# An automata theoretic framework for detecting schedulability attacks on cyber-physical systems

***Abstract*—This paper proposes an automated framework for detecting schedulability attacks on CyberPhysical systems (CPS) that lead to loss of control performance. Given a set of control components that constitute a CPS, a control objective to be satisfied by the control ensemble, the question of schedulability and synthesis of a scheduler that can ensure the desired control performance has been recently studied in literature. In this paper, we extend this idea further to build an automata theoretic framework for assessment of attacks on schedulability. We have built an end-to-end framework that takes in a set of control components, their variants (after replacement), a control objective to be guaranteed, and performs an automated schedulability assessment. We report some preliminary experiments of our framework on simple benchmarks.**

## I. Introduction

Most critical infrastructures such as power grid, sewage control, power generation plants, industrial automation etc., are cyber-physical systems. Cyber-Physical systems have the following major interacting constituents – a set of control components with physical dynamics governed by laws of physics, modeled with partial differential equations, and another control scheduling component, which co-ordinates control of the physical dynamics and schedules them for execution. For very large and geographically distributed cyber physical systems, the control trajectory is designed to be stable and reliable with very complex distributed as well as centralized control. The control components today routinely involve a non-trivial amount of software, running on embedded control units on-board, to support and run any moderately sophisticated CPS infrastructure.

In recent history, cyber physical systems have been a popular victim of choice for so called cyber attacks, the effects of which have been moderate to as ravaging as blackouts [1] or financial loss. This has prompted NIST [2] to promote cyber security frameworks for critical infrastructure [3] as one of the themes of immediate research attention. Given that no cyber security framework is full proof, significant research thrust is being invested in recent times to develop techniques that allow us to continually monitor the physical dynamics of these systems and look for any anomalies that could be indicative of cyber attack induced problems. Early detection of system dynamics changes would allow us to contain the damage by immediately islanding parts of the system which point to possible origin areas in the infrastructure.

Research in security of CPS is extremely crucial for developing technologies for cyber attack detection, prevention, and countermeasures. Systematic studies on different sources of cyber attacks have revealed myriads of possibilities by which these cyber threats can propagate inside a CPS infrastructure. A person with local or remote access to the equipment of the physical plant, or access to the various interfaces such as programmable logic controllers (PLCs) or other Intelligent Electronic Device (IEDs) that are connected to the physical system for measurement and control, can exploit a vulnerability in these devices to induce an attack on the system. One could also gain access to the control network, and create various kinds of man-in-the-middle attacks by either suppressing measurements or control actuation signals, replaying stale measurements or actuation signals, or even injecting maliciously planned false data. These kinds of attacks would then mislead the controllers, and wrong control actions could lead to disastrous industrial accidents. One could also hack into the controllers or the various other computing elements in the control center such as the process control servers by exploiting vulnerabilities in their design and attack the cyber physical system. In fact, in case of the Stuxnet worm [4], vulnerabilities in the Siemens SCADA system were exploited. Individual areas of cyber security research have received much recent attention in academia, e.g. cryptography; crypto-analysis; network security in the form of firewall and other perimeter security techniques, and intrusion detection, anti-virus software, and static analysis of software to detect vulnerabilities and zero-day attacks; hardware Trojan detection.

**Problem addressed in this paper:** This paper studies the CPS security problem from a different perspective: the focus of study in this paper is *schedulability attacks*, wherein some or parts of a control component are compromised, and the scheduler is unable to schedule the different components in any way to meet the control objectives of the CPS. In an industrial control environment with real time control components, this can lead to disaster since some of the components may malfunction due to lack of input from the scheduler. As an example, slowing down a particular process in industrial manufacturing can cascade a chain of failures in the whole assembly line. In this paper, we propose an idea that will demonstrate that such attacks for real-time SCADA systems can be guarded against by statically analyzing the legitimate control programs, and constructing an omega-regular language based timing signature, which can then be periodically checked on the running components to check for schedulability. The idea presented here is based on the timing signature analysis for omega-regular languages [5], in the context of real-time communication scheduling in CPS.

