A Formal Security Assessment Framework for Cooperative Adaptive Cruise Control

Yuri Gil Dantas
fortiss GmbH
München, Germany
dantas@fortiss.org

Vivek Nigam
fortiss GmbH
München, Germany
nigam@fortiss.org

Carolyn Talcott
SRI International
Melno Park, USA
clt@csl.sri.com

***Abstract*—For increased safety and fuel-efficiency, vehicle platoons use Cooperative Adaptive Cruise Control (CACC) where vehicles adapt their state, incl. speed and position, based on information exchanged between vehicles. Intruders, however, may carry out attacks against CACC platoons by exploiting the communication channels used to cause harm, e.g., a vehicle crash. Therefore, during design-phase, engineers should provide evidence supporting platoon security. This paper proposes a formal framework for the security verification of CACC platoons to provide such evidence based on precise mathematical models. Our vehicle platoon models support the specification of both cyber, e.g., communication protocols, and physical, e.g., speeds, position, vehicle behaviors. Moreover, we propose intruder models that are parametric on his capabilities of manipulating communication channels, i.e., message injection and blocking. Our model is implemented enabling the automated formal verification involving both platoon and intruder models. We validate our machinery with a number of attacks taken from the literature and novel attacks discovered by using our formal machinery.**

***Index Terms*—attacks, formal verification, platoon, security**

I. Introduction

Cooperative Adaptive Cruise Control (CACC) increases fuel-efficiency [21] and safety of vehicle platoons [2], typically heavy-weight cargo vehicles (*e.g.*, trucks). This is accomplished by reducing vehicle reaction times by relying on information, such as speed, direction and position, exchanged between vehicles in addition to the vehicles' sensors.

The use of CACC also greatly increases a platoon's attack surface as communication channels may be exploited by intruders. For example, as pointed out by [22], intruders may inject messages with false information into the CACC communication channels leading to vehicle crashes, thus causing harm and financial losses. Intruders can carry out such attacks for financial motivations to, *e.g.*, steal the transported cargo.

Designing secure systems is challenging as intruders may carry out attacks by exploiting corner-cases or implicit requirements overseen by developers. For example, a number of communication protocols have been shown to be vulnerable to attacks, some of which have been discovered decades after they have been developed [13]. The safety and security of vehicle platooning have the additional complexities of cyber-physical systems, including speed, time to react, and position. Engineers have to ensure that intruders cannot exploit these aspects, as in the injection attacks described by [22].

This paper proposes the use of formal verification as a means to provide further evidence about the security of platoons using CACC. An advantage of formal verification over, *e.g.*, simulation analysis, lies on the fact that its methods are based on precise mathematical models that specify the behavior of the analyzed system. By using formal verification, implicit requirements are made explicit thus exposing existing vulnerabilities. Moreover, from such models, automated tools can determine whether undesired events are possible by traversing all behaviors including corner-cases.

Existing formal frameworks for platooning [8], [10] and other agent-based cyber-physical systems [19], [20] have successfully been used to verify the safety of agent-based cyber-physical systems, such as platoon joining maneuvers and strategies used by Unmanned Aerial Vehicles [15]. These frameworks, however, do not take into account security aspects. They do not include intruders and therefore, it is not possible to verify in such frameworks whether an intruder may attack a system and cause harm, *e.g.*, a vehicle crash.

To the best of our knowledge, this paper proposes the first formal framework to consider platooning, CACC and security. Our main contributions are three-fold:

- **Vehicle Platoon Behavior Specification:** Our first contribution is a platoon model that includes specifications of both cyber aspects, *e.g.*, specifications for the communication protocols, and physical aspects, *e.g.*, speed, acceleration, positions of vehicles. Our model enables the specification of a wide range of vehicle strategies for executing platooning based on soft-constraints [4], a general algebraic framework for specifying optimization problems. That is, our model can accommodate a number of strategies including those expressed as classical, fuzzy and probability theories and their combination. For example, strategies for maintaining distances between vehicles that are both safe and fuel-efficient can be reduced to an optimization problem based on soft-constraints.
- **Intruder Models:** Our second contribution consists of formal intruder models that subvert communication channels to carry out attacks. These intruder models are parametric on the intruder capabilities, *i.e.*, the capability of either blocking messages from a communication channel or injecting messages into communication channels.
- **Automated Verification:** Our third contribution is the

implementation of our models, both platoon and intruder models, in Maude [5], an efficient formal verification tool based on Rewriting Logic. Our specification are executable. That is, users can automatically invoke Maude's search mechanisms to formally verify their platooning specifications for the verification of safety, *e.g.*, vehicles not crashing, by taking into account security, *e.g.*, in scenarios where an intruder may block or inject messages.

We validate our machinery in realistic scenarios, some taken from the literature [9], [22], and some new attacks that have been discovered by using our formal framework.

II. Attacks

This section describes both the threat model and a set of possible attacks scenarios against a CACC platoon. To the best of our knowledge, some of the attack scenarios, namely, those described in Sections II-D, II-E and II-F, are new. They have been discovered using formal verification, as potential breaches became clearer after formalizing the platoon model, in particular its communication protocols and the modes (or roles) in which a vehicle can operate.

