Detecting Security Leaks in Hybrid Systems with Information Flow Analysis

Luan Viet Nguyen
luanvn@seas.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania

Gautam Mohan
gmohan1@seas.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania

James Weimer
weimerj@seas.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania

Oleg Sokolsky
sokolsky@seas.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania

Insup Lee
lee@cis.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania

Rajeev Alur
alur@cis.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania

ABSTRACT

Information flow analysis is an effective way to check useful security properties, such as whether secret information can leak to adversaries. Despite being widely investigated in the realm of programming languages, information-flow-based security analysis has not been widely studied in the domain of cyber-physical systems (CPS). CPS provide interesting challenges to traditional type-based techniques, as they model mixed discrete-continuous behaviors and are usually expressed as a composition of state machines. In this paper, we propose a lightweight static analysis methodology that enables information security properties for CPS models. We introduce a set of security rules for hybrid automata that characterizes the property of non-interference. Based on those rules, we propose an algorithm that generates security constraints between each sub-component of hybrid automata, and then transforms these constraints into a directed dependency graph to search for non-interference violations. The proposed algorithm can be applied directly to parallel compositions of automata without resorting to model-flattening techniques. Our static checker works on hybrid systems modeled in Simulink/Stateflow format and decides whether or not the model satisfies non-interference given a user-provided security annotation for each variable. Moreover, our approach can also infer the security labels of variables, allowing a designer to verify the correctness of partial security annotations. We demonstrate the potential benefits of the proposed methodology on two case studies.

CCS CONCEPTS

• **Security and privacy → Formal methods and theory of security**; • **Theory of computation → Timed and hybrid models**; • **Computer systems organization → Embedded and cyber-physical systems**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MEMOCODE '19, October 9–11, 2019, La Jolla, CA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6997-8/19/10…$15.00
https://doi.org/10.1145/3359986.3361212

KEYWORDS

Information Flow Security, Static Analysis, Simulink/Stateflow Models.

ACM Reference Format:
Luan Viet Nguyen, Gautam Mohan, James Weimer, Oleg Sokolsky, Insup Lee, and Rajeev Alur. 2019. Detecting Security Leaks in Hybrid Systems with Information Flow Analysis. In *17th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE '19), October 9–11, 2019, La Jolla, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3359986.3361212

1 INTRODUCTION

Cyber-physical systems (CPS) are networked computing devices communicating with each other and interacting with the physical environment via sensors and actuators. CPS are characterized by both continuous and discrete dynamics, so they are often considered as hybrid systems. Hybrid systems are increasingly utilized in a variety of domains, modeling diverse systems such as smart power grids or autonomous vehicles, and even mission-critical military systems. The rapidly expanding field of CPS has precipitated a corresponding growth in security concerns [6, 19, 21]. Among of them, enforcing information flow security plays an important role to guarantee the safety and and reliability of CPS. Information flow properties such as non-interference [7], non-inference [13], and non-deducibility [18] prevent public users from inferring any high-level (secret) information by observing the low-level behaviors of a system. Violating information flow properties results in compromised safety, integrity, and privacy as intruders can use the secret information to gain insights into the system implementation.

Although information flow security has been widely investigated and enforced in the context of programming languages [17], it has not been studied extensively in the CPS domain. As CPS have mixed discrete-continuous behaviors and are often complex, modeling and analyzing information flow security of CPS is notably challenging. In an industrial setting, CPS designers usually validate a system to guarantee that its safety requirements are satisfied, but often neglect or do not carefully test information flow properties. Consequently, the system might expose information leakage during runtime and be subject to certain classes of attacks where an attacker can physically observe the system behavior and learn how to drive the system toward unsafe behaviors [3, 9–12, 15, 16]. There is a strong need for light-weight, inexpensive analysis methods that can efficiently

identify information flow vulnerabilities to strengthen the security of CPS models, especially as CPS continue to be applied in safety-critical areas.

In this paper, we propose a methodology that can efficiently detect violations of information flow properties for hybrid systems that model CPS. We focus on the property of *non-interference*, which requires that the states observed by low-security users remain unchanged regardless of the actions taken by high-security users. We introduce an algorithm that can generate a set of *security constraints* over the structure of a hybrid system to enforce non-interference. Given a hybrid system modeled as a parallel composition of hybrid automata and a declaration of security labels for each variable, if our static security checker accepts the annotated model, it is guaranteed to have the property of non-interference with respect to the annotation. In addition, our algorithm can effectively infer the security labels for variables that are not explicitly declared by a user. Our algorithm works at a high level in three main steps: 1) a hybrid system is decomposed into individual automata, 2) security constraints are generated for each automaton and represented as a directed dependency graph, 3) each individual graph is combined into one graph, which is used to check for violations and determine the security labels for all variables in the model. The time complexity of our algorithm is linear in the description size of the hybrid system. It is worth noting that our algorithm can both check a composite model for non-interference and infer security labels without using the process of flattening, which increases description size exponentially.

We evaluate the effectiveness of our approach to identify information flow leakage by applying it in two distinct domains: gas transportation and smart power systems. The first case study models the Russia–Ukraine gas pipeline system [5] in which attackers were able to observe the change of gas flow rate in Ukraine and compromise system operation. The second case study is the FREEDM smart grid system [8], where an attacker can infer private information about the status of a battery by observing the power flow in and out of a macrogrid and use that information to inject extra power, potentially causing the battery to explode. For both case studies, we will present the original models in Simulink/Stateflow (SLSF) format, and then demonstrate how our method can be used to detect non-interference security violations, as well as infer valid security labels from a partial user specification. Our main contributions are:

- A formal treatment of non-interference for a hybrid system modeled as a parallel composition of hybrid automata.
- A light-weight static analysis technique to detect information flow violations with complexity linear in the description size of the system.
- An analysis tool that detects information leaks in real-world hybrid systems. Our tool works on SLSF models that are widely used in both industry and academia.

Related work. Until now, only a few results have been reported for verifying information flow properties of hybrid systems. The most relevant work to this paper is presented in [14], which introduces a set of typing rules that enforces non-interference for a hybrid system expressed as a programming language. In contrast, we generate security constraints over the structure of hybrid automata. Moreover, we implement a static security checker and use it to verify non-interference property for two case studies while the work proposed in [14] does not feature any real-world applications of their work. Whalen et al. proposed a model checking approach that formalizes non-interference property through a notion of trace equivalence and then utilizes model checking tools to analyze non-interference property of Simulink models [22]. Other works of [20] and [1] also apply model checking to verify information flow properties for the discrete models of the gas pipeline system and the FREEDM smart grid system, respectively. Moreover, the recent work of [4] introduces a hybrid dynamic logic for verifying information-flow properties of a hybrid system modeled as a hybrid program. Such logic is impressive; however, it is too expensive to apply the corresponding analysis for larger-scale CPS.

2 HYBRID SYSTEMS MODELING

Hybrid automata [2] are a popular modeling formalism 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.

Definition 1 (Hybrid automata). *A hybrid automaton is a tuple* $\mathcal{A} \triangleq \langle \mathcal{V}, Mode, Trans, Init \rangle$ *which includes the following components:*

- $\mathcal{V}$*: the finite set of variables, partitioned as* $\mathcal{X} \cup \mathcal{U}$*, where* $\mathcal{X}$ *is the finite set of* n *state variables, and* $\mathcal{U}$ *is the finite set of* m *input variables. We denote* $\mathbf{x} \in \mathbb{R}^n$ *as the valuations (i.e., a function mapping each variable to a point in* $\mathbb{R}$*) of state variables. The valuations of* m *input variables are assigned by an input signal* $\mathbf{u}$*.*
- *Mode: the finite set of discrete modes. For each mode* $m \in Mode$*,* $m.inv$ *is a Boolean expression over* $\mathcal{V}$ *which denotes the invariant of mode* m*, and* $m.flow$ *is a set of differential equations (e.g., in which the left-hand side is* $\dot{x}$ *and the right-hand side is an expression over* $\mathcal{V}$*) that describes the rate of change of state variables. A state* s *is a pair* $(m, \mathbf{x})$*, where* $m \in Mode$ *and* $\mathbf{x} \in \mathbb{R}^n$*. We denote* $Q \subseteq Mode \times \mathbb{R}^n$ *as the state-space of* $\mathcal{A}$*.*
- *Trans: the finite set of transitions between modes. Each transition is a tuple* $\tau \triangleq \langle src, dst, grd, rst \rangle$*, where src is a source mode and dst is a target mode that may be taken when a guard condition grd, which is a Boolean expression over* $\mathcal{X} \cup \mathcal{U}$*, is satisfied; and rst is an assignment of* $\mathcal{X}$ *after the transition.*
- *Init* $\subseteq Q$*: the set of initial states.*

