Model Checking the Information Flow Security of Real-Time Systems

Christopher Gerking[1](✉), David Schubert[2], and Eric Bodden[1,2]

[1] Heinz Nixdorf Institute, Paderborn University, Paderborn, Germany
christopher.gerking@upb.de
[2] Fraunhofer IEM, Paderborn, Germany

Abstract. Cyber-physical systems are processing large amounts of sensitive information, but are increasingly often becoming the target of cyber attacks. Thus, it is essential to verify the absence of unauthorized information flow at design time before the systems get deployed. Our paper addresses this problem by proposing a novel approach to model-check the information flow security of cyber-physical systems represented by timed automata. We describe the transformation into so-called *test automata*, reducing the verification to a reachability test that is carried out using the off-the-shelf model checker UPPAAL. Opposed to related work, we analyze the real-time behavior of systems, allowing software engineers to precisely identify timing channels that would enable attackers to draw conclusions from the system's response times. We illustrate the approach by detecting a timing channel in a simplified model of a cyber-manufacturing system.

Keywords: Model checking · Information flow · Security · Real time

1 Introduction

Cyber-physical systems [35] are entrusted a fast-growing amount of sensitive data, but are inherently vulnerable to security breaches such as manipulation or leakage of information [15,25]. One subtle attack vector are *timing channels* [11], allowing attackers to infer sensitive information by observing the system's response times. In the worst case, such hidden flows of information could even be exploited to manipulate the physical behavior and compromise the safety of systems. Thus, to make cyber-physical systems secure by design [39], it is essential to verify their *information flow security* before they get deployed.

Model-driven engineering is a widely used approach to securing cyber-physical systems [38], making the software under development accessible to formal verification. A well-established formal definition of information flow security

The stamp on the top of this paper refers to an approval process conducted by the ESSoS Artifact Evaluation Committee.

M. Payer et al. (Eds.): ESSoS 2018, LNCS 10953, pp. 27–43, 2018.
https://doi.org/10.1007/978-3-319-94496-8_3

is *noninterference* [26] which requires that the observable behavior of systems must not depend on secrets. A well-known verification approach for noninterference is *bisimulation*, checking that a system exhibits the same observable behavior as a restricted variant of itself that is known to be secure by definition [19].

Nevertheless, checking the information flow security of cyber-physical systems is a challenging problem for software engineers because they are faced with *real-time systems*, restricted by hard real-time constraints imposed by their physical environment [12]. Therefore, models of cyber-physical systems are based on formalisms like *timed automata* [5], and the real-time behavior of these models needs to be taken into account during verification to detect leaks like timing channels. Verification techniques such as model checking of timed automata are available [4], but involve sophisticated tools that are hard to implement from scratch. In this paper, we therefore address the problem of applying off-the-shelf verification tools to check the information flow security of real-time systems.

Bisimulation of real-time systems is known to be decidable by existing verification techniques [14]. Nevertheless, previous approaches towards applied verification of information flow security have not taken into account real-time behavior [3,17,20,32]. Therefore, they fail to detect leaks such as timing channels precisely. Related work on the information flow security of timed automata exists [6,8,13,34,45], but has spent little effort on how to apply tool-supported model checking techniques in practice. Thus, in summary, none of the previous approaches fully combines real-time behavior with applied model checking.

In this paper, we fill in these gaps by reducing the check for noninterference of timed automata to a *refinement* check, adapting the work by Heinzemann et al. [30] to the application field of information flow security. This check is based on model transformations to construct a *test automaton* [1], introducing a dedicated location that is only reachable when violating a bisimulation between the original automaton and a secure-by-definition variant of itself. By model checking the reachability of these dedicated locations using the off-the-shelf tool Uppaal [9], we obtain a novel verification technique for the information flow security of real-time systems. In contrast to related approaches, our work is based on timed automata, taking into account the real-time behavior of cyber-physical systems. Unlike other related work, we focus on applied model checking to meet the needs of software engineering practitioners.

We illustrate the approach using a simplified model of a cyber-manufacturing system that interacts with a cloud-based service market. The system must not allow market participants to draw any conclusions about business secrets.

In summary, this paper makes the following contributions:

– We propose a model transformation process, reducing the check for information flow security of real-time systems to a model checking problem.

– At the core of this process, we illustrate the construction of test automata to check noninterference of timed automata.
– We give a proof of concept by detecting a timing channel in a simplified model of a cyber-manufacturing system.

Paper Organization: We introduce fundamentals in Sect. 2, and discuss related work in Sect. 3. In Sect. 4, we describe our approach of checking information flow security of real-time systems. We give a proof of concept in Sect. 5, before concluding in Sect. 6.

2 Fundamentals

In this section, we recall timed automata (cf. Sect. 2.1), timed bisimulation (cf. Sect. 2.2), and noninterference (cf. Sect. 2.3). Based on these fundamental concepts, we introduce our motivating example in Sect. 2.4.

2.1 Timed Automata

The formalism of *timed automata* [5] is used to model real-time behavior of stateful systems. A timed automaton is essentially a directed graph containing a finite set of locations, connected by a finite set of labeled edges. We use the definition of timed automata by Bengtsson and Yi [10]. Timed automata extend finite automata by real-valued variables that represent *clocks*. Clocks are initialized with zero, increase at the same rate, and may be set back to zero by using *resets* that can be assigned to edges.

