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

Luan Viet Nguyen, Gautam Mohan, James Weimer, Oleg Sokolsky, Insup Lee, Rajeev Alur
Department of Computer and Information Science
University of Pennsylvania
{luanvn, gmohan1, weimerj, sokolsky, lee, alur}@seas.upenn.edu

ABSTRACT

Model-based design offers a promising approach for assisting developers to build reliable and secure cyber-physical systems (CPSs) 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 mechanism 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 toolkit called REAFFIRM to facilitate the model-based repair for improving the resiliency of CPSs. REAFFIRM takes the inputs including 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 then outputs a repaired model which satisfies the requirement. The overall structure of REAFFIRM contains two main modules, a model transformation, and a model synthesizer built on top of the falsification tool Breach. 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 repaired models for an adaptive cruise control (ACC) system under GPS sensor spoofing attacks, and for a single-machine infinite-bus (SMIB) system under a sliding-mode switching attack.

ACM Reference format:
Luan Viet Nguyen, Gautam Mohan, James Weimer, Oleg Sokolsky, Insup Lee, Rajeev Alur
. 2019. REAFFIRM: Model-Based Repair of Hybrid Systems for Improving Resiliency. In *Proceedings of ICCPS Conference, Montreal, Canada, April 2019 (Conference'19),* 10 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

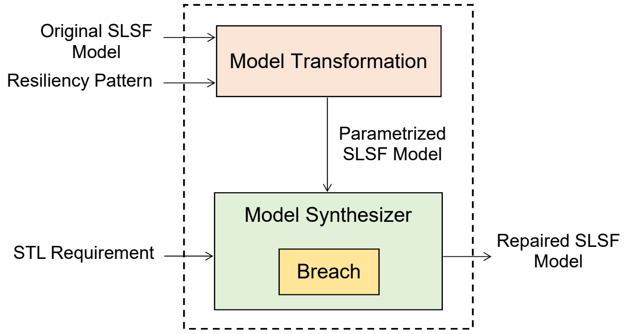


Figure 1: REAFFIRM overview.

1 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 [4, 19, 26, 44, 45]. 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 [4, 41], and compromised the insulin delivery service of a diabetes therapy system [30]. Alternatively, attackers can gain access to communication channels to either manipulate the switching behavior of a smart power grid [33] or disable the brake system of a modern vehicle [28]. Generally, constructing a behavioral model at design time that offers resiliency for all kinds of attacks and failures is notoriously difficult.

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 [5]. 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 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.

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center Pacific (SSC Pacific) under Contract No. N6600118C4007.

Conference'19, Montreal, Canada
© 2019 ACM. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

In this paper, we propose a new methodology and an associated toolkit, which we call 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 mains modules including a *model transformation* and a *model synthesizer* built on top of the falsification tool Breach [13]. 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) [34] formula, and then utilizes the falsification tool Breach to synthesize the repaired SLSF model that satisfies the safety requirement.

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 [5]. Examples of edits to a model include creating new modes of operations, duplicating modes, adding transitions, modifying switching conditions, and substitution of state variables in flow equations. The proposed language allows the designer to write a resiliency pattern in a generic manner, programmatically modify the initial design without knowing the internal structures of a system. The interpreter of HATL is implemented in Python with the backend is extensible for the translations to different modeling frameworks of hybrid systems. The current implementation of HATL supports the translation of a HATL script to an equivalent Matlab script that can perform the model transformation for Stateflow models. To the best of our knowledge, this transformation language is the first effort to design a programmable pattern to repair CPS models for improving resiliency.

For evaluation, we apply REAFFIRM to automatically synthesize the repaired models for two proof-of-concept case studies in the domains of automotive control and smart power systems. 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. The synthesizer tool of 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 is adding a dwell-time variable in each mode of the model, and the minimal dwell-time can be synthesized automatically by REAFFIRM.

In summary, 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 toolkit, which integrates the model transformation and the model synthesis tools to automatically repair CPS models,
(4) the applicability of our approach on two proof-of-concept case studies where the CPS models can be automatically repaired to mitigate practical attacks.

The remainder of the paper is organized as follows. Section 2 presents an overview of our proposed methodology through a simplified example of the ACC system, and introduces the architecture of REAFFIRM. Section 3 describes our model transformation language used to design a resiliency pattern for hybrid systems. Section 4 presents the model synthesizer of REAFFIRM. Section 5 presents two case studies that illustrate the capability of REAFFIRM in automatically repairing the original models of a) the ACC system under a GPS sensor spoofing attack and b) the smart power grid system under a sliding-mode attack. Section 6 reviews the related works to REAFFIRM. Section 7 concludes the paper and presents future research directions for the proposed work.

