Int J Inf Secur (2004) 2: 168–186 / Digital Object Identifier (DOI) 10.1007/s10207-004-0037-9

# Automated analysis of timed security: a case study on web privacy

**Roberto Gorrieri**[1], **Ruggero Lanotte**[2], **Andrea Maggiolo-Schettini**[3], **Fabio Martinelli**[4], **Simone Tini**[2], **Enrico Tronci**[5]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy
e-mail: gorrieri@cs.unibo.it
[2] Dipartimento di Scienze della Cultura, Politiche e dell'Informazione, Università dell'Insubria, Via Valleggio 11, 22100 Como, Italy
[3] Dipartimento di Informatica, Università di Pisa, Via Buonarroti 2, 56127 Pisa, Italy
[4] Istituto di Informatica e Telematica, C.N.R. di Pisa, Via Giuseppe Moruzzi 1, 56124 Pisa, Italy
[5] Dipartimento di Informatica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy

Published online: 9 June 2004 – © Springer-Verlag 2004

**Abstract.** This paper presents a case study on an automated analysis of real-time security models. The case study on a web system (originally proposed by Felten and Schneider) is presented that shows a timing attack on the privacy of browser users. Three different approaches are followed: LH-Timed Automata (analyzed using the model checker HyTech), finite-state automata (analyzed using the model checker NuSMV), and process algebras (analyzed using the model checker CWB-NC). A comparative analysis of these three approaches is given.

**Keywords:** Web privacy – Model checking tools – Timed behavior

## 1 Introduction

Privacy is one of the most relevant aspects of security, as it requires that secret information not be disclosed to third parties. Several papers [1, 6, 12, 14, 25, 27, 31, 33] dealing with privacy consider *two-level* systems, where the *high-level* (or *secret*) behavior is distinguished from the *low-level* (or *observable*) one. In the above papers, systems respect privacy if there is no information flow from the high level to the low level. This means that the secret behavior cannot influence the observable one, or, equivalently, no information on the observable behavior is permitted to infer information about the secret one. Some work has been done to automate this approach to information flow security (e.g., [13]), with some encouraging results.

This approach is to be extended to cope with realistic systems, where concrete information about timing of events is represented. Some work in the direction of studying security properties on real-time systems has been developed by some of the authors in [14, 15, 22], where the addition of the real-time information opens new, sometimes unexpected, sources of information flows (e.g., the so-called *timing* covert channels). By *timed security* we mean "security in a real-time setting". In effect, it may happen that no information leakage is captured if one considers the untimed specification of a given system, whereas the analysis of the timed behavior of the same system reveals information leakage. In fact, some "malicious" users may exploit temporal properties of the behavior of the system to compromise its security by inferring protected information. Timing attacks are studied in several frameworks as Internet security, cryptography, and communication protocols [11, 17, 20, 32].

The main aim of this paper is twofold: on the one hand, we intend to show that timed security analysis can be automated with the help of well-known tools for real-time systems; on the other hand, we intend to develop a real-life case study (about web privacy) on three different timed security approaches in order to offer some comparison results.

*Structure of the paper.* In Sect. 2 we present the case study on web privacy originally considered in [11].

In Sect. 3 we consider dense time temporal behaviors and use *LH-Timed Automata* [22] to model the case study as well as the given privacy property on timed traces. We use HyTech [18, 37] to automatically verify our model with respect to the given property.

In Sect. 4 we consider discrete-time temporal behaviors. We use *finite-state automata* to model the case study and *CTL logic* [9] to define the given privacy property. We use NuSMV [10, 35] to automatically verify our model with respect to the given property.

In Sect. 5 we use the process algebra *TCCS* [38] to model the case study. We use CWB-NC [38] to automatically verify our model w.r.t. the given property. The

Research partially supported by MIUR Progetto Cofinanziato Metodi Formali per la Sicurezza e il Tempo (MEFISTO).

process algebra approach considers a more general definition of security, encompassing also privacy, and uses bisimulation-based semantics.

In Sect. 6 we compare the three approaches above, and, finally, in Sect. 7 we draw some conclusions.

## 2 The case study

We consider the timing attack on web privacy described in [11], where the privacy of a user's web-browsing histories can be violated by a malicious site that is able to observe the responsiveness of the user's browser. More precisely, the malicious site, named $e$, is able to determine whether or not the user $u$ has recently visited some other, unrelated, web page $w$. A Java applet is embedded in the malicious site $e$ and, on visiting it, such an applet is downloaded and run by the user's browser (first step in Fig. 1). The applet first performs a request of the web page $w$ (second step in Fig. 1) and then performs a request to a page $e_1$ of the malicious site (third step in Fig. 1). Page $e_1$ can be accessed only by running the applet, thus implying that it cannot be in the cache of the user's browser. Hence, the malicious site can measure the time elapsed between the original request by the user and the request caused by the applet. If such a time is under a certain bound, the malicious site infers that the page $w$ was already in the cache of the browser of the user, thus implying that $w$ has been recently visited by the user.

In Fig. 2 we sketch the system composed by the user, the browser, the site handling $w$, the cache, and the applet. The site $e$, which interacts with the system, is also sketched in the figure. The several components of the system can be described as follows:

- The user can request to the browser either the web page $w$ (action $req_w$) or a web page of the malicious site $e$ (action $req_e$). Then, the user waits until the requested web page is shown ($show_w$ or $show_e$). Note that, for our purposes, we do not need to model interaction between the user and other web sites.
- If the browser receives from the user a request for a page of the malicious site $e$ ($req_e$), it asks the page to load $e$ ($load_e$), and when the page is received ($receive_e$), the browser shows it ($show_e$). If the browser receives from the user a request for page $w$ ($req_w$), the browser tries to download the page from the cache ($look_c$). If $w$ is in the cache, the cache gives a positive answer ($yes_c$) and the page is shown ($show_w$). Otherwise, the cache gives a negative answer ($no_c$), the browser downloads page $w$ (actions $load_w$ and $receive_w$) from the site handling it, and the page is cached ($write_c$) and shown ($show_w$). Finally, if the browser receives from the applet a request for page $e_1$ of the malicious site $e$ ($req_{e_1}$), it asks the page to load $e$ ($load_{e_1}$), and when the page is received ($receive_{e_1}$), the browser shows it ($show_{e_1}$).

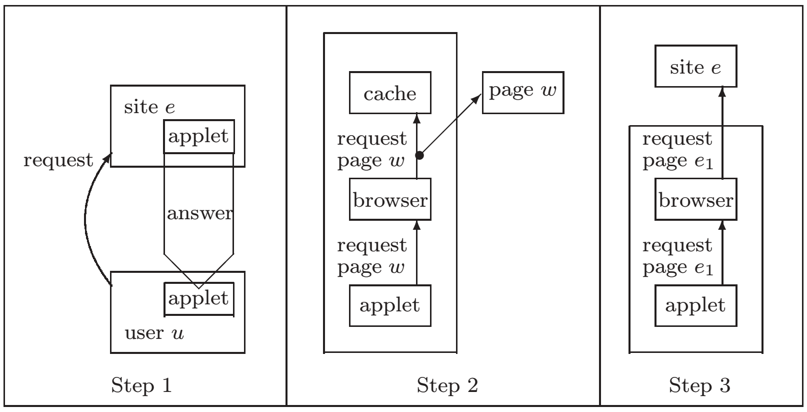


**Fig. 1.** Role of the applet

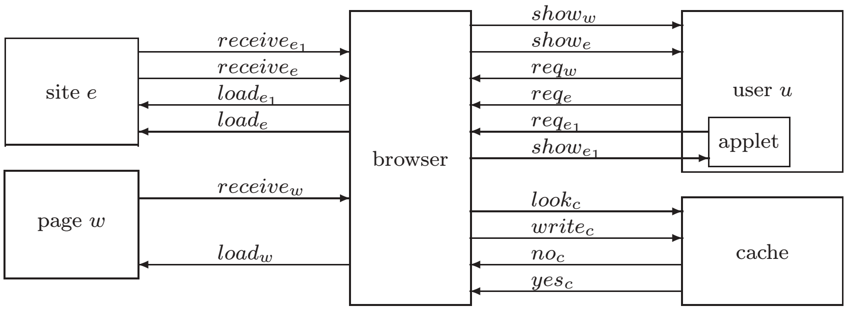


**Fig. 2.** Interaction among parties

- The site handling $w$ receives a request ($load_w$) for web page $w$ and then gives an answer ($receive_w$). We assume that the time elapsing between $load_w$ and $receive_w$ is in the interval $[10, 25]$, meaning that at least 10 time units must pass but that no more than 25 time units can pass before $receive_w$.
- When a page in $w$ is requested by the user ($look_c$) and the page is not yet in the cache, the cache gives a negative answer (action $no_c$). In this case, the browser downloads the page, which is cached (action $write_c$). We assume that if the page is not requested for a time greater than 100, then the page is removed from the cache, and that when the page is in the cache, the time elapsed between a request $look_c$ and an answer $yes_c$ is in the interval $[2, 5]$.
- The Java applet starts its execution when the browser shows the page $e$, i.e., after an action $show_e$. Then the applet forces the browser to download page $w$ (actions $req_w$ and $show_w$) and, subsequently, page $e_1$ from $e$ ($req_{e_1}$ and $show_{e_1}$).

It is reasonable to assume that the malicious site can observe only $load_e$, $receive_e$, $load_{e_1}$, and $receive_{e_1}$, whereas the other actions representing interactions between the browser and the cache and between the browser and $w$ cannot be seen.

Now, if the malicious site $e$ receives the original request $load_e$ and the request caused by the applet $load_{e_1}$ within 10 units of time, it infers that no communication between the user and the site handling $w$ has happened in the meantime, i.e., that web page $w$ was in the cache of the user. In fact, the browser takes at least 10 units of time to download page $w$ from the site handling it.

Now, intervals $[10, 25]$ and $[2, 5]$ between $load_w$ and $read_w$ and between $look_c$ and $yes_c$, respectively, do not affect our analysis, that is, it suffices, and is reasonable, to consider that, to have the attack, the upper bound of the second interval is less than the lower bound of the first interval.

From now on we use the names $A_u$, $A_b$, $A_w$, $A_c$, and $A_a$ to denote, respectively, the user, the browser, the site handling $w$, the cache, and the Java applet.

## 3 The LH-Timed Automata approach

The formalism of LH-Timed Automata [22] is an extension of Alur and Dill's timed automata [2] suitable for modeling two-level systems and for dealing with problems of privacy. LH-Timed Automata are timed automata with the alphabet of the actions partitioned into two sets, the set $H$ of *high actions* and the set $L$ of *low actions*. Moreover, LH-Timed Automata offer an operation of synchronous parallel composition of automata.

### *3.1 The formalism*

Let us recall the definition of the formalism.

**Definition 1.** *Given a set of high actions $H$ and a set of low actions $L$, an* LH-Timed Automaton *is a tuple $\mathcal{A} = (A_1, \ldots, A_m)$, where, for each $1 \leq i \leq m$, $A_i = (L_i, H_i, Q_i, q_i^0, X_i, \delta_i)$ is a* sequential automaton, *with:*

- *A* security alphabet $(L_i, H_i)$, *with $\bigcup_{1\leq i\leq m} L_i = L$ and $\bigcup_{1\leq i\leq m} H_i = H$;*
- *A finite set of* states $Q_i$ *such that $Q_1, \ldots, Q_m$ are pairwise disjoint;*
- *An* initial state $q_i^0 \in Q_i$*;*
- *A set of* clocks $X_i$ *such that $X_1, \ldots, X_m$ are pairwise disjoint;*
- *A set of* transitions $\delta_i \subseteq Q_i \times \Phi(X_i) \times (L_i \cup H_i) \times 2^{X_i} \times Q_i$, *where $\Phi(X_i)$ is the set of* constraints *over the set of clocks $X_i$ that are generated by the following grammar:*

$$\phi ::= x\#c \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid true \ ,$$

*where $\phi \in \Phi(X_i)$, $x \in X_i$, $\# \in \{<, \leq, >, \geq, =, \neq\}$, and $c$ is a constant.*

As an example, the process $A_u$, representing the behavior of the user, is described by the sequential automaton in Fig. 3. The process $A_b$, representing the behavior of the browser, the process $A_w$, representing the behavior of the site handling the page $w$, and the process $A_c$ representing the behavior of the cache, are described in Fig. 4. Finally, the process $A_a$, representing the applet, is described in Fig. 5. Hence, the web system is described by the LH-Timed Automaton $\mathcal{A} = (A_u, A_b, A_c, A_w, A_a)$, where $L = \{load_e, receive_e, load_{e1}, receive_{e1}\}$ and all the other actions are in $H$.

Let us now explain the behavior of $\mathcal{A}$.

A transition $(q, \phi, a, Y, q')$ in $A_i$ fires when state $q$ is active, the value of the clocks in $A_i$ satisfies the constraint $\phi$, and action $a \in L_i \cup H_i$ is performed. In such a case, state $q'$ is entered and the clocks in $Y$ are reset.

