SUFI: A Simulation-based Fault Injection Tool for Safety Evaluation of Advanced Driver Assistance Systems Modelled in SUMO

Mehdi Maleki and Behrooz Sangchoolie
Dependable Transport Systems
RISE Research Institutes of Sweden Borås, Sweden
{mehdi.maleki, behrooz.sangchoolie}@ri.se

***Abstract*—Embedded electronic systems used in vehicles are becoming more exposed and thus vulnerable to different types of faults and cybersecurity attacks. Examples of these systems are advanced driver assistance systems (ADAS) used in vehicles with different levels of automation. Failures in these systems could have severe consequences, such as loss of lives and environmental damages. Therefore, these systems should be thoroughly evaluated during different stages of product development. An effective way of evaluating these systems is through the injection of faults and monitoring their impacts on these systems. In this paper, we present SUFI, a simulation-based fault injector that is capable of injecting faults into ADAS features simulated in SUMO (simulation of urban mobility) and analyse the impact of the injected faults on the entire traffic. Simulation-based fault injection is usually used at early stages of product development, especially when the target hardware is not yet available. Using SUFI we target car-following and lane-changing features of ADAS modelled in SUMO. The results of the fault injection experiments show the effectiveness of SUFI in revealing the weaknesses of these models when targeted by faults and attacks.**

***Keywords*-fault injection, attack injection, advanced driver assistance systems, SUMO;**

I. Introduction

Embedded electronic systems are increasingly used in vehicles with different levels of automation [1]. These vehicles are equipped with systems such as advanced driver assistance systems (ADAS) that come with features such as lane-changing and car-following. Enablers of these features are the different sensors and cameras as well as the connectivity provided by vehicle-to-vehicle and vehicle-to-infrastructure communications. The increasing use of these technology enablers also result in ADAS to be more exposed and thus vulnerable to different types of faults and cybersecurity attacks. As many of the ADAS features are safety-critical, failures of these systems could result in serious consequences, including loss of lives and environmental damage. Therefore, these systems should be thoroughly assessed before they are used in the traffic to make sure they are safe and secure.

Fault injection and attack injection are well-known testing techniques used for assessing system safety and security. Functional safety standards such as ISO 26262 [2] recommends, or highly recommends, the use of fault injection to prove that malfunctions in electrical and/or electronic systems will not lead to violations of safety requirements. Fault and attack injection can be conducted either through the field tests or simulation-based tests. While conducting field tests could be costly and sometimes life-threatening [3], [4], [5], simulation-based tests provide a wide range of advantages, such as lower testing costs, adaptation of tests to a variety of traffic scenarios, and avoiding the life-threatening situations.

Simulation-based fault injection can be performed by building injectors that could inject faults into simulators that model vehicle systems at component and traffic simulation levels. Examples of these simulators are SUMO [6] and CARLA [7] that are used to simulate the urban mobility and ADAS features. In an earlier study, Saurabh et al. [8] targeted the decision-making algorithms and sensor units of an autonomous vehicle in CARLA. The authors did not analyse the impact of fault on the surrounding traffic and limited their analysis focus on the behaviour-change of the target vehicle. Van der Heijden et al. [9] and Singh et al. [10] used SUMO [6] and Veins [11] to analyse the impact of the cybersecurity attacks on cooperative driving and vehicle connectivity. The focus of the analyses is only on the car-following behaviour of the vehicles whereas in this paper, we also investigate the lane-changing behaviour of the vehicles under faults and attacks.

In this paper, we introduce SUFI, a simulation-based fault injector that is capable of injecting faults into ADAS features simulated in SUMO. Using SUFI, one can evaluate the ADAS features by defining variety of traffic scenarios as well as to measure the impact of the injected faults on the entire traffic. The paper provides answers to the following questions:

1) To what extent could fault injection experiments be used to reveal weaknesses of driver assistance functions modelled in SUMO?
2) To what extent could the behaviour of the surrounding traffic be affected by injection of faults in a vehicle?
3) Are there fault models that are more effective in revealing weaknesses of the target systems?
4) To what extent does the fault injection location as well as the duration in which a system is exposed to faults reveal weaknesses of the target systems?
5) To what extent could the injection of faults in multiple locations reveal weaknesses of the target systems that are not revealed when facing with only one fault?
6) What is the impact of the cooperativeness on the vehicle behaviour when it is exposed to faults/attacks? Note that, here, cooperativeness refers to vehicles' cooperation in considering the nearby vehicles' behaviour for lane-change manoeuvres (see §III-A and §IV-D).

II. Background

A. SUMO (Simulation of Urban Mobility)

SUMO [6] is an open-source traffic simulation platform developed to simulate a wide range of traffic scenarios in *microscopic* and *macroscopic* scales. Microscopic offers the simulation of traffic in individual vehicle level e.g., vehicle position and speed, whereas, macroscopic offers the simulation in traffic flow level e.g, density. SUMO controls the vehicle behaviour with *car-following* and *lane-changing* models. The former controls the longitudinal behaviour of the vehicle, such as position, speed, and acceleration, while the latter represents the lateral behaviour of the vehicle which depends on the vehicle state and willingness to perform a lane-change.

