A model-based approach for analysis of data-alteration attacks in co-operative vehicles*

Cinzia Bernardeschi, Maurizio Palmieri, Marta Sanguinetti, and Alessio Vivani

Department of Information Engineering, University of Pisa, Pisa, Italy
{cinzia.bernardeschi,maurizio.palmieri}@unipi.it
{alessio.vivani}@ing.unipi.it

Abstract. This work presents a modular approach for modelling and analysing the effects of data-alteration attacks in co-operative vehicles applications. The approach exploits multi-model simulation and FMI-based co-simulation tools. The system model is extended with an attacker FMU which implements a strategy for the injection of the attack. Co-simulation results can be analysed to show the behavioural traces of the system under various attacks. The approach is applied to a platoon of vehicles.

Keywords: cyber-physical systems · co-simulation · cybersecurity.

1 Introduction

Co-operative vehicular systems are complex Cyber-Physical Systems (CPSs) where a co-ordination algorithm interacts with the vehicles, consisting of the controller of the movement of the vehicle and the dynamic model of the vehicle. With increased connectivity, modern vehicles have become more susceptible to cyber-security attacks [5] [1]. A case is reported in the work [14], in which it is shown how to gain control of a Jeep multimedia Electronic Control Unit (ECU) remotely over the Wi-Fi connection and successively, to access inner vehicle communication system (the CAN bus), listening to the traffic on the CAN bus, and also sending commands over the bus to safety critical units, e.g., the braking ECU. A review of industrial cyber-physical systems from a cybersecurity perspective is in [8].

To protect vehicles from attacks, possible solutions adopted are Intrusion Detection Systems (IDSs), which identify system attacks by watching the CPS for irregularities in network traffic or system behaviors [9] [15] [13] [21].

The modelling of a CPS is complex due to inherent multi-disciplinary issues, in particular CPSs are characterized by the co-existence of discrete behaviors

* This work received funding from the European Union – Next-GenerationEU – National Recovery and Resilience Plan (NRRP) – MISSION 4 COMPONENT 2, INVESTMENT N. 1.1, CALL PRIN 2022 PNRR D.D. 1409 14-09-2022 – FORESEEN: FORmal mEthodS for attack dEtEction in autonomous driviNg systems, CUP N.I53D23006130001.

(the controller), and continuos behavious (the physical laws), which may exhibit different aspects, such as mechanical or electromagnetic ones. Therefore CPSs can benefit from the availability of different modeling languages and tools, each tailored to different components or aspects. For this reason, in CPSs co-simulation technique is widely used [6]. Co-simulation enables the global simulation of a coupled system via the composition of various simulators. System's components can be modelled by different formalisms and exported as simulation units. An orchestrator then co-ordinates the execution of the units. A generally recognized standard for co-simulation is the *Funtional Mock-Up Interface (FMI)* [4]. Many modelling and simulation tools export the models as FMU, that can be run under the supervision of an FMI-based orchestration engine.

In this work, a method is proposed to analyse the beahaviour of co-operative vehicles systems under attacks, assuming vehicle to vehicle communications. The method is based on model-based attack injection and co-simulation technique. We assume that an attacker has gained access to the communication network; and we consider alteration of data sent through the communication network, which are input to the co-ordination algorithm. The effect of such an attack is that the output computed by the co-ordination algorithm (executed locally at the vehicle) is based on corrupted inputs and wrong commands can be sent to the physical system. The INTO-CPS [18] co-simulation framework is used, which offers an integrated "tool chain" for comprehensive model-based design of CPSs. The approach is applied to a platoon of vehicles [20].

The paper is organised as follows: Section 2 briefly presents co-simulation technique, and platooning application. Section 3 introduces the Attack FMU, which is in charge of implementing attack injection in the CPS. Section 4 presents the application of the proposed approach to a simple platoon of vehicles. Section 5 reports the conclusions.

2 Background

2.1 Co-simulation

An extensive survey on co-simulation has been published by Gomes et al. in [7], providing definitions of the fundamental concepts and a taxonomy of the literature based on the discrete events and continuous time computational models.

The standard most widely used for co-simulation of dynamic systems is the Functional Mockup Interface (FMI) [16]. It has as key components, elements called Functional Mockup Units (FMUs), that represent a part of the system to be modeled. Each one of them is responsible for simulating the element that it is modeling and is independent from the other ones. The models are all connected, via an Interface, to a co-simulation Orchestrator which is in charge of the communications between FMUs. A co-simulation schema with three simulation units is shown in Fig. 1.