**Contributions of this work:** In this paper, we address the problem of detecting schedulability attacks. Given a set of

control components, a control objective to be satisfied by the control ensemble, the question of schedulability and synthesis of a scheduler that can ensure the desired control performance has been recently studied in literature [5],[6], [7]. In this paper, we extend the same philosophy to build an automata theoretic framework for assessment of replacement attacks on schedulability. The foundation of our attack analysis framework is based on the notion of infinite automata-based reasoning of control performance and schedulability analysis, as illustrated in the following section. Automata theoretic modeling frameworks have been quite popular in literature for a wide variety of applications. Additionally, the power of finite automata over infinite words (Büchi automata in particular) have been exploited in recent literature in control performance and stability analysis. Our work is another step in the same direction for assessing the effect of schedulability attacks in cyber-physical control.

**Organization:** This paper is organized as follows. Section II presents related work. Section III describes the problem definition for this work along with a motivating example. Section IV presents the solution architecture. Section V presents an overview of the tool we have developed. Section VI concludes this discussion.

## II. Related Work

The idea of automata based control scheduling has been discussed in [8]. They express the communication interface among the control components using a formal language. This work illustrates how the interfaces of discrete-time switched linear systems can be expressed using a Büchi automaton. Authors in [6] describe CPU scheduling in terms of finite machines over infinite words. The infinite words depict the particular time slot for which a particular resource can be allotted to a specific control component. The work in [5] applies automata based scheduling on networks, to formalize the effect of bus scheduling and network stability. Further, [7] describes how the interfacing among the control actions can be modeled by a Büchi automaton to guarantee stable and reliable scheduling. This work presents a methodology for construction of a scheduler automaton where each state represents one particular control schedule while generating the permissible schedule in terms of a $\omega$-regular language. Our work is an application framework based on the contributions above, with specific focus to schedulability attack analysis. While one of the major control objectives discussed in the above contributions has been exponential stability, which we assume to be satisfied for individual control components, we propose to use a fairness schedule in our work, that requires a different modeling strategy, as explained in the following. More importantly, our work here can be considered as an end-to-end framework for automata-based control performance modeling and assessment.

## III. Problem Model

To explain our problem, we consider a control application system consisting of an ensemble of concurrently active

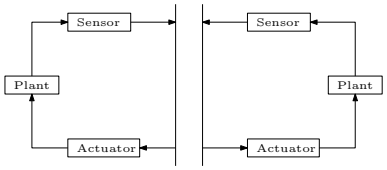


Fig. 1: *Control loops*

control applications / loops. Each control application is partitioned into a set of tasks. The control tasks communicate via a shared bus. The messages on the shared bus are also scheduled using a given arbitration policy. When a control loop executes, it carries out the set of tasks as specified by different states in its control state machine. Among all these states, some states require bus access, for which the control loop presents a request to the scheduler. These requests are considered by the scheduler which decides on a scheduling strategy to grant bus access to the different control loops over time at different time slots.

As discussed in recent literature, we consider each participating control application is modeled as a Büchi automaton [9]. Büchi automata have been a popular mechanism of choice for modeling applications that require finite automata over infinite strings or words. The basic structure of such automata is similar to their finite counterparts, with the exception of the acceptance conditions, as described below.

*Definition 1:* A Büchi automaton is described as a five tuple, $A = (Q, I, \delta, q_0, F)$, where $Q$ is a finite set of states, $I$ is the input alphabet, $\delta : Q \times I \rightarrow Q$ is the transition function, $q_0$ is an initial state ($q_0 \in Q$) and $F$ is a set of final states that model an infinitary acceptance condition. □

*Definition 2:* An infinite word $\lambda \in I^{\omega}$ over an alphabet $I$ takes a Büchi automaton $A = (Q, I, \delta, q_0, F)$ through an infinite sequence of states $q_0, q_1, q_2, ....$, which describes a run of the automaton such that $q_{k+1} \in \delta(q_k, \lambda_k)$ where $k \in \mathbb{N}$ and $\lambda_k$ refers to the $k^{th}$ symbol on the input word $\lambda$. An infinite word is accepted by $A$ if some accepting state $q_f \in F$ appears in the run $q_0, q_1, q_2, ....$ infinitely often. □

Each control loop is designed with a control objective in view, and the overall functioning of the system is dependent on the individual control loops meeting control objectives, along with a strategy for co-operative control of the shared message bus through which the control loops communicate with each other. We consider one such control system functioning correctly if both the above conditions are met. Meeting control objectives has been addressed in several recent articles, and is therefore, not discussed here. In contrast, we focus more on the schedulability aspect, as described below.