SUMO allows the user to dynamically interact with the simulation objects such as vehicles, pedestrians and traffic lights during the simulation run-time. This is provided by a library called TraCI (Traffic Control Interface) [12] which communicates with SUMO via TCP/IP. We chose SUMO in this study due to its high-portability, fast computation speed, and inclusion of representative car-following models for ADAS features.

B. Simulation-based Fault and Attack Injection

Fault injection is an established method used for test and assessment of dependable computer systems. Fault injection has also been used to exploit properties such as cybersecurity [13], [14] through the injection of attacks; this is also known as attack injection. The"Row hammer" exploit in DDR3 DRAM [15] is a great example of when bit-flip faults in memory results in gaining kernel privileges on x86-64 Linux [16]. In this study, we use the simulation-based fault injection and introduce SUFI (SUMO-based Fault Injector) (see §III-E), where faults are injected into various driver assistance functions modelled in SUMO [6]. Simulation-based techniques inject faults into hardware/software models as opposed to physical techniques which uses actual physical systems or prototypes.

The simulation-based fault injection technique used in this paper allows us to design and evaluate traffic scenarios that are infeasible to build when conducting field testing due to their life threatening implications. Moreover, the technique allows us to extend the analysis of the impact of the injected faults from the *target vehicle* to the entire *surrounding traffic* and *traffic flow*. The technique is also cost-worthy as thousands of fault injection experiments could be conducted in a short period of time. However, mapping of the results to those obtained by evaluating real-world ADAS features is tightly connected to the accuracy of the features modelled in SUMO. Moreover, SUFI is capable of injecting faults/attacks in application level and does not make connections to the low-level hardware details. This is due to the unavailability of accurate models of the underlying hardware and high evaluation timing cost that will be introduced by such modellings.

C. Fault and Attack Models

The SUMO-based fault injector (see §III-E) introduced in this paper is capable of injecting different types of faults and attacks into the system under test. In this section, we present some of these fault and attack models. Note that additional models could be implemented and integrated into the injector.

Stuck-at-value In this model, the value stored in a location is stuck at a certain value. If the value is stuck permanently, it could be used to model manufacturing defects, such as the ones in sensors. However, if it is stuck temporarily, it could be used to model cybersecurity attacks such as the *replay* attacks (resending old data), which target freshness, non-repudiation and integrity of communication data or commands.

Single bit-flip In this model, one of the bits of the value stored in a target location is flipped. The single bit-flip has been used in the past for modelling soft errors occurring inside a processor's core or different pipeline registers [17], [18], [19]. In this study, single bit-flip is used to model cybersecurity attacks such as *corrupt message* (e.g., jamming attack), which target the integrity of communication data or commands.

Double bit-flip In this model, we flip two of the bits of the value stored in a target location simultaneously. The model has been used in the past e.g., to emulate hardware faults affecting two bits of an architectural register [20], [19], [21]. Similar to the single bit-flip model, in this study, the double bit-flip is used to model cybersecurity attacks such as the jamming attack, which target the integrity of communication data or commands. This model is also used to compare the results obtained for single and double bit-flip injections.

Chain-of-faults In this model, two faults are injected into two different locations. The first fault is injected into the ego-vehicle; the second fault is either injected into the ego-vehicle or in the follower-vehicle (see Fig. 1). This fault is injected either simultaneously with the first fault or with delays of 3, 5, and 7 seconds. Chain-of-faults in this study models the failure of two different units. It is also used to model cybersecurity attacks such as the *delay* attack, which target the integrity and non-repudiation of communication data or commands.

It is also important to model the duration in which a component is exposed to a fault. To do so, we model *transient* and *semi-permanent* faults. We use transient to model faults that only appear in the system for one simulation time step in order to model temporal cybersecurity attacks or unfavourable weather conditions affecting a sensor temporarily. The semi-permanent is used to model faults that remain in the system until the end of a simulation run such as long-lasting cybersecurity attacks or complete blockage of a sensor. Note that, we call this semi-permanent since the system is fault-free in the beginning of each experiment run as opposed to permanent faults that remain in the system until it is repaired.

III. Experimental Setup

A. Models Under Evaluation

1) Car-Following Model: The car-following model represents the longitudinal behaviour of vehicles in the simulation environment, where it controls the vehicle with or without a vehicle in front. Several car-following models are implemented in SUMO, such as Krauss [22], IDM (Intelligent Driver Model) [23], Kerner [24], ACC (Adaptive Cruise Control)

[25], and CACC (Cooperative Adaptive Cruise Control) [25]. In this study, we use the ACC [25] and the CACC [25] models as they are developed to represent the ADAS features, which are widely used in autonomous vehicles. The primary feature of these models is using the range sensor's data e.g., distance and relative speed, to control the vehicle in traffic.

