonomous driving has been explored by Derrick Dominic et al. [36], who discuss the specific challenges and methodologies for assessing risks in cooperative automated driving systems. Furthermore, the National Highway Traffic Safety Administra-

tion (NHTSA) has published guidelines on the cybersecurity of firmware updates [91], which include insights from interviews with experts in other relevant industries to understand and mitigate security risks associated with Over-The-Air (OTA) updates.

Regulations on Automotive Cybersecurity. The growing number of automotive cyberattacks in recent years underscores the urgent need for standards and regulations that enforce automotive cybersecurity. The United Nations Economic Commission for Europe (UNECE) introduced WP29 R155e [140] as a compulsory regulation that Original Equipment Manufacturers (OEMs) and Tier suppliers in UNECE countries must adhere to. This regulation mandates OEMs to establish a CyberSecurity Management System (CSMS) for managing security risks throughout a vehicle's lifecycle. Although R155e enumerates potential automotive cyberattacks and corresponding defenses as references for CSMS, it does not offer specific guidance on configuring a CSMS to meet the requirements. The International Organization for Standardization (ISO) proposed ISO 21434 [58] as a non-mandatory standard that supplies general guidelines for managing security risks across the automotive lifecycle. Contrasting WP29 R155e, which is obligatory, ISO 21434 provides suggestions on how to construct a CSMS. GB/T 40861-2021 [23], published in China, is a standard that stipulates general requirements for ensuring automotive security. This standard outlines cybersecurity threats faced by modern vehicles across six dimensions, encompassing software and hardware systems, in-vehicle and long-distance communication, and in-vehicle data.

3.3 Interview Methodology

3.3.1 Study Setup

We present the methodology of our interview in this section, including the design of the interview protocol, the recruitment, the interview procedure, the data analysis process, and the detailed interview structure.

Design of the Interview Protocol. The preliminary interview protocol was developed in accordance with the three exploratory motivations: 1) identifying challenges and gaps in the implementation of security activities within the industry; 2) evaluating the effectiveness and relevance of current regulations in addressing specific threats; and 3) exploring the limitations and providing recommendations for enhancing existing regulations. In particular, a qualitative analysis of current regulations was conducted to: 1) establish metrics for assessing existing threats, and 2) create an initial threat database by integrating knowledge from multiple regulations. Specifically, we first collected the threat descriptions from current regulations [140, 58, 23], and two authors performed iterative coding on them to derive (a). a list of initial threats that are expected to be expanded during the interview process, and (b). the evaluation criteria on assessing these threats. This qualitative analysis contributes to the design of the interview protocol. This qualitative analysis extracted 38 threats distributed in 6 codes, and our interview study finally expanded this database to 119 threats in 28 codes. The protocol can be accessed in [20]. After 10 rounds of interviews, the protocol was finalized and remained consistent for all subsequent interviews.

Recruitment. We invited experts working in the field of automotive cybersecurity from both first-party automotive manufacturers and third-party suppliers

to participate in the interview (two employer companies play the role as 1st-party OEM and 3rd-party provider at the same time). The information of the 15 interviewees is presented in Table 3.1. On average, they had about 6 years of experience in the security field, and there are senior experts with experience over 10 years (P1 and P14). 8 out of 15 are from 1st-party OEMs, and 9 are from 3rd-party suppliers, with two overlaps. Their roles included TARA, security testing, project management, and regulation study, ensuring that all participants were experienced experts from diverse companies who could provide convincing opinions in the field. Particularly, the 1st party manufacturers include companies from multiple countries (e.g., China and Germany), and the 3rd party suppliers also offer security services (e.g, security testing and security consulting) for automotive companies from all over the world (e.g., including car brands from Germany, Japan, America, China, and others). After the 15th round of interview, we identified a saturation of new opinions and the threat cases that the experts can offer, and stopped recruiting more participants.

3.3.2 Procedure and Data Analysis

Interview Procedure. The interviews were conducted through online meetings. During the interviews, the interviewer (i.e., author of this paper) shared the screen to display the interview protocol and related materials (e.g., content of the regulations under discussion) to the interviewees. Both audio and video of the interviewer's screen were recorded for further analysis. We began the interview by collecting basic information about the interviewees, and then proceeded to discuss the specific topics in the protocol. The interviews were conducted in a semi-structured

Table 3.1: Interviewee demographics

ID	Sex	Exp[1]	Company [2]	Position[3]	Duration
P1	M	10	C1: 1st Party	TARA	2:52:47
P2	M	3	C2: 3rd Party	TARA, Manag	1:18:16
P3	M	5	C3: 1st Party	TARA, Manag, Reg	1:05:21
P4	M	4	C4: 3rd Party	Test	0:59:55
P5	M	3	C4: 3rd Party	Test	0:43:31
P6	M	3	C5: 3rd Party	Test	1:05:36
P7	M	3	C4: 3rd Party	Test, TARA	0:55:44
P8	M	7	C6: 1st & 3rd	Test	0:38:50
P9	M	5	C7: 3rd Party	Test, TARA, Manag	0:56:32
P10	F	3	C8: 1st Party	Test, TARA, Manag	0:53:15
P11	M	3	C8: 1st Party	TARA, Mang	1:27:38
P12	M	5	C8: 1st Party	Test, TARA	1:33:18
P13	M	6	C6: 1st & 3rd	TARA, Manag	1:37:45
P14	M	20	C9: 3rd Party	TARA, Manag, Reg	2:11:10
P15	M	3	C3: 1st Party	TARA, Manag	1:05:34

[1] Years of working experience in security;
[2] From 1st party vehicle manufacturer or 3rd party supplier;
[3] TARA: Threat Analysis and Risk Assessment; Manag: Project manager; Reg: Regulation-related study; Test: Security testing.

manner, allowing the interviewees to freely express their thoughts. After the interview study, we derived an automotive threat database with 119 threats under 28 codes, and sent back this database to all participants for suggestions on final modifications. The interview process was started in November 2022 and finished in March 2023.

Data Analysis. We first transcribed the recorded audio to text for further analysis. We then carried out an iterative open-coding process on the collected data [25, 11]. First, an initial codebook was established by all authors based on the interview protocol. Then, two authors separately performed multiple rounds of iterative open coding on all interview data. After that, the two authors verified each other's coding results and resolved the conflicts, and meanwhile updated the codebook. We continued with the iterative coding process until no new code emerged [11]. The final codebook is available in [20].

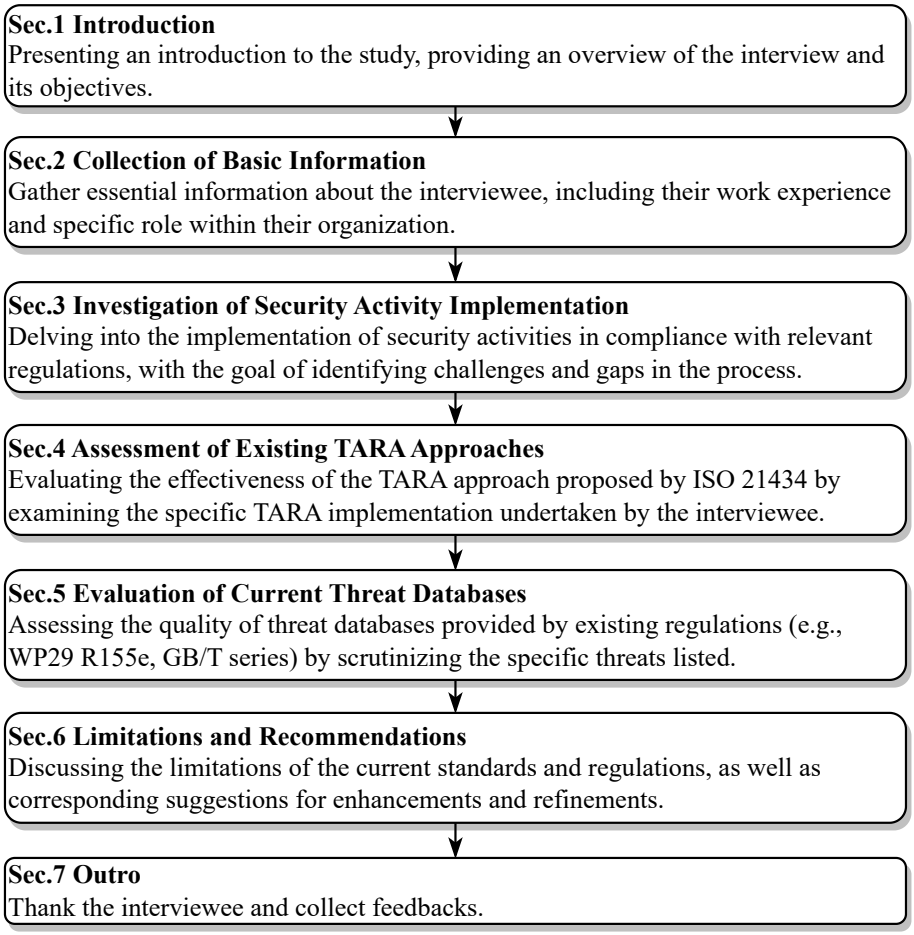


Figure 3.1: The flow of our semi-structured interview. Each section unfolds with particular question, in the meantime interviewees can freely express their thoughts that might discover insights beyond the current section.

3.3.3 Interview Structure

The interview, as depicted in Fig.3.1, starts with an introduction where objectives are outlined and interviewees are encouraged to share personal opinions on security activity implementations and regulations (**Sec.1 Introduction**). We then collect basic information about the interviewees' work experience and role within their organizations (**Sec.2 Collection of Basic Information**). In the next stage, the interview delves into the specifics of how security activities are carried out per

existing regulations (**Sec.3 Investigation of Security Activity Implementation**). Following Sec.3, the effectiveness of TARA approach proposed by ISO 21434 is assessed by scrutinizing its implementation within the interviewee's group (**Sec.4 Assessment of Existing TARA Approaches**). The quality of threat databases provided by existing regulations is then evaluated, alongside the showcasing of an integrated threat database derived from a preliminary study (**Sec.5 Evaluation of Current Threat Databases**). The interview proceeds to discuss the limitations of current standards and regulations and collects suggestions for improvements (**Sec.6 Limitations and Recommendations**), before concluding with an expression of gratitude and feedback collection for improving the interview process (**Sec.7 Outro**).

3.4 Interview Results

In this section, we present our findings based on 15 semi-structured interviews conducted with experts. The structure of this section follows the interview flow presented in Fig. 3.1. Each subsection reports the detailed findings of the corresponding interview section, with specific Key Points (KPs) identified and summarized at the end of each part. Notably, we highlight direct quotes from the interviewees by italicizing them and using quotation marks. Particularly, we mainly report KPs discovered from Sec.4, 5 and 6 in Fig.3.1, and more insights are available in [20].

3.4.1 Assessing TARA

We identified the following key points revealing the weakness of current TARA, which corresponds to the Sec.4 in Fig.3.1.

• **KP.1: Asset identification is difficult.** We identified that the very first step of TARA: asset identification, is a challenge stage due to the often-missing information, and the complexity of the the automotive system. Specifically, as reported by P14: "*The asset identification often costs more than half of the time of the whole TARA process. This is because the materials we rely on are often insufficient to list all assets, and we need to consistently contact the provider for the necessary information and improve the comprehensiveness of the listed assets.*" (P14).

• **KP.2: Lack of objective definitions and criteria.** Another major limitation we identified from ALL interviewees (11) working on TARA is that the current TARA is a high-level methodology, and there is a lack of specific definitions and criteria to ensure the effectiveness and consistency of the TARA results (e.g., when TARA is performed by different groups). E.g., as reported by P12: "*The evaluation of certain criteria in current TARA can vary a lot between different persons or groups, and there is a lack of more specific criteria. For example, we will do a TARA on the specific product, and our suppliers will also do a TARA on it, but the result of our TARA can be very different as the analysis is based on the subjective expertise instead of objective metrics.*" (P12).

• **KP.3: Low level of automation and low efficiency.** We also identified from ALL interviewees working on TARA that the TARA process is often in a low level of automation, and a huge manual effort is still required to finish TARA. E.g., as reported by P11: "*A lot of effort is needed to analyze the attack path in our TARA*

process. Currently, this process still heavily relies on our own experience and expertise. " (P11). Additionally, as reported by P12: "*It is still difficult to craft an automated TARA process because this process is complex and require certain expertise. At least, we currently still rely on our expertise to do the very specific TARA on the products.* " (P12).

Summary on TARA: We identified that currently the TARA applied by practitioners suffers from limitations including requiring heavy manual effort, low efficiency, and the lack of objective definitions and criteria. Although ISO 21434 has presented the high-level TARA methodology, it still remains a challenge on how to conduct TARA efficiently.

3.4.2 Evaluating Threat Database

This section reveals the key points related to the specific threats listed by existing regulations, which corresponds to the Sec.5 in Fig.3.1.

• **KP.4. More common and automotive-specific threats are needed, rather than copying existing threats from other areas.** The automotive system consists of multiple sub-components (e.g., the cloud, the app side, the IoT-related modules). However, the majority of the interviewees (14/15) agree that currently listed threats are largely copied from other domains but not the practical or commonly-seen threats in automotive systems. E.g., as reported by P2: "*Many existing threats are just copied from other areas, rather than describing the truly common threats for automotive systems. I do not think it necessary to detail these already known threats.* " (P2). As also reported by P10: "*We are expecting the regulations to give more common and detailed threats that are really related to current automotives. For example, some manufactures are adding some fancy functions to their prod-*

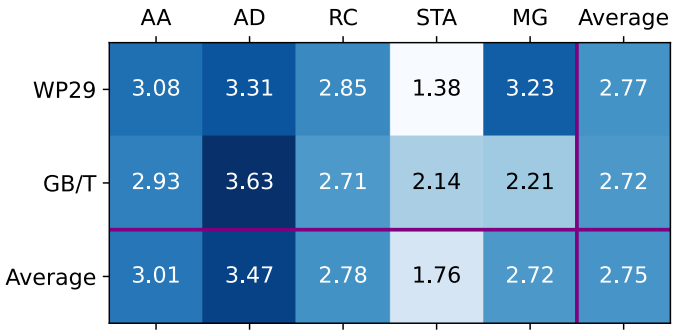


Figure 3.2: Average score from 5 evaluation criteria for WP29 R155e [140] and GB/T [23].

ucts, such as remotely heating the seat. It is OK for the regulations to not mention these unique functions. However, I think it necessary for the regulations to give very detailed guidelines on the very common functions, such as remotely opening the door, which I believe is a function that the majority of vehicles have already applied. " (P10).

• **KP.5: Low scores are given to existing threats by practitioners.** During the interview, we asked the experts to evaluate the quality of threats in current regulations from five metrics, including the Attack Description (AD), the Root Cause (RC) of the threat, the Security Testing Approach (STA) to identify the threat, and the MitiGation (MG). Specifically, they were asked to choose a score from 1 to 5 to present how satisfy they were about the existing threats from the above 5 aspects, and the overall scores are shown in Fig.3.2. Note that the average scores for WP29 R155e and GB/T are 2.77 and 2.72, respectively, representing that experts are overall unsatisfied with the quality of current threats. Moreover, extremely low scores are identified from the aspect of STA.

Summary on threats in regulations: From the practitioners' perspectives, the specific threats listed by existing regulations are far from being satisfying. Particularly, there is a lack of specific threats for automotive systems, and currently listed threats are short of a comprehensive description from various dimensions (Fig.3.2).

3.4.3 Limitations and Recommendations for Existing Regulations

We present the rest key points related to the open-ended discussion of the limitations on current regulations.

• **KP.6: More detailed information would certainly help the security groups.** Multiple previous key points have revealed that the missing of particular details brings challenges to security groups. Particularly, we identified that ALL interviewees agree that a more detailed regulation would certainly help their work, including being more specific on provided threat, giving clear threshold and objective criteria, etc. E.g., as reported by P10: "*Our group mainly relies on our TARA results to express the specific threats to other groups. However, this process would be much more efficient if more details could be found in current regulations.*" (P10).

• **KP.7: Gaps exist between traditional IT threats and automotive threats.** We identified that current regulations failed to give guideline on how to define the severity of the specific threats, especially when the threat exists in the automotive system instead of traditional IT networks, which could result in an incomplete understanding of the threat. This KP also corresponds to the previous KP.4. Particularly, as reported by P10: "*I think there is a significant gap when we switch*

our concepts from the traditional IT threats to the automotive threats, because we are unsure about how to define and analyze the threats when they are connected to the automotive system with the very specific hardware. For example, an engineer with only software security background would find it challenging to precisely define the threat in automotive system. For example, our group would think a vulnerability allowing attacker to remotely open the door is very critical, but other groups would tell me that this would not be a critical case according to the functional safety regulation. I would expect the regulations to give more details on how we should understand the severity of the specific threats. " (P10).

• **KP.8: Non-security groups lack proper security knowledge.** Due to the complexity of modern vehicles, manufacturer companies often consist of a wide variety of groups working together, including the development groups, security groups and others. In our interview, a common challenge identified by ALL interviewees is that non-security groups often lack consensus in security. E.g., as P11 reported: "*Our group knows clearly the meaning of the critical/high/low risks in the TARA results, but other groups do not even know what TARA is.*" (P11). Accordingly, this fact makes it laborious for the security groups and non-security groups to reach a consensus for decisions on particular threats, and it's common that security groups have to show the practical attack results to other groups to present the rationale for security-related requests. E.g., as P11 reported: "*Our development groups actually care so little about security: they totally do not understand why it is necessary to update the system components. For example, the development group thinks that the built-in components in the IVI Android system are safe to use, even when their versions are out-of-date. As a result, we have to craft a practical PoC attack chain to show that the out-of-date codes are vulner-*

able and the significance of system updates." (P11). Overall, as identified by our interview, it is common that the interpretation of the rationale to perform security activities requires a lot of effort, and due to the lack of automatic tools for risk assessment, this process heavily relies on the manual effort and is very inefficient.

• **KP.9: Complex supply chains bring new challenges.** As modern vehicles are becoming increasingly complex in its interfaces and in-vehicle architecture, the corresponding supply chains also get complex and thus bringing new challenges. In particular, as reported by P13: "*We are consistently pushing the security requirements to our supplier, including performing security testing and providing the specific software materials to us. However, it is very common that suppliers are still not attaching enough importance to cybersecurity, and thus they are not able to meet our requirements.*" (P13). Additionally, P14 presented the challenge from the 1st-party OEM: "*It is also very challenging for the 1st-party OEM to ensure the supply chain security: they need to present the very specific cybersecurity requirements to the supplier, and also be capable of reviewing whether the requirement is met. For example, they should give very detailed information about what TLS version and what encryption algorithm should be applied in the specific case, instead of just saying 'follow the best security practice'.*" (P14).

• **KP.10: Conflicts with other groups are common.** As identified from ALL 1st-party interviewees, one fact that they face is that security activities are "costy" and cannot be translated into immediate and direct benefits. As reported by P12: "*We are always compromising with the development groups, and this is inevitable: the design of a strictly secured system requires extra efforts for development group, and often causes a decrease in user experience. As a result, we are always looking for a balance for 'just sufficient' security and reasonable development effort.* "

(P12). Additionally, as reported by P15: "*Overall, implementing fancy features is the top priority for development groups, and security does not directly add attractiveness to the product. As a result, we are always trying to reach the sufficient security design and also try to reduce the workloads of development.* " (P15). This key point is also consistent with KP.8: to reach a common security consensus, security groups tend to make a lot of effort to explain the specific threats.

• **KP.11: Information is not transparently shared between groups.** As indicated by KP.8, current 1st-party companies are consist of many different groups, with different responsibilities and team values, even with possible competitions. Accordingly, another intriguing challenge presented by P13 is that the limited information sharing can affect the security activities: "*Information gathering is the very essential stage for our penetration testing, but the information we can access is often very limited, which could affect the efficiency of our testing. For example, other groups may not pay attention to some malfunctions or bugs, but they could be identified as the critical vulnerabilities in our testing. However, other groups may refuse to offer the explicit details in the first place.*" (P13).

• **KP.12: Security activities get inconsistent between various security groups.** It is common that multiple security groups contribute to the cybersecurity of the same car. For example, when 1st-party manufacturers have assembled the vehicle, they might ask multiple security groups to perform testing on the final product. However, the inconsistent testing output from various groups could cause problems. E.g., as reported by P8: "*It is common that multiple testing groups cannot reach a final decision due to the lack of information sharing. For example, when other groups have identified the security problems we missed, we might not be able to validate them due to the limited information provided. Vice versa, when*

we identified a problem that other groups failed to find, we may not be able to locate the relevant responsible party, or further validate whether the problem is fixed in the final product." (P8).

• **KP.13: Lack of concrete support for rationales behind security-related CRs.** The development of automotive products is often based on Change Requests (CRs). However, it is identified by all 1st-party interviewees that the development groups often think that the security-related CR (e.g., fixing a bug) lacks rationale or concrete supports. E.g., as reported by P13: "*Development groups are often not willing to accept our CRs to fix certain bugs, because they do not think 'the CR solely based on our inner-group testing' is convincing.*" (P13). Moreover, P14 gave more comments about this challenge: "*Currently, it is a fact that the security activities are short of concrete support, especially from the compulsory regulations. It is common that other groups may challenge security requests, and try to 'lower' the security baseline.*" (P14).

• **KP.14: Reactive TARA overwhelmed Proactive TARA.** Another challenge we identified is that the TARA process tends to be reactive instead of proactive (by P10, P11, P12), which may raise concerns. In particular, as reported by P11: "*I think ISO 21434 would want us to frequently perform TARA in a proactive way, to ensure the cybersecurity consistently. However, how we use TARA is more like a reactive way: we only use TARA when specific events happen, for example, when development groups want to remove some security functions, or add some new functions. In this case, we will use TARA to demonstrate the corresponding risk. But in other cases, we would not do TARA very frequently or proactively.*" (P11).

• **KP.15: What companies care the most is how to pass the test.** One fact we identified from ALL 1st-party OEM is that current regulations, especially WP29

R155e and GB/T series that listed specific threats, are not treated as the *gold standard* or *de facto oracle* to ensure cybersecurity. Instead, they are merely the security baseline that companies are trying to meet, with adequate or even minimal effort, and this fact is consistent with the challenges we identified from KP.4 and KP.9. E.g., as reported by P2: "*The specific descriptions for the threats and attacks are just auxiliary content. For us 1st-party OEM, what we care about the most is how to meet the requirement for each listed clause.* " (P2). As also reported by P3: "*Our company have grown quite mature in cybersecurity, and we have already considered all threats listed in R155e. Accordingly, we never expect to rely on this regulation to ensure cybersecurity, and all we care about is how to pass the standard set up by the regulation.* " (P3).

• **KP.16: Companies are unsure of what level of protection is sufficient.** Unfortunately, companies do not know to what extent the protection is sufficient, as none of current regulations has make this requirement clear, which is identified by ALL interviewees from 1st party. As stated in KP.15, companies are always trying to find the *just sufficient* cybersecurity solution with adequate effort, but current regulations are extremely short of this information. E.g., as stated by P2: "*We are unsure about whether our mitigations are sufficient as the current regulations themselves are not clear about that. For example, when protecting the scenario of using digital keys to open doors, does it mean that attacker should not be able to get in the communication channel at all, or it would be sufficient if we can make sure no damage will be done even if the attacker can inject the channel? Currently no regulations are making these details clear.* " (P2).

• **KP.17: Current regulations lack quantifiable criteria for evaluating threat cases.** KP.17 is also identified from ALL interviewees, and is the main reason

leading to KP.16: because no quantifiable criteria is set up, company do not know how to prepare the protection. E.g., as stated by P3: "*We urgently need a very specific and quantifiable criteria, so that we can prepare our cybersecurity solutions accordingly. However, current situation is that, neither we manufacturers nor the certification authority knows how to perform the cybersecurity test.* " (P3). Also as stated by P11: "*We are constantly evaluating our products based on the threats given by WP29 R155e. However, the threats given by R155e are very high-level, and they are often interpreted by the certification authority, and what they say goes. I think it is strange that these specific metrics are explained by the third parties, instead of the regulations themselves, and I think it is one of the most significant weakness.* " (P11).

• **KP.18: Mitigation listed in current regulations is more like remedies rather than high-level solutions.** A majority of interviewees (12/15) agreed that current listed threats seem to focus on discrete remedies instead of high-level cybersecurity solutions. E.g., as reported by P1: "*Currently listed mitigations for specific threats are more like some discrete remedies, rather than some high-level solution that could be considered and applied in the development stage. Although it is challenging to provide detailed and practical high-level solutions, our group is currently working towards this goal and I am expecting such a content in future regulations.* " (P1).

• **KP.19: Clearer guidelines are needed for long-term security management.** We identified that the long-term management of the product cybersecurity is an extremely challenging tasks, due to the insufficient contents of the regulations, and some other difficulties. Particularly, as reported by P14: "*Although ISO 21434 has provided quite detailed guidelines on how to ensure the cybersecurity in the*

development stage, it currently failed to give clear guide on the long-term security management. The long-term management of the automotive cybersecurity is a very challenging task, and all the automotive companies are exploring how to establish a sound long-term risk management system. I hope future regulations will give more insights on this process. " (P14).

• **KP.20: There is a lack of an open platform for sharing threat cases.** Another interesting insight we identified is that the sharing of information, especially the knowledge about specific threats, is often very difficult. This is often due to the very strict examination process to prevent possible leaks of specific threats. Such a examination process is necessary, but would inevitably hinder the communication between different groups. As reported by P13: "*Although there are various ways to access new knowledge, it is common that many details are still missing in the public document, for example, some vulnerability disclosure documents. As a result, we can only derive some general insights rather than technique details, making it hard to actually try the vulnerability in our own. The situation is the same for us: when we identify the threats which are not so common in the moment, it is also difficult for us to communicate with other groups or to output our content to the industry. As a result, it would be very helpful if future regulations could set up a secure and efficient way to share the identified threat.* " (P13).

Summary: The industry is currently facing a series of challenges in implementing security activities. The reasons for this are multifaceted, including the unique nature of security teams (e.g., difficult to directly generate profits), the complexity of modern vehicle architectures, and the inadequacy of current regulations.

3.4.4 Summary on Interview Study

Compared with the previous studies revealing the "business decision" impact of security parties [50, 136, 101], we emphasize the following key points specific to automotive security, which are not covered by the previous works. First, we have identified a series of limitations of the specific regulations on automotive cybersecurity, including the insufficiency of threat database, TARA guidance, security testing approach, and many others. Second, we also identified the challenges in conducting security activities in automotive system, including the lack of automation in TARA process, the lack of quantifiable criteria for risks assessment, conflicts with other groups, and many others. Specifically, these challenges can be attributed to the following two aspects:

Lack of high-quality threat database. The threats offered by existing regulations are insufficient from various dimensions (KP.2, KP.4, and KP.5). Due to the above gap, practitioners have to rely on the experience and expertise of the group to perform security activities, which could be incomplete or inefficient.

Lack of efficient tool for TARA. ISO 21434 [58] presents the high-level methodology of TARA, but the specific implementation is still facing various challenges. Particularly, the lack of criteria can lead to the inconsistent TARA results (KP.2), and the lack of automatic tools can make TARA very inefficient (KP.3).

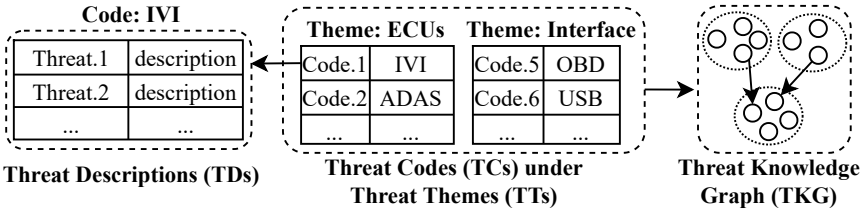


Figure 3.3: The proposed hierarchical framework to describe automotive cybersecurity threats.