**Schedulability violation attacks:** We now characterize the schedulability requirement below.

*Definition 3:* A set of concurrent control loops, each expressed as a Büchi automaton, is schedulable if there exists a strategy to schedule the individual control loops on the shared bus *infinitely often*. The scheduling objective is infinitary acceptance, as is the case with Büchi objectives.□

Intuitively, a scheduler should be able to allocate bus access to each individual control loop infinitely often. More specifically, given any infinite run of the composed system, at least one constituent bus accessing / accepting state from each control loop should repeat infinitely often. We consider a system *safe* if this holds, *unsafe* otherwise. We conclude that a *schedulability attack* has been carried out if a system, initially safe and schedulable, turns out to be non-schedulable.

## IV. Solution Architecture

In this section, we explain our solution architecture for schedulability assessment. As described in the previous section, we are given a set of $n$ control loops, with their bus access states marked as the accepting ones. Recall that our scheduling objective is to be able to grant bus access infinitely often to each control loop. To check if the system is safe and schedulable, we carry out the following steps:

- We compute the intersection of the individual control loops using the construction explained below.
- Once the product is computed, we look for the presence of a cycle that contains at least one state from each control loop. In other words, on any infinite run of the product automaton, each control loop repeats infinitely often and is therefore, granted bus access infinitely often as well.

*Definition 4:* Intersection of Büchi automata [10]:

- Let $B_1 = (\Sigma, Q_1, \Delta_1, {Q_1}^0, F_1)$ and $B_2 = (\Sigma, Q_2, \Delta_2, {Q_2}^0, F_2)$
- We can build an automaton for $L(B_1)$[1]$\cap L(B_2)$ as follows
- $B_1 \cap B_2 = (\Sigma, Q_1 \times Q_2 \times 0,1,2, \Delta, Q_1^0 \times Q_2^0 \times 0, Q_1 \times Q_2 \times 2)$
- We have $(\langle r,q,x\rangle, a, \langle r^{'}, q^{'}, y\rangle) \in \Delta$ iff the following conditions hold:
  - $(r, a, r^{'}) \in \Delta_1$ and $(q, a, q^{'}) \in \Delta_2$
  - The third component depends on the accepting conditions of $B_1$ and $B_2$.
    - If $x = 0$ and $r^{'} \epsilon F_1$ then $y = 1$.
    - If $x = 1$ and $q^{'} \epsilon F_2$ then $y = 2$.
    - If $x = 2$ then $y = 0$.
    - Otherwise, $y = x$
- The third component essentially guarantees that accepting states from both $B_1$ and $B_2$ appear infinitely often.

Intersection Automaton construction: For the sake of simplicity and ease of illustration, we explain the intersection construction in terms of two automata. Let us consider the two Büchi automata P and Q, as shown in Fig.2 and defined by the conventional five tuple:

$P = \{(p_0, p_1, p_2), (a, b), p_0, Q \times \sum \rightarrow Q, p_2\}$
$Q = \{(q_0, q_1), (a, b), q_0, Q \times \sum \rightarrow Q, q_1\}$

From the diagrams given in Fig.2, we see that the automaton $P$ can accept $(ab)^\omega$ and the $Q$ automaton can accept $(b)^\omega$.

[1]L(name of automaton) represents the language accepted the the automaton

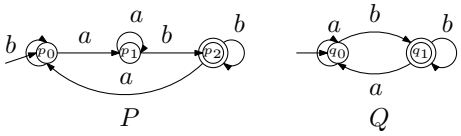


Fig. 2: *Individual automata*

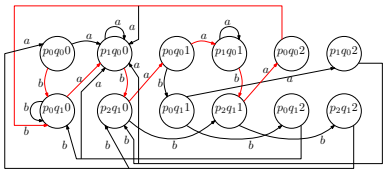

Fig. 3: *Intersection of the input automata*

The resulting intersection automaton will accept $(ab)^\omega$.

