Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

Università di Pisa

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

Cinzia Bernardeschi, Gianluca Dini, Maurizio Palmieri,
Alessio Vivani

Department of Information Engineering
Unversity of Pisa, Pisa, Italy

FORESEEN Project
PRIN PNRR 2022

Summary

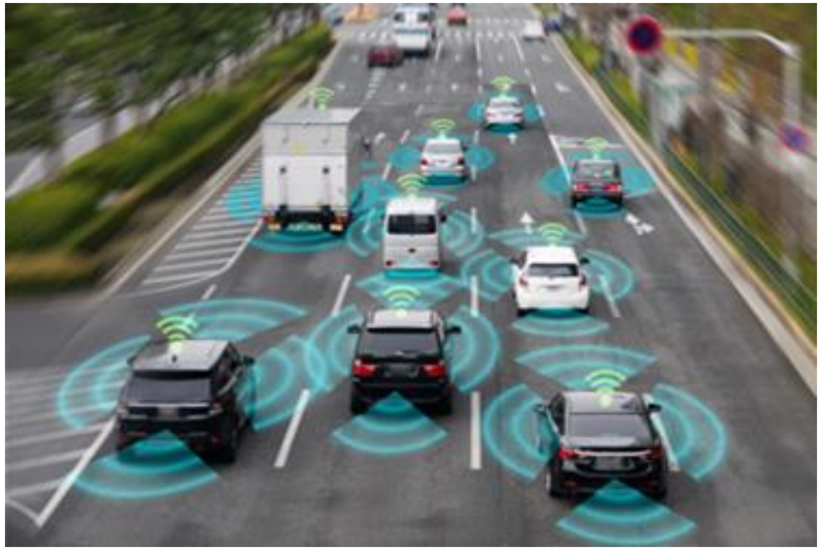
- Introduction
  - Cyber-Physical Systems
  - Cybersecurity in Modern Vehicles
  - Intrusion Detection Systems
- Background
  - Co-Simulation
- Case Study - platoon
- Attack Injection
  - Constant/Periodic attacks
  - Methodology
- Analysis of results
- Conclusions

Introduction - Cyber-Physical Systems

- Cyber-Physical Systems are engineered systems which are increasingly widespread thanks to their versatility.
- A CPS can be seen as a fusion of real-time systems, embedded systems, controls and distributed sensor systems.
- The main peculiarity of those type of systems is the thight coupling between the continuous physical world and the discrete software one.

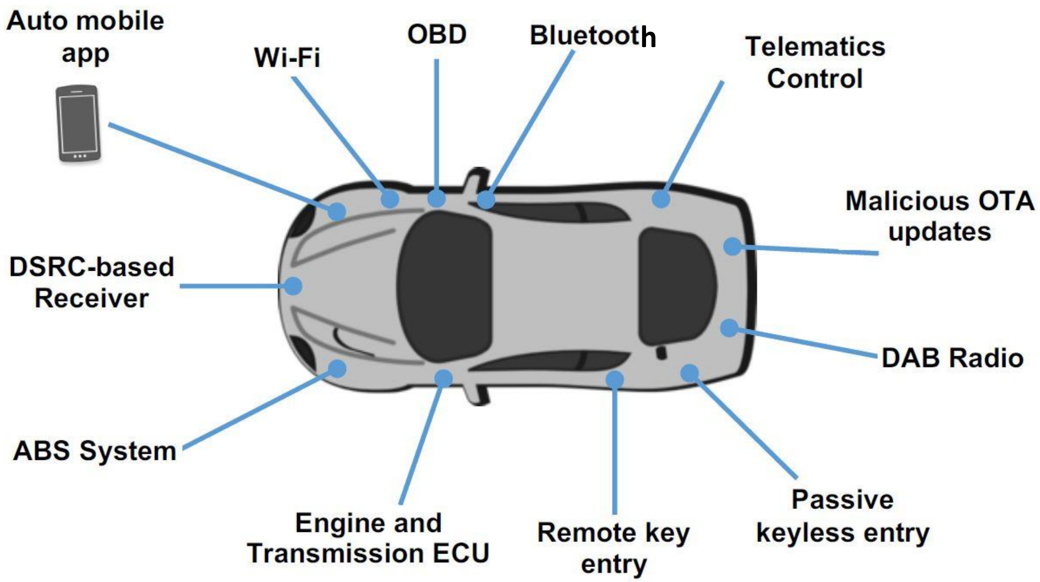
Introduction - Cyber-Physical Systems

- They have many applications which span from automotive to defense and robotic systems.
- A simple example could be a modern vehicle.
  It has many sensors for different purposes, such as carriage keeping and obstacle detection.
- Sensor's output are connected to the controller which can for example start the brakes.



Introduction – Cybersecurity in Modern Vehicles

- Modern cars are highly computerized systems, often connected to a network, thus making them vulnerable to cyber attacks.
- State of the art clearly highlight a large set of threats, for example an attacker could gain access leveraging the OBD port, Bluetooth connection or even the CD player.
- At the 2015 Black Hat conference, it has been shown by Miller and Valasek that exploiting vulnerabilities in the multimedia unit it is possible to gain access and remotely drive a vehicle.
- For those reasons, addressing *Cybersecurity* measures is necessary to ensure *safety*.



C. Miller and C. Valasek, “Remote exploitation of an unaltered passenger vehicle” Black Hat USA, vol. 2015, no. S 91, pp. 1–91, 2015.