3.5 Improved Threat Database

3.5.1 Hierarchical Framework for Automotive Threats

In response to the lack of high-quality automotive threat database, we construct a new threat database that is improved by the collected threats from the interview. Particularly, we use a hierarchical framework to present the automotive-specific threats (in Fig.3.3), in which the involved concepts are explained as follows:

Threat Description (TD). A threat description (TD) is the smallest element in the framework. It is a set of natural language sentences to describe the details of one particular threat, including the specific Attack Description (AD), the Root Cause (RC) of the threat, the Security Testing Approach (STA) to identify the threat, and the MitiGation (MG) to prevent the threat.

Threat Code (TC). A threat code (TC) is a group of TDs under a particular category. Here the word *"code"* comes from the qualitative analysis methodologies [25], in which the process of *coding* is to give labels to the qualitative data (e.g., interview texts). For example, in Fig.3.3, *Code.1 IVI* is the code containing the threat descriptions under the in-vehicle infotainment (IVI) ECU.

Threat Theme (TT). A threat theme (TT) is a group of threat codes following a particular high-level classification logic. For example, in Fig.3.3, the *Threat*

Theme: ECUs includes the threat codes representing the in-vehicle *ECUs* (e.g., IVI, ADAS), while *Threat Theme: Interface* includes threats related to vehicular interfaces (e.g., OBD, USB).

Threat Knowledge Graph (TKG). We derive the concept of knowledge graph (KG) [147, 16, 59] to further represent the relations between the threat codes. Specifically, a knowledge graph can be represented by a set of triplets: *(head entity, relation, tail entity)*, meaning that the *head entity* and the *tail entity* has the particular *relation*. In our scenario, the entities are the threat codes, and the triplet *(TC.1, relation, TC.2)* represents the logical relation between the two codes. For example, the triplet *(Code.1 IVI, vulnerable to threats in , Code.6 USB)* connects the code IVI and code USB because the USB interface is a common interface on IVI.

3.5.2 Detailed Threats

The final result of our threat database is shown in Fig.3.4, with the following specific threat theme and codes:

T1: General Requirements. The various ECUs can share a set of theats that are general to various implementations, and this T1 describes these common threats from five threat codes: *C1.Hardware, C2.Software, C3.RTOS, C4.Complex OS, and C5.Data*. The advantage of setting up this theme is that *we do not need to repeat these common threats in the specific ECU categories*. For example, secure boot is the de facto mitigation that should be deployed on various types of ECUs. There are 24 threat descriptions under T1.

T2: In-Vehicle Components. T2 describes the threats to specific components

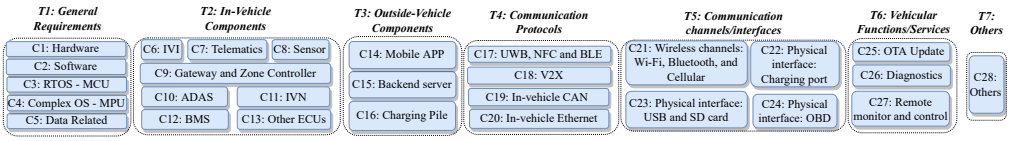


Figure 3.4: An improved hierarchical threat database derived from the interview study, containing 28 threat codes (TCs) under 7 threat themes (TTs). This database serves as an improvement to existing regulations both qualitatively and quantitatively, and is available in [20].

in the vehicle, including the threats on various ECUs and on the In-Vechile Network (IVN). T2 contains the following 8 codes: *C6.IVI, C7.Telematics, C8.Sensor, C9.Gateway and Zone Controller, C10.ADAS, C11.IVN, C12.BMS, and C13.Other ECUs*. These codes focus on the threats that are particular to the function of the ECU. For example, the *C6-10: browser threat*, is the very specific threat that exists in the IVI but not on other ECUs, because the browser module has been widely used in the IVI system to support rich infotainment functions. There are 36 threat descriptions under T2.

T3: Outside-vehicle Components. T3 describes the threats for specific components outside the vehicle, but can communicate with the vehicle and affect automotive cybersecurity. Specifically, T3 contains the following 3 codes: *C14.Mobile APP, C15.Backend Server, C16.Charging Pile*. The vulnerabilities in these external components can pose a threat to the vehicle itself. For example, the private data can be leaked through the charging pile. There are 14 threat descriptions under T3.

T4: Communication Protocols. T4 describes the threats to the communication protocols implemented in the automotive context. Specifically, T4 contains the following 4 codes: *C17.UWB, NFC and BLE, C18.V2X, C19.CAN, and C20.Ethernet*. The unsafe implementation of these protocols can introduce risks. For example, lack of encryption on the data transmitted via the protocol can lead to information leak. There are 16 threat descriptions under T4.

T5: Communication Channels/Interfaces. T5 describes the threats on the communication channels and interfaces on the vehicle. Specifically, T5 contains the following 4 codes: *C21.Wi-Fi, Bluetooth and Cellular, C22.Charging Port, C23.USB and SD card, C24.OBD*. Unsafe implementation of these interfaces leads to threats when these interfaces are exposed to the attacker. For example, the attacker can modify vehicular parameters through the OBD port due to the lack of proper authentication. There are 15 threat descriptions under T5.

T6: Vehicular Functions/Services. T6 describes threats to vehicular function and services, with the following 3 codes: *C25.OTA, C26.Diagnostic, C27.Remote monitor and control*. The implementation of these "trendy" functions can vary for different manufacturers and car models, and can introduce risks when the design is insecure. For example, the unsafe implementation of the secret keys for remote control can be exploited to launch attacks. There are 12 threat descriptions under T6.

T7: Others. T7 includes other threats (e.g., insider attack) that do not fit into other themes. There are 3 threat descriptions under T7. The complete database is available in [20].

3.6 CarVal: Approach

In response to the lack of efficient tool for TARA, we introduce CarVal: the first Datalog-based approach designed to automatically generate attack paths in IVNs, calculate corresponding risk values, and thus make TARA in automotive systems more efficient. We first present the challenges encountered during the design of CarVal in §3.6.1. Then, in §3.6.2, we describe the workflow of CarVal in detail,

including how attack paths are inferred and how risk values are calculated. An example is given in §3.6.3 to provide a clear demonstration of the approach. Finally, we provide the implementation details of CarVal in §3.6.4.

3.6.1 Challenges and Solutions

Challenges. Previous research has proposed datalog-based approaches for automatic attack path generation in enterprise networks (e.g., MulVAL [98, 97, 99]). However, these approaches cannot be directly applied to the automotive domain due to the particular challenges. Firstly, traditional attack path reasoning engine [98, 97, 99] relies on manually crafted reasoning rules in IT networks. However, there are no existing rules that could be applied to reasoning attack paths in IVN. Secondly, unlike enterprise networks, where each node is treated as a host with an identical set of rules, the IVN consists of electronic control units (ECUs) with various hardware and software settings. Unfortunately, previous approaches [98, 97, 99] do not account for these new features in IVN and cannot represent the up-to-date IVN model, and thus it is unclear how to transform the IVN network into Datalog representation. Thirdly, previous works only discussed how to calculate the feasibility (i.e., likelihood) of specific attacks in the attack path [146, 43, 145], and failed to consider the attack impact indicated by ISO 21434 TARA [58], which leads to incomplete output in the specific automotive system.

Solutions. Firstly, we construct the reasoning ruleset and define cybersecurity attacks based on the threat database collected from industry experts through an extensive interview study. Secondly, we introduce a hybrid model combining the *bus model* that represents the broadcasting nature of in-vehicle bus (e.g., CAN bus)

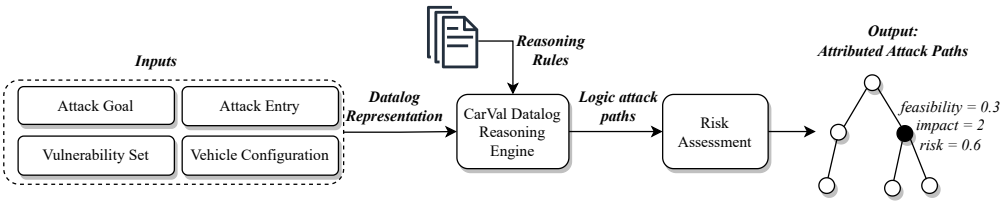


Figure 3.5: CarVal workflow: Automatic attack path reasoning and risk assessment in automotive system.

with the *star model* that represents the up-to-date gateway design (further shown in Fig.3.7). This model can precisely present the IVN model and contribute to correct reasoning of attack paths. Thirdly, we enhance the reasoning engine by calculating the *attack impact* of each node on the path, in addition to the *attack feasibility* indicated by ISO 21434. This is a improvement specifically for the automotive systems.

3.6.2 Workflow

We propose CarVal, an automatic approach for attack path reasoning and risk assessment in IVN, which is shown in Fig.3.5.

3.6.2.1 Input

There are the following four parts of input to the CarVal reasoning engine:

Attack Goal. This component specifies the particular attack that serves as the objective for datalog reasoning. For instance, the following clause represents the attack goal for performing a cross-domain attack on the Body Control Module (BCM) ECU, which involves sending malicious commands like unlocking the doors or opening the car windows:

```
crossDomainAttack(bcm).
```

The attack goal describes a specific attack in the IVN and is derived from a set of primitive nodes, which include the IVN information and possible vulnerabilities. These nodes are referred to as derived Attack Nodes (AN).

Attack Entry. This component describes how the attacker can gain access to the automotive system, which serves as the starting point of the attack path. For instance, the following clause assumes that the attacker can access the In-Vehicle Infotainment (IVI) ECU via the Wi-Fi channel:

```
attackerCanAccess(ivi, wifi).
```

In the attack path, such an attack entry is referred to as the Entry Node (EN).

Vulnerability Set. This component consists of the possible vulnerabilities that can lead to specific attacks. For instance, the following clauses describe the vulnerabilities that exist in two ECUs: IVI and BCM, and these vulnerabilities will serve as the prerequisites to the Attack Node (AN):

```
vulExists(ivi, 'lowPrivCodeExec').
vulExists(ivi, 'unauthorizedBroadcast').
vulExists(bcm, 'lackMessageAuth').
```

Such vulnerabilities are referred to as Vulnerability Nodes (VNs) on the attack path.

Vehicle Configuration. This component includes vehicle-specific information required for attack path reasoning. Specifically, such information comprises the IVN topology and the attributes of the Electronic Control Units (ECUs) and buses in the IVN. For instance, the following clauses indicate that the IVI ECU and Gateway (GTW) ECU are both located on the *infoCAN* bus, and the broadcasting nature of this Controller Area Network (CAN) bus may lead to specific attacks:

```
ecuOnBus(ivi, infoCAN).
ecuOnBus(gtw, infoCAN).
busTypeBroadcast(infoCAN).
```

Such information is referred to as Fact Nodes (FN) on the attack path.

3.6.2.2 Attack Path Reasoning

CarVal initiates datalog reasoning upon receiving the aforementioned inputs to determine the feasible attack path from the attack entry to the attack goal. The effectiveness of this reasoning process depends on the carefully-designed reasoning rules. For instance, the following rule explains how an attacker can enhance the attack impact after executing malicious code in the ECU:

```
attackerBroadcastOnBus(ECU, Bus) :- // AN
    execCode(ECU, Priv), // AN
    ecuOnBus(ECU, Bus), // FN
    busTypeBroadcast(Bus), // FN
    vulExists(ECU, 'unauthorizedBroadcast'). // VN
```

In this reasoning rule, the attacker can broadcast the attack message on a particular bus (e.g., CAN bus) bus after achieving code execution in the particular ECU (e.g., IVI). It is worth noting that this rule is derived from one Vulnerability Node (VN), two Fact Nodes (FN), and one Attack Node (AN).

3.6.2.3 Risk Assessment of Generated Attack Paths

The datalog reasoning module generates a logical attack path, representing how an attacker can achieve the attack goal from the attack entry. Subsequently, CarVal conducts automatic risk assessment of the derived attack nodes along the attack

Table 3.2: Explanations for the symbols used for risk assessment.

Symbol	Range	Explanation
f_{EN}	(0, 1]	How likely the attacker can access this particular attack surface of EN
f_{VN}	(0, 1]	How likely such a vulnerability in VN can exist in automotive system
i_{VN}	>1	How severe the potential impact brought by VN can be
f_{AN}	(0, 1]	How likely the AN can happen when all prerequisite nodes are satisfied
i_{AN}	>1	The intrinsic severity of the AN
F_{AN}	(0, 1]	Cumulative feasibility of this AN on the specific attack path
I_{AN}	>1	Cumulative impact of this AN on the specific attack path
R_{base}	>0	Baseline risk value of the inferred AN
R_{AN}	>1	Cumulative risk of this AN on the specific attack path
$N_1 \rightarrow N_2$	N/A	On the attack path, N_1 is one of the prerequisite to infer N_2

path. As per the ISO 21434 regulation, the risk value of a threat in an automotive system is not solely determined by its feasibility, but also by its impact. Initially, starting from the attack entry, CarVal calculates the attack feasibility and attack impact of all ANs (representing specific threats) along the path. Finally, it evaluates the risk values by taking into account both the feasibility and impact.

Definitions. The risk assessment module uses two metrics: *feasibility* and *impact*. The specific definitions of these metrics are presented in Table 3.2. The intrinsic *feasibility* and *impact* of a particular node are represented by lower case f and *i*, respectively. These values are fixed for all generated attack path. The cumulative metrics F_{AN}, I_{AN}, and R_{AN} represent the *cumulative feasibility*, *cumulative impact*, and *risk value*, respectively, which can vary for different attack nodes in different attack paths.

Attack Feasibility Calculation. The cumulative on-path attack feasibility of an attack node is calculated as the multiplication of the feasibility value in all its premise nodes. Since all feasibility values are within the range (0,1], the cumulative attack feasibility decreases as the attack path becomes deeper. The calculation is expressed in the following equation:

$$F_{AN} = f_{AN} \times \prod_{AN_i \to AN} F_{AN_i} \times \prod_{EN_i \to AN} f_{EN_i} \times \prod_{VN_i \to AN} f_{VN_i} \tag{3.1}$$

Attack Impact Propagation. Similarly, the cumulative on-path attack impact of an AN is the multiplication of the impact value in all its premise nodes. As all impact values are greater than 1, the cumulative attack impact will increase as the attack path gets deeper. This calculation can be expressed using the following equation:

$$I_{AN} = i_{AN} \times \prod_{AN_i \to AN} I_{AN_i} \times \prod_{VN_i \to AN} i_{VN_i} \tag{3.2}$$

Risk Value Calculation. The determination of the risk value associated with a particular threat in automotive systems is based on two factors, namely, its *feasibility* and *impact*. A higher feasibility and impact imply a greater risk value. According to ISO 21434 regulation [58], the final risk value of the node is a baseline value added by the multiplication of the feasibility and impact:

$$R_{AN} = R_{base} + F_{AN} \times I_{AN} \tag{3.3}$$

R_{base} can be assigned with any fixed value (ISO 21434 [58] assigned this value to be 1 for demonstration). After the above calculation, each derived attack node is assigned with the cumulative attack feasibility, impact, and risk value on the specific attack path.

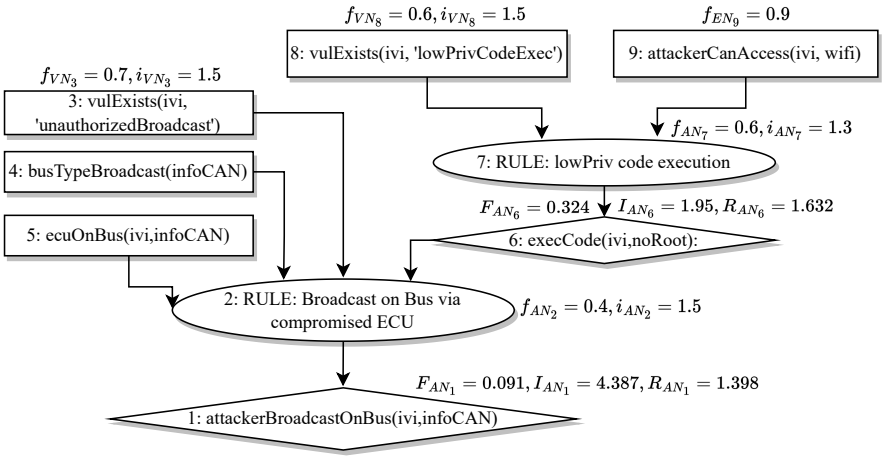


Figure 3.6: Example: an attributed attack path generated by CarVal.

3.6.3 A Demonstrating Example

The output of CarVal is the Attributed Attack Path (AAP), which provides both the logical flow of how an attack can be carried out in the IVN and the quantitative metrics, such as attack feasibility, impact, and risk value, associated with each attack node on the path. Figure 3.6 illustrates an example attack path generated by CarVal, in which the attack goal is Node.1: *attackerBroadcastOnBus(ivi, infoCAN)*, indicating that the attacker can broadcast malicious messages on the *infoCAN* bus in the IVN, by exploiting a compromised *IVI* ECU. Node.2 represents the reasoning rule connecting four prerequisite nodes: Nodes 3, 4, 5, and 6, with Node.6 being the prerequisite attack node (AN), indicating that the attacker needs to first achieve code execution in IVI. Node.3 is a vulnerability node (VN) indicating that a vulnerability exists in IVI that allows the attacker to send crafted messages on the internal bus. Nodes 4 and 5 are fact nodes (FN) representing supplementary conditions, including that the target bus (*infoCAN*) has a broadcast nature (i.e., CAN bus) and IVI is connected to this particular bus. Moreover,

Node.6 *execCode(ivi, noRoot)* is inferred from two additional nodes: Node.8 indicating a vulnerability in IVI that allows the attacker to execute malicious code at a low privilege level, and Node.9 indicating the attack surface – the attack starts from the Wi-Fi channel on IVI.

The risk assessment module automatically computes the cumulative feasibility, impact, and risk value of all derived attack nodes on the path, as calculated by Equations 3.1 to 3.3.The attack node is represented by two separate nodes: the *RULE* node, indicating the intrinsic feasibility and impact (Nodes 2 and 7 in Figure 3.6), and the derived attack node, indicating the cumulative metrics (Nodes 1 and 6). The cumulative metrics of Node.6 are derived from Nodes 7, 8, and 9, while the final cumulative metrics of the attack goal Node.1 are derived from Nodes 2, 3, and 6. Note that Fig.3.6 is only for helping to understand how CarVal works, rather than being comprehensive. More sophisticated attack paths that we exploited on real cars will be detailed in §3.7.

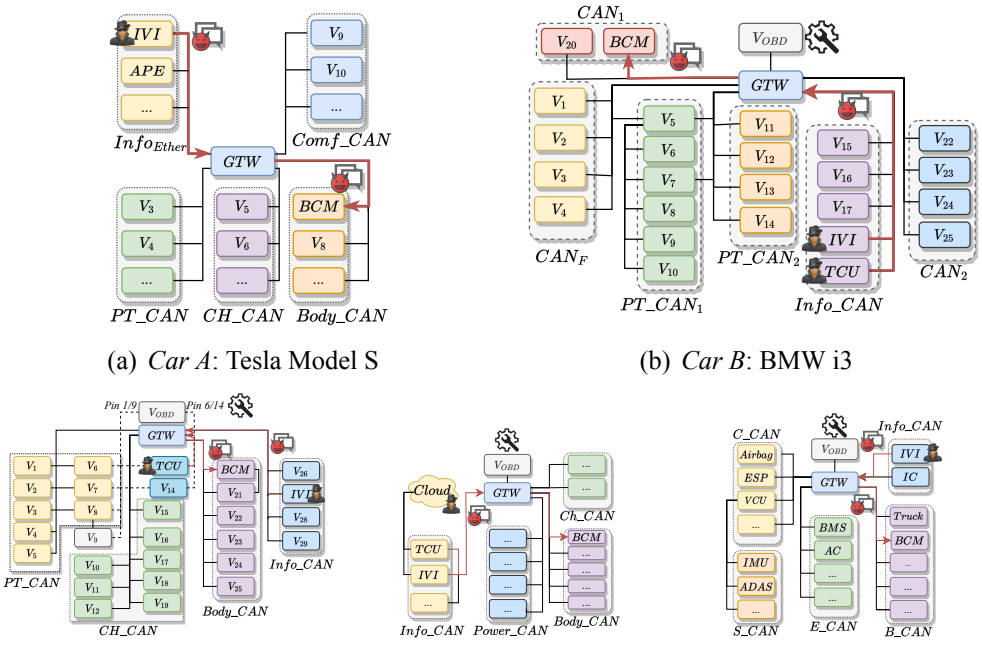
3.6.4 Implementation

The datalog reasoning engine of CarVal is implemented based on the MulVAL reasoning framework [99] and XSB database system [151]. Particularly, the calculation of the attack impact, attack feasibility, and overall risk value is implemented by Python with treelib library [135]. The code of CarVal is open-sourced in [20].

3.7 Experimental Analysis on Real Vehicles

In this section, we demonstrate how CarVal can be applied to real cars to assist the security analysis to exploit realistic threats in automotive systems.

3.7.1 Vehicles Under Examination



(a) *Car A*: Tesla Model S

(b) *Car B*: BMW i3

(c) *Car C*: Roewe Marvel X

(d) *Car D*: Mercedez Benz E-class

(e) *Car E*: Anonymous vehicle

Figure 3.7: The IVN topologies and POC attack paths of five investigated vehicles.

We apply CarVal to five modern vehicles: a Tesla Model S, a BMW i3, a Roewe Marvel X, a Mercedez Benz E-Class, and another anonymous vehicle, which we will refer to as *Car A*, *Car B*, *Car C*, *Car D*, *Car E*. Particularly, *Car E* is anonymized because the corresponding vulnerabilities are still under the process of being fixed. Meanwhile, for *Car A*, *Car B*, *Car C*, and *Car D*, all vulnerabilities have been reported and fixed by the responsible party. All five vehicles offer sophisticated IVI systems, supporting entertainment activities through audio/video players and Web browsers, and their manufacturers provide mobile applications for remote control, which are the new attack surfaces that previous works failed to consider [68, 21, 84]. In addition, all five vehicles' In-vehicle network have adopted the up-to-date domain E/E architecture instead of the old two-

bus in-vehicle network. Overall, the above new features distinguish our works from previous ones that exploited the *"old"* vehicles [68, 21, 84].

3.7.2 IVN Topology Discovery

We obtain the IVN topology of target vehicles from professional diagnostic tools ([70, 134, 4]), which are the devices programmed to communicate with the vehicle through diagnostic protocols to diagnose the possible problems of the ECUs. Particularly, these diagnostic tools are embedded with the *IVN topology* of the target vehicles, in which how the ECUs are connected is presented to help the experts quickly gain an overview of the vehicular architecture. Fig.3.7 shows the IVN topologies of the five target vehicles we derived from the diagnostic tool.

The derived topologies are then parsed to corresponding datalog clauses, which will serve as the *Vehicle Configuration*, as stated in §3.6.2. Particularly, we set various *Attack Goals* and *Attack Entry* as the input to CarVal, and the specific topologies serve as the *Vehicle Configuration*, to output specific attack paths. Based on the attack paths generated by CarVal, we perform security analysis accordingly and finally exploited the practical attack chains and launched PoC attacks on real cars. In the following sub-sections, we will detail these attack paths and our corresponding security analysis.

3.7.3 Path 1: Bypassing gateway: from IVI browser to BCM

The IVI ECU, which is responsible for entertainment and communication functionalities within the vehicle, often contains a wide range of attack surfaces, and is often equipped with functionalities to control the BCM ECU. However, as shown

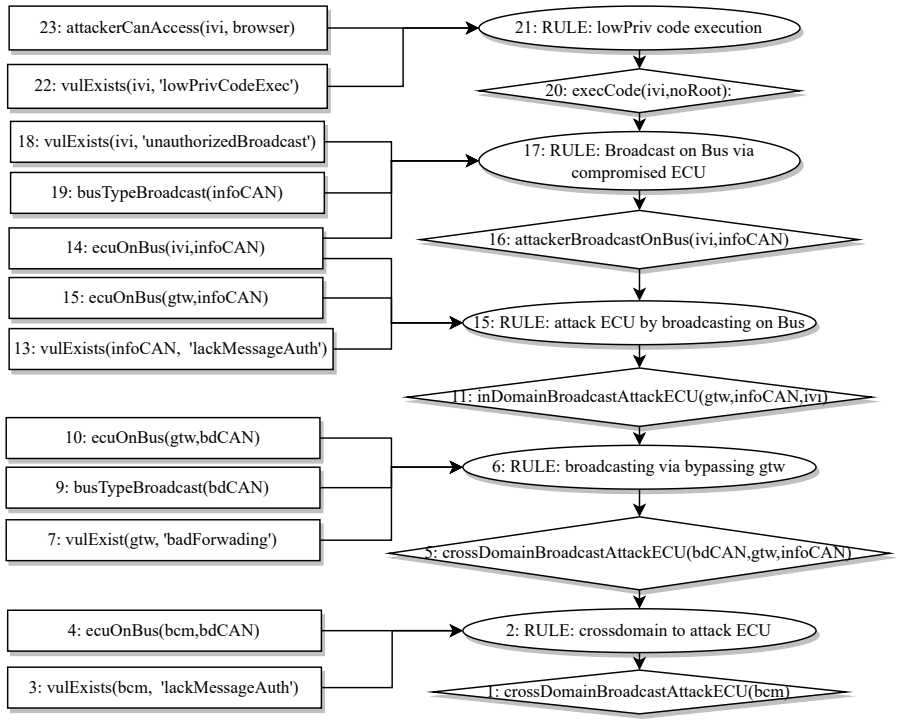


Figure 3.8: Attack Path 1: the attacker obtain code execution in IVI via IVI browser, and finally controls the BCM ECU via sending crafted bypass messages to gateway. This attack path is exploited on *Car A*, *Car B*, and *Car C*.

by the specific topologies, the IVI and BCM are segmented by the gateway ECU, making previous injection attacks [68, 21] infeasible in this new IVN topology. To demonstrate the capability of CarVal to generate *multi-stage* attack paths in the increasing complex IVN topology, we set the attack entry on the IVI ECU, and the attack goal on the BCM ECU, to generate the corresponding attack path that shed light on the subsequent analysis including TARA and security testing. The logical attack path is shown in Fig.3.8, in which five attack nodes (Node.20, 16, 11, 5 and 1) are involved to reach the final attack goal. First, the attacker exploits IVI vulnerabilities via the IVI browser to obtain code execution (Node.20). By compromising the interface between IVI and in-vehicle bus *infoCAN*, the attacker broadcasts messages on infoCAN (Node.16), affecting the gateway ECU

(Node.11). Then, the attacker crafts malicious messages to bypass the gateway and reach the *bdCAN* (Node.5), and finally transmits the attack message to the BCM ECU (Node.1).

PoC attack. The attack path in Fig.3.8 is validated on three vehicles: *Car A*, *Car B* and *Car C*, and we have conducted PoC attacks on these real vehicles. This attack path is demonstrated as the red bold line in Fig.3.7.(a), (b), (c) (from IVI to GTW and then to BCM). We first send malicious web pages to IVI browsers, obtain code executions on IVI, and compromise the interface from IVI to in-vehicle networks. By crafting bypass messages, we make the gateway transmit our malicious messages to the BCM ECU, allowing remote vehicle control.

Insights. By exploiting the attack path in Fig.3.8, we have demonstrated the significance potential threats in IVI software, with the representation of a new attack surface - browser, which is a general user interface that allows remote access and can be especially vulnerable if developed without caution [37, 40]. Additionally, we demonstrated the capability of CarVal to infer the multi-stage attack in complex IVN architecture.

