Towards a Simulation-based Framework for the Security Testing of Autonomous Vehicles

Eduardo dos Santos
Cyber Security Centre for Doctoral Training
Computer Science Department
University of Oxford
Oxford, OX1-3PR, UK
eduardo.dossantos@cs.ox.ac.uk

Dominik Schoop
Esslingen University of Applied Sciences
73732 Esslingen am Neckar, Germany
dominik.schoop@hs-esslingen.de

ABSTRACT

Autonomous cars depend on data collection and processing in order to operate safely. Attacks against sensors can lead to an incorrect perception of the environment. Testing the impact of sensor attacks on the road is costly and dangerous. Simulation tools for autonomous driving research enable the simulation of the urban environment, thus providing a cheap and risk-free platform for the testing of perception systems. In this paper, we extend an open-source autonomous driving simulator with models of attacks to camera sensors. These attacks are modelled as computer graphics effects, which are subsequently added to the simulated environment. We also present a formal model, developed in the process algebra CSP, by which systematic testing of all possible sensor-attack combinations may be yielded. Although our results are at a preliminary stage, we are mindful of its contributions to the security — and therefore, safety — of autonomous driving.

Keywords

Autonomous vehicles, data fusion, security testing, formal models.

1. INTRODUCTION

The mass deployment of autonomous vehicles (AVs) is expected to bring about improvements in road safety and urban mobility. Commercial vehicles that are equipped with autonomous functions already exist, with examples including Tesla Autopilot [8]. However, these functions have their operation constrained to certain road and weather conditions [15].

Despite being a promising new technology, the security of autonomous vehicles has attracted attention from the cybersecurity community. As an example, Petit *et al.* identify 12 sub-systems that are likely to be target to attacks, including: infrastructure signs, machine vision, GPS, Lidar and Radar [22]. In this paper, our focus is on security concerns that do not overlap with those of non-autonomous cars, especially in regard to the security of internal bus systems [16] and vehicular network [36]. The autonomous driving paradigm is based on the fact drivers are removed from the decision-making loop — or, in the case of partial autonomy, they are less present [30].

Because of their over-reliance on sensors and computing, there is the fear that autonomous vehicles might be easily fooled [22]. This might lead to various consequences, one of which is the vehicle making the wrong decision based on incorrect data. This can have potential safety consequences for its occupants, the occupants of other vehicles and nearby pedestrians.

Learning algorithms are at the core of autonomy. Such algorithms are heavily dependent on data, from which they can learn and deliver more accurate results (in this case, vehicle control decisions). Machine learning programs have been known as non-testable [18, 39]. We are not worried about the problem of whether data used to train such algorithms is correct. Rather, we are worried whether the trained algorithms will have the ability to distinguish between normal data and abnormal data — which may arise from illicit manipulation of sensor readings.

Given that abnormal data (or, in our context, *insecure* data) is not easily found in the environment, and, as a result, is not usually incorporated into training datasets, there is the risk a vehicle's perception systems will not be able to detect such discrepancies. An example is the situation illustrated in Figure 1. A stereo (3D) binocular camera uses two or more image sensors to create a 3D view of the environment (top image). If one of its sensors captures too much noise, as in the form of a red laser beam pointed at it [40], the resulting 3D view may equally be noisy or, even, incomprehensible (bottom image). This can have a severe impact on systems that depend on good quality image data to make decisions.

Testing is a critical part of an automobile's development life cycle. Car manufacturers have used road testing as a means to assess the safety of autonomous features. Although road testing gives more confidence in practical real-life scenarios, it cannot safeguard against rarely-seen traffic scenarios. Certain scenarios are simply too dangerous or complex to test

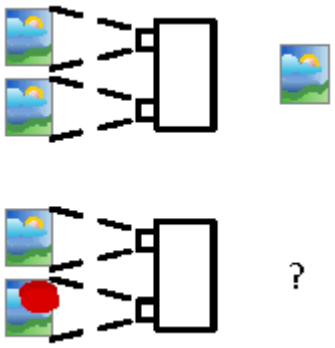

Figure 1: Overview of insecure data with a 3D (stereo) camera example.

(e.g. a baby crawling in the road, or birds flying at vehicle height).

These limitations can be overcome by simulation. Recent years have seen the development of tools that simulate the operation and environment autonomous vehicles operate (e.g. [9, 31]). Simulating the environment allows data collection and algorithm testing without the costs of operating a real car [9]. Collected simulated data can later be used to train in-car algorithms [5]. Furthermore, simulating the vehicle's internal functioning opens the door for research groups to design, implement and test new automotive bus systems without underdoing the complexity of working with real hardware.

This research is driven by the desire to *develop an automated method to test the perception of an autonomous vehicle against attacks to its perception sensors.* To accomplish this, we use a formal model that leverages Communication Sequential Processes (CSP) to enable a directed and achievable test generation effort. As source of data for tests, we extend and use an openly-available urban driving simulator. Our primary focus is the provision of models for testing existing driving algorithms (as opposed to creating new, more secure, algorithms). However, we do provide directions on how this can be achieved. The contributions of this work fall in the categories of software testing and secure software engineering

A note on terminology. We use the words *'vehicle'* and *'car'* interchangeably. For simplicity, other types of road transport (e.g. trucks, motorcycles) are referred to as cars. Whenever we write autonomous vehicle (AV), we explicitly refer to autonomous cars.

2. BACKGROUND

This section introduces the main supporting literature with a view to motivating and framing our contribution. Section 2.1 contextualises autonomous vehicles. A realistic related threat model is presented in Section 2.2. Next, Section 2.3 reviews the literature on attacks on perception-enabling sensors, emphasizing attacks to LiDAR scanners, global navigation satellite systems, and cameras. Finally, the research problem subject of this paper is summarised in Section 2.4.

2.1 Autonomous Vehicles

Modern road vehicles incorporate a number of advanced driver assistant systems (ADAS) that make use of a variety of sensors to offer functions making driving safer and more convenient. Examples of such ADAS include adaptive cruise control, automatic emergency braking, lane departure warnings, and parking pilots. The latter function is also available when the driver is outside the vehicle and has a wireless control channel to the vehicle. Currently, these functions require the driver to be observant and to be able to take over control of the vehicle in an instant. The functions are only partly autonomous.

However, there have already been commercialisation of vehicles equipped with higher autonomous functions that require driver attention. An example is Tesla Model S, which has a self-driving function. Audi is currently introducing a "traffic jam pilot", which drives the vehicle in traffic jams autonomously, into the Audi A8. In addition, national laws are being changed to allow for such functions to be used on public roads (e.g. [7, 33]).

