Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 855–859
DOI: 10.15439/2022F12
ISSN 2300-5963 ACSIS, Vol. 30

Sensor Data Protection in Cyber-Physical Systems

1st Anton Hristozov
Polytechnic Institute
Purdue University
West Lafayette, Indiana, USA
ahristoz@purdue.edu

2nd Dr. Eric Matson
Polytechnic Institute
Purdue University
West Lafayette, Indiana, USA
ematson@purdue.edu

3rd Dr. Eric Dietz
Polytechnic Institute
Purdue University
West Lafayette, Indiana, USA
jedietz@purdue.edu

4th Dr. Marcus Rogers
Polytechnic Institute
Purdue University
West Lafayette, Indiana, USA
rogersmk@purdue.edu

***Abstract*—Cyber-Physical Systems (CPS) have a physical part that can interact with sensors and actuators. The data that is read from sensors and the one generated to drive actuators is crucial for the correct operation of this class of devices. Most implementations trust the data being read from sensors and the outputted data to actuators. Real-time validation of the input and output of data for any system is crucial for the safety of its operation. This paper proposes an architecture for handling this issue through smart data guards detached from sensors and controllers and acting solely on the data. This mitigates potential issues of malfunctioning sensors and intentional sensor and controller attacks. The data guards understand the expected data, can detect anomalies and can correct them in real-time. This approach adds more guarantees for fault-tolerant behavior in the presence of attacks and sensor failures.**

***Index Terms*—CPS, robots, software architecture, fault tolerance, resilience, ROS**

I. Introduction

IT IS not always possible to trust sensor data because of reliability issues in sensors, intentional sensor attacks, and issues like EMI interference [1]. Since sensor data are used in control loops, it is better if we could make sure that the data is not compromised and has not deviated from an acceptable range. This problem is very pertinent for control algorithms. In the era of AI solutions, the data streams can determine the success or failure of neural networks or other machine learning algorithms used in the device. Therefore the issue applies to ML solutions.

Using data guards exploits a separation of concerns approach, applicable to off-the-shelf controllers or AI solutions that are complex and hard to understand. Data guards are much more straightforward in their operation and focus on guarding the validity of the data, independently of how complex the controllers or sensors are. Tuning such complex components is difficult and guaranteeing that they will work in all conditions is sometimes impossible. Therefore using data guards before data is fed to the controller is a security and reliability guarantee which enables safer system operation. A further benefit of separating data guards from the rest of the system is that they can be validated separately, as they tend not to contain too much code and complexity. In this respect, they can be similar to the enforcers presented in the literature [2] but are focused on data instead of behavior.

The paper's main contribution is the proposal of dedicated smart data guards that take care of sensor data in real-time. The definition of data contracts is another contribution. The enforcement of such contracts in an existing architecture is a way to enhance systems and enforce security and safety properties. The contribution related to this is that we propose to use separation of concerns through data-centric components that abide by the data contracts we define, depending on the data.

The paper starts with analyzing what data protection means and why we need it. Then the next section discusses data contracts and provides formal presentation examples. The following section discusses how they can improve an existing control architecture. A data guard implementation is also discussed. A reference implementation in PX4 is next. The paper ends with future directions and a conclusion section.

II. Data Protection Analysis

Sensor data requires attention since it is used for control decisions and can affect the safety of a CPS and is therefore critical. Controllers and observers operate by using sensor data, trusting it is correct. Most of the controllers are designed with the assumption of data correctness. In reality, possible attacks and noise in the data, as well as sensor degradation and failure, are facts that cannot be ignored. Both sources of sensor data incorrectness can be dealt with if we take precautions to validate the data in real-time.

A. Sensor Attacks

If we consider a Cyber-Physical System such as an autonomous vehicle or a UAV, there are generally many sensors used by such a system. Some sensors, such as cameras and other object detection sensors, may need to be processed by complex machine learning algorithms to integrate them into the system. Many other sensors, though, are simpler and can generate fewer data per unit of time. The main issue with trusting sensor data is that there is no way to know if the data are valid since there is usually no authentication and encryption of the data sent from the sensors to the controller. There is also no guarantee that the measured physical value is not affected by an attack. A class of attacks can affect physical values without coming into contact with the sensors using EMI or acoustic waves, for example, [3].