When a co-simulation starts, the Orchestrator (Master) will initialize the models of the FMUs (Slaves) by calling the functions **fmi2Instantiate** and

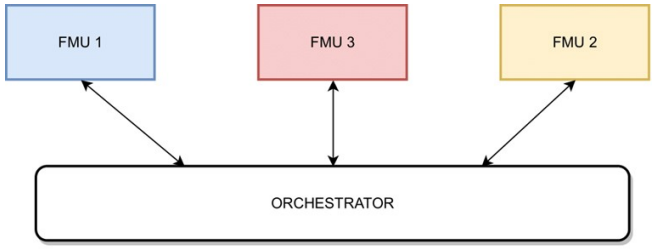

Fig. 1. Co-simulation architecture

fmi2SetupExperiment. Then it will call the **fmi2DoStep** which will increase the simulation time by a time step, that can be set at simulation design time, and then will pass data between FMUs via *set* and *get* methods called **fmi2Set<type>** and **fmi2Get<type>** (*type* ranges over data types, *set* is used to output data at the interface and *get* is used to read data at the interface). The **fmi2DoStep** method will call another function that will simulate the behavior of the model, in the language used to develop such model. At the end of the run, it will free the allocated resources by calling **fmi2Terminate** and **fmi2FreeInstance**. An example of simulation FMI-compliant is provided in Figure 2.

In our work, we will use the INTO-CPS framework [10] for co-simulation of the cyber physical system. INTO-CPS is a collection of tools developed to aid the development of CPSs. INTO-CPS supports FMU obtained from models developed with Modelica, Simulink, Python and other various modeling tools. After the generation of the FMUs, it is possible to create multi-models by defining the components, the interconnections between each other and the values to be assigned to each parameter. The Orchestrator has a Graphical User Interface that allows to create and modify multi-models and run simulations, together with the possibility to visualize the results obtained from a run inside a graph and on .csv files that get generated after the co-simulation. We use a fixed co-simulation step. The simulator reads input data available at the Orchestrator FMI interface, executes for a time equal to the co-simulation step and produces output data that are made available at the Orchestrator interface.

Co-simulation has been applied extensively in different application fields. For example, co-simulation has been used by some of the authors in [2] for space coverage tasks of drones; and in [17], for the comparison between a vehicle-to-vehicle communications against a centralised multi-access edge computing paradigm in vehicle platoons.

2.2 Platoon of vehicles

Platooning has been a subject of research for almost four decades as a promising area in automotive engineering, in particular for improving safety and efficiency [12, 20].

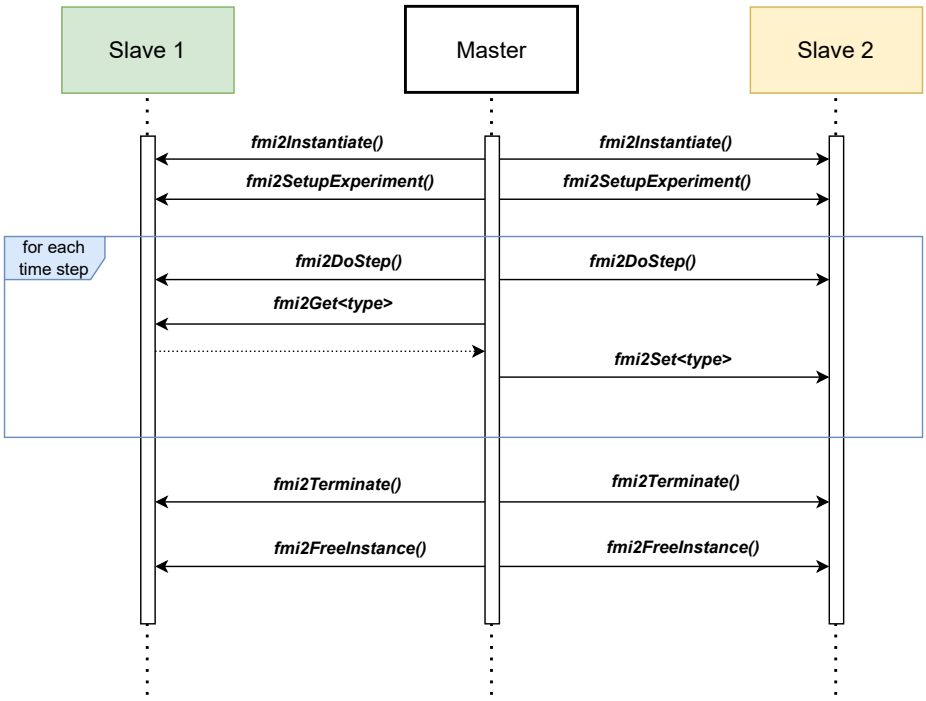

Fig. 2. Example of Simulation FMI-compliant

Simple platooning is the coordinated movement by two or more autonomous vehicles, with each following vehicle maintaining a safe distance from the preceding vehicle. The vehicle at the front of the line is the `leader` of the platoon. As the leader moves, the vehicles behind (`followers`) move and adjust their position.

In our case study, we consider a platoon of 4 cars, see Fig. 3: the leader and three followers. The leader moves with a variable speed, depending on an acceleration function, which is implemented as a function, e.g., a sinusoidal function, with amplitude and frequency taken as input parameters of the co-simulation. Each vehicle has a co-ordination module that is wirelessely connected with the other cars, leader included. In particular,

- every car receives from the leader its speed, position and acceleration, called `v_leader`, `x_leader` and `acc_leader`;
- a car `i+1` that is behind another car `i` receives from the preceding one its speed, position and acceleration, called `v_i`, `x_i` and `acc_i`;
- each car will have to notify to its co-ordination module, locally, at which speed it is moving, and its actual position, obtaining in return the desired acceleration (`acc des`) to have in order to obtain a steady state in which every car has the same speed of the leader, while being at a specific distance one to the other, in our case, the safety distance (`d_safe`).