A *configuration* of the LH-Timed Automaton $\mathcal{A}$ is a tuple $s = ((q_1, v_1), \ldots, (q_m, v_m))$ such that, for each $1 \leq i \leq m$, $q_i$ is a state in $Q_i$ and $v_i$ is a clock valuation over clocks $X_i$ (i.e., a mapping from $X_i$ to time values). The *initial configuration* $s_0$ is $((q_1^0, v_1^0), \ldots, (q_m^0, v_m^0))$, with $q_i^0$ the initial state of $A_i$ and $v_i^0$ the valuation such that $v_i^0(x) = 0$ for each clock $x \in X_i$.

The LH-Timed Automaton $\mathcal{A}$ performs a *step* from a given configuration $s = ((q_1, v_1), \ldots, (q_m, v_m))$ to a given configuration $s' = ((q_1', v_1'), \ldots, (q_m', v_m'))$ at time $t$ with

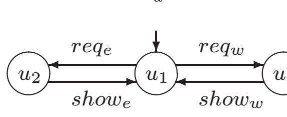


**Fig. 3.** The user

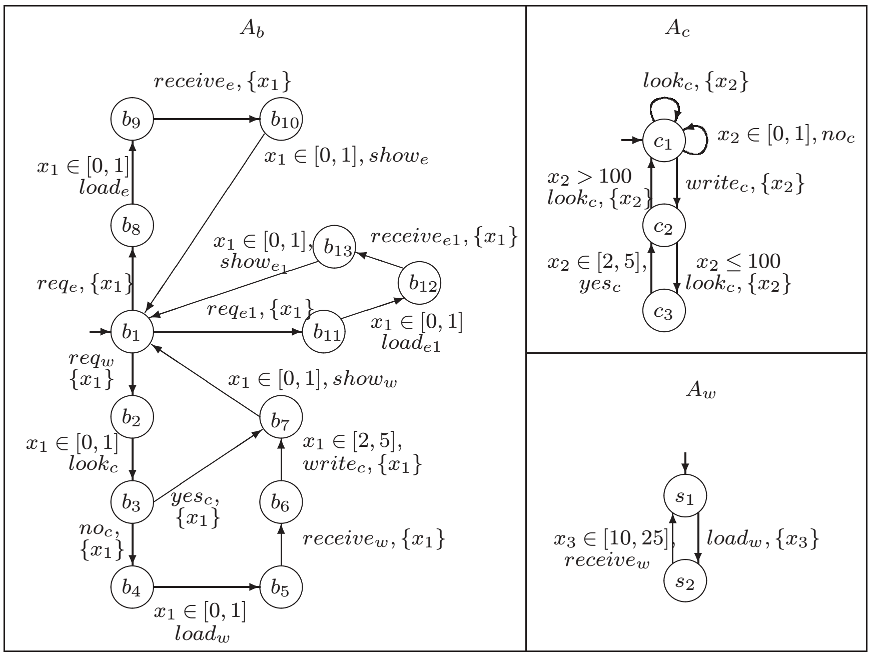


**Fig. 4.** The browser, the cache, and the site handling $w$

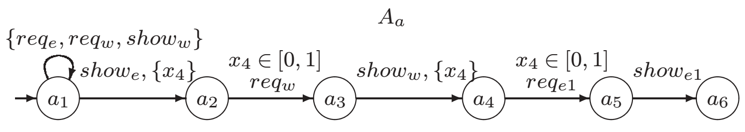


**Fig. 5.** The Java applet

action $a \in H \cup L$, written $s \to_t^a s'$, iff, for each $1 \leq i \leq m$, one of the following two properties holds:

– $a \in L_i \cup H_i$ and there is a transition $(q_i, \phi_i, a, Y_i, q_i') \in \delta_i$ such that:
  – The constraint $\phi_i$ is satisfied by the clock valuation $v_i + t$ obtained from $v_i$ by adding the value $t$ to the valuation of each clock;
  – $v_i'$ is obtained from $v_i + t$ by resetting the clocks in $Y_i$.

– $a \notin L_i \cup H_i$, $q_i' = q_i$ and $v_i'$ is the valuation $v_i + t$.

The LH-Timed Automaton $\mathcal{A}$ accepts *timed words*

$$\omega = (a_0, t_0), (a_1, t_1), (a_2, t_2) \dots ,$$

with $a_i$ an action in $L \cup H$ and $t_i$ a value in a given time domain $T$. A timed word $\omega = (a_0, t_0), (a_1, t_1), \dots$ is *accepted* by $\mathcal{A}$ if there exists an infinite sequence of steps $s_0 \to_{t_0}^{a_0} s_1 \to_{t_1}^{a_1} \dots$ from the initial configuration $s_0$. The *language* accepted by $\mathcal{A}$ (denoted by $\mathcal{L}(\mathcal{A})$) is the set of timed words accepted by $\mathcal{A}$. Intuitively, $\omega$ describes the behavior of a system that performs action $a_i$ at time $\sum_{h=0}^{i} t_h$. We denote with $\omega_L$ the projection of $\omega$ on $L$, namely, the (possibly finite) sequence $(l_{i_1}, \sum_{l=0}^{i_1} t_l), (l_{i_2}, \sum_{l=i_1+1}^{i_2} t_l), \dots$ such that, for each index $i_j$, $l_{i_j} = a_{i_j} \in L$, and for each $i_j < k < i_{j+1}$, $a_{i_j} \in H$. The sequence $\omega_L$ describes the observable part of $\omega$. Moreover, we denote by $F_\omega$ the function that gives the index in $\omega$ of the low action in position $j$ in $\omega_L$, namely, $F_\omega(j) = i_j$.

## 3.2 The No-privacy property

Given sequences $d$ and $d'$, let $d \leq_P d'$ denote the fact that $d$ is a prefix of $d'$.

Let $h \in H$, $d$ be a finite sequence $(l_1, t_1), \dots, (l_n, t_n)$ with $l_1, \dots, l_n \in L$ and $t_1, \dots, t_n \in T$, and $i$ be an index $0 \leq i < n$. We define the No-privacy property $NPr(d, i, h)$ for an LH-Timed Automaton $\mathcal{A}$ as follows:

$$\forall \omega \in \mathcal{L}(\mathcal{A}), d \leq_P \omega_L \Rightarrow h \in \{a_{F_\omega(i)+1}, \dots, a_{F_\omega(i+1)-1}\}.$$

Intuitively, $NPr(d, i, h)$ expresses that, whenever the sequence $d$ of low symbols is read, the high symbol $h$ is read between the low-level actions $l_i$ and $l_{i+1}$, and therefore there is an information flow from high level to low level. Consider the observable sequence $d_1 = (load_e, 30)(receive_e, 10)(load_{e1}, 8)$. Then, the property $NPr(d_1, 2, yes_c)$ holds. This means that whenever 8 units of time elapse between the first answer from $e$ ($receive_e$) and the request caused by the applet ($load_{e_1}$), the malicious site is sure that the requested web page $w$ is in the cache of the user.

On the contrary, if we take the low sequence $d_2 = (load_e, 30)(receive_e, 10)(load_{e1}, 20)$, then the No-privacy property $NPr(d_2, 2, yes_c)$ does not hold, since the delay of 20 units of time between the first answer from $e$ ($receive_e$) and the request caused by the applet ($load_{e_1}$) is due to the communication between the user and the site handling $w$. Actually, we have that $NPr(d_2, 2, no_c)$ holds. The fact that $NPr(d_2, 2, yes_c)$ does not hold and $NPr(d_2, 2, no_c)$ holds is not a contradiction. In fact, the No-privacy reveals information flows with respect to a certain symbol. Since $NPr(d_2, 2, yes_c)$ does not hold, we can deduce that $yes_c$ does not cause any problem for $d_2$, but this fact does not imply that there is no attack for $d_2$.

### *3.3 HyTech*

HyTech is a tool for the automated analysis of embedded systems [3, 18, 37].

HyTech's input consists of a text file containing a system description and a list of iterative analysis commands.

#### 3.3.1 System description

The system description consists in the specification of variables representing clocks and the specification of all sequential automata running in parallel. In Fig. 7 we specify clocks x_1, x_2, x_3, x_4, and x_5 by

```
var x_1,x_2,x_3,x_4,x_5 : clock
```

and in Figs. 6, 7, and 8 we specify the automata of Figs. 3, 4, and 5, respectively.

Each automaton has a name and a set of synchronization labels, corresponding to LH-Timed Automata's actions. As an example, in Fig. 7 the following declaration gives the name of the automaton "browser" and its synchronization labels:

```
automaton user
synclabs : req_e, show_e, req_w, show_w;

initially u_1 ;

loc u_1: while True wait {}
         when True sync req_e goto u_2;
         when True sync req_w goto u_3;
loc u_2: while True wait {}
         when True sync show_e goto u_1;
loc u_3: while True wait {}
         when True sync show_w goto u_1;

end -- user
```

**Fig. 6.** The user

```
-- the-web-system
var       x_1,x_2,x_3,x_4,x_5,

automaton browser
synclabs : req_e,load_e,receive_e,show_e,req_e1,load_e1,
           receive_e1,show_e1, req_w, load_w, receive_w,
           show_w, look_c, yes_c, no_c,  write_c;
initially b_1 & x_1=0;
loc b_1: while True wait {}
         when True sync req_w do {x_1'=0} goto b_2;
         when True sync req_e do {x_1'=0} goto b_8;
         when True sync req_e1 do {x_1'=0} goto b_11;
loc b_2: while True wait {}
         when x_1<=1 & x_1 >=0 sync look_c goto b_3;
loc b_3: while True wait {}
         when True sync yes_c do {x_1'=0} goto b_7;
         when True sync no_c do {x_1'=0} goto b_4;
loc b_4: while True wait {}
         when x_1<= 1 & x_1>=0  sync load_w goto b_5;
loc b_5: while True wait {}
         when True sync receive_w do {x_1'=0} goto b_6;
loc b_6: while True wait {}
         when x_1<= 5 & x_1>=2  sync write_c do {x_1'=0}
         goto b_7;
loc b_7: while True wait {}
         when x_1>=0 & x_1<=1 sync show_w  goto b_1;
loc b_8: while True wait {}
         when x_1>=0 & x_1<=1 sync load_e  goto b_9;
loc b_9: while True wait {}
         when True sync receive_e do {x_1'=0} goto b_10;
loc b_10: while True wait {}
          when x_1>=0 & x_1<=1 sync show_e  goto b_1;
loc b_11: while True wait {}
          when x_1>=0 & x_1<=1 sync load_e1  goto b_12;
loc b_12: while True wait {}
          when True sync receive_e1 do {x_1'=0} goto b_13;
loc b_13: while True wait {}
          when x_1>=0 & x_1<=1 sync show_e1  goto b_1;
end -- browser

automaton cache
synclabs: yes_c, look_c, write_c, no_c;
initially c_1 & x_2=0;
loc c_1: while True wait {}
         when True sync look_c do {x_2'=0} goto c_1;
         when x_2>=0 & x_2 <= 1  sync no_c goto c_1;
         when True sync write_c do {x_2'=0} goto c_2;
loc c_2: while True wait {}
         when x_2 <= 100 sync look_c do {x_2'=0} goto c_3;
         when x_2 > 100  sync look_c do {x_2'=0} goto c_1;
loc c_3: while True wait {}
         when x_2 <= 5 & x_2>=2 sync yes_c goto c_2;
end -- cache

automaton siteW
synclabs: load_w, receive_w;
initially s_1 & x_3=0;
loc s_1: while True wait {}
         when True sync load_w do {x_3'=0} goto s_2;
loc s_2: while True wait {}
         when x_3 <= 25 & x_3 >= 10 sync receive_w goto s_1;
end -- siteW
```

**Fig. 7.** The browser, the cache, and the site handling $w$

```
automaton browser
synclabs : req_e, load_e, receive_e, show_e,
           req_e1, load_e1, receive_e1,
           show_e1, req_w, load_w, receive_w,
           show_w, look_c, yes_c, no_c,
           write_c;
```

For each automaton we must specify the initial configuration. As an example, in Fig. 7, by

```
initially b_1 & x_1=0
```

we have specified that the browser has the state $b_1$ as initial state, and that the clock x_1 is initially set to zero.

```
automaton applet
synclabs: req_e, show_e,req_e1, show_e1, req_w,show_w;

initially a_1 & x_a=0;

loc a_1: while True wait {}
         when True sync show_e do {x_4'=0} goto a_2;
         when True sync req_e   goto a_1;
         when True sync req_w   goto a_1;
         when True sync show_w  goto a_1;
loc a_2: while True wait {}
         when x_4>=0 & x_4 <=0 sync req_w  goto a_3;
loc a_3: while True wait {}
         when True sync show_w do {x_4'=0}  goto a_4;
loc a_4: while True wait {}
         when  x_4>=0 & x_4 <=0 sync req_e1  goto a_5;
loc a_5: while True wait {}
         when True sync show_e1  goto a_6;
loc a_6: while True wait {}
         when True  goto a_6;

end -- applet
```

**Fig. 8.** The Java applet

Now, besides describing the states and the transitions of each automaton, in HyTech one can also specify invariants that must hold when the automaton is in a certain state. In our case each invariant is simply equal to $True$, since LH-Timed Automata do not offer invariants. Moreover, in HyTech one can define variables with rates. In our case all variables are clocks, and so we do not need to specify the rate. As an example, in Fig. 7 with

```
loc b_6: while True wait {}
```

we specify that the invariant of state b_6 is $True$ and that the set of rates of variables is empty.