2 OVERVIEW OF THE METHODOLOGY

In this section, we will explain our methodology through a simplified example of the ACC system. This system has been designed to satisfy safety requirements pertaining to vehicle spacing that varies with vehicle speed. 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[1]. Then, we present how REAFFIRM can automatically perform a model transformation and synthesis to construct a new ACC model with resiliency.

2.1 A Simplified Example of ACC System

For simplicity, we assume that the designer initially model 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*. 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 states in which 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, $d_{rad} = d + n_{rad}$. 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

[1] 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 in the ensuing discussion of the paper.

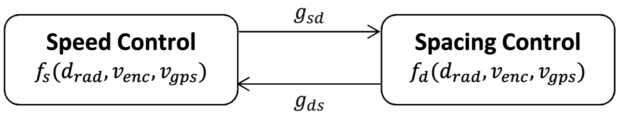

Speed Control
$f_s(d_{rad}, v_{enc}, v_{gps})$
g_{sd}
Spacing Control
$f_d(d_{rad}, v_{enc}, v_{gps})$
g_{ds}

Figure 2: An original ACC model.

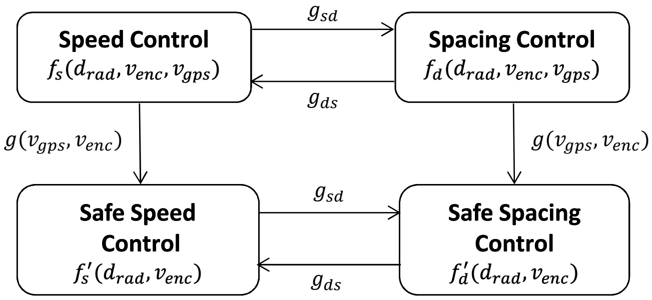


Figure 3: A repaired ACC model without a reference to GPS sensor under spoofing attacks.

control by executing the transition g_{ds}. The *safety specification* of the system is specified 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 5.

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* [25, 42]. 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 with a resiliency pattern.

Example of resiliency pattern: ignoring GPS measurement. Since the ACC system has redundancy in the sensory information of its estimated velocity, 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 resiliency pattern 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 legacy speed and spacing modes of the original model to the new copied speed and spacing modes of the copy that uses only the wheel encoder as a velocity measurement source. 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}.

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 must 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.

2.2 REAFFIRM Toolkit

Our REAFFIRM prototype for the model-based design and repair with resiliency is built in Matlab and consists of two main modules, corresponding to a *model transformation* and a *model synthesizer*. 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 mentioned previously, the inputs of REAFFIRM are the SLSF model of the original ACC system 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

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

The model transformation tool of REAFFIRM takes the initial SLSF model and the resiliency pattern, and then generates the new SLSF model of the ACC system that contains a parameter θ captured 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 performs a parameter 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 [13] to synthesize values for the desired parameters. If the synthesizer can find the best value of θ over the given range, REAFFIRM outputs a competed SLSF model which satisfies φ_{ACC} under the GPS attacks. Otherwise, the tool will suggest a designer to either search over different parameter ranges or try different resiliency patterns to repair the ACC model.

REAFFIRM was tested using Matlab 2017a and Matlab 2018a 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.

3 MODEL TRANSFORMATION

3.1 Representation of Hybrid System

Hybrid automata [5] 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 intervals of real-time [5][2]. 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 $\mathcal{X} \cup \mathcal{P}$ 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 $\mathcal{X}$ 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 reader to [5] 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.

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 MathWorks Simulink. Continuous-time Stateflow chart[3] supplies methods for engineers to quickly model as well as efficiently refine, test, and generate code for hybrid automata. The syntactic components of a continuous-time Stateflow chart are described similar to a hybrid automaton where 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*. A Stateflow diagram 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 [8] 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 [6, 35, 36].

```
model = getModelByName("ACC") # retrieve the ACC model
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
```

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

3.2 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

[2] In this paper, we only consider *deterministic* systems, where the system produces the same output for a given input.Note the contrast with *stochastic* systems in which one or more elements of the system have randomness associated with them; for example, the value of some system parameter may be extracted from a probability distribution. As a result, the stochastic system may yield different outputs for a given input.