A. Threat Model

We consider a CACC platoon, with one leader and n followers, where new vehicles may join the platoon after a negotiation phase. We assume that the platoon vehicles navigate on a straight road, and that vehicles can communicate using peer-to-peer connections or by broadcasting messages. We also assume that all messages are signed using vehicles secret keys that cannot be guessed by intruders, and contain adequate measures to ensure freshness, such as using timestamps or nonces, to avoid replay attacks.

The goal of our intruder is to cause a crash between two legitimate vehicles. To this end, the intruder either injects false messages into the CACC communication channels or jams (*i.e.*, blocks) legitimate messages from the CACC communication channels. The actual capability used by the intruder depends on the attack scenario. We consider scenarios where the intruder (1) injects false messages only, (2) blocks messages only, and (3) both injects and blocks messages. To ensure that injected messages are valid, we assume that the intruder is able to obtain encryption keys from any vehicle in the platoon. The same assumption is considered by previous related work like, *e.g.*, [22] and [9]. For simplicity, we assume that the intruder has obtained the leader's encryption key.

Given the leader's encryption key, the intruder makes valid connections with a target vehicle (*i.e.*, a follower or a joining vehicle). For example, assume an attack scenario where both capabilities (*i.e.*, injecting and blocking) are required. The intruder blocks all messages originated from the leader and injects (impersonating the leader) false messages to either followers or vehicles joining the platoon.

B. Injection of False Msgs against Follower

In this attack, an intruder sends false position and speed values to a vehicle in order to cause a crash with the preceding vehicle. This attack works because CACC algorithms ensure that a vehicle maintains a desired distance from the preceding vehicle based on the received messages from other vehicles in the platoon (especially from the leader). This attack has been previously demonstrated through simulations by *e.g.*, [22].

The attack scenario is illustrated in Figure 1a. This scenario is composed of two vehicles: a leader (*ldr*) and a follower (flw_1). Illustrated by the green arrows, such vehicles exchange information to ensure that flw_1 keeps a safe distance from *ldr*. The red cross illustrates that the legitimate messages from the leader are blocked by the intruder while the attack is in progress. Next, the intruder impersonates *ldr* to send high position and speed values to flw_1. The follower flw_1 adapts its distance based on the high false values sent by the intruder. As a result, a crash between flw_1 and *ldr* is expected, as illustrated by the right-hand side of Figure 1a.

C. Slow-Injection of False Msgs

The goal of the previous attack (Section II-B) is a quick crash between two vehicles. To this end, the intruder injects extreme false position and speed values into the CACC communication channels. As discussed by [22] and [9], however, existing countermeasures (a.k.a plausibility checks) are able to detect such extreme values, and thus mitigate the attack.

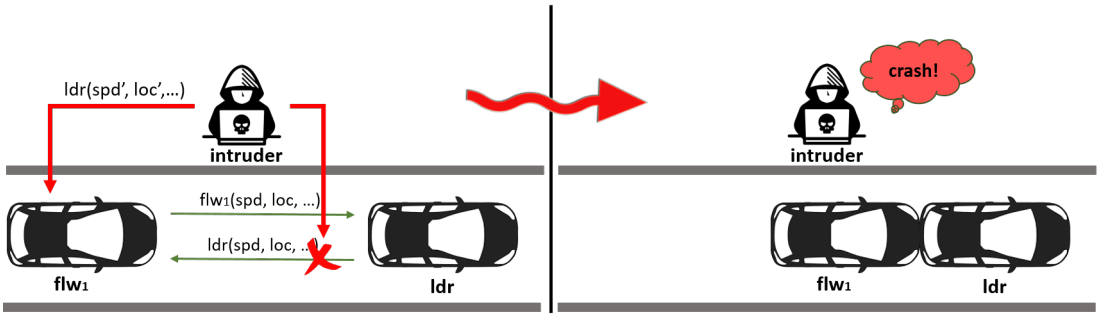
Recently, [9] proposed a smarter variation of the previous attack in order to bypass existing countermeasures that checks whether incoming values highly deviates from the previous received ones. To this end, the intruder injects messages with false information into the CACC communication channels modifying the values of speed and position with a small increase rate after each message. This attack has been demonstrated through simulations by [9].

D. Injection of False Msgs against Joining Vehicle

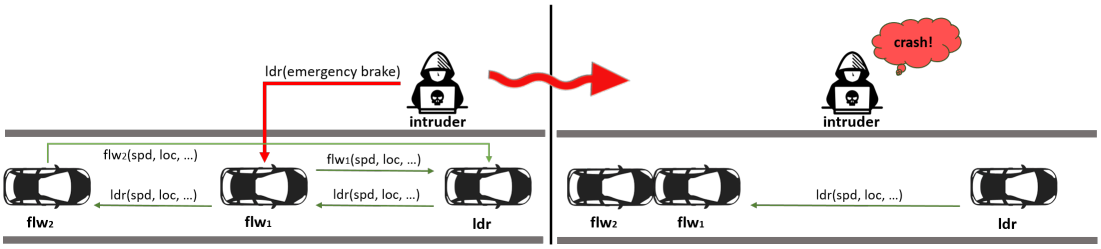
A new vehicle may join a platoon after a negotiation phase (a.k.a synchronization handshake) with the leader of the platoon. During this negotiation phase, the leader sends the platoon information to this vehicle, including the position and speed of the last vehicle, so that the joining vehicle can adapt itself to catch up to the platoon.