Clock constraints restrict the behavior of an automaton with respect to the valuation of its clocks. A clock constraint is a conjunction of atomic constraints of the form $x \sim n$ or $x - y \sim n$, where x and y are clocks, $n \in \mathbb{N}$, and $\sim \in \{\leq, <, =, >, \geq\}$ [10]. Clock constraints are used as *invariants* and *time guards*. Invariants are assigned to locations. An active location is forced to be left by firing an edge before the location's invariant expires. Therefore, invariants have to be downward closed, i.e., only $\leq$ and $<$ operators are permitted. Time guards are assigned to edges. An edge may fire (i.e., it is *enabled*) only if its time guard evaluates to true. In addition, edges are labeled with actions, whereas firing an edge represents the execution of the action its is labeled with. To represent edges without an action, we refer to τ as the empty action.

Assuming a set $\mathcal{C}$ of clocks, a set $\mathcal{B}(\mathcal{C})$ of clock constraints, and an alphabet Σ of actions, the syntax of a timed automaton is defined as follows [10]:

Definition 1. *A timed automaton A is a tuple $\langle L, l_0, E, I \rangle$ where*

– *L is a finite set of locations,*
– *$l_0 \in L$ is the initial location,*
– *$E \subseteq L \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times L$ is the set of edges where $\rho \in \mathcal{B}(\mathcal{C})$ is the time guard, $\mu \in \Sigma$ is the action, and $\lambda \in 2^{\mathcal{C}}$ is the set of clock resets,*
– *$I : L \to \mathcal{B}(\mathcal{C})$ assigns invariants to locations.*

Uppaal[1] [9] is an off-the-shelf model checker for timed automata that is commonly applied by software engineering practitioners, and frequently integrated into domain-specific verification tools [43]. In the scope of this paper, we apply Uppaal to verify the information flow security of timed automata.

2.2 Timed Bisimulation

Bisimulation is a notion of observational equivalence that requires the observable behavior of two systems to be indistinguishable. *Timed bisimulation* is an extension of bisimulation for real-time systems which is known to be decidable [14]. Intuitively, two systems are equivalent in terms of timed bisimulation if they execute the same sequences of observable actions in the same time intervals. We refer the reader to [10] for a formal definition in the context of timed automata.

There are two variants of timed bisimulation. *Strong* timed bisimulation is more restrictive as it considers all actions of a system as being observable, including τ actions. In the context of our paper, this assumption is too strong because we consider τ actions as internal and, therefore, not observable. In contrast to this, *weak* timed bisimulation ignores the execution of internal τ actions [14]. In the following, we consider only this weak variant of timed bisimulation between two timed automata A and B (denoted by $A \approx B$), and refer to it as timed bisimulation for brevity.

2.3 Noninterference

Noninterference was introduced by Goguen and Meseguer [26] to define information flow security of deterministic finite automata, such that the publicly observable behavior must not depend on sensitive information. If so, public observations never enable an unprivileged actor to distinguish whether or not sensitive information was processed. In particular, no conclusions are possible about *which* sensitive information was actually received. To characterize the sensitivity of information, noninterference is based on a separation between sensitive (or *high*) actions Σ_H and public (or *low*) actions Σ_L with $\Sigma_H, \Sigma_L \subseteq \Sigma$.

For nondeterministic systems such as timed automata, noninterference is frequently defined on the basis of bisimulation [19,34,45]. Noninterference holds if the publicly observable behavior of a system cannot be distinguished from a restricted behavior that is secure by definition. To define this property more precisely, we distinguish between *input actions* received from the environment, and *output actions* sent to the environment. Based on this distinction, noninterference reduces to a bisimulation of the publicly observable behavior between

1. the original system, and
2. a secure-by-definition system with all sensitive input actions *disabled*.

[1] http://uppaal.org

Disabling sensitive input actions ensures that the secure-by-definition system behaves as if no sensitive information is ever processed. By disabling only input actions, we assume that all sensitive information is received from the environment, and never generated internally without depending on sensitive inputs [29].

To identify deviations only in the publicly observable behavior, non-public actions need to be *hidden* from the bisimulation, i.e., treated as non-observable. In the following definition, $\backslash_I$ denotes the disabling of inputs, and $/$ the hiding of actions, whereas $\Sigma_{\bar{L}} = \Sigma \setminus \Sigma_L$ is the set of non-public actions.

Definition 2. *Timed noninterference holds for a timed automaton* A*, if and only if* $A \;/\; \Sigma_{\bar{L}} \approx (A \backslash_I \Sigma_H) \;/\; \Sigma_{\bar{L}}$.

2.4 Motivating Example

In our approach, we assume timed automata to be embedded in component-based software architectures, which are commonly used for the software design of cyber-physical systems [16]. In Fig. 1, we show a software component named ManufacturingSystem as a model of our example announced in Sect. 1. The component embeds a timed automaton that drives the application-level communication between the system and its environment. The communication is carried out by means of asynchronous message passing, whereas the set of messages corresponds to the alphabet Σ. Accordingly, when messages are received, they are buffered until they are processed by an input action of the automaton. Asynchronous communication is a characteristic property of cyber-physical systems because they are often spatially divided and dynamically interconnected over wireless networks. In Fig. 1, we use / to separate input from output actions.

