REAFFIRM: Model-Based Repair of Hybrid Systems for Improving Resiliency

Luan Viet Nguyen*, Gautam Mohan†, James Weimer†, Oleg Sokolsky†, Insup Lee†, and Rajeev Alur†
* Department of Computer Science, University of Dayton, OH, USA,
† Department of Computer and Information Science, University of Pennsylvania, PA, USA

***Abstract*— Model-based design offers a promising approach for assisting developers to build reliable and secure cyber-physical systems in a systematic manner. In this methodology, a designer first constructs a model, with mathematically precise semantics, of the system under design, and performs extensive analysis with respect to correctness requirements before generating the implementation from the model. However, as new vulnerabilities are discovered, requirements evolve aimed at ensuring resiliency. There is currently a shortage of an inexpensive, automated software that can effectively repair the initial design, and a model-based system developer regularly needs to redesign and reimplement the system from scratch.**

In this paper, we propose a new methodology along with a MATLAB software called REAFFIRM to facilitate the model-based repair for improving the resiliency of cyber-physical systems. REAFFIRM takes as inputs 1) an original hybrid system modeled as a Simulink/Stateflow diagram, 2) a given resiliency pattern specified as a model transformation script, and 3) a safety requirement expressed as a Signal Temporal Logic formula, and outputs a repaired model which satisfies the requirement. The tool consists of two main modules, model transformation followed by model synthesis. While the latter component is built on top of the falsification tool Breach, to implement the former, we introduce a new model transformation language for hybrid systems, which we call HATL, to allow a designer to specify resiliency patterns. To evaluate the proposed approach, we use REAFFIRM to automatically synthesize the repaired models of four different case studies.

***Keywords*—Model-based repair, resiliency, transformation language, hybrid systems.**

I. Introduction

A cyber-physical system (CPS) consists of computing devices communicating with one another and interacting with the physical world via sensors and actuators. Increasingly, such systems are everywhere, from smart buildings to autonomous vehicles to mission-critical military systems. The rapidly expanding field of CPSs precipitated a corresponding growth in security concerns for these systems. The increasing amount of software, communication channels, sensors and actuators embedded in modern CPSs make them likely to be more vulnerable to both cyber-based and physics-based attacks [1], [2], [3], [4], [5]. As an example, *sensor spoofing* attacks to CPSs become prominent, where a hacker can arbitrarily manipulate the sensor measurements to compromise secure information or to drive the system toward unsafe behaviors. Such attacks have successfully disrupted the braking function of the anti-lock braking systems [6], [4], and compromised the insulin delivery service of a diabetes therapy system [7]. Alternatively, attackers can gain access to communication channels, and this can be used to manipulate the switching behavior of a smart power grid [8] or disable the brake system of a modern vehicle [9]. Generally, constructing a behavioral model at design time that offers resiliency for all kinds of attacks and failures is notoriously difficult.

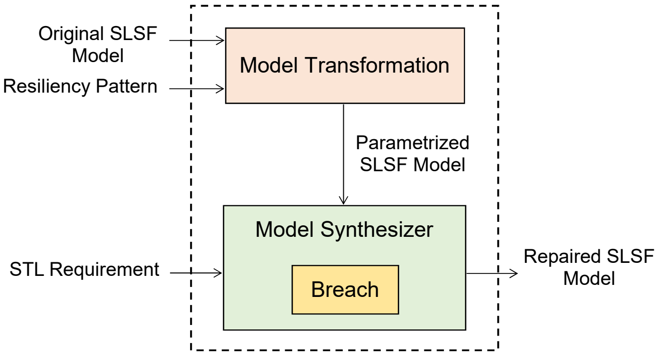


Fig. 1. REAFFIRM overview.

Traditionally a model of a CPS consists of block diagrams describing the system architecture and a combination of state machines and differential equations describing the system dynamics [10]. Suppose a designer has initially constructed a model of a CPS that satisfies correctness requirements, but at a later stage, this correctness guarantee is invalidated, possibly due to adversarial attacks on sensors, or violation of environment assumptions. Current techniques for secure-by-design systems engineering do not provide a formal way for a designer to specify a resiliency pattern to automatically repair system models based on evolving resiliency requirements under unanticipated attacks.

In this paper, we propose a new methodology and an associated software, called REAFFIRM, to assist a designer in repairing the original model so that it continues to satisfy the correctness requirements under the modified assumptions. The proposed technique relies on designing a collection of *potential edits* (or *resiliency patterns*) to the original model to generate the new model whose parameters values can be determined by solving the *parameter synthesis problem*. Figure 1 shows an overview of REAFFIRM, which contains two main modules 1) a *model transformation*, and 2) a *model synthesizer* built on top of the falsification tool Breach [11].

978-1-7281-9148-5/20/$31.00 © 2020 IEEE

REAFFIRM takes the following inputs 1) the original system modeled as a Simulink/Stateflow (SLSF) diagram, 2) the resiliency pattern specified by the designer and 3) the safety requirement expressed as a Signal Temporal Logic (STL) [12] formula, and outputs the repaired SLSF model that satisfies the safety requirement such that no counterexample found by the falsifier or the maximum number of iterations specified by a user is reached.

To allow a designer to specify resiliency patterns we have developed a new *model transformation language* for hybrid systems, called HATL (Hybrid Automata Transformation Language). A HATL script is a sequence of statements that describe the modifications over the structure of hybrid systems modeled as hybrid automata [10]. Examples of edits to a model include creating new modes of operations, duplicating modes, adding transitions, modifying switching conditions, and substituting state variables in flow equations. The proposed language allows the designer to write a resiliency pattern in a generic manner, and programmatically modify the initial design without knowing the internal structures of a system. The HATL interpreter is implemented in Python with an extensible backend to allow interoperability with different hybrid systems modeling frameworks. The current implementation of HATL supports MATLAB and performs transformations on SLSF models.

For evaluation, we apply REAFFIRM to automatically synthesize the repaired models for four case studies in the domains of automotive control, smart power systems and aerospace applications. The first case study is a simplified model of an adaptive cruise control (ACC) system under a GPS sensor spoofing attack, and the resiliency pattern to fix the model is to ignore the GPS measurement and only use the wheel encoders, which are additional (redundant) sensors for estimating a vehicle's velocity. REAFFIRM automatically synthesizes the condition that triggers a switch to a copy of the model that ignores the GPS measurement.

The second case study is a single-machine infinite-bus (SMIB) model, which is an approximation of a smart power grid, under a sliding-mode attack. In this case, the mitigation strategy is to increase the minimal dwell-time to avoid rapid changes between different operation modes. Thus, the resiliency pattern adds a dwell-time variable in each mode of the model, and the minimal dwell-time can be determined automatically by REAFFIRM.

The third case study is a waypoint tracking system (WT) which is an example of Simplex Architecture [13] where the complex controller fails to maintain a safe operation. Here, the resiliency pattern is adding a decision module in which a switching condition from a complex controller to a safety controller can be synthesized using REAFFIRM.

The fourth case study is the missile guidance system (MG) provided by Mathworks, which is a good representative of a practical MG system as it has more than 300 SLSF blocks. For the MG system, we investigate two different kinds of attacks: 1) a gyroscope sensor attack, and 2) an angular noise injection attack. The principle of a spoofing attack on the gyroscopes of the MG system is similar to the GPS spoofing attack of the ACC system, and then we can apply the same resiliency pattern used to fix the ACC model for repairing the MG model. In the case of the angular noise injection attack, REAFFIRM can perform a global sensitivity analysis over the parameters and control gains of the MG system, and then automatically synthesize their new values that makes the system continue to satisfy the correctness requirement.

In sum, the main contributions of the paper are as follows.

1) The methodology to facilitate the model-based repair for improving the resiliency of CPSs against unanticipated attacks and failures,
2) the design and implementation of an extensible model transformation language for specifying resiliency patterns used to repair CPS models,
3) the end-to-end design and implementation of the software, which integrates the model transformation and the model synthesis tools to automatically repair CPS models,
4) the applicability of our software on four proof-of-concept case studies where the CPS models can be repaired to mitigate practical attacks.