Here, we assume that the output variables of $\mathcal{A}$ are the same as its state variables. We use the dot (.) notation to refer to different components of tuples of the transitions and modes, e.g., $\tau.grd$ refers to the guard of a transition τ, and $m.inv$ refers to the invariant of a mode m. Also, an expression e on the right hand side of a flow equation ($\dot{x} = e$) and a reset assignment ($x = e$) can be a variable, a constant, or an arithmetic combination of constants and variables. The valuation of an expression e at time t, denoted $[\![e]\!](\mathbf{x}(t), \mathbf{u}(t))$, is obtained by substituting each input and state variable in e by their corresponding value from $\mathbf{u}(t)$ and $\mathbf{x}(t)$.

The semantics of a hybrid automaton $\mathcal{A}$ can be defined in terms of executions, which are alternating sequences of continuous trajectories and discrete transitions starting from an initial state $s_0 \in Init$ following an input signal $\mathbf{u}$ over a time interval $[0, \delta)$.

Definition 2 (Execution). *Given an initial state $s_0 \in Init$, an input signal $\mathbf{u}$, an execution is a sequence $\pi(s_0, \mathbf{u}) = \gamma_0, \tau_1, \gamma_1, \tau_2, \ldots, \tau_{n-2}, \gamma_{n-1}$ corresponding to the sequence of time points t_i, where $i \in \{0, \ldots, n\}$, $t_0 = 0$, $t_n = \delta$, and $t_i < t_{i+1}$ such that:*

- *$\gamma_i \in Traj$ is a continuous trajectory which is a mapping function $\gamma_i : [t_i, t_{i+1}) \rightarrow \mathbb{R}^n$ such that for every $t \in [t_i, t_{i+1})$:*
  - *a mode $m \in Mode$ corresponding to γ_i does not change,*
  - *$\dot{\mathbf{x}}(t) = [\![m.flow]\!](\mathbf{x}(t), \mathbf{u}(t))$, i.e., the continuous evolution is consistent with the flow dynamics of the corresponding mode, and*
  - *$[\![m.inv]\!](\mathbf{x}(t), \mathbf{u}(t))$ is true, i.e., all states along the trajectory must satisfy the invariant at every time point.*
- *$\tau_i \in Trans$ is a transition which models an instantaneous update from the current state $s = (m, \mathbf{x})$ to the next state $q = (m_q, \mathbf{x}_q)$ at t_i such that:*
  - *$\tau.src = m \wedge \tau.dst = m_q$,*
  - *$[\![\tau.grd]\!](\mathbf{x}(t_i), \mathbf{u}(t_i))$ is true, and*
  - *$\mathbf{x}_q(t_i) = [\![\tau.rst]\!](\mathbf{x}(t_i), \mathbf{u}(t_i))$.*

An execution always starts in an initial state, and can be *infinite* or *finite*. If an execution π is finite, it ends in a trajectory. An *execution fragment* of π is any alternating sequence of continuous trajectories and discrete transitions appearing in π. If an execution fragment of π is finite and includes an initial state, it is also considered as a *prefix* of π. We write $\pi = \pi_p \circ \pi_f$ to denote that an execution π can be concatenated by its prefix π_p and execution fragment π_f, where the last state of π_p is the first state of π_f. We denote $\mathrm{Exec}(\mathcal{A})$ as the set of all executions of a hybrid automaton $\mathcal{A}$.

Definition 3 (State signal). *Given an input signal $\mathbf{u}$ and an initial state $s_0 = (m_0, \mathbf{x}_0)$, we define $\Sigma(\mathbf{x}_0, \mathbf{u})$ as a state signal which captures the evolution of the state variables starting from $\mathbf{x}_0$ and following along the execution $\pi(s_0, \mathbf{u})$.*

We assume that given an input signal and an initial state, there always exists a corresponding state signal. During the execution of a hybrid automaton $\mathcal{A}$, the updates of its modes are internal and only the updates of its state variables are observable. In what follows, we use the notion $\mathbf{x}$ to represent for the state of $s = (m, \mathbf{x})$. In this paper, we focus on a deterministic hybrid automaton whose behavior is unique according to an input signal $\mathbf{u}$ and an initial state $\mathbf{x}_0$.

Definition 4 (Deterministic hybrid automaton). *A hybrid automaton $\mathcal{A}$ is deterministic if the following conditions hold:*

- *For every transition $\tau \in Trans$, $\tau.grd \cap (\tau.src).inv = \emptyset$, i.e., the guard condition of every outgoing transition of a mode is not intersected with its invariant.*
- *For every pair of transitions $\tau_1, \tau_2 \in Trans$, if $\tau_1.src = \tau_2.src$, then $\tau_1.grd \cap \tau_2.grd = \emptyset$.*
- *For every mode $m \in Mode$, $m.flow$ is a Lipschitz continuous function over the state and input variables so that the solution of $m.flow$ is unique.*

Intuitively, a hybrid automaton is deterministic if at most one of its outgoing transitions can be taken when the mode invariant is violated, and the solutions of the flows equations are unique. Hence, any execution and state signal of a deterministic hybrid automata corresponding to an input signal $\mathbf{u}$ and an initial state $\mathbf{x}_0$ is uniquely defined.

Next, we define a hybrid system as a parallel composition of multiple hybrid automata. Given two hybrid automata $\mathcal{A}_1 \triangleq \langle \mathcal{V}_1, Mode_1, Trans_1, Init_1 \rangle$, and $\mathcal{A}_2 \triangleq \langle \mathcal{V}_2, Mode_2, Trans_2, Init_2 \rangle$, if $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$, then $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible and can be composed.

Definition 5 (Parallel composition). *Given two compatible hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$, the parallel composition of $\mathcal{A}_1$ and $\mathcal{A}_2$ is a hybrid automaton $\mathcal{A}$, written as $\mathcal{A} \triangleq \mathcal{A}_1 \| \mathcal{A}_2$, whose components are specified as follow:*

- *$\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$,*
- *$\mathcal{U} = (\mathcal{U}_1 \cup \mathcal{U}_2) \setminus \mathcal{X}$,*
- *$Mode = Mode_1 \times Mode_2$, and the invariant and flow dynamic of each model $m = m_1 \times m_2 \in Mode$ is the conjunction and disjunction of the corresponding invariant and flow dynamic of $m_1 \in Mode_1$ and $m_2 \in Mode_2$, respectively,*
- *for each transition $\tau_1 \in Trans_1$, and $\tau_2 \in Trans_2$, there exists a corresponding transition $\tau \in Trans$ such that:*
  - *$\tau.src = \tau_1.src \times \tau_2.src$, $\tau.dst = \tau_1.dst \times \tau_2.src$, $\tau.grd = \tau_1.grd$, and $\tau.rst = \tau_1.rst$, or*
  - *$\tau.src = \tau_1.src \times \tau_2.src$, $\tau.dst = \tau_1.src \times \tau_2.dst$, $\tau.grd = \tau_2.grd$, and $\tau.rst = \tau_2.rst$, or*
  - *$\tau.src = \tau_1.src \times \tau_2.src$, $\tau.dst = \tau_1.dst \times \tau_2.dst$, $\tau.grd = \tau_1.grd \wedge \tau_2.grd$, and $\tau.rst = \tau_1.rst \cup \tau_2.rst$,*
- *$Init = Init_1 \times Init_2$.*

The state space of the composed automaton $\mathcal{A}$ is also a product of the state space of each component $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., $Q = Q_1 \times Q_2$. Beside that, all executions of $\mathcal{A}$ are also executions of both $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., $\mathrm{Exec}(\mathcal{A}) \downarrow_{\mathcal{A}_i} \subseteq \mathrm{Exec}(\mathcal{A}_i)$ for $i \in \{1, 2\}$, where $\mathrm{Exec}(\mathcal{A}) \downarrow_{\mathcal{A}_i}$ is a projection of $\mathrm{Exec}(\mathcal{A})$ on the component $\mathcal{A}_i$. The parallel composition of hybrid automata is *commutative* and *associative*. Multiple components can be composed transitively in parallel by recursively composing two components with a third one, and so on. It is worth noting that the parallel composition of two deterministic hybrid automata is also deterministic.

