Strategic Safety-Critical Attacks Against an Advanced Driver Assistance System

A Technical Report submitted to the Department of Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Haotian Ren

Spring, 2022

Technical Project Team Members

Xugui Zhou

Anna Schmedding

Lishan Yang

Philip Schowitz

Evgenia Smirni

Homa Alemzadeh

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Keith Williams, Charles L. Brown Department of Electrical and Computer Engineering

Strategic Safety-Critical Attacks Against an Advanced Driver Assistance System

Xugui Zhou, Anna Schmedding†, Haotian Ren, Lishan Yang†, Philip Schowitz†, Evgenia Smirni†, Homa Alemzadeh
University of Virginia, Charlottesville, VA 22904 {xz6cz, hr3xw, ha4d}@virginia.edu
†William & Mary, Williamsburg, VA 23187 {akschmed, lyang11, philips, esmirni}@cs.wm.edu

***Abstract*—A growing number of vehicles are being transformed into semi-autonomous vehicles (Level 2 autonomy) by relying on advanced driver assistance systems (ADAS) to improve the driving experience. However, the increasing complexity and connectivity of ADAS expose the vehicles to safety-critical faults and attacks. This paper investigates the resilience of a widely-used ADAS against safety-critical attacks that target the control system at opportune times during different driving scenarios and cause accidents. Experimental results show that our proposed Context-Aware attacks can achieve an 83.4% success rate in causing hazards, 99.7% of which occur without any warnings. These results highlight the intolerance of ADAS to safety-critical attacks and the importance of timely interventions by human drivers or automated recovery mechanisms to prevent accidents.**

***Index Terms*—Attack, Fault injection, Hazard Analysis, CPS, Safety Validation, Autonomous Vehicle, ADAS**

I. Introduction

Over 3.5 million passenger cars worldwide are equipped with level 2 autonomous driving features such as Automated Lane Centering (ALC), Adaptive Cruise Control (ACC), and lane change assistance [1], [2]. With level 2 autonomy, the human driver must always be ready to take over the control of the car at any time. Many past studies have shown that unforeseen faults and/or malicious attacks can cause unsafe operation of the autonomous driver assistance systems (ADAS) with catastrophic consequences [3]–[8].

There are a variety of vulnerable components within a vehicle that can be targets for attacks, including Electronic Control Units (ECUs), sensors, in-vehicle networks, and V2X (Vehicle-to-Everything) communication [9]–[12]. Protecting the in-vehicle communication networks is of particular importance because they transmit sensor data, actuator commands, and other safety-critical information among various components. For example, some ADAS (e.g., OpenPilot from Comma.ai [13]) are integrated with the control system of existing vehicles by tapping into the Controller Area Network (CAN) bus interface through the On-Board Diagnostics (OBD) II port [14]. Additionally, critical information is shared through publisher-subscriber messaging systems, such as ROS [15], [16] or Cereal [17] which are shown to be vulnerable to a variety of attacks [18]. Despite efforts towards protecting these communication channels using techniques such as encryption and intrusion detection [19], [20], these protections either cannot detect attacks of a specific type or frequency or are not implemented in most vehicles on the road due to computational costs and the real-time constraints of the vehicle control systems [11], [21]–[23].

Recent works on autonomous vehicle (AV) safety and security have focused on assessing the impact of hardware faults and physical attacks on the ML accelerators [24] and inputs [3], [8], sensor attacks [25], [26], and attacks targeting the controller [27], [28]. But less attention has been paid to targeted safety-critical attacks on the ADAS output and actuator commands sent over the vulnerable communication channels that might go undetected by the existing safety mechanisms or cannot be acted on by a human driver.

More recently, studies have shown the benefit of contextual information and dynamic AV models [3], [5], [6] in designing effective fault injection and attack strategies that result in high hazard coverage. Machine learning (ML) methods such as Bayesian networks [5], neural networks [6], and reinforcement learning [29] are used to explore the fault parameter space and identify the most salient fault and attack scenarios with adverse impacts on safety. However, such approaches depend on large amounts of data from random fault injection experiments for model training.

In this paper, we use an orthogonal *model-driven* approach to the above *data-driven* techniques. Instead of focusing on exploring the entirety of the fault parameter space, we focus on a systematic characterization of the effect of the values of the parameter space (e.g., start time and duration of faults) *in conjunction with* the dynamic state of the vehicle to identify the most opportune system contexts to launch the attacks. We propose a Context-Aware safety-critical attack that can find *the most critical context* during a driving scenario to activate attacks that *strategically corrupt* the ADAS outputs, with the goal of (1) maximizing the chance of hazards and (2) causing hazards as soon as possible, before being detected/mitigated by the human driver or the ADAS safety mechanisms. We base this approach on the high-level control-theoretic hazard analysis [30] and specification of context-dependent safety requirements [3], [31] for a typical ADAS, which is applicable to any ADAS with the same functional and safety specifications.

We assess the resilience of OpenPilot [13], a widely-used ADAS, against such safety-critical attacks by demonstrating that system hardware and software components can be exploited, and actuator commands can be strategically modified to implement such malicious attacks and cause highly effective targeted hazards and accidents such as collision with other vehicles or road-side objects.

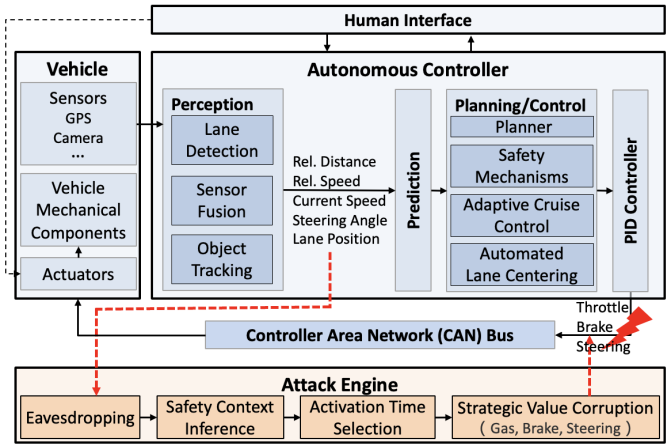


Fig. 1: Overview of the control structure of an ADAS with ACC and ALC, and the proposed attack engine.