3.7.4 Path 2: From Official APP to Car Control

The second attack path we demonstrate involves a replay attack, initiated from the official mobile app, to gain control over the vehicle's BCM. We selected the official mobile apps as the investigated attack surface due to their remote access functionality and the severe consequences of a potential compromise. Compared to dongle apps [149], which require a third-party device attached to the OBD port, the official app functions without external devices and is typically installed on the

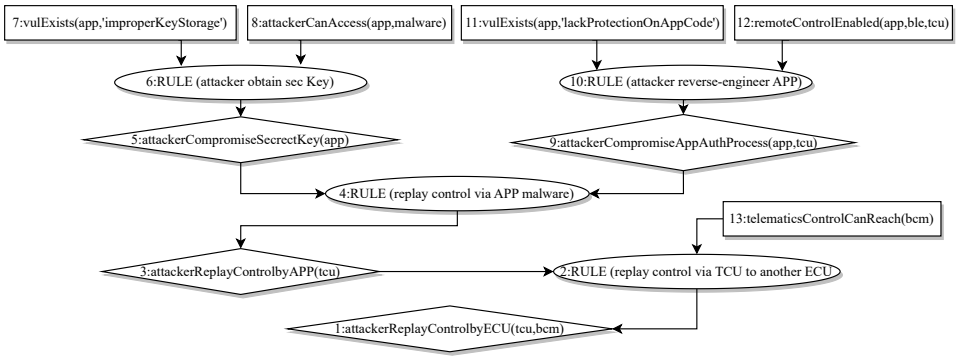


Figure 3.9: Attack Path 2: Control BCM by compromising the wireless communication between mobile app and telematic (i.e., TCU). This attack path is exploited on *Car C*.

vehicle owner's phone. Consequently, if this official app is compromised, many on-road vehicles could be affected, causing significant financial damage to the manufacturer. The associated logical attack path is illustrated in Fig.3.9, involving four attack nodes (Nodes 5, 9, 3, and 1) to achieve the final attack goal. First, due to insufficient application code protection (e.g., lack of code obfuscation or encryption), an attacker can conduct extensive security analyses on the application code (e.g., reverse-engineering) to recover the authentication process between the app and the Telematics Control Unit (TCU) (Node.9). Subsequently, with malware installed on the victim's mobile phone, the attacker can access the secret key used for the authentication process (Node.5). Upon obtaining both the secret key (Node.5) and the authentication algorithm (Node.9), the attacker can impersonate the official app using malware and launch a replay attack on the TCU (Node.3). Finally, since the TCU can invoke control functions on the BCM, the attacker can initiate these controls by launching replay attacks from the malware (Node.1).

PoC Attack. The attack path in Fig.3.9 was validated on *Car C*. This path is represented as the red bold line of *TCU to Gateway (GTW) to BCM* in Fig.3.7.(b).

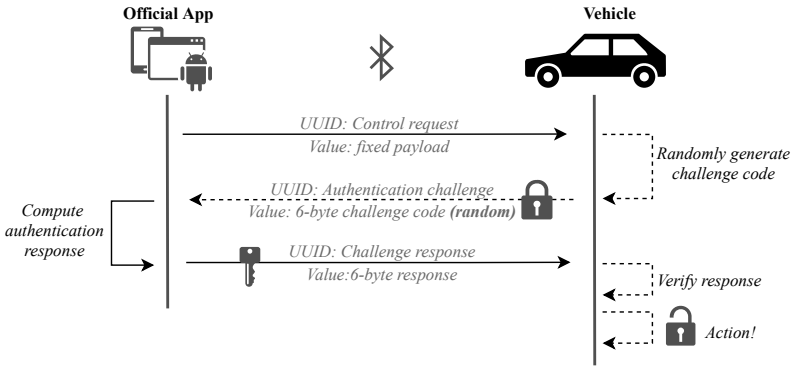


Figure 3.10: *Car C*: Remote control process of the mobile app via BlueTooth Low Energy (BLE).

Specifically, by conducting extensive security analyses on the corresponding mobile app, we recovered the authentication process, as shown in Fig.3.10. Our analysis revealed that: 1) the secret key used for authentication is set to update *every three months*, which is too long and allows the attacker the opportunity to crack this key; 2) the code contained in the APK file is not protected by obfuscation or encryption, enabling the attacker to directly access essential data through static analysis (e.g., the UUID used for BLE communication); 3) the code to generate the challenge response for the control request is called by the Java Native Interface (JNI) [63], and the critical authentication algorithm is stored in a *.so* file without obfuscation or encryption. Consequently, we launched a replay attack based on these vulnerabilities to remotely control the vehicle's door using an unrooted malware that sends crafted BLE messages, ultimately performing car control actions such as unlocking the door and opening the trunk.

Insights. While previous research focused on the security of OBD-dongle apps [149], we present the first practical attack that exploits vulnerabilities in the *official mobile app* to control a vehicle. We demonstrated that the official mobile app is a critical attack surface for modern vehicles and should be stringently protected.

3.7.5 Path 3: Multi-stage root via in-vehicle Ethernet

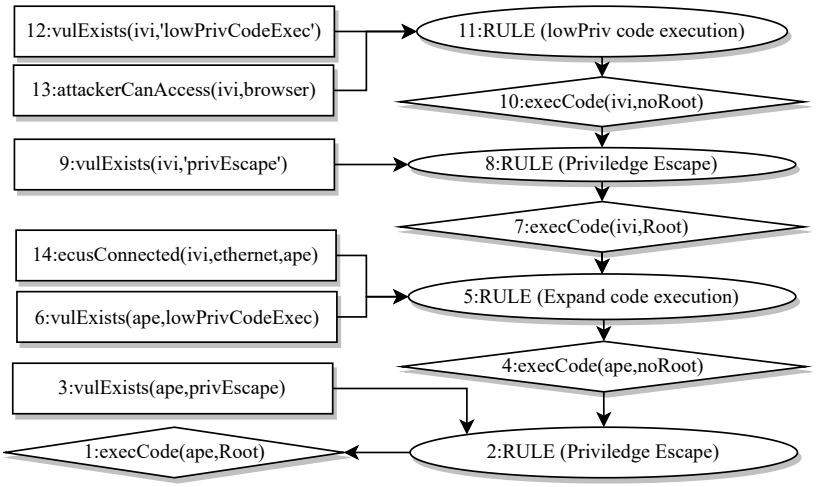


Figure 3.11: Attack Path 3: Multi-stage rooting via in-vehicle Ethernet. This attack path is exploited on *Car A*.

The third attack path we demonstrate illustrates a practical case of how an attacker can expand the scope of attack impact within an IVN. The corresponding logical attack path, based on the topology of *Car A*, is shown in Fig.3.11. In this attack path, the attacker first obtains root code execution in the In-Vehicle Infotainment (IVI) system by exploiting specific vulnerabilities (Nodes 10 to 7). Next, by leveraging a vulnerability between the IVI and Autopilot ECU (APE), the attacker expands code execution privileges from the IVI to the APE (Node.4). Finally, by exploiting a vulnerability in the APE, the attacker gains root execution in the APE (Node.1).

PoC attack. The attack path in Fig.3.9 is validated on *Car A*. Specifically, by conducting extensive security analysis on the IVI, we successfully obtain code execution via the browser's attack surface (Node.10) and further gain *root* code execution by exploiting a vulnerability in the outdated OS implementation (Node.7). Afterward, we identify that the IVI and APE communicate using an unencrypted

UDP [139] protocol. By exploiting a vulnerability in the APE's update process, we successfully execute our code in the APE (Node.4). Finally, by compromising the authentication within the APE, we obtain *root* code execution in the APE (Node.1).

Insights. The exploitation of Path 3 demonstrates how an attacker can expand the impact range in the IVN. Specifically, we show that an attacker can exploit in-vehicle vulnerabilities to gain root access to another ECU (APE) that has no direct communication channel with external clients. As IVN topologies become increasingly complex and information exchange between ECUs intensifies, it is crucial to address such threats to ensure IVN security.

3.7.6 Path 4: From Cloud to Car Control

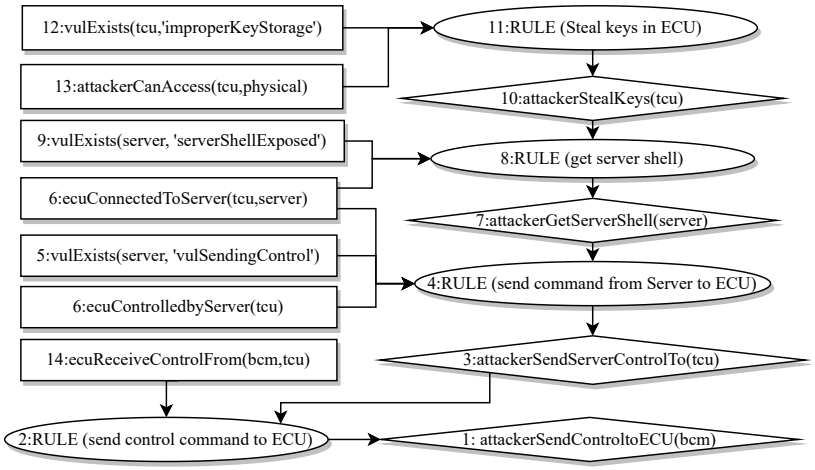


Figure 3.12: Attack Path 4: Compromising the backend server and sending control command back to vehicle. This attack path is exploited on *Car D*.

We also exploit a practical attack chain from a new attack surface, the *back-end server*, to control the car. The corresponding logical attack path is shown in Fig.3.12. Specifically, the Telematics Control Unit (TCU) is responsible for

remote communication, including transmitting messages with the mobile app (as shown in Path.3) and the backend server. Due to improper secret key storage in the TCU, an attacker can steal these keys by analyzing the TCU (i.e., dumping the firmware and performing reverse engineering), as shown in Node.10 of Fig.3.12. After obtaining the keys, the attacker accesses the server and achieves code execution (e.g., obtaining a shell) on the server by exploiting corresponding vulnerabilities (Node.7). With code execution, the attacker further analyzes how the server sends control commands to the TCU and replays these control commands (Node.3). Finally, upon receiving the malicious control messages sent by the attacker from the server side, the TCU triggers the control of the BCM (Node.1).

PoC attack. This attack path is validated and exploited on *Car D*. We first perform reverse engineering on the firmware dumped from the TCU and identify that the certificate used to establish authenticated connections with the intranet server is hard-coded in the firmware, which can be directly accessed (Node.10). Once the intranet server is accessible, we obtain the server shell by exploiting a Server-Side Request Forgery (SSRF) vulnerability (Node.7). With shell access on the server, we further analyze how the server sends commands to the vehicular TCU and can successfully send control commands to *any* vehicle by its Vehicle Identification Number (VIN) (Node.3), ultimately invoking control on the BCM (Node.1). In Fig.3.7.(d), this attack path originates from the Cloud, proceeds to the TCU, and bypasses the Gateway (GTW) to reach the BCM.

Insights. This attack path demonstrates the feasibility of vehicle control from the backend server, a new attack surface in modern vehicles. These servers are responsible for handling sensitive information, such as personal and financial data, as well as controlling critical systems within the vehicle.

3.7.7 Path 5: From IVI malware to Car Control

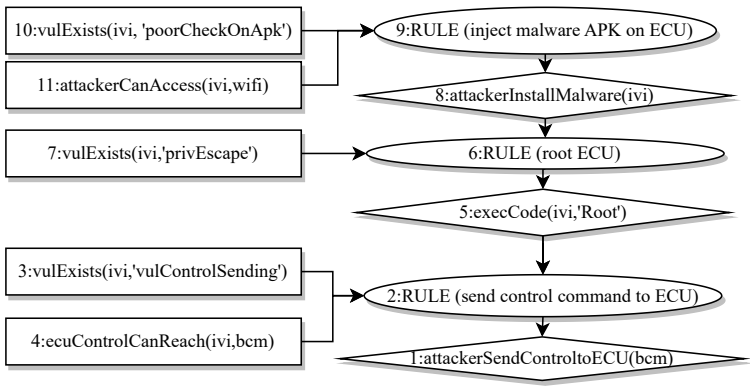


Figure 3.13: Attack Path 5: Invoking vehicular controls of BCM from the malicious application in IVI. This attack path is exploited on *Car E*.

The last attack path we demonstrate involves invoking the control of the BCM from a malicious application installed in the IVI. The corresponding logical attack path is shown in Fig.3.13. In the previous Path 2, we demonstrated that an attacker can launch remote vehicular control from malware installed on a victim's mobile device. However, as IVIs become more complex, many IVIs are now equipped with intelligent operating systems (e.g., Android or Android Automotive) that allow users to install various applications, introducing corresponding risks. Specifically, as shown in Fig.3.13, by exploiting a vulnerability in the installation process (e.g., lack of authentication on the APK file), an attacker can inject a malicious application into the IVI (Node.8). By further exploiting a vulnerability in the IVI, the attacker can escalate from low privilege to root code execution (Node.5), and finally send control commands to the BCM (Node.1).

PoC attack. The attack path is validated and exploited on *Car E*. Specifically, we identify that the IVI of *Car E* is implemented with the Android OS, allowing users to install Android applications. By conducting extensive security analysis

on the IVI and sniffing the network, we identify a vulnerability that allows us to launch a Man-In-The-Middle (MITM) attack and inject a malicious app into the IVI (Node.8). The malicious app contains code to root the IVI using a corresponding vulnerability (Node.5) and finally invoke vehicle controls on the BCM, including unlocking doors and opening trunks (Node.1).

Insights. As modern vehicles become increasingly intelligent and connected, the IVI system grows more complex, making it crucial to ensure the security of in-vehicle applications. This attack path highlights the importance of securing the IVI and related applications, as vulnerabilities in these systems can lead to attackers gaining control over critical vehicle functions.

3.7.8 Responsible Disclosure

We have reported all vulnerabilities identified in *Car A*, *Car B*, *Car C*, and *Car D* to the respective manufacturers, and they have been promptly addressed. For *Car E*, we have recently reported the corresponding vulnerabilities and are awaiting a response. Consequently, we have anonymized the car brand and specific model of *Car E* for the time being. The details of our security analysis on these vehicles, including the security testing process, disclosure timeline, and implemented fixes, are available in [20].

3.7.9 Summary on Our Attacks

We would like to emphasize our contributions from the following aspects:

Attack path guidance. As identified in our interviews, conducting security activities (e.g., TARA and security testing) can be labor-intensive due to the ab-

sence of automated tools. In our experimental analysis, we demonstrated how attack paths generated by CarVal can aid security testing, showing that our automated tool is a valuable complement to the TARA methodology [58].

New attack surfaces. We explored a range of new attack surfaces that previous studies did not address, emphasizing the importance of protecting emerging attack surfaces as the automotive industry rapidly evolves.

Multi-stage attack in complex IVNs. Our attack path reasoning and corresponding security analysis is based on the gateway-segmented IVN [66, 53] in five real vehicles, which is more complex than traditional architectures without gateways [68, 21]. Furthermore, increasingly complex and advanced IVNs are under development [7]. As such, CarVal can automatically reason attack paths in the context of these increasingly complex IVNs.

3.8 Discussion

While previous threat modeling tools [130, 113, 38, 100] provide instructions on performing each sub-task of TARA (e.g., the 7 TARA tasks indicated by ISO 21434), they mainly focus on presenting manual workflows without offering automatic solutions. In comparison, CarVal leverages Datalog to automatically infer attack paths and assess corresponding risks, making the entire TARA process more efficient. Additionally, existing threat modeling tools [130, 113, 38, 100] often offer high-level guidance with limited use cases, leaving gaps in their practical application to real vehicles. In contrast, CarVal provides a specific solution to model the IVN using Datalog clauses, enabling automatic attack path reasoning and risk assessment. We also demonstrated how CarVal aids analysis using real

vehicles as examples. Furthermore, it is important to discuss an earlier work [78] that investigates the safety aspects of autonomous driving systems and identifies similar challenges, such as limited automation and tool support. They also conducted semi-structured interviews with vehicle companies worldwide and went further by including an in-depth survey, which adds quantitative support to their findings and suggests future directions. In comparison, while [78] centers on the safety of autonomous driving systems, our study focuses on security regulations and standards, including WP29 R155 and ISO 21434. This serves as a supplement to their work, together offering a comprehensive view of industry needs from various aspects. Finally, note that CarVal is not intended to fully replace current threat modeling tools. Instead, it can be used in conjunction with other TARA approaches, including existing threat modeling tools. For instance, a team can first perform manual threat modeling of the automotive system (e.g., identifying assets and threat scenarios as indicated by ISO 21434 [58]), and then utilize CarVal to generate attack paths and assess risk values.

Chapter 4

Attacking ADS Perception

4.1 Overview

Autonomous driving is developing rapidly and has achieved promising performance by adopting machine learning algorithms to finish various tasks automatically. Lane detection is one of the major tasks because its result directly affects the steering decisions. Although recent studies have discovered some vulnerabilities in autonomous vehicles, to the best of our knowledge, none has investigated the security of lane detection module in real vehicles. In this paper, we conduct the *first* investigation on the lane detection module in a real vehicle, and reveal that the over-sensitivity of the target module can be exploited to launch attacks on the vehicle. More precisely, an over-sensitive lane detection module may regard small markings on the road surface, which are introduced by an adversary, as a valid lane and then drive the vehicle in the wrong direction. It is challenging to design such small road markings that should be perceived by the lane detection module but unnoticeable to the driver. Manual manipulation of the road markings

to launch attacks on the lane detection module is very labor-intensive and error-prone. We propose a novel two-stage approach to automatically determine such road markings after tackling several technical challenges. Our approach first decides the optimal perturbations on the camera image and then maps them to road markings in physical world. We conduct extensive experiments on a Tesla Model S vehicle, and the experimental results show that the lane detection module can be deceived by very unobtrusive perturbations to create a lane, thus misleading the vehicle in auto-steer mode.

In summary, we make the following major contributions:

- We conduct the first investigation on the security of the lane detection module in real vehicles and reveal that its sensitivity can be exploited by an adversary to generate fake lanes and consequently mislead the vehicle.
- We perform reverse engineering on the firmware of Tesla Autopilot to locate the input camera image and the output lane image. With this information, we propose a novel two-stage approach to generate the optimal perturbations against the lane detection module.
- We conduct extensive experiments on a Tesla vehicle (Tesla Model S)[127] to evaluate our approach. The experimental results show that the lane detection module in Tesla Autopilot is vulnerable to our attack and our approach can quickly generate effective perturbations.

4.2 Background

Deep neural networks (DNNs) have become the cornerstone of many technological advancements, particularly in the realm of computer vision and autonomous

systems. Despite their impressive capabilities, DNNs are not without vulnerabilities. Adversarial attacks, where models are fooled by inputs that are imperceptibly altered to humans, have exposed significant security concerns [121, 103, 75, 154]. These attacks are not limited to adversarial examples but extend to poisoning and backdoor attacks as well [45, 60, 104]. In the specific context of autonomous driving, lane detection is a critical component for vehicle safety, guiding vehicles to stay within their lanes. While traditional methods for lane detection have relied on hand-picked features [15, 65, 123], the advent of DNNs has significantly improved the performance of these systems due to their superior feature extraction capabilities [56, 72, 76, 90]. The security of autonomous driving systems, which often leverage DNNs for processing vision data, is paramount, as failures can have dire consequences [128, 137]. Physical adversarial examples have been demonstrated to mislead DNNs in real-world scenarios, such as misclassifying traffic signs [39] or causing incorrect vehicular navigation [155, 115].

4.3 Attack Methodology

In this section, we first introduce the threat model and then give an overview of our two-stage attack approach.

4.3.1 Threat Model

We assume that an attacker has an autonomous vehicle, whose lane detection module is the same as that of other vehicles of the same model, but does not have any previous knowledge about the module (i.e., black-box setting). The attacker aims to add unobtrusive markings on the ground so that the lane detection module rec-

ognizes them as a valid lane and consequently the victim autonomous vehicle will be misled.

An intuitive attack approach is to place markings at the possible area of the road and check whether the vehicle will be misguided. If not, the attacker can change the position and the shape of the markings and repeat the try-and-error method until the attack succeeds. However, this approach is very labor-intensive and error-prone because of the unlimited number of possible ways to modify and place the markings. Our approach to be described in §4.3.2 tackles these limitations.

4.3.2 Our Approach

This section introduces the workflow of our approach, the challenges to be addressed, and the key ideas of our solutions.

4.3.2.1 Workflow

We first locate the input camera image to the lane detection module and the corresponding output lane image by conducting static and dynamic analysis on the firmware (in §4.4). Then, we carry out the two-stage attack as shown in Fig. 4.1.

Stage 1. Finding the best perturbation in digital world. We formulate an optimization problem based on the visibility of the perturbation and the visibility of the corresponding detected lane to find the best perturbation that can lead to a fake lane but is unnoticeable to human perception.

Stage 2. Deploying markings in physical world according to the best perturbation. According to the best perturbation in digital world, we deploy the markings in physical world and then evaluate the attacks on a real vehicle.

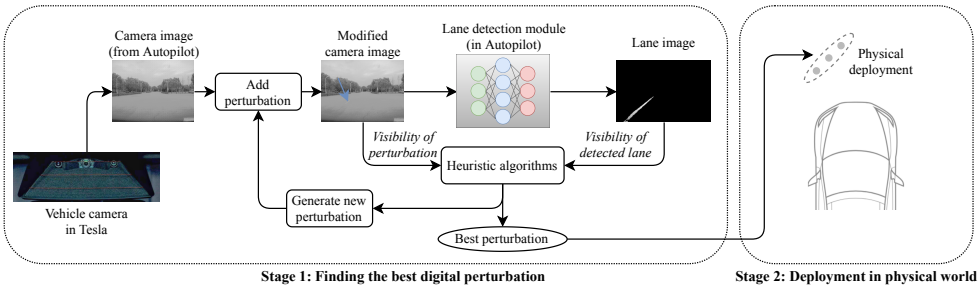


Figure 4.1: Overview of our two-stage approach. **In the first stage**, we add the perturbation, which is based on physical coordinate, to the camera image, and then feed the modified camera image to the lane detection module to generate the corresponding lane image. We formulate an optimization problem based on the visibility of perturbation and that of detected lane and adopt heuristic algorithms to find the best perturbation, which is unobtrusive to human but causes the lane detection module to output an obvious lane. **In the second stage**, we deploy the best perturbation in physical world according to the attributes of the best perturbation.

4.3.2.2 Challenges

Three challenges should be tackled to realize our approach.

C1. How to locate the input camera image and the corresponding output lane image in the vehicle? Our two-stage attack approach needs to access the input camera image and the output lane image. However, it is non-trivial to locate them since the lane detection module is in the closed-source firmware of Tesla Autopilot and the algorithms are executed in GPU using undocumented proprietary instruction sets.

C2. How to add perturbations to input camera image? An intuitive method is to add perturbations at the pixel level without considering the physical deployment. However, it may not be possible to implement such perturbations in physical world because it is not easy to accurately project the pixels to physical world, considering the distortion of the lens.

C3. How to find the best perturbations? The best perturbations should be as unobtrusive as possible so that drivers cannot notice them and meanwhile they can force the lane detection module to output a fake lane. It is challenging to find the best perturbations because the target model is in black-box setting so that the gradient-based optimization methods [109] cannot be applied.

4.3.2.3 Solutions

S1 (§4.4). We reverse engineer the firmware of Tesla Autopilot through static and dynamic analysis to locate the input camera image and output lane image. In particular, by exploiting the observation that Tesla Autopilot is powered by NVIDIA DRIVE technology [126] and its deep-learning computation follows the CUDA programming model [28] and is finished in GPU, we focus on locating and extracting the images in GPU memory. More precisely, after finding the binary responsible for lane detection, we conduct static analysis to find out when the images are available in GPU memory, and then instrument the binary and perform dynamic analysis to determine the memory addresses of the images. After that, we employ CUDA APIs to extract and modify the target images.

S2. We use a vector containing metrics from the physical world to represent the perturbations in digital world, and design the formula, which is based on the pin-hole camera model and camera calibration, to map the digital perturbation to the markings in physical world (in §4.5.1).

S3. We design two metrics to quantify the visibility of the perturbation and that of the corresponding detected lane, and formulate an optimization problem for the best perturbations (in §4.5.2.2). Then, we use five heuristic algorithms to find the best perturbation in digital world.

4.4 Accessing Data in Tesla Autopilot

This section details **S1** for locating the input camera image and the corresponding output lane image in the vehicle.

4.4.1 Overview

4.4.1.1 Firmware under examination

Our target vehicle is Tesla Model S 75, with the Autopilot hardware version of 2.5 and software version of 2018.6.1. It is worth noting that our methodology can be applied to other autonomous vehicles. The vehicle is running an AArch64 Linux operating system and uses NVIDIA GPU for deep learning computation. In the file system of Tesla Autopilot, there is a binary named *vision*. Through reverse engineering, we find that this binary is responsible for vision-related tasks including lane detection. It transmits the data of camera images into the GPU memory and finishes the vision-related computing tasks, in which lane detection is involved. The lane recognized by this binary will affect the steering decision when Autopilot is in auto-steer mode (demonstrated in §4.6). Since this *vision* binary can directly interact with the camera image and lane image in GPU memory, we carry out static and dynamic analysis on it to locate and access the target images.

4.4.1.2 CUDA

Tesla Autopilot uses NVIDIA GPU to execute its deep-learning algorithms, whose implementation follows the CUDA programming model [28]. We first introduce some necessary knowledge about CUDA programming because it is exploited by us to locate the target images.

CUDA programs usually involve two kinds of hardware: host (CPU) and device (GPU). If CPU needs to access data in GPU memory, it invokes a special kind of function named *kernel*s. A kernel is a function executed in the GPU as an array of threads in parallel [28]. These kernels will be launched and executed on GPU and manipulate data in GPU memory. In other words, kernels are the functions that run on GPU and launched by CPU. Since the lane detection is finished in GPU and the target images (camera image and lane image) are related to lane detection, the target images will be stored in GPU memory at certain time, and thus all we need to do is to determine ”when” and ”where”.

CUDA provides memory management functions [27] to access and manipulate data in GPU memory.

• ***cudaMalloc**** [30]: Functions whose names begin with *cudaMalloc* are used to allocate memory in GPU (except *cudaMallocHost* that allocates memory on CPU). We denote such functions as *cudaMalloc**, each of which has two types of parameters. One is the pointer to the allocated memory and the other represents the data's size information. *cudaMalloc** will act as the instrumentation location for locating the lane image in GPU memory (in §4.4.3 and §4.4.4).

• ***cudaMemcpy**** [31]: Functions whose names begin with *cudaMemcpy* are used to copy data from one address to another. We denote these functions as *cudaMemcpy**, which take in four types of parameters including source address, destination address, size information, and the mode that represents the direction of the copying operation: host to GPU, GPU to host, host to host or GPU to GPU. *cudaMemcpy** will act as the instrumentation location for locating the camera image in GPU memory (in §4.4.3 and §4.4.4). We also employ these functions to dump the target images from GPU memory after we get their address and size information.

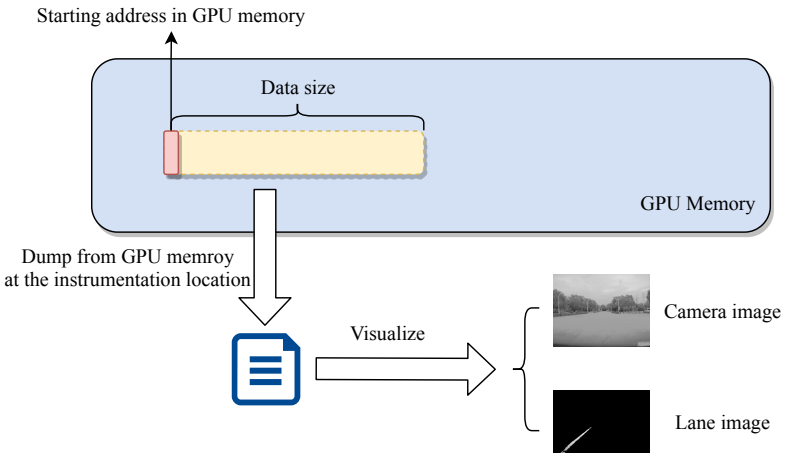


Figure 4.2: The process of dumping and visualize the target data

• ***cudaConfigurecall*** [29]: This function will be called before each kernel is invoked by the host to configure the launch on GPU. Hence, we can locate the kernels by locating the positions of *cudaConfigurecalls* in the binary for analysis. *cudaConfigurecall* will act as the instrumentation location for dumping lane image (in §4.4.3 and §4.4.4).