Definition 6 (Hybrid system). *A hybrid system $\mathcal{H}$ is a parallel composition of two or more hybrid automata, written as $\mathcal{H} \triangleq \mathcal{A}_1 \| \mathcal{A}_2 \| \ldots \| \mathcal{A}_n$.*

3 NON-INTERFERENCE

In this section, we will define the *non-interference* property of a hybrid system based on the semantic execution of a hybrid automaton. To do so, we first define a *security annotation* and *low equivalence*.

Definition 7 (Security annotation). *A security annotation for a hybrid system $\mathcal{H}$ is a function that maps each variable of $\mathcal{H}$ to a security level of low or high.*

A variable is annotated *low* if its value is observable, and *high* if it is confidential. Without loss of generality, we only consider two security levels of *low* and *high*, where $low \leq high$ meaning

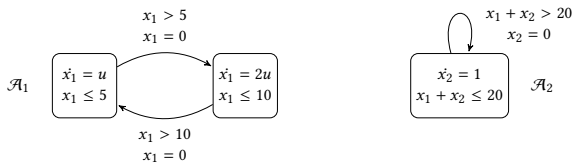

Figure 1: Examples of two hybrid automata.

that a variable annotated as low is less confidential than the one annotated as high. Input or state signals that are observable belong to a low domain, and private signals belong to a high domain.

Definition 8 (Low equivalence). *Two states of a hybrid system $\mathcal{H}$ are said to be low equivalent if their projections on the low domain have the same value.*

We use the notion of $=_{low}$ to denote a *low equivalence* relation. For instance, the notion $\mathbf{x}_0 =_{low} \mathbf{x}_0'$ means that initial states $\mathbf{x}_0$ and $\mathbf{x}_0'$ are equivalent on the low domain. The above definition can directly be applied for state and input signals. Two state (or input) signals are low equivalent if their low domain projections are indistinguishable over time.

Definition 9 (Non-interference). *A hybrid system $\mathcal{H}$ is non-interference secure iff for every pair of initial state values $\mathbf{x}_0, \mathbf{x}_0' \in Init$, and a pair of input signals $\mathbf{u}$, $\mathbf{u}' \in \mathbf{U}$ for $\mathbf{x}_0$, $\mathbf{x}_0'$ respectively, the following condition holds:*

$$(\mathbf{x}_0, \mathbf{u}) =_{low} (\mathbf{x}_0', \mathbf{u}') \implies \Sigma(\mathbf{x}_0, \mathbf{u}) =_{low} \Sigma(\mathbf{x}_0', \mathbf{u}'). \quad (1)$$

Intuitively, the Condition 1 can be interpreted as: if two initial states share the same values on a low domain, then the behaviors of the system executed w.r.t the same low inputs are indistinguishable by public observers.

Non-interference prevents two important kinds of information leaks: *explicit* and *implicit* information flows. An explicit flow occurs when the value of a low variable is directly derived from the value of a high variable. For instance, an assignment $x_2 = x_1$ causes an explicit flow from a high variable x_1 to a low variable x_2. On the other hand, an implicit flow occurs when a low variable is updated indirectly due to information read from a high variable. As an example, a transition which has the following guard condition and reset action: $x_1 > x_2; x_2 = 1$ implicitly discloses the value of a high variable x_1 to a low variable x_2.

Problem 1 (Non-interference checking for hybrid systems). *Given a hybrid system $\mathcal{H} \triangleq \mathcal{A}_1 \| \mathcal{A}_2 \| \ldots \| \mathcal{A}_n$ with a set of security annotations $\mathcal{S}$ for all variables of $\mathcal{H}$ such that $\mathcal{S} = \mathcal{S}_{low} \cup \mathcal{S}_{high}$, where $\mathcal{S}_{low}$ and $\mathcal{S}_{high}$ are the sets of low and high security annotations for input and state variables, respectively, we want to check whether $\mathcal{H}$ satisfies non-interference property.*

Example 1. *We consider two hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$ shown in Figure 1, where the security annotations of their state and input variables decide whether they satisfy non-interference. For $\mathcal{A}_1$, the valuation of a state variable x_1 explicitly depends on the input u. If u is high and x_1 is low, the Condition 1 is violated as a state signal of x_1 is influenced by the input signal u. In the case that u is low and x_1 is either low or high, $\mathcal{A}_1$ is non-interference secure. Hence, to ensure that $\mathcal{A}_1$ is non-interference secure, the security level of u cannot be higher than x_1. For $\mathcal{A}_2$, since the update of x_2 implicitly depends on x_1, $\mathcal{A}_2$ only satisfies Condition 1 if the security label of x_1 is not higher than that of x_2. The hybrid system $\mathcal{H} \triangleq \mathcal{A}_1 \| \mathcal{A}_2$ preserves the same security constraints over the state variables x_1, x_2, and the input variable u such that the security level of u is not higher than that of x_1, and the security level of x_1 is not higher than that of x_2. Thus, if the security annotations provided by a user satisfy these constraints, $\mathcal{H}$ is non-interference secure.*

4 ANALYSIS

To detect information flow violations, we break up a composed hybrid system into individual hybrid automata, recursively break up each hybrid automaton into model components, and constrain each component's security labels to ensure no information leaks can occur. Then, we check the validity of a user's label annotations against a graph representation of the component label dependencies and see if any violation occurs. In the case of a violation, the flow producing the information leak is returned. At a high level, our algorithm is laid out as follows:

Algorithm 1 Non-Interference Violation Detection

Input: a hybrid system $\mathcal{H}$, a set of security annotations $\mathcal{S}$ for variables of $\mathcal{H}$

1: **procedure** Non-Interference Check
2: $A \leftarrow$ decompose($\mathcal{H}$) ▷ decompose a hybrid system into constituent automata
3: $G \leftarrow$ foreach $\mathcal{A}_i$ in A: buildDependencyGraphs($\mathcal{A}_i$)
4: $G' \leftarrow$ mergeDependencyGraphs(G)
5: **return** checkForViolations($\mathcal{S}, G'$) ▷ either OK or a Violation
6: **end procedure**

4.1 Constraint Generation

For a hybrid system to have the property of non-interference, each individual automaton composed in parallel must have the property in isolation. The first step of our analysis is to verify a single automaton in isolation. This is accomplished by breaking an automaton down into recursive components, assigning labels to each component, and enforcing constraints between labels to eliminate invalid annotations.

Definition 10 (Security labeling). *A security labeling is a function: $sl : \mathcal{H} \rightarrow \{low, high\}$ mapping each component of $\mathcal{H}$ to a security level of low or high.*

We note that the security labeling for the hybrid system $\mathcal{H}$ subsumes the security annotations for its variables provided by a user. Security labeling indicates the security domain in which a component can be executed corresponding to given security levels of variables.

A component with a certain label is valid only if it executes in the label's security context. This allows us to reject labelings based on the semantics of the hybrid system being executed. Components impose constraints on their subcomponents, restricting the set

of valid executions. For instance, specifying a transition as high prevents low variables from being written in its reset equations, which creates further labeling restrictions on the variables and expressions in each of the reset flows.

The two different constraints that can be generated between components a, b are $sl(a) = sl(b)$ or $sl(a) \leq sl(b)$. We decompose a hybrid automaton into different components: variables, expressions, flows, invariants, modes, transitions, guards, resets, and models, and then generate constraints for each component that determine valid security labels.