The following line of code

```
 when x_1<= 5 & x_1>=2  sync write_c do
     {x_1'=0} goto b_7;
```

specifies that, if the condition x_1<= 5 $\wedge$ x_1>=2 is verified, and the action write_c is performed, then the clock x_1 is reset and the automaton enters state b_7.

### 3.3.2 Checking No-privacy

In [22] we proved that the property $NPr(d, i, h)$ is decidable.

More precisely, an algorithm is given that takes an LH-Timed Automaton $\mathcal{A}$ and a property $NPr(d, i, h)$ and checks whether $NPr(d, i, h)$ holds or not. In the present paper, instead of using such an algorithm, we reduce the verification problem to a reachability problem. More precisely, we synthesize an automaton that recognizes the timed words with low part equal to $d$ and not performing $h$ between the low symbol at the $i^{th}$ step and the low symbol at the $(i+1)^{th}$ step. It is obvious that $NPr(d, i, h)$ holds if and only if the automaton resulting from the intersection of the given automaton $\mathcal{A}$ and the synthesized one does not have an empty set of runs.

### 3.3.3 Analysis commands

Let us consider the low sequence $d_1 = (load_e, 30)(receive_e, 10)(load_{e1}, 8)$ and the property $NPr(d_1, 2, yes_c)$.

In Fig. 9 we translate this problem into a reachability problem. To this end, we need first to add the automaton `behavior` with synchronization labels (actions) *load_e*, *receive_e*, *load_e*1, and *yes_c*. The automaton `behavior` recognizes the sequences with low part equal to $(load_e, 30)(receive_e, 10)(load_{e1}, 8)$ and where action $yes_c$ does not appear between $receive_e$ and $load_{e1}$. Hence, if the automaton `behavior` runs in parallel with the system specified in Figs. 6, 7, and 8, then the overall system recognizes at least one sequence iff $NPr(d_1, 2, yes_c)$ does not hold.

We declare variables corresponding to regions (a region [2] is a set of configurations). The variable `init_reg` represents the region with the initial configuration of the overall system. The variable `final_reg` represents the configurations where automaton `behavior` is in the state r_4. If the automaton `behavior` reaches this state, then there exists a sequence with low part equal to $d_1$ and where $yes_c$ does not appear between $receive_e$ and $load_{e1}$. The variable `reached` represents the configurations from which one can reach a configuration among those expressed by `final_reg`. The definition of variable

```
automaton behavior
synclabs: load_e, receive_e, load_e1,  yes_c;

initially r_1 & x_5=0;

loc r_1: while True wait {}
         when True sync yes_c  goto r_1;
         when x_5=30 sync load_e do {x_5'=0} goto r_2;
loc r_2: while True wait {}
         when x_5=10 sync receive_e do {x_5'=0} goto r_3;
loc r_3: while True wait {}
         when x_5=8 sync load_e1 do {x_5'=0} goto r_4;
loc r_4: while True wait {}
         when True goto r_4;

end -- behavior

var
   init_reg, final_reg, reached : region;

init_reg := loc[user]=u_1 & loc[browser]=b_1 &
            loc[cache]=c_1 & loc[siteW]=s_1 &
            loc[applet]=a_1 & loc[behavior]=r_1 &
            x_1=0 & x_2=0 & x_3=0 & x_4=0 & x_5=0 ;

final_reg := loc[behavior]=r_4;

reached := reach backward from final_reg endreach;

if empty(reached & init_reg)
   then prints "Npriv holds ";
   else prints "Npriv does not hold";
   print trace to init_reg using reached;
endif;
```

**Fig. 9.** $NPr(d_1, 2, yes_c)$

```
..........Number of iterations required for reachability: 11
NPriv holds
```

**Fig. 10.** Answer to $NPr(d_1, 2, yes_c)$

```
automaton behavior
synclabs: load_e, receive_e,load_e1,  yes_c;

initially r_1 & x_5=0;

loc r_1: while True wait {}
         when True sync yes_c goto r_1;
         when x_5=30 sync load_e do {x_5'=0} goto r_2;
loc r_2: while True wait {}
         when x_5=10 sync receive_e do {x_5'=0} goto r_3;
loc r_3: while True wait {}
         when x_5=20 sync load_e1 do {x_5'=0} goto r_4;
loc r_4: while True wait {}
         when True goto r_4;

end -- behavior


var
   init_reg, final_reg, reached : region;

init_reg := loc[user]=u_1 & loc[browser]=b_1 &
            loc[cache]=c_1 & loc[siteW]=s_1 &
            loc[applet]=a_1 & loc[behavior]=r_1 &
            x_1=0 & x_2=0 & x_3=0 & x_4=0  & x_5=0;

final_reg := loc[behavior]=r_4 ;

reached := reach backward from final_reg endreach;

if empty(reached & init_reg)
   then prints "Npriv holds ";
   else prints "Npriv does not hold";
   print trace to init_reg using reached;
endif;
```

**Fig. 11.** $NPr(d_2, 2, yes_c)$

`reached` uses the command `reach backward`, which performs a backward search on regions, applied to variable `final_reg`.

Now, the condition `(reached & init_reg)` selects the regions with the configurations that are initial and from which one can perform a sequence with low part equal to $d_1$ and where $yes_c$ does not appear between $receive_e$ and $load_{e1}$. Hence, $NPr(d_1, 2, yes_c)$ holds iff `(reached & init_reg)` selects the empty set of regions. Now, operator `empty` applied to `(reached & init_reg)` returns true iff `(reached & init_reg)` selects the empty set of regions. In such a case, HyTech prints "Npriv holds", otherwise it prints "Npriv does not hold". The answer given by HyTech is shown in Fig. 10. It is computed in 0.11 s.

Let us take $d_2 = (load_e, 30)(receive_e, 10)(load_{e1}, 20)$ and the property $NPr(d_2, 2, yes_c)$. In Fig. 11 we translate this problem into a reachability problem. The answer given by HyTech is shown in Fig. 12. It is computed in 0.16 s. In Fig. 13 an error trace is shown for $NPr(d_2, 2, yes_c)$.

```
..................Number of iterations required
                  for reachability: 20
NPriv does not hold
```

**Fig. 12.** Answer to $NPr(d_2, 2, yes_c)$

```
===== Generating trace to specified target region =======
Location: u_1.b_1.c_1.s_1.a_1.r_1
------------------------------
 VIA 29 time units and req_e
------------------------------
Location: u_2.b_8.c_1.s_1.a_1.r_1
------------------------------
 VIA 1 time units and load_e
------------------------------
Location: u_2.b_9.c_1.s_1.a_1.r_2
------------------------------
 VIA 10  time units and receive_e
------------------------------
Location: u_2.b_10.c_1.s_1.a_1.r_3
------------------------------
 VIA: show_e
------------------------------
Location: u_1.b_1.c_1.s_1.a_2.r_3
------------------------------
 VIA: req_w
------------------------------
Location: u_3.b_2.c_1.s_1.a_3.r_3
------------------------------
 VIA: look_c
------------------------------
Location: u_3.b_3.c_1.s_1.a_3.r_3
------------------------------
 VIA: no_c
------------------------------
Location: u_3.b_4.c_1.s_1.a_3.r_3
------------------------------
 VIA: load_w
------------------------------
Location: u_3.b_5.c_1.s_2.a_3.r_3
--------------
 VIA 18 time units and receive_w
------------------------------
Location: u_3.b_6.c_1.s_1.a_3.r_3
--------------
 VIA 2 time units and  write_c
------------------------------
Location: u_3.b_7.c_2.s_1.a_3.r_3
------------------------------
 VIA: show_w
------------------------------
Location: u_1.b_1.c_2.s_1.a_4.r_3
------------------------------
 VIA: req_e1
------------------------------
Location: u_1.b_11.c_2.s_1.a_5.r_3
------------------------------
 VIA: load_e1
------------------------------
Location: u_1.b_12.c_2.s_1.a_5.r_4
=========== End of trace generation ===========
```

**Fig. 13.** Error trace for $NPr(d_2, 2, yes_c)$

## 4 The NuSMV approach

In this section we consider a discrete-time version of the No-privacy property defined in the previous section. Although discretization causes a loss of precision, we aim to understand whether discretization can make automatic verification more efficient. For this purpose, we use the model checker NuSMV [10, 35], which works with finite-state systems.

NuSMV is a reimplementation and extension of SMV [9, 24, 36], which is the first model checker based on OBDDs (*ordered binary decision diagrams*) [8, 9]. The input language of NuSMV is the same as that of SMV. Thus the same model can be given as input to both model checkers. We used NuSMV here since for our verification problem it turned out to be more efficient than SMV.

### *4.1 Basic notation*

Here we briefly describe the NuSMV formal frame. We refer the reader to [9] for further details.

In NuSMV we can define finite-state processes with their transition relation. More precisely, for each process $P$, we define a boolean expression (also named $P$) defining the state transition relation of $P$ and a boolean expression $I$ defining the set of initial states for $P$.

We use C-like identifiers to denote variables ranging on states of $P$. We use $'$ to denote the next state operator. Thus, if the variable $\mathbf{x}$ ranges over the states of $P$, then $\mathbf{x}'$ ranges oover the next states of $P$. Since NuSMV models (as usual for Kripke structures) only have the notion of state, *actions* are just seen as parts of *states.* The pair $(\mathbf{x}, \mathbf{a})$ gives the overall state of $P$. The boolean expression $P(\mathbf{x}, \mathbf{a}, \mathbf{x}', \mathbf{a}')$, with $L \cup H$ being the domain of variables $\mathbf{a}$ and $\mathbf{a}'$, defines the transition relation of process $P$. The boolean expression $I(\mathbf{x}, \mathbf{a})$ defines the set of initial states for $P$.

For example, the set of initial states and the transition relation for the automaton in Fig. 3 can be defined, respectively, with the following expressions:
$I(\mathbf{x}, \mathbf{a}) = (\mathbf{x} = u_1)$,
$P(\mathbf{x}, \mathbf{a}, \mathbf{x}', \mathbf{a}') = ((\mathbf{x} = u_2) \wedge (\mathbf{a} = \mathit{show}_e) \wedge (\mathbf{x}' = u_1)) \wedge ((\mathbf{x} = u_1) \wedge (\mathbf{a} = \mathit{req}_w) \wedge (\mathbf{x}' = u_3)) \wedge ((\mathbf{x} = u_3) \wedge (\mathbf{a} = \mathit{show}_w) \wedge (\mathbf{x}' = u_1)) \wedge ((\mathbf{x} = u_1) \wedge (\mathbf{a} = \mathit{req}_e) \wedge (\mathbf{x}' = u_2))$.

In the example above, no constraint on $\mathbf{a}'$ is imposed by $P(\mathbf{x}, \mathbf{a}, \mathbf{x}', \mathbf{a}')$. This is a general fact when modeling processes using a state-based approach. In fact, at each transition step, $\mathbf{a}$ can take nondeterministically any value. This models correctly the fact that we do not choose the *next* action. For this reason we write $P(\mathbf{x}, \mathbf{a}, \mathbf{x}')$ rather than $P(\mathbf{x}, \mathbf{a}, \mathbf{x}', \mathbf{a}')$ when denoting the transition relation of a process (an action labeled automaton). Moreover, we write $I(\mathbf{x})$ instead of $I(\mathbf{x}, \mathbf{a})$ since the initial conditions do not contain any constraint on $\mathbf{a}$.

We use *synchronous* parallel composition. For processes $P(\mathbf{x}, \mathbf{a}, \mathbf{x}')$ and $Q(\mathbf{x}, \mathbf{a}, \mathbf{x}')$, their composition is represented by $P(\mathbf{x}, \mathbf{a}, \mathbf{x}') \wedge Q(\mathbf{x}, \mathbf{a}, \mathbf{x}')$.

We will use arithmetic operators freely in our boolean expressions. They can be translated into boolean operators as in an ALU.

### *4.2 The No-privacy property*

Of course the No-privacy property can be formalized within the NuSMV frame following the No-privacy definition given in Sect. 3.2. Anyway, to help the reader understand NuSMV coding of No-privacy, in this section we show how No-privacy can be formalized within the NuSMV frame.

Let $P(\mathbf{x}, \mathbf{a}, \mathbf{x}')$, and $I(\mathbf{x})$ be the expressions defining, respectively, the transition relation and the set of initial states of a given process $P$.

An $(I,\ P)$ sequence for $P$ is a sequence of states $\omega = (\mathbf{x}_0, \mathbf{a}_0),\ (\mathbf{x}_1, \mathbf{a}_1),\ \dots$ such that: $I(\mathbf{x}_0) = \mathtt{true}$ and, for all $i = 0, 1, \dots, P(\mathbf{x}_i, \mathbf{a}_i, \mathbf{x}_{i+1}) = \mathtt{true}$. An $(I, P)$ sequence can be finite as well as infinite.