Introduction – Intrusion Detection Systems

- To address the issues mentioned in the previous slide, the most commonly used solution is the application of Intrusion Detection Systems (IDS).
- IDS can be made out of machine learning software components or rule based checkers.
- The goal of our work is to exhaustively analyze the effects of various type of data alteration attacks in a **platooning application**, in order to gain an insight on how the system behaves under attack and later how to be able to detect it using rule based checkers.
- In order to do the above, we built a digital twin of the Cyber Physical System using multi-model simulation.

Background – Co-Simulation

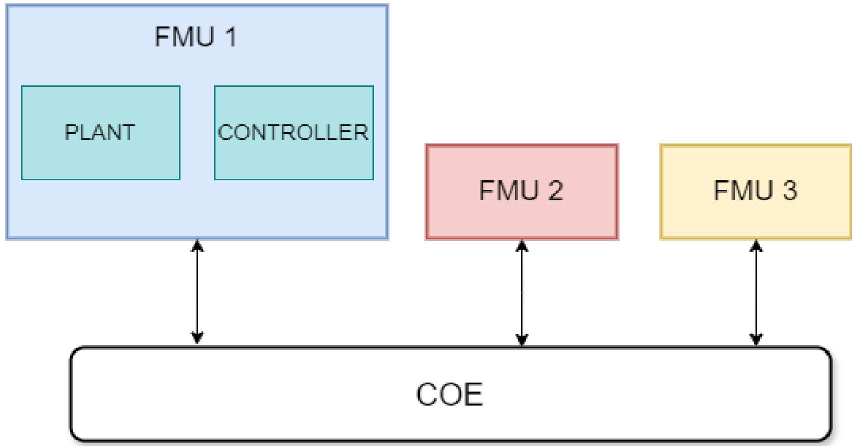
- In model-based design, a commonly used approach to simulate cyber-physical systems is the use of co-simulation.

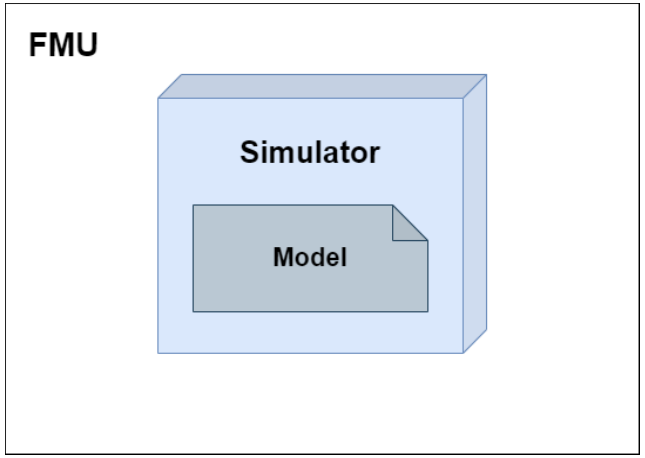
- Co-simulation is a flexible and modular solution that allows to globally simulate a system composed of many smaller components that can be developed in different modelling environments and languages.

- Functional Mockup Interface is a standard for Co-Simulation, where the key components are the Functional Mockup Units (FMUs)

- Each FMU models the behavior of a component of the physical system.

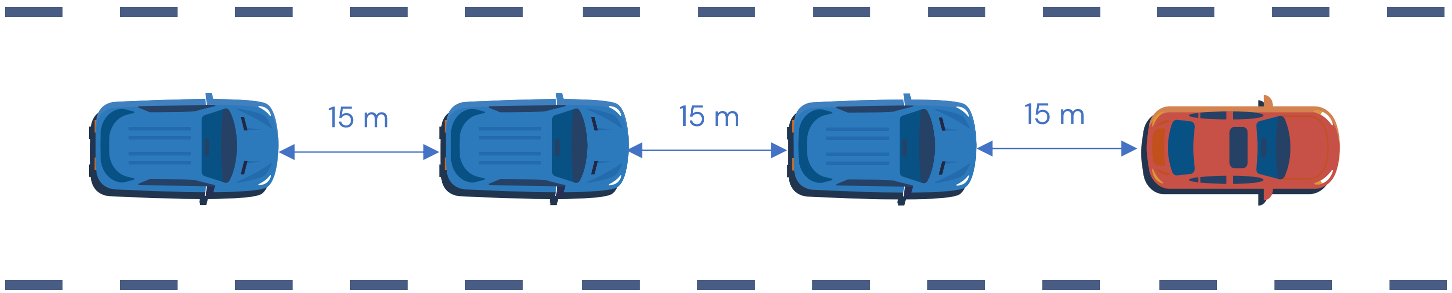
- FMUs are coordinated by a Cosimulation Orchestration Engine.





Case Study - Platoon

The system consists of a platoon of vehicles with one leader car and three following cars, these reach a steady state in which they mantain a constant spacing of 15 meters and the same speed.
A Cooperative Adaptive Cruise Control system on each following vehicle will try to ensure the above condition.