4.4.1.3 Factors required for dumping target images

We leverage the documented CUDA APIs to determine ”where” and ”when” to get the target images. In particular, we need to know the following three factors.

1. Instrumentation location. Since the lane detection is finished in GPU, the input camera images and the output lane images should be available in GPU memory after some specific functions are executed. We add instrumentation right after the invocation of such functions to get the target images.

2. Starting address of the images in GPU memory. It refers to the memory address where the image is stored in GPU memory. We need such addresses to locate the target images.

3. Data size. We need the size information to dump the images because they are stored in GPU memory as raw bytes. Moreover, to visualize the raw data (i.e., show the images), we need to know the image resolution (i.e., rows and columns) and the bit depth in each pixel. Fig. 4.2 shows how we dump and visualize the images from GPU memory. With the known starting address and data size, we instrument the binary to dump the image from the GPU memory in dynamic execution. The raw data in GPU memory are saved into a file and visualized according to the learnt resolution and bit depth.

We perform the following steps to determine these factors.

(1) Estimating data size (§4.4.2). We estimate the data size of camera images from the relevant document of the hardware camera [126]. For lane image, we conclude the data size from a file in Tesla Autopilot.

(2) Conducting static analysis to collect instrumentation location candidates (§4.4.3). We aim to dump the camera image right after *cudaMemcpy** is used to copy the image into GPU memory. Similarly, we dump the lane image right after the kernel for lane detection finishes its task. We conduct static analysis on the *vision* binary to find a list of candidates, including the invocations of *cudaMemcpy** (i.e., candidates for dumping the camera images) and the kernels (i.e., candidates for dumping the lane images).

(3) Performing dynamic analysis to determine instrumentation location and starting address in GPU memory (§4.4.4). Since the specific GPU memory address and the context can only be revealed during execution, we perform dynamic analysis to determine the correct instrumentation location and starting address. Specifically, for input camera image, we hook all *cudaMemcpy** calls and locate the one responsible for copying camera image by checking its parameters. Simi-

larly, we first hook all *cudaMalloc** to find the starting address of the output lane image, and then determine the kernel by checking the visualized lane image after all possible kernels based on the data size and starting address.

4.4.2 Estimating Data Size

Size of camera image. We find the camera image's resolution (i.e., 1280×960 pixels) according to its hardware [126], however, the bit depth is still missing. Therefore, we compute 32 possible data sizes according to the possible bit depth, namely from 1-bit to 32-bit, to cover most of the possible bit depth used in digital images. For example, if an image is in 16-bit bit depth, the data size is 1280 $\times$ 960 $\times$ 16 = 19,660,800 bits (or 2,457,600 bytes). After this estimation, we get a list of the possible data size for camera image. The specific bit depth will be determined in dynamic analysis in §4.4.4 by hooking the *cudaMemcpy** calls.

Size of lane image. We find a file in the file system of Tesla Autopilot, which provides information about the architecture of the deep neural network used for object detection tasks (including lane detection), such as data size and pixel depth of the data matrix in each layer. This network has several outputs and the lane detection result is one of them, which is a 640×416 matrix with 32 bits float values. With this information, we can estimate the data size of the lane image output, which should be $640 \times 416 \times 32 = 8,519,680$ bits (or 1,064,960 bytes). The size of the lane image will be the key information for hooking the *cudaMalloc** in order to find the starting address of the lane image (in §4.4.3 and §4.4.4).

4.4.3 Conducting Static Analysis

Using IDA-Pro [57], we conduct static analysis on *vision* binary to determine the instrumentation locations and add instrumentation code. We detail the instrumentation locations for collecting camera images and lane images, respectively.

1.Instrumentation locations for collecting camera images. Since lane detection is finished in GPU, *cudaMemcpy* will be used to copy the input camera image into GPU memory before processing. Hence, we add instrumentation right after the invocation of *cudaMemcpy* for copying data into GPU memory. The instrumentation code will collect the parameters passed to the *cudaMemcpy*, including (1) source address, (2) destination address, (3) data size, and (4) mode of transfer, when being executed in dynamic analysis.

2. Instrumentation locations for collecting lane images. We are interested in two kinds of instrumentation locations:

• **Hooking *cudaMalloc* to determine the starting address***. Since *cudaMalloc** is responsible for allocating memory in GPU, the memory of the lane image will be allocated by *cudaMalloc**. In this case, we add instrumentation right after the invocation of each *cudaMalloc**, and collect the (1) memory address and (2) data size passed to *cudaMalloc**. By locating the *cudaMalloc** whose data size is equal to the estimated lane image size, we can determine the *cudaMalloc** that allocates the memory of the lane image, thus knowing the starting address of the lane image in GPU memory.

• **Hooking kernels to determine instrumentation location for dumping lane images**. Since kernel functions are responsible for the computation in GPU, we first enumerate all kernels according to the invocation of *cudaConfigureCall*. There

are totally 75 calls of *cudaConfigureCall* by 22 different callers. Then, we add instrumentation right after the invocation of each kernel, because one of them will be responsible for lane detection and we can collect the lane image right after it finishes. The instrumentation code will dump the lane image in GPU memory according to the given starting address (found by hooking *cudaMalloc**) and data size.By checking whether the visualized image is the desired lane image, we identify the kernel function for lane detection.

4.4.4 Performing Dynamic Analysis

We execute the instrumented *vision* binary to (1) get the parameters passed to the hooked *cudaMemcpy** for obtaining the starting address and data size of the camera image and determining the correct instrumentation location; (2) get the parameters passed to the hooked *cudaMalloc** for obtaining the starting address of the lane image, and (3) dump the lane image after each kernel candidate to determine the instrumentation location of the lane image. The processes for camera images and lane images are described as follows.

1. Camera image. Through dynamic analysis, we collect the following information relevant to the input camera image: (1) data size, (2) the call of *cudaMemcpy** which copies the camera image to GPU memory, (3) the starting address of camera image in GPU memory. As specified in static analysis, we add instrumentation after each *cudaMemcpy** and collect the parameters passed to *cudaMemcpy** in dynamic execution. From the experiment results, among the 32 different estimated sizes, only a data size of 2,457,600 bytes is found, meaning the bit depth of the input image is 16-bit.

2. Lane image. For lane image, we have determined the data size in §4.4.2, and list the 75 candidate kernels. Through dynamic analysis, we obtain the following information: (1) the starting address of the lane image, and (2) the kernel that is responsible for lane detection among the candidates. We first finish task (1) by hooking the *cudaMalloc**, and accomplish task (2) based on the found GPU address in task (1). Next, we describe how we determine the starting address (task (1)) and how we determine the instrumentation location (task (2)) of the lane image, respectively.

• **Determining starting address of the lane image.** As specified in static analysis, we select a list of instrumentation locations for *cudaMalloc** to find the starting address of the lane image. Using IDA-Pro, we find 77 calls of *cudaMalloc**. We add instrumentation to check the parameters passed to *cudaMalloc** every time it is called, and aim to find the *cudaMalloc** call whose data size is our estimated size. After dynamic execution, we find the specific call of *cudaMalloc** whose size is our estimated size (1,064,960 bytes), and locate the address of the lane images by this specific *cudaMalloc**.

• **Determining instrumentation location of the lane image.** As mentioned in static analysis, for lane image, we find 75 possible places in *vision* binary for instrumentation. Based on the found GPU memory address of the lane image, we add instrumentation to dump the images after all these kernel candidates. By visualizing the dumped data, we learn that the kernel in the function named *t_cuda_lane_detection::compute* is responsible for lane detection.

Remark. We summarize the factors for camera image and lane image. For camera image, the instrumentation location is right after the invocation of *cudaMemcpy**; the starting address is the destination address passed as a parameter to the spe-

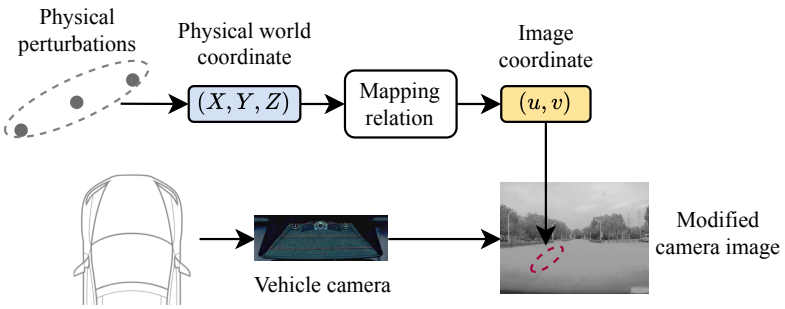


Figure 4.3: Mapping the coordinate of (X, Y, Z) on markings in physical world to the coordinate of (u, v) on perturbations in digital world.

cific *cudaMemcpy**; the data size is 2,457,600 bytes, with 1280×960 resolution and 16-bit bit depth. For lane image, the instrumentation location is right after the execution of function *t_cuda_lane_detection::compute*; the starting address is the address passed to the specific *cudaMalloc** which allocates the memory for the lane image; the data size is 1,064,960 bytes, with 640×416 resolution and 32-bit bit depth.

4.5 Two-Stage Attack

This section describes how we add digital perturbations based on the physical metrics and how to find the best perturbations, which are the solutions to **C2** and **C3**, respectively.

4.5.1 Adding Digital Perturbations

This subsection describes the solution to **C2**. The goal is to obtain the digital perturbation which is defined by physical-world attributes for easy physical deployment.

Parameters	Explanation
len	Length of a single perturbation
wid	Width of a single perturbation
D_1	Longitudinal distance from the vehicle camera to the edge of the first perturbation
D_2	Lateral distance from the vehicle camera to the edge of the first perturbation
D_3	Distance between adjacent perturbations
ΔG	Increment of grayscale value of the perturbed pixels
θ	Rotation angle of the perturbation
n	Number of the perturbations

Table 4.1: Parameters determining the added perturbations

4.5.1.1 Projecting Physical World Markings

As shown in Fig.4.3, we use (X, Y, Z) to denote the coordinate of each pixel on the markings in real world, which is the coordinate relative to the vehicle camera, and utilize (u, v) to denote the coordinate of the corresponding pixel on the perturbation added to the image. With the pinhole camera model, we project (X, Y, Z) to (u, v). We also undistort the image to eliminate errors due to lens distortion through camera calibration, thus making the projection more accurate. With this mapping relationship, we can map any physical world coordinate (X, Y, Z) to image coordinate (u, v). Hence, given a set of coordinates describing the position of the perturbations in physical world, we can project them to digital world and find their corresponding pixels in the camera image. Moreover, by modifying the grayscale value of the corresponding pixels, we can add the digital perturbations according to the physical perturbations. The reason is that in physical world the colors of the lane lines are mostly white and yellow, and they are brighter than the ground. Consequently, the lane line pixels in digital images are also brighter than the surrounding pixels on the ground. Therefore, raising the grayscale value

(representing brightness) of the selected pixels in the captured digital image can result in the perturbations.

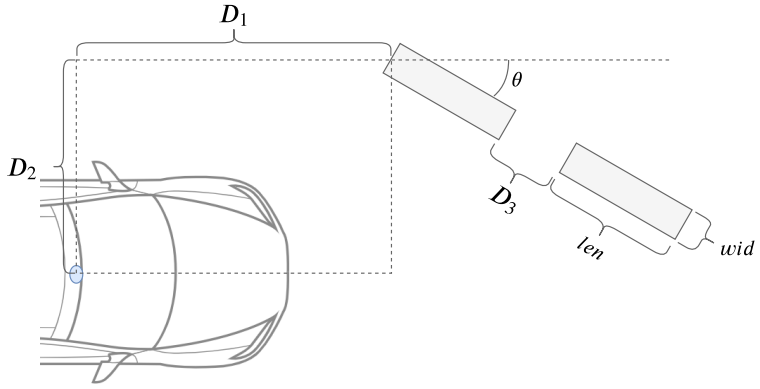


Figure 4.4: Illustration of the parameters of perturbations.

4.5.1.2 Parameterized Perturbations

For the ease of deployment, we use 8 parameters, which are listed in Table 4.1 and shown in Fig.4.4, to characterize the digital perturbations. len and wid determine the shape of the perturbations. D_1, D_2, and D_3 determine the position of the perturbations. ΔG is the increment of grayscale value of the pixels on the perturbation. n represents the number of perturbations (for example, $n = 2$ in Fig.4.4). Higher value of ΔG and more number of perturbations n make the added perturbation more obvious. θ is the rotation angle of the perturbations. The 8 parameters comprise a vector x:

$$x = (len, wid, D_1, D_2, D_3, \Delta G, \theta, n) \in X \tag{4.1}$$

The measurement of len, wid, D_1, D_2, D_3 and θ is based on physical metrics. The unit of len, wid, D_1, D_2, D_3 is centimeter, and that of θ is degrees. ΔG is

an 8-bit number ranging from 0 to 255 (we convert the 16-bit camera image into 8-bit for the ease of computing and visualization). Note that when $n = 1$, D_3 is invalid and has no influence on the added perturbation, because there is only one perturbation in view.

The range of x is denoted as X. len, wid, D_1, D_3, ΔG and n should be positive values. D_2 and θ can be positive or negative. Positive values of D_2 mean that the perturbation is on the left side of the vehicle, and negative means the right side. Positive value of θ represents that the perturbation is rotated towards the right direction of the vehicle, and negative means left.

4.5.2 Finding the Best Perturbations

We design two metrics to quantify the quality of the perturbations in digital world, based on which we construct an optimization problem for finding the best perturbations.

4.5.2.1 Quality of Perturbations

Since a good perturbation should be unnoticeable to the driver but cause the lane detection module to generate a fake lane, we quantify its quality from the following two aspects:

Visibility of lane. The perturbations should lead to a strong and stable fake lane in the output lane image.

Visibility of perturbation. The perturbations should be as unobtrusive as possible.

We define the following two metrics to quantify the visibility of lane and that

of perturbation, respectively:

$$V_{lane}(x) = \sum_{p \in lane_o(x)} G_p \tag{4.2}$$

$$V_{perturb}(x) = \sum_{p \in perturb_i(x)} \Delta G, \quad \Delta G \in x \tag{4.3}$$

We also define $S(x) = \frac{V_{lane}(x)}{V_{perturb}(x)}$ to be the overall score of perturbations. The explanations of the equations are listed in Table 4.2.

Parameters	Explanation
p	One single pixel in the image
$lane_o(x)$	Lane pixels in the output image
$perturb_i(x)$	Pixels on the added perturbations
G_p	Grayscale value of pixel p
$V_{lane}(x)$	Visibility of the fake lane created by x
$V_{perturb}(x)$	Visibility of the perturbations added by x
$S(x)$	Overall score of the parameter x

Table 4.2: Equation parameters explanations

$V_{lane}(x)$ denotes the visibility of the fake lane in the output lane image. It is computed by summing up the grayscale values of each lane line pixel (each G_p represents the confidence of the current pixel). The higher value of $V_{lane}(x)$ represents higher visibility of the fake lane.

$V_{perturb}(x)$ is the visibility of the perturbation added to the input camera image. This score combines the number of added pixels and the increment of grayscale values of these pixels to represent visibility. The lower value of $V_{perturb}(x)$ means that the perturbations are more unobtrusive to human.

$S(x)$ is the overall score of the crafted perturbation. A high value of $S(x)$ means that the perturbation leads to a strong fake lane while being unobtrusive at

the same time. If the perturbations fail to create a fake lane, $S(x)$ should be zero.

4.5.2.2 Optimization problem

To achieve the best attack performance, we look for x^* that results in the highest overall score $S(x)$.

$$x^* = \max_{x \in X} S(x), \tag{4.4}$$

where x is a 8-dimension vector in range X, and the output score $S(x)$ is a real number. We use five heuristic algorithms to find x^*, namely beetle antennae search (BAS), particle swarm optimization (PSO), beetle swarm optimization (BSO), artificial bee colony (ABC) and simulated annealing (SA). To solve the optimization problem, these algorithms first initialize one or more random input vector(s), and iteratively improve the input vector(s) based on the output score.

These algorithms could be differentiated according to two aspects. First, is the algorithm greedy or not? ”Greedy” means that the algorithm always updates the searching position to the direction where the target value is likely to be higher. BAS, PSO, and BSO are ”Greedy”, because they always encourage the searching position to move to coordinates where the value is higher, based on the hints found by the algorithms. ABC and SA are not ”Greedy”, because they essentially randomly update the position, and accept better solutions with higher possibilities. Second, do the searching individuals of an algorithm adopt a cooperative way to share information or not? ”Cooperative” means that the searching individuals will share information with others, and update positions based on the group information. PSO, BSO, and ABC are ”Cooperative”, because each individual in the group shares his own information to help other individuals. By contrast, in BAS and SA,

each individual works independently.

Note that n and θ are not put into the algorithms due to two reasons. First, since perturbation number n is a discrete variable while other parameters are all continuous variables, the optimization problem will become a mixed discrete-continuous optimization problem if n is considered and it is hard to find the optimal result. We will investigate it in future works. Second, since rotation angle θ is determined by the intention of the attack, a value of θ found by the algorithms may not meet the demand of the attacker. Therefore, we fix n and θ to constants, and discuss their impact in §4.6.

We evaluate our attack on the lane detection module by answering six research questions (RQs).

RQ1: How efficient are the heuristic algorithms to find the best perturbation?

Motivation: We want to identify the most efficient heuristic algorithm for finding the best perturbations.

Approach: We carry out the experiment with five heuristic algorithms, namely BAS, PSO, BSO, ABC, and SA, where PSO, BSO and ABC require multiple inputs working together, because these inputs will share information with each other, whereas BAS and SA work with a single input. For fair comparison, we also let BAS and SA have multiple inputs.

When looking for the best x, we record both the highest score $S(x)$ of the perturbations in history (top-1 score) and the average score of the top 10 perturbations (top-10 averaged score) to rule out contingency (i.e., an algorithm *accidentally* finds the best solution). If one algorithm achieves high score in both top-1 score and top-10 averaged score, its efficiency is no coincidence and is reproducible.

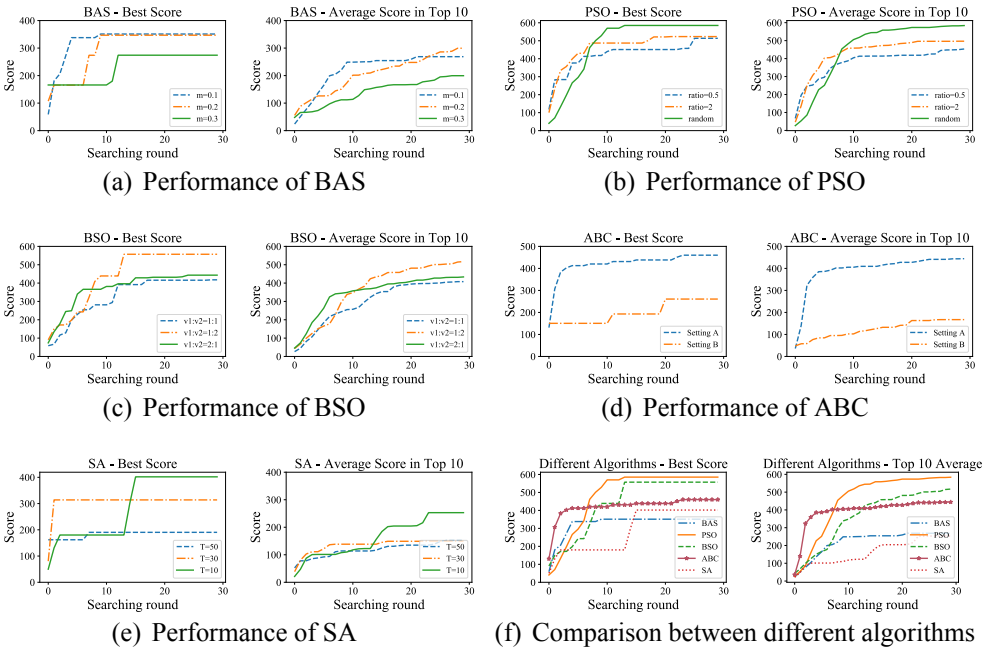
Since the effect of parameter n and θ is evaluated in *RQ2*, in $RQ1$ we let $n = 1$

and $\theta = 0$. Moreover, we focus to generate perturbation only on the left-hand side in RQ1 and discuss the right-hand side in RQ2. We implement the five algorithms with Python, and evaluate their performance with different parameters.

Results: Fig.4.5(a)-(f) show the experimental results. The X-axis is the number of search rounds, and the Y-axis is the best $S(x)$ (left figure) or the top-10 averaged $S(x)$ (right figure) of the current search round. For an efficient algorithm, it should (1) converge quickly, and (2) achieve high score in both top-1 $S(x)$ and top-10 averaged $S(x)$. Fig.4.5(a)-(e) represent the performance of BAS, PSO, BSO, ABC and SA, respectively, and Fig.4.5(f) compares the best results of the five algorithms. As shown in Fig.4.5(f), the five algorithms have different performance. The experimental results show that "Greedy" and "Cooperative" algorithms (e.g. PSO, BSO) converge faster and find higher score in both top-1 $S(x)$ and top-10 $S(x)$, than other algorithms. Moreover, according to Fig.4.5.(f), PSO finds the highest $S(x)$ (both top-1 and top-10) among all five algorithms. Only ABC converges faster than PSO, however, the top-1 and top-10 averaged $S(x)$ found by ABC are much lower than that of PSO.

4.6 Evaluation

Fig.4.6 shows one of the best perturbations. Given the original input camera image, the lane detection module does not output a lane. After an unobtrusive perturbation (pointed out by the arrow in the image) is added, a clear lane is detected and shown in the output lane image, although the perturbation is nearly invisible to human perception and is unlikely to be treated as a valid lane. The parameters of this perturbation is: $wid = 1cm$, $len = 92cm$, $D_1 = 1365cm$, $D_2 = 233cm$,



(a) Performance of BAS (b) Performance of PSO

(c) Performance of BSO (d) Performance of ABC

(e) Performance of SA (f) Comparison between different algorithms

Figure 4.5: Results of the different algorithms. Overall, PSO has the best performance, and is the most suitable heuristic algorithm in our research.

$\Delta G = 12$ (n, θ and D_3 have no influence in the setting here).

Answer: All heuristic algorithms can find best perturbations. PSO is the most efficient one and thus we use it in other experiments.

RQ2: How do the perturbation number n and the rotation angle θ affect the best perturbation?

Motivation: As mentioned in §4.5.2.2, we do not put perturbation number n and rotation angle θ into the heuristic algorithms. In this RQ, we study how n and θ influence $S(x)$.

Approach: We adopt the same image used in RQ1 as input to generate the perturbations. The perturbation number is set from 1 to 5, and the absolute value of θ is 0 to 30 degrees with the interval of 5 degrees. In this case, we have 5 settings of n (from 1 to 5), and 14 settings of θ (from 0 to 30 degrees on both sides of the image).

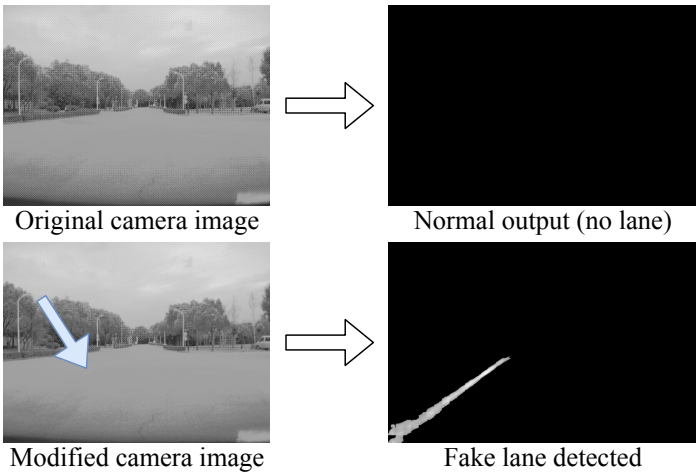
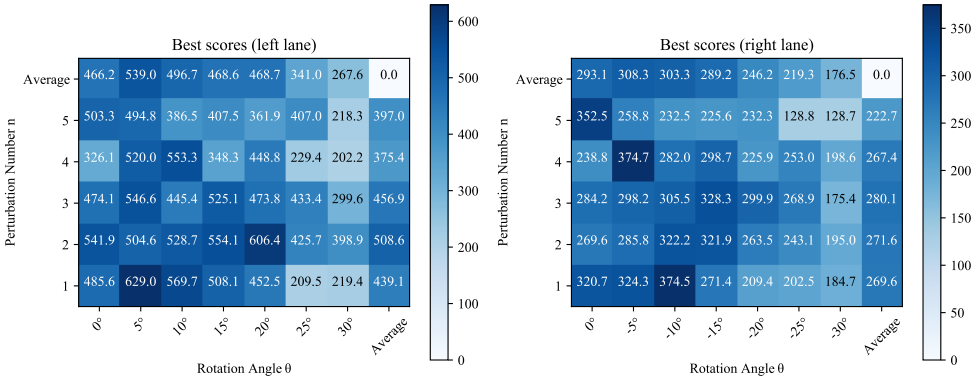


Figure 4.6: Effect of a best perturbation. The added perturbation is only 1cm wide in physical world, but it causes the lane detection module to generate a fake lane.

We consider all the possible settings for n and θ, thus getting totally $5 \times 14 = 70$ different settings of n and θ. Then, we search for the best perturbations on each setting, and record their $S(x)$s.

Results: Fig.4.7 shows the scores of the best perturbations found in different number n and rotation angle θ. The X-axis represents θ and Y-axis represents n. The intersection of two coordinates represents the best score $S(x)$ under the corresponding settings. The first row of the figure represents the average $S(x)$ under the specific θ, and the last column represents the average $S(x)$ under the specific n. The average $S(x)$ under each setting represents the overall effectiveness for this setting. For example, the third element on the first row represents the average $S(x)$ when $\theta = 10°$ and n is from 1 to 5, and this $S(x)$ represents the overall effectiveness of the perturbations when $\theta = 10°$.

By observing the average $S(x)$ in each θ (first row of Fig.4.7), we find that the average $S(x)$ decreases with θ, for both left and right lane. Specifically, when $\theta = 25°$ and $30°$, the average $S(x)$ is obviously lower than that in other settings of



(a) Left lane (b) Right lane

Figure 4.7: Best scores ($S(x)$) in different setting of n and θ in $RQ2$. The perturbations works well in different perturbation number n, and the score reduces with perturbation angle θ increasing.

θ. Similarly, by observing the average $S(x)$ in each n (last column of Fig.4.7), for left lane, we find that the perturbations with $n \leq 3$ have the higher average $S(x)$ than $n = 4$ and $n = 5$, while for right lane, the average $S(x)$ is similar among all settings of n.

Answer: Perturbation number n does not have significant effect on $S(x)$. Rotation angle θ reduces $S(x)$ when it increases.

RQ3: How is the performance of our approach given different input camera images?

Motivation: The experiments for answering RQ1 and RQ2 are based on the same input image shown in Fig.4.6. To answer RQ3, we generate perturbations on different input images to evaluate the effectiveness of our approach.