Such higher autonomous driving functions are not fully autonomous, i.e. they do not operate safely in all situations. Therefore, the function can only be activated in certain situations and the driver must be able to take over control again within a few seconds. Since the Tesla Autopilot has become available to the public, there have been deaths associated with the use of it [10]. Nevertheless, the goal of some companies, e.g. Waymo (formerly Google Self-driving Car), is the development of fully autonomous vehicles that do not have a driver.

The Society of Automotive Engineers (SAE) defines five levels of autonomy in vehicles. To be considered fully autonomous (Level 5), cars must operate autonomously in any kind of environment and under any weather condition [6]. Currently, cars are in the middle of the level spectrum (Level 3, conditional automation), with only some autonomy available.

Understanding the environment is a complex task. Vehicles equipped with autonomous functions need a diverse array of sensors available. Each class of sensor measures different physical properties and contribute to form a different perception of the environment. Sensor allocation in cars is driven by cost, having no definite standards across manufacturers.

An AV's perception system is commonly composed by any of the following classes of sensors:

1. **Lidar:** up to 300 metres range, can provide 2D and 3D surface models, used for object recognition, e.g. pedestrian detection;
2. **Long-range radar:** 40 to 300 metres range, used for distance measurements of objects, e.g. adaptive cruise control;
3. **Short-/Medium-range radar:** used for e.g. blind spot detection;
4. **Cameras:** 2D or 3D, used for e.g. lane departure warning and for relative positioning in known environments
5. **Ultrasonic sensor:** used for detection of close obstacles, for e.g. parking pilot;

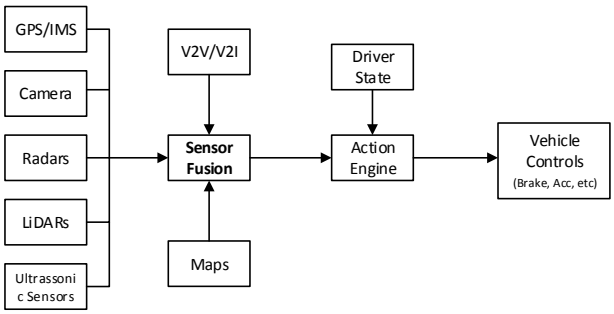

Figure 2: Outline of an autonomous vehicle's perception system.

6. **Satellite positioning system:** e.g. GPS, provides global location information with accuracy up to several meters; differential GPS (dGPS) has precision in few centimetres but is still too expensive to be used in general; positioning data has to be augmented with camera, IMS, map data and/or dead reckoning from wheel rotation;
7. **Inertial Movement Sensor (IMS):** measures changes in directional velocity

Furthermore, an AV can leverage other data sources such as (dynamic) road maps and Vehicle-to-Vehicle/Infrastructure communication (V2X), which can provide the vehicle with information about the location and speed of other vehicles as well as about information about the environment and the traffic situation.

Each sensor class has its strengths and weaknesses, expressed in their range, the materials and shapes they can observe, how they cope with signal reflections and so on. Initially, the AV has to identify objects (perception). The data of the sensors might augment or contradict each other in this step. The data of all sensors has to be consolidated to detect and to identify individual objects and their properties such as stationary, speed, soft/hard, etc. (sensor fusion). In a second step, the identified objects have to be interpreted to understand the current scenario. This interpretation can make use of historic data and will lead to an action planning, which has to be carried out eventually by the vehicle controls (Figure 2). Note that this processing has to be done in real time as far as possible. This real-time constraint in addition to the computational restrictions of an automotive system makes the correct interpretation of sensor data even more challenging.

From the point of view of regulations, AVs are required to obey the same safety standards and traffic laws as 'normal' vehicles. Consequently, the ISO 26262 standard [13], which defines guidelines for vehicle development must be followed. Since in this context safety relies also on security, the AV must operate safely even in case of sensor input being maliciously influenced.

Cyber-security research on autonomous vehicle's security has concentrated on attack detection and trust issues. For example, van der Heijden *et al.* investigate misbehaviour detection in networks of vehicles [37]. Further, while automated testing methods already exist for vehicular networks [3], these do not have security in the scope.

Table 1: Relevant assets

human health	injuries to or death of AV passengers and other road users
hardware	availability and integrity of vehicles, sensors
functionality	availability and integrity of (safety) functions of the AV
electronic data	availability and integrity of data stored in the AV

2.2 The Threat Model

We now present a brief overview of our threat model. Since, as mentioned above, the security of an autonomous vehicle can impact upon its safety, there is a need for a threat assessment and risk analysis (TARA) as well as a hazard risk assessment (HARA) for a given specific system [6]. In this paper, we consider the abstract system of an autonomous vehicle as presented in Section 2.1 and constrain ourselves to the identification of security and safety assets, as well as the most prominent threats.

The minimal asset categories to consider are *human health*, *hardware*, *functionality* and *electronic data*. An attack that interferes with the data input of sensors of the AV can cause the AV to behave inappropriately in the given physical environment and therefore can cause an accident, endangering human health. An attack might lead to the permanent malfunction (non-availability of functionality) or destruction of a sensor, possibly de-calibrating the sensor or destroying its physical components. Malicious input might lead to integrity loss of data stored in the car, e.g. a falsified dynamic map (Table 1). Other assets such as the privacy of personal data do not seem to be directly relevant here.

The attackers (threat agents) we consider are possibly active road users but do not have access to the internal systems of the AV they want to attack. The attack is purely carried out by providing data input to the external sensors of the AV. Following an attacker categorization inspired by [24], we assume that an attacker is not in control of global data input, i.e. the attacker does not control the navigation satellites. Therefore, any individual attack is limited to an individual sensor or a limited geographic area where a small number of vehicles is affected. However, we can assume that attackers are colluding and therefore can attack several sensors at the same time as well as carry out a sequence of attacks in time and space. These attacks are active and dynamic in the sense that the attacker can adapt its attack in real time dependent on the behaviour of the AV. The attacker might be malicious, i.e. not caring about resource use or consequences, or rational. As a rational attacker they would target their goal with the minimal effort possible.

The sensors of an AV will have to deal with erroneous data of many kind. A successful attack will, therefore, need to use specialized attack tools with the specific knowledge how to use them. Although design information of automotive systems are usually proprietary, attackers can carry out reverse engineering with some effort and therefore obtain detailed knowledge about the systems to attack.