Our study shows that the proposed Context-Aware strategy judiciously selects the most opportune start times and durations for attacks and is efficient in exploiting the safety-critical states of ADAS. We also find that lane invasions are common and can happen even without injecting faults, that the forward collision warning is not activated at all during attacks, and that the *steering angle* is the most vulnerable target. Our experimental results further highlight the importance of driver alertness for timely intervention and hazard prevention and the importance of robust automated safety mechanisms at the latest computational stage, just before execution on actuators.

II. Preliminaries

A. Advanced Driver Assistance Systems: OpenPilot

Fig. 1 presents the overall structure of an ADAS that includes sensing, perception, prediction, planning, and control units. ADAS features such as ALC and ACC rely on machine learning (ML) to perceive the vehicle state and surrounding environment using different sensors (e.g., camera, RADAR, GPS) [32]. Planning and control algorithms are used to adjust gas, brake, and steering to achieve target speed while keeping a safe distance from other vehicles, road lanes, and objects on the road.

OpenPilot is an alpha quality open-source ADAS that performs the functions of ALC and ACC for over 150 supported car makes and models, including Honda, Toyota, Hyundai, Nissan, Kia, Chrysler, Lexus, Acura, Audi, and VW [13]. In the past six years, over 1,500 monthly active users have collectively driven more than 40 million autonomous miles using OpenPilot. User driving data (e.g., camera, CAN, GPS, operating system logs) are collected by Comma.ai [33] and are used to train and test machine learning models.

Safety Mechanisms: OpenPilot is designed as a fail-safe passive system that requires the driver to be alert at all times. To enforce driver alertness, it also provides a monitoring feature that warns (jolts) the driver when distracted.

The following safety principles (as required by international standards, e.g., ISO 22179) are incorporated into the design of OpenPilot to ensure that the vehicle does not alter its trajectory too quickly, therefore allowing the driver to safely react [34]:

- The maximum acceleration *limit* is set to 2m/s² and maximum deceleration is set to -3.5m/s².
- There is a 1-second delay before the vehicle significantly deviates from its original path (e.g., crosses lane lines), allowing the driver time to react to an erroneous steering command.
- The driver can override OpenPilot with minimal effort, i.e., less than 3Nm extra torque on the steering wheel.

OpenPilot and the firmware used in some of the car models controlled by OpenPilot also implement additional automated safety mechanisms such as Forward Collision Warning (FCW) [35] and Autonomous Emergency Braking (AEB) [36].

B. Cyber-Physical System Context

An ADAS is designed by the tight integration of cyber and physical components with a human in the loop. Safety, as an emergent property of Cyber-Physical Systems (CPS), is context-dependent and should be controlled by enforcing constraints on the system behavior and control actions given the overall system state [31], [37]. In every control cycle t, an ADAS uses sensor measurements to estimate the physical system state x_t (e.g., current speed, relative distance to lead vehicle) and decides on a control action, u_t, from a finite set of high-level control actions (e.g., *Acceleration*, *Deceleration*, or *Steering*). The high-level control actions issued by ADAS are then translated by a low-level controller into control commands (e.g., *gas* and *brake*) which are sent to the actuators. Upon execution of the control command by the actuators, the physical system transitions to a new state x_{t+1}.

A sequence of *cyber control actions* $U_t = \{u_{t-k+1}, ..., u_{t-1}, u_t\}$ issued in k consecutive control cycles is unsafe if upon its execution in a given state sequence $X_t = \{x_{t-k+1}, ..., x_{t-1}, x_t\}$, the system eventually transitions to a state $x_{t'}$ that is hazardous [31] (e.g., too close to the lead vehicle). Thus, both the current system state and the control commands issued by ADAS contribute to the vehicle safety status. We use this insight to design a Context-Aware safety-critical attack that *infers the most critical states* during vehicle operation to *strategically corrupt the control commands* that are sent to the actuators.

Previous studies have shown that there is often a time gap between the activation of faults and the final propagation of unsafe control commands to the physical layer, resulting in hazards [31], [38]. We define the time between activation of an attack to the occurrence of a hazard as *Time-to-Hazard (TTH)* which indicates the maximum time budget for detecting anomalies and engaging in mitigation actions (see Fig. 2).

The *Driver Reaction Time* is defined as the time difference between the perception of an alert or anomaly (e.g., seeing an alert raised by the ADAS or recognizing an anomaly) and the start of physically taking an action (e.g., hitting the brake). In the AV literature, the overall driver reaction time (perception and reaction) is reported to be 2.5 seconds on average [8], [39]. We define the *Mitigation Time* as the time it

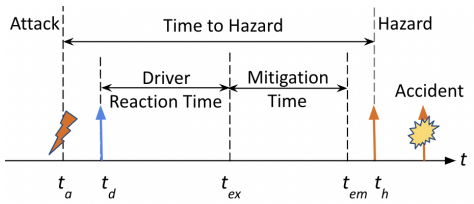


Fig. 2: Timeline of attack propagation. (t_a: Attack activated; t_d: Attack is detected by the ADAS or the anomaly is sensed by the human driver; t_{ex}: Human driver starts to engage; t_{em}: End of mitigation; t_h: Hazard occurs.)

takes for any corrective actions (e.g., braking) to be completed. This timing provides a window of opportunity for attackers to cause hazards before being overruled by the human driver or automated safety mechanisms. Fig. 2 shows an example where mitigation successfully completed before the occurrence of the hazard ($t_{em} < t_h$). A successful attack should evade detection and/or lead to hazards before the ADAS or the driver engage ($t_h < t_{ex}$) or complete any mitigation actions ($t_{ex} < t_h < t_{em}$).

III. Technical Approach

This section describes the proposed Context-Aware attack strategy and the capabilities and actions needed by an attacker to implement it on OpenPilot.

