An automata theoretic framework for analyzing effects of code replacement attacks on scheduling

***Abstract*—Cyber Physical Systems (CPS) such as a manufacturing plant, a power generator, a power transmission substation etc. are usually controlled by a Supervisory Control and Data Acquisition System. Due to the rise of global terrorism, and cyber criminals, these systems are ripe targets of cyber attacks. Among the various attack surfaces recognized by the community, the possibility of replacement of previously vetted control software, or other software components in the system by malicious variants by insider attackers is an acute possibility. In this paper, we propose an automated framework that addresses the effect of code replacement attacks from the perspective of loss of control performance. 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. In this paper, we extend the same philosophy to build an automata theoretic framework for assessment of code replacement attacks on schedulability. We have built an end-to-end framework that takes in a set of control components, their variants (after code replacement), a control objective to be guaranteed, and performs an automated schedulability assessment. We report some preliminary experiments of our framework on simple benchmarks.**

***Keywords*-CyberPhysical System, Cooperative Control, Code Replacement Attack, Insider threat, ω-regular language, Büchi Automaton, Schedulability**

I. Introduction

Most critical infrastructures such as power grid, water / sewage control, power generation plants, industrial automation etc., are cyber-physical systems (CPS). Cyber-Physical systems have two major interacting components – a component with physical dynamics governed by laws of physics – modeled with partial differential equations, another component – control, computing and networking which controls the physical dynamics. 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 seem to be the origin areas in the infrastructure.

Research investment in cyber security of CPS is extremely crucial for developing novel technology for cyber attacks 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 popular and widely acknowledged means of cyber-attack, *phishing attacks*, originate when someone whose desktop is connected to the control network opens an email containing a payload, which can then take over the control of the business network, and in turn the control network. However, since much of the cyber threat models also assume insider knowledge or sabotage, even an isolated control network is not devoid of cyber-attack possibilities. A popular example is the notorious Sony Pictures hack [4] which leaked a release of confidential data from the film studio Sony Pictures. 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 measurements 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 some vulnerabilities in their design and attack the cyber physical system. In fact, in case of the Stuxnet worm [5], vulnerabilities in the Siemens SCADA system were exploited. While individual areas of cyber security research have received much 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) much of these research focus on guarding IT systems in business and corporate information infrastructure.

The focus of study in this paper is code replacement attacks, wherein some or parts of a control component are compromised by tampering their operational modality by modifying the software running inside. Code replacement [6] by internal employees, phishing attacks or other means of code injection have been known to be a vector for cyber-attacks for a while. Stuxnet analysis showed that such attacks were part of the repertoire in that case. In an industrial control environment with real time control components, code replacement can lead to disaster, for example, slowing down a particular process in the 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 distinguish a replaced component from the original component. The timing signature analysis of omega-regular languages was originally proposed by researchers in [7], in the context of real-time communication scheduling in SCADA systems. In this paper, we plan to take the idea further for addressing the problem of guarding against code replacement attacks on real-time control systems. 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 [7], [8], [9]. In this paper, we extend the same philosophy to build an automata theoretic framework for assessment of code replacement attacks on schedulability.

This paper is organized as follows. Section III describes the background theory for this work, while Section IV formally defines our problem statement. Section V presents the solution architecture. In the following discussion, Sections VI and **??** present some illustrative examples and experiments respectively. Section VII concluded this discussion.

II. Related work

Weiss et al[10], depicts the idea of automata based control scheduling. They expressed the communication interface among the control components by using formal language. In their paper they have illustrated how the interfaces of discrete-time switched linear system can be expressed using finite Büchi automaton. [8] have described CPU scheduling in terms of finite states over infinite words. This infinite words depicts the particular time slot for which a particular resource can be allotted to CPU. Weiss et al[7] has also applied automata based scheduling on network, to formalize the effect of bus scheduling and stability of the network. Further, [9] describes how the interfacing among the control actions can be modeled by Büchi automaton to guarantee stable and reliable scheduling. In this paper they have demonstrated, how to construct the scheduler automata where each state represents one particular control schedule and generated the permissible schedule in terms of ω-regular language.