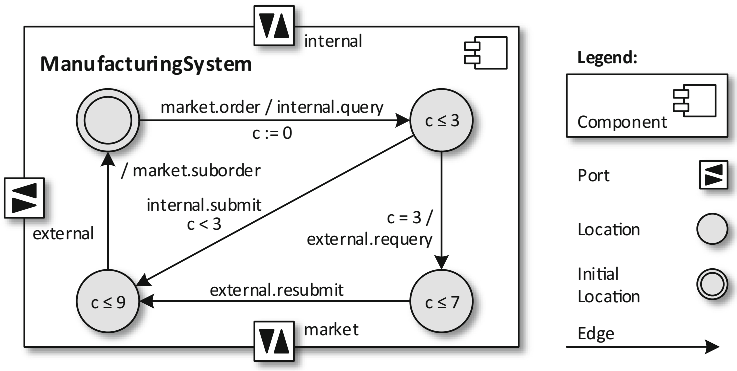


Fig. 1. Example timed automaton of a cyber-manufacturing system

Our example component uses three ports to pass messages. The market port is used to interact with the service market. Whenever a production order is received from the market, the internal port is the preferred way to access the product specification, provided that it is available as an in-house resource. Alternatively, access to the specification may also be purchased from a business partner over the external port. The messages passed over the market port are public (represented by the set Σ_L), whereas the messages passed over the internal port are sensitive (represented by Σ_H). In our example, messages passed over the external port are not further characterized as public or sensitive because information flow needs to be detected only from the internal to the market port.

In the following, we describe the behavior of the automaton from Fig. 1. Whenever the automaton is in its initial location and receives an order message from the market, it sends a query for the product specification over the internal port. At the same time, it sets the clock c to zero which acts as a timeout. If the internal port does not provide the specification in terms of a submit message within three time units (i.e., the company does not possess the specification), a requery message is sent over the external port to purchase the specification from a business partner. In this case, the example system assumes that the specification is provided in terms of a resubmit message within four further time units ($c \leq 7$), i.e., a deadlock caused by an overdue message can never occur. Finally, if the specification is delivered from either the internal or external port, the system orders the corresponding subproducts from the market by sending a suborder message in the time interval $c \leq 9$.

However, the system violates timed noninterference because the effective timing of the public suborder message depends on whether or not the specification is possessed by the company. If possessed, the specification is provided by the sensitive internal port, and suborder may be sent when $c < 3$. Otherwise, when provided by the external port, the suborder message can only be sent when $c \geq 3$. This deviation represents a timing channel that allows market participants to infer whether the company possesses the product specification or not. This knowledge is sensitive information that could be exploited in a future attack to gain access to the specification. Due to the subtleties of real time, such leaks can easily remain undetected during software design, and thus require a tool-supported verification technique.

3 Related Work

In Sect. 3.1, we recall general approaches towards checking information flow security, which are complementary to our work. In Sect. 3.2, we discuss related work on the information flow security of time-dependent systems.

3.1 Complementary Approaches

Unwinding [27] is a traditional verification technique to infer global information flow security from local properties of individual system actions (e.g., state transitions). In the context of real-time systems, this approach is hindered by the infinite, real-valued state space which makes such local properties hard to identify. *Language-based security* [42] is concerned with secure information flow at the level of programming languages, thus using a different model of computation compared to our automata-based approach. In this area, *type systems* are often used to enforce information flow security of programs statically. Furthermore, a technique called *self-composition* has been proposed [7], reducing language-based security to a logical formulation that is amenable to automated verification, similar to our approach in the context of automata-based systems. We refer the reader to [21,37] for a comparison of information flow security under different models of computation. Another complementary approach is the one by Finkbeiner et al. on model checking *hyperproperties* [18]. Unlike standard *safety* or *liveness* properties, hyperproperties relate different executions of a system. Thereby, they cover information flow security properties like noninterference. Whereas hyperproperties involve a novel theory of specification and verification, our focus is on applied verification using off-the-shelf tools.

3.2 Time-Dependent Information Flow Security

In Table 1, we compare related work on the information flow security of time-dependent systems against the core characteristics of our approach, which combines dense real-time behavior with applied verification. Furthermore, we build on automata as the underlying model of computation, which are commonly used as a natural, well-established modeling approach [36]. Finally, according to our example given in Sect. 2.4, we focus on application-level modeling, i.e., we abstract from responsibilities like scheduling.

Table 1. Comparison of related works on time-dependent information flow security

	Dense real-time	Automata-based	Application level	Applied verification
Evans and Schneider [17]	✗	✗	✓	✓
Focardi et al. [20]	✗	✗	✓	✓
Akella et al. [3]	✗	✗	✓	✓
Agat [2]	✗	✗	✓	✗
Giacobazzi and Mastroeni [24]	✗	✗	✓	✗
Rafnsson et al. [40]	✗	✗	✓	✗
Köpf and Basin [32]	✗	✓	✗	✓
Roscoe and Huang [41]	✓	✗	✓	✗
Son and Alves-Foss [44]	✓	✗	✗	✗
Kashyap et al. [31]	✓	✗	✗	✗
Cassez [13]	✓	✓	✓	✗
Lanotte et al. [34]	✓	✓	✓	✗
Benattar et al. [8]	✓	✓	✓	✗
Vasilikos et al. [45]	✓	✓	✓	✗
Barbuti and Tesei [6]	✓	✓	✓	(✓)
This paper	✓	✓	✓	✓

The works by Evans and Schneider [17], Focardi et al. [20], and Akella et al. [3] analyse the security of process algebras. Existing verification techniques like theorem proving [17] or partial model checking [20] are applied, even in the context of cyber-physical systems [3]. By using process algebra, the authors differ from our automata-based approach in terms of their model of computation. In the context of language-based security, the work by Agat [2], Giaccobazzi and Mastroeni [24], as well as Rafnsson et al. [40] uses imperative programs as yet another model of computation. In contrast, the work by Köpf and Basin [32] on synchronous systems is automata-based and also amenable to applied verification. However, all of the above approaches are limited to discrete time, which is insufficient to capture the real-time behavior of cyber-physical systems.