An attacker might attack an AV just for fun, to stop the ve-

hicle and rob the passengers, to car-jack, with the intention to endanger specific road users, or for terrorist attacks.

A HARA according to ISO 26262 [13] considers probability, severity and controllability for the determination of the risks. Since safety is also determined by security, the probability must consider the outcome of a TARA, e.g. using one of the standard methods TVRA or EVITA [6]. Because of the threat model discussed above, the probability will be quite low. However, the severity is very high and the controllability is not existent in an AV of level 5.

2.3 Sensor Attacks

Generally-speaking, any computer system, that makes decisions based on data outside to it, must validate the origin of the data before committing to a decision. However, in cyber-physical systems, like the automotive, dependant data is created by the environment, not by other systems.

1. **Lidar.** Relay attacks to Lidar scanners can cause objects to be detected at the wrong position: far away objects are detected as being closer to the scanner (car) and vice-versa [23]. On the other hand, spoofing attacks can introduce non-existing objects to the scanner and tracking[23]. Besides spoofing and relay attacks, Lidar scanners are also subject to jamming, relay, and denial-of-service (saturating) attacks [32]. A common issue with Lidar attacks is the need for synchronicity. To be successful, an attacker needs to inject fake light pulses at a very short time interval[23, 32] . However, as this technology becomes cheaper and more accessible, so does its use by attackers.
2. **Camera.** Camera blinding can confuse the auto controls [23]. Moreover, evidence suggests that pointing a laser beam to a camera may permanently damage its CMOS/CCD sensors [40]. A common drawback of camera attacks is aim[23]. Because of the distance and the size of the camera lens aperture, it is difficult to aim the laser beam precisely from the distance [23]. However, a practical assessment with a Tesla Model S unit revealed permanent damage to the sensor can be caused while the car is parked. Custom-built Arduino devices mounted on the front or rear car may be used as a tool to track and automatically direct a laser beam towards a camera while on the move[23] .
3. **GPS.** Spoofing attacks to GPS systems have been known since 2002 [38].

On another study, Yan *et al.* [40] measure the impact of sensor attacks against a Tesla Model S unit equipped with Autopilot and self-parking features. Their attack surface was broad, considering a wide set of sensors, comprising millimetric wave, radars, ultrasonic sensors, and cameras. All attacks were able to confuse the vehicle while its autonomous features were being executed.

Outside the automotive domain, there have been advances on sensor security. Bezemskij *et al.* [2] develops a method to distinguish between normal and changed inputs to sensors. Moreover, Pustogarov *et al.* [26] use program analysis to synthesize sensor spoofing attacks, with a view to security testing.

Concerns over jamming and spoofing attacks to vehicle sensors have already been the subject of government guidance [29], however, there still is no clear answer to the problem. Whatever anti-jamming and anti-spoofing techniques be devised, these will inevitably need to be tested on real (or closely real) situations. This is where our work stands. Having another instance of the arms race, between security engineers and attackers, that the software engineering community has had for more than 20 years is not desirable.

Countermeasures

Countermeasures at both software- and hardware-levels have been proposed to mitigate against sensor attacks. For GPS, Warner and Johnston [38] suggest the monitoring of GPS signal strength as well as timing analysis as a way to mitigate against GPS spoofing attacks. Mitigation strategies against Lidar attacks include random probing or shortening pulse periods [23]. Blinding attacks against cameras, in turn, can be mitigated via near-infrared light filters and the use of photochromic lenses [23]. One study observed permanent damage to the camera after a blinding attack even while the vehicle was turned off [40].

Although hardware redundancy has been proposed as a solution for the sensor attack problem[23], it comes with its own set of constraints, due to limited physical space and restrictions to cost. Noteworthy to say that countermeasures based solely on hardware do not mitigate against all risks.

Implementation of software countermeasures may be easier in comparison to hardware countermeasures. However, the implementation of sensor processing software can vary across different hardware suppliers. Sensors manufactured by different suppliers need tp interoperate with each other.

Plus, even though countermeasures have been proposed, they may not be definitive. Depending on environmental conditions, certain countermeasures may be more adequate than others. Furthermore, one should not underestimate the odds that the countermeasures themselves introduce new vulnerabilities.

2.4 Summary of the Problem

Attacks to sensors can lead to misperception of the environment by autonomous cars. In the current design of autonomous vehicles, the driver can still override autonomous features if required to do so. However, the expected popularisation as well as advancements in autonomous driving, can render items like steering wheel and brake pedals obsolete. Therefore, an higher safety and security standards must be met.

From the main research question, introduced in Section 1, we derive sub-research questions. Our work intends to be a first starting point in addressing each of them.

1. *Can autonomous vehicles operate safely when one of its perception (e.g. Lidars, cameras, radars, etc) sensors malfunction? How long should attacks run for before*

placing the vehicle on an unsafe state?

2. *If perception sensors experience attacks while the car is on the move, can an AV continue its normal operation? For how long? What functions and corresponding sensors are affected?*

Furthermore, there is a need to ensure regression while new automotive software is developed and improved. Even if currently-deployed countermeasures are proven effective at a particular point in time, this must remain so throughout the development life cycle. This must be valid for new automotive functions.

We propose, in this work, that these research questions be tackled via software-based simulation. This is so because of the inherent costs and safety risks involved with the operation of an autonomous car in normal roads.

3. COMMUNICATION SEQUENTIAL PROCESSES (CSP)

Our formal models are developed in Communication Sequential Processes (CSP). CSP is a kind of process algebra formalism, which provides a formal framework for modelling and analysing concurrent systems. Several industry-grade tools provide support for CSP: the refinement checker, *Failures Divergences Refinement* (FDR) [20], and process animator ProB [25].

The choice of CSP can be justified by the systems of systems (SoS) approach to building modern vehicles, which can be modelled in CSP via composition of processes. In this context, each individual sub-system is modelled as an individual CSP process. Processes, in turn, can exchange messages between each other in a similar way to automotive sub-systems.

Further motivation for the selection of CSP lies on the fact it has widely been known by the automotive and safety industries. Ran *et. al.* [27] models the TTCAN protocol in CSP and verify its properties with complementary formalisms. Peng *et. al.* [21] uses CSP to verify safety properties of an engine management system. More recently, CSP has been used to uncover vulnerabilities on a car's firmware updating protocols [17].

In this following, we provide a brief introduction to the language of CSP.