A. Context-Aware Attack Strategy

The attacker's goal is to manipulate the control commands, including gas, brake, and steering angle, to maximize the chance of hazard occurrence (e.g., violating longitudinal or lateral safety distance) while avoiding detection by the ADAS safety mechanisms and the human driver. The attacker is interested in causing one of the following accidents:

- **A1:** Collision with the lead vehicle.
- **A2:** Rear-end collision, causing traffic congestion.
- **A3:** Collision with road-side objects or other vehicles in the neighboring lane.

by forcing the system to transition into one of the following hazardous states:

- **H1:** AV violates safe following-distance constraints with the lead vehicle, which may result in A1.
- **H2:** AV decelerates to a complete stop although there is no lead vehicle, which may lead to A2.
- **H3:** AV drives out of lane, which may lead to A3.

To increase the chance of hazards, we adopt a control-theoretic hazard analysis method [30] to identify the specific combinations of system states and control actions that most likely lead to hazards. Table I shows an example context table that describes unsafe system contexts, including the specific high-level system context under which specific types of control actions may be unsafe and lead to safety hazards. For example, row 1 states that when the Headway Time is less than a safety limit t_{safe} (e.g., 2s), and the AV speed is faster than that of the lead vehicle ($RS > 0$), an $Acceleration$ control action is unsafe as it will result in a forward collision with the lead vehicle. This high-level identification of context-dependent unsafe control actions can be done by an attacker based on the knowledge of typical functionality of an ADAS and be applied to any ADAS with the same functional specification. The unknown thresholds t_{safe}, β_1 and β_2 can be specified based on domain knowledge or past data [31].

TABLE I: Safety context table for an ADAS with ALC and ACC

Rule	System Context	Control Action	Potential Hazard
1	$HWT \leqslant t_{safe} \wedge RS > 0$	u_1	H1
2	$HWT > t_{safe} \wedge RS \leqslant 0 \wedge Speed > \beta_1$	u_2	H2
3	$d_{left} \leqslant 0.1m \wedge Speed > \beta_2$	u_3	H3
4	$d_{right} \leqslant 0.1m \wedge Speed > \beta_2$	u_4	H3

* HWT: Headway Time = Relative Distance/Current Speed;
* RS: Relative Speed = Current Speed - Lead Speed;
* d_{left}, d_{right}: Distance to the left/right edge of current lane;
* $u_{1,2,3,4}$: Acceleration, Deceleration, Steering Left, Steering Right.
* $t_{safe} \in [2,3]s$, $\beta_1, \beta_2 \in [20,35]mph$

Our proposed Context-Aware attack strategy uses the critical system contexts described in Table I as the trigger for injecting unsafe control commands [40]. To evade detection, the control actions generated by the attack must be within the limits that are not noticeable to a human operator and are checked by the ADAS safety mechanisms, while minimizing the Time to Hazard (TTH) (see Fig. 2) and maximizing the chance of resulting in any hazards. To achieve these goals, the following optimization problem is formulated:

$$minimize_{_{TTH}} max\{Pr\{x_{t_{+TTH}} \in Hazardous\}\} \quad (1)$$

$$s.t.\ brake \geq limit_{brake}$$

$$accel \leqslant limit_{accel}$$

$$\Delta steering < limit_{steer}$$

$$\hat{v}_{t+1} \leqslant 1.1 v_{cruise}$$

$$\hat{v}_{t+1|t} = \hat{v}_t + accel * \Delta t \quad (2)$$

$$\hat{v}_{t+1} = \hat{v}_{t+1|t} + K_t * (v_{t+1} - \hat{v}_{t+1|t}) \quad (3)$$

where $brake$, $accel$, and $steering$ indicate the modified values of control commands, and $\hat{v}_{t+1}$ represents the predicted speed of the Ego vehicle at the next time step, which can be estimated using Eq. 2 that approximates the dynamics of the vehicle by assuming linear acceleration for a short time period Δt (10 ms). A Kalman filter [41] (with the Kalman Gain parameter K_t, see Eq. 3) is used to update the estimation using the measured speed v_{t+1} at the next time step. $limit_{accel}$, $limit_{brake}$, $limit_{steer}$ are the constraints on the output control commands defined by the safety checking rules of the target vehicle, including those of its ADAS.

B. Attack Model

To implement the Context-Aware attack strategy, the attacker needs 1) access to the sensor measurements and/or information shared through the in-vehicle communication network to estimate the current system state and 2) the capability of modifying the actuator commands with faulty values. Possible entry points for executing such malicious actions include the wireless networks [42], in-vehicle networks (e.g., CAN, FlexRay, Ethernet, Bluetooth, or telematics devices) [11], [43], the passive keyless entry and start system [44], the vehicle to everything communication, and/or vulnerable components supplied by different vendors [45], [46]. The attackers can gather information about the system configuration by monitoring and decoding the communication traffic and can identify potential vulnerabilities through publicly available documents such as open-source code [13].

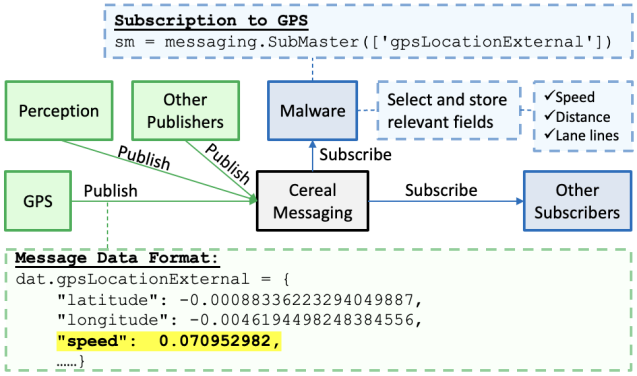


Fig. 3: Cereal messaging eavesdropping.

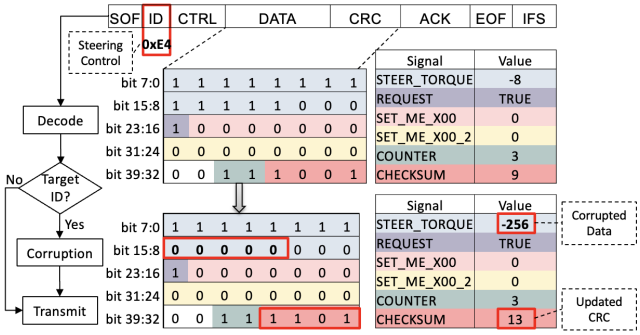


Fig. 4: An example of changing a steering output CAN message.