The remainder of the paper is organized as follows. Section II presents an overview of our proposed methodology through a simplified example of the ACC system, and introduces the architecture of REAFFIRM. Section III describes our model transformation language used to design a resiliency pattern for hybrid systems. Section IV presents the model synthesizer of REAFFIRM. Section V presents four case studies that illustrate the capability of REAFFIRM in automatically repairing the original models of 1) the ACC system under a GPS sensor spoofing attack, 2) the SMIB system under a sliding-mode attack, 3) the WT system in the case of safety failure, and 4) the MG system under a gyroscope sensor attack and an angular noise injection attack. Section VI reviews the related works to REAFFIRM. Section VII concludes the paper and presents our future works.

II. Overview of the Methodology

In this section, we will explain our methodology through a simplified example of the adaptive cruise control (ACC) system. Assume that a designer has previously modeled the ACC system as a combination of the vehicle dynamics and an ACC module, and GPS measurements were considered trusted in the initial design. In the following, we will describe the ACC system as originally designed, an attack scenario, and an example of resiliency pattern to repair the ACC model. Then, we present how REAFFIRM can automatically perform a model transformation and synthesis to construct a new ACC model with resiliency. We note that the ACC model presented herein is not a representative of the complexity of a true ACC system, but a simplified example in which the dynamics and control equations are chosen for simplicity of presentation, and sensor measurements are not considered complex mechanisms such as sensor fusion and Kalman filtering.

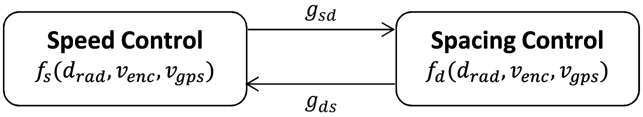

Speed Control
$f_s(d_{rad}, v_{enc}, v_{gps})$
Spacing Control
$f_d(d_{rad}, v_{enc}, v_{gps})$
g_{sd}
g_{ds}

Fig. 2. An original ACC model.

A. A Simplified Example of ACC System

For simplicity, we assume that the designer initially models the ACC system (including vehicle dynamics) as a hybrid system shown in Figure 2. The original ACC system operates in two modes: *speed control* and *spacing control* whose dynamics are governed by the differential equations f_s and f_d, respectively. In speed control, the host car travels at a driver-set speed. In spacing control, the host car aims to maintain a safe distance from the lead car. The vehicle has two state variables: d is the distance to the lead car, and v is the speed of the host vehicle. The ACC system has two sensors that measure its velocity v via noisy wheel encoders, $v_{enc} = v + n_{enc}$, and a noisy GPS sensor, $v_{gps} = v + n_{gps}$, where n_{enc} and n_{gps} denote the encoder and GPS noises, respectively. Additionally, the ACC system has a radar sensor that measures the distance to the lead vehicle, $d_{rad} = d + n_{rad}$, where n_{rad} captures a corresponding noise. The ACC system decides which mode to use based on the real-time sensor measurements. For example, if the lead car is too close, the controller triggers the transition g_{sd} to switch from speed control to spacing control. Similarly, if the lead car is further away, the ACC system switches from spacing control to speed control by executing the transition g_{ds}. The *safety specification* of the system is that d should always be greater than d_{safe}, where $d_{safe} = v + 5$. We will describe the ACC model in more details in Section V.

Safety violation under GPS sensor attack. In this example, we assume that after designing and verifying the initial ACC system, it is determined that the GPS sensor can be *spoofed* [14], [15]. GPS spoofing occurs when incorrect GPS packets (possibly sent by a malicious attacker) are received by the GPS receiver. In the ACC system, this allows an attacker to arbitrarily change the GPS velocity measurement. Thus, a new scenario occurs when the original assumption of GPS noise, e.g., $|n_{gps}| \leq 0.05$ is omitted, and the new assumption is $|n_{gps}| \leq 50$. As a result, the safety specification could be violated under the GPS sensor attacks, and a designer needs to repair the original model using a known mitigation strategy.

Example of resiliency pattern: ignoring GPS measurement. The ACC system has redundancy in the sensory information of its estimated velocity, one derived from the GPS and the other from the wheel encoders. Thus, to provide resilience against the GPS attacks, a mitigation strategy is to ignore the GPS value, and use only the wheel encoders to estimate velocity. Thus, a potential fix is first to create a copy of the original model where the controller simply ignores the GPS reading as it can no longer be trusted. Then, adding new transitions from the modes of the original model to the corresponding instances of the copy that uses only the wheel encoder to measure velocity. We note that this transformation is *generic*, that is, it can be applied in a uniform manner to any given model simply by creating a duplicate version of each original mode and transition, copying the dynamics in each mode, but without a reference to the variable v_{gps}.

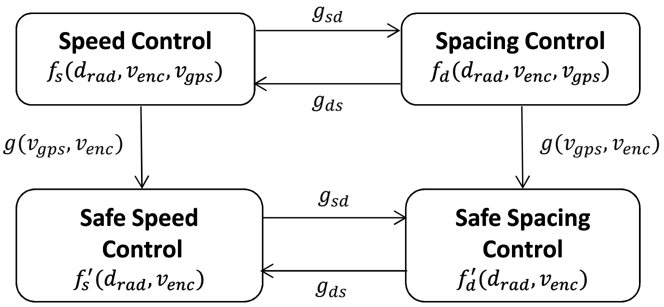


Fig. 3. A repaired ACC model without a reference to GPS sensor under spoofing attacks.

Figure 3 illustrates the repaired model in which the transition from the original speeding and spacing control modes to their copies is an expression over v_{gps} and v_{enc}. Observe that while it would be possible to use only the wheel encoders all the time, a better velocity estimate can be obtained by using an average velocity measurement (from both the GPS and wheel encoders) when the GPS sensor is performing within nominal specifications. The main analysis question is when should the model switch from the original modes to the copied modes during the spoofing attack. From a practical standpoint, such a transition should occur when the GPS measurement significantly deviates from the wheel encoder measurement, and a transition condition can be specified as $g(v_{gps}, v_{enc}) = |v_{gps} - v_{enc}| \geq \theta$, where θ is an unknown parameter. Since $v_{enc} = v + n_{enc}$ and $v_{gps} = v + n_{gps}$, we can rewrite the transition condition as $g(v_{gps}, v_{enc}) = |n_{gps} - n_{enc}| \geq \theta$. Then, one needs to synthesize the suitable value of the parameter θ that specifies the threshold for switching from the original copy to the new copy so that the safety requirement is satisfied.

B. REAFFIRM Software

Our REAFFIRM prototype for the model-based repair is built in MATLAB and consists of two main modules, corresponding to *model transformation* and *parameter synthesis*. To synthesize the model with resiliency to unanticipated attacks, users need to provide the following inputs to REAFFIRM:

- the initial design of a hybrid system modeled in MathWorks SLSF format,
- the resiliency pattern specified as a model transformation script that transforms the initial model to the new model with resiliency to unanticipated attacks, and
- the correctness (safety) requirement of the system specified as an STL formula.

In the case of the ACC example, the inputs of REAFFIRM are the initial SLSF model shown in Figure 2, the resiliency pattern that creates the copied version of the original model without a reference to the variable v_{gps}, and the safety requirement encoded as an STL formula,

$$\varphi_{ACC} = \square_{[0,\infty)} d[t] > 5 + v[t]. \tag{1}$$

The model transformation tool of REAFFIRM takes the initial SLSF model and the resiliency pattern (e.g., the transformation script shown in Figure 4), and then generates the new SLSF model that contains a parameter θ that appears in the switching condition based on the difference between the GPS measurement and the wheel encoder measurement. Then, the model synthesizer tool of REAFFIRM takes the parametrized ACC model in SLSF and the STL formula φ_{ACC} as inputs, and then applies synthesis to find the desired value of θ over a certain range, to ensure that φ_{ACC} is satisfied. Internally, the model synthesizer of REAFFIRM utilizes an open-source model falsification tool—Breach [11] to synthesize the desired parameters values. If the synthesizer can find the best value of θ over the given range, then REAFFIRM outputs a competed SLSF model which satisfies φ_{ACC} under the GPS attacks. Otherwise, the tool will suggest the designer to either search over different parameter ranges or try different resiliency patterns to repair the ACC model.