3.1 Syntax

The *alphabet* of a CSP process is the set of events that it is willing to communicate. An *event* requires the agreement of both the environment and the communicating process, occurs instantaneously, and is atomic. The set of all possible events within the context of a particular specification is denoted Σ.

The simplest possible process is the one that offers to participate in no events: $STOP$. Another simple process, $SKIP$, represents successful termination (via the internal event, $\checkmark$). The *prefixing* operator, $\rightarrow$, allows us to prefix an event $e \in \Sigma$ to any process: the process $e \rightarrow STOP$ will refuse to communicate anything other than e, after which it will offer no further events. Recursion allows us to capture infinite behaviour using a finite description. For example, $B = b \rightarrow B$ is the process that will communicate b indefinitely.

External (or *deterministic*) choice is denoted by $\Box$: we write $P \Box Q$ to describe the process that offers the choice between the initial events of both processes, before behaving as one of P or Q. *Internal* (or *nondeterministic*) choice is denoted by $\sqcap$: the process $P \sqcap Q$ may behave as P or as Q, but the environment has no choice over which. Both forms of choice have an associated indexed form: $\Box\, i : I \bullet A(i)$ and $\sqcap\, i : I \bullet A(i)$.

A parallel composition of processes tells us not only which processes are to be combined, but also describes the events that are to be synchronised on. The *synchronous parallel* operator, $\parallel$, requires component processes to agree on all events. As an example, the process $A \parallel B$, where $A = a \rightarrow b \rightarrow STOP$ and $B = b \rightarrow B$, will deadlock as the combination cannot agree on the first event. *Generalised parallel composition*, of the form $A \underset{X}{\parallel} B$, requires cooperation on the events of X; all other events can proceed independently. For example, in $A \underset{\{b\}}{\parallel} B$, the event a occurs without B's cooperation. Some processes do not synchronise on any events; *interleaved* processes are written $A \,|||\, B$.

Channels are the means by which *values* are communicated: $c.x$ is the communication of the value x on the channel c. A natural extension to this is to consider input and output values — of the form $c?x$ and $c!x$ respectively. Input along a channel can also be expressed using external choice: we might write $\Box\, x : X \bullet c.x \rightarrow P\,(x)$.

3.2 Semantics

It is often useful to consider a trace of the events which a process can communicate: the set of all such possible finite paths of events which a process P can take is written *traces* $[\![P]\!]$. However, it is well understood that traces on their own are not enough to fully describe the behaviour of a process. For a broader description of process behaviour, we might consider what a process can refuse to do: the refusals set of a process — the set of events which it can initially choose not to communicate — is given by $refusals[P]$. By comparing the refusals set with the traces of a process, we can see which events the process *may* perform: $refusals[A \Box B] = \{\}$ and $refusals[A \sqcap B] = \{\{\}, \{a\}, \{b\}\}$ (assuming A and B as defined above, and $\Sigma = \{a, b\}$). It follows that the *failures* of a process P — written *failures* $[\![P]\!]$ — are the pairs of the form (t, X) such that, for all $t \in$ *traces* $[\![P]\!]$, $X = refusals[P/t]$, where P/t represents the process P after the trace t.

The aforementioned FDR tool compares processes in terms of refinement. We write $P \sqsubseteq_M Q$ when Q refines P under the model M: Q is 'at least as good as' P. If we were only to consider traces, then

$$P \sqsubseteq_T Q \Leftrightarrow \textit{traces}\,[\![Q]\!] \subseteq \textit{traces}\,[\![P]\!]$$

Similarly, we can define failures refinement (our primary

concern in the following) as

$$P \sqsubseteq_F Q \Leftrightarrow traces\,[\![Q]\!] \subseteq traces\,[\![P]\!] \wedge failures\,[\![Q]\!] \subseteq failures\,[\![P]\!]$$

4. FORMAL MODELS

4.1 Modeling Sensors

The first step is to model sensors, and corresponding functions, as processes in CSP. We follow the standard CSP modelling convention: processes represent objects or physical entities, and events represent actions undertaken by that object (process)[28].

Sensors work continuously (i.e. they are never supposed to stop), therefore they are modelled as recursive processes. Events ending with ${}_df$ indicate communication with the process *DATAFUSION* defined below. We use the external choice operator ($\Box$) to model two execution flows within the same main process. Both flows are also valid processes. Choice between which process (flow) to execute depends on external parameters (to be covered later).

For simplicity, we model processes for Lidar, camera and GPS only. These sensors are represented as CSP processes as follows:

$$\begin{aligned} &LIDAR = pulse \rightarrow LIDAR \\ &\quad \Box\ receive\_pulses \rightarrow classify\_objects \rightarrow LIDAR \\ &\quad \Box\ lidar\_df \rightarrow LIDAR \\ &CAMERA = view \rightarrow CAMERA \\ &\quad \Box\ camera\_df \rightarrow CAMERA \\ &GPS = get\_position \rightarrow GPS \\ &\quad \Box\ gps\_df \rightarrow GPS \end{aligned}$$

Beside sensor processes, another important process is the data fusion process (*DATAFUSION*). It unites data collected from sensors at a centralised point (as in Figure 2). Note that in our representation, it only contains $*_df$ channels, which, according to our model, these channels represent the communication between sensor processes and the data fusion process.

$$\begin{aligned} &DATAFUSION = lidar\_df \rightarrow DATAFUSION \\ &\quad \Box\ camera\_df \rightarrow DATAFUSION \\ &\quad \Box\ gps\_df \rightarrow DATAFUSION \end{aligned}$$

Finally, we model the whole system being executed, as the *SYSTEM* process. It uses a replicated alphabetised parallel operator to indicate synchronisation between all involved processes. It uses α-sets to denote what events the process synchronizes with. Those sets define which side of external choice to follow. By adding or removing elements to the α-sets, one can reduce the number of traces generated when processes are run in FDR.

$$\begin{aligned} &SYSTEM = ||\ (A, P) : \{ \\ &\quad (Alpha\_DataFusion, DATAFUSION), \\ &\quad (Alpha\_Lidar, LIDAR), \\ &\quad (Alpha\_Camera, CAMERA), \\ &\quad (Alpha\_Gps, GPS) \\ &\} \bullet [A]P \end{aligned}$$

4.2 Modeling Distributed Sensors

We also formally model fusion at a distributed level. This is the case where more than one fusion level exists. For example: a distributed fusion (at sensor level), with data from the same type of sensor, and a centralised fusion, with more diverse data sources.