We denote by $\omega_L$ the projection of $\omega$ on $L$. That is, $\omega_L$ is the sequence of states $(\mathbf{x}_{i_1}, \mathbf{l}_{i_1}),\ (\mathbf{x}_{i_2}, \mathbf{l}_{i_2}),\ \dots$, where, for each index $i_j$, $l_{i_j} = a_{i_j} \in L$, and, for each $i_j < k < i_{j+1}$, $a_k \in H$. We denote by $F_\omega$ the function that gives the index in $\omega$ of the low action in position $j$ in $\omega_L$, namely, $F_\omega(j) = i_j$.

When modeling with *finite-state automata* (FSA) there is no explicit notion of time (as, e.g., in HyTech; see Sect. 3). Thus it is the *modeler*'s responsibility to correctly model *time* (e.g., using counters) for the specific application. Of course, only discrete time can be modeled using FSA. Since in the following we want to measure the time elapsed between two consecutive low actions, without loss of generality we assume that such a measure is available in our model (i.e., in our process transition relation $P$) and can be *read* from the state variables of $P$. For this reason we partition $P$ state variables $\mathbf{x}$ into two disjoint parts: $Loc(\mathbf{x})$ and $Time(\mathbf{x})$ s.t. $\mathbf{x} = (Loc(\mathbf{x}), Time(\mathbf{x}))$. We call *location* $Loc(\mathbf{x})$ and *timer* $Time(\mathbf{x})$. Timer $Time(\mathbf{x})$ denotes the (discrete) time elapsed since the last low action. Location $Loc(\mathbf{x})$ contains all state variables that are not used to define the timer. Of course we use the above state partition for the sake of simplicity. The actual NuSMV model can be organized in more efficient ways that, however, are conceptually equivalent to the above schema. We usually denote by the letter $\mathbf{q}$ (with or without subscripts/superscripts) locations and by the letter $\mathbf{t}$ (with or without subscripts/superscripts) timers.

An *$L$ spy sequence of length $n$* for $P$ is a sequence $<\mathbf{l}_1,\ [t^1_{min},\ t^1_{max}),\ \mathbf{l}_2,\ [t^2_{min},\ t^2_{max}),\ \dots\ \mathbf{l}_{n-1},\ [t^{n-1}_{min},\ t^{n-1}_{max}),\ \mathbf{l}_n>$, satisfying the following conditions:

1. For all $i = 1, \dots, n$, $\mathbf{l}_i \in L$;
2. For all $i = 1, \dots, n-1$, $t^i_{min}$, $t^i_{max}$ are integers s.t. $0 \le t^i_{min} < t^i_{max}$.

Let $d = <\mathbf{l}_1,\ [t^1_{min},\ t^1_{max}),\ \mathbf{l}_2,\ [t^2_{min},\ t^2_{max}),\ \dots\ \mathbf{l}_{n-1},\ [t^{n-1}_{min},\ t^{n-1}_{max}),\ \mathbf{l}_n>$ be an $L$ spy sequence of length $n$ for $P$, and let $\omega$ be an $(I,\ P)$ sequence for $P$. We write $d \le \omega$ iff, for all $1 \le i \le n$, $\mathbf{l}_i = a_{F_\omega(i)}$, and, for all $1 \le i < n$, $t^i_{min} \le Timer(\mathbf{x}_{F_\omega(i+1)}) < t^i_{max}$.

Let $h \in H$, $d$ be an $L$ spy sequence of length $n$ for $P$, and $i$ be an index $0 \leq i < n$. The *No-privacy* property $NPr(d,i,h)$ for $P$ holds iff the following condition is satisfied:

$$\forall(I,P) \text{ sequences } \omega \text{ for } P, \text{ if } d \leq \omega \text{ then } h \in \{a_{F_\omega(i)+1}, \cdots, a_{F_\omega(i+1)-1}\}.$$

## *4.3 Modeling the web attack with NuSMV*

In this section we describe how NuSMV can be used to check No-privacy. Note that, as a matter of fact, our NuSMV model follows quite closely the HyTech model of Sect. 3.3, the main difference being that clocks and synchronization have to be modeled explicitly in NuSMV.

### 4.3.1 System description

The system description consists of the specification of constants, names for expressions, variables, error states, and processes running in parallel. In Figs. 14–19 we show the specification of our case study.

All descriptions require the module `main`, which follows the keyword `MODULE` (Fig. 14).

The keyword `DEFINE` (Fig. 14) is used to define constants as well as to assign names to expressions. It has the same meaning as the C language `#define` preprocessing directive. For example, the first line after `DEFINE` in Fig. 14 means that all occurrences of `Sync_Au` in our NuSMV code will be replaced by the *definition* of `Sync_Au`, that is, {`req_e`, `show_e`, `req_w`, `show_w`}.

The keyword `VAR` is followed by the list of the module variables together with their (finite) ranges. For example, variable `Ac` in Fig. 14 ranges over the states of the cache automaton in Fig. 4. Discrete clocks are also modeled using variables with a suitable range.

NuSMV has no synchronization primitives and uses *shared variables* for communication. Hence, synchronization must be realized using the (shared) variables defined in `VAR` in Fig. 14. This means that a next state value must be defined for *all* possible actions, not just for the *expected* actions as for the automata in Fig. 4. For example, consider the NuSMV variable `Ac` modeling the cache automaton. When `Ac = 1`, a next state value must be defined also for the *illegal* action `yes_c`. This is not needed in LH-

```
MODULE main

DEFINE
Sync_Au := {req_e, show_e, req_w, show_w};
Sync_Ab := {receive_e, load_e,  show_e, req_e, receive_e1,
            load_e1,  show_e1, req_e1, receive_w, load_w,
            show_w, req_w, look_c, write_c, no_c, yes_c};
Sync_Ac := {look_c, no_c, write_c, yes_c};-- sync labels Ac
Sync_Aw := {receive_w, load_w};  -- sync labels automaton Aw
Sync_Aa := {req_e, req_w, show_w, show_e, req_e1, show_e1};
            -- sync labels Aa
Sync_B := {load_e, receive_e, load_e1, yes_c};--sync labels B

K := 100;
D1_min := 0;
D1_max := 10;
D2_min := 0;
D2_max := 20;  -- With D2_max = 20 NPr holds;
               -- with  D2_max = 21 NPr does not hold.

ERROR_FLAG := (Ab = -1) | (Au = -1) | (Ac = -1) | (Aw = -1) |
              (Aa = -1) | (B = -1) | (x1 >= 7) | (x2 >= 255) |
              (x3 >= 31) | (x4 >= 3) | (x5 >= 255) ;

VAR
act : {receive_e, load_e,  show_e, req_e,receive_e1, load_e1,
       show_e1, req_e1, receive_w, load_w, show_w, req_w,
       look_c, write_c, no_c, yes_c, nop}; -- action set

Au : -1 .. 3; -- user
Ab : -1 .. 13; -- browser
x1 : 0 .. 7;   -- timer Ab
Ac : -1 .. 3; -- cache
x2 : 0 .. 255; -- timer Ac  (max K + 1)
Aw : -1 .. 2; -- web
x3 : 0 .. 31;  -- timer Aw  (max 26)
Aa : -1 .. 6; -- applet
x4 : 0 .. 3;  -- timer Aa   (max 2)
B : -1 .. 4; -- behaviour (sequence d)
x5 : 0 .. 255;  -- timer B   (max 31)
```

**Fig. 14.** State variables and their ranges

```
ASSIGN   -- cache
init(Ac) := 1;
next(Ac) :=
case
   ERROR_FLAG : -1;  -- error state
   (Ac = 1) & (act = look_c): 1;
   (Ac = 1) & (0 <= x2) & (x2 <= 1) & (act = no_c): 1;
   (Ac = 1) & (act = write_c): 2;
   (Ac = 2) & (x2 <= K) & (act = look_c) : 3;
   (Ac = 2) & (x2 > K) & (act = look_c) : 1;
   (Ac = 3) & (2 <= x2) & (x2 <= 5) & (act = yes_c): 2;
   (act in Sync_Ac) : -1; -- error state;
   1: Ac;
esac;

ASSIGN   -- cache clock x2
init(x2) := 0;
next(x2) :=
case
   Ac <= 0 : 0; -- reset
   (Ac = 1) & (act = look_c): 0;
   (Ac = 1) & (act = write_c): 0;
   (Ac = 2) & (x2 <= K) & (act = look_c) : 0;
   (Ac = 2) & (x2 > K) & (act = look_c) : 0;
   (x2 < 254) : x2 + 1;
   1 : x2;
esac;
```

**Fig. 15.** The cache

```
ASSIGN         -- user
init(Au) := 1;
next(Au) :=
case
   ERROR_FLAG : -1;  -- error state
   (Au = 1) & (act = req_e): 2;
   (Au = 1) & (act = req_w): 3;
   (Au = 2) & (act = show_e): 1;
   (Au = 3) & (act = show_w): 1;
   (act in Sync_Au) : -1; -- error state;
   1 : Au;
esac;
```

**Fig. 16.** The user

```
ASSIGN  --  browser
init(Ab) := 1;
next(Ab) :=
case
   ERROR_FLAG : -1;  -- error state
   (Ab = 1) & (act = req_w): 2;
   (Ab = 1) & (act = req_e): 8;
   (Ab = 1) & (act = req_e1): 11;
   (Ab = 2) & (0 <= x1) & (x1 <= 1) & (act = look_c): 3;
   (Ab = 3) & (act = yes_c): 7;
   (Ab = 3) & (act = no_c): 4;
   (Ab = 4) & (0 <= x1) & (x1 <= 1) & (act = load_w): 5;
   (Ab = 5) & (act = receive_w): 6;
   (Ab = 6) & (2 <= x1) & (x1 <= 5) & (act = write_c): 7;
   (Ab = 7) & (0 <= x1) & (x1 <= 1) & (act = show_w): 1;
   (Ab = 8) & (0 <= x1) & (x1 <= 1) & (act = load_e): 9;
   (Ab = 9) & (act = receive_e): 10;
   (Ab = 10) & (0 <= x1) & (x1 <= 1) & (act = show_e): 1;
   (Ab = 11) & (0 <= x1) & (x1 <= 1) & (act = load_e1): 12;
   (Ab = 12) & (act = receive_e1): 13;
   (Ab = 13) & (0 <= x1) & (x1 <= 1) & (act = show_e1): 1;
   (act in Sync_Ab) : -1; -- error state;
   1 : Ab;
esac;

ASSIGN  -- browser clock
init(x1) := 0;
next(x1) :=
case
   Ab <= 0 : 0; -- reset
   (Ab = 1) & (act = req_w): 0;
   (Ab = 1) & (act = req_e): 0;
   (Ab = 1) & (act = req_e1): 0;
   (Ab = 3) & (act = yes_c): 0;
   (Ab = 3) & (act = no_c): 0;
   (Ab = 5) & (act = receive_w): 0;
   (Ab = 6) & (2 <= x1) & (x1 <= 5) & (act = write_c): 0;
   (Ab = 9) & (act = receive_e): 0;
   (Ab = 12) & (act = receive_e1): 0;
   (x1 < 6) : x1 + 1;
   1 : x1;
esac;
```

**Fig. 17.** The browser

```
ASSIGN  -- WEB SITE
init(Aw) := 1;
next(Aw) :=
case
   ERROR_FLAG : -1;  -- error state
   (Aw = 1) & (act = load_w): 2;
   (Aw = 2) & (10 <= x3) & (x3 <= 25) & (act = receive_w):1;
   (act in Sync_Aw) : -1; -- error state;
   1: Aw;
esac;

ASSIGN  -- WEB SITE CLOCK x3
init(x3) := 0;
next(x3) :=
case
  Aw <= 0 : 0; -- reset
  (Aw = 1) & (act = load_w): 0;
  (x3 < 30) : x3 + 1;
   1 : x3;
esac;
```

**Fig. 18.** The site handling $w$

Timed Automata. The next state for an illegal action is an *error state* that we model with the value `-1`. If a process is in an error state, then it can only stay there. The set of error states is defined by the constant `ERROR_FLAG` in Fig. 14. (NuSMV uses the symbol "&" for the boolean "and" ($\wedge$), the symbol "|" for the boolean "or" ($\vee$), and the symbol "!" for the boolean "not" ($\neg$)).

```
ASSIGN   -- APPLET
init(Aa) := 1;
next(Aa) :=
case
   ERROR_FLAG : -1;  -- error state
   (Aa = 1) & (act = show_e): 2;
   (Aa = 1) & (act = req_e): 1;
   (Aa = 1) & (act = req_w): 1;
   (Aa = 1) & (act = show_w): 1;
   (Aa = 2) & (0 <= x4) & (x4 <= 1) & (act = req_w): 3;
   (Aa = 3) & (act = show_w) : 4;
   (Aa = 4) & (0 <= x4) & (x4 <= 1) & (act = req_e1): 5;
   (Aa = 5) & (act = show_e1): 6;
   (act in Sync_Aa) : -1; -- error state;
   1: Aa;
esac;

ASSIGN  -- applet clock
init(x4) := 0;
next(x4) :=
case
  (Aa <= 0) : 0; -- reset
  (Aa = 1) & (act = show_e): 0;
  (Aa = 3) & (act = show_w) : 0;
  (x4 < 2) : x4 + 1;
   1: x4;
esac;
```

**Fig. 19.** The applet

In Fig. 15 we define the cache process corresponding to the cache automaton of Fig. 4.