2) Lane-Changing Model: We use LC2013 [26] model to represent the lateral behaviour of a vehicle during driving [26]. Depending on the motivation behind a lane-change manoeuvre, this model classifies the manoeuvre into four groups of strategic, cooperative, tactical, and keep right hand side. In other words, the activation of these manoeuvres are tightly connected to the defined traffic scenario (e.g., road shape). In this paper, we focus on the cooperative and tactical lane-change manoeuvres since they get activated in our traffic scenario (see §III-B). A vehicle changes its lane to cooperate with other vehicles in the surrounding traffic. In other words, the reason for the change of lane is facilitation of lane-change manoeuvres for other vehicles in the traffic. Whereas, a vehicle performs a tactical lane-change to avoid driving into or getting stuck behind another vehicle or to avoid driving in lanes with certain speed limitation.

B. Traffic Simulation Scenario

An important part of constructing a fault injection setup is the selection of the workload or stimuli that, in the case of our paper, is referred to as the traffic simulation scenario. Building representative scenarios to evaluate the safety of autonomous vehicles is of great importance. As ADAS features are most commonly used in highway roads, in this study, we use a three-lane road network scenario, where 10 vehicles are driving on the road during a simulation run (see Fig. 1). The road's speed limit is 36 m/s and its length is 750 m. We inject most of the faults into the red vehicle. If this vehicle shifts lane to the leftmost lane, the blue vehicle becomes the follower vehicle. In the remaining of this paper, we refer to the red vehicle as our *ego-vehicle* and the blue vehicle as our *follower-vehicle*. Moreover, the green vehicles are our traffic participants.

C. Fault Injection Time Interval

The total simulation run-time for the defined traffic scenario is 42 s, where the first 11 s are considered as the time it takes for all the vehicles to appear in the vicinity of the target vehicle, which allows us to measure the impact of the injections on the entire traffic. We chose the time between 11 s and 21 s as the fault injection time interval. This is because it is very likely for the ego-vehicle to perform a lane-change within this period, which allows us to evaluate the car-following and lane-changing models.

Within the fault injection time interval, the ego-vehicle is blocked by a slower vehicle, which contributes to the willingness of the ego-vehicle to overtake the slower vehicle by performing a tactical lane-change to the left hand side lane. This lane is however occupied by the follower-vehicle at this point (see Fig. 1). It is worth noting that, there is no lane-change manoeuvre in the fault-free (golden) run of the scenario; thus through the injection of faults, we could monitor the changes in the willingness of the ego-vehicle in conducting the manoeuvre.

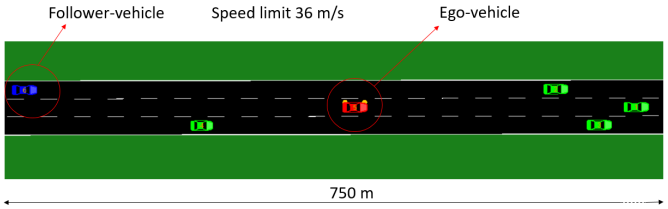

Fig. 1: Traffic simulation scenario defined.

D. Fault Injection Locations

The vehicle behaviour could be affected by injection of faults in a variety of parameters used by the car-following and lane-changing models. However, the effectiveness of these injections in manipulating with vehicle behaviour is tightly connected to the defined traffic scenario. Here, effectiveness refers to the changes in the deceleration profiles of the vehicles (see §III-F). Therefore, in this paper, we inject faults into parameters that could affect the longitudinal and lateral behaviour of the vehicle. These parameters are *LC-assertive*, *LC-overtake-right*, *error state*, and *reaction time* that are detailed in Table I. Note that, LC is an acronym for Lane-Change.

The *LC-assertive* and *LC-overtake-right* parameters are two of the variables used within the lane-changing algorithm implemented in SUMO to model lane-changing manoeuvres; while the *error state* parameter, used to model errors in gap and speed estimation, is a variable used within the car-following algorithm. The *reaction time* parameter, on the other hand, is a variable in SUMO which defines the update frequency of the calculations connected to controlling the longitudinal and lateral behaviour of vehicles.

Table II presents details about the fault injection models used. For the LC-assertive parameter, in the stuck-at-value model, the value selection is performed by sampling the range (1.0 - 1000.0) with 100 values. For this reason, the *random.randrange* function in Python is used which returns a uniformly selected integer from a given range. The value range of this parameter in SUMO is real positives where *1.0* refers to low eagerness to perform a lane-change and *1000.0* refers to high eagerness. The selection of 1000.0 as the upper bound is because in our chosen scenario and for each injection time point, larger values would result in an outcome classification (see §III-F) that is the same as when 1000.0 is selected. For the error state parameter, the usage of this parameter when estimating the gap and speed could allow us to select an appropriate value range for the fault injection experiments. The gap and the speed are estimated using the following equations:

$$\begin{aligned} &estimatedGap = realGap + realGap * 0.75 * errorState \\ &estimatedSpeed = realSpeed + realGap * 0.15 * errorState \end{aligned} \quad \text{(Eq. 1)}$$

Here, *0.75* and *0.15* are headway error coefficient and speed difference error coefficient, respectively [28]. Note that, the

TABLE I: Description of Fault Injection Locations.