Approach: Besides the input image shown in Fig.4.6, we use four other images taken by the vehicle camera in different environments to carry out the experiment. They are shown in Fig.4.8 and their environmental features are listed in Table 4.3. NUM.1 is the input image used in *RQ1* and *RQ2*. NUM.2 and NUM.3 are in the

same outdoor environment but under different light conditions. NUM.4 is taken in an underground garage, where the ground is clean and the light is dim. NUM.5 is a corner where the ground is dirty. The corresponding output lane images of these original input images do not have a lane on the expected side before we add any perturbation to them. Similar to the settings in **RQ1**, we let $n = 1$ and $\theta = 0$ and use $S(x)$ to evaluate the effectiveness of our attack.

Num	Environmental Features
1	Clean and bright ground, without other disturbing objects in view
2	Clean and bright ground, with disturbing objects in view
3	Clean and dark ground, with disturbing objects in view
4	Clean and dark ground, without other disturbing objects in view
5	Dirty and bright ground, with disturbing objects in view

Table 4.3: Environmental features of different input images

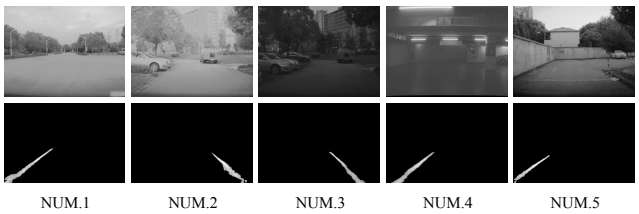
Results: The input images with the best perturbations and the corresponding lane images are shown in the upper row and the lower row of Fig.4.8.(a), respectively. The $S(x)$ of these examples are shown in Fig.4.8.(b). NUM.1 and NUM.4 lead to higher score than the others, because the grounds in both images are clean and a small perturbation can easily result in a fake lane in the output lane image. Although the scores of NUM.2/3/5 are relatively low, the perturbations are unnoticeable to human eyes and the fake lane is valid and strong.

Answer: Given different input images, our approach can successfully generate high-score perturbations that can mislead the lane detection module without being noticed by the driver.

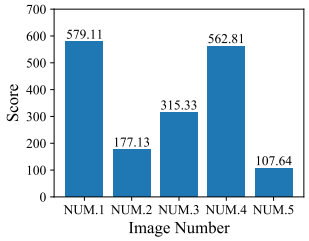
RQ4: What are the common characteristics of the best perturbations?

Motivation: We want to summarize the common characteristics of the best perturbations obtained in different scenarios and discuss their implication.

Method: We analyze the parameters x of the best perturbations obtained in the five



(a) Perturbations in different input images and the corresponding outputs



(b) Scores of the perturbations

Figure 4.8: $RQ3$: The output lane and corresponding scores based on different input images. In all five different settings, we manage to find the unobtrusive perturbations which fool the lane detection module.

Num \ x	wid	len	D_1	D_2	ΔG
NUM.1	1cm	117cm	15.30m	2.23m	12
NUM.2	5cm	59cm	13.37m	2.27m	28
NUM.3	3cm	72cm	12.53m	1.51m	12
NUM.4	1cm	133cm	11.68m	1.79m	7
NUM.5	1cm	83cm	10.14m	2.38m	25
Average	2cm	93cm	12.60m	2.04m	17

Table 4.4: Parameter values of the best perturbations generated for five different input camera images.

different scenarios for answering $RQ3$ and summarize the common characteristics. x is a 8-dimension vector but we focus on five dimensions in x, including wid and len that denote the shape of the perturbation, D_1 and D_2 that indicate the relative position, and ΔG represents the increment of grayscale value of the perturbations. We do not study n and θ because they are fixed in the experiments. Moreover, D_3 is meaningless when $n = 1$.

Results: Table 4.4 lists the values of these five dimensions of the best perturbations to different images. We summarize the characteristics from the following three aspects.

• **Shape** In all scenarios, wid is much smaller than len, meaning that the 'narrow

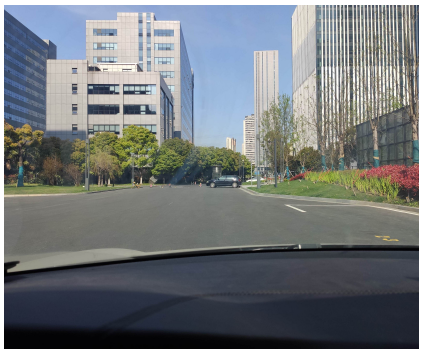


Figure 4.9: The road with the crafted markings from the driver's view. The sticker on the left side of the road is very unobtrusive and can hardly be noticed by human.

but long' perturbations are more effective than the 'wide but short' perturbations.

• **Position** For the position of the perturbation, D_1 ranges from 10.14m to 15.30m, and D_2 ranges from 1.51m to 2.23m.

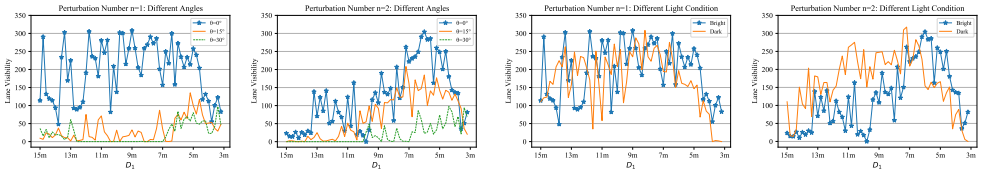
• **Increment of grayscale value:** The value of ΔG varies in different input images. For clean ground (NUM.1 and NUM.4) or dark grounds (NUM.3 and NUM.4), a small increment can make the lane in the output image very obvious, whereas 'dirty' grounds (NUM.2 and NUM.5) require larger value of ΔG to generate a fake lane.

Answer: 'Narrow but long' perturbations are more likely to create a fake lane. The required increment of grayscale value (ΔG) depends on the brightness and cleanliness of the ground.

RQ5: How effective is the attack in physical world?

Motivation: As RQ1-4 study the attacks in digital world, for RQ5, we evaluate the attacks in physical world by deploying markings on road surface according to the best perturbations.

Approach: We first let the vehicle generate the input camera image in an area for conducting this experiment, and then user our approach to find the best perturba-



(a) Lane visibility of each frame in different n and θ

(b) Lane visibility of each frame in different n and light conditions

Figure 4.10: The visibility of lane changes with D_1. Straight perturbations ($\theta = 0$) have higher lane visibility. Perturbation number n and light condition have little effect on the lane visibility. Interested readers are referred to our demo video[33].

tions. After that, according to the information of the best perturbation, we deploy the markings (i.e., stickers) on road surface and evaluate the visibility of the fake lane in the lane image. We adopt the following settings for this experiment.

• **Perturbation number** n**.** Since the answer to RQ2 shows that n has little effect on $S(x)$ of the perturbations, we choose $n = 1$ and $n = 2$ for the ease of deployment.

• **Rotation angle** θ**.** Since the answer to RQ2 shows that θ will reduce the value of $S(x)$, to evaluate whether the visibility of the lane will also be affected in physical world, we set different values to θ (0, 15° and 30°) in the experiment.

• **Light condition.** We conduct the experiment in both light and dark environments to evaluate the effect.

• **Longitudinal Distance** D_1**.** After deploying the stickers, we drive the vehicle from far to close to them, and record the visibility of the lane image ($V_{lane}(x)$) to evaluate the effectiveness of the attack with different D_1. Specifically, we drive from $D_1 = 15m$ to $D_1 = 3m$, and record 60 frames of the lane images during the process.

• ΔG**.** It is difficult to implement ΔG precisely in physical world because it will be affected by some uncontrollable factors, such as the environment's light condition

of and the texture of the physical perturbations. In this experiment, we use white stickers, which offers high value of ΔG, to construct the perturbations in physical world.

We use the algorithm (PSO) to find the best digital perturbation for the scenarios of $n = 1$ and $n = 2$, respectively. When $n = 1$, its length len is 1.5m, and its width wid is 1cm. when $n = 2$, the length of each perturbation is 0.4m, the width is 1cm, and the adjacent distance is (D_3) 0.7m.

Fig.4.9 shows the driver's view of the road with the crafted markings (single perturbation). The stickers are placed on the left side of the vehicle, and can hardly be noticed by human.

Results: The lane visibility in this experiment is represented in Fig.4.10. The X-axis is the longitudinal distance (D_1) of each frame, and the Y-axis is the lane visibility $V_{lane}(x)$. Larger value of $V_{lane}(x)$ means that the attack is more effective. We also have the following observations.

• **Perturbation number** n**.** Compared with the setting of $n = 2$, the lane visibility is higher in the setting of $n = 1$ when $D_1 \geq 9m$. Therefore, the fake lane can be detected with different perturbation numbers. Even a single perturbation can work in physical world.

• **Rotation angle** θ**.** Fig.4.10(a) shows the influence of θ, when $n = 1$ and $n = 2$, respectively. In both scenarios, the lane visibility with $\theta = 15°$ and $\theta = 30°$ is obviously lower than the lane visibility with $\theta = 0$. Therefore, straight perturbations ($\theta = 0$) are more likely to be detected.

• **Light condition.** Fig.4.10(b) shows the influence of light condition, when $n = 1$ and $n = 2$, respectively. When $n = 1$, the lane visibility under bright and dark condition is similar in all values of D_1. When $n = 2$, the lane visibility under dark

condition is higher than that in the bright condition, when $D_1 \geq 7m$. Hence, the perturbations work in both bright and dark environments. Darker environments even makes the lane visibility higher (see $n = 2$ in Fig.4.10.(b)).

• **Longitudinal Distance D_1.** When $n = 1$, the lane visibility is higher when $5m \leq D_1 \leq 12m$. When $n = 2$, the lane visibility is higher when $5m \leq D_1 \leq 7m$. Therefore, the fake lane can be detected in a large range of D_1 (from 15m to 3m) if the perturbations are properly implemented (like $n = 1$, $\theta = 0$ in Fig.4.10.(a)). Closer distances ($D_1 \leq 9m$) makes the perturbation easier to be detected.

Answer: The crafted perturbations can be detected as fake lanes while staying imperceptible to humans. A demo video for physical attacks can be found at [33].

RQ6: Can we misguide the vehicle in physical world?

Motivation: The over-sensitivity of the target lane detection module has been demonstrated in both digital world and physical world through the answers to the previous RQs (i.e., RQ1, 2, 3, 4 for digital world and RQ5 for physical world). This RQ aims to investigate whether the control policy of the Autopilot will be affected by the crafted markings. Specifically, if Autopilot reacts to the fake lane, our attacks can impose a severe threat to the security and safety of the victim vehicle.

Approach: We find that in a commonly-seen crossroads scenario (i.e., the straight lanes disappear in front of the vehicle), the perturbations can mislead the vehicle to the oncoming traffic lane (illustrated in Fig.4.11). Specifically, in a crossroads scenario, we generate the perturbations that can trick the lane detection module to output an obvious lane. After physical deployment, we switch the vehicle to auto-steer mode and let it pass the crossroads where the markers have been added.

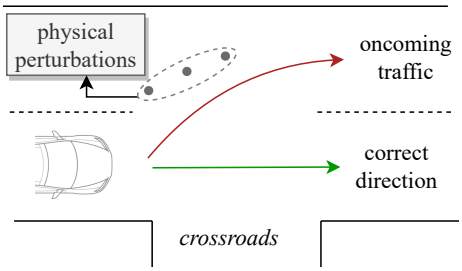
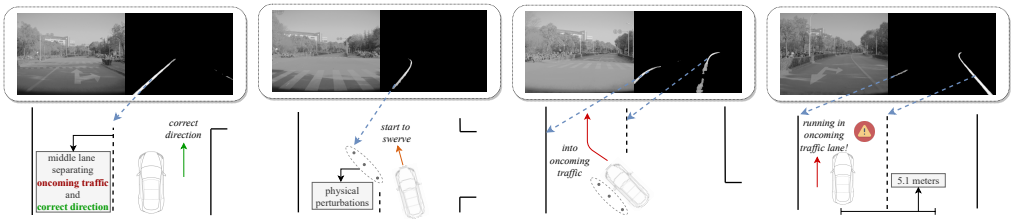


Figure 4.11: RQ6: Misguide the vehicle into the oncoming traffic in the crossroads scenario.



(a) Vehicle is running on the correct direction.

(b) Fake lane is detected and vehicle starts to swerve.

(c) Vehicle follows the fake lane into oncoming traffic.

(d) Vehicle finally runs in the oncoming traffic lane!

Figure 4.12: RQ6: The vehicle in auto-steer mode is misled into the oncoming traffic.

Results: We record the video showing the camera images and lane images when the vehicle is passing the crossroads. The result shows that the perturbations can lead to a fake lane which makes the vehicle swerve. Moreover, the vehicle was deviated by 5.1 meters (more than 2.5 times the width of the vehicle), and followed the fake lane to the oncoming traffic, demonstrating a severe and threat in real world.

Fig.4.12 illustrates the whole process. In each subfigure, the upper row includes the camera image and lane image, and the lower row shows what was happening when the frame was recorded. The interpretation of each frame is as below: Fig.4.12 (a). Before approaching the crossroads, the vehicle runs on the right-hand side (correct direction) of the road, and the middle lane, which separates the two directions, is correctly recognized as the left-hand side lane (shown in the lane

image).

Fig.4.12 (b). Right before the vehicle runs into the crossroads, the perturbations are detected and recognized as the fake lane, and therefore the vehicle starts to swerve along with the detected lane.

Fig.4.12 (c). The vehicle follows the fake lane and swerves to the left-hand side of the road (oncoming traffic lane). During this process, the middle lane (right lane in lane image) was recognized as the right-hand side lane. Based on this detection result, the vehicle runs into the oncoming traffic.

Fig.4.12 (d). Finally, the vehicle is deviated by 5.1 meters (more than 2.5 times the width of the vehicle), and is misled into the oncoming traffic lane, and further keeps running on this wrong direction.

Note that there is no human operation in the above process. The vehicle is in auto-steer mode, and its average speed is above 40km/h, which is already very dangerous in real world. Interested readers are referred to our demo video [33].

Answer: The experimental result shows that the fake lane resulted from the unobtrusive perturbations can successfully fool the vehicle in auto-steer mode to swerve, and even misguide the vehicle into oncoming traffic (might hit other cars in the oncoming traffic lane), thus demonstrating the potential severe threats in real world.

4.7 Discussion

4.7.1 Defense

Enhancing the lane detection module. The lane detection module can be improved to distinguish crafted perturbations by two ways: *(1) Detecting abnormal lane lines by features.* Since the attackers want to make the perturbations unobtrusive, the size of the perturbations for generating the fake lane should be much smaller than the normal lanes. Moreover, as the attackers want to mislead the vehicle to cause safety and/or security consequences, the detected fake lane will be inconsistent with the real lanes (e.g., generating sharp turns [87]). As a result, the lane detection module can leverage these features to reject the abnormal lanes in advance. *(2) Including adversarial examples in training data.* As suggested by Goodfellow et al. [46], adding adversarial examples in the training data can make the model more robust to adversarial attacks. Hence, images with perturbations can be included in the training data to help the lane detection module distinguish between crafted perturbations and real lane lines.

Enhancing the control policy. To make the control policy more robust is another option for defense: *(1) Taking into consideration other visual elements.* The vehicle is vulnerable to our attacks if the steering control policy just relies on the lane detection result. Hence, it can be enhanced by involving other visual elements (i.e., coming traffic, pedestrian) to assist the steering control. *(2) Multi-Sensor fusion.* In Tesla Autopilot, the lane detection module relies on visual data. A possible defense method is to adopt multi-sensor fusion. That is, the control policy should also take into account the information from sensors like LiDAR, Radar, sonar and GPS. For example, the data from GPS and Radar can be used to de-

tect whether the vehicle is deviated or running in the oncoming traffic lane. *(3) Advanced warning.* As the security of autonomous driving may not be fully guaranteed, the vehicle should warn the driver in advance when any abnormal lane line is detected (e.g., the size of the lane is too small or the angle of the lane is too sharp, etc.). Moreover, to ensure safety, the vehicle should demand the driver for manual control and quit auto-steer mode.

4.7.2 Limitations

Since our attacks exploit the over-sensitivity of the lane detection module to mislead the vehicle, the crafted perturbations need to be detected by the lane detection module and thus they cannot be completely invisible. Hence, the driver may notice them if she knows the attack and pays full attention to the ground. However, our attack still poses severe threats to current autonomous driving because of the following reasons. *First, drivers are likely to pay less attention in auto-steer mode.* Without being informed of our attack, the driver may simply ignore the perturbations, not to mention that the vehicle is in auto-steer mode. According to the statistics given by the surveys [105][129], distracted driving is the top-1 reason for car crashing. In auto-steer mode, drivers are likely to pay less attention so that they may not notice the small perturbations which are quite different from the real lane. *Second, there is not enough time for reaction.* Even if the driver notices the perturbations when the vehicle is going to the place where the crafted perturbations have been deployed, there may not be enough time for the driver to react. For instance, in the experiment for answering RQ6, the speed of the vehicle is around 40km/h, and thus it takes only 0.918 seconds to deviate the vehicle for

5.1m. M. Green [47] shows that the driver's reaction times for unexpected events are between 1.20s and 1.35s ($> 0.918s$ in our experiment). Therefore, there is not sufficient time for the driver to take action against our attack, and severe consequences might have already been caused.

Additionally, while our attack design follows the principles of adversarial attacks-aiming to find the smallest perturbation to achieve the best attack performance, our goal extends beyond deceiving the lane detection model. We also aim to manipulate the subsequent control module to follow the detected fake lane, potentially causing the ADS to deviate and pose safety risks. This was demonstrated in a real-world cross-road scenario, highlighting the broader safety implications of our attack.

Chapter 5

Testing ADS Controller

5.1 Overview

Ensuring the safety of autonomous driving systems has become essential, with a surge in research focusing on identifying safety violations. However, while many studies target violations within the planning module, there's a significant oversight concerning the control module. This oversight is crucial: even if a driving plan is correct, an erroneous control signal can deviate the vehicle off its intended trajectory. However, how to test the control module in Autonomous Driving System (ADS) remains a challenge due to the lack of concrete metrics and simulation scenarios. To address the above gaps, we first propose four novel metrics to evaluate the performance of the control module, and further enhance current scenario-based fuzzing methodology based on these metrics, which can efficiently generate corresponding scenarios for our testing. With the help of the proposed metrics and enhanced fuzzing approach, we conduct the first extensive evaluation on the advanced Model Predictive Controller (MPC) of the industrial-grade ADS

- Apollo. Surprisingly, experimental results revealed significant performance defects in Apollo's controller, rendering it unable to execute basic actions (e.g., making a complete turn). Further investigating the controller code, we identified 14 previously undiscovered bugs responsible for such inadequacies. All identified bugs were acknowledged, and most of them have been promptly addressed by the official team with our assistance.

In summary, the primary contributions of this work are as follows:

• Focusing on the evaluation of the control module in ADS, we introduce 4 innovative metrics to guide the assessment of the ADS controller. Additionally, we enhanced the existing scenario-based fuzzing framework using our proposed metrics.

• Employing the aforementioned metrics, we carried out an extensive evaluation of the advanced control module in the industrial-grade ADS system, Apollo. Our findings indicate that Apollo's controller struggles with basic maneuvers and exhibits deficiencies in aspects including tracking accuracy, responsiveness, stability, and smoothness.

• We proposed an semi-automatic bug analysis approach, assisted by the VLM integrated into a CoT process, which could effectively reason the specific bugs in the code, based on the bug behavior and controller code logic. Assisted by this approach, we discovered a total of 14 new bugs in Apollo's control module. All these bugs have been acknowledged by the official team and addressed [120].

5.2 Background

ADS Architecture. Autonomous Driving System (ADS) architecture can be divided into two main design philosophies: end-to-end design and modular design [48, 22]. In the end-to-end design, raw sensor data such as camera feeds and LiDAR scans are directly processed by a deep neural network, which then outputs the control actions like steering, braking, and acceleration without the need for intermediate steps. This method focuses on a streamlined system to manage various aspects of driving but often lacks in interpretability and flexibility. In contrast, modular design breaks down the complex task of driving into distinct modules such as perception, planning and control. Each module serves a specific function: perception interprets sensor data to understand the environment, planning makes decisions based on that understanding, and control executes these decisions in the form of vehicle maneuvers. Although end-to-end ADS has great potential, the modular design is still the choice in current industrial-level ADS solutions, such as Apollo [2], Openpilot [95], and Autoware [8], primarily due to its greater interpretability, which allows for easier debugging and validation.

ADS Workflow: Planning to Control. In the modular ADS architecture, the *planning* and *control* modules work in close collaboration to ensure smooth and safe vehicle operation. The planning module is responsible for generating a future trajectory that the vehicle should follow, taking into account various factors like road conditions, obstacles, and traffic rules. Once this trajectory is planned, the control module will calculate the optimal control commands needed to adhere to this trajectory, using control algorithms including Proportional-Integral-Derivative (PID) controllers or Model Predictive Control (MPC). While much of

the previous research efforts [144, 67, 74, 79, 132] have been geared toward optimizing the quality of the planned trajectory, it is equally crucial to ensure that the control module is capable of accurately following this path.

Scenario-based Testing. There is a series of related works focus on exploring the scenarios that will make the autonomous driving system go wrong [73, 67, 131, 133, 143, 156, 54, 116]. These previous works share a similar processing of using *fuzzing* to find the violations. Specifically, starting from some basic scenarios, the system will mutate them from various settings (e.g., the trajectory of NPC vehicle), and calculate the fitness score of the mutated scenarios (e.g, the distance to collision), then select the scenarios with higher fitness scores for the next-round mutation. However, almost all previous works suffer from one major limitation: they did not included the *control module*, which is the key module to ensure the vehicle can follow the planned trajectory. Particularly, previous works only evaluate the correctness of the planned trajectory [73, 131, 133, 143, 156, 54, 116], without considering whether the subsequent control module can follow the trajectory. DriveFuzz [67] and Acero [116] are the only two works that involved the control module (i.e., sending the throttle and steering command to the vehicle instead of simply teleporting vehicles to the planned points). However, they still failed to evaluate the inconsistency between the planning and control module. In conclusion, none of previous works investigated the performance of the essential *control module*, and thus it is unknown whether the control module can follow the planned trajectory in a satisfying manner.

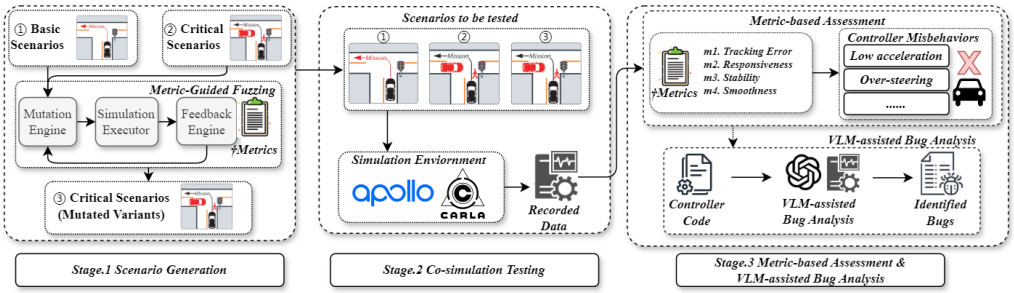


Figure 5.1: Workflow overview.

5.3 Approach

We first present the overview of our approach in §5.3.1, and then introduce the detailed metrics to assess the control module in §5.3.2. Finally we present our improved fuzzing framework in §5.3.3.

5.3.1 Approach Overview

Fig.5.1 provides an overview of our workflow, categorized into three distinct stages:

Stage 1: Scenario Generation (§5.4). The primary objective of this stage is to outline specific scenarios for co-simulation testing. These scenarios are designed to assess the efficacy of the *control module* under test. We propose two classifications of scenarios:

- *Basic Scenarios (①)*: In these scenarios, a vehicle moves between two points without encountering any external disruptions like other vehicles or pedestrians. Throughout this journey, the planning module continually furnishes the control module with a planned trajectory. Subsequently, the control module computes the necessary commands to follow this trajectory. These scenarios act as a standard metric to gauge the control module's performance.

- *Critical Scenarios (②③)*: These scenarios are particularly generated to

challenge the control module in intricate and potentially risky situations. For instance, these could encompass unpredictable vehicular behaviors.

We initiate by constructing the basic scenarios ①, which encompass various objectives (§5.4.1). Subsequently, we incorporate critical scenarios ② from prior studies [73, 131, 133, 143, 156, 54]. Moreover, to enrich our database of critical scenarios, we applied our new metrics to the feedback in current SOTA fuzzing framework - Drivefuzz [67]. The combination of the basic and previously obtained critical scenarios, ①②, act as initial seeds for the fuzzing mechanism. This process then generates modified variants, resulting in additional critical scenarios ③. Finally, the combined set of scenarios ①, ②, and ③ serve as comprehensive test cases for evaluating the control module.

Stage 2: Co-simulation Testing (§5.5). At this stage, the generated scenarios are executed within a co-simulation environment. The simulator (e.g., Carla), continually transmits localization data (e.g., vehicle position and velocity) to the autonomous driving system (e.g., Apollo). In response, the autonomous driving system computes the control command which is then relayed back to the simulator for execution. Throughout each scenario's run, two trajectories are documented: the planned trajectory and the actual trajectory. These recorded trajectories subsequently act as reference points for a detailed evaluation of the control module.

Stage 3: Metric-based Assessment (§5.5) and VLM-assisted Bug Analysis (§5.6). During this stage, an extensive evaluation of the documented trajectories is first undertaken based on the proposed metrics. This process pinpoints specific shortcomings of the control module (i.e., *how* the controller performs poorly). After that, we conduct the VLM-assisted bug analysis on the controller code, leading to the identification of 14 new bugs in the industrial-grade autonomous driving

system, Apollo 8.0 [2], answering *why* the controller performs poorly (detailed in §5.6).

5.3.2 Testing Metrics

Consider a planned trajectory $T_p = \{P_{p1}, \ldots, P_{pn}\}$ alongside an actual trajectory $T_a = \{P_{a1}, \ldots, P_{am}\}$. Each point in these trajectories, whether P_{px} or P_{ax}, includes attributes such as the timestamp (t), velocity (v), positional coordinates (x, y), acceleration (a), and more. This can be exemplified as $P_{px} = \{t_{px}, x_{px}, y_{px}, v_{px}, a_{px}, ...\}$ and $P_{ax} = \{t_{ax}, x_{ax}, y_{ax}, v_{ax}, a_{ax}, ...\}$. Drawing insights from control theory and autonomous driving contexts, we propose the following four *metrics* specifically for the ADS control module.

- m1. Tracking Error (TE). A primary objective of the control module is to accurately trace the planned trajectory. Hence, an ideal performance would indicate the vehicle's actual trajectory perfectly mirroring the planned trajectory. This premise leads to an essential metric: evaluating the Tracking Error between the planned and the resultant actual trajectory:

$$TE = Distance(T_p, T_a) \tag{5.1}$$

Here, $Distance()$ embodies a generalized function indicating the variances between two trajectories. Importantly, this error reflects not just the spatial deviations but also error in richer attributes such as velocity, acceleration, heading angle, and so on. A smaller TE value implies a better performance of the control module.

- m2. Responsiveness (RE). The *Responsiveness* of a control system denotes

how promptly the controller steers the system to its intended state. In the realm of control theory, the term *Transient Response* [124, 93] embodies the behavior of a control system from its activation (or upon receiving a disturbance) up to the point it stabilizes in its steady-state. A cornerstone metric within transient response is the *settling time* [124, 93]. This metric gauges the time span the system output takes to remain within a predefined margin of its steady-state value. In the context of autonomous driving, to calculate the settling time, we pick a timestamp $t_{p,i}$ from the planned trajectory T_p, correlating to a distinct trajectory point $P_{p,i}$. A corresponding point $P_{a,j}$ within the actual trajectory T_a matching $P_{p,i}$ in attributes like position is then identified. The timestamp associated with this congruent point in T_a is $t_{a,j}$. The settling time t_s is derived from the disparity between these two timestamps:

$$t_s = t_{p,i} - t_{a,j} \tag{5.2}$$

The deduced t_s indicates the *temporal* gap between the *planned* moment of reaching a particular state and the *actual* moment when the vehicle attains this state. Specifically, a positive t_s indicates a lag, with the control module requiring an extra t_s duration to attain the intended state (under-controlled). Conversely, a negative t_s suggests the control module achieved the state t_s time units ahead (over-controlled). Therefore, the closer t_s approximates zero, the more responsive the controller proves to be.