[3] In this paper, we focus only on continuous-time Stateflow diagram that does not include hierarchical states.

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.

3.3 Implementation

Our current implementation dynamically interprets HATL scripts in Python and translates them into Stateflow 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 5. There is room for future improvement, such as adding language constructs like object hierarchy or type checking. Expanding HATL's backend to include more modeling frameworks besides SLSF would be useful as well.

4 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 already determined after the model transformation, the model synthesis problem then reduces to the *parameter synthesis problem*. Given a safety specification φ, let $\mathcal{P}_s$ be a set of new parameters of the transformed model $\tilde{\mathcal{H}}$, find the best instance values of $\mathcal{P}_s$ over its domain $\bar{\mathcal{P}_s}$ 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.

4.1 Overview of Breach

In our proposed methodology, 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 [13] 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 [24]. 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 proved to be *undecidable* [20]. 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.

4.2 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 include 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, their certain ranges of 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 parameter domain $\bar{\mathcal{P}_s}$ to the new domain $\bar{\mathcal{P}_s}' \subset \bar{\mathcal{P}_s}$, 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 range $\bar{\mathcal{P}_s}$ 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 STM query and then is determined using an STM solver. However, the result may be *undecidable* due to the undecidability of STL [23]. 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.

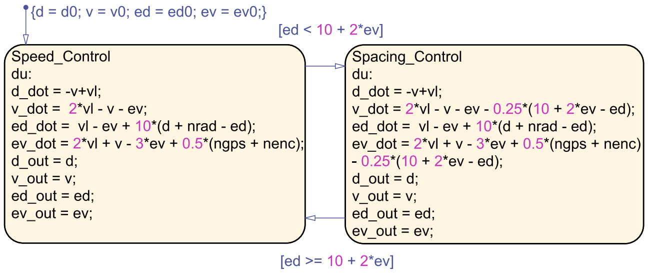
5 MODEL REPAIR FOR RESILIENCY

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. The overall performance of REAFFIRM in repairing the initial model of two case studies to mitigate their corresponding attacks is summarized in Table 1. Next, we will describe two case studies in more details.

5.1 Adaptive Cruise Control System

Original SLSF model. We previously introduced the simplified example of the ACC system in Section 2 to illustrate our proposed 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 travel 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 $|n_{gps}| \leq 0.05$.

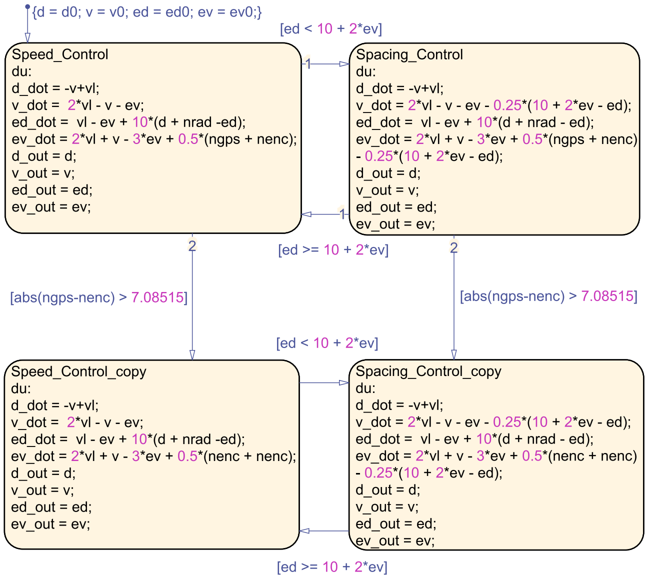
GPS sensor attack. To perform a spoofing attack on the GPS sensor of the ACC model, we continuously inject false data to manipulate its measurement value. In this case, we omit the original assumption $|n_{gps}| \leq 0.05$, and employ the new assumption as $|n_{gps}| \leq 50$. Using the input generator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



```
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;
```

Figure 5: The original SLSF model of the ACC system.