4.2 Component Rules

We assume that the set of variables of a hybrid automaton $\mathcal{A}$ can be partitioned as $\mathcal{V} = \mathcal{V}_{high} \cup \mathcal{V}_{low}$ corresponding to a given set of security annotations, where $\mathcal{V}_{high}$ and $\mathcal{V}_{low}$ are the sets of high and low input and state variables, respectively. We introduce the *syntax-directed* security rules for a hybrid automaton that enforces non-interference security policy. These security rules describe what labels can be assigned to different components of a hybrid automaton to prevent information flowing from high to low variables. Given $h \in \mathcal{V}_{high}$ and $\ell \in \mathcal{V}_{low}$ as the instances of low and high level variables, our security rules are described as follows.

Expression rule: given an expression e and $Vars(e)$ which denotes the set of variables in an expression e, $\forall v \in Vars(e)$, $sl(v) \leq sl(e)$. Thus, the security level of an expression (including a Boolean expression such as an invariant and a guard condition) is at least as restrictive as that of every variable in it.

Assignment rule: for an assignment $a : x = e$, we have that $sl(a) \leq sl(e) \leq sl(x)$. The assignment rule prohibits *explicit flows* of information from high to low domains. An assignment of a low variable is secure only if the expression on the right-hand side is low, e.g., an assignment $\ell = \ell + 1$ is allowed, but an assignment $\ell = h$ is not.

Flow equations rule: A flow equation is a sequence of assignments $a = a_1; a_2; \ldots; a_n$, where $a_i : \dot{x_i} = e_i$ such that its security label is restricted as $\forall i \in \{1, \ldots, n\}$, $sl(a) \leq sl(a_i)$. In other words, the security levels of the flow equations of a mode is equal to the lowest security level of an individual assignment.

Mode rule: for every mode $m \in Mode$, $sl(m) \leq sl(m.inv) \leq sl(m.flow)$. The security level of a mode depends on its invariant and flow dynamics. If a mode has a high invariant, its flow equations must be high. In this case, a mode can be either high or low.

Reset rule: the resets (updates) of variables of a transition is also a sequence of assignments $a = a_1; a_2; \ldots; a_n$ where $a_i : x_i = e_i$ such that $sl(a) \leq sl(a_i)$.

Transition rule: To specify the security level for a transition $\tau \in Trans$, we first investigate whether $\tau.src =_{low} \tau.dst$. Here, we abuse the notion of $=_{low}$ to specify the *observational equivalence* of the source and destination mode of a transition when projecting them on a low domain. For instance, if the source and destination modes' flows both contain low expressions as $\dot{\ell} = 1$, they are equivalent to a public observer. High modes are implicitly observationally equivalent in a low domain. For every tradition $\tau \in Trans$,

- Case 1: if $\tau.src =_{low} \tau.dst$ is true, then $sl(\tau) \leq sl(\tau.grd) \leq sl(\tau.rst)$,
- Case 2: if $\tau.src =_{low} \tau.dst$ is false, then $sl(\tau) \leq sl(\tau.grd) \leq sl(\tau.rst) \wedge sl(\tau.grd) \leq sl(\tau.src) \leq sl(\tau.dst)$

We note that instead of arbitrarily forcing source and destination modes to be high if the conditional guard is high, observational equivalence makes our analysis less conservative. Intuitively, if a transition with a high guard does not either cause an instantaneous update or disrupt the continuous update of low variables, then information will not leak.

Consecutive transition rule: for two transitions $\tau_1, \tau_2 \in Trans$, if $\tau_1.dst = \tau_2.src \wedge \tau_1.src \neq \tau_2.dst$, then $sl(\tau_1) \leq sl(\tau_2)$. Intuitively, if the preceding transition τ_1 is low, the following transition τ_2 can either be low or high. However, if τ_1 is high, τ_2 must be high as well. As a result, if any transition τ in the chain of transitions is high, every transition following τ in the chain must be high. Such a restriction prohibits any update of low variables based on a preceding high condition, ensuring that there are no implicit flows along the path.

4.3 Translating Constraints into a Dependency Graph

To check whether or not a hybrid system obeys non-interference, we need to examine the data flows between various components of the model. We will represent this data flow information in a *dependency graph*, and it will provide some nice analysis properties that we can use to detect security leaks.

Definition 11 (Dependency Graph). *A directed dependency graph $G = (V, E)$ consists of nodes $v \in V$ where each v_i is assigned a security label l_i, and edges $e \in E$ where $e = (v_i, v_j)$ implies that $l_i \leq l_j$. Additionally, G also contains one unique node per security level specified, these labels represent the security levels themselves.*

Note that there are implicit edges from the lowest security level to all other nodes in a dependency graph, as well as edges from every node to the highest security type. A dependency graph has two useful properties:

Property 1. *Given a dependency graph G, if there is a path from l_i to l_j, then $l_i \leq l_j$.*

This property directly follows from the definition of a dependency graph and transitivity on the security label relation.

Property 2. *Any nodes in the same strongly connected component (SCC) of G must have the same security type.*

Since any two nodes v_i, v_j in the same SCC have paths to each other, we know that $l_j \leq l_i \wedge l_i \leq l_j$. The only way this can be true is if $l_i = l_j$. The dependency graph captures constraints between different security labels and provides an efficient check for violations. For a hybrid automaton $\mathcal{A}$, the dependency graph $G_{\mathcal{A}}$ expresses constraints between various components of the model.

Dependency graph construction. Given a set of constraints generated from $\mathcal{A}$, we can create its dependency graph $G_{\mathcal{A}}$ by first adding every unique component as a node, and then rewriting each constraint as edges in $G_{\mathcal{A}}$. A constraint $v_j < v_i$ is expressed by adding an edge $e = (v_j, v_i)$ to $G_{\mathcal{A}}$, and $v_i = v_j$ is expressed by

adding edges $e_1 = (v_i, v_j)$ and $e_2 = (v_j, v_i)$ to $G_{\mathcal{A}}$. Representing a set of constraints as a dependency graph will allow us to efficiently determine if there are any violations.

Theorem 1. *Given a hybrid automaton $\mathcal{A}$ with a set of security annotations $\mathcal{S}$ for all variables of $\mathcal{A}$, if the security constraints of all components of $\mathcal{A}$ are satisfied, then there is no path from high to low in the corresponding dependency graph of $\mathcal{A}$, i.e., $\mathcal{A}$ is non-interference secure.*

Proof. We need to prove that if for any pairs of initial states and input signals of $\mathcal{A}$ such that $(\mathbf{x}_0, \mathbf{u}) =_{low} (\mathbf{x}'_0, \mathbf{u}')$, then we have $\Sigma(\mathbf{x}_0, \mathbf{u}) =_{low} \Sigma(\mathbf{x}'_0, \mathbf{u}')$. Since an execution of a hybrid automaton is an altering sequence of 1) a continuous evolution of state variables associated with a mode, and 2) a discrete evolution of state variables corresponding to a transition, we will first prove that Condition 1 holds over each mode, transition, two consecutive transitions, then inductively show that it holds for the entire model.

Case 1 (Mode): for a mode $m \in Mode$, we have that $sl(m) \leq sl(m.inv) \leq sl(m.flow)$. If an invariant is high, then the flow equations of a mode contain only high variables on the left-hand side. Thus it is apparent that the non-interference property holds as there are no updates of low variables.

Otherwise, we consider a case where an invariant is low. Assume there exists at least one differential equation of the form $\dot{\ell} = e$. Based on the security rules of flow dynamic $sl(e) \leq sl(\dot{\ell})$, meaning that an expression e does not contain any high variable. In addition, the solution of a different equation $\dot{\ell} = e$ is unique corresponding to the same initial states and input signals. Therefore, for every $t \in [0, \delta]$, we have $\mathbf{x}(t) =_{low} \mathbf{x}'(t)$. Moreover, $\mathbf{x}(t)$ and $\mathbf{x}'(t)$ both satisfy an invariant with respect to the trajectory semantics. Hence, we have that $\Sigma(\mathbf{x}_0, \mathbf{u}) =_{low} \Sigma(\mathbf{x}'_0, \mathbf{u}')$ is true.