Fault injection location in SUMO	Definition in SUMO [27]	Examples of Faults and Attacks Modelled	Default Value (unit)	Considered Value Range
LC-Assertive	Eagerness of the vehicle to perform a lane-change by accepting lower front/rear gaps in the target lane: Required gap amount is divided to this parameter	(i) sensor fault causing an incorrect gap estimation when changing the lane; (ii) jamming the sensor causing it not to detect another vehicle	1.0 (-)	1.0 - 1000.0
LC-Overtake-Right	Likelihood of breaking the right-hand passing rules in asymmetric roads; overtaking from right is prohibited for speeds higher than 60 km/h	(i) sensor fault causing an incorrect gap/speed estimation when changing the lane; (ii) jamming the sensor causing it not to detect another vehicle	0.0 (-)	0.0 - 1.0
Reaction Time	Update time of the vehicle state (speed, position, etc.)	Jamming the ECU or perception units when they send or receive data, causing the system to react with a delay	0.1 (s)	0.1 - 5.1
Error State	Error in gap and speed estimation for the car-following model (see Eq. 1)	(i) sensor fault causing an incorrect gap/speed estimation when following a vehicle; (ii) jamming the sensor causing it not to detect a vehicle	0.0 (-)	1.0 - 100.0

TABLE II: Description of the chosen Fault Injection Models and implementation details.

Fault injection location in SUMO	Fault Injection Models used (see §II-C)	Value selection	Experiment number
LC-Assertive	Stuck-at-value	*100* values are randomly selected in range *1.0 - 1000.0*	$100 * 20 = 2000$
	Single bit-flip	*64* values are selected by flipping one bit each time	$64 * 20 = 1280$
	Double bit-flip	*100* values are selected by flipping two bits each time (repetitiveness avoided)	$100 * 20 = 2000$
	Chain-of-faults	As a first fault, its value selection is the same with the abovementioned stuck-at-value, and single bit-flip models	$100 * 25 * 7 = 17500$ $64 * 25 * 7 = 11200$
LC-Overtake-Right	Stuck-at-value	*100* values are selected in range *0.0 - 1.0* with interval *0.01*	$100 * 20 = 2000$
	Single bit-flip	*64* values are selected by flipping one bit each time	$64 * 20 = 1280$
	Double bit-flip	*100* values are selected by flipping two bits each time (repetitiveness avoided)	$100 * 20 = 2000$
Reaction Time	Stuck-at-value	*50* values are selected in range *0.1 - 5.1* with interval *0.1*	$50 * 20 = 1000$
	Chain-of-faults	As a second fault, 25 values are selected from range *0.1 - 5.1* with interval *0.2*	-
Error State	Stuck-at-value	*100* values are selected in range *1.0 - 100.0* with interval *1*	$100 * 20 = 2000$

error state parameter could take positive and negative values. After monitoring the simulations, we confirmed that the negative values cause the vehicle to perform an emergency braking, thus effectively revealing the weaknesses of the ADAS features modelled in SUMO. Therefore, here, we limit the value range to positive values between 1.0 and 100.0, where the selection of 100.0 as the upper bound is because in our chosen scenario, larger values would result in an outcome classification (see §III-F) that is the same as when 100.0 is selected.

The experiment numbers indicated in Table II refer to the multiplication of the selected values and injection time steps (e.g., 20 time steps between 11 s and 21 s with an interval of 0.5 s). For the chain-of-faults, there are two fault locations. For the first fault, the value selection is kept the same (100 for the stuck-at-value and 64 for the single bit-flip), while for the second fault, 25 values are selected in range *0.1 - 5.1* with interval *0.2*. Note that for the chain-of-faults model, the time period between 11.0 s and 13.5 s are considered for the injections (7 time steps with an interval of 0.5 s).

E. SUMO-based Fault Injector (SUFI)

In this paper, we present SUFI[1](see Fig. 2), which is a simulation-based fault injector where faults are injected into various driver assistance functions modelled in SUMO (see §II-A). The injector is written using Python scripts. The scripts allow us to implement the fault models (see §II-C) and specify the fault injection locations (see §III-D) and fault injection time interval (see §III-C) as well as to conduct data logging. On the other hand, SUMO acts as a server to execute commands. The Python scripts and SUMO are communicating with each other via TraCI (Traffic Control Interface) [12].