For example, the attack can be designed based on offline code/data analysis to infer the safety constraints and parameters described in Equations (1)-(3). It can then be implemented as a malware which is deployed via compromising over-the-air updates and, thus, can be activated on multiple cars running the same ADAS to maximize damage. Once deployed, the malware injects malicious commands at critical times by strategically selecting unsafe values to maximize the chance of evading detection by ADAS safety mechanisms while avoiding mitigation by the driver, i.e., hazards occur within a period that is shorter than the driver reaction time.

C. Attack Procedure

The overall procedure and steps for executing Context-Aware attacks are summarized as follows:

Eavesdropping: This step is accomplished by listening to the sensor sockets and the in-vehicle communication network, decoding the messages passed among different software components, and extracting the sensor data and critical state information. In OpenPilot, this can be achieved through local or remote subscriptions to the messaging system used for internal packet communication, called Cereal [17]. Cereal is a publisher-subscriber messaging specification for robotic systems (similar to ROS [15]), which is used for publishing messages by sensing and perception modules (e.g., GPS, Radar) and can be subscribed to by other OpenPilot modules (e.g., ACC, ALC) and any malicious software (see Fig. 3).

Since OpenPilot is open-source, the format of cereal messages is publicly available [47]. An example of eavesdropping on the GPS messages is shown in Fig. 3. To extract the information needed for safety context inference, the attacker needs to subscribe to the following events: 1) "*gpsLocationExternal*" events to learn the speed of the Ego vehicle published by GPS; 2) "*modelV2*" events to receive messages from the perception module to learn the lane line positions; 3) "*radarState*" events published by the RADAR to learn the relative speed and distance of the lead vehicle.

Safety Context Inference: Next, the attacker uses the basic state information x_t, including the speed of the Ego vehicle, the vehicle's lateral position, the lane line positions, and the relative distance to the lead vehicle, to infer the more complex and human-interpretable state variables described in the safety specification (Table I). For example, the headway time (HWT) is an important metric for identifying critical system context and can be calculated based on the Ego vehicle's current speed and relative distance to the lead vehicle.

Attack Type and Activation Time Selection: A context matcher detects whether the current system state matches any of the critical system contexts specified in Table I. If a matching case is found, the attack engine decides on the attack action (e.g., *Acceleration*) based on the unsafe action specified for the context and activates the attack. Table II lists the attack types to be activated based on different contexts. If two different context conditions are simultaneously detected, both control actions (e.g., *Acceleration* and *Steering*) are activated.

Strategic Value Corruption: In the last step, the selected attack type (e.g., *Acceleration*) is translated into low-level control commands (e.g., maximum *gas* and zero *brake*). The attack engine dynamically corrupts the control command values to not exceed the safety limits checked by OpenPilot's safety mechanisms (see Eq. 1-3). These safety limits are identified and encoded offline based on open-source code and publicly available documentations. Table III shows the specific safety limits we used for the attack types shown in Table II.

Finally, the faulty commands are sent to the target actuators by manipulating CAN messages. The information in a CAN bus message can be decoded using reverse engineering and the open-source Database Container (DBC) [48] configuration [49] of a specific car model. The attack engine then corrupts the specific CAN message that carries a target control command using the command's unique identifier (e.g., 0xE4 for steering as shown in Fig. 4). The attacker also updates the checksum after corrupting targeted control commands, so the integrity of the corrupted CAN message is maintained.

IV. Experiments

To evaluate the effectiveness of the proposed Context-Aware attack, we develop a simulation platform consisting of the OpenPilot control software integrated with the CARLA urban driving simulator [50], a driver reaction simulator, and a software-implemented fault injection engine. The platform architecture is shown in Fig. 5 and is described next.

The OpenPilot safety mechanisms (see Section II-A) are implemented in its control software and the Panda CAN interface. Panda is a universal OBD adapter developed by Comma.ai [51] that provides access to almost all car sensors

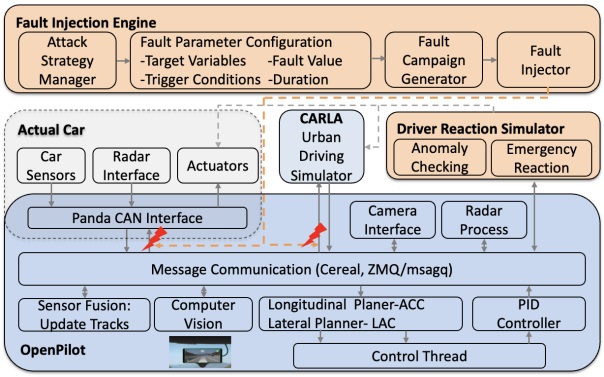


Fig. 5: Overall architecture of OpenPilot, integrated with CARLA, the driver reaction simulator, and the fault injection engine. [Online Available: https://github.com/UVA-DSA/openpilot-CARLA].

through the CAN bus. When integrated with the CARLA driving simulator, the Panda software and hardware are not utilized by OpenPilot. Therefore, Panda safety checks are not enforced. Here, we consider all safety limits checked by Panda as constraints for generating faulty values for the Context-Aware attacks so that they evade detection by Panda when it is engaged in actual driving (Eq. 1).

Our experiments are done on Ubuntu 20.04 LTS, with OpenPilot v0.8.9 and CARLA v9.11. A single simulation of OpenPilot contains 5000 time-steps, each step lasts about 10 ms, which in total equals 50 seconds.

A. Driving Scenarios

Using the CARLA simulator, we create different driving scenarios where the Ego vehicle, cruising at 60mph from 50, 70, or 100 meters away, approaches a lead vehicle with different behaviors as follows.

- S1: Lead vehicle cruises at the speed of 35 mph;
- S2: Lead vehicle cruises at the speed of 50 mph;
- S3: Lead vehicle slows down from an initial speed of 50 mph to 35 mph;
- S4: Lead vehicle accelerates from an initial speed of 35 mph to 50 mph.