An intruder may impersonate the leader to send false information during this negotiation phase. For example, assume an attack scenario composed of two vehicles: the leader (*ldr*) of the platoon and a vehicle (*veh*) that wishes to join the platoon. The intruder may inject (as *ldr*) high position and speed values to *veh* during the negotiation phase, while blocking all messages originated from *ldr*. Eventually, *veh* crashes into *ldr*, as *veh* adapts its acceleration based on the received values.

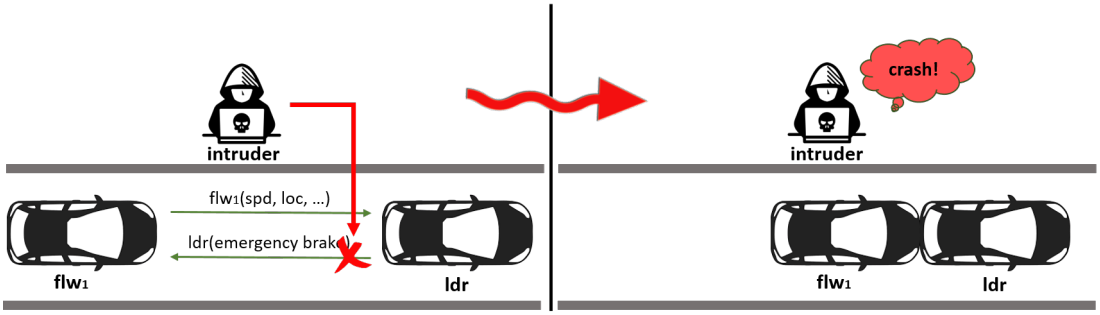
Countermeasures against injection attacks usually check on messages exchanged between platoon members (*e.g.*, followers). The attack scenario presented above targets a vehicle that has not yet joined the platoon. Hence, this attack would be successfully carried out against such countermeasures. To the best of our knowledge, this is the first attack scenario targeting a vehicle before joining the platoon.



(a) Injecting false msgs to follower and blocking legitimate msgs from leader



(b) Injecting false emergency brake to follower



(c) Blocking legitimate emergency brake from leader

Fig. 1: Illustration of three attacks (before and after the attacks have been) carried out by the intruder

E. Injection of False Emergency Brake Msgs

Emergency brake is a safety-type message that may be triggered by any vehicle in the platoon to avoid crashes. For example, the leader may trigger an emergency brake if an obstacle is detected in its path. Then each follower receives an emergency brake message from the leader, and immediately actuates it by stopping the vehicle.

An intruder, however, might take advantage of this situation to carry out attacks. Figure 1b illustrates an attack scenario using emergency brake messages. This scenario is composed of three vehicles: a leader (*ldr*) and two followers (flw_1 and flw_2). The goal of this attack is a crash between flw_1 and flw_2. To this end, the intruder injects a false emergency brake message to flw_1 only. This message results in a crash as flw_1 immediately stops and flw_2 keeps driving, yet following the previously received information (*e.g.*, position and speed). The crash is illustrated by the right-hand side of Figure 1b. Our hypothesis is that the intruder does not need to block messages from the leader in order to successfully carry out this attack.

F. Blocking Legitimate Emergency Brake Msgs

Instead of injecting false emergency brake messages, the intruder may block legitimate emergency brake messages from the CACC communication channels in order to cause a crash. An attack scenario with this purpose is illustrated in Figure 1c.

The intruder monitors the channels till a legitimate emergency brake message is triggered by the leader (*ldr*). At this point, *ldr* stops the vehicle and the intruder blocks the message to avoid that any follower (flw_1) can receive and trigger emergency brake as well. As a result, flw_1 keeps driving the vehicle till crashing into *ldr*. This crash is illustrated on the right-hand side of Figure 1c. This attack scenario is another result from our formal verification.

III. Soft-Agents Model for Platooning

Soft-Agents [19] is a rewriting logic framework for the specification and verification of (autonomous) cyber-physical

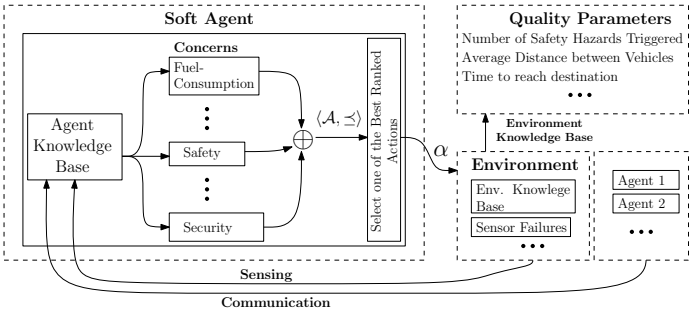


Fig. 2: Soft-Agent Architecture

agents. The framework can be found at [19], [20]. The framework is implemented in the rewriting logic language Maude [5]. It provides the general machinery (data-structures, functions, sorts) for the specification of the behavior of agents, *e.g.*, agent capabilities and effects of actions. The semantics of how the system evolves is specified by a small number of rewrite rules defined in term of the general machinery.