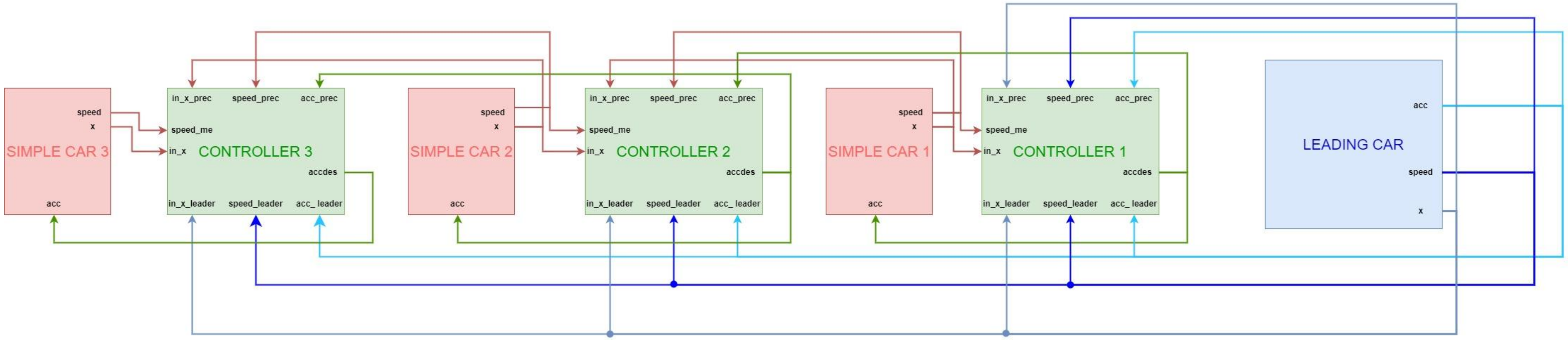


Case Study – Multi-Model

The **Adaptive Cruise Control system** uses the following formula to calculate the desired acceleration for a generic vehicle X, needed to lead the system to steady state.

C_1, K_1, K_2 are parameters of the system, while L is the lenght of the cars, d_{safe} is the safe distance to keep between vehicles in the platoon.

$$a_{car\_x} = C_1 a_{leader} + (1 - C_1)\, a_{front} - K_1(V_{ego} - V_{lead}) - K_2(x_{ego} - x_{front} + L + d_{safe})$$



Attack Injection

In our work we decided to model and study the effects of a specific type of attack, the **Data Alteration attacks**, where for example the Leader provides altered values to a follower vehicle.

Attack injection in a co-simulation environment could be achieved in two different manners

Enhancement of the model by introducing functionalities that models the effects of an attack.

Creation of an Attack FMU, to be used as a man in the middle between elements in the co-simulation schema.

For flexibility we use an Attack FMU.

We also modelled two different type of data alterations:

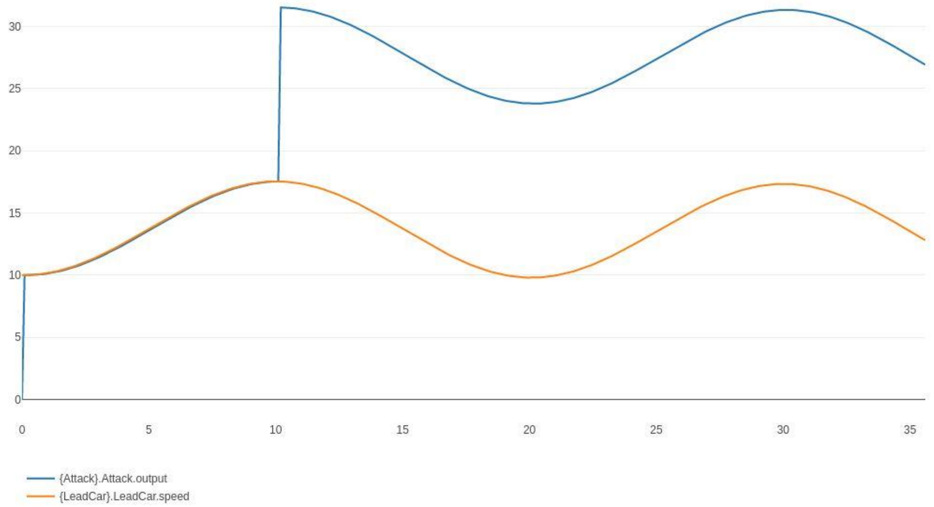
1. Constant data alteration: data is increased (or decreased) by a constant value for the entire duration of the co-simulation.

2. Periodic data alteration: data is modified for a period of time, then the attack goes idle for another period, and so on.

Attack Injection – Constant Attack

In this attack, when we exceed the time **attack_time** the output will always be incremented/decremented of a value equal to **attack_value.**

```
State* tick(State* st) {
    // assert( per_tick(st) );
    if (st->mode == X1 && ( st->time < st->attack_time )) {
        #ifdef DBG
        _dbg_print_condition("st->mode == X1 && ( st->time < st->attack_time )");
        #endif
        leave(X1, st);
        st->output = st->input;
        st->time = st->time + st->step_size;
        enter(X1, st);
    } else if (st->mode == X1 && ( st->time >= st->attack_time )) {
        #ifdef DBG
        _dbg_print_condition("st->mode == X1 && ( st->time >= st->attack_time )");
        #endif
        leave(X1, st);
        st->output = st->attack_value + st->input;
        st->time = st->time + st->step_size;
        enter(X1, st);
    }
    #ifdef DBG
    _dbg_print_state(st);
    #endif
    return st;
}
```