In the resulting automaton, a cycle is accepted, when it passes through the last copy of the product automaton, that signifies the run of the cycle traverses the accepting states of each automaton infinitely often. The remaining cycles, which do not pass through the last copy, are not accepted. We thus augment the classical product construction for finite automata to model our infinitary acceptance requirement.

### A. Verifying the scheduling objective

Once the intersection automaton is constructed, the task of looking for cycles starting from the initial states is carried out. Cycle detection algorithms are standard in literature [11] and we implement the same for this work as well. Once a cycle is found, we traverse the cycle (every cycle is bound to contain a finite number of states) and check if each control loop is represented inside it. If not, we proceed to the next cycle and carry out the same check. If all cycles are exhausted and we do not encounter any one which meets our scheduling objective, we conclude that the schedulability requirement is not met. This step is thus quite straightforward. Consider the example given in Figure 3. It can be seen that there are multiple cycles in the intersection automata, some of which do not contain one representative accepting state from each control loop (e.g. $< p0, q0, 0 > \rightarrow < p1, q0, 0 > \rightarrow < p1, q0, 0 >$). Thus it is necessary to examine each cycle to check for existence of at least one accepting state from each control loop, which is the main idea behind the notion of schedulability we adopt here.

### B. Schedulability violation attack detection

As the system is monitored over time, newer structures of the control loops emerge and we perform the same computation steps outlined above on the modified structures to check if it remains schedulable even in the presence of the modifications. Our mechanism can thus be used as a continuous monitor to safe-guard against intermittent control attacks.

At this onset, we would like to admit that our framework can point out schedulability attacks only. Once the intersection is computed, we may have 3 different possibilities if schedulability is lost. Assuming that the system was schedulable earlier, we may conclude the system has been attacked if any of the following possibilities are detected. Firstly, we may have an empty intersection automaton now. As a second possibility, the product may still be non-empty but a cycle may not be reachable. Finally, a cycle may be found but it may not contain representative tasks from each control loop. All these are indicators of schedulability loss according to our analysis method and we conclude that an attack has been carried out.

## V. Experimental Results

We have built an end-to-end tool in Python for code replacement attack detection. The tool takes in a set of control loop descriptions, computes their intersection and implements the cycle detection step. The tool has been applied to a number of small examples of synthetic control applications and their variants. Due to the lack of standard open source benchmarks in this domain, we have performed all our experiments on synthetic benchmarks of various representative sizes, varying the number of individual control loops, and the number of constituent states of each. For each row, for a fixed number of automata, we took up the initial descriptions and created random attacks by modifying some of the components, and carried out our analysis. As expected, our tool was able to flag out the schedulability violation in each of the cases.

Table I presents overall performance of our experiment. The table shows whether the tool is able to detect the attack or not. We have depicted that the system was schedulable before the attack , in other way if the system is schedulable our tool is able to detect that. Now if one of the participated components get attcked the system will become nonschedulable. Our system can capture this attack while examining the schedulability of the system. The table also shows the scalability of the tool. With the increment of participated components the scheduling time will also increase.

Column 2 mentions the number of control applications participating in the control application, while the next 2 columns show whether the tool is able to detect schedulability and the time taken to perform that. Columns 5 and 6 shows the performance of the tool for detecting the attack.

TABLE I: Performance of the tool before and after attack

| Case | No. of automata | All safe? (before attack) | Scheduling time in secconds | All are safe? (after attack) | Scheduling time in seconds |
|---|---|---|---|---|---|
| 1 | 2 | YES | 0.00069 | NO | 0.00038 |
| 2 | 4 | YES | 0.00354 | NO | 0.00054 |
| 3 | 6 | YES | 0.00661 | NO | 0.00920 |
| 4 | 8 | YES | 0.02248 | NO | 0.00693 |
| 5 | 10 | YES | 0.23043 | NO | 0.14550 |
| 6 | 12 | YES | 0.69302 | NO | 0.67120 |
| 7 | 14 | YES | 1.78644 | NO | 1.74657 |
| 8 | 16 | YES | 4.08354 | NO | 3.90508 |