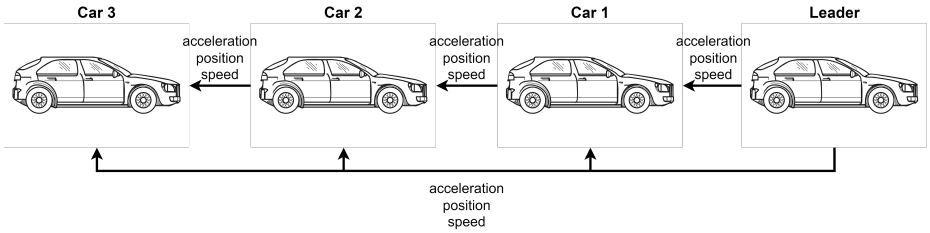

Fig. 3. A simple platoon

The main goal of a platoon co-ordination algorithm is to maintain a specific inter-vehicle space and to guarantee *platoon stability*. One of the most popular control law is Cooperative Adaptive Cruise Control (CACC) [19].

The desired acceleration (`accdes`) of each follower is computed locally to the vehicle as follows:

```
accdes_ego = C1 acc_leader + (1-C1) acc_front - K1 (v_ego - v_lead)
- K2 (x_ego - x_front + L)
```

where `ego` identifies a generic follower, `front` is the car behind it, `K1` and `K2` are gain parameters of the controller, `C1` is the weighting factor between the acceleration of the leader and the acceleration of the preceding vehicle and `L` is the vehicle's length.

In our case study, we assume safety distance `d_safe = 15 meters` and vehicles of the same length: `L= 4 meters`.

A simple block diagram of such system is shown in Fig. 4, where LEADER is the leader and the $i-th$ follower car is named CAR i; the coordination module local to CAR i is named CACC i.

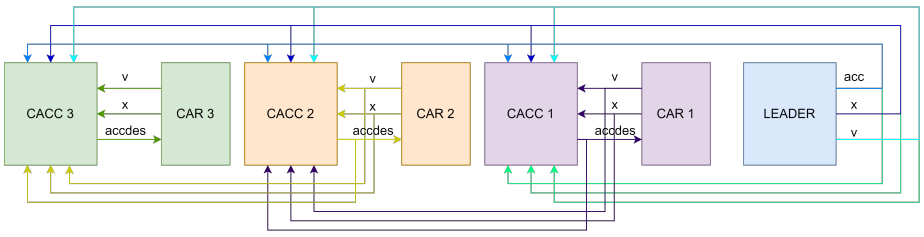

Fig. 4. Block diagram of the system

In absence of attacks, Fig. 5 and Fig. 6 report the speed and the position of the vehicles during co-simulation generated by the INTO-CPS tools. The results show the stability of the platoon.

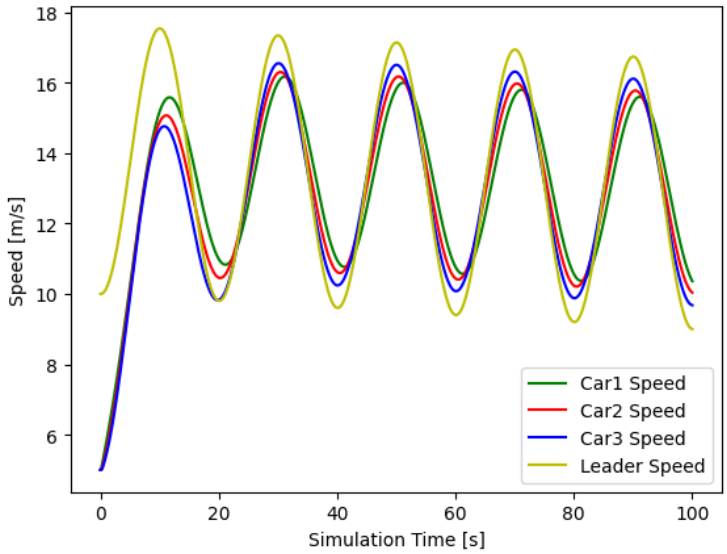

Fig. 5. Graph of the speed of vehicles in the platoon.

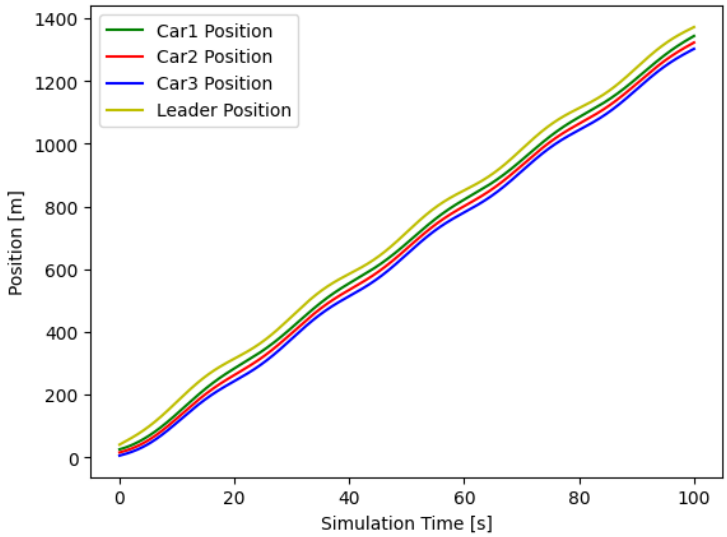

Fig. 6. Graph of the position of vehicles in the platoon.