III. Model Transformation

A. Representation of Hybrid System

Hybrid automata [10] are a modeling formalism popularly used to model hybrid systems which include both continuous dynamics and discrete state transitions. A hybrid automaton is essentially a finite state machine extended with a set of real-valued variables evolving continuously over time [10]. The main structure of a hybrid automaton $\mathcal{H}$ includes the following components.

- $\mathcal{X}$: the finite set of n continuous, real-valued variables.
- $\mathcal{P}$: the finite set of p real-valued parameters.
- *Mode*: the finite set of discrete modes. For each mode $m \in Mode$, $m.inv$ is an expression over $\mathcal{X} \cup \mathcal{P}$ that denotes the invariant of mode m, and $m.flow$ describes the continuous dynamics governed by a set of ordinary differential equations.
- *Trans*: the finite set of transitions between modes. Each transition is a tuple $\tau \triangleq \langle source, destination, guard, reset \rangle$, where *source* is a source mode and *destination* is a target mode that may be taken when a guard condition *guard*, which is an expression over $\mathcal{X} \cup \mathcal{P}$, is satisfied, and *reset* is an assignment of variables in $\mathcal{X}$ after the transition.

We use the dot (.) notation to refer to different components of tuples, e.g., $\mathcal{H}.Trans$ refers to the transitions of automaton $\mathcal{H}$ and $\tau.guard$ refers to the guard of a transition τ. Since our goal is to repair a hybrid automaton syntactically, we will not discuss its semantics in this paper, but refer a reader to [10] for details. We note that the *model transformation language* proposed in this paper transforms a hybrid automaton based on modifying the syntactic components of the hybrid automaton in a generic manner. The transformation tool of REAFFIRM can take a HATL transformation script and translate it into an equivalent script that performs a model transformation for different modeling framework of hybrid automata including a continuous-time Stateflow chart.

```
# original model is retrieved from command line arguments
model_copy = model.copyModel() # make a model copy
# start a transformation
model.addParam("theta") # add new parameter theta
formode m = model.Mode {
    m_copy = model.addMode(m)
    m.replace(m_copy.flow,"ngps","nenc")
    model.addTransition(m,m_copy,"abs(ngps-nenc)>theta")
}
fortran t = model_copy.Trans {
    # get source and destination modes of transition t
    src = t.source
    dst = t.destination
    # retrieve copies of source and destination modes
    src_copy = model.getCopyMode(src)
    dst_copy = model.getCopyMode(dst)
    model.addTransition(src_copy,dst_copy,t.guard)
}
# end of the transformation
```

Fig. 4. An example of a resiliency pattern written as a HATL script for the ACC system.

Continuous-time Stateflow chart. In this paper, we represent hybrid automata using *continuous-time* Stateflow chart, which is a standard commercial modeling language for hybrid systems integrated within Simulink. A continuous-time Stateflow chart supplies methods for engineers to quickly model as well as efficiently refine, test, and generate code for hybrid automata. The syntactic description of a continuous-time Stateflow chart is basically a hybrid automaton, with a small few differences. In particular, a mode is a *state* associated with different types of actions including a) *entry* action executed when entering the state, b) *exit* executed when exiting the state, and c) *during* (or *du*) action demonstrates the continuous-time evolution of the variables (i.e., *flow* dynamics) when no transition is enabled. A variable can be specified as *parameter*, *input*, *output*, and *local variable*. Also, an SLSF model which includes a continuous-time Stateflow chart is deterministic since its transition is urgent and executed with priorities. Intuitively, a transition in a Stateflow chart is triggered as soon as the transition guard condition is satisfied, while a hybrid automaton can stay at the current mode as long as its invariant still holds. To overcome this gap, a recent work proposed in [16] provides an equivalent translation for both classes of deterministic and non-deterministic hybrid automata to Stateflow diagrams. Other significant research have been done to translate back and forth between hybrid automata and SLSF models [17], [18], [19].

B. Hybrid Automata Transformation Language

In our approach, the partial model of the system, which satisfies functional but not necessarily resiliency requirements is originally modeled in the form of hybrid automata. The model transformation that is at the core of the REAFFIRM tool will then attempt to modify the components of the automata such as modes, flows, or switching logic, by applying user-defined resiliency patterns.

In order to specify resiliency patterns for hybrid automata, we introduce a new language for model transformation called HATL (Hybrid Automata Transformation Language). The goal of HATL is to allow a designer to repair an original model in a programmatic fashion. HATL scripts abstract model implementation details so engineers do not need to learn the intricacies of an individual framework. A key use of HATL is to write generic scripts that are applicable to many models, promoting resiliency scripts which are reusable. A script written in HATL is a sequence of *statements* that specify the changes over the structure of given hybrid automata. HATL's syntax and semantics are designed to make it intuitive to anyone who is familiar with imperative languages. HATL includes *loop* statements that iterate over sets of objects, such as modes or transitions of a model. It uses *dot references* to index into structures to obtain data fields or to call object-specific methods. *Assignments* are mutable, and scoped within statement blocks. Functions and methods can have variable numbers of arguments which are eagerly evaluated.

The model transformation tool built in REAFFIRM takes a resiliency pattern in the form of a HATL script, and then translates each of the statements of the script into equivalent transformation operations on continuous-time Stateflow models. Figure 4 shows an example of a HATL script that specifies a transformation from the original ACC model shown in Figure 2 to the parametrized model shown in Figure 3. In this script, we first create a copy of the original model. Next, we iterate over each mode of the model by calling the *formode loop*, make a copy with replacing the variable n_{gps} by n_{enc}, and then add a new transition from the original mode to the copied mode with a new guard condition. This guard condition is a constraint specified over the difference between n_{gps} by n_{enc} (i.e., the difference between v_{gps} and v_{enc}) and a new parameter θ, which is added into the model using a function call *addParam*. Finally, we need to copy all transitions between original modes (stored in a copied version of the original model) and assign them to the corresponding duplicated modes.

C. Implementation

Our current implementation dynamically interprets HATL scripts in Python and translates them into SLSF model transformations via the MATLAB Engine. Our interpreter checks argument values at runtime to ensure only valid transformed models are produced. If a malformed program statement is detected, HATL will throw a verbose error message and roll back any changes it has applied already before exiting. Additionally, these error messages are reported in terms of generic HATL models, so an engineer writing a resiliency pattern does not need to worry about the underlying implementation.

Currently, HATL provides enough programming abstraction to express concise model transformations that function as valid resiliency patterns, and more examples of these scripts will be introduced in Section V. There is room for future improvement, such as adding language constructs like type checking to verify the type correctness of the model before and after repair.

IV. Model Synthesis

In this section, we present the model synthesizer incorporated in REAFFIRM which takes a parameterized model produced by the model transformation, and a correctness requirement as inputs, and then generates a completed model with parameter values instantiated to satisfy the correctness requirements. Since the structure of the completed model is already determined after the model transformation, the model synthesis problem then reduces to the *parameter synthesis problem*. Let $\mathcal{P}_s$ be the set of parameters of the transformed model $\tilde{\mathcal{H}}$, given a safety specification φ and sets of parameter values $\bar{\mathcal{P}}_s$, find the best instance values of $\mathcal{P}_s$ over $\bar{\mathcal{P}}_s$ so that $\tilde{\mathcal{H}} \models \varphi$. For example, the transformation of the ACC model shown in Figure 3 introduces a new parameter θ whose value needed to be determined so that the completed model will satisfy the safety requirement with respect to the same initial condition of the state variables and parameters domains of the original model.