In contrast, other existing approaches consider dense real-time behavior. Roscoe and Huang [41] use process algebra and thereby differ from our automata-based approach. Son and Alves-Foss [44] as well as Kashyap et al. [31] both focus on scheduling of real-time tasks, i.e., do not address the application level.

In the context of timed automata, Cassez [13] presents a real-time security property called *timed opacity* as a generalization of noninterference. The author proves the undecidability of the verification problem, i.e., is not concerned with applied verification. Lanotte et al. [33] consider real-time privacy properties of timed automata, and reduce the verification of such properties to a reachability analysis [28], similar to this paper. In [34], the same authors consider noninterference of timed automata extended by probabilistic behavior. However, the application of existing model checking techniques is beyond the scope of their approach. Benattar et al. [8] enable the synthesis of controllers that ensure noninterference of timed automata. According to this constructive approach, they do not consider applied verification as well. Vasilikos et al. [45] address the security of timed automata that leak some information intentionally. The authors propose an algorithm to impose local security constraints on the elements of an automaton, however, do not enable applied verification using off-the-shelf tools.

Barbuti and Tesei [6] verify noninterference of timed automata. Similar to our approach, they reduce the verification to a reachability analysis using applied model checking. However, their approach only checks that sensitive information does not influence the reachability of locations. This approximation gives rise to both *false positive* and *false negative* errors, and thus is not capable to provide any security guarantee. Nevertheless, the approach by Barbuti and Tesei [6] is the only one that resembles the core characteristics of our paper (cf. Table 1).

4 Checking Noninterference of Timed Automata

In the following, we describe our approach of checking the information flow security of real-time systems. In Sect. 4.1, we give an overview on our approach, and describe the construction of the underlying test automata in Sect. 4.2.

4.1 Refinement Checking

We reduce the verification of timed noninterference to a refinement check for real-time systems as proposed by Heinzemann et al. [30]. The aforementioned work allows to verify refinement relations between real-time systems to check that an abstract behavior is correctly refined by a concrete behavior. The authors reduce the verification to a reachability test [1] that is carried out using model checking techniques. One possible refinement definition is timed bisimulation, as also used to define timed noninterference (cf. Definition 2). Thus, in this paper, we adopt the notion of refinement to check timed noninterference. In Fig. 2, we give an overview on our approach as an extension of the work by Heinzemann et al. [30].

In step ①, we transform a timed automaton A, as described in Sect. 2.4, into an auxiliary automaton A_{sec} that is secure by definition because sensitive inputs are disabled. This restriction corresponds to the automaton $A \setminus_I \Sigma_H$ from Definition 2. We disable sensitive inputs by removing the corresponding edges from A. In the context of the motivating example, Fig. 3a depicts the removal of the edge that processes the submit input over the sensitive internal port.

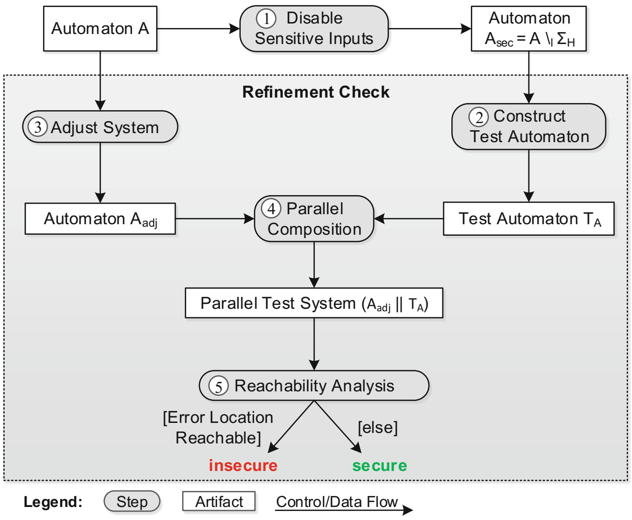


Fig. 2. Reduction of the noninterference check to a refinement check [30]

The resulting automaton A_{sec} enables us to execute a specialized version of the refinement check for timed bisimulation, as proposed by Heinzemann et al. [30]. Thereby, we detect cases where a timed bisimulation between the original automaton A and the secure-by-definition automaton A_{sec} is violated because A deviates from the publicly observable behavior of A_{sec}. At the core of the approach is a *test automaton* [1] that acts as an oracle for the information flow security of the original automaton. In particular, the test automaton detects cases in which the behavior of the original automaton violates timed noninterference. To this end, step ② transforms the automaton A_{sec} into a test automaton T_A, introducing a dedicated *error* location that is reachable when violating timed noninterference by deviating from the secure-by-definition behavior.

As a challenge for the construction of the test automaton, we need to hide all non-public actions from the bisimulation (cf. Definition 2) because only deviations in the public behavior are violations of timed noninterference. In Fig. 3b, we depict those actions that need to be hidden. A natural approach to hide an action is to remove it from the corresponding edge [6,34], i.e., to replace it by a τ action. However, removing input actions may lead to an increased nondeterminism. The reason is that this approach potentially produces multiple τ transitions that are all executable on the same condition because they are no longer distinguishable by their input actions. However, the refinement check proposed by Heinzemann et al. is restricted to systems with a deterministic transition function [30], i.e., at most one edge can fire on a certain condition. Therefore, in contrast to the default test automata for timed bisimulation [30], our test automata must take responsibility for hiding non-public actions. We describe the construction of these specialized test automata in the upcoming Sect. 4.2.