3 Model-based attack injection: the Attack FMU

In the following we concentrate on the effect of data alteration attacks. These data are used in the co-ordination algorithm to compute commands to be sent to the vehicle. As a result vehicle's acceleration can be wrong, causing a wrong movement of the vehicle, too low or too fast.

The Attack FMU receives as input a signal, and produces as output the same signal. In case of no attack, the value of the ouput is equal to the value of the input; in case of attack, the output is changed accordingly to a function specified by the designer, e.g., increment by a constant.

The FMU models an attack that at a certain time, `attack_time`, chosen by a parameter of the system, the value received in input will be changed by adding a constant `attack_value`, that is yet again a parameter. We assume permanent and periodic attacks. Variable `attack_type` is used to specify a permanent attack (`attack_type = 1`) or periodic attack (`attack_type = 2`). In case of `attack_type` different from 1 and 2, we assume that the attack is not active. Variable `period` is used when triggering periodic attacks. Variable `current_time` is the current simulation time. The system evolves every cosimulation step (`step_size`). A periodic attack starts at `attack_time`; it lasts `period` time; then it is triggered again after `period` time. Setting the period appropriately, an attack that is executed only once can be modeled.

The Attack model is described in Python, and exported as an FMU using UniFMU toolbox [11] which allows exporting models in various languages into an FMU. The code is shown below: in case of triggered attack, the output value is assumed to be equal to the input value summed with a constant (the `attack_value`):

```
    def do_step(self,current_time,step_size,no_step_prior):
        if self.attack_type == 1 and self.attack_time <=
    current_time:
            self.output = self.input + self.attack_value
        elif self.attack_type == 2 and self.attack_time <=
    current_time:
            if current_time >= self.attack_time + self.counter
     * self.period:
                self.output = self.input + self.attack_value
                if current_time >= self.attack_time + (self.
    counter + 1.0) * self.period:
                    self.counter = self.counter + 2.0
            else:
                self.output = self.input
        else:
            self.output = self.input
        return Fmi2Status.ok
```

Listing 1.1. Attack FMU example - Python code

Function `do_step` is invoked by **fmi2DoStep** to simulated the behaviour of the attacker in the co-simulation.

Fig. 7 shows a continuous attack in the case in which the signal is the position of CAR 1 that is received from CAR 2. In this type of attack, we make CAR 2 believe that the first car is in a different position with respect to the actual one. Increasing the position seen by the second car will cause an increase of its speed in order to reach the preceding one.

Fig. 8 shows an example of a periodic attack, assuming a period equal to 1.5s.

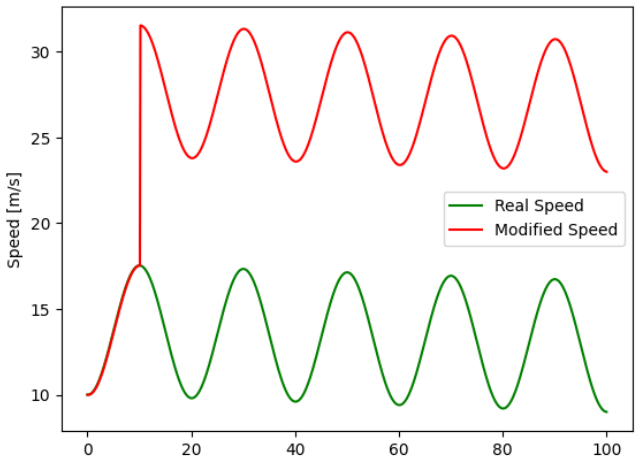

Fig. 7. Example of continuous attack

4 The platoon case study

We decided to perform various attacks in different positions using the Attack FMU, this allowed us to deeply understand how the system was working in all those cases. In particular, we focus on the three cases described below. For each of those cases we kept the following initial values:

```
- LEADER: v0=10 m/s x0=40 m
- CAR 1: v0=5 m/s x0=25 m
- CAR 2: v0=5 m/s x0=15 m
- CAR 3: v0=5 m/s x0=5 m
```

Then we tried to discover the minimum values required to have a crash and we compared the periodic case and the continuous case in terms of time-to-crash using the same values. When deciding the values for the attack we choose them

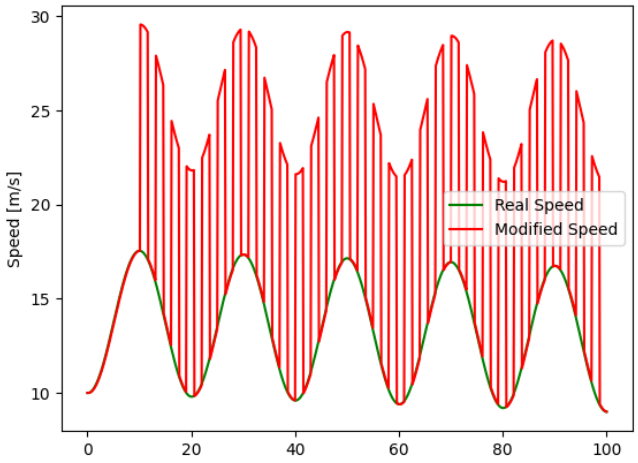

Fig. 8. Example of periodic attack

in order to not exceed the physical limits. For example, we tried to select attack values that changed the speed and the acceleration in a reasonable way, neither increasing nor decreasing them too much.