A. Overview of Breach

We incorporated Breach into the model synthesizer of REAFFIRM as an analysis mechanism to perform the falsification and parameter synthesis for hybrid systems. Given a hybrid system modeled as an SLSF diagram, an STL specification described the safety property, and specific parameter domains, Breach [11] can perform an optimized search over the parameter ranges to find parameter values that cause the system violating the given STL specification. The parameter mining procedure is guided by the counterexample obtained from the falsification, and it terminates if there is no counterexample found by the falsifier or the maximum number of iterations specified by a user is reached. On the other hand, Breach can compute the sensitivity of execution traces to the initial conditions, which can be used to obtain completeness results by performing systematic simulations. Moreover, Breach provides an input generator for engineers to specify different testing input patterns such as step, pulse width, sinusoid, and ramp signals. This input generator is designed to be extensible, so users can write a specific input pattern to test their model against particular attack scenarios.

We note that although Breach cannot completely prove the system correctness, it can efficiently find bugs existing in the initial design of CPS that are too complex to be formally verified [20]. These bugs are essential for an engineer to specify resiliency patterns to repair the model. Moreover, the general problem of verifying a CPS modeled as a hybrid system is known to be *undecidable* [21]. Instead, the falsification algorithms embedded within Breach are scalable and work properly for black-box hybrid systems with different classes of dynamics. Thus, in practice, engineers prefer to use counterexamples obtained by a falsification tool to refine their design. Our prototype REAFFIRM utilizes the advantages of SLSF modeling framework and the falsification tool Breach to design a resiliency pattern and perform the model synthesis for a repaired CPS model with resiliency.

B. Model Synthesis using Breach

Next, we describe how REAFFIRM uses Breach to synthesize parameters values for the parametrized model returned from the model transformation tool. The parameter synthesis procedure consists of following steps.

1) We first specify the initial conditions of state variables and parameters, the set of parameters $\mathcal{P}_s$ that need to be mined, the sets of parameter values $\bar{\mathcal{P}}_s$, and the maximum time (or number of iterations) for the optimization solver of Breach.
2) Next, we call the falsification loop within Breach to search for a counterexample. For each iteration, if the counterexample is exposed, the unsafe values of $\mathcal{P}_s$ will be returned. Based on these values, the tool will automatically update the sets of parameter values $\bar{\mathcal{P}}_s$ to the new sets of parameter values $\bar{\mathcal{P}}'_s \subset \bar{\mathcal{P}}_s$, and then continue the falsification loop.
3) The process repeats until the property is satisfied that means the falsifier cannot find a counterexample and the user-specified limit on the number of optimized iterations (or time) for the solver expires.
4) Finally, the tool returns the best (and safe) values of $\mathcal{P}_s$, updates the parametrized model with these values, and then exports the completed model. If the synthesizer fails to find the values of $\mathcal{P}_s$ over the given sets of parameter values $\bar{\mathcal{P}}_s$ so that the safety requirement is satisfied, it will recommend a designer to either search over different parameter ranges or try another resiliency pattern.

Monotonic Parameters. The search over the parameter space of the synthesis procedure can be significantly reduced if the satisfaction value of a given property is monotonic w.r.t to a parameter value. Intuitively, the satisfaction of the formula monotonically increases (respectively decreases) w.r.t to a parameter p that means the system is more likely to satisfy the formula if the value of p is increased (respectively decreased). In the case of monotonicity, the parameter space can be efficiently truncated to find the *tightest* parameter values such that a given formula is satisfied. In Breach, the check of monotonicity of a given formula w.r.t specific parameter is encoded as an SMT (Satisfiability Modulo Theories) query and then is determined using an SMT solver. However, the result may be *undecidable* due to the undecidability of STL [22]. In this paper, the synthesis procedure is based on the assumption of satisfaction monotonicity. If the check of monotonicity is undecidable over a certain parameter range, a user can manually enforce the solver with decided monotonicity (increasing or decreasing) or perform a search over a different parameter range.

V. Model Repair for Resiliency

In this section, we demonstrate the capability of REAFFIRM to repair CPSs models under unanticipated attacks. We first revisit the ACC example and evaluate three resiliency patterns that can be applied to repair the ACC model under the GPS sensor spoofing attack. Second, we investigate a sliding-mode switching attack that causes instability for a smart grid system and how REAFFIRM can use a dwell-time pattern to repair the model under this attack automatically. Third, we use REAFFIRM to synthesize a switching condition from the complex controller to the safety controller of the WT system to avoid a safety failure. Finally, we apply the three resiliency patterns used to fix the ACC model to repair the MG system under a gyroscope sensor attack, and also demonstrate how REAFFIRM can efficiently tune the control gains and parameters of the system to mitigate an angular noise attack. REAFFIRM was tested using MATLAB 2018a and MATLAB 2018b executed on an x86-64 laptop with 2.8 GHz Intel(R) Core(TM) i7-7700HQ processor and 32 GB RAM. All performance metrics reported were recorded on this system using MATLAB 2018a. In Breach, we choose the CMAES solver, and the maximum optimization time is 30 seconds for each iteration of the falsification loop. REAFFIRM and all case studies investigated in this paper are available to download at https://github.com/LuanVietNguyen/reaffirm. The overall performance of REAFFIRM in repairing the initial models of four case studies to mitigate their corresponding attacks is summarized in Table I. The transformation time reported in the table is the actual time required for the model transformation by neglecting the overhead of loading the MATLAB Engine in Python. Next, we will describe four case studies in more details.

A. Adaptive Cruise Control System

Original SLSF model. We previously introduced the simplified example of the ACC system in Section II to illustrate our approach. In this section, we present the ACC system in more details. The ACC system can be modeled as the SLSF model shown in Figure 5. The model has four state variables where d and e_d are the actual distance and estimated distance between the host car and the lead vehicle, v and e_v represent the actual velocity and estimated velocity of the host car, respectively. In this model, we assume that the lead vehicle travels with a constant speed v_l. The transition from speed control to spacing control occurs when the estimate of the distance is less than twice the estimated safe distance, i.e., $e_d < 10 + 2e_v$. A similar condition is provided for switching from spacing control to speed control, i.e., $e_d \geq 10 + 2e_v$. In this case study, we assume that the designer has verified the initial SLSF model of the ACC system against the safety requirement φ_{ACC} under the scenario when $d(0) \in [90, 100]$, $v(0) \in [25, 30]$, $|d(0) - e_d(0)| \leq 10$, $|v(0) - e_v(0)| \leq 5$, $v_l = 20$, $|n_{rad}| \leq 0.05$, $|n_{enc}| \leq 0.05$ and $|n_{gps}| \leq 0.05$.

GPS sensor attack. To perform a spoofing attack on the GPS sensor of the ACC model, we continuously inject false data to manipulate its measurement value. In this case, we omit the original assumption $|n_{gps}| \leq 0.05$, and employ the new assumption as $|n_{gps}| \leq 50$. Using the input generator in

Model	BD	Attack/Failure	Resiliency Pattern		Unknown Condition	PR	SV	TT	ST
ACC	11	GPS spoofing	Ignore GPS, measurement, use wheel encoders value	Pattern 1	When to switch to a safe copy	$\theta \in [0, 50]$	7.08515	2	88
				Pattern 2			7.08515	2	88
				Pattern 3	Ratio of GPS/encoders measurements	$\theta \in [0.1, 0.9]$	0.1543	1.75	56
SMIB	15	Sliding-mode switching	Add a dwell-time to avoid rapid switching	Pattern dwell-time	Minimal dwell-time	$\theta \in [0, 0.3]$	0.12	2	45
WT	25	Out of safe boundary	Add a transition from a complex controller to a safety controller	Pattern simplex architecture	Switching boundary	$\theta \in [0, 1]$	0.625	2	15
MG	310	Gyroscopes spoofing	Ignore untrusted, measurements, use the trusted ones	Pattern 1	When to switch to a safe copy	$\theta \in [0, 0.5]$	0.06714	2	78
				Pattern 2			0.06714	2	92
				Pattern 3	Ratio of gyroscopes measurements	$\theta \in [0.01, 0.1]$	0.01127	1.75	55
		Angular noise injection	Change control feedback (no model transformation)	Sensitivity analysis + max-satisfaction	What control gains or parameters should be tuned	$tors \in [0, 0.25]$	0.23421	0	31

TABLE I
REAFFIRM PERFORMANCE RESULTS FOR THE ACC, SMIB, WT AND MG CASE STUDIES. BD IS THE NUMBER OF BLOCKS IN SLSF MODELS. PR IS THE PARAMETER RANGE. SV IS THE SYNTHESIZED VALUE. TT AND ST ARE THE TRANSFORMATION AND SYNTHESIS TIME IN SECONDS, RESPECTIVELY.