Case 2 (Transitions): According to the security rule of transition, if the guard condition is low, then the reset component, source and destination modes can be either high or low. We assume that there is at least one low variable ℓ has been updated by a transition. Since an instantaneous update $\ell = e$ is low which means that an expression e is low, i.e., e does not contains any high variables. Because $[\![e]\!](\mathbf{x}(t), \mathbf{u}(t)) =_{low} [\![e]\!](\mathbf{x}'(t), \mathbf{u}'(t))$, the value of a low variable updated by a transition at time $t \in \mathbb{T}$ is only dependent on its previous value. Moreover, applying the proof of the mode case to the source and destination modes of a transition, the Condition 1 is trivially true.

On the other hand, if the guard condition is high, then the reset component must be high. Thus, there is no instantaneous update of low variables. In that case, the source and destination modes are observational equivalence on a low domain which means there is also no continuous update of low variables. Thus, the Condition 1 certainly holds.

Case 3 (Consecutive transitions): Assume that τ_1 and τ_2 are two consecutive transitions, so $sl(\tau_1) \leq sl(\tau_2)$. Let $\pi(\mathbf{x}_0, \mathbf{u}) = \pi_1 \circ \pi_2$, where π_1 is a prefix π corresponding to τ_1, and π_2 is an execution fragment of π corresponding to τ_2, we need to show $\pi(\mathbf{x}'_0, \mathbf{u}') = \pi'_1 \circ \pi'_2$ are equivalent to $\pi(\mathbf{x}_0, \mathbf{u})$ on a low domain. If τ_1 is high, so does τ_2. Thus, all states along the executions with the same initial state are identical on a low domain as there are no updates of low variables, then $\pi(\mathbf{x}_0, \mathbf{u}) =_{low} \pi(\mathbf{x}'_0, \mathbf{u}')$ is always true. In the case that τ_1 is low, τ_2 can either be low or high. From the proof of the case of individual transition, τ_1 is low that implies $\pi_1 =_{low} \pi'_1$. As the result, π_2 and π'_2 share the same first state on a low domain, so $\pi_2 =_{low} \pi'_2$ regardless whether τ_2 is either high or low. Thus, we have $\pi(\mathbf{x}_0, \mathbf{u}) =_{low} \pi(\mathbf{x}'_0, \mathbf{u}')$, implying that $\Sigma(\mathbf{x}_0, \mathbf{u}) =_{low} \Sigma(\mathbf{x}'_0, \mathbf{u}')$ always holds.

Now we apply the induction hypothesis on an arbitrary execution $\pi = \pi_p \circ \pi_f$ over the time interval $[0, \delta)$, where π_p is a prefix of π over the time interval $[0, t_p)$, and π_f is an execution fragment of π over the time interval $[t_p, \delta)$. Assume that $\pi_p(\mathbf{x}_0, \mathbf{u}) =_{low} \pi_p(\mathbf{x}'_0, \mathbf{u}')$, we need to prove that $\pi_f(\mathbf{u}, \mathbf{x}_{0_f}) =_{low} \pi_f(\mathbf{u}', \mathbf{x}'_{0_f})$, where $\mathbf{x}_{0_f}$, and $\mathbf{x}'_{0_f}$ are the first states of π_f and π'_f. Since $\mathbf{x}(t_p) = \mathbf{x}'(t_p)$ and the last state of π_p is the first state of π_f, it is apparent that $\mathbf{x}_{0_f} =_{low} \mathbf{x}'_{0_f}$, respectively. Without loss of generality, we assume that π_f is a continuous trajectory corresponding to a mode. Thus, for every $t \in [t_p, \delta)$, $(\mathbf{u}(t), \mathbf{x}_{0_f}) =_{low} (\mathbf{u}'(t), \mathbf{x}'_{0_f})$, we have $\pi_f(\mathbf{u}, \mathbf{x}_{0_f}) =_{low} \pi'_f(\mathbf{u}', \mathbf{x}'_{0_f})$ by applying the proof for Case 1 (mode). As a result, $\Sigma(\mathbf{x}_0, \mathbf{u}) =_{low} \Sigma(\mathbf{x}'_0, \mathbf{u}')$ is true over the time interval $[0, \delta)$. Considering π_f as an execution fragment corresponding to a transition and two consecutive transitions, the similar proofs can also be derived according to the proofs of Case 2 and Case 3. Thus, given any pairs of initial states and input signals of $\mathcal{H}$ such that $(\mathbf{x}_0, \mathbf{u}) =_{low} (\mathbf{x}'_0, \mathbf{u}')$, then we have $\Sigma(\mathbf{x}_0, \mathbf{u}) =_{low} \Sigma(\mathbf{x}'_0, \mathbf{u}')$. □

4.4 Non-interference Checking

For an individual automaton, a path from high to low in its corresponding dependency graph represents a non-interference violation, and the components along that path together allow an information leak. If there is no such violation, we can use Property 2 to determine the inferred security labels of various components by computing the SSC that contains each node with high and low labels. Any components that are not in a component containing a label type cannot be inferred, we call these components "remainders". The presence of remainders means that we cannot definitively tell if the model satisfies non-interference or not based on the partial annotation; the designer must further specify additional labels to complete the analysis.

The case for a hybrid system modeled as a parallel composition of hybrid automata is more complex. The only way composed automata can interact is through shared variables. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are independent hybrid automata (i.e., do not share any variables), then it is apparent that no information can flow between either one; whether or not there is an information leak that depends on the results of checking each model individually. If the two models share any variables however, this is not the case.

To address the case of two parallel automata sharing variables, we rename the conflicting variables so they are unique and analyze the automata separately, adding a constraint that the renamed variables are equivalent. For example, given two automata sharing x would result in one of them having x renamed to y and a constraint $x = y$ added. If there are violations in any of the individual models, those will be reported first. If all of them pass, the constraints equating renamed variables will ensure that no variable has been assigned high that should be low in a different model.

Theorem 2. *If two hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$ both satisfy non-interference property according to the set of security annotations $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, and $\forall x \in \mathcal{V}_c, \mathcal{S}_1(x) = \mathcal{S}_2(x)$, where $\mathcal{V}_c$ is a set of shared variables, then the hybrid system $\mathcal{H} = \mathcal{A}_1 \| \mathcal{A}_2$ also satisfies non-interference property.*

Proof. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-interference secure, according to the Condition 1 we have that:

$$(\mathbf{x}_{0_1}, \mathbf{u}_1) =_{low} (\mathbf{x}'_{0_1}, \mathbf{u}'_1) \implies \Sigma(\mathbf{x}_{0_1}, \mathbf{u}_1) =_{low} \Sigma(\mathbf{x}'_{0_1}, \mathbf{u}'_1),$$
$$(\mathbf{x}_{0_2}, \mathbf{u}_2) =_{low} (\mathbf{x}'_{0_2}, \mathbf{u}'_2) \implies \Sigma(\mathbf{x}_{0_2}, \mathbf{u}_2) =_{low} \Sigma(\mathbf{x}'_{0_2}, \mathbf{u}'_2),$$

where $\mathbf{x}_{0_i}, \mathbf{x}'_{0_i}$ is a pair of initial states, and $\mathbf{u}_i$, $\mathbf{u}'_i$ is a pair of input signals for $\mathbf{x}_{0_i}$, $\mathbf{x}'_{0_i}$ of $\mathcal{A}_i$, $i \in \{1, 2\}$. Assume that the shared variables of $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same security levels, we need to prove that:

$$(\mathbf{x}_0, \mathbf{u}) =_{low} (\mathbf{x}'_0, \mathbf{u}') \implies \Sigma(\mathbf{x}_0, \mathbf{u}) =_{low} \Sigma(\mathbf{x}'_0, \mathbf{u}'),$$

where $\mathbf{x}_0 = \mathbf{x}_{0_1} \times \mathbf{x}_{0_2}$, $\mathbf{x}'_0 = \mathbf{x}'_{0_1} \times \mathbf{x}'_{0_2}$, $\mathbf{u} = (\mathbf{u}_1 \cup \mathbf{u}_2) \setminus (\Sigma(\mathbf{x}_{0_1}, \mathbf{u}_1) \cup \Sigma(\mathbf{x}'_{0_2}, \mathbf{u}_2))$, and $\mathbf{u}' = (\mathbf{u}'_1 \cup \mathbf{u}'_2) \setminus (\Sigma(\mathbf{x}'_{0_1}, \mathbf{u}'_1) \cup \Sigma(\mathbf{x}'_{0_2}, \mathbf{u}'_2))$.