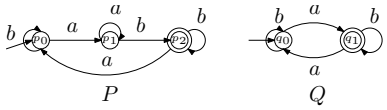

Fig. 1. *Two automata P and Q*

III. Background

In this section, we present two background concepts that are necessary to establish the foundation of this work. We begin with the Büchi automaton.

Büchi Automaton: A Büchi automaton [11] 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, $\delta : Q$ x $I \rightarrow Q$ is the transition function, q_0 is an initial state ($q_0 \in Q$) and F is a set of final states. An infinite word $\delta \in I^{\omega}$ over the input alphabet takes the automaton through an infinite sequence of states $q_0, q_1, q_2, ...$, which describes a run of the automaton such that $q_{k+1} \in \delta(q_k, \delta_k)$ where $k \in \mathbb{N}$ and δ_k refers to the k^{th} symbol on the input word δ. An infinite word is accepted if some accepting state $q_f \in F$ appears in the run $q_0, q_1, q_2, ...$ infinitely often. If $|\delta(q, i)| = 1$ where $q \in Q$ and $i \in I$, then it is a deterministic Büchi automaton, otherwise it is non-deterministic.

Intersection of Büchi Automaton: Intersection on Büchi Automaton is a closure operation. If the given automata are Büchi automaton then after intersection the resulting automaton should also be Büchi automaton. Generally we perform cross product on the given input automata to get the resulting intersection automata. But in case of Büchi Automaton, only cross product is not sufficient to conserve the feature in resulting automata.

Let us consider two automata P and Q depicted in Fig.1 can be defined by the conventional five tuple:

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

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

We need to go through the following steps for intersecting Büchi automaton:

1) First cross product is performed on the given input automata.
2) Consider the resulting product automata, starting from the start state. only reachable states are considered as shown in Fig.2
3) We can see that the states does not contain the tuple $< p_2, q_0 >$, the set of final stattes from both the automata, which signifies that it is an empty automata. But intersection of two Büchi Automaton cannot be an empty automaton. In the following steps we modify the product construction to get a track containing accepting states from both the automata. The resulting intersection automata ensures that both automata visits accepting states infinitely often.
4) After getting the resulting set, where each state is

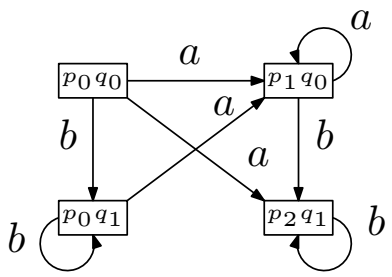

Fig. 2. *cross Product of the input automata*

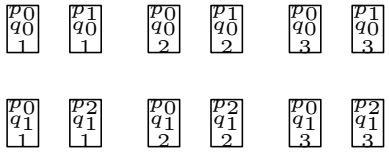

Fig. 3. *product automata associated with flags*

reachable from the start state, the copy of $n+1$ number of such states are made, where n is the number of control loops. In this example we have two automata. So, three copies of the product automata will be made, each of the copy will have a flag with the states as shown in Fig.3. The flags acssociated with the states are waiting to see the acceptance of the automaton.
In the Fig. 3 flag 1 is waiting to see the acceptance state of P, flage 2 is waiting to see the acceptance state of Q, and states of flag 3 signifies that the track has traversed all the accepting states of all the automaton.

5) starting from the first copy, the transitions are made according to the product automaton. A transition goes to the next flag if it gets final state of one control loop. Say there are n number of control loops and $n+1$ number of product automata. Each control loop will correspond to a particular copy. The $n+1$th copy signifies that the track has traversed all the accepting states from each automata. If acceptance condition of that loop is obtained then the transition goes to the next copy, otherwise it remains to the transition of its own copy as shown in the fig.4
6) In this way , when it reaches the last copy, it has traversed the accepting state of each control loop. After reaching to the last state it goes to the first state.
7) In the resulting automaton a cycle is accepted, when it passes through the $(n+1)$th copy of the product automaton, that signifies the run of the cycle traversed the accepting state of each control loop.
8) Other cycles, which do not passes through the $(n+1)$th copy, are not accepted.

