URL: http://www.elsevier.nl/locate/entcs/volume62.html LastPage pages

Formal Models of Timing Attacks on Web Privacy [1]

Riccardo Focardi [a], Roberto Gorrieri [b], Ruggero Lanotte [c], Andrea Maggiolo-Schettini [c], Fabio Martinelli [d], Simone Tini [e], Enrico Tronci [f]

[a] *Dipartimento di Matematica Applicata e Informatica, Università di Venezia, Via Torino 155, 30173 Mestre, Italy*

[b] *Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy*

[c] *Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy*

[d] *Istituto per le Applicazioni Telematiche, C.N.R. di Pisa, Via Giuseppe Moruzzi 1, 56124 Pisa, Italy*

[e] *Dipartimento di Scienze CC.FF.MM., Università dell'Insubria, Via Valleggio 11, 22100, Como, Italy*

[f] *Dipartimento di Informatica, Università di L'Aquila, Via Vetoio, Coppito, 67100 L'Aquila, Italy*

Abstract

We model a timing attack on web privacy proposed by Felten and Schneider by using three different approaches: HL-Timed Automata, SMV model checker, and tSPA Process Algebra. Some comparative analysis on the three approaches is derived.

1 Introduction

One of the main requirements of mobile code is that it must guarantee some kind of security to clients executing it. One of the security requirements is the client's *privacy*, namely that executing mobile code does not imply leaking of private information.

Several papers (see, among the others, [4,5,6,9,10]) 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 mentioned papers,

[1] Research partially supported by MURST Progetto Cofinanziato TOSCA.

systems respect the property of 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 permits to infer information on the secret one.

In this paper we consider the timing attack on web privacy described in [3]. The attack compromises the privacy of user's web-browsing histories by allowing a malicious web site to determine whether or not the user has recently visited some other, unrelated, web page w. A Java applet is embedded in the malicious web site and is run by the user's browser. The applet first performs a request to a file of w, and then performs a new request to the malicious site. So, the malicious site can measure the time elapsed between the two requests which it receives from the user, and, if such a time is under a certain bound, it infers that w was in the cache of the browser of the user, thus implying that w has been recently visited by the user. The system has two levels: the malicious site can observe its interactions with the user, but it cannot observe the interactions between the user, the cache and the web site w.

We analyze the attack with three approaches. In Sect. 2 we use HL-Timed Automata [7], and we consider dense time temporal behaviors. In Sect. 3 and 4 we use SMV [8,2,11] and tSPA [6], respectively, and we consider discrete time temporal behaviors. Finally, in Sect. 5 we draw some conclusions.

2 The HL-Timed Automata approach

The formalism of HL-Timed Automata [7] is an extension of Alur and Dill's Timed Automata [1] suitable to model two-level systems. HL-Timed Automata are Timed Automata with two additional features:

- The alphabet of symbols recognized by an automaton consists of two sets: the set H of *high symbols* and the set L of *low symbols*.
- Automata can run in parallel and are perfectly synchronized, meaning that automata running in parallel can advance only by recognizing the same symbol at the same instant of time.

In Fig. 1 we model the web attack problem with HL-Timed Automata. Automaton A_c represents the cache. The time elapsed between a request r_c and an answer a_c is in the interval $[2, 5]$ (in fact, clock x is reset when the cache receives r_c, and it is required that x is in $[2, 5]$ when the cache gives the answer a_c). Automaton A_w represents the site w. The time elapsed between a request r_w and an answer a_w is in the interval $[100, 250]$. The automaton A_u represents the requests by the user that downloads the page of the malicious site. First of all, it performs a request r_e to the malicious site. Then, when it receives the answer a_e, it performs a communication either with the cache or with the site w. Finally, it performs another request r_e and it waits for an answer from the malicious site.

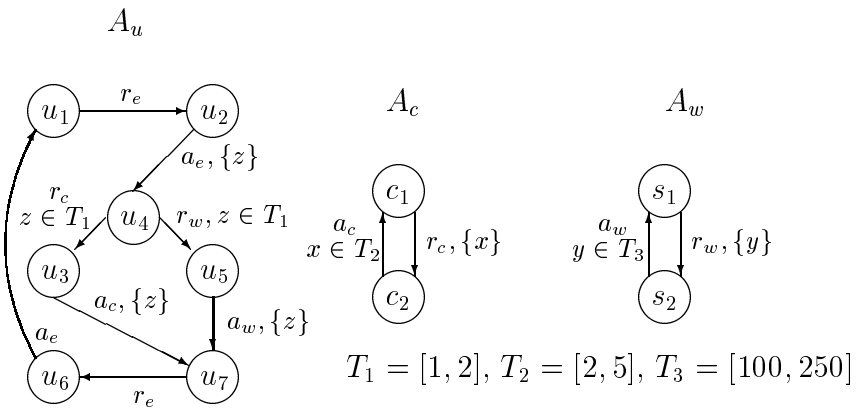

Fig. 1. The web system.