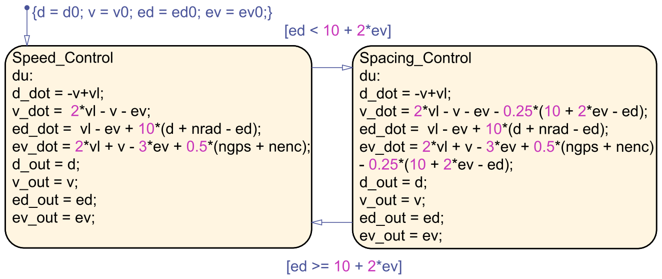

{d = d0; v = v0; ed = ed0; ev = ev0;}
[ed < 10 + 2*ev]
Speed_Control
du:
d_dot = -v+vl;
v_dot = 2*vl - v - ev;
ed_dot = vl - ev + 10*(d + nrad - ed);
ev_dot = 2*vl + v - 3*ev + 0.5*(ngps + nenc);
d_out = d;
v_out = v;
ed_out = ed;
ev_out = ev;
Spacing_Control
du:
d_dot = -v+vl;
v_dot = 2*vl - v - ev - 0.25*(10 + 2*ev - ed);
ed_dot = vl - ev + 10*(d + nrad - ed);
ev_dot = 2*vl + v - 3*ev + 0.5*(ngps + nenc)
- 0.25*(10 + 2*ev - ed);
d_out = d;
v_out = v;
ed_out = ed;
ev_out = ev;
[ed >= 10 + 2*ev]

Fig. 5. The original SLSF model of the ACC system.

Breach, we can specify the GPS spoofing attack as a standard input test signal such as a constant, ramp, step, sinusoid or random signal. The following evaluations of three different resiliency patterns used to repair the ACC model are based on the same assumption that the GPS spoofing occurs at every time point, specified as a random constant signal over the range of [-50, 50] during 50 seconds.

Model repair for the ACC system. Under the GPS sensor spoofing attack, the original SLSF model does not satisfy its safety requirement and a designer needs to apply a certain resiliency pattern to repair the model. The *first resiliency pattern* for repairing the ACC system has been introduced in Section II, which makes the copy of the original model where the controller ignores the GPS reading as it can no longer be trusted. However, we need to determine the best switching condition from the original model to the copy. For the first pattern, REAFFIRM output the repaired model with a synthesized value of $\theta = 7.08515$.

```
# start a transformation
model.addParam("theta")
formode m = model.Mode {
     m.replace(m.flow,"ngps", "2*theta*ngps")
     m.replace(m.flow,"nenc", "2*(1-theta)*nenc")
}
# end of the transformation
```

Fig. 6. The third resiliency pattern for the ACC system based on the linear combination of n_{enc} and n_{gps}.

The *second resiliency pattern* for the ACC model is the extended version of the first one where it includes a switching-back condition from the copy to the original model when the GPS sensor attack is detected and mitigated. An example of such a switching-back condition is when the difference between the n_{enc} and n_{gps} are getting smaller, i.e., $|n_{gps} - n_{enc}| < \theta - \epsilon$, where ϵ is a positive user-defined tolerance. For this pattern, the model transformation script can be written similar to the one shown in Figure 4 with adding the *addTransition* function from the copy mode to the original mode with the guard condition labeled as $|n_{gps} - n_{enc}| < \theta - \epsilon$ in the *formode* loop. The performance of REAFFIRM for the second pattern is similar to the first pattern with the same synthesized value of $\theta = 7.08515$ and $\epsilon = 0$.

Alternatively, the *third resiliency pattern*, where we do not need to modify the structure of the original model, is to model the redundancy in the sensory information as a linear combination of different sensor measurements. For example, instead of taking the average of n_{gps} and n_{enc}, we can model their relationship as $\theta n_{gps} + (1 - \theta) n_{enc}$, and then

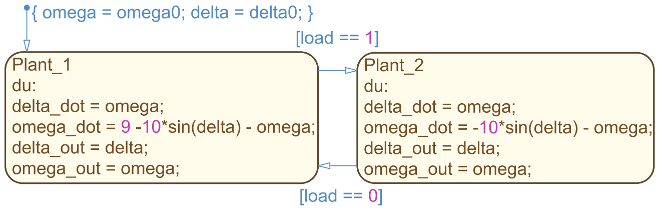


Fig. 7. The original SLSF model of the SMIB system.

synthesize the value of θ so that the safety property is satisfied. The transformation script of this resiliency pattern is given in Figure 6. For this pattern, we assume that a designer still wants to use all sensor measurements even some of them are under spoofing attacks and would like to search for the value of θ over the range of [0.2, 0.8] (instead of [0, 1]). Given the same attack model for the other patterns, the synthesizer in REAFFIRM fails to find the value of θ within the given range to ensure that the safety property is satisfied. However, if we enlarge the range of θ to [0.1, 0.9], the synthesizer successfully finds the safe value $\theta = 0.1543$.

B. Single-Machine Infinite-Bus System

Next, we study a class of cyber-physical switching attacks that can destabilize a smart grid system model, and then apply REAFFIRM to repair the model to provide resilience. A smart power grid system such as the Western Electricity Coordinating Council (WECC) 3-machine, 9-bus system [23], can be represented as a single-machine infinite-bus (SMIB) system described in [24]. The SMIB system is considered as a *switched system* in which the physical dynamics are changed between two operation modes based on the position of the circuit breaker. The system has two states, δ_1 and ω_1, which are the deviation of the rotor angle and speed of the local generator G_1 respectively. The stability (safety) property of the system can be specified as the following STL formula,

$$\varphi_{SMIB} = \Box_{[0,T]}(0 \leq \delta_1[t] \leq 3.5) \wedge (-2 \leq \omega_1[t] \leq 3), \quad (2)$$

where T is a simulation duration.

Original SLSF model. In this paper, we model the SMIB system as the SLSF model displayed in Figure 7. The model contains two operation modes whose nonlinear dynamics characterize the transient stability of the local generator G_1 presented in [24]. The transitions between two operation modes depend on the status of the circuit breaker which is connected or disconnected to the load. In the model, δ_1 and ω_1 are represented by *delta* and *omega*, respectively; and the initial conditions are $delta0 \in [0, 1.1198]$ and $omega0 \in [0, 1]$. The discrete variable *load* captures the open and closed status of the circuit breaker.

Sliding-mode attack. The SMIB system has an interesting property known as a *sliding mode* behavior. This behavior occurs when the state of the system is attracted and subsequently stays within the *sliding surface* defined by a state-dependent switching signal $s(x) \in \mathbb{R}$ [25], [26]. When the system is confined on a sliding mode surface, its dynamics exhibit high-frequency oscillations behaviors, a so-called *chattering* phenomenon, which is well-known in the power system design [27]. At this moment, if an attacker forces rapid switching between two operation modes, the system will be steered out of its desirable equilibrium position. As a result, the power system becomes unstable even each individual subsystem is stand-alone stable [26]. More details of the stages to construct the sliding-mode attack can be found in [24].

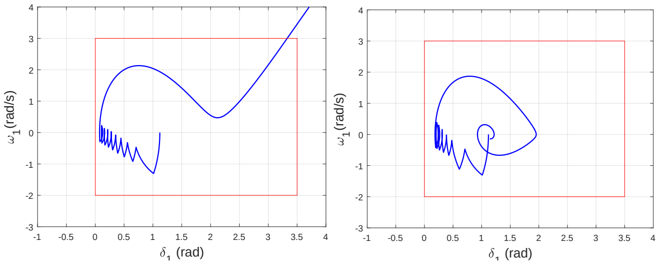


Fig. 8. Left: an unstable trajectory of the original SMIB model under the sliding-mode attack. Right: a stable trajectory of the repaired SMIB model under the sliding-mode attack.

```
# start a transformation
model.addParam("theta")
model.addLocalVar("clock") # add a clock variable
formode m = model.Mode {
    m.addFlow("clock_dot = 1")
}
fortran t = model.Trans {
    # a transition only triggers after theta seconds
    t.addGuardLabel("&&","clock > theta")
    # reset a clock after each transition
    t.addResetLabel("clock = 0")
} # end of the transformation
```

Fig. 9. The dwell-time resiliency pattern for the SMIB system.