An example of a possible attack is an EMI burst that disrupts a sensor ([4]). The other likely attack can happen when the measured data are transferred to the controller, for example,

through CAN bus [5]. Flipping bits or controlling the sensor data bus can be even an easier way to perform an attack. Sensor buses can be in many different forms and can be, for example, i2c or Spi or a dedicated Ethernet and even a wireless connection in some cases [6]. Given that the attack surface is large, we cannot and should not trust sensor data for safety-critical systems. There is a need to add a mechanism to detect and remedy the effects of an attack that manipulates sensor data.

The data coming from sensors could also be encrypted as a security measure, but this is a pretty resource-intensive operation and can severely affect the timing of transporting the data and using it [7]. Controllers are sensitive to delays, and this approach may become impractical for resource-constrained CPS [8]. There are also hardware solutions that enable encryption ,but it is unlikely that all the sensors in an AV can be equipped with such capabilities. Therefore, for this study, we can assume that sensor data travels in the open and can be vulnerable to attacks.

B. Data Guard Components

A sensor typically sends a digital stream of bytes representing a physical parameter, for example, position, velocity, or acceleration in the case of UAVs. In the event of a sensor attack or sensor malfunctioning, we can have data that is not physically realistic at a particular moment, based on the system's state. A data guard can use sensor-specific parameters to guarantee the sensor data [9]. For example, the following parameters can be used in a simple universal approach to sensor validation:

- MAX value - The maximum allowed value for all cases
- MIN value - The minimum allowed value for all cases
- MAX delta - The maximum change in unit time
- MIN delta - The minimum change in unit time
- MAX time for stale data - The maximum time when data can stay the same.
- DEFAULT safe value - When the input value is out of bounds or stale, this value can be fed to the controller. Note that in addition to static default values, we can calculate a default based on historical data when we consider that the system operates under normal conditions.

C. Sensor Data

As one possible example we can have a look at a GPS message in the PX4 autopilot which has the following message abbreviated format in Listing 1:

Listing 1 GPS message details

```
uint64 timestamp
int32 lat
int32 lon
int32 alt
...
```

The individual fields of the GPS message are either integers or real numbers. They can be validated as each message arrives in a software component as part of the system. Such an approach takes care of each message instance but does not help check data deviations between successive message instances. We need an algorithm that contains meta parameters used for all messages, such as delta max, delta min, stale timer values, and default values. All these parameters can be considered as data contract parameters. Each data guard expects the sensor to provide data that is within the data contract parameters [10]. Defining and enforcing such contracts is the main contribution of the paper.

D. Data Guard Utilization

Data guard components can work independently from other components in separate threads. The goal is to provide minimal overhead and be transparent to the rest of the control system. The main goal is to have the guards provide reasonable values when the data stream contains unexpected values. In other words, this is not just filtering specific values but actively reconstructing the sensor data when deemed incorrect. This takes care of spikes or short attacks and can even be used to detect a persistent sensor attack. For example, if the data guard keeps a default value for a certain period, it can then generate a signal indicating that something has gone wrong with that sensor. This is a relatively simple implementation but general enough to be used with many different sensors.

III. Data Contracts

Software contracts have been used in many aspects of software engineering, especially in designing object-oriented systems [11], [12]. In this work, we extend the software contract concepts to be used for the specification of data guards, used to guarantee specific properties in the sensor data and the data sent to actuators. Contracts are based on assumptions and guarantees and can be applied to software interfaces. The general representation is given through equation 1 [11], [12]:

$$C = A, G \tag{1}$$

Where C is the contract, A is the set of assumptions, and G is the set of guarantees. The contract definition can happen during design time since the sensor and actuator data are usually known to the designer. This allows for a thorough analysis of the data and the associated data contract. We start by defining the assumptions A of our data guard contract can be different for different cases and can be expressed as a set according to 2.

$$A = \{A_i\} \tag{2}$$

The data set of each data source by the following set in 3:

$$D = \{D_i\} \tag{3}$$

, where each data member can have a different type.