Keyword `ASSIGN` introduces the automaton definition.

Keyword `init` is used to define the initial value of a variable (i.e., automaton state). The expression `init(Ac) := 1;` sets to 1 the initial state of the process cache as required in Fig. 4.

Keyword `next` denotes the *next state* operator $'$ within NuSMV, that is, the NuSMV function `next` is used to define the transition function of a process. The `case-esac` construct returns the value on the right-hand side of the first condition that evaluates to true.

The first `case` rule (`ERROR_FLAG : -1`) in the definition of `next(Ac)` defines `Ac` behavior in an error state. The second `case` rule (`(Ac = 1) & (act = look_c) : 1`) defines `Ac` behavior when the cache is in state `1` and action `look_c` is performed. Case rule `(act in Sync_Ac) : -1` takes `Ac` in the error state when an *unexpected* action is received (expression *Sync_Ac* was defined in Fig. 14). The last case rule of `next(Ac)` (i.e., `1 : Ac`) leaves the value of `Ac` unchanged. This applies when none of the above case rules applies.

Figures 17, 16, 19, and 18 give, respectively, our NuSMV model for the browser automaton ($A_b$) in Fig. 4, the user automaton ($A_u$) in Fig. 3, the applet automaton ($A_a$) in Fig. 5, and the web automaton ($A_w$) in Fig. 4. Such NuSMV models are obtained using the same approach used in Fig. 15 to model the cache automaton $A_c$ of Fig. 4.

Clocks are also modeled along the same lines.

#### 4.3.2 Checking No-privacy

To show how we can use NuSMV to check No-privacy properties, let us take property $NPr((load_e, [0, 10), receive_e, [0, 20), load_{e_1}), 2, yes_c)$.

The idea is to use the *adversary* process described in Fig. 20 that checks if there exists a spy sequence in which the high action `yes_c` does not occur between the low actions $receive_e$ and $load_{e_1}$ within the given time bounds. That is, if the *time elapsed* (as measured by clock `x5` in Fig. 20) between low actions $receive_e$ and $load_{e_1}$ is less than `D2_max`.

The NuSMV keyword `SPEC` introduces a property to be verified. In Fig. 21 we formalize the property that there is no spy sequence with `yes_c` in the required place. If `SPEC` fails, then No-privacy holds.

The actual NuSMV code is obtained by concatenating the codes of Figs. 14–21.

```
ASSIGN   -- BEHAVIOR (ADVERSARY)
init(B) := 1;
next(B) :=
case
  ERROR_FLAG : -1;  -- error state
  (B = 1) & (act = load_e) : 2;
  (B = 1) : B;
  (B = 2) & (D1_min <= x5) & (x5 <= D1_max)
          &  (act = receive_e) : 3;
  (B = 2) & (D1_min <= x5) & (x5 <= D1_max) : 2;
  (B = 2) & ((D1_min > x5) | (x5 > D1_max)) : -1; -- fail
  (B = 3) & (D2_min <= x5) & (x5 <= D2_max)
          & (act = load_e1) : 4;
  (B = 3) & (x5 > D2_max) : -1; -- too late, fail
  (B = 4) : B; -- NPr fails since spy sequence
               -- without yes_c seen.
  (act in Sync_B) : -1; -- error state;
  1: B;
esac;


ASSIGN  -- BEHAVIOR CLOCK
init(x5) := 0;
next(x5) :=
case
  (B <= 1) : 0; -- reset
  (B = 2) & (D1_min <= x5) & (x5 <= D1_max)
          & (act = receive_e) : 0;
  (B = 3) & (D2_min <= x5) & (x5 <= D2_max)
          & (act = load_e1) : 0;
  (B = 3) & (x5 > D2_max) : x5;  -- saturation
  (B = 4) : x5;
  (x5 < 254) : x5 + 1;
  1: x5;
esac;
```

**Fig. 20.** The adversary

```
--Spec NPR({load_e,[0, 11),receive_e,[0, 20),load_e1},2,yes_c)
--i.e. B = 4 is reachable

SPEC
AG (!ERROR_FLAG  -> B != 4)
```

**Fig. 21.** NuSMV specification

```
Iteration 0: Early evaluation of specifications
-- specification AG (!ERROR_FLAG -> B != 4) is true
Iteration 1: Early evaluation of specifications
....
Iteration 275: Early evaluation of specifications
-- specification AG (!ERROR_FLAG -> B != 4) is true
The verification is complete.

resources used:
user time: 154.99 s, system time: 0.72 s
BDD nodes allocated: 978224
Bytes allocated: 16908288
BDD nodes representing transition relation: 6156 + 19
```

**Fig. 22.** A glimpse of SMV output when the No-privacy property holds (`D2_max` $= 20$)

Figure 22 sketches NuSMV output for our NuSMV program. In this case the property is verified.

Changing the value of `D2_max` in Fig. 14 to "`D2_max := 21`" the No-privacy property no longer holds. The result of running NuSMV in this case is sketched in Fig. 23, where a counterexample is given. Intuitively, the larger our temporal window is, the weaker our constraints are. Eventually (e.g., when `D2_max` $= 21$), we are no longer able to guarantee that on all computation sequences satisfying our temporal constraints high action `yes_c` occurs, as we would like to expect.

## 5 A process algebra approach

In this section we show how the process algebra approach of [12–14] can be exploited to analyze our case study. In order to make an automated analysis feasible, we restate the theory in the $TCCS$ language supported by the CWB-NC tool [38].

### *5.1 The process algebra used: TCCS*

$TCCS$ is a $CCS$-like process algebra extended with operators to model the elapsing of time. The time modeling approach is fairly simple: There is one special *tick* action to represent the elapsing of one time unit, while all the other actions are without duration.[1] As a matter of fact, this seems reasonable if we choose a time unit such that the actual time of an action is negligible w.r.t. the time unit [5]. A global clock is supposed to be updated, whenever all the processes of the system agree on this, by globally synchronizing on action *tick*. Hence, the computation proceeds in lock-steps: between two global synchronizations on action *tick*, all the processes proceed asynchronously by performing actions without duration.

Another feature of $TCCS$ is the so-called *maximal progress assumption* according to which *tick* actions have

[1] These are the features of the so-called *fictitious clock* approach (e.g., see [28]).

```
Iteration 0: Early evaluation of specifications
-- specification AG (!ERROR_FLAG -> B != 4) is true
...
Iteration 24: Early evaluation of specifications
-- specification AG (!ERROR_FLAG -> B != 4) is true
Iteration 25: Early evaluation of specifications
-- specification AG (!ERROR_FLAG -> B != 4) is false
-- as demonstrated by the following execution sequence

 state 1.1:   | state 1.2:       | state 1.6:   | state 1.19:     | state 1.23:
 act = req_e  | act = load_e     | act = look_c | act = receive_w | act = show_w
 Au = 1       | Ab = 8           | Ab = 2       | x2 = 12         | Ab = 7
 Ab = 1       | ...              | ...          | x3 = 10         | ...
 x1 = 0       |                  |              | x5 = 15         |
 Ac = 1       | state 1.3:       | state 1.7:   |                 | state 1.24:
 x2 = 0       | act = receive_e  | act = no_c   | state 1.20:     | act = req_e1
 Aw = 1       | Ab = 9           | Ab = 3       | act = nop       | Au = 1
 x3 = 0       | ...              | ...          | Ab = 6          | ...
 Aa = 1       |                  |              | ...             |
 x4 = 0       | state 1.4:       | state 1.8:   |                 | state 1.25:
 B = 1        | act = show_e     | act = load_w | state 1.22:     | act = load_e1
 x5 = 0       | Ab = 10          | Ab = 4       | act = write_c   | Ab = 11
 ...          |                  | ...          | x1 = 2          | ...
              | state 1.5:       |              | x2 = 15         |
              | act = req_w      | state 1.9:   | x3 = 13         | state 1.26:
              | Ab = 1           | act = nop    | x5 = 18         | act = nop
              | ...              | Ab = 5       |                 | Ab = 12
              |                  | ...          |                 | ...
No more specifications left.

resources used:
user time: 5.2 s, system time: 0.05 s
BDD nodes allocated: 249487
Bytes allocated: 5242880
BDD nodes representing transition relation: 6153 + 19
```

**Fig. 23.** A glimpse of SMV output when the No-privacy property does not hold (`D2_max` = 21)

lower priority w.r.t. internal $\tau$ actions: a communication (or an internal activity $\tau$) prevents time from occurring, hence the clock synchronization on *tick* takes place only when all local processes have completed the execution of all the possible communications in that round.

Let us formally introduce the syntax and semantics of $TCCS$. As usual, the set of actions $L \cup H$ (low/high) is enriched with the *silent* action $\tau$, which models internal computation steps, and with the action *tick*. A complementation function $\overline{(\cdot)}$ is defined over $L \cup H$ such that $\overline{l} \in L$, for each $l \in L$, and $\overline{h} \in H$, for each $h \in H$. The idea is that two complementary actions performed by processes running in parallel can synchronize, thus producing action $\tau$. We let $\alpha, \beta, \ldots$ range over $L \cup H \cup \{\tau\}$. The syntax for $TCCS$ terms is as follows:

$$E ::= nil \,|\, \alpha.E \,|\, n.E \,|\, E_1 + E_2 \,|\, E_1 \| E_2 \,|\, E\backslash F \,|\, E_1[> E_2 \,|\, Z\,,$$

where $n$ is a natural number, $F \subseteq L \cup H$ is closed w.r.t. complementation, and $Z$ is a process constant that must be associated with a definition $Z = E$. (As usual, we assume that constants are guarded, i.e., they must be within the scope of some prefix operator $\alpha.E'$.)

A short informal overview of $TCCS$ operators follows. The process *nil* does nothing.

The process $\alpha.E$ can emit the action $\alpha$ reaching the configuration $E$. Furthermore, it always has the possibility of performing a *tick* action and then behaving again as $\alpha.E$. These actions are called *lazy*. Let $\mathcal{E}$ be the set of $TCCS$ processes, i.e., closed and guarded $TCCS$ terms. A short informal description of the operators follows.

The process $n.E$, with $n > 0$, allows one unit of time to pass, i.e., it emits the action *tick* and reaches the configuration $n-1.E$ ($E$, when $n = 1$).

The process $E_1 + E_2$ acts as a nondeterministic choice between $E_1$ and $E_2$. However, note that the time elapsing is synchronous, i.e., both perform the *tick* action at the same time, and is possible only when neither $E_1$ nor $E_2$ is able to perform a $\tau$ action.

The process $E_1 \| E_2$ is the parallel composition between $E_1$ and $E_2$. $E_1 \| E_2$ can perform an action $\alpha$ when either of the two processes can perform that action. Moreover, when the two processes may perform complementary actions, then a communication step is executed. However, note that the time elapsing is synchronous and is possible only when neither $E_1$ nor $E_2$ is able to perform a $\tau$ action and moreover when they cannot communicate.

The *disabling* construct of $TCCS$, i.e., $E_1[> E_2$, works as follows. If $E_1$ can perform an action, then the whole system can perform this action; however, the right-hand process $E_2$ may interrupt the sequence of actions of the other $E_1$ in each moment by performing one of its actions, and the whole system proceeds as the successor of $E_2$. Moreover, both must synchronize on *tick* actions. This construct may be used to model time-out behavior. Indeed, $n.nil[> l.E$ forces the firing of action $l$ within $n$ units of time. As a matter of fact, after $n$ units of time without firing $l$, a configuration of $nil[> l$ is reached that cannot let time pass.