{d = d0; v = v0; ed = ed0; ev = ev0;}
[ed < 10 + 2*ev]
1
1
[ed >= 10 + 2*ev]
2
2
[abs(ngps-nenc) > 7.08515]
[abs(ngps-nenc) > 7.08515]
[ed < 10 + 2*ev]
[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;
```

```
Speed_Control_copy
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*(nenc + nenc);
d_out = d;
v_out = v;
ed_out = ed;
ev_out = ev;
```

```
Spacing_Control_copy
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*(nenc + nenc)
- 0.25*(10 + 2*ev - ed);
d_out = d;
v_out = v;
ed_out = ed;
ev_out = ev;
```

Figure 6: The repaired ACC model with a synthesized value of $\theta = 7.08515$.

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 2, 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. Figure 6 shows the completed model, where the switching condition is determined by synthesizing the value of θ over the range of [0, 50] using Breach. For the first pattern, the model transformation takes about 2 seconds, and the synthesis procedure takes approximately 88 seconds over 6 iterations of the falsification loop.

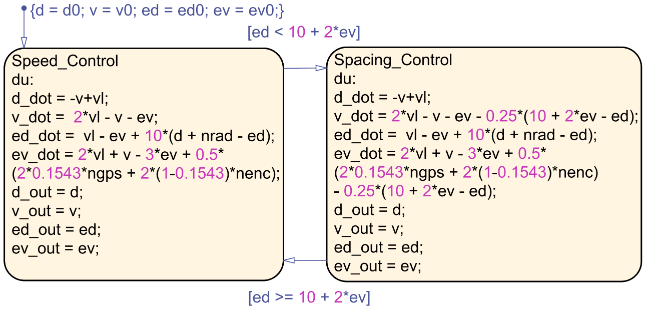
The *second resiliency pattern* for the ACC model is the extended version of the first one where it includes a switching-back condition from the copy to the original model when the GPS sensor attack is detected and mitigated. An example of such a switching-back condition is when the difference between the n_{enc} and n_{gps} are

Model	Attack	Resiliency Pattern		Unknown Condition	Parameter Range	Synthesized Value	Transformation Time	Synthesis Time
ACC	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 seconds	88 seconds
			Pattern 2			7.08515	2 seconds	88 seconds
			Pattern 3	Ratio of GPS/encoders measurements	$\theta \in [0.1, 0.9]$	0.1543	1.75 seconds	55.78 seconds
SMIB	Sliding-mode switching	Introduce a dwell-time to avoid rapid switching	Pattern dwell-time	Minimal dwell-time	$\theta \in [0, 0.3]$	0.12	2 seconds	45 seconds

Table 1: REAFFIRM performance results for the ACC and SMIB case studies.

```
model = getModelByName("ACC") # retrieve the ACC model
# start a transformation
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
```

Figure 7: The third resiliency pattern for the ACC system based on the linear combination of n_{enc} and n_{gps}.