Some common assumptions for the data in the set D are:

- expected data should have no overflow or underflow for these data types.

- Another assumption can be that new data will be received with a certain minimum frequency.

The guarantees G can include a different set for different data. According to our running example, the guarantees G can be composed through a set of rules 4:

$$G = \{G_i\} \tag{4}$$

An example set of guarantees based on our example follows:

- No data item will exceed its expected maximum value
- No data item will become less than its expected minimum value
- No two successive values will exceed the maximum allowed delta for that data item. The delta is the difference between two consecutive readings.
- A data item will not be stale longer than the maximum allowed number of readings
- Any reading should not deviate from the average of the last N readings by a certain delta value.
- When any of the above rules are violated, a default value will be provided for each data item

Some examples from the set of guarantees G can be represented mathematically in the following way:

$$G_1 : D_i <= D_{imax},$$
$$G_2 : D_i >= D_{imin},$$
$$G_3 : D_{i2} - D_{i1} <= D_{i\delta}$$
$$G_4 : D_i - D_j \neq 0\,, \text{ where } j = i + k \text{ over k time periods}$$
$$G_5 : D_i \in D_u \Rightarrow D_i = D_d$$

where D_u is unacceptable value and D_d is default value

IV. Attack Resistant Control Architecture

Using our defined data contract from the previous section, we can show the proposed control architecture in figure 1. This type of control architecture is typical for a variety of CPS, including UAVs [13]. It is very similar to the classic control loop architecture with the addition of data guard components for each sensor and a data guard for the controller output. Having a data guard for each sensor makes handling each sensor data stream's timing and specific data characteristics easier. This also allows for the sensor fusion to be done separately. The approach assumes that all data guards will be running in parallel so that the streams coming from sensors and the controller can work independently. It is also possible to place data guards after the fusion block; in some situations, this may be a better approach.

Figure 2 shows a possible architecture of a generic data guard. There are two independent timers for calculating the deltas and for the detection of stale data. Stale data means a faulty sensor or a sensor under constant attack. These parameters can be part of the data contract established for each sensor data stream in the system. The data guard component checks for range violations in each message as well as jumps in data readings between messages that are not realistic, given the characteristics of a particular sensor. For example, an accelerometer cannot generate impossible values given the abilities of a UAV or UGV.

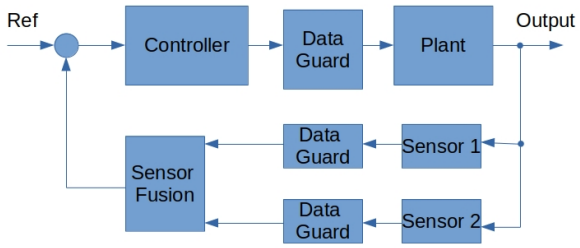

Fig. 1. **Control architecture**

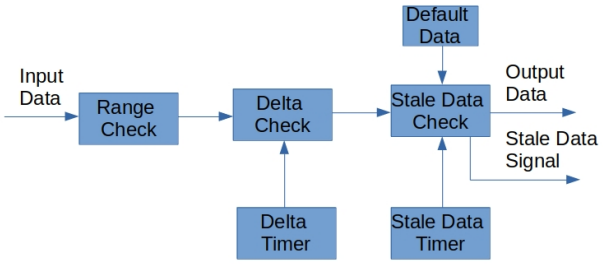

Fig. 2. **Data Guard Component**

A. Implementation of a Data Guard

Data guards can be implemented in a separate module using a variety of programming languages. For existing systems, they will most likely be in the programming language used to develop the system. Our implementation of a data guard follows the generic approach shown in figure 2. An example algorithm for a data guard is written in pseudo-code in Listing 2. This algorithm utilizes two timers and works continuously on incoming data. The implementation can protect against sudden changes that are not physically expected and can detect if a sensor is faulty and does not function anymore. Generating a signal to the system can be used to make a higher-level decision to enter a different fail-safe mode of control. In that respect, the data guard can be the first level in

Listing 2 Pseudo-code example of a data guard

```
read_value()
if data > max or data < min then
data = default_data
endif
if delta_timer_expiration()
check_delta_max()
check_delta_min()
endif
if delta > delta_max
or delta < delta-min then
data = default_data
endif
if stale_time_expired()
check_for_stale_data()
endif
if stale_data() then
send_stale_alarm()
endif
```

the decision-making when implementing fault-tolerant system-wide behavior.