This extended model may be useful to find potential flaws to distinct, but integrated, sensors, for instance, 3D cameras. Three-dimensional cameras have two lenses, each of them capture the environment from a slightly different angle, whose image is then combined. If only one of the lenses is subject of a blind attack, the resulting image is partially compromised. It might be of interest to know what degree of compromise renders a 3D image unusable – or whether any information on a compromised image can be used.

The only difference between this and the previous model is the addition of a ${}_sf$ event in place of ${}_df$ event and the redirection to a second fusion process. An example formal model of a Lidar sensor fusion is shown below. Definitions are analogous for other types of sensors.

$$\begin{aligned} &LIDAR\_1 = pulse1 \rightarrow LIDAR\_1 \\ &\quad \Box\ receive\_pulses1 \rightarrow classify\_objects1 \rightarrow LIDAR\_1 \\ &\quad \Box\ lidar\_sf1 \rightarrow LIDAR\_1 \\ &LIDAR\_2 = ... \end{aligned}$$

The *lidar_sf* events connect at the *LIDAR_SF*, sensor fusion process. The *LIDAR_SUT* process synchronises all *LIDAR* processes. The final *DATAFUSION* synchronizes with the *lidar_df* event in the *LIDAR_SF* process. A new fusion level adds two new processes to the whole formal model. α-set definitions remain the same.

$$\begin{aligned} &LIDAR\_SF = lidar\_sf1 \rightarrow LIDAR\_SF \\ &\quad \Box\ lidar\_sf2 \rightarrow LIDAR\_SF \\ &\quad \Box\ lidar\_df \rightarrow LIDAR\_SF \\ &LIDAR\_SUT = ||\ (A, P) : \{ \\ &\quad (Alpha\_LidarSF, LIDAR\_SF), \\ &\quad (Alpha\_Lidar1, LIDAR\_1), \\ &\quad (Alpha\_Lidar2, LIDAR\_2) \\ &\} \bullet [A]P \\ &DATA\_FUSION = lidar\_df \rightarrow DATAFUSION \\ &\quad \Box\ camera\_df \rightarrow DATAFUSION \end{aligned}$$

Figure 3 outlines this expanded formal model.

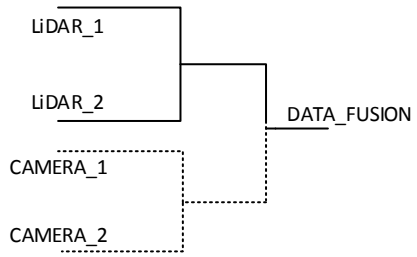

Figure 3: Multi-level sensor fusion with centralised data fusion

4.3 Test Parameters

As pointed out by Petit *et al.* [23], adjusting parameters of sensors may help to mitigate against certain attacks.For example, increasing the width and delay of light pulses can mitigate against attacks to Lidar sensors. Changing sensor's internal parameters can assist attack mitigation, however can equally result in an altered perception of the environment. Therefore, parameters need to change until finding

the most optimum setting. Parameter iteration changes each test case slightly.

Parameters can be represented as CSP datatypes. Using datatypes is an easy way to iterate over a pre-defined set of values. The unit of measurement (e.g. hz or nm) needs not be taken into account. See definition below:

$$datatypePulseParameters = \\ PulseDelay.\{10, 20, 30\} \mid \\ PulseWidth.\{10..20\}$$

Datatypes can then be added to events in processes. The datatype *PulseParameter*, defined above, is added to the event *pulse* in the *LIDAR* process:

$$LIDAR = pulse?x : PulseParameters \rightarrow LIDAR$$

Using parameters can be advantageous to interactively model attacks to sensors. In the example below, the datatype *Image* indicates whether or not the image provided to the camera will come with any modifications that make it resemble like if it were deliberately distorted in an attempt to confuse auto controls. The new datatypes are then combined with corresponding image-altering effects during test execution.

$$datatypeImage = Unaltered \mid WhiteLED \mid \\ RedLED \\ CAMERA = view?x : Image \rightarrow CAMERA$$

Using intense white and red lights, as well as, permanent red "burnt' scratches on the image are some examples of confirmed camera attacks [23, 40] . These attacks can be replicated in simulators. Later, in Section 6, we create a white LED effect in the CARLA simulator. Our view is that, even though these attacks may not be totally avoidable due to easy of physical access, machine vision algorithms should be trained to work under these circumstances. This becomes even more important after the fact the impact of these attacks can be permanent [40].

5. THE FRAMEWORK AND TOOL

5.1 The Framework

Simulation has an important role in the software development life cycle of automotive systems. We have created a framework to allow the use of the introduced formal models, as well as testing methodology, in other systems beside CARLA (Section 6).

Having this heterogeneity in mind, we create a framework to support an easy extension of our test case generation method to other tools. This framework works in the form of an API, which processes existing design project files and generates new project files as test cases. New test cases should be created by alterations of the original project file. This measure prevents loss of fidelity to the original model. We understand that different engineering tools offer options than others.

Currently, we have implemented support for CARLA only. CARLA is an urban driving simulator (more details to follow in Section 6). Another candidate for first-time support was PreScan, a commercial ADAS development software [34]. Similarly to CARLA, PreScan allows the 3D modelling of the environment, and offers support to a wider range of perception sensors, not supported by CARLA at present, for example, Lidar scanners, and Radars. PreScan is popular for ADAS development.

Support to new tools can be added by extending the abstract class *AbstractSimulator* below.

```
/** Main class to be
    implemented to add support
    to new simulators. **/
public abstract class
    AbstractSimulator {
  public abstract void parse(
      File systemModel);
  public abstract List<Attack>
      getSupportedAttacks();
  public abstract saveTestCases
      (File
      destinationDirectory);
  public abstract
      generateTestCases(
      TestGenerationParameters
      parameters)
  //...
}
```

5.2 Tool

Our automated method of generating test cases is supported by a tool. It is implemented in Java and uses the FDR API [35] to interact with and process CSP code. Figure 4 shows its user interface. The project is under development and will be made available to the public once completed.

The tool's functional workflow consists in the following tasks (Figure 5):

1. *System Model Selection.* User selects the system model from which tests will be generated.
2. *System Model Parsing.* The tool parses the file, retrieving details for every relevant element in the simulation (e.g. cars and sensors).