```
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*
(2*0.1543*ngps + 2*(1-0.1543)*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*
(2*0.1543*ngps + 2*(1-0.1543)*nenc)
- 0.25*(10 + 2*ev - ed);
d_out = d;
v_out = v;
ed_out = ed;
ev_out = ev;
```

Figure 8: The repaired ACC model with a synthesized value of $\theta = 0.1543$ for the resiliency pattern shown in Figure 7.

getting smaller, i.e., $|n_{gps} - n_{enc}| < \theta - \epsilon$, where ϵ is a positive user-defined tolerant. 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 $\theta = 7.08515$.

Alternatively, the *third resiliency pattern*, which 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 synthesize the value of θ so that the safety property is satisfied. The transformation script of this resiliency pattern is given in Figure 7. 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 $\theta = 0.1543$ that appears in the repaired model shown in Figure 8. In this scenario, the model transformation takes about 1.75 seconds, and the synthesis procedure takes approximately 55.78 seconds over 5 iterations of the falsification loop. This result indicates that the third pattern can repair the model if the portion of the GPS measurement contributed to estimating the velocity is significantly smaller than that of the wheel encoders. However, if the GPS spoofing attack specified over a broader range (e.g., $|n_{enc}| \leq 100$), the pattern will fail to repair the model.

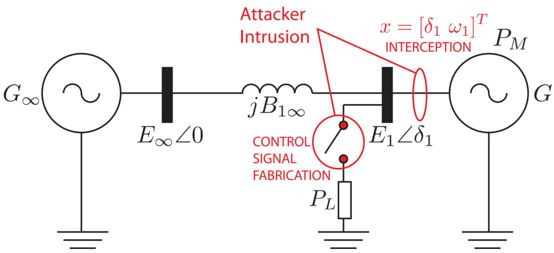


Figure 9: Single-machine infinite-bus system [15].

5.2 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 [40], can be represented as a single-machine infinite-bus (SMIB) system shown[4] in Figure 9. In this system, G_∞ and G_1 correspondingly represent the SMIB and local generators; B_∞ and B_1 denote the infinite and local bus, respectively; E_∞ is the infinite bus voltage; E_1 is the internal voltage of G_1; $B_{1\infty}$ is the transfer susceptance of the line between B_1 and B_∞; and P_M is the mechanical power of G_1. The local load P_L is connected or disconnected to the grid by changing a circuit breaker status. 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 G_1 respectively, and $x = [\delta_1, \omega_1]^T$ is the state vector of G_1. The stability (safety) property of the system can be specified as the following STL formula,

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

where T is a simulation duration.

[4] the figure is copied from [15].

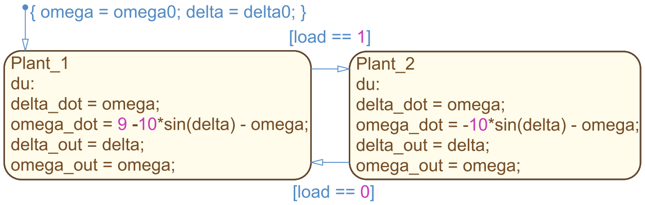


Figure 10: The Stateflow chart models the plant of the SMIB system.

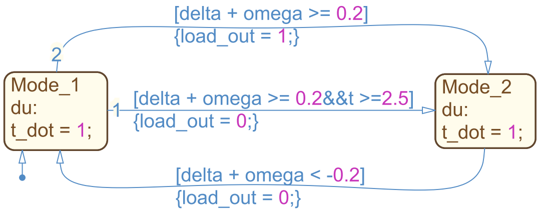


Figure 11: The Stateflow chart models the sliding-mode attack to the SMIB system.

Original SLSF Model. In this paper, we model the SMIB system as the SLSF model displayed in Figure 10. The model contains two operation modes whose nonlinear dynamics characterize the transient stability of the local generator G_1 presented in [15]. 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 $s(x) \in \mathbb{R}$ [12, 32]. An example of a sliding surface is $s(x) = 0$. When the system is confined on a sliding mode surface, its dynamics exhibit high-frequency oscillations behaviors, so-called a *chattering* phenomenon, which is well-known in the power system design [39]. At this moment, if an attacker conducts the fast switches 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 [32].

To perform the sliding-mode attack on the original SMIB model, we model the attack as the SLSF model shown in Figure 11. In this model, we assume that the attacker selects a sliding surface $s(x) = \delta_1 + \omega_1 = 0.2$, and the local variable t captures a simulation duration. We note that two transitions from the first mode to the second mode are executed with priorities such that the load is permanently disconnected at some instance where $t \geq 2.5$ seconds. More details of the stages to construct the sliding-mode attack can be found in [15]. Figure 14 illustrates the examples of stable (i.e., without an attack) and unstable (i.e., a counterexample appearing under the sliding-model attack) behaviors of the SMIB system returned by running the falsifier of Breach, respectively. The red box defines the stable (safe) operation region of the SMIB system that can be formalized by the STL formula φ_{SMIB}.