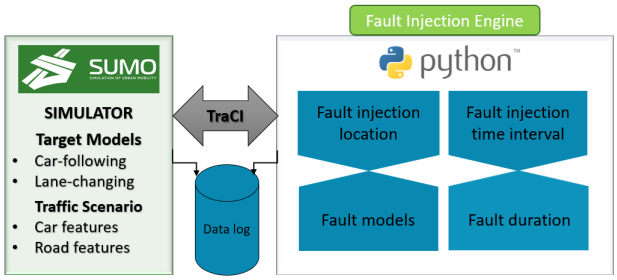

Fig. 2: Architecture of SUFI - SUmo-based Fault Injector

F. Outcome Classifications

In this paper, we recorded all microscopic data such as speed, deceleration, delay, and trajectory of each vehicle for all fault injection experiments. However, we only use the deceleration parameter (braking rate) to measure the effectiveness of the fault injection experiments and classify them into one

[1] *https://github.com/RISE-Dependable-Transport-Systems/SUFI*

of the classes presented in this section. Deceleration is chosen as previous studies for analysing the rear-end accidents show that at least 83% of the rear-end crashes are caused as a result of the lead vehicle's deceleration or stop [29].

To evaluate the impact of the injected faults, we monitored the behaviour of all vehicles for each experiment. However, we only chose the most vulnerable vehicle-behaviour to classify the result. Therefore, it should be noted that the considered vehicle-behaviour to classify the result may differ in each experiment. Note that, the braking (b) rate 0.78 m/s^2 is the maximum deceleration recorded in the golden run for the target time interval (see §III-C) and the maximum comfortable braking rate and the maximum emergency braking rate are defined as 5.0 m/s^2 [30] and 8.0 m/s^2 [31], respectively.

- **Non-effective** The injected fault has no effects on the behaviour of the vehicles and the simulation ends with no indication of failures.
- **Negligible** The injected fault has modified the behaviour of at least one of the vehicles. The change of behaviour is however negligible as the recorded maximum deceleration is less than or equal to 0.78 ($b \leq 0.78\ m/s^2$).
- **Benign** The injected fault has modified the behaviour of at least one of the vehicles, leading to a deceleration value greater than 0.78 m/s^2. The safety implications of the change is considered to be benign as it does not lead to a deceleration value greater than the maximum comfortable braking rate ($0.78\ m/s^2 < b \leq 5.0\ m/s^2$).
- **Severe** The injected fault has modified the behaviour of at least one of the vehicles, leading to a deceleration value greater than the maximum comfortable braking rate ($5.0\ m/s^2 < b \leq 8.0\ m/s^2$), some of which also result in car collisions. Note that, we classify car collisions in this group since we consider any emergency braking to be severe even if it does not cause a car collision. Moreover, according to our observations, all collisions start with an emergency braking.
- **Crash** The simulator crashes after the fault injection. Note that, these crashes do not correspond to car collisions, which are classified as severe (see above), and instead are connected to a failure caused in the simulator.

IV. Experimental Results

A. Effectiveness of the Fault Models in Revealing the System Weaknesses

Table III shows the results of the fault injection experiments conducted. The table shows a great number of severe cases for when the stuck-at-value model is used to inject faults into the *LC-assertive* parameter. The single and double bit-flip models, on the contrary, seem to be significantly less effective in causing severe results when the *LC-assertive* parameter is targeted. The results of injections in this parameter also show that the bit-flip models could result in a simulation crash. Our observations revealed that these crashes are caused when targeting the second highest significant bit of the *LC-assertive* parameter, which changes the value of the parameter to "infinity" that is unacceptable for SUMO.

Table III also shows that when targeting the *LC-overtake-right* parameter, the stuck-at-value model is significantly more effective (when compared with the other fault models) in causing benign results. Moreover, the table shows that the stuck-at-value model was effective in causing benign or severe results when targeting the *error state* and *reaction time* parameters.

> **RQ1 and RQ3-Answer:** Fault injection is successfully used to reveal weaknesses of driver assistance functions modelled in SUMO. The stuck-at-value fault model is more effective in causing severe results when compared with bit-flip models.

B. Importance of the Fault Injection Location and Duration

Table III shows that the *LC-assertive* parameter is more vulnerable than the *LC-overtake-right* when it comes to resulting in severe results. However, after monitoring the simulation run when targeting the *LC-overtake-right* parameter, we learned that the ego vehicle attempts to change its lane close to the end of the simulation run time. This means that a longer simulation run time could have also resulted in a greater number of severe cases for when *LC-overtake-right* is targeted. This shows the importance of designing representative traffic scenarios when using fault injection experiments. Table III also shows that all the faults injected into the *error state* and *reaction time* parameters are effective. The majority of the faults have in fact classified as benign or severe.

Table III also reveals the impact of the duration in which the target models are exposed to the faults (see §II-C). The number of severe cases obtained shows that the difference between the results obtained by the transient and semi-permanent models are insignificant except for when targeting the *reaction time* parameter. This is an interesting result obtained as prior to conducting the experiments, one could assume that the semi-permanent model should be much more effective in causing severe results due to the fact that the system under test is exposed to the faults over a longer period of time compared for when the transient fault model is used.

> **RQ4-Answer:** The fault injection location plays an important role in revealing the weaknesses of driver assistance functions modelled in SUMO. Moreover, the duration in which a system is exposed to a fault is insignificant in causing a severe failure in most of the target locations, except for the reaction time parameter.