Fig. 6(a-b) show different views of a simulated scenario.

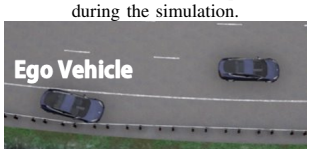
B. Driver Reaction Simulator

To mimic the situation where the human driver takes over the control of the vehicle in an emergency situation,

(a) An example initial position of Ego Vehicle (EV) and other reference vehicles.

(b) The user interface of OpenPilot during the simulation.

(c) EV collides with the lead vehicle.



(d) EV collides with the guardrail.

Fig. 6: Driving scenarios in OpenPilot.

TABLE II: Fault injection experiments

Attack Type	Accel	Brake	Steering Angle	No. Attacks
Acceleration	$limit_{accel}$	0		60
Deceleration	0	$limit_{brake}$		60
Steering-Left			$-limit_{steer}$	60
Steering-Right			$limit_{steer}$	60
Acceleration-Steering	$limit_{accel}$	0	$\pm limit_{steer}$	60
Deceleration-Steering	0	$limit_{brake}$	$\pm limit_{steer}$	60

we designed a driver reaction simulator (see Fig. 5). The simulated driver is alerted when the ADAS raises any safety alarms (e.g., FCW) or the driver observes any anomalies in the vehicle status that last for a noticeable period of time (i.e., at least 1 second). Anomalies include hard brake ($|Brake| > |limit_{brake}|$), unexpected increase in acceleration ($Accel > limit_{accel}$) or steering ($Steering > limit_{steer}$), or the vehicle speed exceeding cruising speed by more than 10% ($Speed > 1.1 v_{cruise}$). To make the attack more challenging, anomalies that occur within one simulation time step (10ms) attract the driver's attention. The driver physically takes action after 2.5 seconds (the average driver reaction time) to process an alert or anomaly [39]).

Typically human drivers respond to sudden unintended acceleration with a hard brake within 1.5 seconds. We model this using an exponential function that approximates the general brake curve as follows [52]:

$$brake = e^{10t-12}/(1+e^{10t-12}) \quad (4)$$

We also apply the same reaction to sudden steering. The attack engine stops the attack as soon as the driver engages.

C. Attack Types

For each driving scenario, we simulate six types of attacks (see Table II) by injecting faults into each output variable as well as their combinations. For example, for the *Acceleration-Steering* attack, we inject faults to *Gas* and *Steering Angle* (either left or right angle). We limit the injected values within the acceptable ranges by the OpenPilot control software. Each scenario is tested with three different initial positions for the lead vehicle and is repeated 20 times to capture variations due to changes in the simulated driving environment and attack timings. This results in 60 (20×3) simulations per attack type and a total number of 1,440 ($60 \times 6 \times 4$) for all simulated attacks and driving scenarios.

D. Baselines

In addition to the Context-Aware strategy, we design three baseline strategies to test the ADAS resilience to different attacks (see Table III). The first baseline (referred to as *Random-ST+DUR*) uses a random start time uniformly distributed within [5, 40] seconds (5 seconds after the start of simulation

TABLE III: Overview of attack strategies

Attack Strategy	Start Time	Duration	Attack Values	No. Attacks
Random-ST+DUR	Uniform [5,40]s	Uniform [0.5,2.5]s	Fixed[1]	14,400
Random-ST	Uniform [5,40]s	2.5s	Fixed	1,440
Random-DUR	Context-Aware	Uniform [0.5,2.5]s	Fixed	1,440
Context-Aware	Context-Aware	Context-Aware	Strategic[2]	1,440

[1] Fixed: use the maximum limit of each output command defined in OpenPilot: $limit_{steer} = 0.5°$, $limit_{brake} = -4m/s^2$, $limit_{accel} = 2.4m/s^2$.
[2] Strategic: dynamically choose the attack value according to Eq. 1-3 ($limit_{steer} = 0.25°$, $limit_{brake} = -3.5m/s^2$, $limit_{accel} = 2m/s^2$).

TABLE IV: Attack strategy comparisons with an alert driver.

Attack Strategy	Alerts	Hazards	Accident	Hazards& no Alerts	LaneInvasion (No. Event/s)	TTH(s) (Avg. ± Std.)
No Attacks	2 (0.1%)	0	0	0	0.46	
Random-ST+DUR	3248 (22.6%)	5727 (39.8%)	3293 (22.9%)	3083 (21.4%)	1.03	1.61±1.96
Random-ST	346 (24.0%)	771 (53.5%)	516 (35.8%)	474 (32.9%)	0.68	1.49±0.73
Random_DUR	210 (14.6%)	388 (26.9%)	332 (23.1%)	229 (15.9%)	0.46	1.92±1.17
Context-Aware	4 (0.3%)	1201 (83.4%)	641 (44.5%)	1197 (83.1%)	0.66	2.43±1.29

till 10 seconds before the end), with attack duration uniformly distributed within [0.5, 2.5] seconds. We run *Random-ST+DUR* strategy for 14,400 simulations to maximize coverage of the critical attack parameters. For the second baseline (referred to as *Random-ST*), we randomly choose a start time but fix the attack duration to be equal to the average driver reaction time (2.5 seconds). To test the relationship between hazards and attack duration, we also design a third baseline (referred to as *Random-DUR*) by randomly choosing the attack duration from a range of [0.5, 2.5] seconds with the start time inferred based on context. All the attack values are within the range of OpenPilot safety checks. Note that aggressive random attacks (e.g., bombarding the CAN-bus with out-of-the-range values) may get detected by existing intrusion detection mechanisms for in-vehicular networks [11], [19] and the OpenPilot safety checks, so they are not considered here.

E. Results

1) System Resilience Evaluation: We evaluate the resilience of OpenPilot in presence of an alert driver by running the simulations with and without the attacks. Table IV shows that under normal system operation, when no attacks are engaged, no hazards or accidents occur. However, 2 *steer saturated* alerts were raised due to the steering angle exceeding the predefined safety limits in OpenPilot. Fig. 7 shows an example of the performance of the ALC system. We observe that the ALC system does not keep the Ego vehicle in the center of the lane at all times, and lane invasions occur with an average frequency of 0.46 times per second, which can lead to out-of-lane hazards or collision with road-side objects. This indicates that the ALC and ACC systems do not cooperate well, which is a defect in the control software and needs to be fixed.