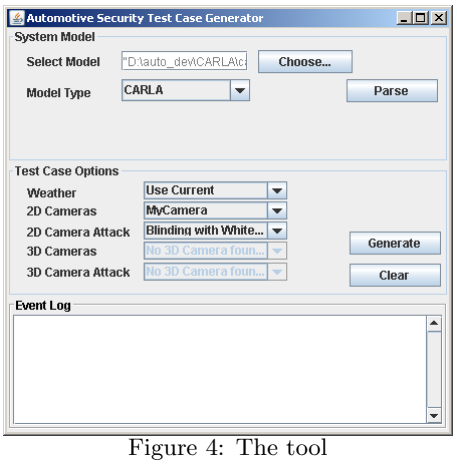

Figure 4: The tool

3. *Dynamic CSP_M Code Generation.* From the choices of attacks, provided by the user, the system generates the corresponding CSP_M code.

4. *Trace Generation and Enumeration with FDR.* The dynamically-generated CSP_M code is run in FDR. The result is parsed.

5. *Test Case Generation.* With the parsed traces obtained in the previous step, new test cases are created, each of which may or may not correspond to an individual trace. Test cases are slightly altered versions of the original system model.

6. *Test Case Execution.* Execution of test cases in their associated tool. This task is not covered by the research.

7. *Result Collection and Assessment.* Executing test cases generate data, which needs to be analysed. This analysis will result in corrective actions on the perception systems involved and provide evidence and replicability of (potentially serious) issues. This task is not covered by the research.

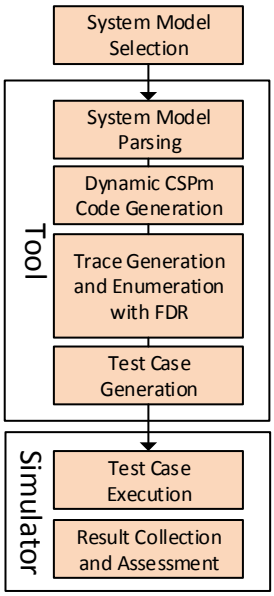

Figure 5: The tool's functional workflow

6. CASE STUDY: CARLA

CARLA (CAR Learning to Act) is an open-source simulator project for autonomous driving research [9]. The project started in late 2017, and has had an active repository since then [11]. CARLA can be used as tool for data collection and training of learning-based algorithms.

CARLA inherits the physics dynamics and environment models from Unreal Engine version 4.17 [1]. It can run as a single instance or as a client-server architecture.

At the time of writing, three types of sensors are currently supported by CARLA[1]: RGB cameras and pseudo-sensors that provide ground-truth depth, and semantic segmentation. Number and position of sensors can be specified by the client. There is support for 2D and 3D camera modes.

In CARLA, system models (simulation scripts), are saved as *.ini* files. Scripts define characteristics of the simulated environment. For instance: number of pedestrians, number of non-player cars, weather, etc. An excerpt of a simulation script can be found below.

```
%; Example of settings file for
    CARLA.
[CARLA/Server]
UseNetworking=true
WorldPort=2000

[CARLA/LevelSettings]
NumberOfVehicles=15
NumberOfPedestrians=30
WeatherId=1
```

Sensors are set as below. The type of sensor is defined via the *PostProcessing* option as below. Possible values are *SceneFinal* (for a RGB camera), *DepthTruth*, and *SemanticSegmentation*. Each of these visualization provide different information about the image image. Adding support to Lidar sensors is in the project's short-term goals [11].

```
[CARLA/SceneCapture/MyCamera]
PostProcessing=SceneFinal
; PostProcessing= [Depth,
    SemanticSegmentation]

 [CARLA/SceneCapture/
     CameraStereoLeft/RGB]
  [CARLA/SceneCapture/
      CameraStereoRight/RGB]
```

Moreover, additional camera parameters can be defined. For instance, rotation, positioning, field of view (FOV), as well as size of the recorded image, These parameters can be set for each camera individually. See the excerpt below.

A running instance of CARLA is shown in Figure 6. There are three cameras represented in it: a forward-facing stereo camera setting, viewing in *SceneFinal* mode, and a backward-facing 2D camera, viewing in semantic segmentation mode.

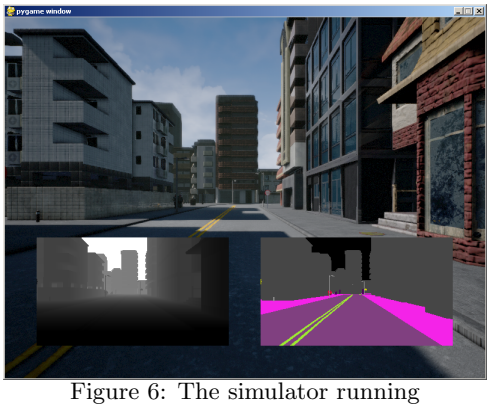

Figure 6: The simulator running

6.1 Simulating Attacks in CARLA

[1]Version 0.7.1 (February 2018).

The formal CSP models introduced in Section 4 are useless without the representation of attacks. Attacks to camera sensors can be visualized as changes to the image perceived by the camera sensor. In CARLA, we model these attacks as new *PostProcessing* effects assigned to each camera definition.

At present, the following effects have been modelled: black screen (Figure 7(a)), intense white LED (Figure 7(b)), and broken lens (Figure 7(c)). The black screen effect is useful to represent a situation in which the assigned camera has gone completely dark. This can be the consequence of either a remote attack shutting off the camera, or simple camera malfunction. Cameras are an easy target by vandals as they are accessible from the outside to the vehicle (i.e. while the vehicle is parked). A white LED effect represents the situation in which a camera is subject to intense lighting, as to mess with the camera's auto-exposure controls [23]. Finally, a broken glass effect simulates the situation in which a camera lens has been broken, followed a physical attack to it. Admittedly, this effect is not realistic enough, as we have skipped modelling light distortion caused by the broken glass.

Attacks, such as the white LED and red laser beam above, can be directed towards any one of the simulated cameras in CARLA. In an stereo-vision setting, for instance, if any of the lenses is attacked, the resulting three-dimensional image can be distorted. There is a need to understand how perception and control system algorithms react to such disturbing events. Making cameras redundant has been proposed as a mitigation technique to physical attacks. However, the close proximity between main and redundant cameras makes it easy for the second to be targeted as well. Cameras can be the target of vandals, while the vehicle is parked, therefore we deem the black screen and broken lens effect feasible.

6.2 Generating Simulation Scripts

From the original simulation script, our tool derives further security test scripts. Generation of simulation scripts iterate over chosen attacks, sensors to which the attack is directed to, non-security-related simulation parameters (e.g. weather). With it, we are able to automatically generate test cases (simulation scripts) for the research questions introduced in Section 2.4.