Figure 2 depicts the general architecture of a soft-agent, or simply agent. An agent has its own local knowledge base that contains, *e.g.*, its current perceived speed, position, and direction of the other agents. Further data may be obtained by sensing the environment or by sharing of information between agents through communication channels. Using its local knowledge base, the agent decides which action (α) to perform according to its different concerns specified as a soft constraint (optimization) problem [4]. For example, if the distance to the vehicle in front is too great, the fuel consumption concern kicks in and attempts to reduce it by accelerating. Similarly, if the distance is dangerously short, then the safety concern kicks in and attempts to increase it by decelerating. As soft constraints subsume other constraint systems, *e.g.*, classical, fuzzy and probabilistic, it is possible to formally specify a wide range of decision algorithms.

A. Platooning Model

We instantiated the general framework (data structures, sorts, types, soft constraints) provided by the Soft Agents framework for specifying platoon scenarios, enabling their formal verification. While the complete implementation can be found at [6], we describe some of this machinery below.

Knowledge Base: Vehicles have a local knowledge base (`lkb`). It represents the vehicle's view of the world, *e.g.*, the speed and position of itself and of the other vehicles. Formally, a vehicle knowledge base is composed by a set of grounded facts, p, *i.e.*, facts not containing variable symbols, of the form p, or associated with a timestamp, $p@t$, where t is natural number. We list the main facts below. We assume that each vehicle has a unique identifier written `id`.

- `clock(t)` denotes that the current time is `t`;
- `atloc(id,pos) @ t` denotes that the vehicle `id` has at time `t` the position of value `pos`. We assume that vehicles navigate on a straight road. Therefore, `pos` is a value representing the position on this road.
- `speed(id,spd) @ t` denotes that the vehicle `id` has at time `t` the speed of value `spd`;
- `maxAcc(id,acc)` denotes that the vehicle `id` can accelerate (and for simplicity also decelerate) at any time with value `acc`.
- `platoon(idL,[id1,...,idn]) @ t` denotes that at time `t`, the platoon led by `idL` has the sequence of follower vehicles `id1,...,idn`;
- `mode(id,md) @ t` denotes that the vehicle `id` at time `t` is in mode `md` which include: `nonplatoon` when all the vehicle's platooning functionalities are not active, *i.e.*, the vehicle is driven by a human driver; `leading()` when the vehicle leads a platoon; `following(idL)` when `id` is following the platoon led by `idL`; `emergency` when `id` is in emergency brake mode; `fuseRear(idL,idB)` when `id` is in the process of joining platoon led by `idL` and shall join be behind vehicle `idB`.
- `safe(id,min,max)` denotes that the distance to the preceding vehicle of `id` is considered safe if it is between the values `min` and `max`;
- `fuel(id,min,max)` denotes that the distance to the preceding vehicle of `id` is considered to be fuel efficient if it is between the values `min` and `max`;
- `histSpd(id,id1,spd1;...;spdn) @ t` denotes that vehicle `id` has the n last speed values, `spd1;...;spdn`, of vehicle `id1`. This fact is used to build plausibility checks as detailed in Section V.
- `histGap(id,gap1;...;gapn) @ t` denotes that vehicle `id` has the n last gap measurements, `gap1;...;gapn`, to the following vehicle.

Example 3.1: The following local knowledge base of vehicle `v(1)` specifies that he is following vehicle `v(0)`. The vehicle `v(1)` has speed 20 and position 945 distance units. He believes to be immediately behind vehicle `v(0)` with a gap of 55 distance units. The vehicle `v(1)` has a maximum acceleration of 3 acceleration units. Moreover, he keeps track of the three last speed values, 25, of `v(0)`.

```
lkb  : (clock(3) (atloc(v(1),loc(945)) @ 3)
 (mode(v(1),following(v(0))) @ 3)
 (speed(v(1),20) @ 3) (gapNext(v(1),55) @ 3)
 (idNext(v(1),v(0)) @ 3) maxAcc(v(1),3)
 (histSpd(v(1),v(0),25 @ 3; 25 @ 2; 25 @ 1) @ 3)
 (histGap(v(1),55 @ 3; 55 @ 2; 55 @ 1) @ 3)
 fuel(v(1),1,3) safe(v(1),2,4))
```

Sensors: A vehicle is equipped with three sensors `locS`, `speedS` and `gapS`. They measure, respectively, the vehicle's location, speed and the gap to the vehicle immediately ahead. As we illustrate below, at each tick, vehicles use these sensors to query the environment knowledge base and update the vehicle's local knowledge base. While it is not the focus of this work, it is possible to evaluate the robustness of agents with respect to sensor faults as described in [15].

Communication Channels and Protocols: We assume that vehicles may communicate using peer-to-peer connections or by broadcasting messages. Based on this assumption, we implement a number of protocols for platooning including:

- Heartbeat from Follower to Leader (HFL): A follower

vehicle sends periodically a (time-stamped) message to the leader with information such as its current speed, position.
- Heartbeat from Leader to Follower (HLF): The platoon leader sends periodically a message to each follower with information of all vehicles in the platoon such as their speeds and positions.
- Emergency Brake: Any vehicle in the platoon may broadcast an emergency brake message informing that it is activating its emergency brakes.
- Heartbeat from Joining Vehicle to Leader (HJL): A vehicle that wants to join a platoon sends a heartbeat to the platoon leader, such as its current position and speed.
- Heartbeat from Leader to Joining Vehicle (HLJ): The platoon leader sends to the vehicle that is joining the platoon information, such as the position and speed of the last vehicle in the platoon.