4.1 Attacks on the speed

The first attack that we chose to analyze was an alteration of the `v_leader` input, from LEADER to CACC 1, as we can see from the block diagram in Fig.9.

In particular, if we use a positive `attack_value`, the first car will increase its speed and will move towards the leader, in order to keep the spacing at 15 meters, while if we use a negative `attack_value` we will make CAR 1 slow down, because it believes that the leader is moving at a lower speed. For each scenario we used both the periodic and the constant attack FMU.

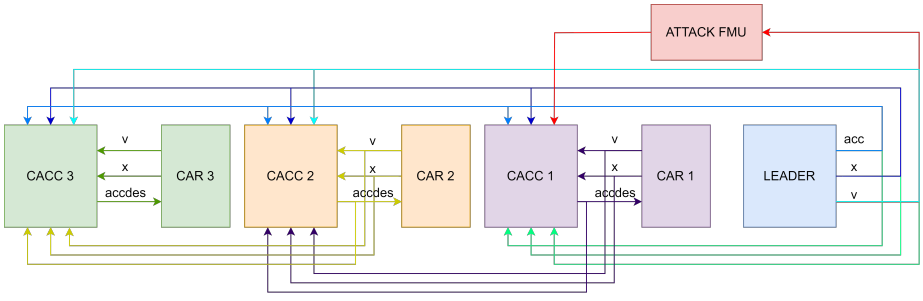

Fig. 9. Block diagram with an attack on the speed

Positive attack value Here we tried different values depending on the FMU used. For both of them we found out that, depending on the `attack_value`, the alteration of the data may lead to a crash between CAR 1 and the Leader. Moreover, we have seen that for the periodic attack the minimum value to use to get an impact is 12 m/s, while for the constant attack we only need 7 m/s. If we use values below that threshold there will be no impact, but the platoon will be damaged, since the first car and the leader will not have a safe spacing. The CAR 2 and CAR 3 will not be directly affected by this attack, they simply will be following the CAR 1, meaning that they will be accelerating, but the algorithm will make them stay at a distance of 15 meters from the preceding car. Here we show an example of impact. The simulation used `attack_value` = 14 and constant attack FMU. Fig. 10 reports the position of vehicles in case of the previous attack. As it can be seen, after some simulated time Car 1 and the Leader will crash. The simulation will still continue as there is not a mechanism for early stopping, for this reason we see that Car 1 will also surpass the Leader during the simulation. In some of the other Figures provided later in the paper we will see similar behaviours as the one shown in Fig. 10, except for Fig. 14 where there is no crash at all.

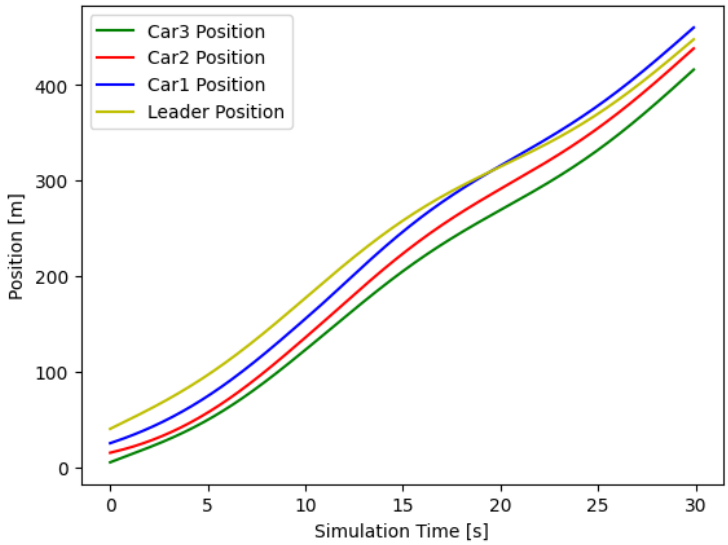

Fig. 10. Graph of the positions in the continuous case

Negative attack value The attack value we used is –8 m/s. For this value we obtained, for both the attack FMUs, that there isn't an impact between the CAR 1 and the CAR 2, that's because the first car doesn't slow down fast enough to make it impossible for the algorithm to avoid the collision. Even though there

won't be a crash between cars, the system will still be damaged, since the first car will always believe that the leader is going slower than it actually is, meaning that it will stay further away from it, making the following cars do the same, with respect to the leader. Of course, when using the periodic attack, we will observe a smaller spacing between the platoon and the leader, but the effects are still pretty significant, as we can see in this image, in which we have a distance of 25 instead of 15 (we have to remember that each car is assumed to be 4 meters long). Fig. 11 reports the position of vehicles in case of the previous attack.