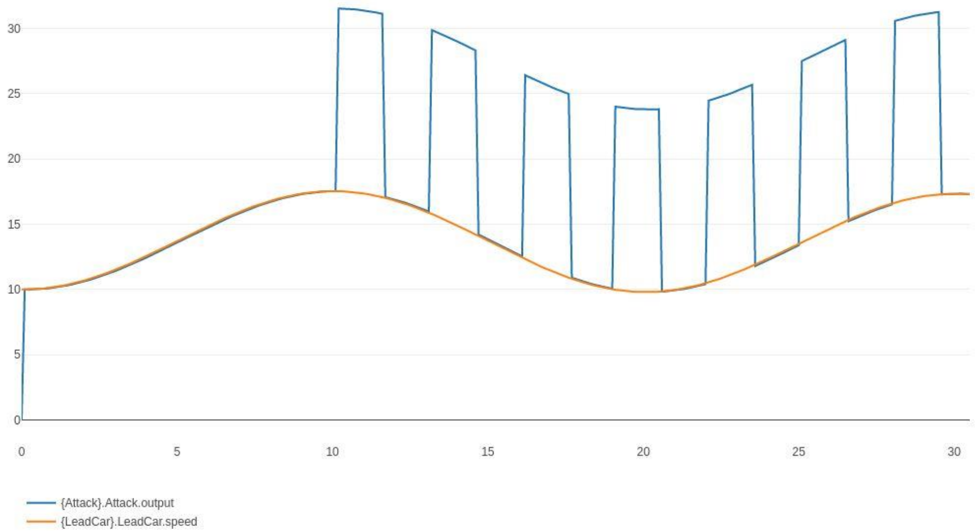


Attack Injection – Periodic Attack

Here instead, after the instant of time chosen as the start of the attack, we change the output only for a constant amount of time, then for the same period we output again the correct value, and so on.

```
float i = 0.0f;
float period = 1.5f;
State* tick(State* st) {
    // assert( per_tick(st) );
    if (st->mode == X1 && ( st->time < st->attack_time + i*period)) {
        #ifdef DBG
        _dbg_print_condition("st->mode == X1 && ( st->time < st->attack_time )");
        #endif
        leave(X1, st);
        st->output = st->input;
        st->time = st->time + st->step_size;
        enter(X1, st);
    } else if (st->mode == X1 && ( st->time >= st->attack_time + i*period)) {
        #ifdef DBG
        _dbg_print_condition("st->mode == X1 && ( st->time >= st->attack_time )");
        #endif
        leave(X1, st);
        st->output = st->input + st->attack_value;
        st->time = st->time + st->step_size;
        if(st->time >= st->attack_time + (i+1.0f)*period){
                i += 2.0f;
        }
        enter(X1, st);

    }
    #ifdef DBG
    _dbg_print_state(st);
    #endif
    return st;
}
```



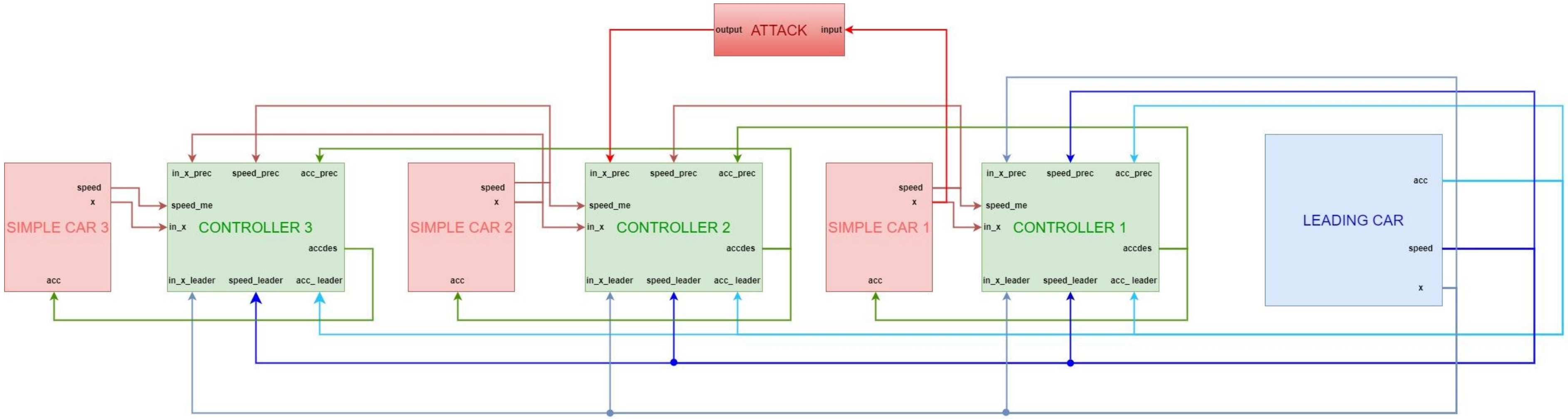
Attack Injection - Methodology

- After the creation of the Attack FMU, it has been used to create many different co-simulation configurations to inject the attack in different locations and between different input/output connections.
- The data type that can be altered are the Position, Speed and Acceleration.
- The Attack FMU can be placed between the Leader and any other vehicle, or between two adiacent vehicles, or even between a car and its CACC Algorithm FMU.
- In these slides we provide a few examples, one per each possible data type.

Attack Injection – Position Attack

Case 1 (**attack_value > 0**): we make the controller 2 believe that car 1 is **further away** than it is, car 2 will then **accelerate** to reach the desired spacing

Case 2 (**attack_value < 0**): we make the controller 2 believes that car 1 is **closer** than it is so that car 2 **slows down** to reach the desired spacing



Attack Injection – Position Attack, Positive

Constant case:

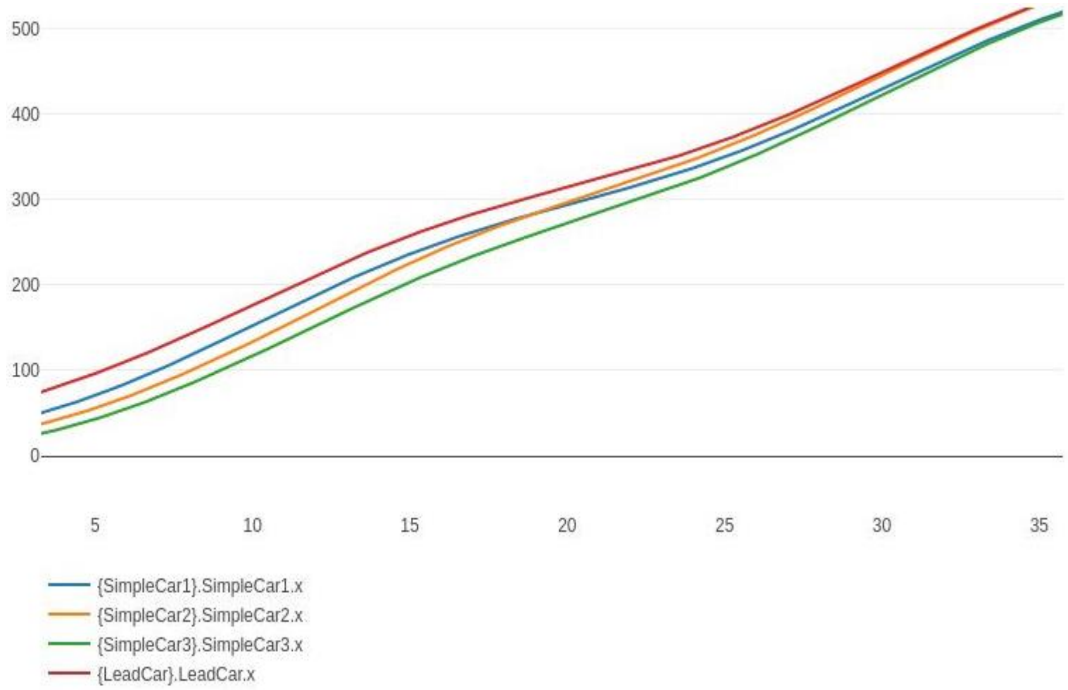
Minimum value to cause crash:
~ 17 m

Attack_value = **40 m**
Time-to-crash = **4 s**

Periodic case:

Minimum value to cause crash:
~ 35 m

Attack_value = **40 m**
Time-to-crash = **16 s**



Attack Injection – Position Attack, Negative

Constant case:

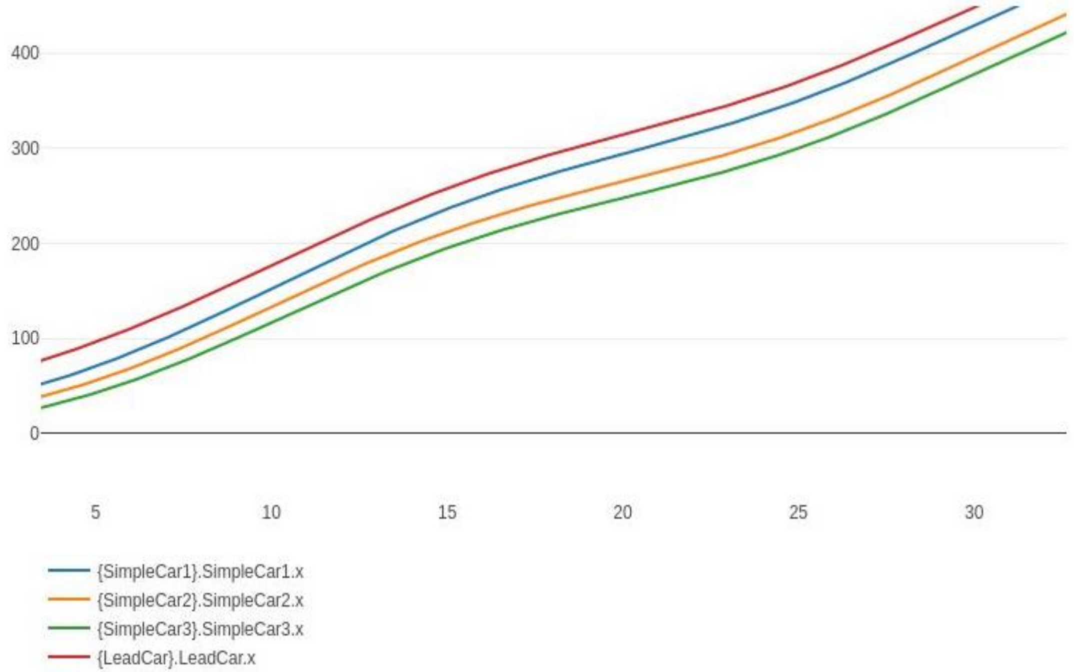
Attack_value = **-14 m**

Effect of the attack: distance between car 1 and car 2 equal to **31 m**

Periodic case:

Attack_value = **-14 m**

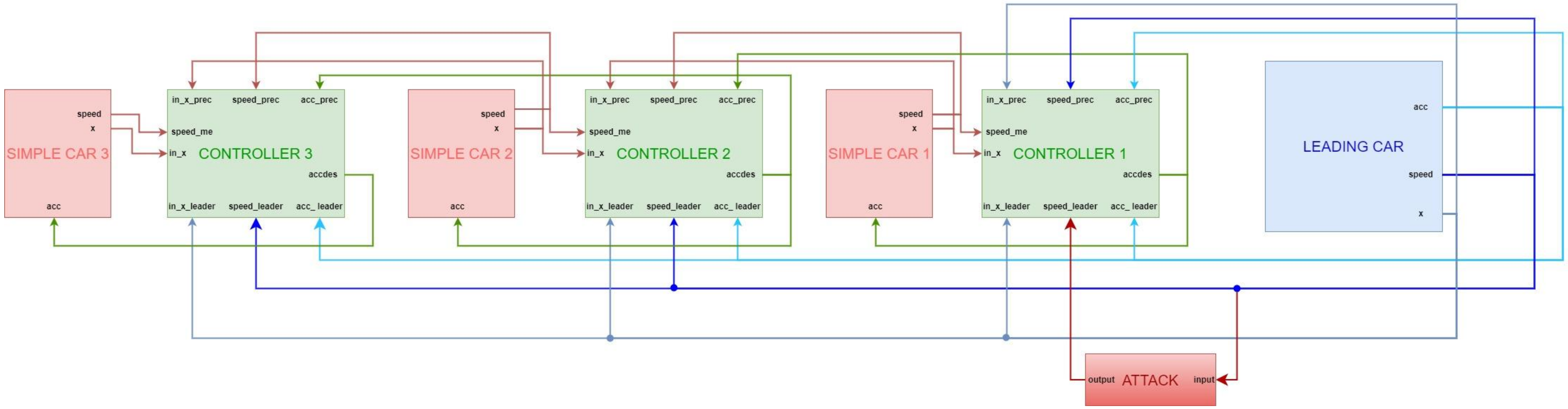
Effect of the attack: distance between car 1 and car 2 equal to **24 m**