Proof by contradiction. We assume that there is a case $(\mathbf{x}_0, \mathbf{u}) =_{low} (\mathbf{x}'_0, \mathbf{u}')$, but $\Sigma(\mathbf{x}_0, \mathbf{u}) \neq_{low} \Sigma(\mathbf{x}'_0, \mathbf{u}')$. We now project the Condition 1 of a hybrid system $\mathcal{H}$ on each hybrid automaton $\mathcal{A}_1$ and $\mathcal{A}_2$. Since the shared variables of $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same security levels, the projection of $(\mathbf{x}_0, \mathbf{u}) =_{low} (\mathbf{x}'_0, \mathbf{u}')$ on each automaton $\mathcal{A}_1$ and $\mathcal{A}_2$ result in $(\mathbf{x}_{0_1}, \mathbf{u}_1) =_{low} (\mathbf{x}'_{0_1}, \mathbf{u}'_1)$ and $(\mathbf{x}_{0_2}, \mathbf{u}_2) =_{low} (\mathbf{x}'_{0_2}, \mathbf{u}'_2)$, regardless the shared variables are both low or both high. On the other hand, the projections of $\Sigma(\mathbf{x}_0, \mathbf{u}) \neq_{low} \Sigma(\mathbf{x}'_0, \mathbf{u}')$ on each automaton $\mathcal{A}_1$ and $\mathcal{A}_2$ result in at least either $\Sigma(\mathbf{x}_{0_1}, \mathbf{u}_1) \neq_{low} \Sigma(\mathbf{x}'_{0_1}, \mathbf{u}'_1)$ or $\Sigma(\mathbf{x}_{0_2}, \mathbf{u}_2) \neq_{low} \Sigma(\mathbf{x}'_{0_2}, \mathbf{u}'_2)$, or both are true. Since both $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic and have unique state signals corresponding to given initial states and input signals. As a result, either $(\mathbf{x}_{0_1}, \mathbf{u}_1) =_{low} (\mathbf{x}'_{0_1}, \mathbf{u}'_1) \implies \Sigma(\mathbf{x}_{0_1}, \mathbf{u}_1) \neq_{low} \Sigma(\mathbf{x}'_{0_1}, \mathbf{u}'_1)$, or $(\mathbf{x}_{0_2}, \mathbf{u}_2) =_{low} (\mathbf{x}'_{0_2}, \mathbf{u}'_2) \implies \Sigma(\mathbf{x}_{0_2}, \mathbf{u}_2) \neq_{low} \Sigma(\mathbf{x}'_{0_2}, \mathbf{u}'_2)$ are true. Hence, at least one of the hybrid automaton $\mathcal{A}_1$ and $\mathcal{A}_2$ is not non-interference secure, which contradicts to the original assumption that they both satisfy non-interference property. Therefore, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-interference and their shared variables have the same security levels, their parallel composition $\mathcal{H}$ is also non-interference secure. □

Intuitively, if two automata have conflicting restrictions on shared variables to be non-interference secure, their composition cannot be secure, since the security labels on each variable would clash. This can be seen as refining the conjunction of label constraints on shared variables, and keeping only the combinations where the constraints on the shared variables agree. Table 1 shows how certain labeling assignments can be valid for each individual automata, but are still rejected in the composed system. For instance, choosing x_2 as low forces x_1 to be low, and choosing u as high forces x_1 to be high. This would result in a conflict for the composed system, even though each individual automaton is non-interference secure on its own.

Figure 2 shows a part of the dependency graph of the hybrid system $\mathcal{H}$ in Example 1 indicating the non-interference violation corresponding to the annotations $\{x_1 : high, x_2 : low\}$ (u can be labeled either high or low). We can see that the violation contains only the relevant components that contribute towards the information leak. In the example, the supplied label annotation is indicated by the dashed blue line. This creates a leak, shown as the highlighted red path from high to low. The leak results from the transition component, since the high value can be inferred through the switching condition of the guard expression.

x_1	x_2	u	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_1 \| \mathcal{A}_2$
low	low	low	✓	✓	✓
low	low	high	X	✓	X
low	high	low	✓	✓	✓
low	high	high	X	✓	X
high	low	low	✓	X	X
high	low	high	✓	X	X
high	high	low	✓	✓	✓
high	high	high	✓	✓	✓

Table 1: Non-interference satisfaction of a hybrid system $\mathcal{H}$ in Example 1 corresponding to different security annotations for variables, where ✓: satisfied, X: not satisfied

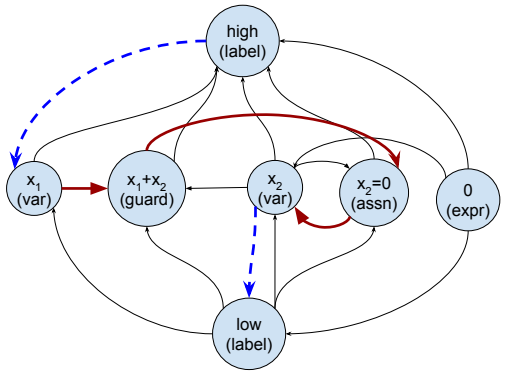

Figure 2: A part of the dependency graph of $\mathcal{H}$ in Example 1 showing the non-interference violation when x_1 is labeled *high*, x_2 is labeled *low*.

Remark 1 (Soundness). *The proposed algorithm is sound as if a hybrid system satisfies the Condition 1, then it is non-interference secure. However, there exists a case that our algorithm may reject non-interference hybrid systems (false positives). For instance, the algorithm will report an assignment $\ell = h * 0$ as a violation. However, such an assignment does not violate non-interference property as ℓ is always equal to 0. As another example, a transition which has the following guard condition and reset action: $h > h + 2; \ell = h$ is also not allowed although it will never be executed. In this paper, we assume that a hybrid automaton does not contain any spurious expression like $h * 0$, $h - h$, or $h > h$.*

4.5 Time Complexity

The run time of our algorithm is linear in the size of the hybrid system. Each component of the model is assigned a unique label,

and all constraints are expressed in terms of those labels. As we have one node per component in the dependency graph, then the total number of nodes is proportional to the model input. Since the algorithm for detecting violations is a graph search, its runtime is linear in the number of nodes and vertices, and therefore linear in terms of the model input.

5 IMPLEMENTATION

We provide an implementation of the information flow analysis written in Haskell that works on SLSF models. Hybrid systems are expressed in SLSF using *hierarchies*, where a model may contain any number of child models executed in parallel. Our tool converts the hierarhical SLSF model representation into an equivalent parallel composition of individual automata that fits our algorithm.

Given a text file containing security annotations for variables in the SLSF model and a .SLX or .MDL file, our tool will output whether or not the model satisfies non-interference with respect to these annotations. In the case of a violation, a detailed error message is printed containing the offending components as a backtrace. The user can use the information in this trace to pinpoint the exact components of the original SLSF model that cause the information leak. They can also provide a partial security annotation. If the tool is not able to fully infer the security labels of all variables, the variables whose security labels cannot be deducted are returned as an error message, and the user is prompted to add more annotations before re-running the analysis.

Our analysis tool is open-source and available on Github[1] for researchers and model designers to use. Currently it supports a restricted set of SLSF functionality that is sufficient to implement hybrid automata matching our definitions in Section 2. Future improvements involve parallelizing the static analysis, as well as providing support for other modeling tools.

6 CASE STUDIES

Currently it is difficult to evaluate information flow analysis on CPS because there are few existing models that have been analyzed for security leaks. We chose the following case studies because they have existing annotations that have been rigorously analyzed and shown to be valid in previous literature. Though there are many existing SLSF models to choose from, they lack an in-depth security analysis, making them unsuitable for evaluating our tool.

We validate that our inexpensive analysis technique detects the same security violations as existing tools. Additionally, the hybrid systems we analyze are written in a standard state machine-based modeling paradigm as opposed to the more obscure language-based representation favored by existing approaches. Furthermore, our tool is able to output the model components that cause a security leak, making it easier for designers to repair the model.