Now, the only visible symbols for the malicious site are r_e and a_e, since the malicious site can observe neither interaction between the user and the cache, nor interaction between the user and the web page w. So, $L = \{r_e, a_e\}$ and $H = \{r_c, a_c, r_w, a_w\}$. Moreover, to synchronize the automata, we assume that there are transitions from state c_1 to state c_1 labeled with symbols r_e, a_e, r_w and a_w, and that there are transitions from state s_1 to state s_1 labeled with symbols r_e, a_e, r_c and a_c.

Formally, given sets of high symbols H and low symbols L, a *HL-Timed Automaton* is a tuple $\mathcal{A} = ((L, H), A_1, \ldots, A_m)$, where, for each $1 \leq i \leq m$, $A_i = (Q_i, q_i^0, \delta_i, X_i)$ is a *sequential automaton*, with:

- a finite set of *states* Q_i
- an *initial state* $q_i^0 \in Q_i$
- a set of *clocks* X_i
- a set of *transitions* $\delta_i \subseteq Q_i \times \Phi(X_i) \times (L \cup H) \times 2^{X_i} \times Q_i$, 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 $\# \in \{<, \leq, >, \geq, =, \neq\}$, and where c is a constant.

The sets of clocks $X_1, \ldots, X_m$ are pairwise disjoint.

Intuitively, a transition (q, ϕ, a, Y, q') of an automaton A_i can fire when state q is active, symbol a is recognized and clocks satisfy the clock constraint ϕ. In such a case, state q' is entered and the clocks in Y are reset. Moreover, it is required that automata $A_1, \ldots, A_m$ are synchronized, meaning that they can advance only when all of them can read a given symbol in $H \cup L$.

The HL-Timed Automaton $\mathcal{A}$ recognizes *timed words* $\omega = (\omega_1, \omega_2)$ such that $\omega_1 : \mathbb{N} \to (L \cup H)$ and $\omega_2 : \mathbb{N} \to T$, with T a time domain. Intu-

itively, ω describes the behavior of a system that performs action $\omega_1(i)$ at time $\sum_{h=0}^{i} \omega_2(h)$. We denote with ω_L the projection of ω on L, namely the (possibly finite) sequence $(\omega_1(i_1), \sum_{h=0}^{i_1} \omega_2(h)), (\omega_1(i_2), \sum_{h=i_1+1}^{i_2} \omega_2(h)), \ldots$ such that, for each index i_j, $\omega_1(i_j) \in L$ and, for each $i_j < k < i_{j+1}$, $\omega_1(k) \in H$. The sequence ω_L describes the observable part of ω. Moreover, we denote with F_ω the function that gives the index in ω of the low action in position j in ω_L, namely $F_\omega(j) = i_j$.

The *language* accepted by $\mathcal{A}$ (denoted by $\mathcal{L}(\mathcal{A})$) is the set of timed words recognized by $\mathcal{A}$.

We refer to [7] for a formal definition of the behavior of HL-Timed Automata.

2.1 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 $a \in H$, d be a finite sequence $(a_1, t_1), \ldots, (a_h, t_h)$ with $a_1, \ldots, a_h \in L$ and $t_1, \ldots, t_h \in T$, and i be an index $1 \leq i < h$. We define the $No-privacy$ property $NPr(d, i, a)$ for a HL-Timed Automaton $\mathcal{A}$ as follows:

for each $\omega \in \mathcal{L}(\mathcal{A}), d \leq_P \omega_L$ implies $a \in \{\omega_1(F_\omega(i)+1), \ldots, \omega_1(F_\omega(i+1)-1)\}$.

Intuitively, $NPr(d, i, a)$ expresses that, whenever the sequence d of low symbols is read, the high symbol a is read between the low level actions a_i and a_{i+1}, and, therefore, there is an information flow from high level to low level, namely information on the secret behavior can be inferred from information on the observable behavior.

2.2 Checking No-Privacy

In [7] it is proved that the property $NPr(d, i, a)$ is decidable.

More precisely, an algorithm is given which takes an HL-Timed Automaton $\mathcal{A}$ and a property $NPr(d, i, a)$, and checks whether $NPr(d, i, a)$ holds or not. The algorithm exploits the *region graph* [1] of $\mathcal{A}$ and simulates all the behaviors of $\mathcal{A}$ whose observable prefix is described by d.