C. Impact of the Injected Fault on the Surrounding Traffic Behaviour

The results of our experiments show that the faults injected, in addition to affecting the speed profile of the target vehicle, influences the speed profile of multiple other nearby vehicles. In this section, we look into both benign and severe cases presented in Table III to identify the vehicles that have been

TABLE III: Experiment results (stuck-at-value and bit-flip models) for when the cooperativeness of the ego-vehicle is enabled.

Fault Model	Duration	Non-Effective	Crash	Negligible	Benign	Severe	Total
LC-assertive parameter							
Stuck-at-value	Semi-permanent	9	0	597	98	1296	2000
	Transient	3	0	600	100	1297	2000
Single bit-flip	Semi-permanent	1220	20	12	2	26	1280
	Transient	1222	20	10	2	26	1280
Double bit-flip	Semi-permanent	1843	54	22	7	74	2000
	Transient	1842	56	22	5	75	2000
LC-overtake-right parameter							
Stuck-at-value	Semi-permanent	0	0	25	1975	0	2000
	Transient	620	0	207	1173	0	2000
Single bit-flip	Semi-permanent	1260	0	11	9	0	1280
	Transient	1260	0	3	17	0	1280
Double bit-flip	Semi-permanent	1937	0	33	30	0	2000
	Transient	1943	0	7	50	0	2000
Reaction time parameter							
Stuck-at-value	Semi-permanent	0	0	282	572	146	1000
	Transient	0	0	550	450	0	1000
Error state parameter							
Stuck-at-value	Semi-permanent	0	0	0	1998	2	2000
	Transient	0	0	300	1700	0	2000

affected the most by the injected faults. The inclusion of benign cases in our analysis is due to the low number of severe cases caused when injecting faults into the *LC-overtake-right* and *error state* parameters.

For the *LC-assertive* parameter, the follower-vehicle is the one flagging all the benign and severe cases. For the *LC-overtake-right* parameter, however, the ego-vehicle flagged these cases. For the *reaction time* parameter, 94.6% of all cases were flagged by the ego-vehicle and 5.4% were flagged by other vehicles in traffic except the follower-vehicle. Moreover, 99.8% of the cases were flagged by the ego-vehicle and 0.2% by the other vehicles when targeting the *error state* parameter.

RQ2-Answer: Faults injected into a vehicle could significantly influence the behaviour of the target vehicle or other vehicles in traffic. Therefore, the impact of the injected fault to the entire traffic should be taken into account when classifying the fault injection results.

D. Impact of the Cooperativeness Feature on the Fault Injection Results

Cooperative behaviour in traffic is the ability of the vehicle to consider the nearby vehicles behaviours when controlling itself. This behaviour can have positive impacts on traffic quality such as enhancing the road safety, reducing the traffic congestion and improving the traffic efficiency. However, what are the safety implications of using the cooperativeness feature in the presence of faults? In order to better understand the impact of cooperativeness on the vehicle behaviour that is exposed to faults, we performed another set of experiments on the ADAS features while disabling the cooperativeness of the ego-vehicle. The results are presented in Table IV, which could be compared with those obtained when the cooperativeness was enabled (see Table III).

When comparing Table III and Table IV, we observed that disabling the cooperativeness feature of the ego-vehicle for when targeting the *LC-assertive* parameter result in a higher number of severe cases for the stuck-at-value model and a lower number of severe cases for the bit-flip models. After analysing the results, we learned that disabling the cooperativeness feature results in a smaller gap to the lead vehicle on the target lane, which consequently resulted in the bit-flip model to be less effective in causing the ego-vehicle to perform a lane-change. We also observed that for the *reaction time* parameter, the number of severe cases are lower for when the cooperativeness is disabled, while the number of severe cases is higher when the *error state* parameter is targeted. This variability of the results shows that the cooperative feature of vehicles influences the fault injection results differently for when targeting different locations.

RQ6-Answer: The cooperativeness feature could significantly jeopardise the system safety. Therefore, this feature would need to be thoroughly evaluated with respect to faults and attacks when designing and implementing advanced driver assistance systems.

E. Effectiveness of the Chain-of-Faults Model

When analysing the result of fault injection experiments for the *LC-assertive* parameter, we learned that none of the faults injected during the period between 11.0 s and 14.0 s resulted in a severe case. Therefore, we decided to further investigate this interval using the chain-of-faults model. This way, in addition to injecting two simultaneous faults, we can model attacks such as the delay attack and evaluate the target system for cases in which the second fault is injected 3, 5, and 7 seconds after the injection of the first one. Note that, the 7 seconds delay is selected as the upper bound to be able to inject the second fault within the injection time

TABLE IV: Experiment results (stuck-at-value and bit-flip models) for when the cooperativeness of the ego-vehicle is disabled.