Actions: Vehicles decide to accelerate or decelerate. Since there may be infinitely many possibilities of acceptable speeds (for safety and fuel efficiency), we abstract actions by using facts of the form `act(id,vmin,vmax)` denoting a set of actions of changing `id`'s speed to values between `vmin` and `vmax`. Actions are evaluated with a value that is the result of a soft constraint problem specification described next.

Soft Constraints: The evaluation of possible actions is done by taking account the vehicle's concerns specified as a soft constraint problem. To evaluate our verification machinery, we implemented a strategy that depends on the vehicle's mode.

When in following mode, a vehicle has two main concerns: Fuel-Saving and Safety. The former attempts to close the gap to the vehicle immediately in front, while the latter attempts to keep a safe distance to the vehicle immediately in front. These are specified by the knowledge items `safe` and `fuel`.

Our machinery uses these two parameters to determine which (set of) actions are the most highly ranked. This is accomplished by attempting to satisfy both concerns, safety and fuel-saving. If this is not possible, then safety is given priority over fuel-saving. When in emergency mode, the vehicle has only the concern of stopping the vehicle.

Example 3.2: Assume the knowledge-base of vehicle `v(1)` in Example 3.1. Since its maximum acceleration is 3 and its current speed is 20, it may choose any speed between 17 and 23. From the safety concern, it attempts to keep a distance (in terms of time) between 2 and 4. From the current gap of 55 units and speed of the vehicle in front of 25, all speeds between 17 and 23 are acceptable. However, the fuel-saving concern attempts to keep a distance (in terms of time) between 1 and 3. Since the gap is too great and the vehicle in front is faster, only speeds between 75/4 and 23 are acceptable. The vehicle picks the average speed of 20.75, *i.e.*, `v(1)` accelerates.

Vehicle Configuration and System Configuration: A *vehicle configuration* contains its local knowledge base (`lkb`), sensors (`sensors`), and the events (`evs`) that are to be processed. The events are used to define the execution semantics described in Section III-B.

Example 3.3: The following is an example of a vehicle configuration for `v(1)`, where `LKB` is the local knowledge base in Example 3.1. Finally, it has a single event `tick @ 0` that specifies that this vehicle is ready to observe the state and schedule new actions.

```
[v(1) : veh | lkb : LKB
  sensors : (locS speedS gapS),
  evs : (tick @ 0)]
```

Finally, a *system configuration* is composed of a collection of vehicle configurations, `vconfi`, for $0 \leq i \leq n$, and an environment `[eId | kb]`: { `[eId | kb] vconf1 ... vconfn` }. The environment knowledge base, `kb`, contains the true state of the world which may be different to the information of the local knowledge bases of vehicles.

B. Executable Semantics

The execution semantics of our platooning model follows the general semantics described in [19], [20]. Formally, an execution is a finite sequence of system configurations, written $\mathcal{S}_0 \longrightarrow \mathcal{S}_1 \longrightarrow \cdots \longrightarrow \mathcal{S}_n$, where each transition $\mathcal{S}_i \longrightarrow \mathcal{S}_{i+1}$ follows the executable semantics described below. In practice, an execution can be constructed in an automated fashion using the rewriting tool Maude [5]. We illustrate the semantics by using the platooning model described above.

The execution semantics follows an event-based approach. Vehicles have events of form `ev @ t` denoting an event `ev` that should be executed after `t` time units. If `t` is zero, then it is executed immediately. There are two types of events: tasks and actions. All events of the form `task @ 0` are executed, typically producing actions to be executed and new tasks for later execution. Suppose the smallest non-zero task delay is `d`. Then `d` time units pass as follows: execute all actions with zero delay; update the configuration to pass one unit of time; repeat until `d` units of time have passed.

Consider as an example the initial system configuration with two vehicles `v(0)` and `v(1)`, where `LKB` is as in Example 3.1 and `LKB1` is similar, just that the facts for the platoon, location and speed of `v(0)` are as specified, respectively, by `(platoon(v(0),v(1)) @ 0) (atloc(v(0),loc(1000)) @ 0) (speed(v(0),25) @ 0)`:

```
{ [eId] | kb ]
[v(0) : veh | lkb : LKB1,
  sensors : (locS speedS gapS),
  evs : (tick @ 0)]
[v(1) : veh | lkb : LKB,
  sensors : (locS speedS gapS),
  evs : (tick @ 0)] }
```

At this point, the system checks for tasks of each vehicle, namely the following tasks:

1) Update their local knowledge base with the information extracted by the sensors, location, speed and gaps to vehicles in the front.
2) Executes their soft constraint machinery to determine which range of actions they shall perform. Since `v(0)` is the leader, it decides to maintain its speed at 25. This is specified by the event {`u(1),actSpeed(v(0),22,28)`} `@ 0` specifying that any speed between 22 and 28

is ranked as maximum denoted by `u(1)`. Thus the vehicle picks the average 25. On the other hand, as described in Example 3.2, `v(1)` computes the event {`u(1),actSpeed(v(1),75/4,23)`} `@ 0)`, choosing to accelerate to speed 20.75.

3) Finally, the vehicles follow the communication protocols described in Section III-A. For this example, the leader vehicle `v(0)` creates an event to send a heartbeat to `v(1)`:
`(actSnd(v(0),msg(v(0),v(1),`
`hbl2f(v(0),25,loc(1000),kb) @ 0) @ 0)`
denoting the action to send a message from `v(0)` to `v(1)` containing as payload a heartbeat from leader to follower (`hbl2f`) with `v(0)`'s current speed and location, and `kb` that is a collection of facts which is elided.
Similarly, the following vehicle `v(1)` creates an event to send a heartbeat to the leader `v(0)` containing `v(1)`'s speed and position.