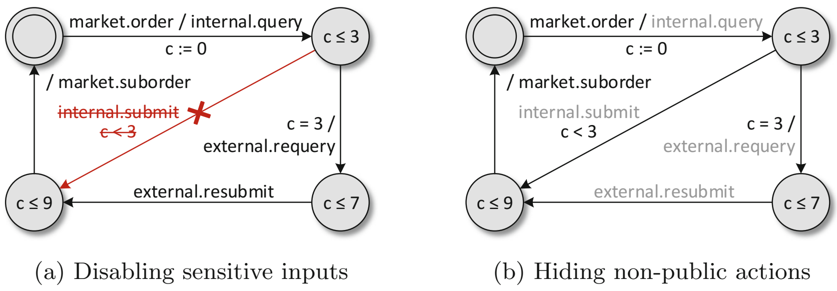


Fig. 3. Disabling and hiding of actions in the motivating example

To ensure that the test automaton acts as the oracle for the original automaton, we need to couple both automata with each other. Therefore, step ③ creates an adjusted automaton A_{adj} that has the same behavior as A. However, it synchronizes with the test automaton whenever both execute the same action. Furthermore, A_{adj} supports the hiding of non-public actions in the same fashion as the test automaton. In step ④, we compose both T_A and A_{adj} in parallel to enable synchronized execution of the automata. In the final step ⑤, the check for timed noninterference reduces to analyzing the resulting parallel test system for reachability of the *error* location. This reachability test [1] is carried out by means of the UPPAAL model checker, using its parallel composition operator $||$ to enable the synchronizations between both automata [10]. In the end, the automaton A is secure in terms of timed noninterference, if and only if the *error* location is unreachable on all execution paths.

4.2 Test Automata Construction

To generate test automata, we adjust the construction schema for timed bisimulation proposed by Heinzemann et al. [30] such that it hides non-public communication, as demanded by Definition 2. We adopt the notion of a dedicated *error* location (named *Err* in our case) that is reachable if and only if timed noninterference is violated. Figure 4 illustrates our construction schema including the *Err* location. We apply this schema for each edge $S \rightarrow S'$ of A_{sec}. Our construction must ensure that T_A will ❶ accept secure communication (allowed by A_{sec} and correctly present in the original automaton A), ❷ reject insecure communication (i.e., public communication that is not allowed by A_{sec} but incorrectly present in A), and ❸ detect the absence of communication (i.e., public communication that is allowed by A_{sec} but incorrectly absent in A). Before going into details about the three cases, we introduce additional notation used in Fig. 4. Synchronous input and output actions are denoted by $\mu?$ and $\mu!$ respectively. The set $\Theta(S)$ includes all actions that are *allowed* in a location S, i.e., all actions of outgoing edges of S. The invariant of S is denoted by $I(S)$. In accordance with Definition 1, ρ is a time guard, and λ is a set of clock resets.

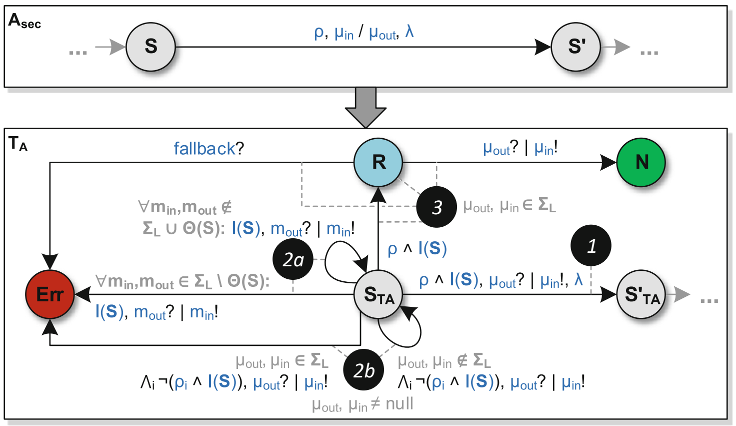

A_sec
...
S
ρ, μin / μout, λ
S'
...
T_A
fallback?
R
μout? | μin!
N
∀min,mout ∉ ΣL ∪ Θ(S): I(S), mout? | min!
3
μout, μin ∈ ΣL
2a
ρ ∧ I(S)
1
Err
∀min,mout ∈ ΣL \ Θ(S): I(S), mout? | min!
S_TA
ρ ∧ I(S), μout? | μin!, λ
S'_TA
...
μout, μin ∈ ΣL
∧i ¬(ρi ∧ I(S)), μout? | μin!
2b
μout, μin ∉ ΣL
∧i ¬(ρi ∧ I(S)), μout? | μin!
μout, μin ≠ null

Fig. 4. Construction schema for test automata

Accepting Secure Communication. The edge labeled with ❶ in Fig. 4 ensures that secure communication is accepted by T_A. For each edge $S \rightarrow S'$ in A_{sec}, we add an edge $S_{TA} \rightarrow S'_{TA}$ to T_A. Asynchronous actions of $S \rightarrow S'$ are transformed into synchronous actions to enable synchronization with A_{adj} when processing the same messages during parallel composition. Here, a synchronous output action $\mu_{in}!$ is created from an asynchronous input action μ_{in}, or a synchronous input action $\mu_{out}?$ is created from an asynchronous output action μ_{out}. The set of clock resets λ is transferred to $S_{TA} \rightarrow S'_{TA}$. Finally, we need to preserve the time intervals in which actions are executed. To that end, the time guard of $S_{TA} \rightarrow S'_{TA}$ is the conjunction of the original time guard ρ and the invariant $I(S)$ of S. Thus, $S_{TA} \rightarrow S'_{TA}$ is enabled whenever $S \rightarrow S'$ is enabled.