```
model = getModelByName("SMIB") # retrieve the plant model
# start a transformation
model.addParam("theta") # add a new parameter 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
```

Figure 12: A dwell-time resiliency pattern for the SMIB system.

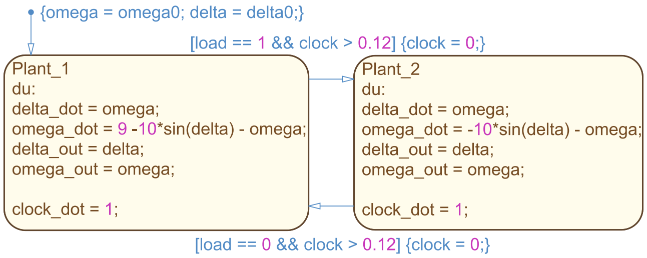

{omega = omega0; delta = delta0;}
[load == 1 && clock > 0.12] {clock = 0;}
Plant_1
du:
delta_dot = omega;
omega_dot = 9 -10*sin(delta) - omega;
delta_out = delta;
omega_out = omega;
clock_dot = 1;
Plant_2
du:
delta_dot = omega;
omega_dot = -10*sin(delta) - omega;
delta_out = delta;
omega_out = omega;
clock_dot = 1;
[load == 0 && clock > 0.12] {clock = 0;}

Figure 13: The repaired SMIB model with a synthesized dwell-time.

Model Repair for the SMIB system. As a sliding-mode attack is constructed based on switching back-and-forth the circuit breaker quickly to trap the system inside the sliding surface before guiding its state variables evolving outside the stability boundary, a potential strategy to mitigate such an 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 12 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 12, 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 final model, which is stable, and its simulation trajectories contain within a red box similar to the most left subfigure shown in Figure 14, is displayed in Figure 13, where the synthesized value of θ equals to 0.12. Overall, the model transformation takes about 2 seconds, and the synthesis procedure takes approximately 45 seconds over 8 iterations of the falsification loop.

6 RELATED WORKS

Model-based design of resilient CPSs. Examples of model-based approaches to ensure resiliency include the approach proposed in [16] that can be used to design a resilient CPS through co-simulation of discrete-event models, a modeling and simulation

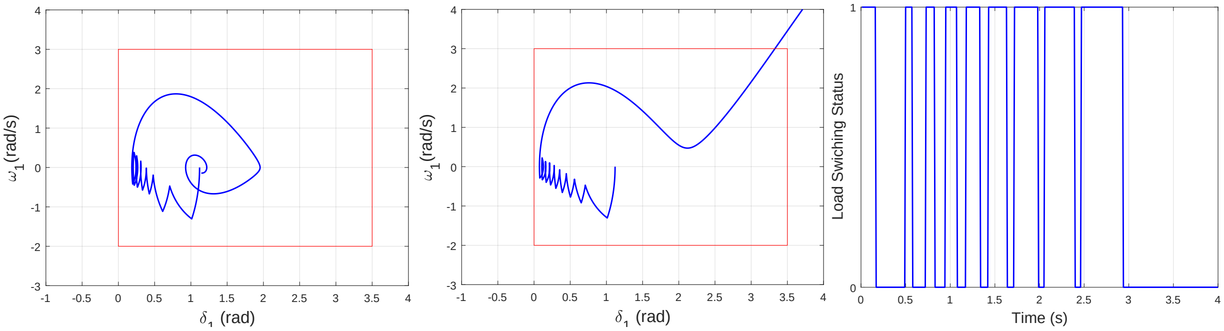


Figure 14: From left to right: 1) the stable system trajectory without an attack, 2) the counterexample represents the unstable system trajectory under the sliding-mode attack, and 3) the status of a circuit breaker during the attack, where 0 and 1 represent the disconnection and connection of the load P_L, respectively.

integration platform for secure and resilient CPS based on attacker-defender games proposed in [29] with the corresponding testbed introduced in [37], the resilience profiling of CPSs presented in [22], and the recent works of the design, implementation, and monitor of attack-resilient CPSs introduced in [38, 46]. Although these approaches can leverage the modeling and testing for a resilient CPS, they do not offer a model repair mechanism as well as a generic approach to design a resiliency pattern when vulnerabilities are discovered. Our proposed method is complementary to those works 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 formal proof of the system correctness. Depending on different types of hybrid systems, other automatic verification tools can be considered to perform a reachability analysis or formally prove whether a system satisfy a given safety property. For examples, d/dt [7] and SpaceEx [17] and are well-known verification tool for linear/affine hybrid systems; Flow*[11] and dReach[27] can be used to compute a reachable set of nonlinear hybrid automata; and C2E2 is a verification tool for Stateflow models [14]. However, scalability is the main factor that limits the applications of those verification approaches to the formal analysis of hybrid systems.

Model transformation languages of hybrid systems. In the context of the model transformation, GREAT is a metamodel-based graph transformation language that can be used to perform different transformations on domain-specific models [1, 2]. GREAT has been used to translate SLSF models to Hybrid Systems Interchange Format (HSIF) [3]. 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 [10], Hybrid Communicating Sequential Processes [31], Lustre [43], and SpaceEx [36]. HYST [9] is a conversion tool for hybrid automata which allows the same model to be analyzed simultaneously in several hybrid systems analysis tools. HYST takes a source input model in SpaceEx XML format [18], parses it to an intermediate representation, and then prints a resulting output to some desired formats. HYST can automatically translate hybrid automata to *trajectory-equivalent* SLSF models, which enables a *correct-by-construction* compositional design for CPS with embedding hybrid automata into large-scale SLSF models [8]. Unlike these approaches that focus on the translations between different hybrid system modeling frameworks, our goal is to provide a lightweight programmable model transformation language for hybrid systems in which a designer only needs to write a simple script to repair a CPS model.