Attack Injection – Speed Attack

Case 1 (**attack_value > 0**): we make the controller 1 believe the leader car is going **faster** so that the car 1 **accelerates**

Case 2 (**attack_value < 0**): we make the controller 1 believe the leader car is going **slower** so that the car 1 **slows down**



Attack Injection – Speed Attack, Positive

Constant case:

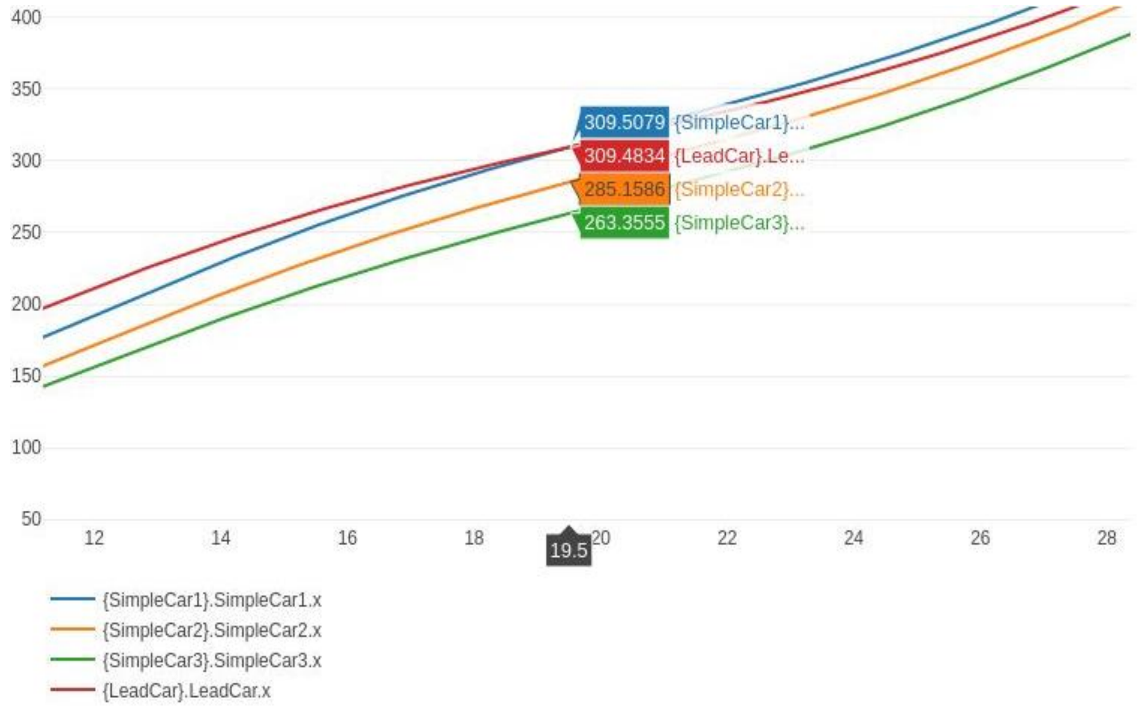
Minimum value to cause crash:
~ 7 m/s

Attack_value = **14 m/s**
Time-to-crash = **9 s**

Periodic case:

Minimum value to cause crash:
~ 12 m/s

Attack_value = **14 m/s**
Time-to-crash = **26 s**



Attack Injection – Speed Attack, Negative

Constant case:

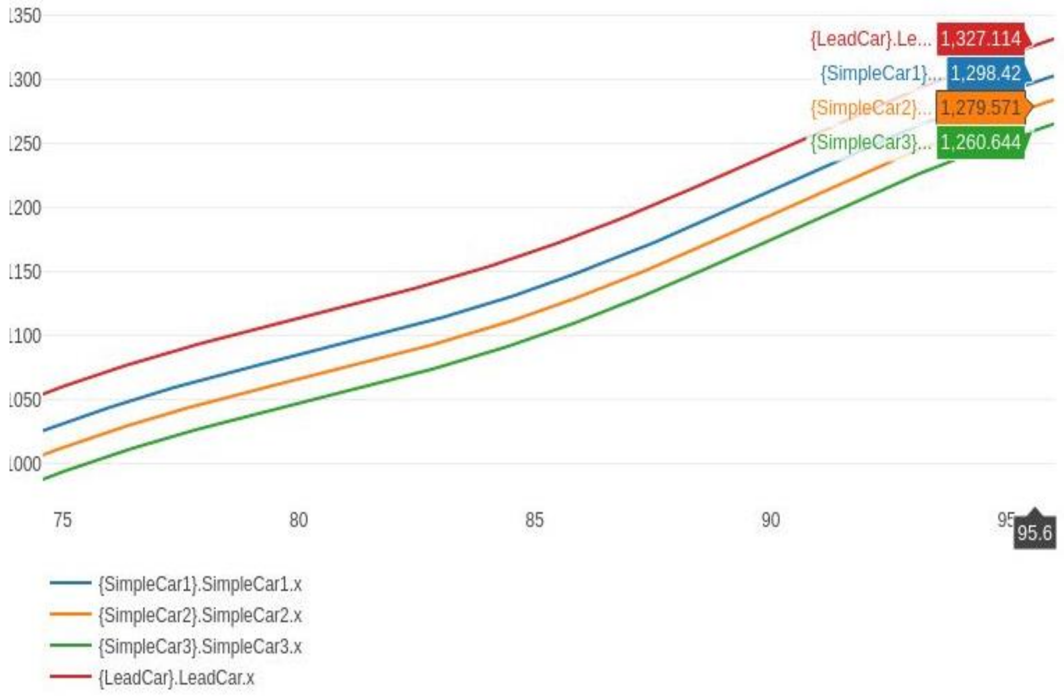
Attack_value = **-8 m/s**

Effect of the attack: distance between leader and car 1 equal to **35 m**

Periodic case:

Attack_value = **-8 m/s**

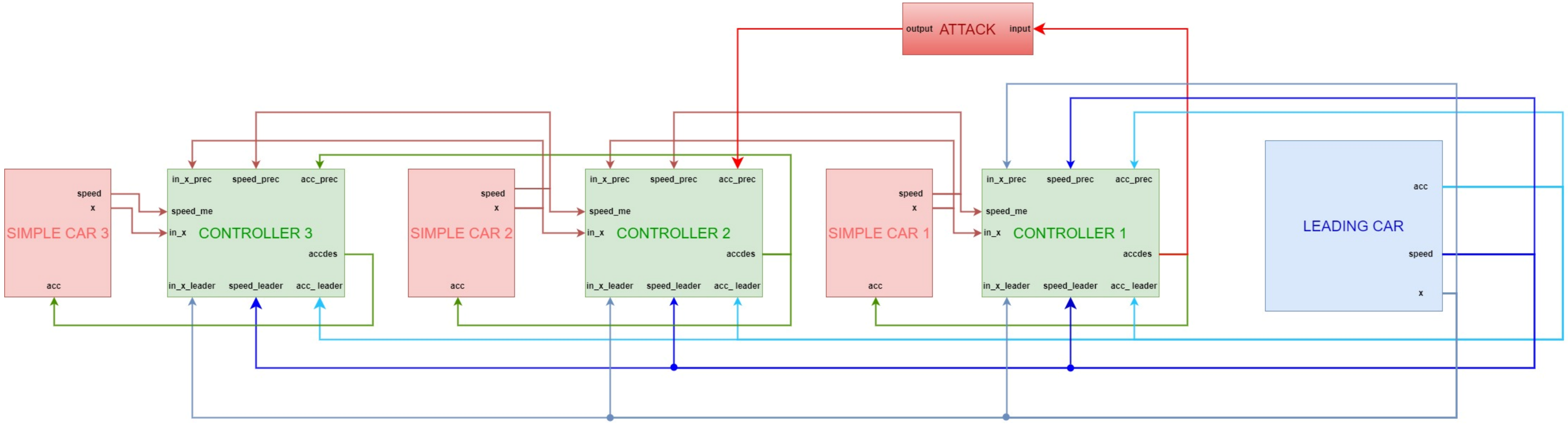
Effect of the attack: distance between leader and car 1 equal to **25 m**



Attack Injection – Acceleration Attack

Case 1 (**attack_value > 0**): we make the controller 2 believe the simple car 1 is **accelerating** so that the simple car 2 **accelerates**

Case 2 (**attack_value < 0**): we make the controller 2 believe the simple car 1 is **decelerating** so that the simple car 2 **slows down**



Attack Injection – Acceleration Attack, Positive

Constant case:

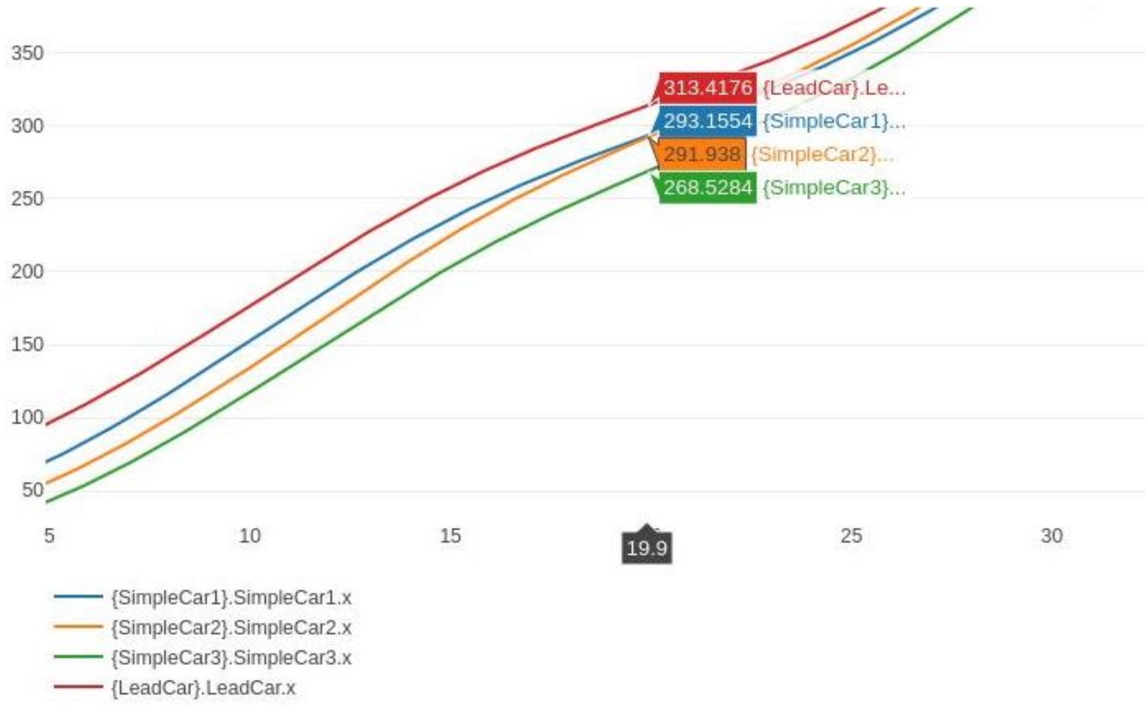
Minimum value to cause crash:
~ 1.2 m/s^2