- m3. Stability (ST). The *Stability* of a control system when influenced by control inputs is often assessed by the Input-to-State Stability (ISS) [117, 81]. Formally, a system, steered by the state equation $\dot{x} = f(x, u)$, is deemed Input-

to-State Stable when, under a confined input $u(t)$, the state $x(t)$ remains bounded for all instances t, considering every initial condition $x(0)$. Evaluating the ISS of a nonlinear system involves finding a Lyapunov function V that quantifies the *energy* disparity between the actual state and the intended state. Particularly, the rate of change of this energy should be confined within specific bounds. Suppose P_{p_t} and P_{a_t} represent points from the planned and actual trajectories at a coinciding timestamp t. The Lyapunov function can be construed as the state difference between these trajectory points:

$$V(t) = dis(P_{p_t}, P_{a_t}) \tag{5.3}$$

Here, $dis()$ is a generalized function quantifying the state difference, such as positional distance, between two trajectory markers and yields a scalar output. Consequently, to guarantee ISS stability, the *derivative* of this Lyapunov function must not exceed a defined threshold V_T:

$$\dot{V} \leq V_T \tag{5.4}$$

- m4. Smoothness (SM). To quantify smoothness, we focus on *acceleration*, a direct reflection of the forces felt by the vehicle's occupants. The smoothness includes both *linear acceleration* and *angular acceleration*. Linear acceleration, denoted as a, represents the rate at which velocity (v) changes over time, and is captured by the equation $a = \frac{dv}{dt}$. Conversely, angular acceleration, represented by α, measures the rate of change in angular velocity (ω) relative to time, formulated as $\alpha = \frac{d\omega}{dt}$. Together, these metrics offer a holistic perspective on the dynamics of the vehicle's motion, enabling a nuanced assessment of how the control module

influences ride smoothness.

5.3.3 Improved Fuzzing Framework

As presented in Tab.2.2, Drivefuzz [67] stands out as the most relevant work because it incorporates the control module into the fuzzing process. Specifically, Drivefuzz quantifies the *driving quality* of each iteration using a fitness score. Actions, such as *hard acceleration* and *hard turn*, contribute to a *decreased* driving quality score. However, Drivefuzz suffers from the following two limitations: First, despite involving the control module during testing, it fails to exam the non-trival error between the planned and actual trajectories, and thus failed to point out specific inadequacies of the control module. Second, Drivefuzz *randomly* generates NPC vehicles and pedestrians, neglecting their spatial relation to the ego vehicle (the vehicle under test). Consequently, numerous redundant scenarios occur (e.g., when the NPC vehicle is distantly situated and remains irrelevant to the ego vehicle's performance), which decreases testing efficiency. To address these challenges, we have refined Drivefuzz in the following two ways:

Enhanced fitness score. We augmented the fitness score computation by integrating the metrics presented in §5.3.2. The refined score S is defined as:

$$S = DQ + \theta\_diff(T_p, T_a) + Dis\_diff(T_p, T_a) + Max_a + Var_a \tag{5.5}$$

Here, DQ denotes the *Driving Quality* score, which is inherited from Drivefuzz's original scoring system. Particularly, DQ counts the number of bad-control actions including hard acceleration, hard braking, hard turns, etc. However, this score failed to involve the *inconsistency* between the planned and actual trajectory.

Accordingly, we construct a more comprehensive score by adding DQ with four additional metrics. $\theta\_diff(T_p, T_a)$ and $Dis\_diff(T_p, T_a)$ represent the mean error between (planned heading angle and actual heading angle) and between (planned and actual positions), respectively. Max_a and Var_a define the peak acceleration (incorporating both angular and linear accelerations) and acceleration variances. Specifically, $\theta\_diff(T_p, T_a)$ and $Dis\_diff(T_p, T_a)$ encapsulate the metrics m1, m2, and m3, while Max_a and Var_a resonate with m4, as a supplement to the original DQ.

Enhanced Mutation Strategy. We found that Drivefuzz always generate NPCs *at random positions across the entire map*. As a result, many of the generated NPCs are too far to interact with the tested ADS vehicle, and thus making this scenario useless. As a subsequent improvement, we no longer generate NPCs arbitrarily across the entire map. Instead, we focus on creating and modifying NPCs *within a certain range around the ego vehicle*. To implement this, we define a threshold to specify the maximum distance dis_{max} permissible between the NPC's spawn point and the ego vehicle. This constraint ensures that NPCs only appear within this confined region. Leveraging this approach significantly reduces redundant scenarios and increases the number of critical scenarios (e.g., collisions).

Summary. Our improved fuzzing framework is summarized in Fig.5.2. First, we restrict the range of the generated NPCs within the threshold of dis_{max}, in which case NPCs will have more chances to interact with ego vehicle, and thus critical scenarios can be found more effectively. Second, we construct fitness score with our new metrics, as detailed in Equation.5.5. This new fitness score considers metrics including tracking error, responsiveness, stability and smooth-

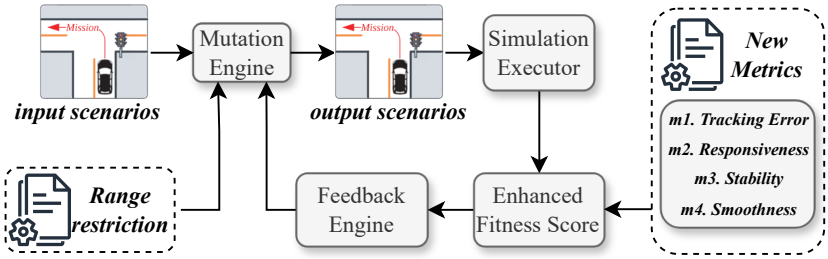


Figure 5.2: Improved fuzzing framework: We 1). restrict the range of generated NPCs and 2). use new metrics to construct a more comprehensive fitness score.

ness, providing a more comprehensive metric than Drivefuzz [67].

Remark about our contributions. Our fuzzing framework is designed to generate effective test scenarios for the control module, diverging from the aim of previous works [73, 131, 133, 143, 156, 54, 116] that focused on uncovering diverse violations. We enhanced Drivefuzz by enhancing its fitness score and mutation strategy, making it suitable for our objective of creating scenarios that rigorously test control performance. Our modified framework not only meets this goal but also demonstrates superior performance to Drivefuzz (in §5.4.3). Our main contribution lies in the new metric and testing methodologies for the control module, with extensive bug analysis and the identification of practical bugs in Apollo (in §5.6). These advancements are facilitated by our refined fuzzing technique, which lays the groundwork for the aforementioned contributions.

5.4 Scenario Generation

In this section, we detail how the basic scenarios and critical scenarios were derived in §5.4.1 and §5.4.2, respectively. These scenarios will be used in further co-simulation testings to evaluate the performance of the control module, based on the proposed metrics.

5.4.1 Basic Scenarios.

Basic scenarios aim to test the basic functionality of the control module, and there is no external interrupt (e.g., NPC vehicles and pedestrians). We constructed a total of 30 basic scenarios, segregated into 5 classes: B1. Straight driving (driving straight and then stopping), B2. Sharp left turn (Taking a 90-degree left turn and then stopping), B3. Sharp right turn (Taking a 90-degree right turn and then stopping), B4. Soft left turn (Taking a soft left turn and then stopping), B5. Soft right turn (Taking a soft right turn and then stopping). In each scenario, the autonomous vehicle initiates from a standstill (0 speed). As the simulation proceeds, the planning module continuously generates the requisite trajectory. Concurrently, the control module derives the control commands to adhere to this generated trajectory. The above scenarios will be used (in §5.5.1) to assess the control module's capability to finish the basic maneuvers (① in Fig.5.1).

5.4.2 Critical Scenarios.

Collected Critical Scenarios. We collected the critical scenarios from previous works as part of the input seed and tested scenarios. Although all previous works [73, 67, 131, 133, 143, 156, 54, 116] claimed to have discovered various types of violation, there are overlap in their results. Particularly, the previously discovered critical scenarios can be divided into the following five types: (1). stopping due to DoS vulnerability (from [143, 67]), (2), crash on static objects (from [67]), (3) collision while current lane is being invaded (from [74, 131, 133, 54]), (4) collision while invading other lanes (from [74, 131, 133, 54]) and (5) collision in complex crossroad scenarios (from [131, 156]). For each type of the above

violation, we have constructed 5 variations with different relative NPC locations, leading to totally 25 critical scenarios as the input seed (i.e., 25 critical scenarios served as ② in Fig.5.1).

Mutated Critical Scenarios. For each type of the basic scenarios (①), we select one scenario as the feed to our fuzzing framework; for each type of the collected critical scenarios (②), we put all 5 variations into the fuzzing engine. Starting from basic scenarios (①) in which no NPC is introduced, the fuzzing framework will randomly generate and mutate NPCs. Starting from critical scenarios (②) in which NPCs already exist, the fuzzing framework will mutate the positions (including the starting position and destination position) of the NPCs based on the original scenario. Finally, for each type of the input seed (five types from ① and five from ②), we select the top-5 scenarios that have the highest fitness scores (indicating the worst controlling performance). As a result, there are totally 50 scenarios in ③, in which 25 are mutated from ① and 25 from ②. That is to say, there are 50 *mutated* critical scenarios served as ③ in Fig.5.1.

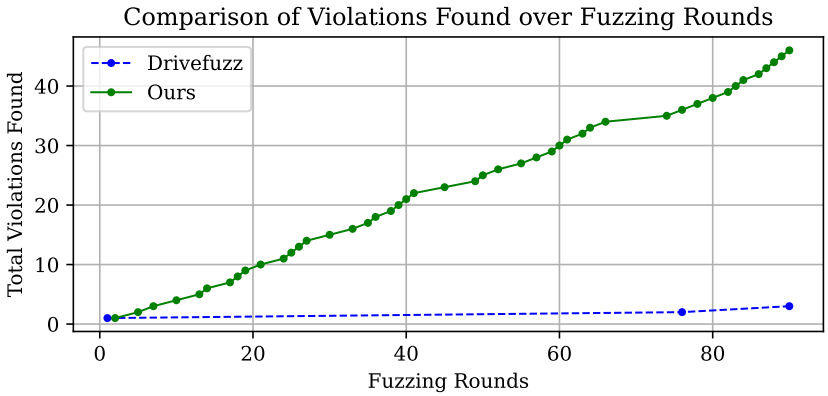
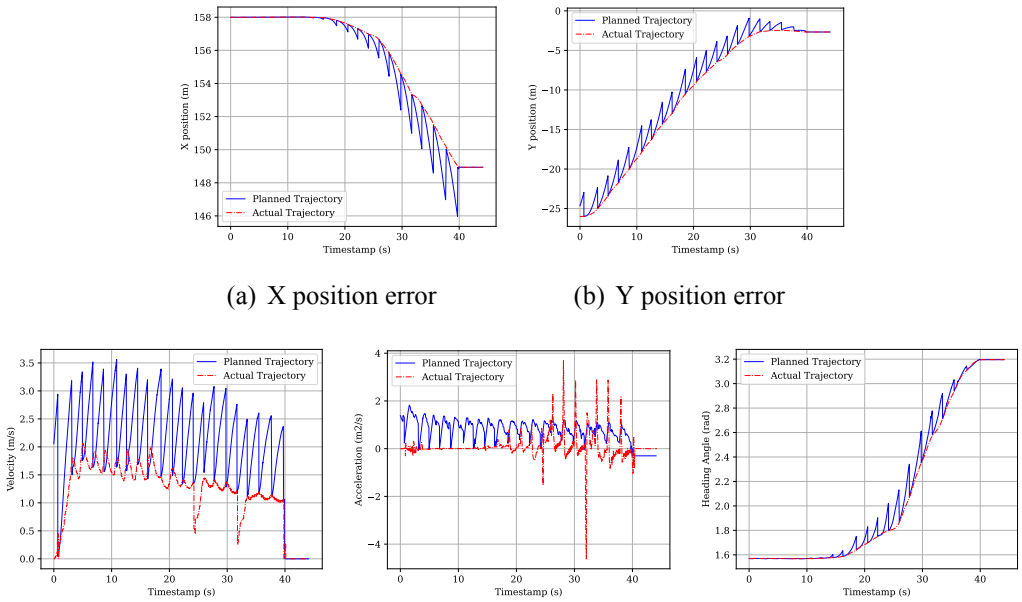


Figure 5.3: Comparison with Drivefuzz [67]: Our improved fuzzing framework can find violations more efficiently (46 violations vs 3 after 100 rounds).



(a) X position error (b) Y position error

(c) Velocity error (d) Acceleration error (e) Heading angle error

Figure 5.4: *B2_03. Sharp left turn*: Visualization of the Tracking Error (Error): X-axis is the timestamp and Y-axis is the state value, including position (subfigure (a) and (b)), velocity (c), acceleration (d) and heading angle (e).

5.4.3 Comparison with Drivefuzz

Fig.5.3 shows the comparison between our improved approach and Drivefuzz [67]. Specifically, a critical scenario in crossroad is selected as the initial seed. The X-axis represents the mutation rounds, and the Y-axis represents the total number of violations (i.e., whether the ego vehicle crashes on other objects) found. The value of dis_{max} is set to 30 meters. During a 100 rounds of fuzzing, our approach found 46 safety violations while Drivefuzz found only three. Due to the improved fitness score calculation and improved mutation strategy (in §5.3.3), our framework can find violation much more effectively than the state-of-the-art Drivefuzz [67].

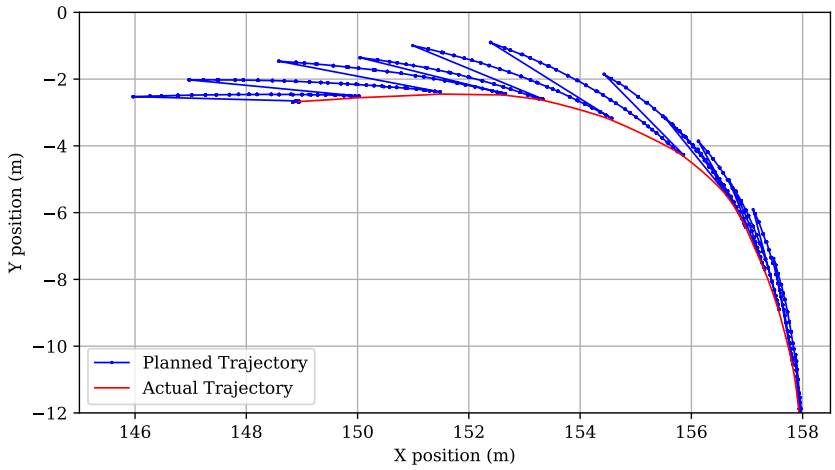


Figure 5.5: *B2_03. Sharp left turn*: Comparison between the planned trajectory and the actual trajectory. Obvious deviation can be observed from two trajectories.

Table 5.1: **m1: Tracking Error between the planned trajectory and actual trajectory.** Each row represents the average error value for one type of basic scenario. $e_{p,max}$: maximum position error; $\bar{e}_p$: average position error; $e_{\theta,max}$: maximum θ error; $\bar{e}_\theta$: average θ error; $e_{v,max}$: maximum velocity error; $\bar{e}_v$: average velocity error; $e_{a,max}$: maximum acceleration error; $\bar{e}_a$: average acceleration error. The percentage after $e_{v,max}$, $\bar{e}_v$, $e_{a,max}$, $\bar{e}_a$ show the percentage of how much this error is based on the average state value.

	Position Error (m)		θ Error ($degree$)		v Error (m/s)		a Error (m/s^2)	
	$e_{p,max}$	$\bar{e}_p$	$e_{\theta,max}$	$\bar{e}_\theta$	$e_{v,max}$	$\bar{e}_v$	$e_{a,max}$	$\bar{e}_a$
B1_mean	3.46	1.14	1.62	0.14	(2.01, 144.64%)	(0.84, 61.09%)	(3.21, 2283761.40%)	(0.92, 1202970.00%)
B2_mean	3.14	0.98	10.56	0.96	(2.44, 197.96%)	(0.86, 70.67%)	(3.18, 10197.51%)	(0.83, 2746.92%)
B3_mean	3.21	1.05	17.60	2.02	(2.60, 288.06%)	(0.86, 92.33%)	(8.85, 88615.00%)	(0.98, 36052.36%)
B4_mean	2.99	1.14	3.43	0.48	(2.79, 177.14%)	(0.80, 50.98%)	(3.18, 6480.07%)	(1.12, 2322.88%)
B5_mean	2.84	1.08	2.65	0.52	(2.78, 182.83%)	(0.80, 52.77%)	(2.87, 5610.02%)	(1.10, 2076.94%)
Overall_mean	**3.13**	**1.08**	**7.17**	**0.82**	**(2.52, 198.13%)**	**(0.83, 65.57%)**	**(4.26, 478932.80%)**	**(0.99, 249233.82%)**

5.5 Scenario Assessment

In this section, we perform extensive analysis on the control module of Apollo, using the metrics proposed in §5.3 generated scenarios from §5.4. Specifically, we use the 30 basic scenarios to assess the basic capabilities of the control module in §5.5.1. We then use the rest 75 critical scenarios to evaluate whether the controller can finish the task in more challenging settings in §5.5.2, and summarize our findings in §5.5.3. The experiments were conducted under Apollo 8.0

equipped with the advanced Model Predictive Control (MPC [81]) and Carla simulator 0.9.14. The experiments were conducted on a computer which is capable of running co-simulation effectively, with all Apollo modules operating at the recommended frame rate (planning over 10Hz and controlling over 100Hz).

5.5.1 Basic Scenarios.

We investigate the detailed performance of the autonomous driving system (i.e., Apollo 8.0 in our context) under the basic scenarios in §5.4.1.

- Completeness. Before delving into the detail metrics to evaluate the control module, a basic requirement - the *completeness* of the task should be assessed (i.e., whether the vehicle can reach the destination point). Specifically, if the ego vehicle reaches the destination without hitting any other objects (e.g, NPC vehicle or road curb), we deem such a scenario as successful, otherwise (e.g., collision happened or failed to reach destination) we think the control module failed to finish the execution of this scenario. When the distance between the actual destination and the planned destination is smaller than 1m, we think the task is completed. However, out of the 5 types of basic scenarios in §5.4.1, only the *B2* succeeded to complete the tasks, resulting a very low completeness rate of $\frac{6}{30} = 20\%$. In all other basic scenarios involving making a turn, the vehicle will stop halfway and cannot reach the destination (Bug#16, #17 and #21 in Tab.5.5).

- m1. Tracking Error (TE). We first introduce how TE on each state is calculated with a specific scenario, and then present the comprehensive results on all basic scenarios. Fig.5.4 shows the comparison of these states between the *planned trajectory* and *actual trajectory*, under one basic scenario under *B2*. *Sharp left*

turn. For the proposed 5 types of basic scenarios, the tracking error is shown in Tab.5.1. In perfect situation, all data in Tab.5.1 is *zero*, in which case the control module perfectly follow the planned trajectory. However, according to Tab.5.1: (1). *position error:* the actual trajectory and planned trajectory have a maximum of *3.13m* error and an average of *1.08m* error, which is neglectable; (2). θ: the maximum error on heading angle θ reaches over 17.6° under and B3, indicating that the control module failed to steer the vehicle to the planned θ; (3). v: based on the average actual velocity, the maximum and average error on velocity reaches 198.13% and 65.57%, indicating the control module cannot drive the vehicle to the planned velocity; (4). a: huge relative error is identified on a, indicating the control module cannot accelerate the vehicle as planned (the percentage is huge because the average actual acceleration is often very small).

Summary on m1: *For all basic scenarios, neglectable tracking error is identified in all state values including position, heading angle, velocity and acceleration (as indicated by Fig.5.4 and Tab.5.1). The tested control module CANNOT accurately follow the planned trajectory, and even failed to reach the destination under most basic scenarios.*

- m2. Responsiveness (RE). We calculate the settling time t_s according to Equation.5.2 to assess the responsiveness. Additionally, some planned state were never reached during the whole routine ($t_s = \infty$), in which case the corresponding $t_{a,j}$ of the $t_{p,i}$ cannot be found. Such a case is marked as a failure, and the percentage of such failure is noted as FR (Failure Rate). Specifically, the good responsiveness is represented by a small FR and a t_s that is close to 0. Tab.5.2 shows the result of the responsiveness assessment for all 5 types of basic scenarios. Particularly, we do not evaluate the responsiveness of θ under B1, because all

scenarios under B1 is *straight driving*, in which case the θ difference is very small (can be observed in Tab.5.1) and calculating t_s on such small θ will lead to great error which cannot represent the actual responsiveness. Delving into Tab.5.2, comparing the planned position and the actual position, there are on average 10.72% planned positions were never reached; moreover, an average t_s of 1.22s is needed to reach the planned position. For the heading angle θ, high FR is observed in B2 (10.79%) and B3 (17.28%), indicating the unsatisfying responsiveness of the steering control. Particularly, very poor responsiveness is observed in v and a: for v and a, there are 65.50% and 46.58% planned state was never reached, and for those reached states, the average time delay is 3.20s and 10.48s, respectively.

Summary on m2: *The tested control module CANNOT promptly drive the vehicle into the planned states. Particularly, significant delay is observed on v (3.20s) and a (10.48s), indicating the control module cannot produce prompt acceleration.*

Table 5.2: **m2: Responsiveness assessment of basic scenarios.** Each row represents the average value for one type of basic scenario. For each of the four states (i.e., position, θ, v and a), FR is the Failure Rate presenting how many planned states were never reached; $\bar{t}_s$ is the average settling time, and $t_{s,max}$ is the maximum settling time.

	Position			θ			v			a		
	FR	$\bar{t}_s$	$t_{s,max}$	FR	$\bar{t}_s$	$t_{s,max}$	FR	$\bar{t}_s$	$t_{s,max}$	FR	$\bar{t}_s$	$t_{s,max}$
B1_mean	5.24%	1.57s	8.63s	\	\	\	70.54%	2.89s	44.98s	63.44%	12.44s	70.51s
B2_mean	12.81%	0.78s	3.81s	10.79%	0.53s	9.75s	69.88%	2.15s	20.76s	52.17%	6.08s	29.19s
B3_mean	21.83%	1.18s	6.24s	17.28%	1.01s	14.24s	69.72%	2.35s	27.40s	37.34%	8.41s	37.34s
B4_mean	5.70%	1.36s	7.59s	1.93%	0.33s	9.09s	56.68%	5.51s	86.29s	38.26%	10.96s	84.29s
B5_mean	8.02%	1.23s	6.69s	1.79%	0.35s	4.88s	60.67%	3.12s	46.74s	41.67%	14.50s	61.97s
Overall_mean	**10.72%**	**1.22s**	**6.59s**	**7.95%**	**0.56s**	**9.49s**	**65.50%**	**3.20s**	**45.23s**	**46.58%**	**10.48s**	**56.66s**

- m3. Stability (ST). As stated in §5.3.2, the Lyapunov function is defined as the absolute value of the difference between the planned state and actual state at each timestamp t. We calculate the Lyapunov function value of all 4 states, and further calculate its derivatives to evaluate the stability of the system, as indicated

Table 5.3: **m3: Stability assessment based on the derivatives of the Lyapunov function value.** $\dot{V}_{max}$: Maximum derivative of V; R: $\frac{\dot{V}_{max}}{V_T}$.

	Position (m/s)		θ $(degree/s)$		v (m/s^2)		a (m/s^3)	
	$\dot{V}_{max}$	R	$\dot{V}_{max}$	R	$\dot{V}_{max}$	R	$\dot{V}_{max}$	R
B1	3.37	421.25%	\	\	2.48	477.31%	21.32	2479.3%
B2	3.05	169.28%	19.41	2695.69%	1.95	295.76%	32.45	11589.29%
B3	3.24	539.67%	34.77	7901.36%	3.69	635.69%	84.81	10872.95%
B4	3.82	329.14%	4.84	931.35%	2.55	155.55%	21.15	7051.0%
B5	3.36	336.3%	3.50	603.45%	2.22	205.46%	21.91	10954.5%
Mean	**3.36**	**359.13%**	**15.63**	**4597.06%**	**2.58**	**390.61%**	**36.33**	**7568.54%**

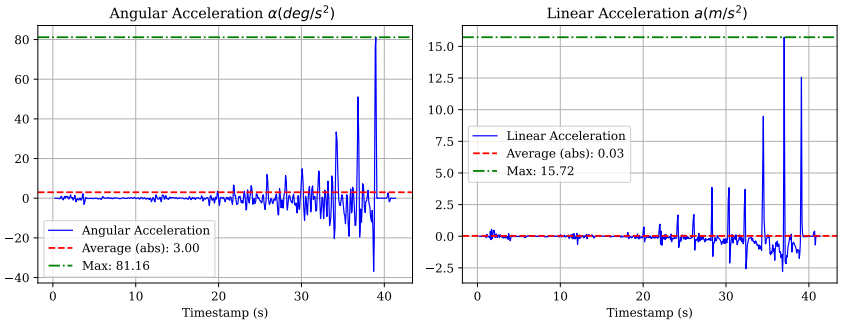


Figure 5.6: *B3_03. Sharp right turn*: **Smoothness** evaluation of a specific scenario (B3_03). Abrupt changes were observed for both α and a, and the maximum acceleration can be over 500 times larger than the average value.

by Equation.5.3 and 5.4. Without losing generality, we establish the threshold at 20 times the average statistical value of the data (i.e., $V_T = 20 \times \dot{V}$). Tab.5.3 shows the average and maximum values of $\dot{V}$. The $\dot{V}$ of θ under B1 is not taken into account due to the same reason elaborated in ***B1***. $R = \frac{\dot{V}_{max}}{V_T}$ represents the ratio between the maximum derivative and the threshold, and $R > 1$ means that the system is observed to be NOT stable. According to Tab.5.3, significantly large $\dot{V}_{max}$ is observed under all 4 states, with the average R being much larger than 100% (359.13%, 4597.06%, 390.61%, 7568.54% respectively).

Summary on m3: *The tested control module is poor in stability: it CANNOT maintain the vehicle state around the planned state in a stable manner. The value of the defined Lyapunov function value $\dot{V}$ (representing stability) can be over 100 times greater than the set threshold, violating the requirement of Equation.5.4.*

- m4. Smoothness (SM). As stated in §5.3.2, we use both the angular acceleration $\alpha = \frac{d\omega}{dt}$ and linear acceleration $a = \frac{dv}{dt}$ to evaluate the smoothness. Particularly, as there is no *golden metric* to determine how large value of acceleration can be identified as *unsmooth*, we record absolute accelerations during the execution of the whole scenario, and calculate the ratio between the maximum acceleration and average acceleration. Fig.5.6 shows the two accelerations of a specific basic scenario B3_03, in which we can derive two intuitive facts: (1). the trend of both α and a is very unsmooth, with multiple intensive peaks and falls; (2). the recorded maximum acceleration can be hundreds times larger than the average value. Moreover, such patterns were observed not only on B3_03 but also on all other scenarios. Results of quantitative analysis is shown in Tab.5.4, in which $R_\alpha = \frac{\alpha_{max}}{\bar{\alpha}}$ and $R_a = \frac{a_{max}}{\bar{a}}$ represent how big the observed maximum acceleration is compared with the average acceleration. It is observed that the maximum acceleration can be dozens of times larger than the average acceleration, indicating that the control system failed to meet the requirement of being *smooth*.

