CORESAFE: A Formal Approach against code Replacement Attacks on cyber Physical Systems

Cyber Physical Attacks

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. Others CPS example include drive-by-wire automotive, fly-by-wire flight control systems etc. 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. Recent studies have shown that almost 29% of all attacks are insider attacks.
Example:

- Iranian nuclear enrichment plant with Stuxnet in 2009.
- German steel plant in 2014.
- Ukrainian power system in 2015 and 2016.

Aim:
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.

Problem addressed

- Multiple control loops work concurrently and share various resources including the communication bus [1].
- A possible insider attack could be the replacement of a previously vetted control application.
- Replacements can lead to disaster.[2]

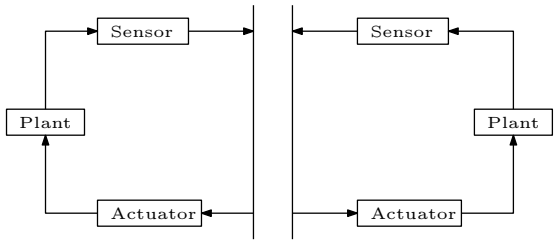

Figure : *Control Loops*

- Focus of this project: replacement attacks. [3]
- Our proposal: such attacks can be guarded against by statically analyzing the legitimate control programs, and constructing an omega-regular language based timing signature.

[1] Through Communication bus control loops interact with sensors and actuators

[2] For example, slowing down a particular process in the industrial manufacturing can cascade a chain of failures in the whole assembly line.

[3] One or more control component is replaced with a modified one.

Contributions of this work

We addressed the problem of detecting component replacement attacks from the schedulability perspective:

- A control objective to be satisfied by the control ensemble to ensure the desired control performance
- Modeling infinite automata-based reasoning of control performance[4]
- Schedulability analysis

[4] The power of finite automata over infinite words (Büchi automatons 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 replacement attacks in cyber-physical control

Schedulability violation attacks:

Definition: 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 defined in the lines of infinitary acceptance, as is the case with Büchi objectives. Our schedulability requirement:

- A scheduler should be able to allocate bus access to each individual control loop infinitely often
- Given any infinite run of the composed system, at least one constituent bus accessing / accepting state from each control loop should repeat infinitely often
- A system is safe if the above two holds, unsafe otherwise

We conclude that a replacement attack has been carried out if a system, initially safe and schedulable, turns out to be non-schedulable.

Problem Model

- A control application system consisting of an ensemble of concurrently active 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 bus arbiter
- These requests are considered by the bus arbiter / scheduler which decides on a bus scheduling strategy to grant bus access to the different control loops over time at different time slots

Definition: 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,* $\delta : Q \times I \longrightarrow Q$ *is the transition function,* q_0 *is an initial state* $(q_0 \epsilon Q)$ *and* F *is a set of final states that model an infinitary acceptance condition.*
Definition: An infinite word $\lambda \epsilon I^{\omega}$ *over an input alphabet* I *takes a Büchi automaton* $A = (Q, I, \delta, q_0, F)$ *through an infinite sequence of states* $q_0, q_1, q_2, \ldots,$ *which describes a run of the automaton such that* $q_{k+1} \epsilon \delta(q_k, \delta_k)$ *where* $k \epsilon N$ *and* δ_k *refers to the* k *the symbol on the input word* δ*. An infinite word is accepted by* A *if some accepting state* $q_f \epsilon F$ *appears in the run* $q_0, q_1, q_2, \ldots$ *infinitely often.*

We consider one such control system functioning correctly if both the above conditions are met[5].

[5] 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.

Solution Architecture

- We are given a set of n control loops
- Bus access states marked as the accepting ones
- 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:

- Computing the product of the individual control loops using Büchi automata Intersection construction
- Looking for the presence of a cycle [6]

[6] After 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.

Intersection automata construction

Given a collection of control loops, each expressed as Büchi automatons, we describe below the methodology for computing their product [7] which is an important step in our work. For the sake of simplicity and ease of illustration, we explain the product construction in terms of two automatons.

Intersection of Büchi automata

- 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 $L(B_1) \cap L(B_2)$ 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 $(\langle r, q, x \rangle, a, \langle r^{'}, q^{'}, y \rangle) \epsilon \Delta$ iff the following conditions hold:
  - $(r, a, r^{'}) \epsilon \Delta_1$ and $(q, a, q^{'}) \epsilon \Delta_2$
  - The third component is affected by 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 is responsible for guaranteeing that accepting states from both B_1 and B_2 appear infinitely often.