Rejecting Insecure Communication. Insecure communication includes two cases: 2a executing a public action that is not allowed in a location S because no outgoing edge is labeled with it, and 2b executing a public action that is allowed in S, but violates the timing defined in A_{sec}. For case 2a, Fig. 4 introduces an edge $S_{TA} \rightarrow Err$ for each public action m_{in} or m_{out} that is not allowed in S, i.e., for all actions in $\Sigma_L \setminus \Theta(S)$. The time guard $I(S)$ of these edges ensures that Err is reachable only by actions executed during the activity of S. In contrast to the construction by Heinzemann et al. [30], only public actions make Err reachable. For the opposite case of non-public communication, we add a loop $S_{TA} \rightarrow S_{TA}$ for each message that is not in $\Sigma_L \cup \Theta(S)$. Thereby, instead of switching to the Err location, T_A hides all non-public actions that are not allowed.

To handle the timing violations of case 2b, we create one more edge $S_{TA} \rightarrow Err$ labeled with the allowed action μ_{out} or μ_{in} of $S \rightarrow S'$. However, this edge is enabled exactly at those times when A_{sec} does not allow the action. To this end, the time guard of the edge is the conjunction of the negated enabling conditions for all edges $S \rightarrow S^i$ in A_{sec} that execute the same action as $S \rightarrow S'$. The resulting time guard is $\bigwedge_i \neg(\rho_i \wedge I(S))$ where ρ_i is the time guard of $S \rightarrow S^i$. Thereby, we ensure that the edge can fire when exceeding the upper bounds of the time intervals in which the action is allowed to execute, or when falling below the lower bounds. In contrast to Heinzemann et al. [30], the edge leads to Err only in case of public messages from Σ_L. For non-public messages, analogously to case 2a, we construct a loop $S_{TA} \rightarrow S_{TA}$. This edge has the same enabling conditions, however, hides the corresponding non-public action instead of switching to Err.

Detecting Absent Communication. Timed noninterference demands that all public actions executable by A_{sec} are executable by the original automaton A in the same time intervals. To check such restrictions, Heinzemann et al. [30] add the constructs labeled with 3 in Fig. 4. We adopt this construction only if μ_{out} or μ_{in} is public (i.e., in Σ_L). In this case, the location R represents a check for required communication. Due to the time guard $\rho \wedge I(S)$, it is reachable during the full time interval in which the edge $S \rightarrow S'$ is enabled.

If A preserves the time interval in which A_{sec} can execute the public action, then the edge $R \rightarrow N$ is enabled whenever R is entered. The location N represents a neutral state of the analysis that is reachable whenever the required public action is properly executed. N has no outgoing edges because the execution does not have to be further explored from here. Instead, the location S'_{TA} is always reachable when N is reachable and ensures regular execution.

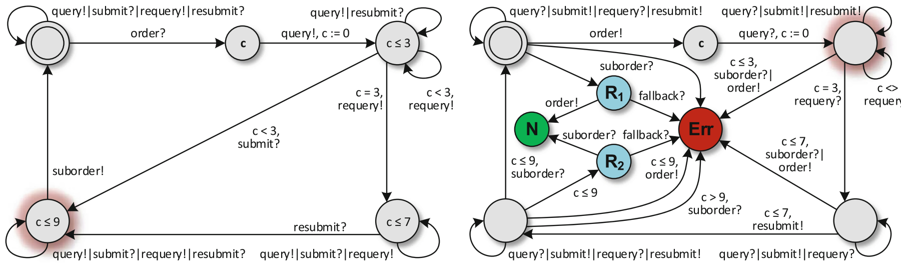
If at some time during its interval, the required public action can not be executed (because A lowers the upper bound or raises its lower bound), then $R \rightarrow N$ is not enabled. In this case, the edge $R \rightarrow Err$ fires by synchronizing over an auxiliary channel named $fallback$. Synchronization over this channel is always enabled, however, it has the lowest priority compared to all other channels used for communication. Thereby, *Err* is only reachable when the required public communication is absent.

5 Proof of Concept

In this section, we showcase the utility of our approach in the scope of the example given in Sect. 2.4. To this end, we demonstrate that our technique outperforms the related work by detecting a timing channel that remains undetected using the approach by Barbuti and Tesei [6]. We show that our technique rejects the insecure system, but accepts a mitigated system that is noninterferent.

To verify the information flow security of timed automata, Barbuti and Tesei check that disabling sensitive actions does not affect the reachability of locations. In the scope of our example, the corresponding transformation was shown in Fig. 3a. Clearly, disabling the sensitive submit action does not affect the reachability of locations (compared to Fig. 3b) because all locations are still reachable. Thus, the timing channel described in Sect. 2.4 remains undetected by the approach, and therefore represents a *false negative* because the insecure system is regarded as secure.

In contrast, Fig. 5 illustrates the parallel composition of the adjusted automaton A_{adj} (Fig. 5a) and the test automaton T_a (Fig. 5b) as proposed in this paper. As an artifact for reproduction, we provide a corresponding model that is verifiable by the UPPAAL model checker [23]. Due to lack of space, Fig. 5 merges multiple edges between the same source and target locations into a single edge with alternative synchronization labels. Furthermore, we omit the names of ports over which messages are sent or received. Finally, since edges with a conjunction of actions are not allowed in UPPAAL, we use a *committed* location [10] (labeled with c in Fig. 5) to divide the order and query actions into two consecutive edges. Figure 5 also depicts the additional loops added to both automata for hiding non-public communication, as described for the test automata in Sect. 4.2.