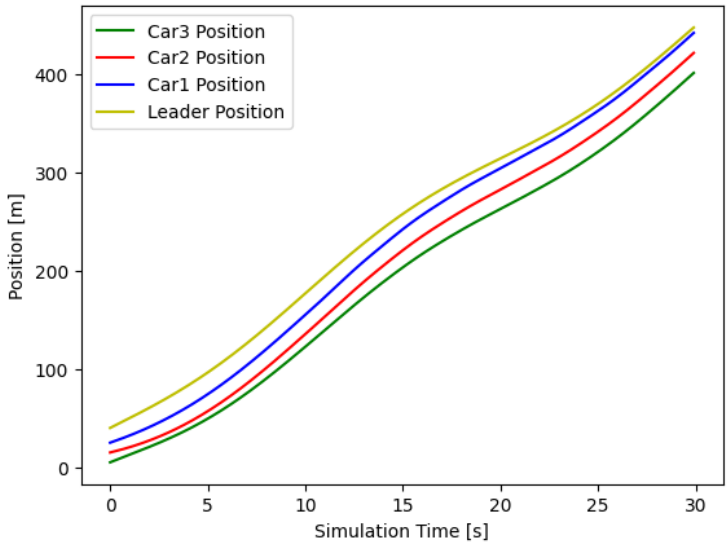

Fig. 11. Graph of the positions in the periodic case

4.2 Attacks on the acceleration

Here we model a data alteration between CACC 1 and CACC 2 considering the transmission of information about the acceleration of the first car. The concept of this attack is simple, by increasing the actual acceleration of the first car, the following one will try to catch up and will increase its speed to do so and that may lead to a crash, while giving negative values as inputs to the attack, the second car will think that the preceding one is going to stop, causing issue on the platoon since the last two cars will slow down to avoid impact. Fig. 12 show the block diagram in case of the previous attack.

Positive attack value In this case we tried a couple of values. In particular we determined which were the minimum values for attack_value that can lead to a crash. For the periodic attack it's 2.5 ms^2, with an impact after 25 seconds, while

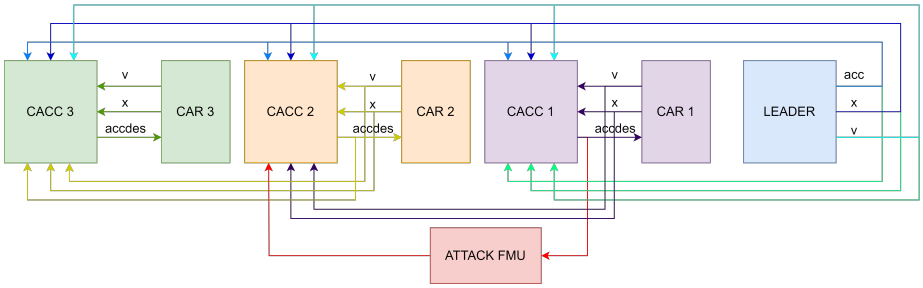

Fig. 12. Block diagram in case of an attack on the acceleration

for the constant attack it's 1.5 ms^2. With lower values we only break the platoon because CAR 1 and CAR 2 will be too close one to the other. Then we tried the attack value at 2.5 ms^2 for the constant attack FMU as well, obtaining a crash in 5 seconds. Fig. 13 reports the position of vehicles in case of the previous attack.

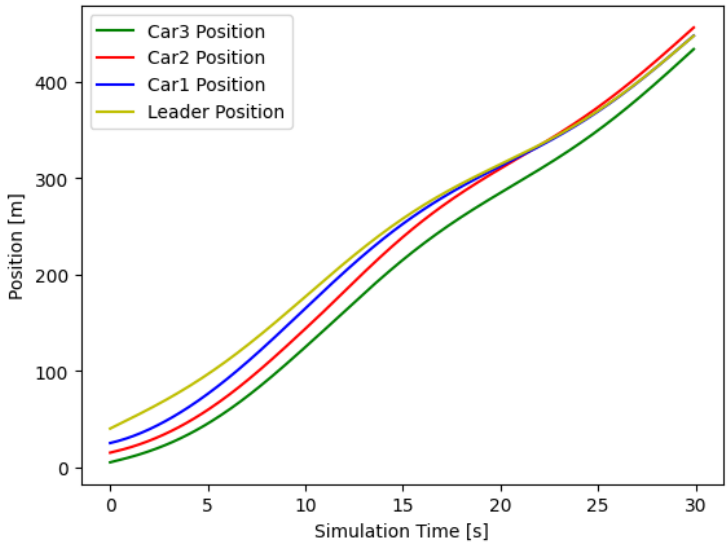

Fig. 13. Graph of the positions in the continuous case, positive attack value

Negative attack value Here we chose to check the lowest acceptable acceleration value, that is –9 ms^2. We obtained that for the periodic attack we heavily damage the platoon, making cars 2 and 3 keep a very long distance from CAR 1 and the LEADER. The constant attack led instead the system to a crash, in fact the second car braked too fast for the algorithm to handle it, so the last

car hadn't been able to stop in time to avoid the impact. We can also easily observe that for smaller values for the attack we won't have any crash, but we will still damage the overall system by separating some cars from the others. Fig. 14 reports the position of vehicles in case of the previous attack.