Model repair for the SMIB system. A potential strategy to mitigate a sliding-mode attack is to increase the minimum switching time of the circuit breakers. Indeed, the designer can repair the original model by including a minimum dwell time in each mode of the system to prevent rapid switching. Figure 9 shows a resiliency pattern written as a HATL script that introduces the *clock* variable as a timer, and the switching time relies on the value of θ.

The model transformation of REAFFIRM takes the dwell-time pattern shown in Figure 9, and then convert the model to a new version that integrates the pattern with the unknown parameter θ. Then, the model synthesis of REAFFIRM calls Breach to search for the best (i.e., minimum) value of θ over and the range of $[0, 0.3]$ that ensures the final model satisfies φ_{SMIB} (with $T = 10$ seconds) under the sliding-mode attack. The tool returns the best value of θ as 0.12. Figure 8 shows the unstable behavior of the original model and the stable behavior of the repaired model under the sliding-model attack, respectively, where the red box defines the stable (safe) operation region of the SMIB system that can be formalized by the STL formula φ_{SMIB}.

C. Waypoint Tracking System

Next, we study the waypoint tracking system (WT) [28], which is a typical example of Simplex Architecture [13] that uses a safety controller to steer the system to a safe state. The WT system is briefly discussed here, with more details in [28]. The controllers of the WT system drive an autonomous vehicle following a predefined sequence of waypoints and keep it operating within a safe region. The vehicle motion is governed by the non linear equations: $\dot{x} = v\cos(\theta), \dot{y} = v\sin(\theta)$, where (x, y), v and θ are the position, velocity and heading angle, respectively. The WT system has two controllers: 1) a *complex controller* which captures all the possible behaviors within the physical limits of the actuator, and 2) a *safety controller* which slows down and stops the vehicle as fast as possible. The key challenge is to design a *decision module* which can switch from the complex controller to the safety controller whenever the system likely evolves toward an unsafe region.

Traditionally approaches such as Lyapunov-function-based techniques [29], [30] or reachability-based analysis [28], [31] have been applied to determine a switching condition between the two controllers of a simplex model. However, these approaches require the knowledge of the system dynamics and may have expensive computational cost. In this paper, we use the understanding of a safety specification to synthesize a switching condition for a decision module.

Original SLSF model. We build the SLSF model of the WT system based on the corresponding hybrid automata of the plant, complex controller and safety controller of the system presented in [28]. The safe operation region of the WT system, which has an ellipse form can be defined by the following STL formula,

$$\varphi_{WT} = \square_{[0,T]}\left(\frac{(x+5)^2}{900} + \frac{(y+10)^2}{400} \leq 1\right). \tag{3}$$

For our analysis, we assume that the original model does not contain a decision module and a safety controller, and the complex controller of the original model fails to keep the vehicle operating within the safe region defined by φ_{WT}.

Model repair for the WT system. To repair the WT model, we need to add a given safety controller and determine a transition from the complex controller to the safety controller. Based on the safety property of the WT system, that transition can be specified as $\frac{(x+5)^2}{900} + \frac{(y+10)^2}{400} > \theta$, where $\theta \in [0, 1]$. The resiliency pattern of the WT system is given in Figure 10. We note that this pattern is generic; that is, it can be applied uniformly to any simplex model with a given safety property. For the repaired WT model, without being excessively conservative, we aim to find the largest value of θ so that φ_{WT} always holds. Figure 11 shows a violation where the behavior of the original model evolves beyond the safe operation boundary, and a safe behavior of the repaired model with the safe switching boundary, $\theta = 0.625$, synthesized by REAFFIRM.

```
# start a transformation
model.addParam("theta")
cc = model.getMode("complex controller")
sc = model.getMode("safety controller")
# add a transition based on a safety requirement
model.addTransition(cc,sc,"(x+5)^2/900 + (y+10)^2/400 > θ")
# end of the transformation
```

Fig. 10. The resiliency pattern for the WT system.

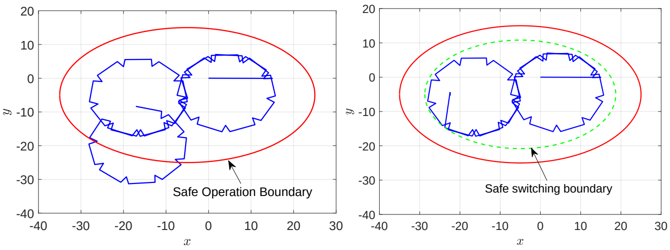


Fig. 11. Left: a violating trajectory of the original system. Right: a safe trajectory of the repaired model with a safe switching boundary.

D. Missile Guidance System

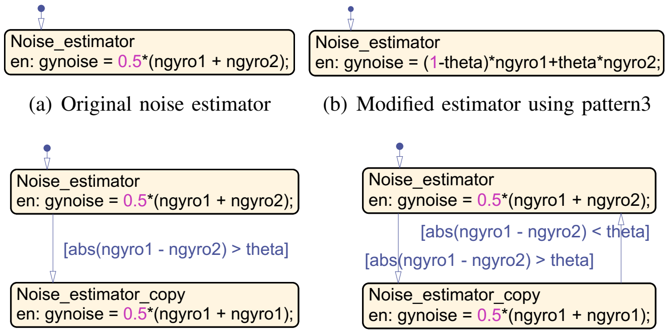
Original SLSF model. We consider the example of the missile guidance system (MG) provided by Mathworks, which is a good representative of a practical MG system. The original SLSF model has more than 300 blocks. The details of the model can be found at https://www.mathworks.com/help/simulink/examples/designing-a-guidance-system-in-MATLAB-and-simulink.html. In our study, we make a slight modification in the Sensors of the Airframe & Autopilot subcomponent of the original MG model. In the modified version, we model the missile body rate measurements as an array of two different gyroscopes, and the body rate estimation is obtained by using the average of the measurements obtained from the two gyroscopes. Such modification is reasonable as a practical MG system usually uses an array of gyroscopes to estimate the body rate of a missile. The correctness requirement of the model is that the missile will eventually approach the target where their distance is less than 10. This requirement can be formulated as an STL formula

$$\varphi_{MG} = \lozenge_{[0,T]} range[t] < 10. \tag{4}$$

In the original setting, the MG model satisfies the STL requirement and the noisy levels of two gyroscopes are assumed as $|n_{gyro1}| \leq 0.05$ and $|n_{gyro2}| \leq 0.05$, respectively.

Gyroscope sensor attack. The principle of a spoofing attack on the gyroscopes of the MG system is similar to the GPS spoofing attack of the ACC system. In this case, we omit the original assumption $|n_{gyro2}| \leq 0.05$, and employ the new assumption as $|n_{gyro2}| \leq 1$. As a result, the MG model no longer satisfies the STL requirement under this assumption.

Model repair for the MG system under gyroscope sensor attack. To repair the MG model under the gyroscope spoofing attack, we can reuse the three different patterns used to fix


Noise_estimator
en: gynoise = 0.5*(ngyro1 + ngyro2);
Noise_estimator
en: gynoise = (1-theta)*ngyro1+theta*ngyro2;
Noise_estimator
en: gynoise = 0.5*(ngyro1 + ngyro2);
[abs(ngyro1 - ngyro2) > theta]
Noise_estimator_copy
en: gynoise = 0.5*(ngyro1 + ngyro1);
Noise_estimator
en: gynoise = 0.5*(ngyro1 + ngyro2);
[abs(ngyro1 - ngyro2) < theta]
[abs(ngyro1 - ngyro2) > theta]
Noise_estimator_copy
en: gynoise = 0.5*(ngyro1 + ngyro1);

(a) Original noise estimator

(b) Modified estimator using pattern3

(c) Modified estimator using pattern1

(d) Modified estimator using pattern2

Fig. 12. The original and modified noise estimators.

the ACC model under GPS spoofing attack. Figure 12 shows the original gyroscope noise estimator which is vulnerable to the spoofing attack and three different repaired versions generated using the three resiliency patterns, respectively. For each resiliency pattern, the synthesized values of θ and the performance of our Reaffirm software is reported in Table I.