Some attack scenarios, represented as automatically generated simulation scripts, are illustrated below. In the first, the 2D camera *MyCamera* has broken lens and suffers a white LED attack. These scenarios are modelled in two distinct script files. Following CARLA's client-server architecture, these scripts are used at the client side while retrieving images from the server. We provide a small Python program to integrate the generated simulation scripts into the original client code.

```
; File 1:
[CARLA/SceneCapture/MyCamera]
 PostProcessing=BrokenLens

; File 2:
[CARLA/SceneCapture/MyCamera]
 PostProcessing=WhiteLED
```

In the second example attack scenario, one of the cameras in a bifocal 3D camera setting is totally malfunctioning, rendering a black screen:

```
; File 3: Left camera renders a
    black screen
 [CARLA/SceneCapture/
     CameraStereoLeft/RGB]
  PostProcessing=BlackScreen
[CARLA/SceneCapture/
    CameraStereoRight/RGB]
  PostProcessing=SceneFinal

; File 4: Right camera renders a
    black screen
 [CARLA/SceneCapture/
     CameraStereoLeft/RGB]
  PostProcessing=SceneFinal
[CARLA/SceneCapture/
    CameraStereoRight/RGB]
  PostProcessing=BlackScreen

; File 5a: (both cameras are
    blacked out)
 [CARLA/SceneCapture/
     CameraStereoLeft/RGB]
  PostProcessing=BlackScreen
[CARLA/SceneCapture/
    CameraStereoRight/RGB]
  PostProcessing=BlackScreen
```

7. DISCUSSION

Although attacks to sensors might seem unlikely at the moment, they may become popular as soon as autonomous driving starts to grow. The cost of purchasing some attack materials, e.g. other Lidar scanners, make it impractical for the average attacker. However, this can change. As sensors become cheaper and more widely available, so does tools that exploit vulnerabilities on them.

It is also worth-mentioning that vehicles equipped with autonomous functions still represent a very small portion of the total vehicle fleet in whatever country. In the case of Tesla's Autopilot, although it has run about hundreds of millions of miles in real roads, this is still a small number if compared to the non-autonomous fleet [14]. Although Tesla was praised by the safety of its Autopilot function [19], it unclear whether it would perform equally safely while its sensors are attacked.

The use of Tesla Autopilot has already been involved in two deaths. In one of the accidents' official report [19], it was shown that Autopilot failed to detect the approaching of a heavyweight white trailer while its sensors were exposed to direct bright sunlight . There is a strong parallel here between this incident and the white LED camera blinding attack, idealised and tested by other researchers — and replicated by us via simulation.

Millions of driven miles may not be enough to safeguard against all traffic scenarios, especially those with low-probability of occurrence. Our proposal is that simulation of attacks to sensors be used as a tool to provide autonomous vehicles with increased levels of assurance. There is scope for the list of attacks be expanded, encompassing not only security-related issues, but also safety-related ones. For machine vision, one can imagine simulated blurred lenses, or

(a) Black Screen (b) White LED (c) Broken Lens

even totally malfunctioning cameras. AVs will inevitably be sent to maintenance at some point in their lives due for maintenance. We need to know how well they may perform from detecting the incident until getting to the nearest autonomous repair shop.

The growing link between safety and security is present here[12]. Our options for modelling simulated sensor attacks were constrained by the options offered by current tools(i.e. CARLA had not been implemented support to Lidar scanners).

8. CONCLUSION

In this paper, we have presented a framework to assist with the testing of autonomous vehicles. Our framework is underpinned by a formal model in CSP to guide the test generation process. The testThe formal model is supported by tool and an accompanying API. This API can be used to add support to new simulation software, thus contributing to expand the array of supported applications.

In our case study, we have used our framework and tool to generate cases of test for CARLA, an open-source autonomous driving simulator. We also modelled the effects of attacks to CARLA's simulated cameras to assist the visualization of attacks. By simulating attacks to cars' perception sensors, this will assist with security testing of their systems. Practical, in-the-road testing is expensive and risky. Another benefit of our work is that, by including realistic representations of attacks in-between collected data, we will be able to better train perception algorithms (e.g. object tracking, navigation and localization) to detect and prevent attacks. We were, however, limited by the type of sensors provided by our simulation tool of choice. Have it also had support for other sensors, e.g. Lidar, additional attacks and sensor support would have been added. In fact, we idealised a second experiment, in which the performance of learning algorithms, with and without attacks after the tdata training stage, would be compared.[2].

Limitation. We acknowledge some limitations of the work. Although the generation of test cases can happen automatically, their execution is still manual. This can be easily automated via test suites and test scripts. Second, without a good selection criteria, the state space of CSP traces, and consequently, the number of generated cases may become large. This problem can be circumvented by executing the generated tests in the cloud (or any other high processing computer platforms). As an example, CARLA can be configured to run in client-server mode. We were constrained by the set of sensors currently supported by CARLA. Had other sensors been implemented (e.g. Lidar), we would have developed attacks for these. Moreover, there should also be more options regarding the criterion for selection of tests by the user.

Future work. As future work, the authors will continue adding models of simulated attacks to CARLA. We believe that our testing idea may be applicable to testing the fusion of data. In this case, the system model would take the form of programming code, to be run against the function's implementation. There is also margin for a broader study on the impact of (simulated) attacks to sensors, in which the impact of different attack parameters (e.g. angle, intensity and size of a white LED camera blinding attack) are evaluated in controlled experiments.

What has been proposed in this work – automated generation of security tests for sensors – may already be of knowledge by the automotive industry. However, results of experimental assessment studies lead us to believe this is not yet the case. Due to the high liability of the automotive industry, this will likely remain unknown. Purchasing a car for conducting experimental research, needs disassembling of both hardware and software components, although beneficial, can only be done by well-funded and sourced research groups (as in the first automotive security research: [4, 16]).

[2]However, until the time of this writing, this experiment had not concluded. In the case this paper is accepted, the two previous statements herein will be removed and the experiment may be added.

9. REFERENCES

[1] Unreal Engine 4.17 Released! `https://www.unrealengine.com/en-US/blog/unreal-engine-4-17-released`, Aug. 2017.

[2] A. Bezemskij, R. J. Anthony, G. Loukas, D. Gan, and others. Threat evaluation based on automatic sensor signal characterisation and anomaly detection. 2016.