## VI. Conclusion and Future Work

In this work, we present an early foundation of a framework for schedulability attack detection for cyber-physical systems. We adopt standard techniques to build an automata-based control analysis foundation. We have built an end-to-end tool that implements the proposed ideas. We are currently examining the applications of other infinitary acceptance conditions (e.g. Rabin, Muller) in this domain, with respect to their ability to detect different attack scenarios. Additionally, we are also exploring new algorithms for intersection automaton construction that might prove helpful as the number of control loops grow. We believe that our initial results will open up more interesting avenues in this research.

## References


[1] "Report on the grid disturbances on 30th july and 31st july 2012," National Institute of Standards and Technology, Tech. Rep., August 2012.

[2] "National institute of standards and technology." [Online]. Available: http://www.nist.gov

[3] "Framework for improving critical infrastructure cybersecurity," National Institute of Standards and Technology, Tech. Rep., February 2014.

[4] "The stuxnet worm." [Online]. Available: spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet

[5] G. Weiss, S. Fischmeister, M. Anand, and R. Alur, "Specification and analysis of network resource requirements of control systems," in *Proc. of HSSC*, 2009, pp. 381–395.

[6] R. Alur and G. Weiss, "Regular specifications of resource requirements for embedded control software," in *Proc. of RTAS*, 2008, pp. 159–168.

[7] S. K. Ghosh, A. Mondal, S. Dutta, A. Hazra, and S. Dey, "Synthesis of scheduler automata guaranteeing stability and reliability of embedded control systems," in *Proc. of VDAT*, 2016, pp. 127–132.

[8] G. Weiss and R. Alur, "Automata based interfaces for control and scheduling," in *Proc. of HSCC*, 2007, pp. 601–613.

[9] W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 1990, pp. 133–192.

[10] "Büchi automata and model checking." [Online]. Available: http://flolac.iis.sinica.edu.tw/flolac09/lib/exe/buechi_automata_4on1.pdf

[11] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[12] F. Chevalier, D. D'Souza, R. M. Matteplackel, and P. Prabhakar, "Automata and logics over signals," in *Modern Applications of Automata Theory*, 2012, pp. 555–584. [Online]. Available: https://doi.org/10.1142/9789814271059_0018

[13] S. Ghosh, J. Hiser, and J. W. Davidson, "Replacement attacks against vm-protected applications," in *Proc. of VEE*, 2012, pp. 203–214.

[14] M. S. Prabhu, A. Hazra, and P. Dasgupta, "Reliability guarantees in automata-based scheduling for embedded control software," *Embedded Systems Letters*, vol. 5, no. 2, pp. 17–20, 2013.

[15] M. S. Prabhu, A. Hazra, P. Dasgupta, and P. P. Chakrabarti, "Handling fault detection latencies in automata-based scheduling for embedded control software," in *Proc. of CACSD*, 2013, pp. 1–6.

[16] M. Zamani, S. Dey, S. Mohamed, P. Dasgupta, and M. M. Jr., "Scheduling of controllers' update-rates for residual bandwidth utilization," in *Proc. of FORMATS*, 2016, pp. 85–101.

[17] "Sony pictures hack." [Online]. Available: https://en.wikipedia.org/wiki/Sony_Pictures_hack

[18] J. Ming, Z. Xin, P. Lan, D. Wu, P. Liu, and B. Mao, "Impeding behavior-based malware analysis via replacement attacks to malware specifications," *J. Computer Virology and Hacking Techniques*, vol. 13, no. 3, pp. 193–207, 2017.

[19] S. Ghosh, J. Hiser, and J. W. Davidson, "Replacement attacks against vm-protected applications," in *Proceedings of the 8th International Conference on Virtual Execution Environments, VEE 2012, London, UK, March 3-4, 2012 (co-located with ASPLOS 2012)*, 2012, pp. 203–214.

[20] D. Kirovski and F. A. P. Petitcolas, "Replacement attack on arbitrary watermarking systems," in *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, 2002, pp. 177–189.