The following step in the execution semantics is to advance time and carry out all these events. This results in updating the speed of the vehicles, and sending both heartbeat messages. The contents of speed and location in these messages are then processed, updating the vehicle's local knowledge bases.

C. Safety Verification (w/o Intruders)

We illustrate next how we can use the machinery described above to reason about the behavior of platoons and demonstrate properties. In particular, we are interested in determining whether two vehicles can crash with each other.

A vehicle crash may happen if the control measures are not adequately set or its assumptions are not met. Example 3.4 illustrates how our machinery can be used to demonstrate this. Moreover, an intruder may exploit the communication channels to cause an accident, as shown in Section IV.

Example 3.4: Consider the system configuration, `S0`, from the previous section with two vehicles `v(0)` leading the platoon, and `v(1)` following `v(0)`. Assume the same parameters for the concerns `safe` and `fuel` as in Example 3.1.

Formally, an execution leads to a crash if it leads to a system configuration such that the location of `v(0)` is less or equal to the location of `v(1)`. Let `crash(S)` return true if `S` consist of system configuration with a crash and false otherwise.

We can use the following command in Maude:

```
search[1] S0 => S1 such that crash(S1) .
```

to search for an execution starting at `S0` and ending at a configuration `S1` such that `crash(S1)` returns true, *i.e.*, a configuration where the vehicles `v(0)` and `v(1)` crash.

Running this command, Maude does not find any such `S'`, thus providing evidence that the parameters for `safe` and `fuel` are correctly set.

However, if we use `S0'`, where the speed of `v(1)` is 40 instead of 20, and run the command

```
search[1] S0' => S1 such that crash(S1) .
```

then Maude finds in 52ms a configuration `S1` where a crash happened. This result means that the chosen parameters do not work w.r.t. safety. Indeed, one could expect a crash when the speed of a vehicle is much greater than the speed of its preceding vehicle.

By using this type of reasoning, engineers can verify whether the specified behavior of platoon is safe according to the assumptions used, recalibrating concerns whenever needed. In the example above, one should make sure that the vehicle `v(1)` is not much greater that the speed of `v(0)`.

IV. Intruder Model

This section introduces an intruder model, formalizing the threat model discussed in Section II. The intruder is capable of impersonating an honest vehicle, injecting messages, and blocking message from communication channels. These capabilities enable us to carry out similar verification done for safety, but now considering a malicious intruder. For example, it is possible to analyze whether platoons are vulnerable to the attacks enumerated in Section II.

Our intruder model is similar to [17], for the security verification of Industry 4.0 applications, in that the intruder model is parametrized by its capabilities. Here we consider two capabilities: injecting messages signed by honest participants and blocking specific messages from communication channels.

An intruder model (`intSpec`) is integrated to a system configuration `system`, forming an *intruder configuration*: { `system ; intSpec` }, where `intSpec` has the following shape:

```
[iid : intruder | (v2vMsgsL : msgList)
                  (blockActSnd : ids) caps]
```

It contains the intruder id `iid`; the sequence of messages, `msgList`, that the intruder may inject; and the vehicles, `ids`, whose output communications are blocked by the intruder.

The execution semantics described above is extended to accommodate the intruder:

- **Message Injection (INJ):** The intruder may choose *at any moment* of a system execution to inject the first message, `msg`, in its list of messages `msgList`. This results in the injection of `msg` to its destination in the system configuration `system`, and the list `msgList` is updated by deleting `msg`.
- **Blocking (BLK):** The vehicles in `ids` are jammed during the whole attack execution. This means that all outgoing messages of a vehicle in `ids` are blocked.

Our model is parametric w.r.t. the intruder capabilities. It requires little effort to include other capabilities to the intruder model in `caps`. For example, it is possible add capabilities where the intruder tampers, *i.e.*, modifies messages sent by vehicles; or periodically sends messages from a set of messages, instead of in a list; or only starts blocking a message after some particular time has elapsed.

As we describe in Section V, however, the intruder can carry out all the attacks described in Section II using the two capabilities specified above. The following example illustrates how one of the attacks can be modeled using our machinery:

Example 4.1: Consider the system described in Section III-A with two vehicles `v(0)` and `v(1)`, with, respectively, speeds

	Attack Scenario	Capability	Countermeasure	# States	Execution Time (min)	Attack Successful
II-B	Injection of False Msgs against Follower	INJ + BLK	N	15351	0.034	Y
	Injection of False Msgs against Follower	INJ + BLK	COMM	-	120	-
	Injection of False Msgs against Follower	INJ + BLK	SNSR	-	120	-
	Injection of False Msgs against Follower	INJ	N	-	120	-
II-C	Slow-Injection of False Msgs against Follower	INJ + BLK	N	3315284	52.764	Y
	Slow-Injection of False Msgs against Follower	INJ + BLK	COMM	3286681	53.251	Y
	Slow-Injection of False Msgs against Follower	INJ + BLK	SNSR	-	120	-
	Slow-Injection of False Msgs against Follower	INJ	N	-	120	-
II-D	Injection of False Msgs against Joining Vehicle	INJ + BLK	N	9408	0.023	Y
	Injection of False Msgs against Joining Vehicle	INJ + BLK	COMM	9408	0.027	Y
	Injection of False Msgs against Joining Vehicle	INJ + BLK	SNSR	9407	0.023	Y
	Injection of False Msgs against Joining Vehicle	INJ	N	-	120	-
II-E	Injection of False Emergency Brake Msgs	INJ + BLK	N	593	0.002	Y
	Injection of False Emergency Brake Msgs	INJ	N	2218	0.011	Y
II-F	Blocking Legitimate Emergency Brake Msgs	BLK	N	6539	0.013	Y

TABLE I: Evaluation of the attack scenarios described in Section II. Some experiments were aborted after 120 minutes.