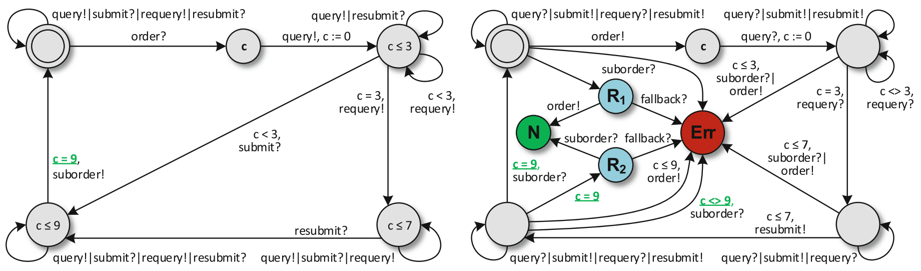


(a) Adjusted automaton A_{adj} (b) Test automaton T_a

Fig. 5. Parallel test system for the motivating example

In the situation depicted in Fig. 5, the system has already processed the message sequence order, query, submit (in the time interval $c < 3$). Since sensitive inputs are disabled in T_a, it can only execute a loop when processing the sensitive submit message. Next, the public suborder message to be sent by A_{adj} corresponds to case 2a of our construction in Sect. 4.2. Thus, T_a will regard the message as insecure, and reject it by switching to the Err location. The reason for this violation is that A_{adj} sends the public suborder message too early, i.e., in the time interval $c \in [0, 3]$. Thus, in its current location (cf. Fig. 5b), T_a regards the message as not allowed and switches to Err.

Figure 6 shows the countermeasures taken to mitigate the timing channel. In Fig. 6a, we depict the time guard $c = 9$ added to delay the suborder message. Consequently, the timing of the message does no longer depend on whether or not the product specification was provided over the sensitive internal port. Figure 6b depicts the resulting changes of the test automaton. Since the timing of the suborder message is now fixed, the Err location is no longer reachable. Accordingly, our construction correctly identifies the mitigated system as noninterferent.



(a) Adjusted automaton A_{adj} (b) Test automaton T_a

Fig. 6. Mitigation of the timing channel in the motivating example

6 Conclusions and Future Work

This paper proposes a novel check for the information flow security of real-time systems given in the form of timed automata. Our approach is based on noninterference as a well-established definition of secure information flow. To provide a verification technique that applies existing tools and takes into account real-time behavior, we adapt the work on refinement checking by Heinzemann et al. [30] to the field of security. We describe the construction of test automata, introducing a dedicated location that indicates violations of noninterference whenever it is reachable during execution. Thereby, we reduce the problem to a reachability test that is supported by model checking techniques used in software engineering practice. In particular, we apply the well-established Uppaal model checker for timed automata as our underlying verification engine. Our proof of concept demonstrates the advantages of our approach by detecting a timing channel that would remain undetected using the most closely related work.

The proposed idea provides software engineering practitioners with a tool-supported verification technique for the information flow security of timed automata, taking into account specific characteristics of cyber-physical systems like real-time behavior and asynchronous communication. Thereby, we enable engineers to identify information leaks such as timing channels early, and ensure security by design. For cyber-physical systems, this is of vital importance to avoid product recalls or even safety-critical attacks.

Our approach is part of ongoing work on tracing information flow security in cyber-physical systems engineering [22]. In future work, we will provide tool support for our approach in the context of a model-driven software design method for cyber-physical systems. In particular, to check the information flow security of hierarchical component architectures, our work needs to be extended to a compositional verification approach. Thereby, we seek to preserve security when composing overall software systems from single secure components.

Acknowledgments. The authors would like to thank Johannes Geismann and Marie Christin Platenius for helpful comments on drafts of this paper.

References

1. Aceto, L., Burgueño, A., Larsen, K.G.: Model checking via reachability testing for timed automata. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 263–280. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054177
2. Agat, J.: Transforming out timing leaks. In: POPL 2000, pp. 40–53. ACM (2000)
3. Akella, R., Tang, H., McMillin, B.M.: Analysis of information flow security in cyber-physical systems. Int. J. Crit. Infrastruct. Prot. **3**(3–4), 157–173 (2010)
4. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking in dense real-time. Inf. Comput. **104**(1), 2–34 (1993)
5. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
6. Barbuti, R., Tesei, L.: A decidable notion of timed non-interference. Fundamenta Informaticae **54**(2–3), 137–150 (2003)
7. Barthe, G., D'Argenio, P.R., Rezk, T.: Secure information flow by self-composition. Math. Struct. Comput. Sci. **21**(6), 1207–1252 (2011)
8. Benattar, G., Cassez, F., Lime, D., Roux, O.H.: Control and synthesis of non-interferent timed systems. Int. J. Control **88**(2), 217–236 (2015)
9. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: UPPAAL—a tool suite for automatic verification of real-time systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) HS 1995. LNCS, vol. 1066, pp. 232–243. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0020949
10. Bengtsson, J., Yi, W.: Timed automata: semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27755-2_3
11. Biswas, A.K., Ghosal, D., Nagaraja, S.: A survey of timing channels and countermeasures. ACM Comput. Surv. **50**(1), 6:1–6:39 (2017)
12. Broman, D., Derler, P., Eidson, J.: Temporal issues in cyber-physical systems. J. Indian Inst. Sci. **93**(3), 389–402 (2013)