6.1 Gas Pipeline System

The first case study is motivated by the Russia–Ukraine gas disputes according to natural gas export and transit prices in 2005. Russia accused Ukraine of utilizing leaked information during the gas transit to illegally consume gas for domestic purposes without payment [1, 5]. A similar natural gas pipeline system to the one in

[1]https://github.com/gautammohan/hybrid-sectypes

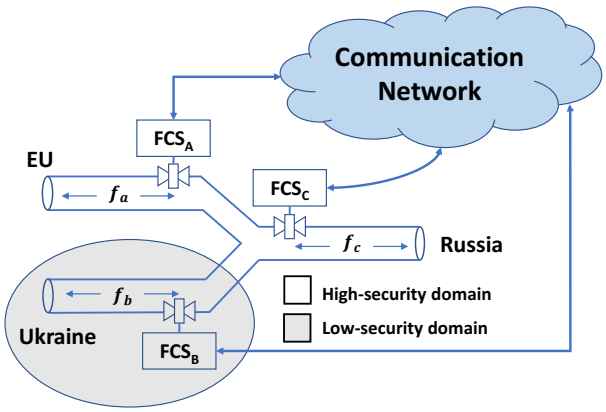

Figure 3: The Russia-Ukraine natural gas grid with subnetworks: C (Russia), B (Ukraine), and A (European Union).

the actual Russia–Ukraine gas dispute is illustrated in Figure 3. The system begins in Russia and divides into two branches in Ukraine, where one transfers gas to the European Union (EU) while the other supplies gas for Ukraine. The transit gas flows are regulated by Flow Controller Systems (FCS). Based on the change of flow demand at Ukraine and the EU, their corresponding FCS_A and FCS_B will send a message to FCS_C to request the increasing or decreasing of flow supplied from C. All three FCSs are located in Ukraine, but Ukraine has physical access only to FCS_B. For simplicity, we assume that 1) the gas flow change at C is a result of either the change of the gas demand at A or B 2) if the gas demand at B (or A) increases, the additional gas can be transferred from A (or B) or C, depending on the gas demand at A (or B), and 3) only the flow change at B is observable.

SLSF model. We model a simplified version of the Russia–Ukraine gas pipeline system as an SLSF diagram shown in Figure 4. The model has four parallel components, each of them represents the physical dynamics of the system and the flow controller at location A, B and C, respectively. Each controller is modeled as a time automaton such that the time instance at which the gas flows in and out at each location is relied on the change of the gas demand at A and B. In the model, Vb and fb are state variables that denote the gas volume and flow rate at B, respectively; an input signal eb represents the change of gas demand at B; zb is a state variable representing the local clock at B; tempB is a state variable captured the value of eb at a time instance; Tcb and Tab are the required time for gas being transferred from C and A to B, respectively. Other notations w.r.t A or C such as Va, Vc can be interpreted in the same fashion as that of B. Since the flow at B is observable, Vb, fb, zb, tempB, and eB are considered as low variables. Other variables are considered as high variables. The safety requirement of the system is that fa + fb $\leq$ fc always holds.

Non-interference detection. Our checker rejects the model as there is an implicit flow from high to low. The update of low state variable such as Vb, fb, zb, and tempB depend on the high input signal ea. Intuitively, if the increased gas demand of B is less than or equal to the decreased gas demand of A, gas will be transferred

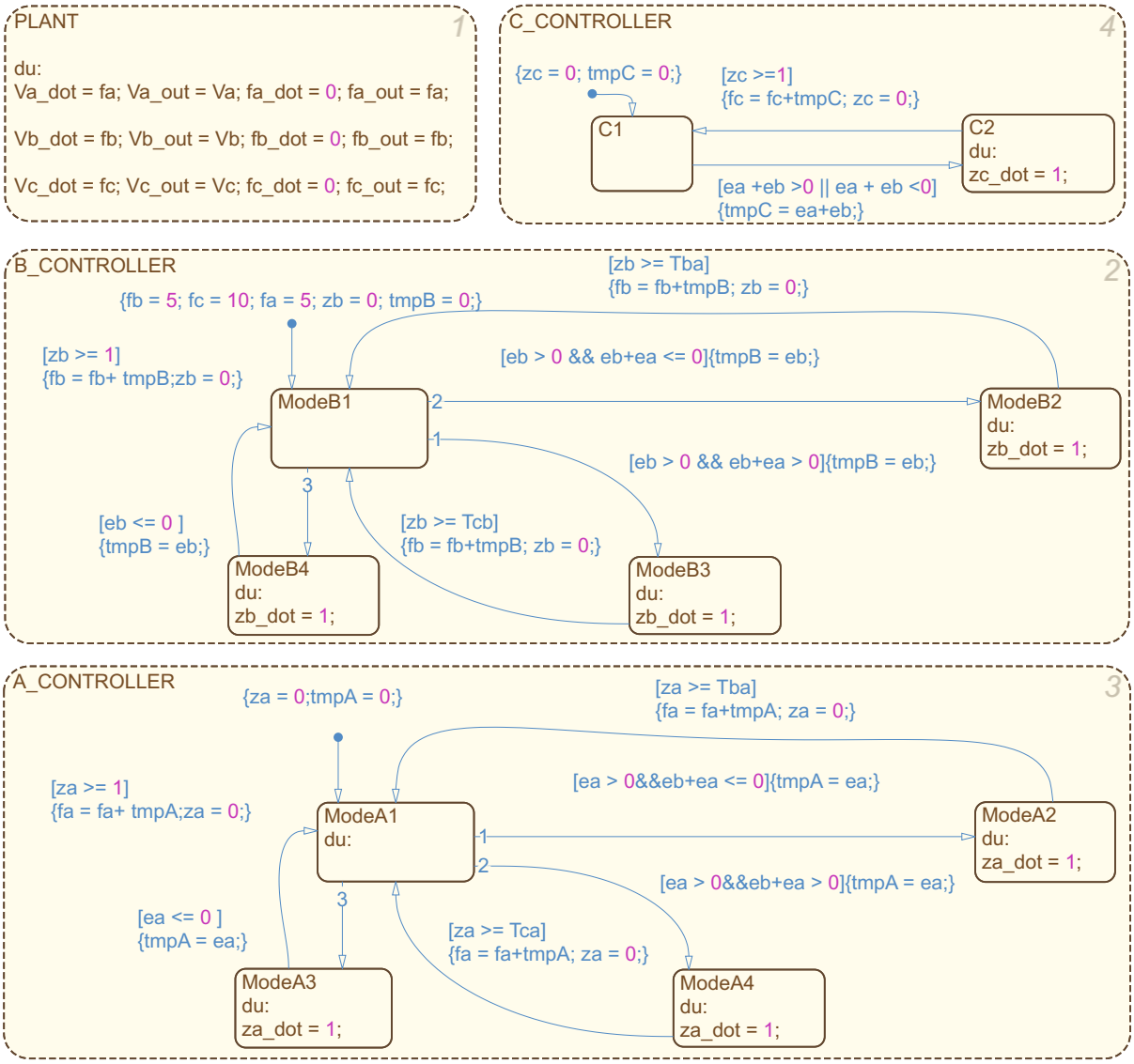

Figure 4: The SLSF model of the simplified example of the Russia-Ukraine natural gas pipeline system.

from A to B. As the transfer time from A to B is less than that of from C to A, i.e., Tab < Tcb, the flow rate into B may be increased at different time instances. Thus, a low-level observer at B can infer the change of gas demand of A based on observing the change of gas demand of B. Based on this knowledge, an observer at B can perform a man-in-the-middle attack to secretly transfer gas from A to B without either A or C being aware of the attack. On the other hand, in the case that the security levels of Vb, fb, zb, tempB, and eB are not given, our tool infers that all of them should be high.

6.2 FREEDM Smart Grid System

In the second case study, we investigate the potential information leakage of the NSF FREEDM smart grid system introduced in [8], and further studied in [1, 4]. Particularly, we focus on the hybrid model of the FREEDM system proposed in [4] which illustrates the migration of power between two neighboring transformers connected to a macrogrid over a shared line. Each transformer has a separate battery that can store and supply power to the transformer. Depending on if the battery is full or not, power is either drawn from or sold back to the macrogrid. In this model, the macrogrid power flow is assumed to be publicly observable.