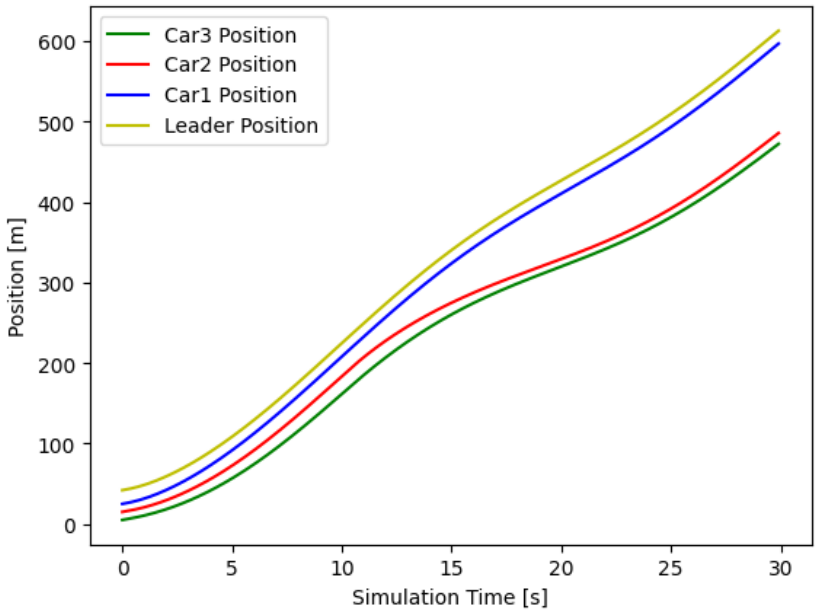

Fig. 14. Graph of the positions in the continuous case, negative attack value

4.3 Attacks on the position

In this type of attack, we want to make the CAR 2 believe that the first car is in a different position with respect to the actual one. Increasing the position seen by the second car will cause an increase of its speed in order to reach the preceding one, in the other case it will slow down to avoid an impact. Fig. 15 shows the block diagram in case of the previous attack.

Positive attack value Again, we firstly looked for the minimum value that led the system to a catastrophic event. We determined that for the periodic attack it is 35 m, while for the constant one it is 17 m. It's important to note that the attack_value must be greater than 15 m if we want to cause an accident, in fact in this way we are sure that the car will accelerate since it will try to reduce the spacing leading it to be 15 m. Then we tried both attacks with a value of 40 m, obtaining a crash in 16 seconds for the periodic FMU, and in 4s for the constant one.

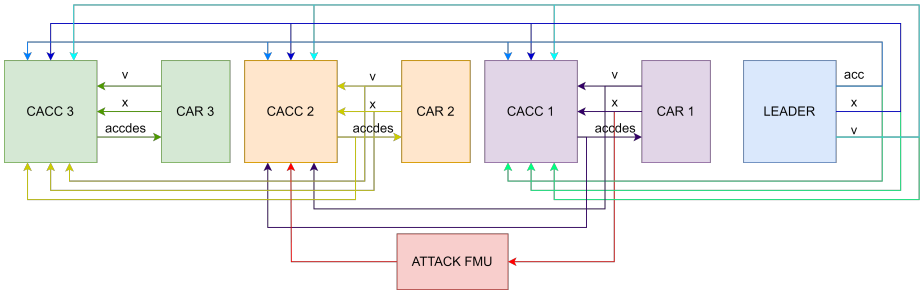

Fig. 15. Block diagram in case of an attack on the position

Negative attack value In this case we have a limitation in the value that we can put as input, in fact, since the system expects to stay in a steady condition in which each car is at 15 meters of spacing, obtaining a value for the position of the preceding car that is less or equal to the one of the ego car might make the attack easily detectable by a simple sanity check of the input data. For this reason, we tried the lowest possible value, that is –14m and we obtained in both cases that the platoon gets damaged without leading to a crash.

4.4 Analysis of results

First of all, we noticed that the attacks that impact the most on the system are those in which we make the controlled car accelerate, since the precedent car has no information on the parameters of the following car. If instead we perform an attack in which we make the controlled car brake, we have a crash only if the safety distance between the controlled car and the following one is violated as a consequence of an immediate slowdown. Anyway, in these kinds of attacks even if we don't have a crash, we always cause some damages since the platoon is divided, without making it possible to achieve the desired constant spacing. Another result that we obtained is that performing different simulations with different initial values, the behavior of the system consequently to the attack didn't depend a lot on the initial position and initial speed of the vehicles. This is probably because the desired accelerations depend on the differences of speed, position and acceleration, which should be independent from the initial values once the system has stabilized.

We have also observed that putting the attack FMU on analogous connections the behavior of the attack is the same and change only the cars involved. For example, an attack between `acc_des` of CACC 1 and `acc_prec` of CACC 2 will be similar to the one performed between `acc_des` of CACC 2 and `acc_prec` of CACC 3.