B. PX4 autopilot as a prototyping platform

PX4 can be used within a Software in the Loop (SITL) environment with Jmavsim, or the Gazebo flight simulator [14]. In either case, the sensors of the drone are part of the simulator, and the data from them is sent periodically to PX4, where the data is analyzed and dispatched to other modules. This is done via the Mavlink protocol, which is a standard protocol for messaging in UAVs [15]. Adding a new custom module and intercepting the data stream from one or more sensors is relatively easy, and this is the chosen approach for the experiments. Similarly, defining new messages and writing code for them is very well supported, and we took advantage of it in this work.

V. Reference Implementation of Data Guards

Figure 3 demonstrates the reference implementation with the Gazebo simulator and PX4. Gazebo is a physical simulator used to perform robotic vehicle simulations. One excellent characteristic of Gazebo is that it has plugins, which are essentially software modules that the user can add or modify. This allows for easy additions and modifications of the simulator. There are several types of plugins, among which are sensor plugins. There is one sensor plugin for each sensor as part of the PX4 integration with Gazebo. For the purposes of this implementation, the GPS plugin was chosen so that perturbations for the GPS signal could be introduced. A simple scheme such as randomly generating a spike in the altitude by generating a random number every ten readings or so is a simple way to perturb the GPS data stream. This emulates a GPS sensor attack or a malfunction in the GPS module. Gazebo sends all sensor data to PX4 through the Mavlink protocol, including the periodic GPS message.

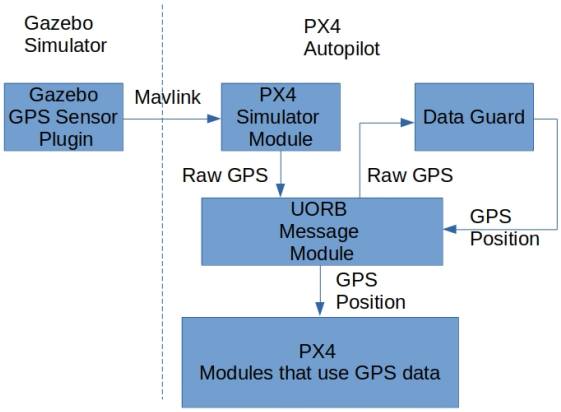

Fig. 3. **Reference architecture**

The reference implementation is a representative example of the approach based on a popular platform for UAVs and the fact that GPS spoofing is such a common GPS attack. The results show that it is fairly straightforward to introduce a data guard component in a publish-subscriber architecture such as PX4. The same can be done for similar architectures, for example ROS and ROS2 [16], [17]. The performance penalty is minimal if we perform just simple checks. This may not be the case if the data is fairly complex. In this case, our data guard may need to use some ML algorithm or fuzzy logic to achieve its goals. Our experiments showed no degradation of the autopilot's performance, and we expect that in many cases, some spare CPU cycles can be utilized to provide the necessary data protection controllers need.

VI. Future Directions

The data guards that we discussed were mainly static as their behavior was specified at design time. However, there may be situations when data fluctuations may require adaptable data guards with more changeable behavior based on ML algorithms. This direction is pretty exciting and also more ambitious. Still, in the era of the ever-increasing use of better hardware and pervasive AI solutions, it is not something that is far from reality. Adaptive behavior has been used in control for a long time, as well as in digital filters. Some of the already established ideas can be applied to complex and variable sensor data with the goal of their online sanitation.

Another possibility is to have an automatic code generator of data guards based on a custom language defining the rules that govern them. This kind of approach can make their implementation even more straightforward and widespread. Automatic code generation can be done from a modeling language or from a data flow language such as Lustre [18]. The objective is to capture the relationships in data processing at a higher level in a fairly representative way and then generate code for the target system. This future direction can be achieved by developing unique tools for the task.