Angular noise injection attack. The original design of MG system has an assumption that the angular noise in the Tracker and Sight-line Rate Estimator components is negligible (i.e., set to 0). However, a practical missile guidance system often has an angular noise which may cause a significant impact on guidance performance. Angular noise can be considered as a combination of thermal noise, glint effect in radar seeker receiver and the external noise generated by a stand-off jammer or due to the change of the surrounding environment [32]. To evaluate the MG system under angular noise injection attack, we omit the original assumption that the angular noise level (denoted as $agun$) equal to 0 and employ the new assumption that $|agun| \leq 0.0875$. Under the new assumption, the MG model does not satisfy φ_{MG}.

Model repair for the MG system under angular noise injection attack. To repair the MG model under the angular noise injection attack, we do not need to perform a model transformation. Instead, a potential solution is to modify the tracking or stabilization loop parameters and gains of the Seeker/Tracker subsystem to compensate for the angular tracking error caused by the angular noise. The primary analysis question is a) which parameters/gains need to be tuned and b) what are their corresponding values that can mitigate the angular noise injection attack. In our approach, we answer the first part of the question by applying the Morris's elementary effects screening method [33], which is the most well-established global sensitivity analysis approach to identify the important factors that may have substantial effects on the satisfaction value of the correctness requirement. Then, we address the second part of the question by using Breach to synthesize the control gains and parameters values that can provide the maximum robustness satisfaction w.r.t the requirement φ_{MG}.

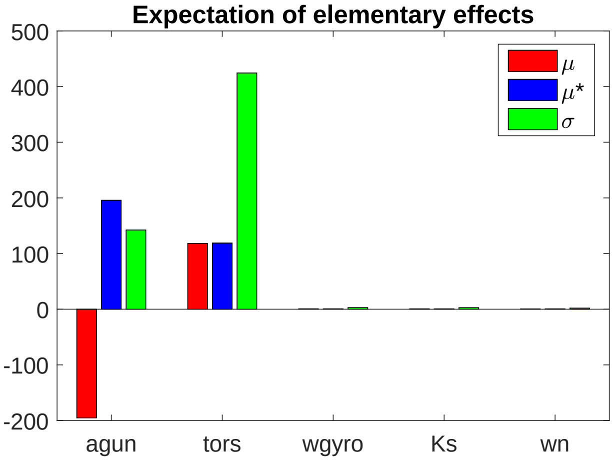


Fig. 13. The elementary effects of five different factors of the Seeker/Tracker of the MG system w.r.t the satisfaction value of φ_{MG}.

Sensitivity analysis. We apply Morris method to screen the elementary effects of five different factors of the Seeker/Tracker subsystem including a) $agun$: angular noise, b) $tors$: tracking loop time constant, c) $wgyro$: rate gyro bandwidth, d) Ks: rate loop bandwidth, and e) wn: estimator bandwidth to the satisfaction value of φ_{MG}. In the original design, these parameters are chosen as $agun = 0$, $tors = 0.05$, $wgyro = 200\pi$, $Ks = 40\pi$, and $wn = 7$, respectively. Figure 13 shows the elementary effects of five different factors on the satisfaction value of φ_{MG} under the assumption that $agun \in [0, 0.0875]$, $tors \in [0.05, 0.25]$, $wgyro \in [180\pi, 220\pi]$, $Ks \in [38\pi, 42\pi]$, and $wn \in [6.95, 7.05]$.

In Figure 13, μ and σ denote the mean and the standard deviation of the elementary effects, and $\mu^\star$ represents the mean of the absolute values of the elementary effects of the input factors, respectively. We can observe that three factors $wgyro$, Ks, and wn have negligible impacts on the satisfaction value while the angular noise and the tracking loop time have strong effects. However, the average effect of $agun$ is high and negative that means the MG system very sensitive to the angular noise injection attack, and it will violate φ_{MG} under the assumption $|agun| \leq 0.0875$. In contrast, the values of μ and $\mu^\star$ of $tors$ are equal and have the same sign, which indicates that increasing the value of $tors$ yields a positive effect on the satisfaction value of φ_{MG}. Therefore, to mitigate the angular noise injection attack, a potential fix is increasing the tracking loop time.

Max-satisfaction. The model synthesizer of REAFFIRM built on top of Breach supports an efficient optimization approach to address the max-satisfaction problem, i.e., finding the parameters values that make the system satisfy the specification as robust as possible. In the case of the angular noise injection attack, we want to find the best value of $tors \in [0.05, 0.25]$ that makes the MG system continue to satisfy φ_{MG} under the angular noise attack. Here, the tool finds the best value of $tors$ as 0.23421, as shown in Table I.

VI. Related Work

Model-based design of resilient CPSs. Examples of model-based approaches to ensure resiliency include the approach proposed in [34] that can be used to design a resilient CPS through co-simulation of discrete-event models, a modeling and simulation integration platform for secure and resilient CPS based on attacker-defender games [35] with the corresponding testbed [36], the resilience profiling of CPSs presented in [37], and the recent works of the design, implementation, and monitor of attack-resilient CPSs introduced in [38], [39]. Although these approaches can leverage the modeling and testing for a resilient CPS, they do not offer a model repair mechanism or a generic approach to design a resiliency pattern when vulnerabilities are discovered. Our proposed method is complementary to these efforts as we provide a generic, programmable way for a designer to specify a potential edit that can effectively repair the model for improving resiliency.

Formal analysis of hybrid systems. Our approach utilizes Breach to synthesize an SLSF model due to its advantages in performing falsification, systematic testing and parameter synthesis for hybrid systems. However, Breach cannot give a guarantee that the system satisfies the correctness specification. In certain cases, tools for verification of hybrid systems based on computing the set of reachable states can be used to get such a guarantee for the repaired model: examples of such tools are d/dt [40], SpaceEx [41], Flow*[42], and dReach[43].We choose Breach as it is more scalable.

Model transformation languages of hybrid systems. In the context of the model transformation, GREAT is a metamodel-based graph transformation language used to perform different transformations on domain-specific models [44], [45]. GREAT has been used to translate SLSF models to Hybrid Systems Interchange Format (HSIF) [46]. Such a translation scheme is accomplished by executing a sequence of translation rules described using UML Class Diagram in a specific order. Other approaches that also perform a translation from Simulink diagrams to hybrid systems formalisms such as Timed Interval Calculus [47], Hybrid Communicating Sequential Processes [48], Lustre [49], and SpaceEx [19]. HYST [50] is a conversion tool for hybrid automata which allows the same model to be analyzed simultaneously in several hybrid systems analysis tools. However, the problem of designing a scripting language to facilitate transforming models of hybrid systems has not been addressed before.

VII. Conclusion and Future Works

We have presented a new methodology, along with the software REAFFIRM that can effectively assist a designer to repair CPS models under unanticipated attacks automatically. The model transformation tool takes a resiliency pattern specified in the transformation language HATL and generates a new model including unknown parameters whose values can be determined by the synthesizer tool such that the safety requirement is satisfied. We demonstrated the applicability of REAFFIRM by using the software to efficiently repair the CPS models of four case studies under different attack scenarios.

We plan to extend REAFFIRM in several directions. First, we intend to perform a systematic classification of common attacks of various types of CPS based on the work presented in [51] and then develop an extensible library of resiliency patterns that encapsulates general mitigation strategies to repair CPS models under these common attacks. Second, we will leverage REAFFIRM to automatically search through the space of resiliency patterns to solve the model synthesis problem. Third, we plan to consider more complex safety and security specifications of CPS that can be specified using signal temporal logic (STL) for hyperproperties (HyperSTL) [52]. Besides using Breach, we also want to incorporate various verification tools such as Flow* and dReach into REAFFIRM to verify the repaired models formally.