7 CONCLUSION AND FUTURE WORK

In this paper, we have presented a new methodology and the toolkit REAFFIRM that 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 toolkit to efficiently repair CPS models under realistic attacks including the ACC models under the GPS sensor spoofing attack and the SMIB models under the sliding-model attack.

Future Work. We first plan to perform a systematic classification of common attacks of various types of CPSs based on the work presented in [21] and then develop an extensible library of resiliency patterns that encapsulates general mitigation strategies to repair CPS models under these common attacks. Beside using Breach, we also intend to extend the model synthesizer of REAFFIRM to incorporate various verification tools such as Flow* and dReach to verify the repaired model formally. From the application perspective of REAFFIRM, we are interested in investigating the SLSF model of a missile guidance system provided by Matlab[5], the attack scenarios that destabilizes or drives the system toward unsafe behaviors and the resiliency patterns that can be used to repair the SLSF model for improving resiliency.

[5] The model is available at https://www.mathworks.com/help/simulink/examples/designing-a-guidance-system-in-matlab-and-simulink.html.

REFERENCES

[1] Aditya Agrawal, Gabor Karsai, and Ákos Lédeczi. 2003. 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, 8–15.

[2] Aditya Agrawal, Gabor Karsai, and Feng Shi. 2003. Graph transformations on domain-specific models. *Journal on Software and Systems Modeling* 37 (2003), 1–43.

[3] Aditya Agrawal, Gyula Simon, and Gabor Karsai. 2004. Semantic translation of Simulink/Stateflow models to hybrid automata using graph transformations. *Electronic Notes in Theoretical Computer Science* 109 (2004), 43–56.

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

[5] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. 1995. The algorithmic analysis of hybrid systems. *Theoretical computer science* 138, 1 (1995), 3–34.

[6] Rajeev Alur, Aditya Kanade, S Ramesh, and KC Shashidhar. 2008. Symbolic analysis for improving simulation coverage of Simulink/Stateflow models. In *Proceedings of the 8th ACM international conference on Embedded software*. ACM, 89–98.

[7] Eugene Asarin, Thao Dang, and Oded Maler. 2002. The d/dt tool for verification of hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 365–370.

[8] Stanley Bak, Omar Ali Beg, Sergiy Bogomolov, Taylor T Johnson, Luan Viet Nguyen, and Christian Schilling. 2017. Hybrid automata: from verification to implementation. *International Journal on Software Tools for Technology Transfer* (2017), 1–18.

[9] Stanley Bak, Sergiy Bogomolov, and Taylor T Johnson. 2015. 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, 128–133.

[10] Chunqing Chen, Jin Song Dong, and Jun Sun. 2009. A formal framework for modeling and validating Simulink diagrams. *Formal Aspects of Computing* 21, 5 (2009), 451–483.

[11] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. 2013. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 258–263.

[12] Raymond A DeCarlo, Stanislaw H Zak, and Gregory P Matthews. 1988. Variable structure control of nonlinear multivariable systems: a tutorial. *Proc. IEEE* 76, 3 (1988), 212–232.

[13] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*. Springer, 167–170.

[14] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. 2015. C2E2: a verification tool for stateflow models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 68–82.

[15] Abdallah K Farraj, Eman M Hammad, Deepa Kundur, and Karen L Butler-Purry. 2014. Practical limitations of sliding-mode switching attacks on smart grid systems. In *PES General Meeting— Conference & Exposition, 2014 IEEE*. IEEE, 1–5.

[16] John Fitzgerald, Ken Pierce, and Carl Gamble. 2012. 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, 1–6.

[17] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *Computer Aided Verification (CAV) (LNCS)*. Springer.

[18] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 379–395.

[19] Thoshitha T Gamage, Bruce M McMillin, and Thomas P Roth. 2010. 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, 158–163.

[20] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. 1995. What’s decidable about hybrid automata?. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*. ACM, 373–382.

[21] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. 2017. Cyber-physical systems securityfi!?A survey. *IEEE Internet of Things Journal* 4, 6 (2017), 1802–1831.

[22] Mark Jackson and J Fitzgerald. [n. d.]. Resilience profiling in the model-based design of cyber-physical systems. In *14th Overture Workshop: Towards Analytical Tool Chains, Technical Report ECE-TR-28*. 1–15.