The algorithm is linear w.r.t. the size of the region graph, which, in turn, is exponential in size w.r.t. the size of the automaton.

Now, if we consider the observable sequence $d = (r_e, 10)(a_e, 20)(r_e, 40)$, then $NPr(d, 2, a_c)$ holds. This means that whenever the two requests from the user to the malicious site are separated by 30 units of time, the malicious site is sure that the web page w is in the cache of the user.

On the contrary, if we take $d = (r_e, 10)(a_e, 20)(r_e, 200)$, then $NPr(d, 2, a_c)$ does not hold, since the delay of 190 units of time between the two requests from the user can be due either to the communication between the user and the malicious site, or to the communication between the user and w.

```
--  Cache
INIT
Ac = 1  &  x = 0
TRANS
((Ac = 1) & (!(act = r_c)) & (next(Ac) = 1) & (next(x) = 0)) |
((Ac = 1) & (act = r_c) & (next(Ac) = 2) & (next(act) = nop) &
                                          (next(x) = 1)) |
((Ac = 2) & (act = nop) & (x < 5) & (next(Ac) = 2) & (next(act) = nop) &
                                          (next(x) = x + 1)) |  -- idle
((Ac = 2) & (act = nop) & (x >= 2) & (next(Ac) = 1) & (next(act) = a_c) &
                                          (next(x) = 0))
```

Fig. 2. Cache automaton

3 The SMV approach

In this section we show how the model checker SMV [8,2,11] can be used to automatically verify the No-privacy property as defined in the previous section. Since SMV only handles finite state systems, we use a discrete time model rather than the continuous time model of Timed Automata.

3.1 Basic notation

Here we shortly describe the SMV frame. We refer to [2] for more details.

In SMV we can define a process (i.e. a *Finite State System*, FSS) with its transition relation. Thus for each process P we have a boolean function (also named P) defining the transition relation of P. We use C-like identifiers to denote arrays of boolean variables ranging on P present states. We use ($'$) to denote the next state operator. Thus, e.g. if $\mathbf{x}$ ranges on P present states then $\mathbf{x}'$ ranges on P next states. The boolean expression $P(\mathbf{x}, \mathbf{x}')$ defines the transition relation of process P. Since $P(\mathbf{x}, \mathbf{x}')$ univocally defines a process we will also use $P(\mathbf{x}, \mathbf{x}')$ as a name for P.

We use *synchronous* parallel process composition. Thus given processes $P(\mathbf{x}, \mathbf{x}')$ and $Q(\mathbf{x}, \mathbf{x}')$ their composition is represented by the process $R(\mathbf{x}, \mathbf{x}')$ $= P(\mathbf{x}, \mathbf{x}') \wedge Q(\mathbf{x}, \mathbf{x}')$.

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

The set of initial states of a process can be defined with a boolean function returning 1 (true) on the set of initial states.

SMV comment lines start with `--`. SMV keyword `MODULE` is followed by the module name. Module `main` must always be present. Keyword `VAR` is followed by the list of the module variables together with their (finite) ranges. SMV uses the symbol "&" for the boolean "and" ($\wedge$), the symbol "|" for the boolean "or" ($\vee$) and the symbol "!" for the boolean "not" ($\neg$).

In Fig. 2 we specify using SMV the cache automaton of fig. 2.

Keyword `INIT` is followed by the boolean expression defining the set of initial states. (It requires that automaton state 1 is active and clock x equals 0.) Keyword `TRANS` is followed by the boolean expression defining the transi-

tion relation. (It relates the current automaton state (**Ac**), action performed (**act**) and clock value (**x**) with the next automaton state, action performed and clock value.) Note that SMV next state operator ($'$) is denoted by `next`.

3.2 The No-privacy Property

Let $P(\mathbf{x}, \mathbf{a}, \mathbf{x}', \mathbf{a}')$ be an FSS, with $L \cup H$ the range of $\mathbf{a}$. Let $I(\mathbf{x}, \mathbf{a})$ be the (boolean expression defining the) set of initial states for P. An (I, P) sequence for P is a sequence of states ($\mathbf{x}_0$, $\mathbf{a}_0$), ($\mathbf{x}_1$, $\mathbf{a}_1$), ... such that: $I(\mathbf{x}_0, \mathbf{a}_0) = 1$ and for all $i = 0, 1, \ldots$ $P(\mathbf{x}_i, \mathbf{a}_i, \mathbf{x}_{i+1}, \mathbf{a}_{i+1}) = 1$. An ($I$, P) sequence can be finite as well as infinite.