IV. Problem Statement and Motivating Example

The aim of our work is to detect code replacement attack. Code replacement attack signifies some changes to the system. We are trying to capture the results of the attack from the perspective of scheduling. Here in this work, we are examining

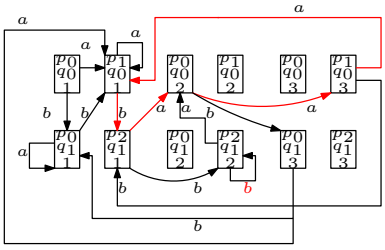

Fig. 4. *cross Product of the input automata*

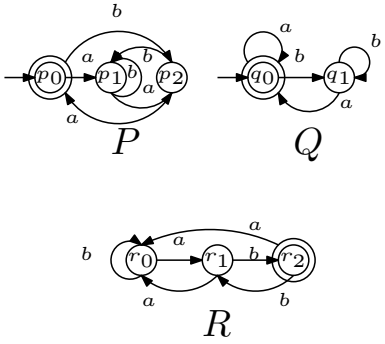

Fig. 5. *Three control automata representing 3 concurrent control loops*

how this changes to the system effects the schedulability of the system.

To explain our problem we are considering a system consisting of many control applications. Each of these control applications is partitioned into a set of tasks. Each of these tasks are mapped onto different processors. These tasks of the controllers communicates via shared buses. The tasks on each processors are scheduling using a given scheduling policy. The messages on the shared buses are also scheduled using a given arbitration policy. When a process is executed by processor it goes through a number of states before termination. Among all these states some states require bus access. Those states are allocated to the bus when their turn comes.

In this paper we are proposing the idea of how to detect code replacement attack. we are considering that each participating control application is a Büchi Automaton. In this work we have performed intersection on the given Büchi automaton as stated in the Background section and want to get such a cycle where the run of the cycle passes through the accepting state of each automaton.

In Fig.5 we can see three automaton $P, QandR$ representing three different control applications. Each has one initial state and acceptance state. The acceptance states are the bus accessing state of the automaton. As we have stated earlier that we are trying to observe the effect of code replacement attack in terms of scheduling sequence, our system will be examinig the schedulability after a particular interval of time. At one particular time we will get the intersection automaton as shown in Fig.6

Let us consider, a code replacement attack take place on this system. The diagram of the automata after the attack is