20 and 25, and locations, 1000 and 945. Moreover, consider the intruder with a single message:

```
msg(v(0),v(1),hbl2f(v(0),70,loc(1070), none))
```

The intruder impersonates `v(0)` informing `v(1)` that his speed is 70 and location is 1070. Since `v(1)` does not double check the contents of this message, it will start accelerating. This will lead to a crash as the actual speed of `v(0)` is 20.

This can be determined in an automated fashion using the following Maude's search command, where `isys0` is the intruder configuration described above:

```
search isys0 =>* isys1 such that crash(isys1) .
```

Maude finds an instance of the intruder configuration `isys1` in which `v(0)` and `v(1)` crash.

V. Verification Results

Our goal is to evaluate our machinery on a number of attack scenarios, including the ones described in Section II. To this end, we formalized such attack scenarios using the intruder model presented in Section IV. We run the search command in Maude to automatically check whether two vehicles crash under the presence of the intruder. We run all experiments on a 1.90GHz Intel Core i7-8665U with 16GB of RAM running Ubuntu 18.04 LTS with kernel 5.4.0-47-generic and Maude 3.

Table I summarizes our main results. It contains each attack scenario described in Section II, the capability used by the intruder, *i.e.*, either injection (INJ) or blocking (BLK), whether a countermeasure has been used against the attack, the number of states explored, the execution time of each search command, and whether the intruder was able to bypass any countermeasure and successfully cause a crash between two vehicles.

For the attacks described in Sections II-B, II-C, and II-D we evaluate their effectiveness with and without a plausibility countermeasure. We formalized two types of countermeasures, abbreviated in Table I as COMM (communication-based) and SNSR (sensor-based) based on similar countermeasures proposed by [9]. The communication-based countermeasure works as follows. Whenever a vehicle receives a message with the speed of the preceding vehicle, the countermeasure checks it against the local `histSpd`. The countermeasure is triggered if the incoming speed value deviates from 30% w.r.t. the average of the last `n` speed values received by the vehicle. The sensor-based countermeasure estimates the speed of the preceding vehicle based on the information obtained from the gap sensor. That is, we estimate the speed of the preceding vehicle by computing `(spd + (gap2 - gap1))`, `spd` as the speed of the vehicle and `gap2` and `gap1` as the last two gap distance measurements (taken from the local `histGap`). The sensor-based countermeasure is triggered if the incoming speed value deviates from 30% w.r.t. the estimated speed of the preceding vehicle.

Our intruder using both capabilities has successfully carried out the attacks II-B, II-C, and II-D against a platoon without countermeasure. The attack II-B, however, has not led to a crash when the countermeasures were deployed. In fact, this result was expected as attack II-B sends high speed values to a target vehicle. We run the search command to look for a crash between two vehicles without the countermeasure being triggered. We could not find any crash (in 120 minutes).

The attack II-C bypassed the communication-based countermeasure, but not the sensor-based countermeasure. This result confirms the findings of [9] that feeding countermeasures with information from local sensors can be effective against slow-injection attacks. Next, the attack II-D led to a crash even when the countermeasures were deployed. The attack II-D is carried against a vehicle during the negotiation phase. This attack is effective as the considered countermeasures are only valid for platoon members (using their local `histSpd` or `histGap`).

Interestingly, neither of those three attacks led to a crash using the injection capability only (*i.e.*, no blocking at the same time). This happens because the target vehicle receives legit and false messages during the attack, and dynamically adapts its acceleration based on the received messages. That is, the target vehicle accelerates when receiving a false message from the intruder, *e.g.*, with high speed values, and decelerate when receiving legit messages from the leader. Therefore, we speculate that anti-jamming countermeasures, *e.g.*, UFH-UDSSS [18], could serve as an additional layer of defense

against injection attacks for CACC platoons.

Finally, both the attacks using emergency brake messages led to a crash. In particular, the attack II-E is effective even without blocking messages from the communication channels. This is due to the fact that vehicles immediately stop driving upon receiving an emergency brake message regardless of any message (usually blocked by the intruder) sent by the leader.

VI. Related Work

We have been inspired by [3], [22], [9] that investigated the attacks described in Sections II-B and II-C. This paper advances these previous work by proposing a formal framework enabling engineers to formalize platoon and intruder behaviors, and formally verify these models in an automated fashion. Our framework provides evidence for the security of platooning based on precise mathematical models, thus complementing the evidence obtained with the use of simulation based methods as in [3], [22], [9]. Further, we also propose three new attacks that have not been considered before.