Let ω be an (I, P) sequence for P. We denote with ω_L the projection of ω on L. That is $\omega_L = <\omega(i_0), \omega(i_1), \ldots>$, where for each index i_j, $\omega(i_j) \in L$ and for each $i_j < k < i_{j+1}$, $\omega(k) \in H$. We denote with F_ω the function that gives the index in ω of the low action in position j in ω_L, namely $F_\omega(j) = i_j$.

An ($\mathbf{a}$, L) *spy* sequence of length n for P is a sequence $<\mathbf{a}_0, [t^0_{min}, t^0_{max}), \mathbf{a}_1, [t^1_{min}, t^1_{max}), \ldots \mathbf{a}_{n-1}, [t^{n-1}_{min}, t^{n-1}_{max}), \mathbf{a}_n>$, where: 1. for all $i = 0, \ldots n$, $\mathbf{a}_i \in L$; 2. for all $i = 0, \ldots n-1$, t^i_{min}, t^i_{max} are non-negative integers; 3. for all $i = 0, \ldots n-1$, $0 \leq t^i_{min} < t^i_{max}$.

Let $d = <\mathbf{a}_0, [t^0_{min}, t^0_{max}), \mathbf{a}_1, [t^1_{min}, t^1_{max}), \ldots \mathbf{a}_{n-1}, [t^{n-1}_{min}, t^{n-1}_{max}), \mathbf{a}_n>$ be an ($\mathbf{a}$, L) spy sequence of length n for P and ω be an (I, P) sequence for P. We write $d \leq \omega$ iff for all $0 \leq i \leq n$, $\omega(F_\omega(i)) = \mathbf{a}_i$ and for all $0 \leq i < n$, $t^i_{min} \leq F_\omega(i+1) - F_\omega(i) < t^i_{max}$.

Let d be an ($\mathbf{a}$, L) spy sequence of length n for P, $0 \leq i < n$ and $a \in H$. The *No-privacy* property $NPr(d, i, a)$ for P is defined as follows. Property $NPr(d, i, a)$ holds iff for each (I, P) sequence ω for P, if $d \leq \omega$ then $a \in \{\omega(F_\omega(i)+1), \ldots, \omega(F_\omega(i+1)-1)\}$.

3.3 SMV Model

In this section we describe how SMV can be used to check No-privacy.

Fig. 3 gives a list of the variables used together with their (finite) ranges. Keyword `DEFINE` in Fig. 3 is used to define constants as well as to assign names to expressions.

Figs. 2, 4, 5 give, respectively, our SMV model for the cache process (Ac) in Fig. 1, the user process (Au) in Fig. 1, the web process (Aw) in Fig. 1.

We check the No-privacy property using an *observer* process. Essentially our observer checks that the No-privacy property holds along a given computation path. That is the observer checks that on a given computation sequence satisfying our temporal constraints (i.e. the given *spy* sequence) the high action `a_w` occurs where (w.r.t. low actions) and when (w.r.t. time constraints) we required. This is easily done by implementing our No-privacy definition in section 3.2 with a process that we call *observer*. Our SMV model for the observer process is given in Fig. 6.

```
MODULE main
VAR
Au : 1 .. 7; -- user state
Ac : 1 .. 2; -- cache state
Aw : 1 .. 2; -- web state
z : 0 .. 7; -- user timer (Au)
x : 0 .. 7; -- cache timer (Ac)
y : 0 .. 255;  -- web timer Aw
act : {a_e, r_e, a_c, r_c, a_w, r_w, nop};  -- req/ack action set
seen : boolean;  -- 1 only after action a_w has been seen
obs : -1 .. 4;
t : 0 .. 255;

DEFINE
H := {a_c, r_c, a_w, r_w, nop}; -- high actions
L := {a_e, r_e};  -- low action
D2_min := 18;
D2_max := 22;
D3_min := 13;
D3_max := 220;
```