Summary on m4: *The tested control module CANNOT output smooth control commands: frequent abrupt changes were identified for both angular and linear acceleration.*

Table 5.4: **m4: Smoothness assessment based on the absolute angular acceleration and linear acceleration.** $\alpha_{abs} = |\frac{d^2\theta}{dt^2}|$: Angular acceleration ($deg/s^2$); $a_{abs} = |\frac{dv}{dt}|$: Linear acceleration ($m/s^2$). R_α, R_a: ratio between the maximum value and average value. For simplicity, the suffix abs is omitted in the Table.

	$\bar{\alpha}$	α_{max}	R_α	$\bar{a}$	a_{max}	R_a
B1_mean	0.24	4.68	2062.63%	0.002460	1.86	4645842.97%
B2_mean	1.38	19.89	1566.03%	0.036704	1.69	4913.93%
B3_mean	1.58	38.45	2161.37%	0.069051	8.75	38509.53%
B4_mean	0.92	8.62	921.28%	0.051861	2.88	6262.35%
B5_mean	0.98	11.88	1080.51%	0.055409	2.48	4822.78%
Overall	**1.02**	**16.70**	**1558.36%**	**0.043097**	**3.53**	**940070.31%**

- Summary on basic scenarios. Unfortunately, the tested control module (MPC controller in Apollo 8.0) failed to complete the basic maneuvers under all four metrics. The controller cannot drive the vehicle accurately to the planned trajectory (m1), and the responses are laggy (m2); it cannot maintain the state around the desired stable state (m3), and failed to produce smooth control commands (m4).

5.5.2 Critical Scenarios

As stated in §5.4.2, we derived totally 75 critical scenarios, and will further assess the performance control module under these scenarios in which various NPCs were involved. For these critical scenarios, we do not repeat the analysis on all metrics as they have been thoroughly analyzed on basic scenarios in §5.5.1. Instead, we focus on assessing the *completeness* of the critical scenarios (i.e., checking whether the ego vehicle can reach destination without collision).

After running all 75 critical scenarios, there are only 10 scenarios in which Apollo completed the task, leading to a success rate of only 13.33%. For the other 65 failed scenarios, there are 61 in which the vehicle stopped permanently due to the internal bugs in planning and control module (will be detailed in §5.6), and in the left 4 scenarios the ego vehicle collided with other objects.

5.5.3 Summary on Assessment

Overall, the actual performance of the tested control module (MPC controller in Apollo) has very unsatisfying performance in all four proposed metrics. For all 105 tested scenarios (30 basic and 75 critical), the overall success rate is only

15.24% (6 success in basic ones and 10 success in critical). Moreover, based on our metric-based assessment in §5.5, the tested control module was shown to have bad performance in all involved metrics, and could not even complete the basic maneuvers (e.g., making a 90-degree turn).

5.6 VLM-assisted Bug Analysis

Based on the proposed metrics, we have identified specific deficiencies in the tested control module in §5.5 (i.e., *how* the controller is deficient). However, the reason *why* the controller performs poorly remains unknown. In this section, we propose a VLM-assisted CoT bug analysis to determine the practical bugs in the Apollo controller codebase (answering *why*), based on the bug behaviors observed in the previous section (from *how*).

Motivation: why LLM? Unlike traditional software bugs (e.g., a specific crash *deterministically* corresponds to certain lines of code), the correspondence between controller under-performance and the code-level root cause can be highly *non-deterministic*. Specifically, one type of bug behavior could be due to various root causes, including ill-tuned parameters, faulty configurations, improper code logic, etc. As a result, traditional tools cannot determine the correspondence due to this complex mapping relation. In such cases, Large Language Models (LLMs) become a feasible assistant due to their (1) extensive knowledge base (having consumed various data) and (2) strong flexibility in output when dealing with different input prompts. However, applying LLMs to our bug analysis, particularly for the controller code, presents several challenges.

Challenges. Before we can use LLMs to assist in our bug analysis, we must

address the following three challenges:

- C1. Understanding bug behavior. As previously stated, one type of bug behavior (e.g., large m1.tracking error) could be due to various reasons in the buggy code. Assuming an LLM could help reason such correlations, we must first prompt the LLM to *understand* the particular bug behavior, which is the first challenge. Directly feeding the recorded trajectory data (planned and actual) to the LLM is impractical for two reasons. First, the planned and actual trajectories are recorded as discrete points, each marked with its states, including timestamp, x-axis and y-axis position, heading angle θ, angular acceleration α, velocity, and linear acceleration, totaling 7 states. As a result, LLMs will struggle to deal with such complex data, thereby failing to extract the *bug behavior* within. Second, even if LLMs could consume these data, token limitation would be another challenge. For example, in the scenario of Fig.5.4, there are a total of 8,856 trajectory points (each labeled with 7 states) recorded over 42 seconds. Directly feeding such a large amount of data to an LLM would not only degrade its performance (it is known that the longer the input text, the more difficult it is for LLMs to extract key insights) but also likely reach the token limitation. In conclusion, making LLMs understand the bug behavior is the first challenge.

- C2. Understanding code. Assuming LLM has comprehended the bug behavior, the next step is to prompt LLM to reason the possible bugs within the code. In this case, another challenge arises in enabling LLMs to comprehend the controller code [9]. While previous studies have successfully employed LLMs for code understanding [86] and debugging [71], these efforts often involved simpler codebases and utilized pre-designed prompts. In contrast, the Apollo MPC controller code is highly complex, spanning hundreds of lines and integrating multiple inter-

related modules such as data loading, intricate calculations, and post-processing. LLMs may struggle to grasp the high-level interactions between these modules and the dependencies that govern the system's behavior. Moreover, the MPC controller relies on sophisticated mathematical models and control theory, requiring a deep understanding of domain-specific knowledge that LLMs may lack. Consequently, directly feeding the code into an LLM is unlikely to yield effective bug identification or reasoning.

- C3. Identify practical bugs. Assuming that LLM has comprehended the planning-control inconsistency, the controller code, and identified potential bugs based on the correspondence between the two, the last challenge remains *how to identify the practical and real bugs from the potential bugs*. This is a challenge because the potential bugs output by LLM is based on its observations on the anomaly and code, without systematically testing the code. As a result, false positives could exist in LLM's output.

Solutions. To address the aforementioned challenges, we propose the following solutions:

- S1. Leveraging Vision Language Models (VLMs). As highlighted in challenges C1 and C2, traditional LLMs struggle with *long and complex* data (i.e., trajectory points and controller code). We seek a more efficient method to input this data into LLMs. In this context, a new type of LLM, the Vision Language Model (VLM) [153], becomes relevant. Unlike traditional LLMs that process only text input, VLMs are trained on both visual and text inputs, enabling them to process visual data according to text prompts. Utilizing VLMs' capability to handle visual input, we convert complex data (i.e., trajectory points and controller code) into *graphical representations* and feed them to the VLM along with tailored input

prompts. Specifically, we first transform the recorded trajectories into a *Planning-Control Inconsistency Graph* based on the metrics proposed in Sec.5.3.2 and the failure cases identified in Sec.5.5. This inconsistency graph illustrates the trends and comparisons between the planned and actual trajectories, effectively highlighting the bug behavior. Subsequently, we construct a *Controller Workflow Diagram* from the Apollo controller codebase using reliable static code analysis, including the creation of function call graphs and dataflow graphs. This workflow diagram, presented visually, provides a clear and high-level abstraction of the controller code, making it easier for the VLM to comprehend compared to directly inputting large volumes of code.

- S2. Implementing a Three-stage CoT Process. Building on S1, we developed a three-stage CoT process to guide the VLM in identifying potential bugs in the controller code. In the first stage, we input the planning-control inconsistency graph into the VLM to help it understand the bug behavior (i.e., *how* the controller is deficient). In the second stage, we provide the controller workflow diagram to the VLM, enabling it to understand the controller's operation in the context of the bug behavior. In the third stage, after the VLM has grasped both the bug behavior and the code, we extract code snippets from the codebase for line-of-code (LoC) level bug pinpointing. Following the three-stage CoT process, the VLM is expected to identify potential bugs with specific lines of code in the controller codebase (answering *why* the controller is deficient), starting from the bug behavior (*how*) and the controller codebase itself.

- S3. Conducting Reliable Dynamic Analysis. To identify practical bugs in the codebase, we further scrutinize the potential bugs identified by the VLM and conduct dynamic analysis for reliable bug identification. We instrument debug

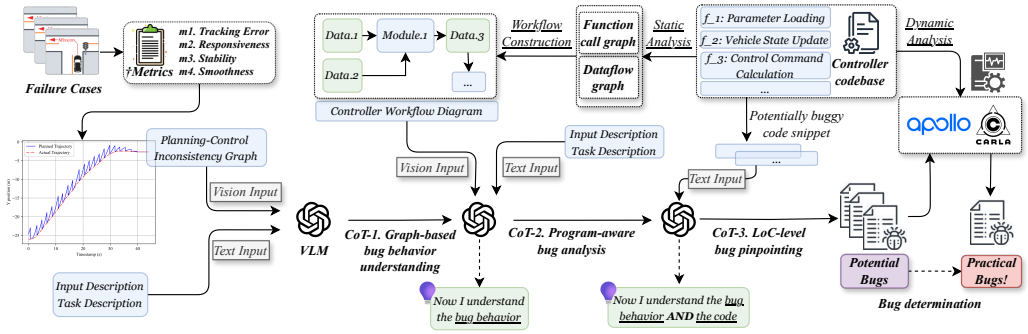


Figure 5.7: Workflow: VLM-assisted CoT bug analysis.

code in the potentially buggy LoCs and recompile the controller code to facilitate practical bug identification using debug information (e.g., dynamic values of key variables). Finally, we determine the *practical and real* bugs within the codebase through the three stages of CoT and the subsequent dynamic analysis.

5.6.1 Approach

Fig.5.7 provides an overview of the VLM-assisted CoT bug analysis workflow, which involves three stages of CoT and a final stage of bug determination based on dynamic analysis.

CoT-1. Graph-based Bug Behavior Understanding. The first stage of CoT involves prompting the VLM to understand the bug behavior. To achieve this, we constructed four types of inconsistency graphs based on our proposed metrics, including inconsistencies in 2D relative position (x-position and y-position), velocity, acceleration, and angular error (Fig.5.4). Each graph contains two lines representing the planned and actual trajectories, respectively. We then feed these inconsistency graphs to the VLM with explicit prompts, asking the VLM to (1) observe the trend of the two trajectories, (2) interpret the inconsistency between them, and (3) perform an initial analysis to indicate possible bugs. After CoT-1, the

VLM should have *understood* the bug behavior, making it ready for subsequent analysis.

CoT-2. Program-aware Bug Analysis. Following the understanding of the bug behavior, the second stage of CoT involves prompting the VLM to understand the controller logic and code. An intuitive approach is to convert the source code into a *function call graph* or *dataflow graph* as the visual input. However, we found that these types of graphs do not clearly represent the control logic for the VLM to understand, due to the complexity of the original code. Therefore, we sought a more efficient visual representation. Starting from the function call graph and dataflow graph, we abstracted the controller logic into a *Controller Workflow Diagram*, which is a high-level representation of how the controller works (detailed in §5.6.2). To ensure the correctness and quality of this diagram, the construction process is manually performed based on our understanding of the function call and dataflow graphs as well as the source code. Note that this construction process is a one-time effort as we focus on the particular controller code. With the diagram constructed, we feed it into the VLM, with prompts to (1) explicitly introduce the diagram and (2) require the VLM to infer possible bugs in each functional module of the diagram, based on the inconsistencies identified in CoT-1. Overall, after CoT-2, the VLM should have *understood* both the bug behavior and the controller code.

CoT-3. LoC-level Bug Pinpointing. In CoT-3, we further retrieve the potentially buggy code of specific modules in the controller diagram and prompt the VLM to pinpoint the specific bugs at the code level. Specifically, we prompt the VLM to first carefully comprehend the code in the context of the diagram provided in CoT-2, and then pinpoint the specific lines of buggy code, with detailed

explanations of *how* and *why* it is a bug. After CoT-3, we have a list of potential bugs with corresponding line numbers in the source code.

Bug Determination via Dynamic Analysis. The potential bugs identified by the VLM in CoT-3 are not necessarily the real bugs in the code, so we conduct dynamic analysis to determine the practical bugs. Specifically, we first review whether it is a highly probable practical bug in the context of the codebase. Then we instrument debug code in the potentially buggy LoCs and recompile the controller code for practical bug identification assisted by the debug information (e.g., the dynamic values of key variables). Finally, we attempt to fix the bug based on our observations and re-evaluate the metrics to determine whether the controller performs better after the bug fix. Through this dynamic analysis, starting from the potential bugs identified by the VLM, we can successfully identify the *practical and real* bugs within the codebase.

5.6.2 Experiments

Experimental Setup. To ensure the effectiveness of the bug-finding process, we selected one of the most powerful VLMs, GPT-4o (2024-08-06 API version [94]), for our CoT bug analysis. Note that other VLMs (e.g., Claude-3.5, open-source models) can also be implemented, provided they can accept visual modality input. The temperature of GPT-4o was set to *zero* to ensure the stability and reproducibility of the output.

Construction of Planning-Control Inconsistency Graphs. In Sec.5.5, we tested the Apollo MPC controllers using our proposed four metrics in various scenarios. Specifically, we chose the *B2_03. Sharp left turn* scenario (Fig.5.4) to con-

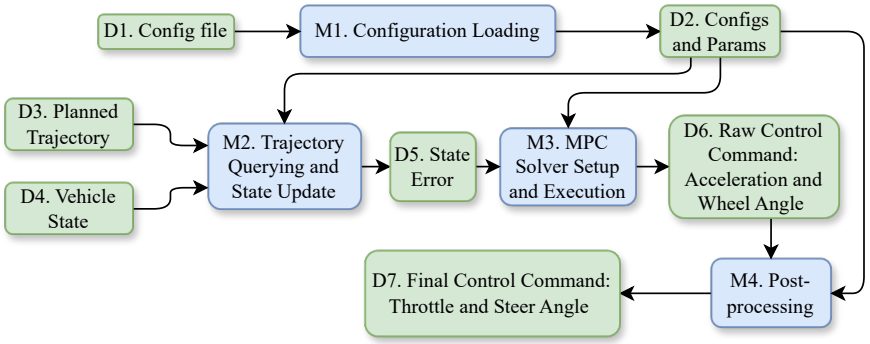


Figure 5.8: Controller Workflow Diagram of the tested controller - Apollo MPC. The diagram, which is derived from reliable program analysis, contains 7 data modules and 4 functional modules, offering an intuitive program representation for VLM to understand.

struct the inconsistency graphs, as this scenario reflects *typical* inconsistencies in all types of states (position, velocity, acceleration, and heading angle), as shown in Fig.5.4. We constructed a total of four graphs reflecting the aforementioned states and converted the graphs into images. Although GPT-4o does not have specific requirements for input image size, we ensured each image had a resolution higher than 512×512 to maintain the details in the graph.

Construction of Controller Workflow Diagram. We constructed the controller workflow diagram from the Apollo MPC controller code [9], which involves complex control logic across a total of 708 lines of code. Specifically, we first comprehended the code logic using traditional static analysis techniques, including constructing the *function call graph* and *dataflow graph*: the function call graph represents the logic of how the functions are called and executed in the *forward* direction, while the dataflow graph represents the logic of how the control commands are derived in the *reverse* direction. Subsequently, we summarized the *Controller Workflow Diagram* as an *intuitive and easy-to-understand* input for the VLM, as shown in Fig.5.8.

As shown in Fig.5.8, there are two types of blocks: data modules (starting with Dx) and function modules (starting with Mx), with each function module having its input and output data modules. Specifically, M1 is responsible for loading the configuration and parameters (D1) that globally affect all other function modules. M2 calculates the state error (D5) between the planning state (D3) and the actual state (D4), and M3 subsequently calculates the raw control command (D6) based on D5 via the MPC module. Finally, the raw command is converted into an executable command (D7) through post-processing (M4).

LoC-level Bug Pinpointing. After CoT-1 and CoT-2, we retrieved the responsible code for the functional modules in Fig.5.8 as input for the VLM for LoC-level bug identification. Among the four types of state errors involved, for position, velocity, and heading angle, 15 bugs were discovered for each, while 21 bugs were found for acceleration, resulting in a total of 66 bugs with clear LoC markings and explanations on why these lines of code are buggy.

Bug Determination. For the 66 bugs identified in CoT-3, each was accompanied by a detailed explanation of how and why it could be a bug affecting control performance. Following these insights provided by the VLM, we conducted further dynamic analysis to determine the practical bugs. Specifically, for each potential bug, we (1) instrumented the corresponding code to derive debug information during dynamic execution (ADS vehicle running under the instrumented MPC controller) and checked whether the expected anomaly existed; (2) fixed the bug and re-ran the controller to see if the performance improved (i.e., the inconsistency decreased after fixing). We ultimately identified 14 practical bugs within the controller, as shown in Tab.5.5.

Table 5.5: **Identified Bugs in Baidu Apollo *Planning-to-Control* flow.** All bugs have been acknowledged by the Apollo official [2] and Apollo-Carla Bridge official [119]. All bugs in the table have been fixed by the Bridge official [120].

Bug #	Bug Location Module	Bug Description	Bug Impact
01	M1. Configuration Loading	Ill-tuned parameter for MPC calculation	Under-performance of the MPC controller
02	M1. Configuration Loading	The sign of the actuator value (throttle and brake) is reversed	An acceleration command can be translated into a brake, or deceleration translated into throttle
03	M1. Configuration Loading	Translation between control command and acceleration violates physical rule	Under the same speed, lower acceleration is translated into higher throttle, or vice versa
04	M1. Configuration Loading	Contradictory vehicle parameter	Contradictory tire mass in two config files, resulting in different output control commands
05	M1. Configuration Loading	Unsmooth mapping from acceleration to throttle and brake signal	Abrupt changes in the control signals for acceleration (throttle) and deceleration (brake)
06	M1. Configuration Loading	Mapped throttle values (from acceleration values) are too low to accelerate the vehicle to planned velocity	Autonomous vehicle does not accelerate in simulator
07	M2. Trajectory Query and State Update	Lateral error and longitudinal error are calculated based on inconsistent planning point	Control output is calculated based on non-existing trajectory point
08	M2. Trajectory Query and State Update	No looking-ahead time is reserved when querying trajectory points	Impractical planned point is selected, leading to lack of responsiveness
09	M2. Trajectory Query and State Update	Frequency of the control command (100hz) does not sync with the frequency of state update (20hz)	Controller keeps computing *outdated* control command based on *outdated* state
10	M2. Trajectory Query and State Update	The trajectory point to be followed is queried by real-time timestamp instead of state update timestamp	Controller keeps computing *outdated* control command based on *outdated* state
11	M4. Filtering and Post-Processing	Output steering command could be filtered twice	Over-smoothed steering command
12	M4. Filtering and Post-Processing	Acceleration command is added with redundant feed-forward term, nerfing MPC's utility	Improper implementation of MPC, leading to over-acceleration command
13	M4. Filtering and Post-Processing	Steering command is added with redundant feed-forward term, nerfing MPC's utility	Improper implementation of MPC, of MPC; Over-steering command
14	M4. Filtering and Post-Processing	Lack of filtering on lateral error input	Under-smoothed steering command

5.6.3 Identified Bugs

Bug Details. Tab.5.5 shows the 14 specific bugs we identified in the Apollo MPC controller, with detailed descriptions and their impacts. There are 6 bugs in *M1. Configuration Loading*, 4 bugs in *M2. Trajectory Query and State Update*, and 4 bugs in *M4. Filtering and Post-Processing*. No bugs were identified in *M3. MPC Solver Setup and Execution* because this module was implemented based on a high-quality third-party library [85]. All bugs listed in the table have been acknowledged by the Apollo official [2] and Apollo-Carla Bridge official [119], and were promptly fixed by the Bridge official [120].

Case Study - How VLM was Helpful? We present two specific cases to illustrate how the VLM helped identify practical bugs in the controller.

- Case.1: Redundant compensations. The following is a bug identified by

the VLM after CoT-3, regarding the calculation of the final control command:

> ***- Bug Position:*** *Line 393-408: Feedforward Compensation.*
>
> ***- Potential Bug:*** *The feedforward compensation logic is complex and involves multiple calculations, but there are no checks to ensure that the intermediate values are within reasonable bounds.*
>
> ***- Correspondence to Inconsistency:*** *If any of the intermediate values are incorrect or extreme, it could lead to incorrect feedforward compensation, causing deviations in the Actual Trajectory from the Planned Trajectory.*

As indicated by the VLM, the control signals were derived from several different terms, involving complex feedforward compensations, which could be the key to the buggy behavior. Following this insight, we extensively tested the compensation terms and found that improper compensations were added to both the acceleration (longitudinal) control and steering (lateral) control, leading to the identification of Bug#12 and Bug#13 in Tab.5.5.

- Case.2: Faulty configurations. The following is another bug pointed out by the VLM, regarding the control configurations:

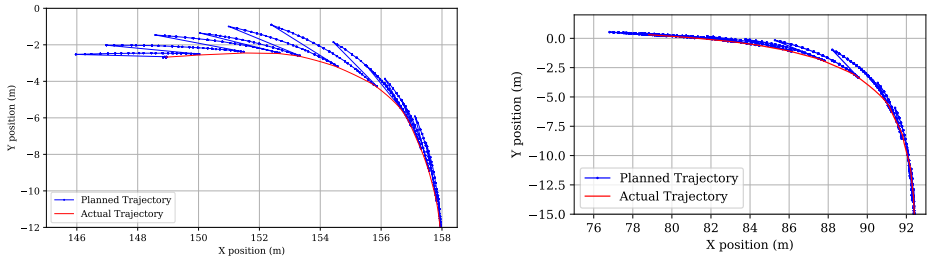
> ***- Bug Position:*** *Line 393-408: Line Numbers: 475-482.*
>
> ***- Potential Bug:*** *If the calibration value is not correctly interpreted, the throttle and brake commands will be incorrect, leading to improper longitudinal control.*
>
> ***- Correspondence to Inconsistency:*** *Incorrect throttle and brake commands can cause the vehicle to deviate from the planned trajectory, particularly during acceleration or deceleration phases.*

In the Apollo MPC controller, one important configuration is the *calibration table*, which is responsible for mapping the raw control commands to executable control commands (from D6 to D7 in Fig.5.8). As indicated by the VLM, the calibration value could be problematic, leading to buggy behavior. Following this insight, we identified Bug#05 and Bug#06 in Tab.5.5.

5.6.4 Bug Fixes

After identifying these bugs, we attempted to fix them to improve the control performance. The fixed controller code is available via [120].

Re-evaluation of Fixed Controller. After fixing the identified bugs, we con-



(a) *Before fixing:* there is obvious and consistent error between the planned and actual trajectory.

(b) *After fixing:* vehicle follows the planned trajectory much more accurately.

Figure 5.9: Recorded planned and actual trajectory when vehicle makes a left turn. (a): before fixing; (b): after fixing.

ducted a new round of metric-based assessments and found that the fixed controller performs significantly better than the original buggy one. Specifically, it follows the planned trajectory more accurately and promptly. The compared of the trajectories before and after fixing is shown in Fig.5.9, validating our bug findings. More details about the bug fixing can be found in [120].

Responsible Disclosure. All discovered bugs have been reported to the relevant parties, including the Apollo official [2] and Apollo-Carla bridge official [119]. Additionally, we have provided detailed bug fixes to them and have been assisting the bridge official [119] in addressing these bugs. Currently, all identified bugs in Tab.5.5 have been resolved in their updated codebase [120].

Bug Significance. Note that Apollo remains a leading open-source ADS, widely recognized for its educational value, industrial benchmarks, and high-quality resources for academic research. Therefore, ensuring the code quality of such an impactful ADS - Apollo, is of vital significance, and the existence of the controller bugs is already a real-world impact, as they will bring realistic problems to educators, industrial practitioners, and researchers.

5.7 Discussion

Manual Efforts in Bug Analysis. Due to the complex and non-deterministic mapping between bug behavior and root cause in ADS controllers, the bug analysis process is *semi-automatic*, necessitating some manual efforts. These include (a) constructing the controller workflow diagram and (b) performing dynamic analysis to determine the final bugs. However, we emphasize that (a) is a one-time effort completed within hours of inspecting the code structure, and such manual efforts are necessary to ensure the quality of the diagram, making it easier for the VLM to understand. As for (b), due to the complexity of the bugs, specific dynamic tests are required for each possible bug. Nevertheless, our VLM-assisted analysis effectively identified a list of potential bugs, starting from the bug behaviors and code logic (as shown in Sec.5.6.3), making the debugging process much more efficient than aimlessly debugging from scratch.

Generalizability. We conducted experiments on the Apollo ADS and Carla simulation. However, our proposed metrics and testing methodology are ADS-agnostic (as long as the system follows a modular design involving a *planning module*), simulator-agnostic, and controller-agnostic, making them readily adaptable to other ADS systems or controllers, and other simulators. For a new ADS with a simulator, one can still collect the planned trajectory T_p and actual trajectory T_a, then evaluate the controller based on the metrics proposed in §5.3.2. Additionally, the controller workflow diagram (Fig.5.8) is a general abstraction of an MPC controller in the ADS context, which we constructed as a one-time effort and could be used for other MPC controllers in the bug analysis process.

Bug novelty compared with Drivefuzz [67]. Drivefuzz [67] also identi-

fied controller-related bugs with their *Driving Quality-Guided Fuzzing*. However, Drivefuzz fails to analyze the root cause of almost all identified bugs (e.g., simply attributes the cause of bugs to *"Faulty conf"*) and these bugs remain unfixed. In comparison, we conducted in-depth analysis on the controller code and found the code-level root causes of identified bugs (Tab.5.5). All identified bugs were acknowledged, and most of them were promptly fixed with our assistance.

Comparison with Other Testing Approaches. It is important to note that previous works have tested ADS using techniques other than fuzzing-based approaches. For instance, Yao et al. [34] proposed a metamorphic testing approach to identify potential issues in the overall ADS behavior. Specifically, they presented a novel declarative rule-based metamorphic testing framework that automatically parses human-written rules into metamorphic relations to generate test cases using a variety of image transformation engines. Compared to fuzzing-based testing approaches (Tab.2.2), the metamorphic-based testing approach can be more efficient in certain cases. In the future, we plan to explore the possibility of combining different testing approaches to leverage the advantages of various methods.

Rationale behind Choosing VLM. VLM was chosen over other multi-modality models like VisualBERT [77] because it is significantly more powerful, thanks to its larger parameter size and training dataset. In the context of bug analysis, we required a highly capable model to perform the challenging task of identifying the code-level root cause of bug behavior. Specifically, we used GPT-4o in our experiments, demonstrating that VLM can effectively pinpoint specific bugs in the code.

Chapter 6

Conclusion

6.1 Conclusion

6.1.1 Revisiting Automotive Attack Surfaces

We conducted the first in-depth interview study with 15 experts working in automotive cybersecurity, revealing the specific challenges when security activities are being conducted, and the limitations of existing regulations. Particularly, we found that the threat cases given by current regulations are insufficient, and conducting TARA is often labor-intensive due to the lack of automatic tools. To address these challenges, we constructed a hierarchical threat database for automotive systems based on the interview data, improving the existing database both quantitatively and qualitatively. Moreover, we propose CarVal, a datalog-based approach that could generate multi-stage attack paths in IVN and calculate risk values. By applying CarVal to five real cars, we conducted extensive security analysis based on the generated attack paths, and successfully exploited corresponding attack chains

in the new gateway-segmented IVN. In conclusion, our experimental analysis on real cars demonstrated the significant potential risks on new attack surfaces emerging in modern vehicles. Moreover, the proposed database and methodology will shed light on how security activities (e.g., TARA and security testing) can be conducted more efficiently, as a supplement to existing regulations.