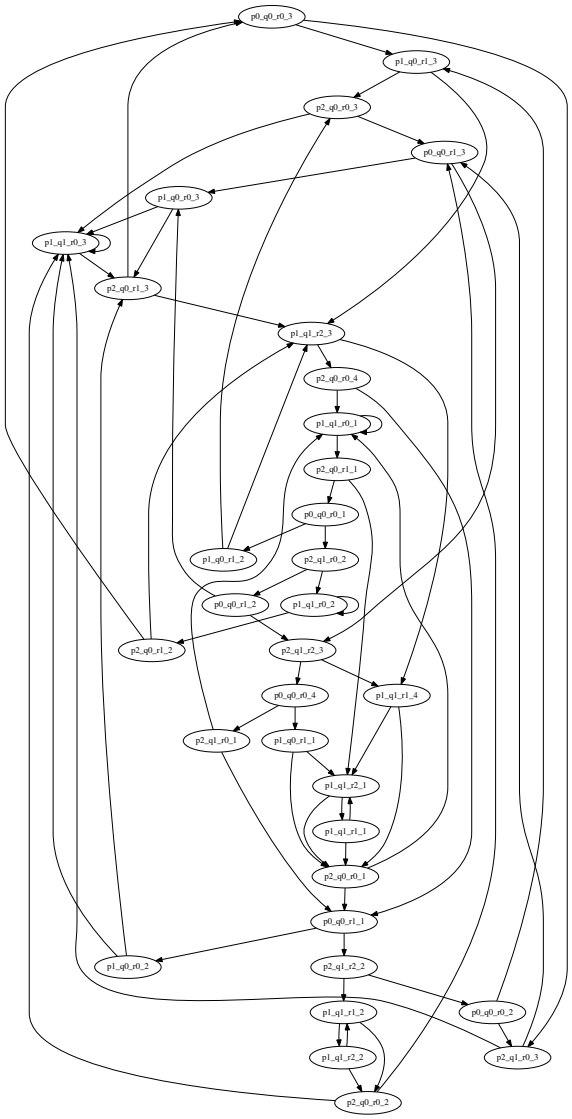

Fig. 6. *state diagram of the resulting intersection automata*

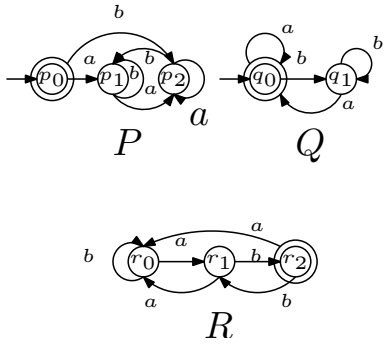

Fig. 7. *Three control automata where control automata P has been replaced*

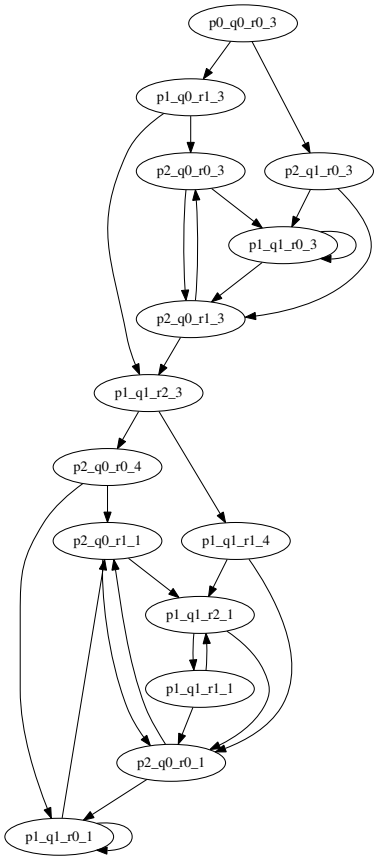

Fig. 8. *The Product Automata after Code Replacement*

depicted in Fig.7 As a result of this attack one control loop got some changes. We can capture the change by examinig the product automaton of the control applications. There can be two consequences, either we will get no cycle in the resulting product automaton or if a cycle can be formed there but the run of the cycle will not contain the accepting sequence from all of the loop. In the Fig.8 we can see that there are cycles but that does not satisfy our expexted condition. So we can say that code replacement attack has taken place.

V. Solution Architecture

Approaches to solve the above stated Problem:

There are n number of control actions. We are considering that each control application can be represented by a Büchi Automaton. In this Büchi Automaton the accepting states are the bus accessing states of that automata. At least one states from each of them are sharing a common bus. In this paper, we are proposing an idea to detect code replacement attack. After a particular interval of time the system will check the schedulability of the system to assure that code replacement attack has not taken place. The main idea behind the algorithm is to check that the resulting automata after the intersection is also a Büchi Automaton. The following algorithm represents the method to solve the problem. Before starting the algorithm we are considering there are n number of control loops.

Algorithm 1: Code Replace attack detection

1: Cross product is calculated on the given n number of automata.
2: Starting from the initial state, only reachable states are considered to get the product automata.
3: From this product automata, applying the procedure as stated in the Background section, we get the resulting intersection automata for the given Büchi Automaton.
4: Next task is to find out at least one cycle from this resulting intersection automata, the run of the cycle should comprise of the states containig the acceting states from each control loop.
5: If such a cycle is not obtained we can conclude that a code replacement attack has taken place.