13. Cassez, F.: The dark side of timed opacity. In: Park, J.H., Chen, H.-H., Atiquzzaman, M., Lee, C., Kim, T., Yeo, S.-S. (eds.) ISA 2009. LNCS, vol. 5576, pp. 21–30. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02617-1_3
14. Čerāns, K.: Decidability of bisimulation equivalences for parallel timer processes. In: von Bochmann, G., Probst, D.K. (eds.) CAV 1992. LNCS, vol. 663, pp. 302–315. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56496-9_24
15. Chattopadhyay, A., Prakash, A., Shafique, M.: Secure cyber-physical systems: current trends, tools and open research problems. In: DATE 2017, pp. 1104–1109. IEEE (2017)
16. Crnkovic, I., Malavolta, I., Muccini, H., Sharaf, M.: On the use of component-based principles and practices for architecting cyber-physical systems. In: CBSE 2016, pp. 23–32. IEEE (2016)
17. Evans, N., Schneider, S.: Analysing time dependent security properties in CSP using PVS. In: Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M. (eds.) ESORICS 2000. LNCS, vol. 1895, pp. 222–237. Springer, Heidelberg (2000). https://doi.org/10.1007/10722599_14
18. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 30–48. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_3
19. Focardi, R., Gorrieri, R.: A taxonomy of security properties for process algebras. J. Comput. Secur. **3**(1), 5–34 (1995)
20. Focardi, R., Gorrieri, R., Martinelli, F.: Real-time information flow analysis. IEEE J. Sel. Areas Commun. **21**(1), 20–35 (2003)
21. Focardi, R., Rossi, S., Sabelfeld, A.: Bridging language-based and process calculi security. In: Sassone, V. (ed.) FoSSaCS 2005. LNCS, vol. 3441, pp. 299–315. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31982-5_19
22. Gerking, C.: Traceability of information flow requirements in cyber-physical systems engineering. In: CEUR Workshop Proceedings, DocSym@MoDELS 2016, vol. 1735 (2016)
23. Gerking, C.: Detection of a timing channel in an UPPAAL model of a cyber-manufacturing system (2018). https://doi.org/10.5281/zenodo.1034024
24. Giacobazzi, R., Mastroeni, I.: Timed abstract non-interference. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 289–303. Springer, Heidelberg (2005). https://doi.org/10.1007/11603009_22
25. Giraldo, J., Sarkar, E., Cárdenas, A.A., Maniatakos, M., Kantarcioglu, M.: Security and privacy in cyber-physical systems: a survey of surveys. IEEE Des. Test **34**(4), 7–17 (2017)
26. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE S&P, pp. 11–20. IEEE (1982)
27. Goguen, J.A., Meseguer, J.: Unwinding and inference control. In: IEEE S&P, pp. 75–87. IEEE (1984)
28. Gorrieri, R., Lanotte, R., Maggiolo-Schettini, A., Martinelli, F., Tini, S., Tronci, E.: Automated analysis of timed security. Int. J. Inf. Secur. **2**(3–4), 168–186 (2004)
29. Guttman, J.D., Nadel, M.E.: What needs securing. In: CSFW, pp. 34–57. MITRE Corporation Press (1988)
30. Heinzemann, C., Brenner, C., Dziwok, S., Schäfer, W.: Automata-based refinement checking for real-time systems. Comput. Sci. - R&D **30**(3–4), 255–283 (2015)

31. Kashyap, V., Wiedermann, B., Hardekopf, B.: Timing- and termination-sensitive secure information flow. In: IEEE S&P, pp. 413–428. IEEE (2011)
32. Köpf, B., Basin, D.: Timing-sensitive information flow analysis for synchronous systems. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 243–262. Springer, Heidelberg (2006). https://doi.org/10.1007/11863908_16
33. Lanotte, R., Maggiolo-Schettini, A., Tini, S.: Privacy in real-time systems. Electron. Notes Theor. Comput. Sci. **52**(3), 295–305 (2001)
34. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Time and probability-based information flow analysis. IEEE Trans. Softw. Eng. **36**(5), 719–734 (2010)
35. Lee, E.A.: CPS foundations. In: DAC 2010, pp. 737–742. ACM (2010)
36. van der Meyden, R., Zhang, C.: Algorithmic verification of noninterference properties. Electron. Notes Theor. Comput. Sci. **168**, 61–75 (2007)
37. van der Meyden, R., Zhang, C.: A comparison of semantic models for noninterference. Theor. Comput. Sci. **411**(47), 4123–4147 (2010)
38. Nguyen, P.H., Ali, S., Yue, T.: Model-based security engineering for cyber-physical systems. Inf. Softw. Technol. **83**, 116–135 (2017)
39. Peisert, S., Margulies, J., Nicol, D.M., Khurana, H., Sawall, C.: Designed-in security for cyber-physical systems. IEEE Secur. Priv. **12**(5), 9–12 (2014)
40. Rafnsson, W., Jia, L., Bauer, L.: Timing-sensitive noninterference through composition. In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 3–25. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_1
41. Roscoe, A.W., Huang, J.: Checking noninterference in timed CSP. Formal Asp. Comput. **25**(1), 3–35 (2013)
42. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE J. Sel. Areas Commun. **21**(1), 5–19 (2003)
43. Schivo, S., Yildiz, B.M., Ruijters, E., Gerking, C., Kumar, R., Dziwok, S., Rensink, A., Stoelinga, M.: How to efficiently build a front-end tool for UPPAAL: a model-driven approach. In: Larsen, K.G., Sokolsky, O., Wang, J. (eds.) SETTA 2017. LNCS, vol. 10606, pp. 319–336. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69483-2_19
44. Son, J., Alves-Foss, J.: A formal framework for real-time information flow analysis. Comput. Secur. **28**(6), 421–432 (2009)
45. Vasilikos, P., Nielson, F., Nielson, H.R.: Secure information release in timed automata. In: Bauer, L., Küsters, R. (eds.) POST 2018. LNCS, vol. 10804, pp. 28–52. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89722-6_2