Fault Model	Duration	Non-Effective	Crash	Negligible	Benign	Severe	Total
LC-assertive parameter							
Stuck-at-value	Semi-permanent	5	0	0	300	1695	2000
	Transient	5	0	0	300	1695	2000
Single bit-flip	Semi-permanent	1240	20	0	3	17	1280
	Transient	1240	20	0	3	17	1280
Double bit-flip	Semi-permanent	1883	55	0	13	49	2000
	Transient	1882	57	0	10	51	2000
LC-overtake-right parameter							
Stuck-at-value	Semi-permanent	2000	0	0	0	0	2000
	Transient	2000	0	0	0	0	2000
Single bit-flip	Semi-permanent	1280	0	0	0	0	1280
	Transient	1280	0	0	0	0	1280
Double bit-flip	Semi-permanent	2000	0	0	0	0	2000
	Transient	2000	0	0	0	0	2000
Reaction time parameter							
Stuck-at-value	Semi-permanent	0	0	689	184	127	1000
	Transient	0	0	1000	0	0	1000
Error state parameter							
Stuck-at-value	Semi-permanent	0	0	0	20	1980	2000
	Transient	0	0	1900	100	0	2000

TABLE V: Experiment results for chain-of-faults (all faults are semi-permanent).

First Target Parameter: LC-Assertive		Second Target Parameter: Reaction Time		Classification of Fault Injection Results					
Fault Model	**Target Vehicle**	**Fault Model**	**Target Vehicle**	**Non-effective**	**Crash**	**Negligible**	**Benign**	**Severe**	**Total**
Simultaneous Injection									
Stuck-at-value	**Ego**	**Stuck-at-value**	Ego	0	0	2703	9697	5100	17500
			Follower	0	0	4000	3000	10500	17500
Single bit-flip			Ego	0	175	2135	7103	1787	11200
			Follower	0	175	4108	4642	2275	11200
3 seconds delay									
Stuck-at-value	**Ego**	**Stuck-at-value**	Ego	0	0	8000	5800	3700	17500
			Follower	0	0	6700	6000	4800	17500
Single bit-flip			Ego	0	175	2172	7253	1600	11200
			Follower	0	175	5011	5306	708	11200
5 seconds delay									
Stuck-at-value	**Ego**	**Stuck-at-value**	Ego	0	0	12900	4100	500	17500
			Follower	0	0	9100	8100	300	17500
Single bit-flip			Ego	0	175	3915	5392	1718	11200
			Follower	0	175	1698	9321	6	11200
7 seconds delay									
Stuck-at-value	**Ego**	**Stuck-at-value**	Ego	0	0	15000	2500	0	17500
			Follower	0	0	14600	2900	0	17500
Single bit-flip			Ego	0	175	5302	4564	1159	11200
			Follower	0	175	272	10753	0	11200

interval i.e., in between 11-21. Table V shows the experiment results for the chain-of-faults model where the first faults target the *LC-assertive* parameter and the second ones target the *reaction time* parameter. The results show that the injections cause several severe cases. However, Table V shows that the injection time of the second fault is important when it comes to influencing the system behaviour. Moreover, the table reveals that injecting the second fault into the follower-vehicle usually has a higher impact than when injecting into the ego-vehicle.

RQ5-Answer: The chain-of-faults is an effective model to reveal the weaknesses of the driver assistance functions modelled in SUMO. The effectiveness could vary depending on the time between the first and second injection.

V. Conclusions and Implications

In this paper, we present SUFI, a simulation-based fault injector that is capable of injecting faults and attacks into ADAS features such as car-following and lane-changing models, simulated in SUMO. Integrating a high detailed and high speed traffic simulator like SUMO in SUFI allows us to construct and evaluate complex traffic scenarios as well as to analyse the impact of the faults/attacks on the entire traffic.

Through conducting 283 840 experiments, we successfully revealed some of the weaknesses of ADAS features modelled in SUMO and showed that simulation-based fault injection is an effective way to evaluate system safety. The fault injection results show that parameters such as the fault model and fault location play an important role in revealing weaknesses of ADAS features modelled in SUMO. However, the duration

in which a system is exposed to faults is, for most target locations, insignificant in causing severe failures. Moreover, the results show that by targeting multiple components to faults/attacks, something that is very likely to be explored by an attacker, we can reveal weaknesses of the system that could not be revealed when only one component is targeted. The results also show the importance of analysing the effect of the injected faults on the entire traffic. Moreover, the results show that cooperative feature of the target vehicle has a high impact on the effectiveness of the fault injection experiments.

The results obtained also show clear dependencies between the results and the parameters used to setup a traffic simulation scenario. Examples of these parameters are the number of road lanes, speed limit and length of the road (simulation run time) as well as features of the vehicles (braking ability, acceleration, etc.) and driving scenario (sinusoidal, braking, etc.). We encourage practitioners to take scenario parameters into account when building their setup for simulation-based fault and attack injection, as they could significantly influence the effectiveness of the faults/attacks in revealing the weaknesses of the system under test. This has motivated us to put one of our future focuses on the selection of traffic scenarios covering a wide range of real-world situations.

As part of the future work, we also plan to perform injections in other locations of the CACC and ACC models as well as to model other groups of faults/attacks. Besides, as physical features of the surrounding environment such as the weather condition are not taken into account in SUMO, we plan to use simulators such as CARLA [7] that provides us with modelling of such features.