The formal behavior of $TCCS$ terms is described by means of the *labeled transition system*, where the transition relation $\longrightarrow$ is the least relation between $TCCS$ terms induced by axioms and inference rules of Fig. 24. Note that the definition uses an auxiliary transition relation $\rightarrowtail$. Indeed, recall that $TCCS$ enjoys maximal progress assumption, i.e., $\tau$ actions have precedence over *tick* ones. The intuition is whenever a communication is possible, it must occur. Note that $\rightarrowtail$ is decidable and henceforth $\longrightarrow$.

$$\frac{}{\alpha.E \overset{\alpha}{\rightarrowtail} E}$$

$$\frac{}{1.E \overset{tick}{\rightarrowtail} E} \qquad \frac{n>1}{n.E \overset{tick}{\rightarrowtail} (n-1).E}$$

$$\frac{E_1 \overset{\alpha}{\rightarrowtail} E_1'}{E_1 + E_2 \overset{\alpha}{\rightarrowtail} E_1'} \qquad \frac{E_2 \overset{\alpha}{\rightarrowtail} E_2'}{E_1 + E_2 \overset{\alpha}{\rightarrowtail} E_2'}$$

$$\frac{E_1 \overset{tick}{\rightarrowtail} E_1' \quad E_2 \overset{tick}{\rightarrowtail} E_2'}{E_1 + E_2 \overset{tick}{\rightarrowtail} E_1' + E_2'}$$

$$\frac{E_1 \overset{\alpha}{\rightarrowtail} E_1'}{E_1 \| E_2 \overset{\alpha}{\rightarrowtail} E_1' \| E_2} \qquad \frac{E_2 \overset{\alpha}{\rightarrowtail} E_2'}{E_1 \| E_2 \overset{\alpha}{\rightarrowtail} E_1 \| E_2'}$$

$$\frac{E_1 \overset{\alpha}{\to} E_1' \quad E_2 \overset{\overline{\alpha}}{\to} E_2'}{E_1 \| E_2 \overset{\tau}{\to} E_1' \| E_2'}$$

$$\frac{E_1 \overset{tick}{\rightarrowtail} E_1' \quad E_2 \overset{tick}{\rightarrowtail} E_2'}{E_1 \| E_2 \overset{tick}{\rightarrowtail} E_1' \| E_2'}$$

$$\frac{Z = E \quad E \overset{\alpha}{\rightarrowtail} E'}{Z \overset{\alpha}{\rightarrowtail} E'} \qquad \frac{E_1 \overset{\alpha}{\rightarrowtail} E_1'}{E_1 \backslash F \overset{\alpha}{\rightarrowtail} E_1' \backslash F} (\alpha \notin F)$$

$$\frac{E_1 \overset{\alpha}{\rightarrowtail} E_1'}{E_1 [> E_2 \overset{\alpha}{\rightarrowtail} E_1' [> E_2} \qquad \frac{E_2 \overset{\alpha}{\rightarrowtail} E_2'}{E_1 [> E_2 \overset{\alpha}{\rightarrowtail} E_2'}$$

$$\frac{E_1 \overset{tick}{\rightarrowtail} E_1' \quad E_2 \overset{tick}{\rightarrowtail} E_2'}{E_1 [> E_2 \overset{tick}{\rightarrowtail} E_1' [> E_2'}$$

$$\frac{P \overset{\alpha}{\rightarrowtail} P'}{P \overset{\alpha}{\longrightarrow} P'} \qquad \frac{P \overset{tick}{\rightarrowtail} P' \quad \neg P \overset{\tau}{\rightarrowtail}}{P \overset{tick}{\longrightarrow} P'}$$

**Fig. 24.** Operational semantics for $TCCS$

Let us consider the following relations between $TCCS$ terms: $E \overset{\tau}{\Longrightarrow} E'$ if $E \overset{\tau}{\longrightarrow}^{*} E'$ (where $\overset{\tau}{\longrightarrow}^{*}$ is the reflexive and transitive closure of the $\overset{\tau}{\longrightarrow}$ relation), and, for $\alpha \neq \tau$, $E \overset{\alpha}{\Longrightarrow} E'$ if $E \overset{\tau}{\Longrightarrow}\overset{\alpha}{\longrightarrow}\overset{\tau}{\Longrightarrow} E'$. We say that a process E performs a (visible) trace $\alpha_1, \ldots, \alpha_n$ whenever $E \overset{\alpha_1}{\Longrightarrow} E_1 \overset{\alpha_2}{\Longrightarrow} \ldots \overset{\alpha_{n-1}}{\Longrightarrow} E_{n-1} \overset{\alpha_n}{\Longrightarrow} E_n$.

As a behavioral equivalence relation between processes, we consider the timed version of observation equivalence as in [15]. This equivalence permits one to abstract, to some extent, from the internal behavior of the system, represented by the internal actions.

A symmetric relation $\mathcal{R}$ over processes is a *timed weak simulation* iff for every $(p, q) \in \mathcal{R}$ we have:

- If $p \overset{\alpha}{\longrightarrow} p'$ for any $\alpha \in L \cup H \cup \{\tau\}$, then there exists $q'$ s.t. $q \overset{\alpha}{\Longrightarrow} q'$ and $(p', q') \in \mathcal{R}$;
- If $p \overset{tick}{\longrightarrow} p'$, then there exists $q'$ s.t. $q \overset{tick}{\Longrightarrow} q'$ and $(p', q') \in \mathcal{R}$.

A *timed weak bisimulation* is a relation $\mathcal{R}$ s.t. both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are timed weak simulations. We represent by $\approx_t$ the union of all the timed weak bisimulations.

Note that timed weak bisimulation is decidable on finite-state processes. The result holds because timed weak bisimulation is defined as a weak bisimulation on an enriched set of labels, and weak bisimulation is indeed decidable [19] on finite-state processes.

## *5.2 Checking No-privacy with CWB-NC*

In this section we show that the CWB-NC is suited also to check No-privacy. Indeed, we may use the $\mu$-calculus [21] built-in model checker of CWB-NC. Basically, given a process and a logical formula, there is a procedure that checks whether the process satisfies the formula. Thus, we need to find a formula that encodes the No-privacy property.

We may redefine the concepts of Sect. 3 as timed-word, low-level view, and so on, by starting from the *labeled transition system* (LTS) of a process.

Consider $NPr(d, i, h)$, where $d = (t_1, l_1) \ldots (t_n, l_n)$. We recall $NPr(d, i, h)$ holds if and only if it is not possible to perform a sequence of actions $d'$ whose visible part is $(t_1, l_1) \ldots (t_n, l_n)$ s.t. no action $h$ has been performed in the part of the computation corresponding to the visible trace $(t_i, l_i)(t_{i+1}, l_{i+1})$.

Consider the property $\phi_t^{H',l}(\psi)$ saying that a process can perform a computation $d'$ whose actions are in $\{tick\} \cup H'$ (where $H' = H \cup \{\tau\}$), s.t. the number of *tick* actions is exactly $t$, the last action is $l$, and after the property $\psi$ holds. Then, a process $E$ enjoys $NPr(d, i, h)$ iff $E$ does not enjoy

$$\phi_{t_1}^{H',l_1}(\ldots(\phi_{t_{i+1}}^{H'\backslash\{h\},l_{i+1}}(\ldots(\phi_{t_n}^{H',l_n}))\ldots)). \qquad (1)$$

Roughly, such a property says that it is possible to perform a trace whose visible part is $d$ without performing $h$ between the timed actions $(l_i, t_i)$ and $(l_{i+1}, t_{i+1})$.

It is possible to code the $\phi$ properties through $\mu$-calculus formulas as shown below.

#### 5.2.1 $\mu$-calculus syntax and semantics

For the sake of completeness, we briefly recall the syntax and semantics of $\mu$-calculus [21]. (The reader familiar with such a logic may wish to skip this subsection.)

Let $a$ be in $\{tick, \tau\} \cup L \cup H$ and $X$ be a variable ranging over a finite set of variables $Vars$. Formulas are generated by the following grammar:

$$A ::= X \mid T \mid F \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \langle a \rangle A \mid [a]A \mid \mu X.A \mid \nu X.A\,.$$

Logical connectives work as usual. The possibility modality $\langle a \rangle A$ expresses the ability to have an $a$ transition to a state that satisfies $A$. The necessity modality $[a]A$ expresses that after each $a$ transition there is a state that satisfies $A$. We consider the usual definitions of bound and free variables. The interpretation of a closed formula $A$ w.r.t. an LTS is the set of states where $A$ is true. The interpretation of a formula $\alpha(X)$ with a free variable $X$ is a function from set of states to set of states. Hence, the interpretation of $\mu X.\alpha(X)$ ($\nu X.\alpha(X)$) is the least (greatest) fixpoint of this function. There is a syntactic restriction on the formulas $\mu X.\alpha(X)$ and $\nu X.\alpha(X)$, i.e., the variable $X$ must be under the scope of an even number of negations ($\neg$). This ensures that the interpretation of a formula with free variable is a monotonic function; so a least (greatest) fixpoint exists.

Formally, given an LTS $\langle S, Act, \{\xrightarrow{a}\}_{a \in Act}\rangle$, the semantics of a formula $A$ is a subset $[\![A]\!]_\rho$ of $S$, defined in Table 1, where $\rho$ is a function from free variables of $A$ to subsets of $S$. The environment $\rho[S'/X](Y)$ is equal to $\rho(Y)$ if $Y \neq X$; otherwise $\rho[S'/X](X) = S'$.

We first consider the family of formulas $B_t^{H'}(C)$ defined as

$$B_0^{H'}(C) = \mu X.(C \vee \bigvee_{h \in H'} \langle h \rangle X)$$

$$\dots$$

$$B_t^{H'}(C) = \mu X.((\langle tick \rangle B_{t-1}^{H'}) \vee \bigvee_{h \in H'} \langle h \rangle X)\,.$$

Roughly, it says it is possible to reach a state where $C$ holds by passing through actions in $H' \cup \{tick\}$ and performing exactly a number $t$ of $tick$ actions.

Thus, the property $\phi_t^{H',l}(\psi)$ is encoded in the formula $A_t^{H',l}(C) = B_t^{H'}(\langle l \rangle C)$, where $C$ is the encoding of $\psi$.

Eventually, one can encode through a $\mu$-calculus formula the property (1) as

$$A_{t_1}^{H',l_1}(\dots(A_{t_{i+1}}^{H' \backslash \{h\}, l_{i+1}}(\dots(A_{t_n}^{H',l_n}))\dots)).$$

**Table 1.** Semantics of *mu*-calculus

| | |
|---|---|
| $[\![T]\!]_\rho$ | $= S$ |
| $[\![F]\!]_\rho$ | $= \emptyset$ |
| $[\![X]\!]_\rho$ | $= \rho(X)$ |
| $[\![\neg A]\!]'_\rho$ | $= S \backslash [\![A]\!]_\rho$ |
| $[\![A_1 \wedge A_2]\!]_\rho$ | $= [\![A_1]\!]_\rho \cap [\![A_2]\!]_\rho$ |
| $[\![A_1 \vee A_2]\!]_\rho$ | $= [\![A_1]\!]_\rho \cup [\![A_2]\!]_\rho$ |
| $[\![\langle a \rangle A]\!]_\rho$ | $= \{s \mid \exists s' : s \xrightarrow{a} s' \text{ and } s' \in [\![A]\!]_\rho\}$ |
| $[\![[a]A]\!]_\rho$ | $= \{s \mid \forall s' : s \xrightarrow{a} s' \text{ implies } s' \in [\![A]\!]_\rho\}$ |
| $[\![\mu X.A]\!]_\rho$ | $= \bigcap\{S' \mid [\![A]\!]_{\rho[S'/X]} \subseteq S'\}$ |
| $[\![\nu X.A]\!]_\rho$ | $= \bigcup\{S' \mid S' \subseteq [\![A]\!]_{\rho[S'/X]}\}$ |

### *5.3 Security properties in the process algebra approach*

Briefly, in this section we recall one information flow security property, $tBSNNI$, which is defined in a real-time setting in [14] for process algebra.

The main rationale is that a system is secure whenever the low-level view of the system does not change when all high-level activities are enabled.

To define the property we need to present the so-called $Top_H^{tick}$ process

$$Top_H^{tick} = \sum_{\alpha \in H} \alpha.Top_H^{tick}$$

that iteratively enables any possible high-level action. Formally, property $tBSNNI$ is defined as follows:

$$E \in tBSNNI \text{ iff } (E \| Top_H^{tick}) \backslash H \approx_t E \backslash H.$$

Note that this property can be checked in polynomial time in the size of finite systems $E$ [19, 23]. Indeed, when $E$ is finite state, then $(E \| Top_H^{tick}) \backslash H$ is also finite state. Moreover, we recall that checking the timed weak bisimulation is equivalent to checking weak bisimulation (these are indeed the same), which is in turn decidable in polynomial time [19] for finite-state processes. Thus $tBSNNI$ can be easily checked using CWB-NC (for finite-state systems).

Weak bisimulation is finer than *traces*, which, on the contrary, we have considered in previous sections of this paper. As has been proved in [12–14], by considering such a finer notion of system behavior one can reveal attacks (leakage of privacy) that cannot be captured by considering only traces.

### *5.4 The web attack analysis with CWB-NC*

In order to formally check our noninterference properties, we use the verification tool *CWB-NC* [38]. This is a software environment for developing and analyzing concurrent and distributed systems. It supports many formal specification languages, such as $CCS$ and $TCCS$ [38].

In Fig. 25 we model the web attack with $TCCS$.

Our way of checking the leakage of privacy of the cache is simply by perturbing its values and then observing whether this perturbation may be perceived by the malicious site. In this setting, the perturbation is obtained by altering the action responses of the cache to the web browser request, and so interfering with the cache values.