Observation 1: Lane invasions can happen even without any attacks.

2) Comparison to Random Attack Strategies: Table IV also shows that the Context-Aware strategy outperforms the three random strategies and achieves the highest hazard coverage of 83.4%, with 99.7% (1197/1201) of hazards occurring without any alerts. Note that 53.4% (641/1201) of hazards result in accidents, including collision with the lead vehicle and road-side objects (see Fig. 6(c-d)). In these cases, the alert raised by ADAS is the *steer saturated* warning, while the more relevant

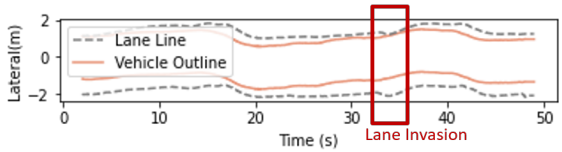


Fig. 7: Trajectory of the Ego Vehicle during an attack-free simulation.

forward collision warning (FCW) is not activated as the brake output is kept less than the safety threshold of OpenPilot. We also observe an increased number of lane invasions per second for almost all attacks due to the occurrence of out-of-lane hazards. Despite achieving the highest hazard coverage, the Context-Aware attacks keep the number of lane invasions and alerts low because of the strategic value corruption.

Observation 2: The Context-Aware attack strategy is efficient in exploiting safety critical states of ADAS. During attacks, the *forward collision* warning does not get activated at all.

From Table IV we also see that the average TTH of Context-Aware attack is larger than the Random attacks due to a higher hazard rate in *Acceleration* attack that has a longer TTH.

3) Evaluation of Attack Duration and Start Time: To further evaluate the importance of attack *duration* and *start* time, we assess the coverage of the fault parameter space by different attack strategies. Fig. 8 illustrates a sample parameter space for durations between 0.5 to 2.5 seconds and start times between 5 to 35 seconds for the *Acceleration* attack type. Each dot in this figure represents an attack simulation. The solid dots represent simulations with hazards. This figure illustrates that an attack does not result in any hazard if not activated within a critical time window (after dashed line at about 24-25 seconds), regardless of how long the attack lasts. After finding the critical launch moment, the attack needs to last for a time period (at least 1.5 seconds) to cause a hazard. Therefore, it is important to find both the opportune time to start an attack and the required duration to increase the hazard success rate.

We also observe that the Context-Aware strategy (marked by orange diamonds) are all solid (hazardous) and located within the critical time window. The dots that correspond to Random-ST and Random-DUR strategies result in a significant number of non-hazardous cases. This figure further attests the efficiency of the proposed Context-Aware strategy.

Observation 3: Context-Aware selection of start time and duration does not waste resources on non-hazardous random injections.

4) Evaluation of the Strategic Value Corruption: In this set of experiments, we further evaluate the performance of the Context-Aware strategy with and without the strategic

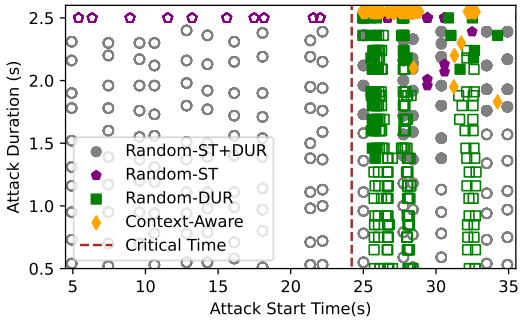


Fig. 8: State space of "Attack start time" and "Duration" for *Acceleration* attacks (solid shapes correspond to hazardous results and empty ones to non-hazardous).

TABLE V: Context-aware attack with or without strategic value corruption and with an alert driver.

Attack Type	No Strategic Value Corruption								With Strategic Value Corruption				
	Alerts	Hazards	Accident	TTH(s) (Avg. ± Std.)	Prevented Hazards	New Hazards	Prevented Accidents	Reduced Accidents	Alerts	Hazards	Accident	TTH(s) (Avg. ± Std.)	Driver Prevention*
Acceleration	4 (1.7%)	200 (83.3%)	120 (50.0%)	3.33±0.23	200 (83.3%)	160 (66.7%)	200 (83.3%)	120 (50%)	1 (0.4%)	160 (66.7%)	160 (66.7%)	5.03±1.22	1/1
Deceleration	1 (0.4%)	99 (41.2%)	0 (0.0%)	2.62±0.04	141 (58.8%)	0	0	0	0 (0.0%)	231 (96.2%)	0 (0.0%)	2.77±0.10	0/0
Steering-Left	122 (50.8%)	187 (77.9%)	175 (72.9%)	1.11±0.86	0	0	0	0	1 (0.4%)	90 (37.5%)	1 (0.4%)	1.33±0.17	0/0
Steering-Right	2 (0.8%)	240 (100.0%)	240 (100.0%)	1.63±0.08	0	0	0	0	0 (0.0%)	240 (100.0%)	240 (100.0%)	1.39±0.10	0/0
Acceleration-Steering	2 (0.8%)	240 (100.0%)	240 (100.0%)	1.51±0.15	0	0	0	-1 (0.4%)	2 (0.8%)	240 (100.0%)	240 (100.0%)	1.47±0.26	0/0
Deceleration-Steering	3 (1.2%)	138 (57.5%)	17 (7.1%)	2.63±0.02	170 (70.8%)	68 (28.3%)	0	-17 (7.1%)	0 (0.0%)	240 (100.0%)	0 (0.0%)	2.77±0.06	0/0
Total	142 (9.9%)	1104 (76.6%)	792 (55.0%)	2.04±1.10	511 (36.8%)	228 (16.4%)	200 (22.4%)	102 (11.4%)	4 (0.3%)	1201 (83.4%)	641 (44.5%)	2.43±1.29	1/1

* The number of hazards/accidents prevented when a human driver simulator is added in the simulation.

value selection. Note that the attacks without strategic value corruption may be detected by Panda's safety checks, if deployed on an actual vehicle. But Panda's safety checks could also be disabled by the attacker or bypassed if the attack is launched after the safety checks (e.g., on the OBD II port).