[7] Yih-Kuen Tsay,Büchi Automata and Model Checking

Intersection automata construction

Let us consider the two Büchi automatons P and Q, and defined by the conventional five tuple:
$P = \{(p_0, p_1), (a, b), p_0, Q \times \sum \rightarrow Q, p_1\}$
$Q = \{(q_0, q_1), (a, b), q_0, Q \times \sum \rightarrow Q, q_1\}$

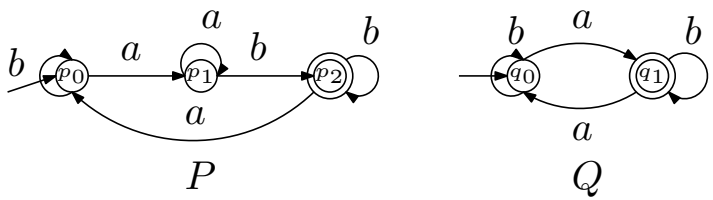

Figure : *Individual automatons*

Considering the reachable states, starting from the start state, the intersection automata of the given input automatons is given below.

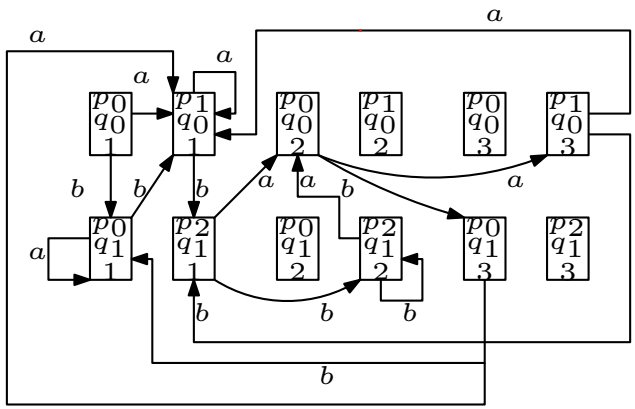

Figure : *Intersection of the input automata*

Checking for Emptiness

Once the intersection automaton is constructed, the task of looking for cycles [8] starting from the initial states is carried out.

Checking Emptiness

- Starting from the initial state, any strongly connected component that contains a cycle over an accepting state is considered as accepting run
- If no connected component has a cycle over accepting run then it is empty

We implement the same for this work as well.

- Once a cycle is found, we traverse one cycle (every cycle is bound to contain a finite number of states)
- check if each control loop is represented inside it
- If not, we proceed to the next cycle
- 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

[8] Clarke,Jr., Edmund M. and Grumberg, Orna and Peled, Doron A.,Model Checking

Checking for Emptiness

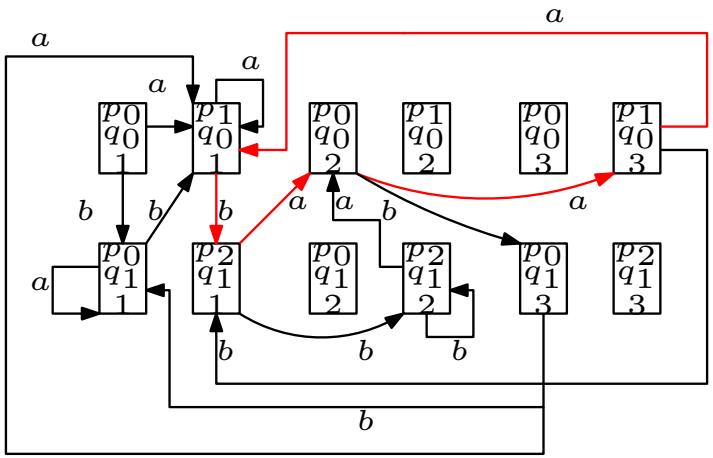

Figure : *Cycle on the Intersection automata*

The diagram depicts that:

- There are multiple cycles in the intersection automata
- Cycle containing one representative accepting state from each control loop (e.g. $\langle p_0, q_0, 1\rangle \rightarrow \langle p_1, q_0, 1\rangle \rightarrow \langle p_0, q_1, 1\rangle \ldots$). will be a valid run
- Necessary to examine each cycle to check for existence of at least one accepting state from each control loop

Schedulability violation attack detection

The system is monitored over time to check the schedulability[9]

- Our framework an be used as a continuous monitor to safe-guard against intermittent control attacks
- Our framework can point out replacement attacks only if schedulability is lost
- Once the intersection is computed,we may have 3 different possibilities [10]:
  - We may have an empty intersection automaton
  - The product may still be non-empty but a cycle may not be reachable
  - A cycle may be found but it may not contain representative tasks from each control loop
- If the system still remains schedulable according to our scheduling objective, our framework is unable to flag the attack, even if a replacement has been carried out

[9] Newer structures of the control loops emerge and we perform the computation steps outlined in the solution architecture on the modified structures to check if it remains schedulable even in the presence of the modifications.