SLSF model. In this paper, we represented the hybrid-dynamical model proposed in [4] as an SLSF model, which is shown in Figure 5. The model includes three different components that represent the plant dynamics, migration controllers, and battery controllers of two neighboring transformers. For each transformer, $i \in \{1, 2\}$, B_i, b_i are the energy and power of its storage battery. d_i is the gross power demand input, and r_i and p_i are the renewable energy

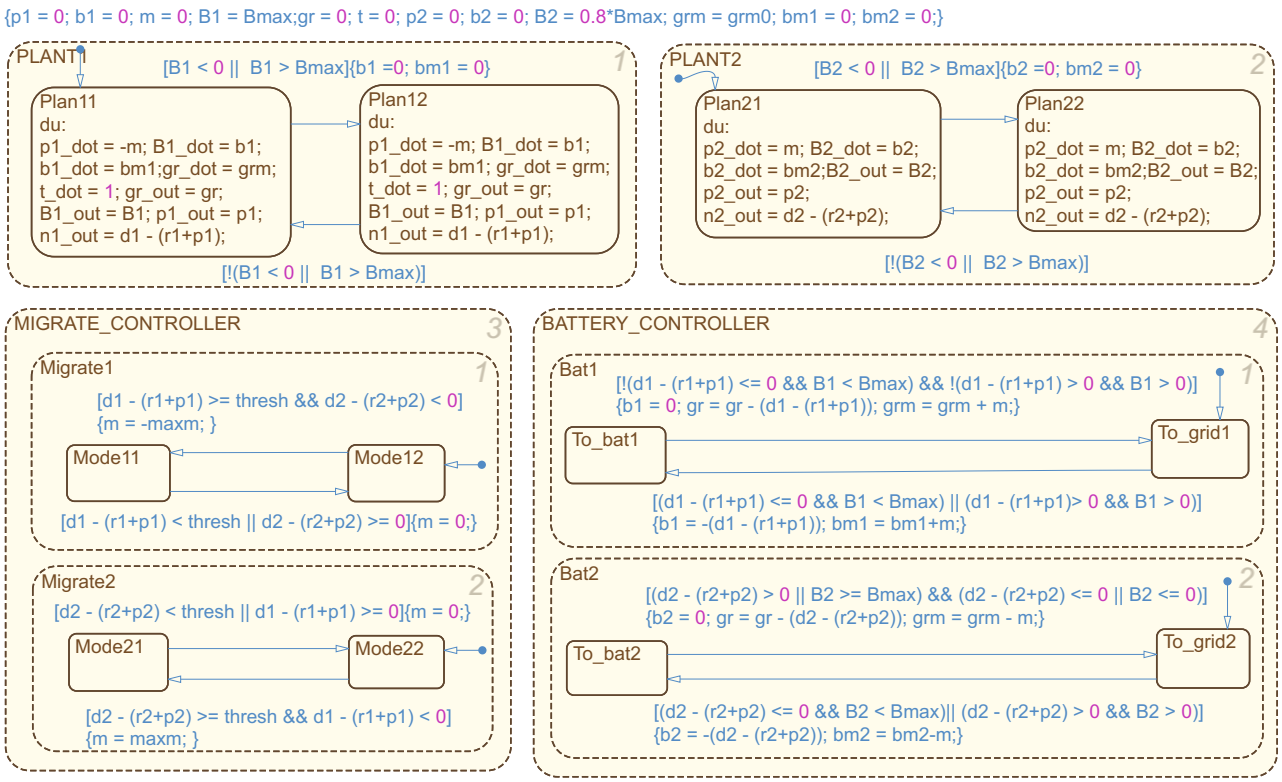

Figure 5: The SLSF model of the FREEDM Smart Grid System proposed in [4].

resource supply and power draw, respectively. *gr* is the power of the macrogrid, and variables which end in *m* denote migration rates. Variables which include *max* are the upper bounds of their values. In this model, we assume that an attacker can observe only the power of the macrogrid, i.e., *gr* is a low variable and other variables are considered high.

Non-interference detection. Our tool reports several security violations in the battery controllers. An explicit flow occurs in the assignment $gr = gr - (d_i - (r_i + p_i))$, where the update of low variable *gr* is based on the values of high variables d_i, r_i and p_i. Furthermore, there is also an implicit flow since the low variable *gr* is updated following the transitions whose guard conditions depend on the status of the battery, which is private. If public observers see that the value of *gr* keeps increasing over time, they can infer that either one or both batteries are at capacity.

7 CONCLUSION

In this paper, we proposed a lightweight approach to detect security leaks in a hybrid system using static information flow analysis. We introduced a set of security rules for hybrid automata that characterizes the non-interference property. Based on those rules, we developed a scalable algorithm that can efficiently perform non-interference checking and inference for hybrid automata corresponding to a given set of security annotations of its variables. We showed that the proposed algorithm is scalable and can be used to analyze a hybrid system without flattening the model. We demonstrated the applicability of our algorithm via two case studies. For future work, we plan to extend our algorithm to capture other information flow properties such as non-inference and non-deducibility.

8 ACKNOWLEDGMENT

The material presented in this paper 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, the National Science Foundation (NSF) GRFP under Grant No. DGE-1845298, and sponsored in part by ONR N000141712012.

REFERENCES

[1] Ravi Akella, Han Tang, and Bruce M McMillin. 2010. Analysis of information flow security in cyber–physical systems. *International Journal of Critical Infrastructure Protection* 3, 3-4 (2010), 157–173.
[2] 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.
[3] Eli Biham and Adi Shamir. 1997. Differential fault analysis of secret key cryptosystems. In *Annual International Cryptology Conference*. Springer, 513–525.
[4] Brandon Bohrer and André Platzer. 2018. A hybrid, dynamic logic for hybrid-dynamic information flow. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 115–124.
[5] Edward Chow and Jonathan Elkind. 2009. Where east meets west: European gas and Ukrainian reality. *The Washington Quarterly* 32, 1 (2009), 77–92.
[6] 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.

[7] Joseph A Goguen and José Meseguer. 1982. Security policies and security models. In *Security and Privacy, 1982 IEEE Symposium on*. IEEE, 11–11.

[8] Alex Q Huang. 2009. Renewable energy system research and education at the NSF FREEDM systems center. In *2009 IEEE Power & Energy Society General Meeting*. IEEE, 1–6.

[9] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. 1998. Side channel cryptanalysis of product ciphers. In *European Symposium on Research in Computer Security*. Springer, 97–110.

[10] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Annual International Cryptology Conference*. Springer, 388–397.

[11] 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.

[12] Paul C Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*. Springer, 104–113.

[13] John McLean. 1994. A general theory of composition for trace sets closed under selective interleaving functions. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*. IEEE, 79–93.

[14] Pavithra Prabhakar and Boris Köpf. 2013. Verifying information flow properties of hybrid systems. In *Proceedings of the 2nd ACM international conference on High confidence networked systems*. ACM, 77–84.

[15] Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. 2004. Tamper resistance mechanisms for secure embedded systems. In *VLSI Design, 2004. Proceedings. 17th International Conference on*. IEEE, 605–611.

[16] Pankaj Rohatgi. 2009. Electromagnetic attacks and countermeasures. In *Cryptographic Engineering*. Springer, 407–430.

[17] Andrei Sabelfeld and Andrew C Myers. 2003. Language-based information-flow security. *IEEE Journal on selected areas in communications* 21, 1 (2003), 5–19.

[18] David Sutherland. 1986. A model of information. In *Proceedings of the 9th national computer security conference*, Vol. 247. Washington, DC, 175–183.

[19] 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.

[20] Jingming Wang and Huiqun Yu. 2014. Analysis of the composition of non-deducibility in cyber-physical systems. *Applied Mathematics & Information Sciences* 8, 6 (2014), 3137.

[21] 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.

[22] Michael W Whalen, David A Greve, and Lucas G Wagner. 2010. Model checking information flow. In *Design and Verification of Microprocessor Systems for High-Assurance Applications*. Springer, 381–428.