Table V shows the results across different attack types, including the number of hazards that are prevented by the driver. For timely hazard mitigation, the driver reaction/mitigation times should be shorter than the Time-to-Hazard (TTH) (See average TTHs in Table V). Our experiments indicate that without the driver reaction, the attacks without strategic value corruption could achieve very high hazard and accident success rates (almost 100% for all attack types; not shown in the table due to space limits). However, when simulating the human driver reaction, 83.3% of hazards are prevented for the *Acceleration* attack, reducing 50% of collision events. Similar hazard reductions are observed for the *Deceleration* (58.8%) and *Deceleration-Steering* (70.8%) attacks.

Observation 4: Human alertness for timely intervention is important in preventing hazards and accidents.

However, the driver reaction does not prevent *Steering* attacks (zero hazards prevented for steering attacks in Table V), and these attacks still achieve very high hazard and accident success rates (e.g., 100% for *Steering-Right* and *Acceleration-Steering*). This is because hazards happen in less than 1.63s, which is much less than the average human driver reaction time (2.5s), indicating that attacks targeting the steering angle are the most difficult to be mitigated by the driver. It should be noted that the *Steering-Left* attacks achieve lower success in causing hazards compared to *Steering-Right* attacks (77.9% vs. 100%) because the Ego vehicle is initialized to a lane closer to the right guardrail while it travels on a left-curved road.

Observation 5: Steering is the most effective attack type that cannot be easily halted by the human driver.

Although driver intervention reduces hazard and accident rates, it may also introduce new hazards. For example, to avoid a collision with the lead vehicle, the Ego vehicle may stop in the middle of a lane causing a rear collision, or collide with curb objects. Table V shows that up to 66.7% new hazards happened after preventing attacks on the gas output.

After adding the strategic value corruption, even though there is an overall 6.8% increase in hazard success rates (76.6% to 83.4%), the total number of alerts generated by the ADAS decreases to 4, and only less than 0.1% of the induced hazards are prevented by the driver, even for the cases where the average TTH is longer than the average driver reaction time (2.5s) (e.g., *Acceleration*, *Deceleration* and *Deceleration-Steering* attacks). This further illustrates the effectiveness of the Context-Aware strategy for evading detection by the ADAS and/or the human driver.

Observation 6: The strategic value corruption is effective in evading human driver detection and safety checks of ADAS.

V. Threats to Validity

Although our attack strategy is efficient in finding potential weaknesses in the ADAS control software, its robustness and efficacy might be affected by the quality of sensor data used for context inference or by existing defense mechanisms (e.g., control invariant detection [53] or context-aware monitoring [31]). Our simulations consider OpenPilot safety checks and human driver interventions. But other safety and security mechanisms that can be implemented in firmware or hardware interface of the car (e.g., Panda's safety checks, AEB, encryption, or intrusion detection) are not included in this study. Further evaluation of the robustness and detectability of the attacks are directions of future work.

VI. Conclusion

This paper presents a strategic Context-Aware attack that targets control commands within an ADAS. This attack finds the most critical times during a driving scenario to activate attacks, as well as the attack durations, which can cause hazards before a human driver or the ADAS safety mechanism can correct the behavior. The proposed attack is efficient since large numbers of random fault injections are not needed to guide the approach. Our experimental results and observations show that steering is particularly vulnerable and that the existing warning system for forward collisions is insufficient. Our results also highlight the importance of human alertness for timely intervention for preventing hazards and accidents, and the importance of automated safety mechanisms that can check the control actions issued by ADAS.

Acknowledgment

This work was partially supported by the Commonwealth of Virginia under Grant CoVA CCI: C-Q122-WM-02 and by the National Science Foundation (NSF) under Grant No. 1748737.

REFERENCES

[1] "Autonomous driving starts to hit mainstream as 3.5 million new cars had level 2 features in q4 2020?" 2021. [Online]. Available: https://canalys-prod-public.s3.eu-west-1.amazonaws.com/static/press_release/2021/CanalysAutomediaalertQ420L2WW.pdf

[2] "SAE Levels of Driving Automation™ Refined for Clarity and International Audience," https://www.sae.org/blog/sae-j3016-update, 2021.

[3] A. H. M. Rubaiyat, Y. Qin, and H. Alemzadeh, "Experimental resilience assessment of an open-source driving agent," in *2018 IEEE 23rd Pacific rim international symposium on dependable computing (PRDC)*. IEEE, 2018, pp. 54–63.

[4] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 586–597.

[5] S. Jha, S. Banerjee, T. Tsai, S. K. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection," in *2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2019, pp. 112–124.

[6] S. Jha, S. Cui, S. Banerjee, J. Cyriac, T. Tsai, Z. Kalbarczyk, and R. K. Iyer, "Ml-driven malware that targets av safety," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 113–124.

[7] National Transportation Safety Board, "Collision Between Vehicle Controlled by Developmental Automated Driving System and Pedestrian, Tempe, Arizona, March 18, 2018 - Accident Report," 2019, https://www.ntsb.gov/investigations/accidentreports/reports/har1903.pdf.

[8] T. Sato, J. Shen, N. Wang, Y. Jia, X. Lin, and Q. A. Chen, "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.

[9] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 447–462.

[10] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, vol. 4, no. 447-462. San Francisco, 2011, p. 2021.

[11] K. Kim, J. S. Kim, S. Jeong, J.-H. Park, and H. K. Kim, "Cybersecurity for autonomous vehicles: Review of attacks and defense," *Computers & Security*, vol. 103, p. 102150, 2021.

[12] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *black hat USA*, vol. 2014, p. 94, 2014.

[13] Comma.ai, "OpenPilot." [Online]. Available: https://github.com/commaai/openpilot

[14] K. McCord, *Automotive Diagnostic Systems: Understanding OBD I and OBD II*. CarTech Inc, 2011.