6.1.2 Attacking ADS Perception

We conduct the first investigation on the lane detection module in a real vehicle, and reveal that its sensitivity can be exploited to launch attacks on the vehicle. Specifically, we propose a novel two-stage approach to automatically determine the best perturbations in digital world and then project them back to the markings in physical world after addressing technical challenges. We conduct extensive experiments on a Tesla Model S vehicle. The experimental results show that the lane detection module can be deceived by crafted perturbations and mislead the vehicle in auto-steer mode.

6.1.3 Testing ADS Controller

We presented the first comprehensive study focusing on the control module of the autonomous driving system, a pivotal component that was often ignored in previous research. We introduced four dedicated metrics designed specifically to test and evaluate the ADS control module and further enhanced the current state-of-the-art scenario-based fuzzing methodology based on these metrics. Using the guidelines set by our proposed metrics, we carried out an in-depth analysis of the control module within the industrial-level ADS system, Apollo. Our findings

revealed significant inadequacies: the control module struggles with even basic operations. Moreover, according to the proposed metrics, we highlighted specific shortcomings of the controller, including the insufficiency in tracking accuracy, responsiveness, stability and smoothness. To identify the specific bugs leading to such insufficiency in the controller code, we proposed an semi-automatic bug analysis approach assisted by VLM in CoT. With the help of the semi-automatic approach, we conducted extensive analysis of Apollo's codebase, and identified 14 new bugs in the Apollo controller. All bugs have been reported and promptly fixed with our assistance.

6.2 Future Work

There are the following direction of future work derived from this thesis.

6.2.1 Revisiting Automotive Attack Surfaces

Currently, the baseline "attack impact" and "attack feasibility" values (f_{EN}, f_{VN}, i_{VN}, f_{AN}, i_{AN} in Table.3.2) were manually assigned based on the specific context. For example, the feasibility to access the physical OBD-II port should be lower than the feasibility to access the wireless channel, and the impact brought by root execution is higher than that brought by low-privilege execution. Note that it is challenging to derive a set of universal or common baseline values that can be applied to all situations. This is because different groups may assign different baseline values to better suit their demand, and these impact and feasibility values can vary in different car models. Overall, these baseline values are flexible for users to set. Additionally, some research is focusing on scoring the individual automotive

threat [10, 148], which can give guidelines about how to set up these baseline risk values. In future work, we plan to further enhance the proposed tool by setting up a systematic baseline metrics for risk assessment. Moreover, since the current interviewees were either from first-party OEMs or third-party suppliers, expanding the pool of interviewees could provide broader insights. In the future, we plan to invite more interviewees from other organizations (e.g., automotive cybersecurity consortiums such as Automotive ISAC [5]) to gain additional valuable insights and enhance the threat database.

Additionally, it is important to acknowledge that not all identified limitations could be resolved within the scope of this thesis. The complexity of modern vehicles and the ever-evolving landscape of cybersecurity threats present a persistent challenge to the industry. For example, it is out of scope to propose a very clear threshold for how to mitigate the threats that all manufacturers must follow. It is important to recognize that fostering a strong cybersecurity culture and refining existing standards and regulations will require continued efforts from the automotive industry, regulatory bodies, and researchers. As a result, it is crucial to continuously update the threat database, and refine the automatic tool to stay ahead of emerging risks, which is another future work. Moreover, the primary limitation in CarVal is the manual effort required to craft reasoning rules for the CarVal Datalog reasoning engine. Designing these rules demands extensive expertise and considerable manual effort. Currently, we have manually created dozens of reasoning rules, which are sufficient for the current implementation of CarVal. In the future, we plan to explore more efficient methods for crafting these rules, such as leveraging Large Language Models (LLMs), to enhance CarVal's comprehensiveness and usability.

6.2.2 Attacking ADS Perception

This work can be extended from the following three aspects. First, we can assess the vulnerability of the lane detection modules in other autonomous driving systems, including Apollo [3] and Openpilot [96]. Second, we can explore the feasibility of launching attacks on the lane detection modules by adding perturbations on real lanes, such as using dark markings to cover part of real lanes or adding markings to change the shape of real lanes. Finally, The core of our attack methodology lies in defining the obtrusiveness of input perturbations and the confidence levels of detected objects to quantify the quality of the perturbations. For other objects like traffic lights or pedestrians, we can derive similar metrics within their respective contexts. Additionally, we can use heuristic algorithms to determine the optimal perturbations to deceive other corresponding detection modules, such as traffic light detection or pedestrian detection.

6.2.3 Testing ADS Controller

Currently our testing is conducted on the Apollo ADS and the Carla Simulator. We plan to also test other ADS systems including Openpilot [95] and Autoware [8]. Additionally, as we also identified several bugs in the planning module, we plan to extend our testing approach to the planning module, instead of mainly focusing on the control module.

References

[1] Ansaf Ibrahem Alrabady and Syed Masud Mahmud. “Analysis of attacks against the security of keyless-entry systems for vehicles and suggestions for improved designs”. In: *IEEE transactions on vehicular technology* 54.1 (2005), pp. 41–50.

[2] Apollo. *Github*. https://github.com/ApolloAuto/apollo. 2024.

[3] *Apollo autonomous driving*. https://github.com/ApolloAuto/apollo.

[4] *AUTEL 919 professional diagnostic tools*. https://item.jd.com/70636576685.html. 2023.

[5] *Automotive ISAC: AUTOMOTIVE INFORMATION SHARING AND ANALYSIS CENTER*. https://automotiveisac.com/.

[6] *Automotive ISAC's Risk Assessment and Management*. https://automotiveisac.com/best-practices.

[7] *Automotive Zonal Architecture - Guardknox*. https://www.guardknox.com/automotive-zonal-architecture/.

[8] Autoware. *Github*. https://github.com/autowarefoundation/autoware. 2024.

[9] *Baidu Apollo: MPC code*. `https://github.com/ApolloAuto/apollo/blob/r8.0.0/modules/control/controller/mpc_controller.cc`. 2024.

[10] Pranshu Bajpai and Richard Enbody. "Towards effective identification and rating of automotive vulnerabilities". In: *Proceedings of the Second ACM Workshop on Automotive and Aerial Vehicle Security*. 2020, pp. 37–44.

[11] Melanie Birks and Jane Mills. *Grounded theory: A practical guide*. Sage, 2015.

[12] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv:1604.07316* (2016).

[13] Adith Boloor et al. "Attacking vision-based perception in end-to-end autonomous driving models". In: *Journal of Systems Architecture* 110 (2020), p. 101766.

[14] Steve Bono et al. "Security Analysis of a Cryptographically-Enabled RFID Device." In: *USENIX Security Symposium*. Vol. 31. 2005, pp. 1–16.

[15] Amol Borkar, Monson Hayes, and Mark T Smith. "A novel lane detection system with efficient ground truth generation". In: *IEEE Transactions on Intelligent Transportation Systems* 13.1 (2011), pp. 365–374.

[16] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. "A comprehensive survey of graph embedding: Problems, techniques, and applications". In: *IEEE Transactions on Knowledge and Data Engineering* 30.9 (2018), pp. 1616–1637.

[17] Yulong Cao et al. “Adversarial sensor attack on lidar-based perception in autonomous driving”. In: *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2019, pp. 2267–2281.

[18] Yulong Cao et al. “Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 176–194.

[19] Nicholas Carlini et al. “Hidden voice commands”. In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016, pp. 513–530.

[20] *CarVal code and supplementary materials*. `https://github.com/VehicleCyberSec/CarVal`.

[21] Stephen Checkoway et al. “Comprehensive experimental analyses of automotive attack surfaces.” In: *USENIX Security Symposium*. Vol. 4. San Francisco. 2011, pp. 447–462.

[22] Li Chen et al. “End-to-end Autonomous Driving: Challenges and Frontiers”. In: *arXiv preprint arXiv:2306.16927* (2023).

[23] *Chinese Standard: GB/T 40861—2021 General technical requirements for vehicle cybersecurity*. `https://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=2977F0AC1719BBEFB9649C0146B0FC55`. 2023.

[24] Kyong-Tak Cho and Kang G Shin. “Error handling of in-vehicle networks makes them vulnerable”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1044–1055.

[25] Juliet Corbin and Anselm Strauss. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage publications, 2014.

[26] Gianpiero Costantino, Marco De Vincenzi, and Ilaria Matteucci. "A Comparative Analysis of UNECE WP. 29 R155 and ISO/SAE 21434". In: *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2022, pp. 340–347.

[27] *CUDA memory management APIs*. `https://bit.ly/3dlFozE`.

[28] *CUDA Toolkit Documentation*. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/`.

[29] *cudaConfigureCall*. `https://bit.ly/2ZucaX1`.

[30] *cudaMalloc*. `https://bit.ly/2M2Qnmb`.

[31] *cudaMemcpy*. `https://bit.ly/3aulsIV`.

[32] Jin Cui et al. "A review on safety failures, security attacks, and available countermeasures for autonomous vehicles". In: *Ad Hoc Networks* 90 (2019), p. 101823.

[33] *Demonstration video: misguiding the vehicle in real world*. `https://youtu.be/a__Se2MrjVs`.

[34] Yao Deng et al. "A declarative metamorphic testing framework for autonomous driving". In: *IEEE Transactions on Software Engineering* 49.4 (2022), pp. 1964–1982.

[35] Mahdi Dibaei et al. “Attacks and defences on intelligent connected vehicles: a survey”. In: *Digital Communications and Networks* 6.4 (2020), pp. 399–421.

[36] Derrick Dominic et al. “Risk assessment for cooperative automated driving”. In: *Proceedings of the 2nd ACM workshop on cyber-physical systems security and privacy*. 2016, pp. 47–58.

[37] Timothy Dougan and Kevin Curran. “Man in the browser attacks”. In: *International Journal of Ambient Computing and Intelligence (IJACI)* 4.1 (2012), pp. 29–39.

[38] Masoud Ebrahimi et al. “A Systematic Approach to Automotive Security”. In: *International Symposium on Formal Methods*. Springer. 2023, pp. 598–609.

[39] Kevin Eykholt et al. “Robust physical-world attacks on deep learning visual classification”. In: *Proc. CVPR*. 2018.

[40] Diogo AB Fernandes et al. “Security issues in cloud environments: a survey”. In: *International Journal of Information Security* 13.2 (2014), pp. 113–170.

[41] Ian Foster et al. “Fast and vulnerable: A story of telematic failures”. In: *9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15)*. 2015.

[42] Aurélien Francillon, Boris Danev, and Srdjan Capkun. “Relay attacks on passive keyless entry and start systems in modern cars”. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science. 2011.

[43] Marcel Frigault and Lingyu Wang. “Measuring network security using bayesian network-based attack graphs”. In: *2008 32nd Annual IEEE International Computer Software and Applications Conference*. IEEE. 2008, pp. 698–703.

[44] Flavio D Garcia et al. “Lock it and still lose it—on the (in) security of automotive remote keyless entry systems”. In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016.

[45] Micah Goldblum et al. “Dataset Security for Machine Learning: Data Poisoning,Backdoor Attacks, and Defenses”. In: *arXiv:2012.10544* (2020).

[46] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv:1412.6572* (2014).

[47] Marc Green. “” How long does it take to stop?” Methodological analysis of driver perception-brake times”. In: *Transportation human factors* 2.3 (2000).

[48] Steffen Hagedorn et al. “Rethinking Integration of Prediction and Planning in Deep Learning-Based Automated Driving Systems: A Review”. In: *arXiv preprint arXiv:2308.05731* (2023).

[49] R Spencer Hallyburton et al. “Security Analysis of {Camera-LiDAR} Fusion Against {Black-Box} Attacks on Autonomous Vehicles”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 1903–1920.

[50] Julie M Haney and Wayne G Lutters. “” It’s {Scary… It’s}{Confusing… It’s} Dull”: How Cybersecurity Advocates Overcome Negative Perceptions of Security”. In: *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. 2018, pp. 411–425.

[51] Zhijian He et al. “A system identification based oracle for control-cps software fault localization”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE. 2019, pp. 116–127.

[52] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. “Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures”. In: *Reliability Engineering & System Safety* 96.1 (2011), pp. 11–25.

[53] Shengtuo Hu et al. “Gatekeeper: A Gateway-based Broadcast Authentication Protocol for the In-Vehicle Ethernet”. In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 2022, pp. 494–507.

[54] Yuqi Huai et al. “Doppelgänger Test Generation for Revealing Bugs in Autonomous Driving Software”. In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 2591–2603.

[55] Abdulmalik Humayed et al. “Cyber-physical systems security—A survey”. In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1802–1831.

[56] Brody Huval et al. “An empirical evaluation of deep learning on highway driving”. In: *arXiv:1504.01716* (2015).

[57] *IDA Pro*. `https://www.hex-rays.com/products/ida/`.

[58] *ISO/SAE 21434:2021: Road vehicles — Cybersecurity engineering*. `https://www.iso.org/standard/70918.html`. 2021.

[59] Shaoxiong Ji et al. “A survey on knowledge graphs: Representation, acquisition, and applications”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.2 (2021), pp. 494–514.

[60] Yujie Ji et al. “Model-Reuse Attacks on Deep Learning Systems”. In: *Proc. CCS*. 2018.

[61] Pengfei Jing et al. “Too Good to Be Safe: Tricking Lane Detection in Autonomous Driving with Crafted Perturbations”. In: *30th {USENIX} Security Symposium ({USENIX} Security 21)*. 2021.

[62] Pengfei Jing et al. “Too good to be safe: Tricking lane detection in autonomous driving with crafted perturbations”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 3237–3254.

[63] *JNI: Java Native Interface*. `https://developer.android.com/training/articles/perf-jni`. 2023.

[64] Hyo Jin Jo et al. “Vulnerabilities of android OS-based telematics system”. In: *Wireless Personal Communications* 92.4 (2017), pp. 1511–1530.

[65] Heechul Jung, Junggon Min, and Junmo Kim. “An efficient lane detection algorithm for lane departure detection”. In: *Proc. IEEE Intelligent Vehicles Symposium*. 2013.

[66] Jin Ho Kim et al. “Gateway framework for in-vehicle networks based on CAN, FlexRay, and Ethernet”. In: *IEEE Transactions on Vehicular Technology* 64.10 (2014), pp. 4472–4486.

[67] Seulbae Kim et al. “Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing”. In: *Proceedings of the 2022 ACM SIGSAC*

Conference on Computer and Communications Security. 2022, pp. 1753–1767.

[68] Karl Koscher et al. “Experimental security analysis of a modern automobile”. In: *2010 IEEE Symposium on Security and Privacy*. IEEE. 2010, pp. 447–462.

[69] Sekar Kulandaivel et al. “Cannon: Reliable and stealthy remote shutdown attacks via unaltered automotive microcontrollers”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 195–210.

[70] Yangyang Liu Le Yu et al. “Towards Automatically Reverse Engineering Vehicle Diagnostic Protocols”. In: *Proc. USENIX Security*. 2022.

[71] Hokyung Lee, Sumanyu Sharma, and Bing Hu. “Bug In the Code Stack: Can LLMs Find Bugs in Large Python Code Stacks”. In: *arXiv preprint arXiv:2406.15325* (2024).

[72] Seokju Lee et al. “Vpgnet: Vanishing point guided network for lane and road marking detection and recognition”. In: *Proc. ICCV*. 2017.

[73] Guanpeng Li et al. “Av-fuzzer: Finding safety violations in autonomous driving systems”. In: *2020 IEEE 31st international symposium on software reliability engineering (ISSRE)*. IEEE. 2020, pp. 25–36.

[74] Guanpeng Li et al. “Av-fuzzer: Finding safety violations in autonomous driving systems”. In: *2020 IEEE 31st international symposium on software reliability engineering (ISSRE)*. IEEE. 2020, pp. 25–36.

[75] Heng Li et al. “Robust Android Malware Detection Against Adversarial Example Attacks”. In: *Proc. WWW*. 2021.

[76] Jun Li et al. “Deep neural network for structural prediction and lane detection in traffic scene”. In: *IEEE transactions on neural networks and learning systems* 28.3 (2016), pp. 690–703.

[77] Liunian Harold Li et al. “Visualbert: A simple and performant baseline for vision and language”. In: *arXiv preprint arXiv:1908.03557* (2019).

[78] Guannan Lou et al. “Testing of autonomous driving systems: where are we and where should we go?” In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2022, pp. 31–43.

[79] Yixing Luo et al. “Targeting requirements violations of autonomous driving systems by dynamic evolutionary search”. In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2021, pp. 279–291.

[80] Dawei Lyu, Lei Xue, and Xiapu Luo Le Yu. *Remote attacks on vehicles by exploiting vulnerable telematics*. 2016.

[81] David Q Mayne. “Model predictive control: Recent developments and future promise”. In: *Automatica* 50.12 (2014), pp. 2967–2986.

[82] Charlie Miller and Chris Valasek. “A survey of remote automotive attack surfaces”. In: *black hat USA* 2014 (2014), p. 94.

[83] Charlie Miller and Chris Valasek. “Adventures in automotive networks and control units”. In: *Def Con* 21.260-264 (2013), pp. 15–31.

[84] Charlie Miller and Chris Valasek. “Remote exploitation of an unaltered passenger vehicle”. In: *Black Hat USA* 2015.S 91 (2015).

[85] *MPC-OSQP*. https://osqp.org/docs/examples/mpc.html. 2024.

[86] Daye Nam et al. “Using an llm to help with code understanding”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024, pp. 1–13.

[87] Ben Nassi et al. “Phantom of the ADAS: Phantom Attacks on Driver-Assistance Systems”. In: *Proc. CCS*. 2020.

[88] Ben Nassi et al. “Phantom of the adas: Securing advanced driver-assistance systems from split-second phantom attacks”. In: *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 2020, pp. 293–308.

[89] Ben Nassi et al. “Phantom of the adas: Securing advanced driver-assistance systems from split-second phantom attacks”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 293–308.

[90] Davy Neven et al. “Towards end-to-end lane detection: an instance segmentation approach”. In: *Proc. IEEE Intelligent Vehicles Symposium*. 2018.

[91] *NHTSA: Cybersecurity of Firmware Updates*. https://www.nhtsa.gov/sites/nhtsa.gov/files/documents/cybersecurity_of_firmware_updates_oct2020.pdf.

[92] *Nvidia, Drive AP2X*. https://www.nvidia.com/en-us/self-driving-cars/drive-platform.

[93] Katsuhiko Ogata. *Modern control engineering fifth edition*. 2010.

[94] *OpenAI: GPT-4o*. https://openai.com/index/hello-gpt-4o/. 2024.

[95] Openpilot. *Github*. https://github.com/commaai/openpilot. 2024.

[96] *Openpilot autonomous driving*. https://github.com/commaai/openpilot.

[97] Xinming Ou, Wayne F Boyer, and Miles A McQueen. "A scalable approach to attack graph generation". In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006, pp. 336–345.

[98] Xinming Ou, Sudhakar Govindavajhala, Andrew W Appel, et al. "MulVAL: A Logic-based Network Security Analyzer." In: *USENIX security symposium*. Vol. 8. Baltimore, MD. 2005, pp. 113–128.

[99] Xinming Ou and Anoop Singhal. "Attack graph techniques". In: *Quantitative Security Risk Assessment of Enterprise Networks*. Springer, 2012, pp. 5–8.

[100] *OWASP Threat Dragon*. https://owasp.org/www-project-threat-dragon/.

[101] Hernan Palombo et al. "An Ethnographic Understanding of Software ({In) Security} and a {Co-Creation} Model to Improve Secure Software Development". In: *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 2020, pp. 205–220.

[102] Lei Pan et al. "Cyber security attacks to modern vehicular systems". In: *Journal of information security and applications* 36 (2017), pp. 90–100.

[103] Ren Pang et al. "A Tale of Evil Twins: Adversarial Inputs versus Poisoned Models". In: *Proc. CCS*. 2020.

[104] Ren Pang et al. “AdvMind: Inferring Adversary Intent of Black-Box Attacks”. In: *Proc. KDD*. 2020.

[105] *Past Statistics on Texting & Cell Phone Use While Driving*. `https://www.edgarsnyder.com/car-accident/cause-of-accident/cell-phone/past-cell-phone-statistics.html`.

[106] Jonathan Petit and Steven E Shladover. “Potential cyberattacks on automated vehicles”. In: *IEEE Transactions on Intelligent transportation systems* 16.2 (2014), pp. 546–556.

[107] Kui Ren et al. “The security of autonomous driving: Threats, defenses, and future directions”. In: *Proceedings of the IEEE* 108.2 (2019), pp. 357–372.

[108] Ishtiaq Rouf et al. “Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study.” In: *USENIX Security Symposium*. Vol. 10. 2010.

[109] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv:1609.04747* (2016).

[110] Takami Sato et al. “Dirty Road Can Attack: Security of Deep Learning based Automated Lane Centering under {Physical-World} Attack”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 3309–3326.

[111] Takami Sato et al. “Dirty road can attack: Security of deep learning based automated lane centering under {Physical-World} attack”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 3309–3326.

[112] Takami Sato et al. “Security of Deep Learning based Lane Keeping System under Physical-World Adversarial Attack”. In: *arXiv:2003.01782* (2020).

[113] Christoph Schmittner et al. “ThreatGet: Threat modeling based approach for automated and connected vehicle systems”. In: *AmE 2020-Automotive meets Electronics; 11th GMM-Symposium*. VDE. 2020, pp. 1–3.

[114] Khaled Serag et al. “Exposing New Vulnerabilities of Error Handling Mechanism in {CAN}”. In: *30th {USENIX} Security Symposium ({USENIX} Security 21)*. 2021.

[115] Junjie Shen et al. “Drift with Devil: Security of Multi-Sensor Fusion based Localization in High-Level Autonomous Driving under GPS Spoofing”. In: *Proc. USENIX Security Symposium*. 2020.

[116] Ruoyu Song et al. “Discovering Adversarial Driving Maneuvers against Autonomous Vehicles”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 2957–2974.

[117] Eduardo D Sontag and Yuan Wang. “On characterizations of the input-to-state stability property”. In: *Systems & Control Letters* 24.5 (1995), pp. 351–359.

[118] Jiachen Sun et al. “Towards robust lidar-based perception in autonomous driving: General black-box adversarial sensor attack and countermeasures”. In: *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2020, pp. 877–894.

[119] SYNKROTRON. *Apollo-Carla Co-simulation Bridge*. `https://github.com/guardstrikelab/carla_apollo_bridge`. 2024.

[120] SYNKROTRON: *Refined Apollo Controller*. https://github.com/guardstrikelab/carla_apollo_bridge/blob/master/docs/RefinedController.md. 2024.

[121] Christian Szegedy et al. "Intriguing properties of neural networks". In: *arXiv:1312.6199* (2013).

[122] J Takahashi and T Fukunaga. "Implementation attacks on an immobilizer protocol stack". In: *11th Embedded Security in Cars Conference Europe, escar Europe*. 2013.

[123] Huachun Tan et al. "A novel curve lane detection based on Improved River Flow and RANSA". In: *Proc. IEEE Conference on Intelligent Transportation Systems*. 2014.

[124] Teng-Tiow Tay, Iven Mareels, and John B Moore. *High performance control*. Springer Science & Business Media, 1998.

[125] *Tesla Autopilot System*. https://www.tesla.com/autopilot.

[126] *Tesla Hardware Information*. https://teslatap.com/undocumented/.

[127] *Tesla Model S*. https://www.tesla.com/models.

[128] *Tesla Model S crash*. https://www.wired.com/story/tesla-autopilot-why-crash-radar.

[129] *Texting and Driving Accident Statistics*. https://www.edgarsnyder.com/car-accident/cause-of-accident/cell-phone/cell-phone-statistics.html.

[130] *THREATGET - THREAT ANALYSIS AND RISK MANAGEMENT*. https://www.threatget.com/. 2023.

[131] Haoxiang Tian et al. “Generating Critical Test Scenarios for Autonomous Driving Systems via Influential Behavior Patterns”. In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–12.

[132] Haoxiang Tian et al. “Generating Critical Test Scenarios for Autonomous Driving Systems via Influential Behavior Patterns”. In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–12.

[133] Haoxiang Tian et al. “MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm”. In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2022, pp. 94–106.

[134] Launch Tech professional diagnostic tools. *X-431 Pad III*. `https://launchtechusa.com/new-product-x431-pad3/`. 2023.

[135] *treelib: a tree data structure for Python*. `https://treelib.readthedocs.io/en/latest/`.

[136] Anwesh Tuladhar et al. “An analysis of the role of situated learning in starting a security culture in a software company”. In: *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*. 2021, pp. 617–632.

[137] *Uber’s Self-Driving Cars Were Struggling Before Arizona Crash*. `https://www.nytimes.com/2018/03/23/technology/uber-self-driving-cars-arizona.html`.

[138] *Udacity Self-driving Car*. https://github.com/udacity/self-driving-car.

[139] *UDP: User Datagram Protocol*. https://en.wikipedia.org/wiki/User_Datagram_Protocol. 2023.

[140] *UN Regulation No.155 - Cyber security and cyber security management system*. https://unece.org/sites/default/files/2021-03/R155e.pdf. 2021.

[141] Roel Verdult, Flavio D Garcia, and Josep Balasch. "Gone in 360 seconds: Hijacking with Hitag2". In: *21st {USENIX} Security Symposium ({USENIX} Security 12)*. 2012, pp. 237–252.

[142] Roel Verdult, Flavio D Garcia, and Baris Ege. "Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer". In: *Supplement to the Proceedings of 22nd {USENIX} Security Symposium (Supplement to {USENIX} Security 15)*. 2015, pp. 703–718.

[143] Ziwen Wan et al. "Too Afraid to Drive: Systematic Discovery of Semantic DoS Vulnerability in Autonomous Driving Planning under Physical-World Attacks". In: *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022. URL: https://www.ndss-symposium.org/ndss-paper/auto-draft-214/.

[144] Ziwen Wan et al. "Too Afraid to Drive: Systematic Discovery of Semantic DoS Vulnerability in Autonomous Driving Planning under Physical-World Attacks". In: *Network and Distributed Systems Security Symposium*. 2022.

[145] Lingyu Wang et al. “An attack graph-based probabilistic security metric”. In: *Data and Applications Security XXII: 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security London, UK, July 13-16, 2008 Proceedings 22*. Springer. 2008, pp. 283–296.

[146] Lingyu Wang et al. *Security risk analysis of enterprise networks using probabilistic attack graphs*. Springer, 2017.

[147] Quan Wang et al. “Knowledge graph embedding: A survey of approaches and applications”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.12 (2017), pp. 2724–2743.

[148] Yinghui Wang et al. “Automotive Cybersecurity Vulnerability Assessment Using the Common Vulnerability Scoring System and Bayesian Network Model”. In: *IEEE Systems Journal* (2022).

[149] Haohuang Wen, Qi Alfred Chen, and Zhiqiang Lin. “Plug-N-pwned: Comprehensive vulnerability analysis of OBD-II dongles as a new over-the-air attack surface in automotive IoT”. In: *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2020, pp. 949–965.

[150] Samuel Woo, Hyo Jin Jo, and Dong Hoon Lee. “A practical wireless attack on the connected car and security protocol for in-vehicle CAN”. In: *IEEE Transactions on intelligent transportation systems* 16.2 (2014), pp. 993–1006.

[151] *XSB: A Logic Programming and Deductive Database system*. `https://xsb.sourceforge.net/`.

[152] Guoming Zhang et al. “Dolphinattack: Inaudible voice commands”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 103–117.

[153] Jingyi Zhang et al. “Vision-language models for vision tasks: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[154] Xinyang Zhang et al. “Interpretable Deep Learning under Fire”. In: *Proc. USENIX Security Symposium*. 2020.

[155] Husheng Zhou et al. “Deepbillboard: Systematic physical-world testing of autonomous driving systems”. In: *arXiv:1812.10812* (2018).

[156] Yuan Zhou et al. “Specification-based Autonomous Driving System Testing”. In: *IEEE Transactions on Software Engineering* (2023).

[157] Yi Zhu et al. “Can we use arbitrary objects to attack lidar perception in autonomous driving?” In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 1945–1960.