[23] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V Deshmukh, and Sanjit A Seshia. 2015. Mining requirements from closed-loop control models. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 34, 11 (2015), 1704–1717.

[24] James Kapinski, Jyotirmoy Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken Butts. 2015. Simulation-guided approaches for verification of automotive powertrain control systems. In *American Control Conference (ACC), 2015*. IEEE, 4086–4095.

[25] Andrew J Kerns, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. 2014. Unmanned aircraft capture and control via GPS spoofing. *Journal of Field Robotics* 31, 4 (2014), 617–636.

[26] Paul Kocher, Ruby Lee, Gary McGraw, Anand Raghunathan, and Srivaths Moderator-Ravi. 2004. Security as a new dimension in embedded system design. In *Proceedings of the 41st annual Design Automation Conference*. ACM, 753–760.

[27] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. 2015. dReach: δ-reachability analysis for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 200–205.

[28] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. 2010. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 447–462.

[29] Xenofon Koutsoukos, Gabor Karsai, Aron Laszka, Himanshu Neema, Bradley Potteiger, Peter Volgyesi, Yevgeniy Vorobeychik, and Janos Sztipanovits. 2018. SURE: A Modeling and Simulation Integration Platform for Evaluation of SecUre and REsilient Cyber–Physical Systems. *Proc. IEEE* 106, 1 (2018), 93–112.

[30] Chunxiao Li, Anand Raghunathan, and Niraj K Jha. 2011. 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, 150–156.

[31] Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou, and Liang Zou. 2010. A calculus for hybrid CSP. In *Asian Symposium on Programming Languages and Systems*. Springer, 1–15.

[32] Shan Liu, Bo Chen, Takis Zourntos, Deepa Kundur, and Karen Butler-Purry. 2014. A coordinated multi-switch attack for cascading failures in smart grid. *IEEE Transactions on Smart Grid* 5, 3 (2014), 1183–1195.

[33] Shan Liu, Xianyong Feng, Deepa Kundur, Takis Zourntos, and Karen Butler-Purry. 2011. 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, 16.

[34] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.

[35] Karthik Manamcheri, Sayan Mitra, Stanley Bak, and Marco Caccamo. 2011. A step towards verification and synthesis from simulink/stateflow models. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*. ACM, 317–318.

[36] Stefano Minopoli and Goran Frehse. 2016. SL2SX translator: from Simulink to SpaceEx models. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM, 93–98.

[37] Himanshu Neema, Bradley Potteiger, Xenofon Koutsoukos, Gabor Karsai, Peter Volgyesi, and Janos Sztipanovits. 2018. Integrated simulation testbed for security and resilience of CPS. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 368–374.

[38] Miroslav Pajic, James Weimer, Nicola Bezzo, Oleg Sokolsky, George J Pappas, and Insup Lee. 2017. Design and implementation of attack-resilient cyberphysical systems: With a focus on attack-resilient state estimators. *IEEE Control Systems* 37, 2 (2017), 66–81.

[39] Asif Sabanovic, Leonid M Fridman, Sarah Spurgeon, and Sarah K Spurgeon. 2004. *Variable structure systems: from principles to implementation*. Vol. 66. IET.

[40] Peter W Sauer and MA Pai. 1998. Power system dynamics and stability. *Urbana* (1998).

[41] Yasser Shoukry, Paul Martin, Paulo Tabuada, and Mani Srivastava. 2013. Non-invasive Spoofing Attacks for Anti-lock Braking Systems. In *Proceedings of the 15th International Conference on Cryptographic Hardware and Embedded Systems (CHES’13)*. Springer-Verlag, Berlin, Heidelberg, 55–72.

[42] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. 2011. On the requirements for successful GPS spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 75–86.

[43] Stavros Tripakis, Christos Sofronis, Paul Caspi, and Adrian Curic. 2005. Translating discrete-time Simulink to Lustre. *ACM Transactions on Embedded Computing Systems (TECS)* 4, 4 (2005), 779–818.

[44] Jiang Wan, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. 2015. Security-aware functional modeling of cyber-physical systems. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE, 1–4.

[45] Armin Wasicek, Patricia Derler, and Edward A Lee. 2014. Aspect-oriented modeling of attacks in automotive cyber-physical systems. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*. IEEE, 1–6.

[46] James Weimer, Radoslav Ivanov, Sanjian Chen, Alexander Roederer, Oleg Sokolsky, and Insup Lee. 2018. Parameter-Invariant Monitor Design for Cyber–Physical Systems. *Proc. IEEE* 106, 1 (2018), 71–92.