VII. Conclusion

This paper demonstrated a flexible architecture for simulating sensor and controller attacks and a mechanism to react to them by introducing data guards. The data guard can be as complex as needed but still be practical for maintaining the real-time response of the system. The approach applies to any sensor and actuator and any controller or module which consumes sensor data. This can include complicated sensors such as image and Lidar sensors. The approach minimizes or eliminates the possibility of affecting the system's stability and normal operation due to sensor data issues. The method is a complementary run-time strategy to data sanitation used during the training of machine learning systems. It makes sensor data more important as part of the set of concerns for robotic systems. Furthermore, the approach applies to the reliability of sensors and malicious modifications of sensor and controller behavior.

References

[1] H. Pearce, S. Pinisetty, P. S. Roop, M. M. Y. Kuo, and A. Ukil, "Smart i/o modules for mitigating cyber-physical attacks on industrial control systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4659–4669, 2020.

[2] D. de Niz, B. Andersson, and G. Moreno, "Safety enforcement for the verification of autonomous systems," in *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*, M. C. Dudzik and J. C. Ricklin, Eds., vol. 10643, International Society for Optics and Photonics. SPIE, 2018, pp. 1 – 10. [Online]. Available: https://doi.org/10.1117/12.2307575

[3] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Xinyan, "Detecting attacks against robotic vehicles: A control invariant approach," *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[4] Y. Zhang and K. Rasmussen, "Detection of electromagnetic interference attacks on sensor systems," in *2020 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2020, pp. 1–1. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2020.00001

[5] J. Park, R. Ivanov, J. Weimer, M. Pajic, and I. Lee, "Sensor attack detection in the presence of transient faults," in *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, ser. ICCPS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1–10. [Online]. Available: https://doi.org/10.1145/2735960.2735984

[6] M. T. Leccadito, "A hierarchical architectural framework for securing unmanned aerial systems," 2017.

[7] A. Allouch, O. Cheikhrouhou, A. Koubaa, M. Khalgui, and T. Abbes, "Mavsec: Securing the mavlink protocol for ardupilot/px4 unmanned aerial systems," *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 621–628, 2019.

[8] J. Zeng, L. T. Yang, M. Lin, H. Ning, and J. Ma, "A survey: Cyber-physical-social systems and their system-level design methodology," *Future Generation Computer Systems*, vol. 105, pp. 1028–1042, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X1630228X

[9] M. Wu, H. Zeng, C. Wang, and H. Yu, "Invited: Safety guard: Runtime enforcement for safety-critical cyber-physical systems," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[10] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming dr. frankenstein: Contract-based design for cyber-physical systems," *European journal of control*, vol. 18, no. 3, pp. 217–238, 2012.

[11] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, *Contracts for System Design*, 2018.

[12] Y. Liu and C. Cunningham, "Software component specification using design by contract," 03 2002.

[13] M. Sadraey, *Unmanned Aircraft Design: A Review of Fundamentals*, 2017.

[14] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen, and U. P. Schultz, "A survey of open-source uav flight controllers and flight simulators," *Microprocessors and Microsystems*, vol. 61, pp. 11–20, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933118300930

[15] A. Kouba, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro air vehicle link (mavlink) in a nutshell: A survey," *IEEE Access*, vol. 7, pp. 87 658–87 680, 2019.

[16] M. Lauer, M. Amy, J.-C. Fabre, M. Roy, W. Excoffon, and M. Stoicescu, "Engineering adaptive fault-tolerance mechanisms for resilient computing on ros," in *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, 2016, pp. 94–101.

[17] I. Malavolta, G. Lewis, B. Schmerl, P. Lago, and D. Garlan, "How do you architect your robots? state of the practice and guidelines for ros-based systems," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2020, pp. 31–40.

[18] J.-L. Colaço, B. Pagano, and M. Pouzet, "Scade 6: A formal language for embedded critical software development (invited paper)," in *2017 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, 2017, pp. 1–11.