References

[1] J. Wan, A. Canedo, and M. A. Al Faruque, "Security-aware functional modeling of cyber-physical systems," in *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE, 2015, pp. 1–4.

[2] A. Wasicek, P. Derler, and E. A. Lee, "Aspect-oriented modeling of attacks in automotive cyber-physical systems," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*. IEEE, 2014, pp. 1–6.

[3] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Moderator-Ravi, "Security as a new dimension in embedded system design," in *Proceedings of the 41st annual Design Automation Conference*. ACM, 2004, pp. 753–760.

[4] M. Al Faruque, F. Regazzoni, and M. Pajic, "Design methodologies for securing cyber-physical systems," in *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2015, pp. 30–36.

[5] T. T. Gamage, B. M. McMillin, and T. P. Roth, "Enforcing information flow security properties in cyber-physical systems: A generalized framework based on compensation," in *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*. IEEE, 2010, pp. 158–163.

[6] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava, "Non-invasive spoofing attacks for anti-lock braking systems," in *Proceedings of the 15th International Conference on Cryptographic Hardware and Embedded Systems*, ser. CHES'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 55–72.

[7] C. Li, A. Raghunathan, and N. K. Jha, "Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system," in *e-Health Networking Applications and Services (Healthcom), 2011 13th IEEE International Conference on*. IEEE, 2011, pp. 150–156.

[8] S. Liu, X. Feng, D. Kundur, T. Zourntos, and K. Butler-Purry, "A class of cyber-physical switching attacks for power system disruption," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*. ACM, 2011, p. 16.

[9] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[10] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical computer science*, vol. 138, no. 1, pp. 3–34, 1995.

[11] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Computer Aided Verification*. Springer, 2010, pp. 167–170.

[12] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[13] L. Sha, "Using simplicity to control complexity," *IEEE Software*, no. 4, pp. 20–28, 2001.

[14] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful gps spoofing attacks," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 75–86.

[15] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[16] S. Bak, O. A. Beg, S. Bogomolov, T. T. Johnson, L. V. Nguyen, and C. Schilling, "Hybrid automata: from verification to implementation," *International Journal on Software Tools for Technology Transfer*, pp. 1–18, 2017.

[17] R. Alur, A. Kanade, S. Ramesh, and K. Shashidhar, "Symbolic analysis for improving simulation coverage of simulink/stateflow models," in *Proceedings of the 8th ACM international conference on Embedded software*. ACM, 2008, pp. 89–98.

[18] K. Manamcheri, S. Mitra, S. Bak, and M. Caccamo, "A step towards verification and synthesis from simulink/stateflow models," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*. ACM, 2011, pp. 317–318.

[19] S. Minopoli and G. Frehse, "Sl2sx translator: from simulink to spaceex models," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM, 2016, pp. 93–98.

[20] J. Kapinski, J. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-guided approaches for verification of automotive powertrain control systems," in *American Control Conference (ACC), 2015*. IEEE, 2015, pp. 4086–4095.

[21] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*. ACM, 1995, pp. 373–382.

[22] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 34, no. 11, pp. 1704–1717, 2015.

[23] P. W. Sauer and M. Pai, "Power system dynamics and stability," *Urbana*, 1998.

[24] A. K. Farraj, E. M. Hammad, D. Kundur, and K. L. Butler-Purry, "Practical limitations of sliding-mode switching attacks on smart grid systems," in *PES General Meeting— Conference & Exposition, 2014 IEEE*. IEEE, 2014, pp. 1–5.

[25] R. A. DeCarlo, S. H. Zak, and G. P. Matthews, "Variable structure control of nonlinear multivariable systems: a tutorial," *Proceedings of the IEEE*, vol. 76, no. 3, pp. 212–232, 1988.

[26] S. Liu, B. Chen, T. Zourntos, D. Kundur, and K. Butler-Purry, "A coordinated multi-switch attack for cascading failures in smart grid," *IEEE Transactions on Smart Grid*, vol. 5, no. 3, pp. 1183–1195, 2014.

[27] A. Sabanovic, L. M. Fridman, S. Spurgeon, and S. K. Spurgeon, *Variable structure systems: from principles to implementation*. IET, 2004, vol. 66.

[28] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*. IEEE, 2011, pp. 3–12.

[29] T. Johnson, "Stability analysis of simplex architecture controlled inverted pendulum," *Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign., http://www. academia. edu/276649*, 2008.

[30] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2009, pp. 99–107.

[31] T. T. Johnson, S. Bak, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 2, p. 26, 2016.

[32] Q.-l. XIA, Y.-y. LIU, Z.-k. QI, and T. GUO, "Study of proportional navigation guidance error caused by angular noise and glint effect [j]," *Systems Engineering and Electronics*, vol. 8, 2008.

[33] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.

[34] J. Fitzgerald, K. Pierce, and C. Gamble, "A rigorous approach to the design of resilient cyber-physical systems through co-simulation," in *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*. IEEE, 2012, pp. 1–6.

[35] X. Koutsoukos, G. Karsai, A. Laszka, H. Neema, B. Potteiger, P. Volgyesi, Y. Vorobeychik, and J. Sztipanovits, "Sure: A modeling and simulation integration platform for evaluation of secure and resilient cyber–physical systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 93–112, 2018.

[36] H. Neema, B. Potteiger, X. Koutsoukos, G. Karsai, P. Volgyesi, and J. Sztipanovits, "Integrated simulation testbed for security and resilience of cps," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 2018, pp. 368–374.

[37] M. Jackson and J. Fitzgerald, "Resilience profiling in the model-based design of cyber-physical systems," in *14th Overture Workshop: Towards Analytical Tool Chains, Technical Report ECE-TR-28*, 2016, pp. 1–15.

[38] J. Weimer, R. Ivanov, S. Chen, A. Roederer, O. Sokolsky, and I. Lee, "Parameter-invariant monitor design for cyber–physical systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 71–92, 2018.

[39] M. Pajic, J. Weimer, N. Bezzo, O. Sokolsky, G. J. Pappas, and I. Lee, "Design and implementation of attack-resilient cyberphysical systems: With a focus on attack-resilient state estimators," *IEEE Control Systems*, vol. 37, no. 2, pp. 66–81, 2017.

[40] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 365–370.

[41] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Computer Aided Verification (CAV)*, ser. LNCS. Springer, 2011.

[42] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 258–263.

[43] S. Kong, S. Gao, W. Chen, and E. Clarke, "dreach: δ-reachability analysis for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 200–205.

[44] A. Agrawal, G. Karsai, and F. Shi, "Graph transformations on domain-specific models," *Journal on Software and Systems Modeling*, vol. 37, pp. 1–43, 2003.

[45] A. Agrawal, G. Karsai, and Á. Lédeczi, "An end-to-end domain-driven software development framework," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 2003, pp. 8–15.

[46] A. Agrawal, G. Simon, and G. Karsai, "Semantic translation of simulink/stateflow models to hybrid automata using graph transformations," *Electronic Notes in Theoretical Computer Science*, vol. 109, pp. 43–56, 2004.

[47] C. Chen, J. S. Dong, and J. Sun, "A formal framework for modeling and validating simulink diagrams," *Formal Aspects of Computing*, vol. 21, no. 5, pp. 451–483, 2009.

[48] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou, "A calculus for hybrid csp," in *Asian Symposium on Programming Languages and Systems*. Springer, 2010, pp. 1–15.

[49] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic, "Translating discrete-time simulink to lustre," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 4, pp. 779–818, 2005.

[50] S. Bak, S. Bogomolov, and T. T. Johnson, "Hyst: a source transformation and translation tool for hybrid automaton models," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 2015, pp. 128–133.

[51] A. Humayed, J. Lin, F. Li, and B. Luo, "Cyber-physical systems security—a survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1802–1831, 2017.

[52] L. V. Nguyen, J. Kapinski, X. Jin, J. V. Deshmukh, and T. T. Johnson, "Hyperproperties of real-valued signals," in *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, 2017, pp. 104–113.