[3] B. Cai, S. Yang, W. ShangGuan, and J. Wang. Test sequence generation and optimization method based on Cooperative Vehicle Infrastructure System simulation. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 69–74, Oct. 2014.

[4] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, and others. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In {*Proceedings of the 20th USENIX Conference on Security*, SEC'11, San Francisco, CA, USA, Aug. 2011. USENIX Association.

[5] F. Codevilla, M. Müller, A. Dosovitskiy, A. López, and V. Koltun. End-to-end driving via conditional imitation learning. *arXiv preprint arXiv:1710.02410*, 2017.

[6] S. V. E. S. S. Committee. SAE J3061-Cybersecurity Guidebook for Cyber-Physical Automotive Systems. *SAE - Society of Automotive Engineers*, 2016.

[7] Deutscher Bundestag. 18. Wahlperiode, Drucksache 18/11300. `http://dip21.bundestag.de/dip21/btd/18/113/1811300.pdf`, Feb. 2017.

[8] M. Dikmen and C. M. Burns. Autonomous Driving in the Real World: Experiences with Tesla Autopilot and Summon. In *Proceedings of the 8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, Automotive'UI 16, pages 225–228, New York, NY, USA, 2016. ACM.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun. CARLA: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[10] M. R. Endsley. Autonomous driving systems: A preliminary naturalistic study of the Tesla model S. *Journal of Cognitive Engineering and Decision Making*, 11(3):225–238, 2017.

[11] GitHub. CARLA: Open-source simulator for autonomous driving research. `https://github.com/carla-simulator/carla`, Feb. 2018.

[12] B. Glas, C. Gebauer, J. Hänger, A. Heyl, J. Klarmann, S. Kriso, P. Vembar, and P. Wörz. Automotive safety and security integration challenges. volume P-240, pages 13–28, 2015.

[13] ISO. ISO/DIS 26262-1: Road vehicles — Functional safety – Part 1: Vocabulary. Technical report, 2011.

[14] N. Kalra and S. M. Paddock. Driving to Safety. `https://www.rand.org/pubs/research_reports/RR1478.html`, 2016.

[15] C. Kohl, M. Schermann, and H. Krcmar. The Effect of System Restrictions on Acceptance of Self-Driving Cars. 2016.

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

[17] H. Mansor, K. Markantonakis, R. N. Akram, and K. Mayes. Don't Brick Your Car: Firmware Confidentiality and Rollback for Vehicles. In *Availability, Reliability and Security (ARES), 2015 10th International Conference On*, pages 139–148. IEEE, 2015.

[18] S. Nakajima and H. N. Bui. Dataset Coverage for Testing Machine Learning Computer Programs. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 297–304, Dec. 2016.

[19] NHTSA. Investigation:PE16-007. `https://static.nhtsa.gov/odi/inv/2016/INCLA-PE16007-7876.PDF`, Jan. 2017.

[20] Oxford University. FDR4 - The CSP Refinement Checker. `https://www.cs.ox.ac.uk/projects/fdr/`, Nov. 2017.

[21] Y. Peng, Y. Huang, T. Su, and J. Guo. Modeling and verification of AUTOSAR OS and EMS application. In *Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium On*, pages 37–44. IEEE, 2013.

[22] J. Petit and S. E. Shladover. Potential Cyberattacks on Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):546–556, Apr. 2015.

[23] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl. Remote attacks on automated vehicles sensors: Experiments on camera and lidar. *Black Hat Europe*, 11, 2015.

[24] C. Ponikwar, H. Hof, S. Gopinath, and L. Wischhof. Beyond the dolev-yao model: Realistic application-specific attacker models for applications using vehicular communication. *CoRR*, abs/1607.08277, 2016.

[25] ProB. The ProB Animator and Model Checker - ProB Documentation. `https://www3.hhu.de/stups/prob/index.php/Main_Page`, Nov. 2017.

[26] I. Pustogarov, T. Ristenpart, and V. Shmatikov. Using Program Analysis to Synthesize Sensor Spoofing Attacks. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 757–770, New York, NY, USA, 2017. ACM.

[27] Q. Ran, X. Wu, X. Li, J. Shi, J. Guo, and H. Zhu. Modeling and Verifying the TTCAN Protocol Using Timed CSP. In *2014 Theoretical Aspects of Software Engineering Conference*, pages 90–97, Sept. 2014.

[28] B. Roscoe. The theory and practice of concurrency. 1998.

[29] Samuelson-Glushko Technology Law & Policy Clinic (TLPC). Jamming and Spoofing Attacks: Physical Layer Cybersecurity Threats to Autonomous Vehicle Systems. Technical report, Nov. 2016.

[30] S. A. Seshia, D. Sadigh, and S. S. Sastry. Formal methods for semi-autonomous driving. volume 2015-July, 2015.

[31] S. Shah, D. Dey, C. Lovett, and A. Kapoor. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*, 2017.

[32] H. Shin, D. Kim, Y. Kwon, and Y. Kim. Illusion and Dazzle: Adversarial Optical Channel Exploits against Lidars for Automotive Applications. 2017.

[33] B. W. Smith and J. Svensson. Automated and Autonomous Driving: Regulation under Uncertainty. *International Transport Forum 2015*, 2015.

[34] TASS International. PreScan Overview. `https://www.tassinternational.com/prescan-overview`, Aug. 2017.

[35] A. B. A. R. Thomas Gibson-Robinson, Philip Armstrong. FDR3 — A Modern Refinement Checker for CSP. In E. Ábrahám and K. Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 187–201, 2014.

[36] S. Ucar, S. C. Ergen, and O. Ozkasap. Security vulnerabilities of IEEE 802.11 p and visible light communication based platoon. In *Vehicular Networking Conference (VNC), 2016 IEEE*, pages

1–4. IEEE, 2016.

[37] R. W. van der Heijden, S. Dietzel, T. Leinmüller, and F. Kargl. Survey on Misbehavior Detection in Cooperative Intelligent Transportation Systems. *arXiv:1610.06810 [cs]*, Oct. 2016.

[38] J. S. Warner and R. G. Johnston. A simple demonstration that the global positioning system (GPS) is vulnerable to spoofing. *Journal of Security Administration*, 25(2):19–27, 2002.

[39] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[40] C. Yan, W. Xu, and J. Liu. Can You Trust Autonomous Vehicles: Contactless Attacks against Sensors of Self-driving Vehicle. Las Vegas, NV, USA, Aug. 2016.