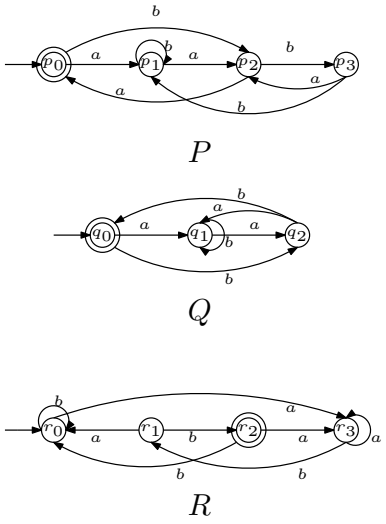

Fig. 9. *Sample automata*

VI. Illustrative examples

In this section, we have described some examples which depicts the difference of intersection automaton before and after the code replacement attack. Let us consider three control automata as shown in Fig.9

The intersection automaton of the given automata will look like Fig.10

Now we will see what happens after each of the automaton get replaced.

After P automaton replacement the automaton representing the control loops will look like Fig. 11

The intersection automaton will look after P automaton replacement Fig.12

After Q automaton replacement the control loops will look like Fig. 13

The intersection automaton will look after Q automaton replacement Fig.14

After R automaton replacement the control loops will look like Fig. 15

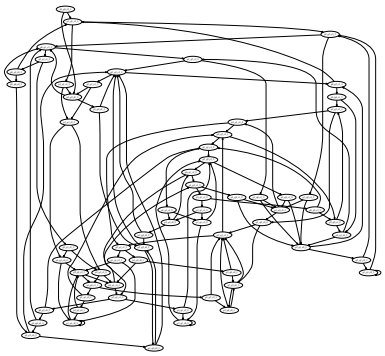

Fig. 10. *Intersection automata of P, Q and R before replacement*

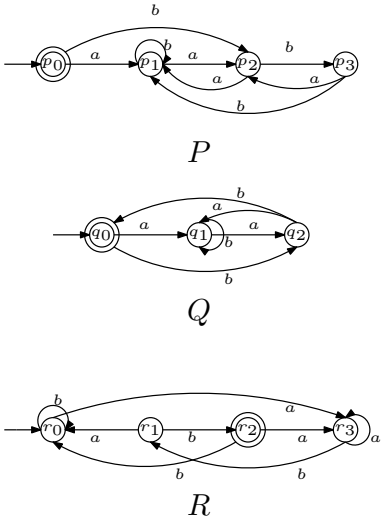

Fig. 11. *sample automata: P automaton replaced*

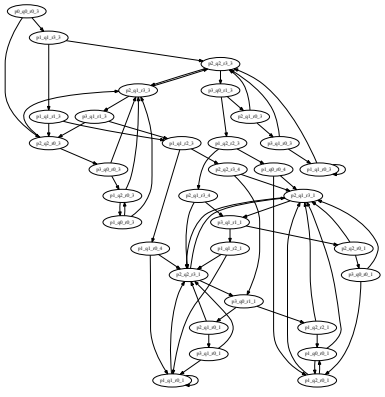

Fig. 12. *Intersection automata of $P, Q and R$ after P got replaced*

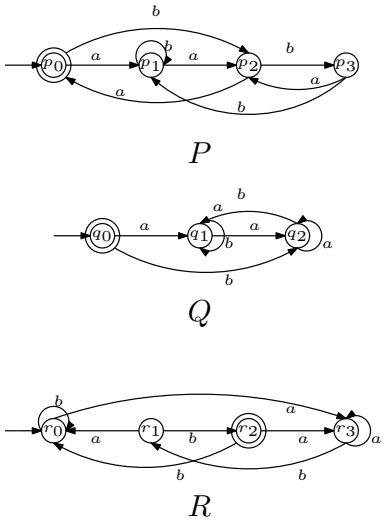

Fig. 13. *sample automata: Q automaton replaced*

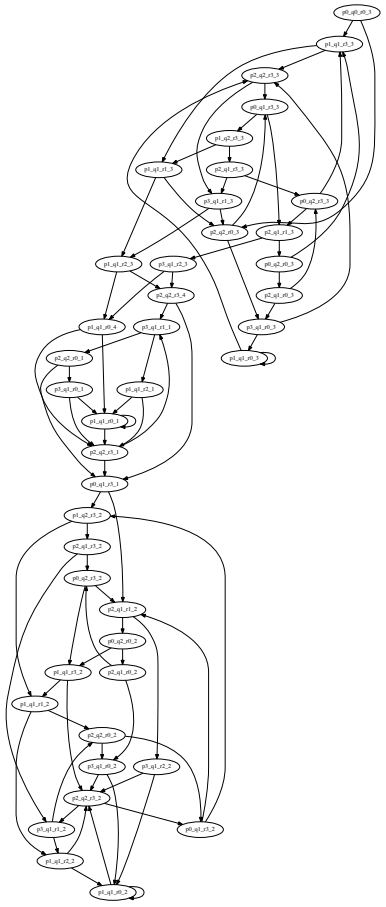

Fig. 14. *Intersection automata of $P, QandR$ after Q got replaced*

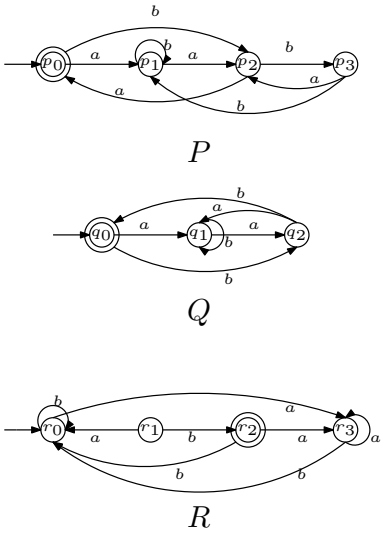