Attack_value = **2.5** m/s^2
Time-to-crash = **4 s**

Periodic case:

Minimum value to cause crash:
~ 2.5 m/s^2

Attack_value = **2.5** m/s^2
Time-to-crash = **25s**



Attack Injection – Acceleration Attack, Negative

Constant case:

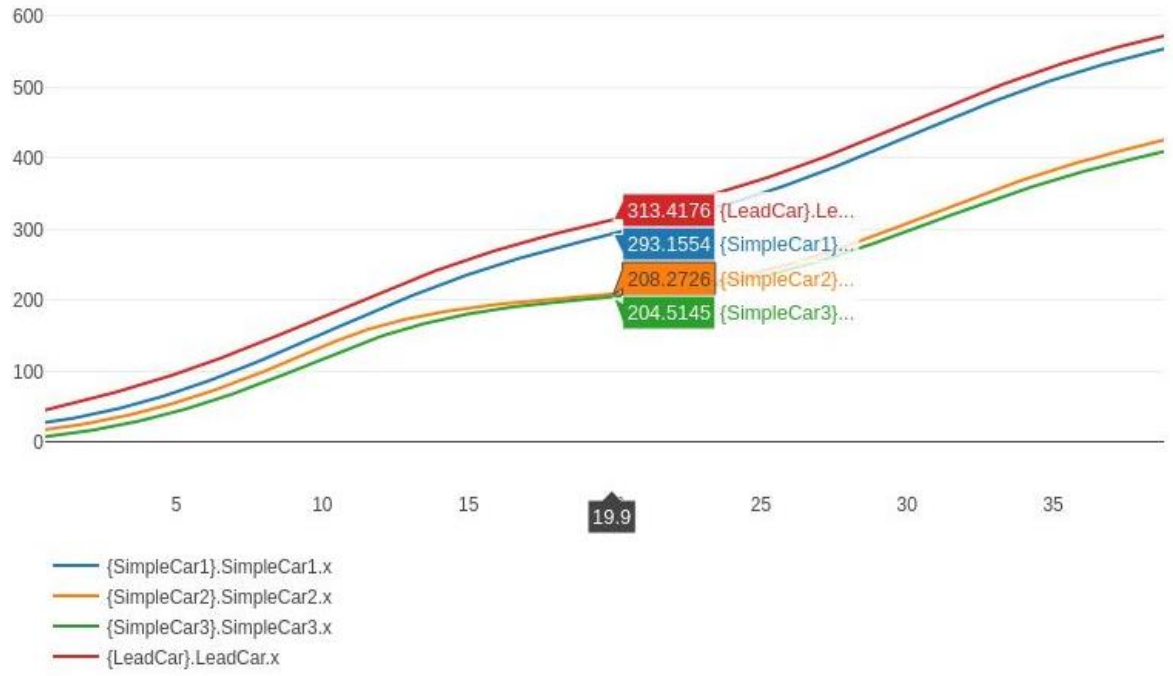
Attack_value = **-9** m/s^2

Time-to-crash = **10 s**
crash between car 2 and car 3

Periodic case:

Attack_value = **-9** m/s^2

Effect of the attack:
no crash but high
spacing between car 1 and car 2
and 3



Analysis of results

- severity of attacks: attacks that make car accelerate are more **dangerous** and **effective** than the ones in which the effect is a slowdown of the car; in the former, the affected car usually **crash** with the preceding one, while in the latter we generally obtain a **split** in the platoon

- simulation of an autonomous system of vehicles in absence/presence of attacks and collection of simulation traces can enable the development of intrusion detection systems
  - generation of formal models for traces of the system and use of formal methods for the identification of trace segments characteristic of attacks. This enables also the development of monitoring services
  - training of ML model for real-time detection of anomalies and classification of potential attacks

Conclusions

- The results obtained after testing various attacks on this ideal case study will be used to extend it to a more realistic case study, the FORESEEN Project (FORmal mEthodS for attack dEtEction in autonomous driviNg systems), PRIN 2022 PNRR, 2024-25.
- There, the vehicles will have a more complex dynamics, the network will not be ideal thus will have delays and possibly packet loss, and the platoon will be larger, with ten or more vehicles, thus allowing to have a better view on the repercussions of an attack to the vehicles in the platoon.
- The design of a framework for improving cybersecutity of cyber-physical echosystems based on a digital-twin and adverary models is under study, the UNTWISTER project (UsiNg digital TWIns to enable SecuriTy in cybER-physical ecosystems); SERICS - “SEcurity and RIghts In the CyberSpace”, 2024- 25