5 Conclusions

This work presents a modular approach for analysis of attacks in CPSs based on co-simulation technique and the development of the Attack FMU, which is added as a simulation unit in the co-simulation. The approach has been applied to a simple platoon of vehicles as case study. In our previous work [3], an approach to inject faults in a co-simulation is proposed that modifies the behaviour of the attacked component by introducing functions that simulate the effects of attacks. The approach has been applied to a line follower robot case study. The main advantage of Attack FMU is modularity, the Attack FMU is simply added to the co-simualtion multi-model, and flexibility, the condition and effects of attacks can be easily changed in the Attack FMU. Moreover, the Attack FMU can be also applied in case of system's components with intellectual property (IP) protection. As future work, new classes of attacks could be modeled and the use of the design space exploration tool of the INTO-CPS framework could be exploited to generalise the observations on the effects of the attacks conducted on small number of runs.

References

1. Alladi, T., Chamola, V., Zeadally, S.: Industrial control systems: cyberattack trends and countermeasures. Comput. Commun. (2020). https://doi.org/10.1016/j.comcom.2020.03.007
2. Bernardeschi, C., Domenici, A., Fagiolini, A., Palmieri, M.: Co-simulation and formal verification of co-operative drone control with logic-based specifications. Comput. J. **66**(2), 295–317 (2023). https://doi.org/10.1093/comjnl/bxab161, https://doi.org/10.1093/comjnl/bxab161
3. Bernardeschi, C., Domenici, A., Palmieri, M.: Formalization and co-simulation of attacks on cyber-physical systems. Journal of Computer Virology and Hacking Techniques p. 63–77 (2020). https://doi.org/10.1007/s11416-019-00344-9
4. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauß, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.V., Wolf, S.: The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In: Proc. 8th International Modelica Conference. pp. 105–114. Dresden, Germany, March, 2011, https://doi.org/10.3384/ecp11063105
5. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive experimental analyses of automotive attack surfaces. In: 20th USENIX Security Symposium (USENIX Security 11). USENIX Association, San Francisco, CA (Aug 2011), https://www.usenix.org/conference/usenix-security-11/comprehensive-experimental-analyses-automotive-attack-surfaces
6. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A survey. ACM Comput. Surv. **51**(3), 49:1–49:33 (2018). https://doi.org/10.1145/3179993
7. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A survey. ACM Comput. Surv. **51**(3) (may 2018). https://doi.org/10.1145/3179993, https://doi.org/10.1145/3179993

8. Kayan, H., Nunes, M., Rana, O., Burnap, P., Perera, C.: Cybersecurity of industrial cyber-physical systems: a review. ACM Comput. Surv. (2022). https://doi.org/10.1145/3510410
9. Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J.: Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity **2:20**, 1–22 (2019). https://doi.org/10.1186/s42400-019-0038-7
10. Larsen, P.G., Macedo, H.D., Neghina, M., König, C., Basagiannis, S., Woodcock, J., Gomes, C., Fitzgerald, J.: The INtegrated TOolchain for Cyber-PhysicalSystems (INTO-CPS): a Guide. Tech. rep., INTO-CPS Association (2018), http://into-cps.org/fileadmin/into-cps.org/Filer/INTO-CPS-Manifesto.pdf
11. Legaard, C., Tola, D., Schranz, T., Macedo, H., Larsen, P.: A universal mechanism for implementing functional mock-up units. pp. 121–129 (2021). https://doi.org/10.5220/0010577601210129
12. Maiti, S., Winter, S., Kulik, L.: A conceptualization of vehicle platoons and platoon operations. Transportation Research Part C: Emerging Technologies **80**, 1–19 (2017)
13. Metzker, E.: Reliably detecting and defending against attacks: Requirements for automotive intrusion detection systems. VECTOR (2020)
14. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle. Black Hat USA **2015**(S 91), 1–91 (2015)
15. Mitchell, R., Chen, I.R.: A survey of intrusion detection techniques for cyber–physical systems. ACM Comput. Surv. **46**(4) (2014)
16. FMI Standard, https://fmi-standard.org/
17. Palmieri, M., Quadri, C., Fagiolini, A., Bernardeschi, C.: Co-simulated digital twin on the network edge: A vehicle platoon. Computer Communications p. 35–47 (2024). https://doi.org/10.1016/j.comcom.2023.09.019
18. Peter Gorm Larsen, John Fitzgerald, J.W.P.F.J.B.C.K.T.L.M.P.O.G.S.B.e.a.: Integrated tool chain for model-based design of cyber-physical systems: the into-cps project (2016)
19. Rajamani, R., Han-Shue Tan, Boon Kait Law, Wei-Bin Zhang: Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons. IEEE Transactions on Control Systems Technology **8**(4), 695–708 (2000). https://doi.org/10.1109/87.852914
20. Sheikholeslam, S., Desoer, C.A.: Longitudinal control of a platoon of vehicles. In: 1990 American Control Conference. pp. 291–296 (1990). https://doi.org/10.23919/ACC.1990.4790743
21. Son, M.: Cybersecurity IDS - MICROSAR Intrusion Detection System (IDS). VECTOR (2020)