Fig. 15. *sample automata: R automaton replaced*

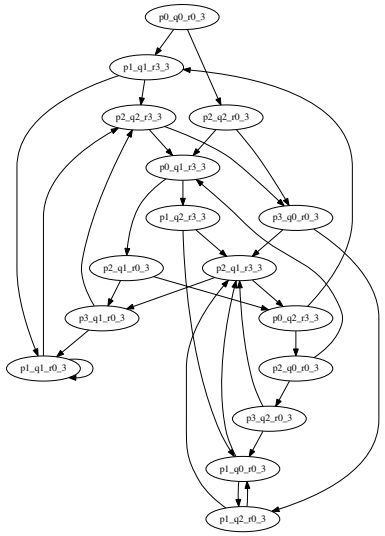

Fig. 16. *Intersection automata of $P, QandR$ after R got replaced*

The intersection automaton will look after R automaton replacement Fig.16

VII. Conclusion and Future Work

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

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

[6] S. Ghosh, J. Hiser, and J. W. Davidson, "Replacement attacks against vm-protected applications," in *Proc. of VEE*, 2012, pp. 203–214.
[7] 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.
[8] R. Alur and G. Weiss, "Regular specifications of resource requirements for embedded control software," in *Proc. of RTAS*, 2008, pp. 159–168.
[9] 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.
[10] G. Weiss and R. Alur, "Automata based interfaces for control and scheduling," in *Proc. of HSCC*, 2007, pp. 601–613.
[11] W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 1990, pp. 133–192.
[12] 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.
[13] 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.
[14] 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.
[15] C.-H. L. Ong, *Automata, Logic and Games*. [Online]. Available: http://www.cs.ox.ac.uk/people/luke.ong/personal/publications/ALG.pdf
[16] Y.-K. Tsay, *Büchi Automata and Model Checkin*. [Online]. Available: http://flolac.iis.sinica.edu.tw/flolac09/lib/exe/buechi_automata_4on1.pdf