[15] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[16] M. Aeberhard, T. Kühbeck, B. Seidl, M. Friedl, J. Thomas, and O. Scheickl, "Automated driving with ros at bmw," *ROSCon 2015 Hamburg, Germany*, 2015.

[17] Comma.ai, "Cereal." [Online]. Available: https://github.com/commaai/cereal

[18] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, and P. Schartner, "Security for the robot operating system," *Robotics and Autonomous Systems*, vol. 98, pp. 192–203, 2017.

[19] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 911–927.

[20] J. Zhou, P. Joshi, H. Zeng, and R. Li, "Btmonitor: Bit-time-based intrusion detection and attacker identification in controller area network," *ACM Trans. Embed. Comput. Syst.*, vol. 18, pp. 117:1–117:23, 2020.

[21] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *Def Con*, vol. 21, no. 260-264, pp. 15–31, 2013.

[22] H.-G. Wahl and F. Gauterin, "An iterative dynamic programming approach for the global optimal control of hybrid electric vehicles under real-time constraints," in *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2013, pp. 592–597.

[23] Y. Luo, "Time constraints and fault tolerance in autonomous driving systems," Tech. rep, Tech. Rep., 2019.

[24] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[25] J. Petit and S. E. Shladover, "Potential cyberattacks on automated vehicles," *IEEE Transactions on Intelligent transportation systems*, vol. 16, no. 2, pp. 546–556, 2014.

[26] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao, "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.

[27] S. Iqbal, A. Haque, and M. Zulkernine, "Towards a security architecture for protecting connected vehicles from malware," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. IEEE, 2019, pp. 1–5.

[28] A. Ding, P. Murthy, L. Garcia, P. Sun, M. Chan, and S. Zonouz, "Mini-me, you complete me! data-driven drone security via dnn-based approximate computing," in *24th International Symposium on Research in Attacks, Intrusions and Defenses*, 2021, pp. 428–441.

[29] M. Moradi, B. J. Oakes, M. Saraoglu, A. Morozov, K. Janschek, and J. Denil, "Exploring fault parameter space using reinforcement learning-based fault injection," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 102–109.

[30] N. Leveson and J. Thomas, "An stpa primer," *Cambridge, MA*, 2013.

[31] X. Zhou, B. Ahmed, J. H. Aylor, P. Asare, and H. Alemzadeh, "Data-driven design of context-aware monitors for hazard prediction in artificial pancreas systems," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 484–496.

[32] A. Lima, F. Rocha, M. Völp, and P. Esteves-Veríssimo, "Towards safe and secure autonomous and cooperative vehicle ecosystems," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, ser. CPS-SPC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 59–70. [Online]. Available: https://doi.org/10.1145/2994487.2994489

[33] H. Schafer, E. Santana, A. Haden, and R. Biasini, "A commute in data: The comma2k19 dataset," 2018.

[34] "OpenPilot - Safety Architecture," 2018. [Online]. Available: https://blog.comma.ai/how-to-write-a-car-port-for-openpilot/#background--safety-architecture

[35] Comma.ai, "Bringing forward collision warnings to our open source self-driving car," 2019. [Online]. Available: https://comma-ai.medium.com/bringing-forward-collision-warnings-to-our-open-source-self-driving-car-7545b6e398cd

[36] "Comma.ai", "AEB: a Case Study Using comma.ai Dataset," 2019. [Online]. Available: https://comma-ai.medium.com/aeb-a-case-study-using-comma-ai-dataset-2fc08a2397f4

[37] N. Leveson, *Engineering a safer world: Systems thinking applied to safety*. MIT press, 2011.

[38] H. Alemzadeh, D. Chen, X. Li, T. Kesavadas, Z. T. Kalbarczyk, and R. K. Iyer, "Targeted attacks on teleoperated surgical robots: Dynamic model-based detection and mitigation," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 395–406.

[39] N. Gavin, K. David S., and G. Steve, "California commercial driver handbook," 2021. [Online]. Available: https://www.dmv.ca.gov/web/eng_pdf/comlhdbk.pdf

[40] N. Neves, J. Antunes, M. Correia, P. Verissimo, and R. Neves, "Using attack injection to discover new vulnerabilities," in *International Conference on Dependable Systems and Networks (DSN'06)*, 2006, pp. 457–466.

[41] G. Bishop, G. Welch *et al.*, "An introduction to the kalman filter," *Proc of SIGGRAPH, Course*, vol. 8, no. 27599-23175, p. 41, 2001.

[42] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking tesla from wireless to can bus," *Briefing, Black Hat USA*, vol. 25, pp. 1–16, 2017.

[43] B. R. Payne, "Car hacking: Accessing and exploiting the can bus protocol," *Journal of Cybersecurity Education, Research and Practice*, 2019.
[44] A. Francillon, B. Danev, and S. Capkun, "Relay attacks on passive keyless entry and start systems in modern cars," in *the Network and Distributed System Security Symposium (NDSS)*, 2011.
[45] P. Tyagi and D. Dembla, "Investigating the security threats in vehicular ad hoc networks (vanets): towards security engineering for safer on-road transportation," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2014.
[46] Z. Cai, A. Wang, W. Zhang, M. Gruffke, and H. Schweppe, "0-days & mitigations: Roadways to exploit and secure connected bmw cars," *Black Hat USA*, 2019.
[47] Comma.ai, "log.capnp." [Online]. Available: https://github.com/commaai/cereal/log.capnp
[48] autopi, "Can dbc file explained." [Online]. Available: https://www.autopi.io/blog/the-meaning-of-dbc-file/
[49] commaai, "Opendbc." [Online]. Available: https://github.com/commaai/opendbc
[50] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.
[51] Comma.ai, "Panda." [Online]. Available: https://github.com/commaai/panda
[52] J. G. Gaspar and D. V. McGehee, "Driver brake response to sudden unintended acceleration while parking," *Transportation Research Interdisciplinary Perspectives*, vol. 2, p. 100039, 2019.
[53] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Xinyan, "Detecting attacks against robotic vehicles: A control invariant approach," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 801–816.