By using the tool we can infer that the system $S = (Ab_1 \| Ac_1 \| Aw \| Aa)$ (which consists of the parallel composition of the browser, the cache, the applet, and the site handling $w$) is not $tBSNNI$. However, for efficiency reasons, we instead check that $S' = S \setminus unVis$ (named *System* in Fig. 25) is not $tBSNNI$, where $unVis$ is the set of actions defined in Fig. 25. Note that these actions should not be visible to a third party. The size of this process is clearly smaller. This is sound, since if $S \in tBSNNI$, then $S' \in tBSNNI$. Indeed, we have

$$\begin{gathered} S \in tBSNNI \iff \\ \text{(By definition)} \\ S \| Top_H^{tick} \setminus H \approx_t S \setminus H \Longrightarrow \\ \text{(since } \approx_t \text{ is a congruence for } \setminus) \\ S \| Top_H^{tick} \setminus H \setminus unVis \approx_t S \setminus H \setminus unVis \iff \\ \text{(since } H \cap unVis = \emptyset) \\ (S \setminus unVis \| Top_H^{tick}) \setminus H \approx_t S \setminus unVis \setminus H \iff \\ S' \in tBSNNI\,. \end{gathered}$$

```
****************************************
*
* Case Study: Modeling the Cache problem
*
****************************************

set HighActions = {yes_c, no_c}

set unVisible = {load_w,receive_w,show_e, req_e,show_w, req_w,
                 req_e1,show_e1, look_c, write_c}

* Code for the Top process
proc Top =yes_c.Top + no_c.Top  + 'yes_c.Top + 'no_c.Top

* Code for the Applet
 proc Aa  =   'req_e.show_e.'req_w.show_w.'req_e1.show_e1.nil

* Code for the cache
proc Ac1 = look_c.Ac1 + ('no_c.1.Ac1 + 1.t.Ac11) + write_c.Ac2
proc Ac11 = look_c.Ac1 +  write_c.Ac2
proc Ac2 = (100.nil [> look_c.Ac3) + 100.look_c.Ac1
proc Ac3 = 2. (3.nil [> 'yes_c. Ac2 )

* Code for the web site w
proc Aw = 'load_w.2.(13.nil [> 'receive_w.Aw)

* Code for the Browser
proc Ab1 = req_e.Ab8 + req_e1.Ab11 + req_w.Ab2
proc Ab2 = ( 1.nil [> 'look_c . Ab3)
proc Ab3 = yes_c . Ab7 + no_c . Ab4
proc Ab4 = (1.nil [> load_w . Ab5)
proc Ab5 = (receive_w . Ab6)
proc Ab6 = 2.(3 . nil [> 'write_c . Ab7)
proc Ab7 = (1.nil [> 'show_w.Ab1)
proc Ab8 = load_e.Ab9
proc Ab9 = receive_e.Ab10
proc Ab10 = (1.nil [> 'show_e.Ab1)
proc Ab11 = load_e1.Ab12
proc Ab12 = receive_e1.Ab13
proc Ab13 = (1.nil [> 'show_e1 . Ab1)

* System without the Top (Enemy)
proc System = (Ab1 | Ac1 | Aw |Aa ) \unVisible \ HighActions

* System with the Top (Enemy)
proc System_attacked  = ((Ab1 | Ac1 | Aw |Aa) \unVisible | Top)
                         \ HighActions
```

**Fig. 25.** *TCCS* specification in the CWB-NC notation. In particular, $\tau$ is denoted by $t$ and complementation is denoted by $'$

We applied CWB-NC to check the $tBSNNI$ property for $S'$, i.e., $S' \setminus H \approx_t S' \| Top_H^{tick} \setminus H$, obtaining `false` as output. Thus, $S' \notin tBSNNI$, from which it follows that $S \notin tBSNNI$.

Indeed, $S' \| Top_H^{tick} \setminus H$ may perform the following visible trace ($d_1$):

$$30.load_e.10.receive_e.8.load_{e1}, \tag{2}$$

while $S' \setminus H$ cannot. Note that $d_1$ is the trace used in Sect. 3 to show that No-privacy holds.

### *5.5 Formal comparison between $tBSNNI$ and No-privacy*

We compare No-privacy and $tBSNNI$ in the process algebra framework.

Recall that a system in $tBSNNI$ should be regarded as "secure", while a system that enjoys a No-privacy property should be regarded as an "insecure" one.

We give some examples. In the following we assume $H = \{h, h_1\}$.

*Example 1.* The process $E = h.l.nil + h_1.l.nil$ is not in $tBSNNI$ because

$$(E \| Top_H^{tick}) \setminus H \approx_t l.nil \not\approx_t E \setminus H \approx_t nil\,.$$

Nevertheless, $NPr((0,l),0,h)$ does not hold. Indeed, $E$ may perform $l$ without performing $h$.

*Example 2.* The process $E = h.l.nil + l.nil$ is in $tBSNNI$ because

$$(E \| Top_H^{tick}) \setminus H \approx_t l.nil \approx_t E \setminus H$$

and $NPr((0,l),0,h)$ does not hold. Indeed, $E$ may perform $l$ without performing $h$.

*Example 3.* The process $E = h.l$ is not in $tBSNNI$ because

$$(E \| Top_H^{tick}) \setminus H \approx_t l.nil \not\approx_t nil \approx_t E \setminus H$$

while $NPr((0,l),0,h)$ holds.

*Example 4.* The process $E = h.(tick.l.nil + h.nil)$ is in $tBSNNI$ because

$$(E \| Top_H^{tick}) \setminus H \approx_t nil \approx_t E \setminus H$$

while $NPr((1,l),0,h)$ holds.

These examples seem to suggest that there is no formal relationship between $tBSNNI$ and No-privacy. However, in the last example, the fact that the process enjoys $tBSNNI$ while No-privacy holds actually depends on the particular semantics we used for the process algebra, i.e., we considered the maximal progress assumption.

Indeed, if we drop the maximal progress assumption, we are able to prove an interesting result, namely, $tBSNNI$ is stronger than No-privacy. Technically we consider the $\rightarrowtail$ relation instead of the $\longrightarrow$ one.

The intuition is the following: whenever in a system we need to perform a high-level action to see a low-level one, it means that the system where all the high-level actions are restricted cannot present such a low action. On the other hand, the process with $Top$ (without maximal progress assumption) can perform the same sequence of actions where the high-level ones are renamed into $\tau$, thus also the low-level one.

**Proposition 1.** *If $E$ is in tBSNNI, then for no $d$ s.t. $E \stackrel{d'}{\Longrightarrow}$ and $d$, with length $n$, is the low-level view of $d'$, $0 \leq i \leq n-1$, $h \in H$, we have that $NPr(d,i,h)$ holds.*

*Proof.* (Sketch) Assume that $E$ is in $tBSNNI$. Assume that for a given sequence $d = (t_1, l_1).\dots.(t_n, l_n)$, $0 \leq i \leq n-1$ and $h \in H$, we have that

$NPr((t_1, l_1).\dots.(t_n, l_n), i, h)$

holds. It means that $E \setminus H$ cannot exhibit a sequence whose timed word is $(t_1, l_1)..(t_{i+1}, l_{i+1})..(t_n, l_n)$. Indeed, the action $l_{i+i}$ is not possible since an action $h$ must occur before this low action in each possible trace and the restriction $\setminus H$ operator prevents it. On the other hand, the sequence can be clearly obtained by $(E \| Top_H^{tick}) \setminus H$ because the $Top_H^{tick}$ will allow one to synchronize with each $h$ action. Indeed, given a trace of $E$ it is easy to prove that $(E \| Top_H^{tick}) \setminus H$ may perform the same trace where all high-level activities are renamed into $\tau$. (Recall that $\tau$ is the internal action.) Thus $(E \| Top_H^{tick}) \setminus H$ and $E \setminus H$ have different traces, and this leads to a contradiction since they cannot be timed weak bisimilar, and so $E$ is not in $tBSNNI$.

## 6 Comparison

In this section we compare the behavior of model checkers HyTech, NuSMV, and CWB-NC on the web attack case study.

### 6.1 The models

The three approaches use three different time models. In LH-Timed Automata one has a dense time domain and clocks that can be reset, NuSMV is a discrete time version of LH-Timed Automata, and $TCCS$ has a discrete time domain and a tick transition. On the one hand, $TCCS$ and NuSMV have the same time-expressiveness, but NuSMV is more succinct with respect to $TCCS$. In fact, to simulate $n$ discrete clocks a process in $TCCS$ must be exponential on $n$. On the other hand, with respect to time issues, the LH-Timed Automata model is more expressive than NuSMV and $TCCS$. For example, it is well known that there are cases in which dense time must be used since there is no discretization (no matter how small) that can faithfully emulate all dense time behaviors [4, 7, 30]. This means that there are cases where dense time is needed, even though the cost to verify the property becomes $C^n \cdot n!$, where $n$ is the number of clocks and $C$ the maximum constant that appears in the automaton.

The three approaches adopt different communication models. LH-Timed Automata use synchronous parallel composition, NuSMV uses shared variables, and $TCCS$ process algebra uses asynchronous parallel composition plus synchronous communication. To describe processes on the network the asynchronous parallel composition seems to be more natural, and moreover $TCCS$ has ad hoc constructs for that purpose. To describe hardware system models assuming synchronous communication may be preferable.

### 6.2 The properties

The two No-privacy properties that have been analyzed with HyTech, NuSMV, and CWB-NC differ from the property $tBSNNI$ that has been analyzed only with CWB-NC, in several ways. The first is that No-privacy considers traces as notions of system behavior, whereas $tBSNNI$ considers (timed) weak bisimulation. As has been proved in [12–14], the finer weak bisimulation semantics enables one to reveal more kinds of attacks. The second reason is that, in order to reveal attacks, with the No-privacy approach one has to check whether or not a given attack is possible (i.e., the attack expressed by the formula), whereas with $tBSNNI$ one simply checks whether *any* attack is possible.

As an example, in the web attack case study, $tBSNNI$ revealed that the system is not secure, whereas with No-privacy it turned out that $NPr(d_1, 2, yes_c)$ revealed the attack and $NPr(d_2, 2, yes_c)$ did not. On the one hand, this example suggested that $tBSNNI$ permits to reveal attacks that cannot be revealed by No-privacy. Proposition 1 in the process algebras approach has shown that when a system is “secure” under the $tBSNNI$, it is “secure” under the No-privacy property. On the other hand, the example shows that No-privacy can reveal actions that are needed for attacking the system.

### 6.3 Tool comparison

For all the tools we considered the worst-case performance is exponential in the input size. However, in practice, things may go quite differently depending on the verification problem at hand. In fact, each model checker is best suited for a certain class of problems. In this section we investigate the behavior of HyTech, NuSMV, and CWB-NC on our verification problem, i.e., the verification of the No-privacy property. This allows us to compare tool performances on our case study.

Be aware that we get a useful guideline on which tool to use only when facing model verification problems similar to the one we have studied.

The metric we will use for our comparison is the `CPU time` of the verification process (when it completes).

In order to compare the three tools we used (HyTech, NuSMV, CWB-NC), we parametrized all clock bounds in the web system model. That is, for each clock bound $v$ in the web system in Figs. 3, 4, and 5, we replaced $v$ with $c \cdot v$. By changing the value of $c$ we can change the *time granularity*. The larger the value of $c$ the smaller our time unit. For example, if $v = 2$ and $c = 10$, the resulting clock bound will be 20. We get the model in Figs. 3, 4, and 5 by setting $c = 1$. Note that for discrete time models, as those used by NuSMV or CWB-NC, $c$ plays the role of the *sampling frequency*.

In Fig. 26 (Fig. 27) we report our experimental results on using HyTech, NuSMV, and CWB-NC to verify the properties $NPr(d_1, 2, yes_c)$ $(NPr(d_2, 2, yes_c))$ defined in Sect. 3.2.

All the experiments have been performed on a 2-GHz Pentium Linux PC with 512 MB RAM.

The first column of Figs. 26 and 27 gives the tool used, whereas the first row gives the value of time granularity $c$. The remaining rows/columns give the CPU time (in seconds) used by the verification process to complete verification. This measure can be obtained, e.g., by using the UNIX system call `getrusage()`. As a result our CPU time measure does not depend on the activity of other processes on our UNIX machine.

A dash sign - in Figs. 26 and 27 denotes an *out of memory* for NuSMV or CWB-NC.

Figure 28 plots the experimental results in Figs. 26 and 27. On the $x$-axis we have the value of $c$, whereas on the $y$-axis we have the time needed to complete verification (when verification completes).

Our experimental results show that HyTech performance on our case study does not depend on the value of $c$. This is reasonable since essentially HyTech represents rational numbers using a fraction of the form $m/n$, where $m$ and $n$ are integers. As long as $m$ and $n$ are not *too large*, the representation size of the transition relation does not depend too much on the value of $c$.

NuSMV uses OBDDs to represent the transition relation of the system at hand. Thus the number of bits needed to represent a state grows with $c$. As a result the size of the OBDD representing the transition relation and the verification time for our system grow with $c$. This is clearly shown in Figs. 26, 27, and 28.

CWB-NC represents states explicitly (rather than symbolically like HyTech and NuSMV). When CWB-NC does not overflow, it is usually faster than NuSMV, but CWB-NC overflows before NuSMV does.