[10] Assuming that the system was schedulable earlier, we may conclude the system has been attacked if any of three possibilities are detected.

Experiment

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.
Table Results presents the details of our experiments

- Column 2 mentions the number of control applications
- Column 3 presents the number of states in the intersection automaton
- Column 4 shows the number of states in the cycle
- Final column presents the total time required by our tool for analyzing the schedulability violation in seconds

[10] 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.

Table : Results

Case	No. of of Statemachines	No. of states in the intersection automata	No. of states in the cycle	Total time taken for scheduling analysis in seconds
1	2	8	6	0.00192
1	2	33	12	0.00106
3	3	67	15	0.00584
4	3	149	22	0.00654
5	5	343	31	0.10965
5	5	354	33	0.02401
7	8	597	97	0.45413
8	12	685	124	5.57695
9	12	981	164	15.21820
10	16	1954	159	33.023548

Shortcommings

- Our methodology is trivial
- Our method can capture the attack iff the system becomes non-schedulable in terms of emptinessn of the intersection automata[11]

[11] Theoritically, Büchi automata is closed under intersection. If all of the participating control loops retains Büchi property then the intrsection automata will also retains Büchi property. If any of them loses Büchi property due to replacement attack, then the intersection automata will not be Büchi and a cycle over the accepted states will not accur. But if after the attack they still retains Büchi property then our method cannot capture.

Planning

We have considered FlexRay architecture[12] to explain our idea. FlexRay demonstrates how a communication protocol can work in automotive domain with high data rates.The architecture is given below:

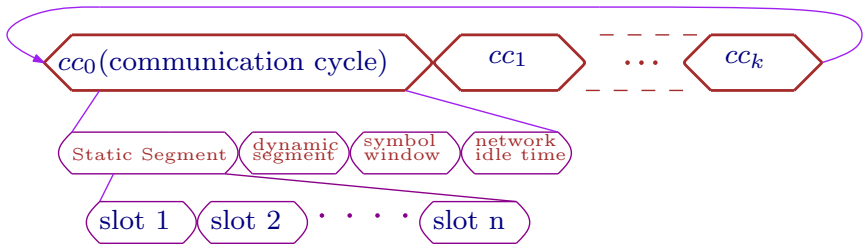

Figure : *Basic structure of FlexRay illustrating communication cycles*

- FlexRay schedule is organised in multiple repeating communication cycles
- Each cycle is divided into 4 different segments
  - Static segment
  - Dynamic segment
  - Symbol window
  - Network idle time
- WE are here only considering static segment:
  - Consists of fixed number slots with equal size
  - Time-triggered static segment used for scheduling

[12] Florian Sagstetter, Martin Lukasiewycz and Samarjit Chakraborty, Generalized Asynchronous Time-Triggered Scheduling for FlexRay, IEEE Trans. on CAD of Integrated Circuits and Systems

FlexRay Parameters

- Number of communication cycle n
- The cycle duration t_c
- Number of available static slots in one cycle n
- Message size l_m and takes period p_m
- Message repetition $r_m = (p_m / t_c)$
- Message m is scheduled in the first cycle called Base cycle b_m
- A Message m is scheduled in a cycle cc_i when $i = (b_m + r_m \cdot a) \mod k, a \epsilon \mathbb{N}_0$
- If $\forall i, j \epsilon \mathbb{N}_0 : (b_{m_1} + r_{m_1} \cdot i) \mod k \neq (b_{m_2} + r_{m_2} \cdot j) \mod k$ then m_1 and m_2 will not intersect

Our Planning

FlexRay parameters shows that whether two messages will intersect or not depends on the base cycle and number of repetitions. So our idea is:

- Initially system has defined static scheduling sequence
- That sequence says the base cycle and the number of repetitions of each messages
- Now if a message misses its allocated cycle due to delay and tries to access another cycle which is not granted for it, then a situation of conflict will arise
- We are trying to get a scheduler Büchi automata, that can capture the conflict

Last Slide

Thank You