ACKNOWLEDGEMENT

This work was supported by VALU3S project, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.

REFERENCES

[1] "J3016 - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," SAE International, Standard, 2018.

[2] "ISO 26262:2018 road vehicles – functional safety," ISO (International Standard Organization), Standard, 2018.

[3] "Tesla Car Accident," https://www.tesla.com/blog/what-we-know-about-last-weeks-accident, accessed: 2021-07-05.

[4] "Waymo Self-driving Car Crash," https://www.theverge.com/2018/5/4/17320936/waymo-self-driving-car-crash-arizona, accessed: 2021-07-05.

[5] "Uber Self-driving Car Test," https://en.wikipedia.org/wiki/Death_of_Elaine_Herzberg, accessed: 2021-07-05.

[6] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using SUMO," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575–2582.

[7] "CARLA Open-source simulator for autonomous driving research," https://carla.org/, accessed: 2021-07-05.

[8] S. Jha, S. S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "AVFI: Fault injection for autonomous vehicles," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018, pp. 55–56.

[9] R. van der Heijden, T. Lukaseder, and F. Kargl, "Analyzing attacks on cooperative adaptive cruise control (CACC)," in *2017 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2017, pp. 45–52.

[10] P. K. Singh, G. S. Tabjul, M. Imran, S. K. Nandi, and S. Nandi, "Impact of security attacks on cooperative driving use case: CACC platooning," in *TENCON 2018-2018 IEEE Region 10 Conference*. IEEE, 2018, pp. 0138–0143.

[11] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, no. 1, pp. 3–15, January 2011.

[12] "Traffic Control Interface," https://sumo.dlr.de/docs/TraCI.html, accessed: 2021-07-05.

[13] B. Sangchoolie and P. Folkesson and J. Vinter, "A study of the interplay between safety and security using model-implemented fault injection," in *2018 14th European Dependable Computing Conference (EDCC)*, 2018, pp. 41–48.

[14] Aliabadi, Maryam Raiyat and Pattabiraman, Karthik, "FIDL: A Fault Injection Description Language for Compiler-Based SFI Tools," in *Int. Conf. on Computer Safety, Reliability, and Security*. Springer, 2016, pp. 12–23.

[15] "Row hammer privilege escalation vulnerability," https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20150309-rowhammer/, accessed: 2021-07-05.

[16] Y. Kim and R. Daly and J. Kim and C. Fallin and J. H. Lee and D. Lee and C. Wilkerson and K. Lai and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *2014 ACM/IEEE 41st Int. Symp. on Computer Architecture (ISCA)*, 2014, pp. 361–372.

[17] J. V. Carreira, D. Costa, and J. G. Silva, "Fault injection spot-checks computer system dependability," *IEEE Spectrum*, vol. 36, no. 8, pp. 50–55, 1999.

[18] J. Carreira, H. Madeira, and J. Silva, "Xception: a technique for the experimental evaluation of dependability in modern computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 125–136, 1998.

[19] F. Ayatolahi, B. Sangchoolie, R. Johansson, and J. Karlsson, "A study of the impact of single bit-flip and double bit-flip errors on program execution," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2013, pp. 265–276.

[20] B. Sangchoolie, K. Pattabiraman, and J. Karlsson, "One bit is (not) enough: An empirical study of the impact of single and multiple bit-flip errors," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 97–108.

[21] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman, "LLFI: An intermediate code-level fault injection tool for hardware faults," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 11–16.

[22] S. Krauß, "Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics," Ph.D. dissertation, 1998.

[23] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[24] B. S. Kerner, "Three-phase traffic theory and highway capacity," *Physica A: Statistical Mechanics and its Applications*, vol. 333, pp. 379–440, 2004.

[25] V. Milanés and S. E. Shladover, "Modeling cooperative and autonomous adaptive cruise control dynamic responses using experimental data," *Transportation Research Part C: Emerging Technologies*, vol. 48, pp. 285–300, 2014.

[26] J. Erdmann, "Lane-changing model in SUMO," *Proceedings of the SUMO2014 modeling mobility with open data*, vol. 24, pp. 77–88, 2014.

[27] "SUMO Parameter's Definition," https://sumo.dlr.de/docs/Definition_of_Vehicles,_Vehicle_Types,_and_Routes.html, accessed: 2021-07-05.

[28] "Perception Errors in SUMO," https://sumo.dlr.de/docs/Driver_State.html, accessed: 2021-07-05.

[29] S. E. Lee, E. Llaneras, S. Klauer, and J. Sudweeks, "Analyses of rear-end crashes and near-crashes in the 100-car naturalistic driving study to support rear-signaling countermeasure development," *DOT HS*, vol. 810, p. 846, 2007.

[30] "SUMO Vehicle Type Parameter Defaults," https://sumo.dlr.de/docs/Vehicle_Type_Parameter_Defaults.html, accessed: 2021-07-05.

[31] P. Greibe, "Braking distance, friction and behaviour," *Trafitec, Scion-DTU*, 2007.