## 7 Conclusions

We have presented a comparative case study on automatic verification of a timed security (web privacy) property via model checking. The property we analyzed is a time-dependent web attack, originally presented by Felten and Schneider [11].

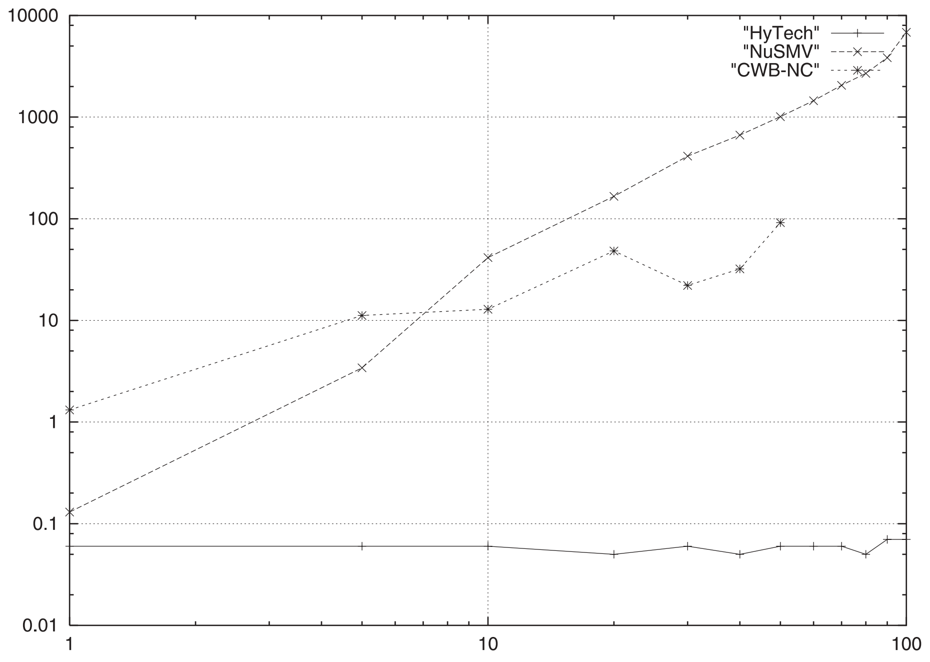
We modeled and verified our system using three model checkers: HyTech (to support an LH-Timed Automata

| Tool/Granularity | 1 | 5 | 10 | 20 | 30 | 40 |
|---|---|---|---|---|---|---|
| HyTech | 0.06 | 0.06 | 0.06 | 0.05 | 0.06 | 0.05 |
| NuSMV | 0.13 | 3.420 | 41.450 | 165.860 | 412.820 | 665.740 |
| CBW-NC | 1.316 | 11.171 | 12.878 | 48.176 | 22.095 | 32.176 |
| Tool/Granularity | 50 | 60 | 70 | 80 | 90 | 100 |
| HyTech | 0.06 | 0.06 | 0.06 | 0.05 | 0.07 | 0.07 |
| NuSMV | 1008.770 | 1447.140 | 2055.450 | 2689.860 | 3844.740 | 6817.120 |
| CWB-NC | 91.680 | - | - | - | - | - |

**Fig. 26.** Time comparison for HyTech, NuSMV, and CBW-NC when No-privacy holds

| Tool/Granularity | 1 | 5 | 10 | 20 | 30 | 40 |
|---|---|---|---|---|---|---|
| HyTech | 0.09 | 0.09 | 0.09 | 0.09 | 0.09 | 0.08 |
| NuSMV | 2.820 | 119.870 | 751.450 | - | - | - |
| CBW-NC | 1.428 | 4.237 | 17.672 | 18.460 | 29.722 | 28.218 |
| Tool/Granularity | 50 | 60 | 70 | 80 | 90 | 100 |
| HyTech | 0.09 | 0.08 | 0.09 | 0.09 | 0.09 | 0.09 |
| NuSMV | - | - | - | - | - | - |
| CWB-NC | 89.755 | - | - | - | - | - |

**Fig. 27.** Time comparison for HyTech, NuSMV, and CBW-NC when No-privacy does not hold



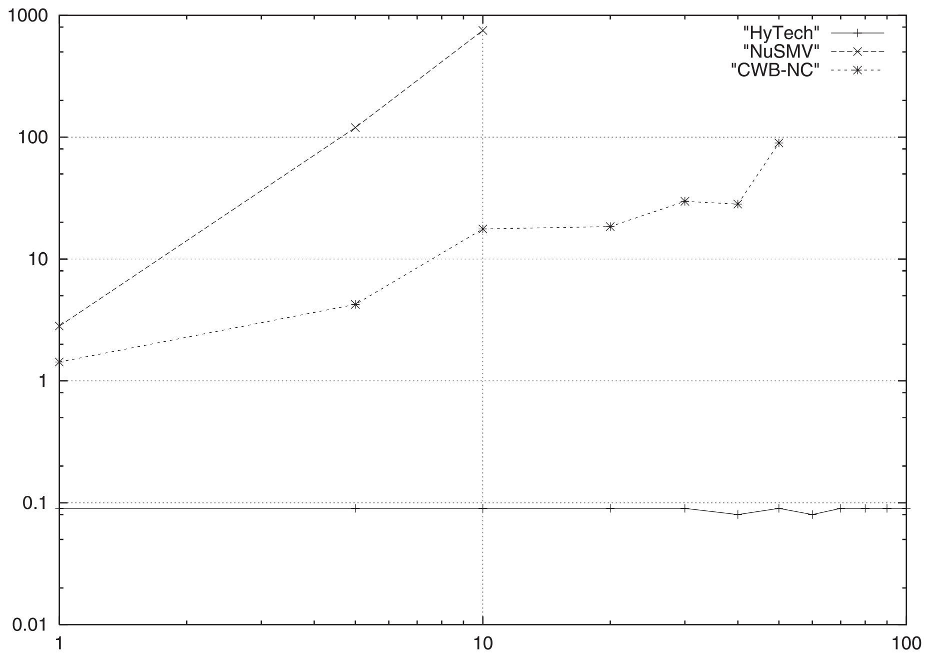


**Fig. 28.** Plotting Figs. 26 and 27. Verification time ($y$-axis) vs. time granularity ($x$-axis)

modeling), NuSMV (to support a finite-state automata modeling), and CWB-NC (to support a $TCCS$ modeling).

All of the approaches (LH-Timed Automata, finite-state automata, and $TCCS$) were capable of modeling the system and in addition offered suitable security properties able to reveal the attack.

We plan to extend our study to other time-critical applications, such as time-dependent cryptographic protocols for electronic commerce.

Summing up: all of the approaches we examined can be used to model and analyze the system under consideration. Depending on the design phase (requirements analysis, implementation, etc.), the actual implementation (hardware or software) and the communication paradigm one approach may be better than the others. As for efficiency, our experimental results show that for our problem HyTech is the most effective tool. When the kind of information leakage is not known a priori, the ability of the $TCCS$ approach of detecting *any* possible source of information flow is more appropriate. More examples are needed to fully clarify the relative merits, especially with respect to the formalization of security properties.

*Acknowledgements.* We would like to thank the anonymous referees for their helpful comments for the presentation of this paper.

## References

1. Abadi M (1999) Secrecy by typing in security protocols. J ACM 46(5):749–786
2. Alur R, Dill DL (1994) A THEORY OF TIMED AUTOMATA. Theor Comput Sci 126(2):183–235
3. Alur R, Henzinger TA, Ho PH (1996) Automatic symbolic verification of embedded systems. IEEE Trans Softw Eng 22(3):181–201
4. Asarin E, Maler O, Pnueli A (1998) On discretization of delays in timed automata and digital circuits. In: Proceedings of the international conference on concurrency theory. Lecture notes in computer science, vol 1466. Springer, Berlin Heidelberg New York, pp 470–484
5. Berry G, Gonthier G (1992) The Esterel Synchronous Programming Language: design, semantics, implementation. Sci Comput Programm 19(2):87–152
6. Bodei C, Degano P, Nielson F, Nielson HR (1998) Control flow analysis for the pi-calculus. In: Proceedings of the international conference on concurrency theory. Lecture notes in computer science, vol 1466. Springer, Berlin Heidelberg New York, pp 84–98
7. Bozga M, Maler O, Tripakis S (1999) Efficient verification of timed automata using dense and discrete time semantics. In: Proceedings of the international conference on correct hardware design and verification methods. Lecture notes in computer science, vol 1703. Springer, Berlin Heidelberg New York, pp 125–141
8. Bryant RE (1986) Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput 35(8):677–691
9. Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ (1992) Symbolic model checking: $10^{20}$ states and beyond. Inf Comput 98(2):142–170
10. Cimatti A, Clarke EM, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, Sebastiani R, Tacchella A (2002) NuSMV 2: An open source tool for symbolic model checking. In: Proceedings of the international conference on computer aided verification. Lecture notes in computer science, vol 2404. Springer, Berlin Heidelberg New York, pp 359–364
11. Felten EW, Schneider MA (2000) Timing attacks on Web privacy. In: Proceedings of the ACM conference on computer and communications security. ACM Press, New York, pp 25–32
12. Focardi R, Gorrieri R (1995) A classification of security properties for process algebras. J Comput Secur 3(1):5–33
13. Focardi R, Gorrieri R (1997) The compositional security checker: a tool for the verification of information flow security properties. IEEE Trans Softw Eng 23(9):550–571
14. Focardi R, Gorrieri R, Martinelli F (2000) Information flow analysis in a discrete-time process algebra. In: Proceedings of the IEEE Computer Security Foundation workshop. IEEE Press, Los Alamitos, pp 170–184
15. Focardi R, Gorrieri R, Martinelli F (2003) Real-time information flow analysis. IEEE J Select Areas Commun 21(1):20–35
16. Groote JF (1993) Transition system specifications with negative premises. Theor Comput Sci 118(2):263–299
17. Handschuh H, Howard Heys M (1999) A timing attack on RC5. In: Proceedings of the international workshop on selected areas in cryptography. Lecture notes in computer science, vol 1556. Springer, Berlin Heidelberg New York, pp 306–318
18. Henzinger TA, Ho PH, Wong-Toi H (1997) HyTech: A model checker for hybrid systems. Int J Softw Tools Technol Transfer 1(1–2):110–122
19. Kanellakis PC, Smolka SA (1990) CCS expressions, finite-state processes, and three problems of equivalence. Inf Comput 86(1):43–68
20. Kocher PC (1996) Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In: Proceedings of the international conference on advances in cryptology. Lecture notes in computer science, vol 1109. Springer, Berlin Heidelberg New York, pp 104–113
21. Kozen D (1983) Results on the propositional $\mu$-calculus. Theor Comput Sci 27(3):333–354
22. Lanotte R, Maggiolo-Schettini A, Tini S (2001) Privacy in real-time systems. In: Proceedings of the workshop on models for timed critical systems. Electronic notes in theoretical computer science, vol 52, Elsevier, Amsterdam
23. Martinelli F (1998) Partial model checking and theorem proving for ensuring security properties. In: Proceedings of the IEEE Computer Security Foundations workshop. IEEE Press, Los Alamitos, pp 44–52
24. McMillan KL (1993) Symbolic model checking. Kluwer, Norwell, Massachusetts
25. Meadows C (1997): Languages for formal specification of security protocols. In: Proceedings of the IEEE Computer Security Foundations workshop. IEEE Press, Los Alamitos, CA, pp 96–97
26. Milner R (1989) Communication and concurrency. Prentice Hall, London
27. Mitchell JC, Mitchell M, Stern U (1997) Automated analysis of cryptographic protocols using Murphi. In: Proceedings of the IEEE symposium on security and privacy. IEEE Press, Los Alamitos, CA, pp 141–153
28. Ostroff JS, Wonham WS (1990) A framework for real-time discrete event control. IEEE Trans Automat Control 35(4):386–397
29. Panda S, Somenzi F, Plessier BF (1994) Symmetry detection and dynamic variable ordering of decision diagrams. In: Proceedings of the IEEE International conference on computer-aided design. IEEE Press, Los Alamitos, CA, pp 628–631
30. Puri A, Varaiya P (1994) Decidability of hybrid systems with rectangular differential equations. In: Proceedings of the international conference on computer aided verification. Lecture notes in computer science, vol 818. Springer, Berlin Heidelberg New York, pp 95–104
31. Smith G, Volpano D (1998) Secure information flow in a multi-threaded imperative language. In: Proceedings of the ACM symposium on principles of programming languages. ACM Press, New York, pp 355–364
32. Song D, Wagner D, Tian X (2001) Timing analysis of Keystrokes and SSH timing attacks. In: Proceedings of the 10th USENIX security symposium, 2001
33. Volpano D, Smith G (1998) Confinement properties for programming languages. SIGACT News 29(3):33–42
34. CUDD Web Page: `http://vlsi.colorado.edu/~fabio/CUDD/`
35. NuSMV Web Page: `http://nusmv.irst.itc.it/`
36. URL: `http://www.cs.cmu.edu/~modelcheck/`
37. URL: `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`
38. URL: `http://www.cs.sunysb.edu/~cwb/`