A number of formal frameworks have been proposed for the specification and verification of cyber-physical agents [8], [10], [14], [15], [19], [20] including for vehicle platooning [8], [10], [14]. A key difference is that our work also considers how intruders may cause harm, whereas these existing frameworks focused on the safety of systems without considering security and intruder models. For example, [8] uses Statistical Model Checking to evaluate the impact of sensor and network faults to the safety of systems. We have in our previous work [15] also used Statistical Model Checking together with the Soft Agents framework for the verification of UAV strategies in the presence of sensor faults. We find it intriguing and leave it to future work the combination of intruder and fault models and their verification techniques for vehicle platooning.

A countermeasure against false position values has been proposed by [12]. It employs low-power beaconing messages to check whether incoming messages indeed comes from physically close vehicles, thus mitigating remote attacks from intruders not located near to platoon members. It seems possible to extend our model to accommodate such aspects following our previous work on Cyber-Physical Security Protocols [16]. We leave this investigation to future work.

VII. Conclusion

This paper proposes to the best of our knowledge the first formal security framework for the specification and verification of vehicle platoon using CAAC. Our model can express communication protocols used by vehicles to exchange information about their physical states, such as speed and position, and can express vehicle behavior including how information exchanged is used to make decision about speed and position. Finally, our framework has an intruder model that is parametric to capabilities, *i.e.*, message injection and blocking. We demonstrated the effectiveness of our framework by formalizing existing attacks and proposing three new attacks.

We are considering a number of future work directions. In particular, we are considering other physical features, *e.g.*, the use of Cyber-Physical Security Protocols [16] to enable verification with proximity-based counter-measures [12]. We are implementing further intruder capabilities that reflect the capabilities of the Dolev-Yao model [7], but taking care not to fall into undecidable verification problems. We are also considering abstract models, such as those considered in [11], to enable completeness of our automated verification. We are extending our framework to also support reasoning with fault models as in [15]. Finally, we are integrating our machinery into an existing MBS tool, namely AutoFOCUS 3 [1].

References

[1] AF3 – AutoFOCUS 3. More information at https://af3.fortiss.org/.

[2] White paper: Automated driving and platooning issues and opportunities. Available at https://tinyurl.com/yxzepft3, 2015.

[3] M. Amoozadeh, A. Raghuramu, C. Chuah, D. Ghosal, H. M. Zhang, J. Rowe, and K. N. Levitt. Security vulnerabilities of connected vehicle streams and their impact on cooperative driving. *IEEE Commun. Mag.*, 53(6):126–132, 2015.

[4] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *LNCS*. Springer, 2007.

[6] Y. G. Dantas, V. Nigam, and C. Talcott. https://github.com/ygdantas/SoftAgents-Platoon.git. 2020.

[7] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[8] S. Hyun, J. Song, S. Shin, and D. Bae. Statistical verification framework for platooning system of systems with uncertainty. In *APSEC*, pages 212–219. IEEE, 2019.

[9] M. Iorio, F. Risso, R. Sisto, A. Buttiglieri, and M. Reineri. Detecting Injection Attacks on Cooperative Adaptive Cruise Control. In *2019 IEEE Vehicular Networking Conference, VNC*, pages 1–8, 2019.

[10] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres. Formal verification of autonomous vehicle platooning. *Sci. Comput. Program.*, 148:88–106, 2017.

[11] M. I. Kanovich, T. B. Kirigin, V. Nigam, A. Scedrov, and C. L. Talcott. Time, computational complexity, and probability in the analysis of distance-bounding protocols. *Journal of Computer Security*, 25(6), 2017.

[12] H. Kim and T. Kim. Vehicle-to-vehicle (V2V) message content plausibility check for platoons through low-power beaconing. *Sensors*, 19(24):5493, 2019.

[13] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS*, pages 147–166, 1996.

[14] P. Mallozzi, M. Sciancalepore, and P. Pelliccione. Formal verification of the on-the-fly vehicle platooning protocol. In *SERENE*, 2016.

[15] I. Mason, V. Nigam, C. L. Talcott, and A. V. D. Brito. A framework for analyzing adaptive autonomous aerial vehicles. In *SEFM*, pages 406–422, 2017.

[16] V. Nigam, C. Talcott, and A. A. Urquiza. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In *ESORICS*, 2016.

[17] V. Nigam and C. L. Talcott. Formal security verification of industry 4.0 applications. In *ETFA*, pages 1043–1050, 2019.

[18] C. Pöpper, M. Strasser, and S. Capkun. Anti-jamming broadcast communication using uncoordinated spread spectrum techniques. *IEEE J. Sel. Areas Commun.*, 28(5):703–715, 2010.

[19] C. Talcott, V. Nigam, F. Arbab, and T. Kappé. Formal specification and analysis of robust adaptive distributed cyber-physical systems. In M. Bernardo, R. D. Nicola, and J. Hillston, editors, *SFM*. 2016.

[20] C. L. Talcott, F. Arbab, and M. Yadav. Soft agents: Exploring soft constraints to model robust adaptive distributed cyber-physical agent systems. In *Software, Services, and Systems - Essays Dedicated to Martin Wirsing*, pages 273–290, 2015.

[21] S. van de Hoef, K. H. Johansson, and D. V. Dimarogonas. Fuel-efficient en route formation of truck platoons. *IEEE Trans. Intell. Transp. Syst.*, 19(1):102–112, 2018.

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