Fig. 3. State variables and their ranges

```
--  USER
INIT
(Au = 1) & (act = nop) &  (z = 0)
TRANS
((Au = 1) & (next(Au) = 1) & (next(act) = nop) & (next(z) = 0)) | -- idle
((Au = 1) & (next(Au) = 2) & (next(act) = r_e) & (next(z) = 0)) |
((Au = 2) & (next(Au) = 2) & (next(act) = nop) & (next(z) = 0)) | -- idle
((Au = 2) & (next(Au) = 4) & (next(act) = a_e) & (next(z) = 0)) |
((Au = 4) & (z < 2) & (next(Au) = 4) & (next(act) = nop) &
                                                  (next(z) = z + 1)) |
((Au = 4) & (z >= 1) & (next(Au) = 3) & (next(act) = r_c) &
                                                  (next(z) = 0)) |
((Au = 4) & (z >= 1) & (next(Au) = 5) & (next(act) = r_w) &
                                                  (next(z) = 0)) |
((Au = 3) & (act = r_c) & (next(Au) = 3) & (next(z) = 0))   |
((Au = 3) & (act = nop) & (next(Au) = 3) & (next(z) = 0)) |  -- idle
((Au = 3) & (act = a_c) & (next(Au) = 7) & (next(act) = nop) &
                                                  (next(z) = 0)) |
((Au = 5) & (act = r_w) & (next(Au) = 5) & (next(z) = 0)) | -- idle
((Au = 5) & (act = nop) & (next(Au) = 5) & (next(z) = 0)) |  -- idle
((Au = 5) & (act = a_w) & (next(Au) = 7) & (next(act) = nop) &
                                                  (next(z) = 0)) |
((Au = 7) & (next(Au) = 7) & (z < 2) & (next(act) = nop) &
                                                  (next(z) = z + 1)) |
((Au = 7) & (next(Au) = 6) & (z >= 1) & (next(act) = r_e) &
                                                  (next(z) = 0)) |
((Au = 6) & (next(Au) = 6) & (next(act) = nop) & (next(z) = 0)) |  -- idle
((Au = 6) & (next(Au) = 1) & (next(act) = a_e) & (next(z) = 0))
```

Fig. 4. User automaton

SMV keyword `SPEC` is followed by a formula describing the property to be verified. In our case the property we want to verify is that along all computation paths (**AG** in SMV parlance) the No-privacy property holds. This indeed amounts to check the No-privacy property NPr as defined in section 3.2. Our specification is given in Fig. 7.

Note that since SMV semantics is synchronous as a matter of fact all `INIT` expressions are put in logical "and". The same holds for `TRANS` expressions.

```
--  Web
INIT
Aw = 1 & y = 0
TRANS
((Aw = 1) & !(act = r_w) & (next(Aw) = 1) & (next(y) = 0))  |
((Aw = 1) & (act = r_w) & (next(Aw) = 2)  & (next(act) = nop) &
                                            (next(y) = 1)) |
((Aw = 2) & (act = nop) & (y < 250) & (next(Aw) = 2) & (next(act) = nop) &
                                            (next(y) = y + 1))  |
((Aw = 2) & (act = nop) & (y >= 100) & (next(Aw) = 1) &
                                            (next(act) = a_w) &(next(y) = 0))
```

Fig. 5. Web automaton

```
--  Obs
INIT
obs = 0 & t = 0 & seen = 0
TRANS
-- fail state used when spy sequence (d) not found ------------------
((obs = -1) & (next(obs) = obs) & (next(t) = 0) &
    (next(seen) = seen))  |  -- obs fail state
((obs = 0) & (act = nop) & (next(obs) = obs) & (next(t) = 0) &
    (next(seen) = seen)) |   -- idling on nop init state
-- handling r_e -----------------------------------------------------
((obs = 0) & (act = r_e) & (next(obs) = 1) & (next(t) = 0) &
    (next(seen) = seen)) |  -- seen r_e
-- handling H* a_e --------------------------------------------------
((obs = 1) & (act in H) & (t < D2_max) & (next(obs) = 1) &
    (next(t) = t + 1) & (next(seen) = seen)) |  -- seen H*
((obs = 1) & (act = a_e) & (D2_min <= t) & (t < D2_max) &
    (next(obs) = 2)&(next(t) = 0)&(next(seen) = seen)) | -- seen H* a_e
((obs = 1) & !((act in H)|(act = a_e)) & (t < D2_max) &
    (next(obs) = -1) & (next(t) = 0)  & (next(seen) = seen)) | -- fail
((obs = 1)&(t >= D2_max)&(next(obs) = -1)&(next(t) = 0) ) | -- too late
((obs = 1) & (act = a_e) & (t < D2_min) & (next(obs) = -1) &
    (next(t) = 0)  & (next(seen) = seen))  |  -- too soon
-- handling H* (a_w) r_e, no reset of t after a_w -------------------
((obs = 2) & (act in H) & (act != a_w) & (t < D3_max) &
    (next(obs) = obs)&(next(t) = t + 1)&(next(seen) = seen)) | --seen H*
((obs = 2) & (act = a_w) & (t < D3_max) & (next(obs) = obs) &
    (next(t) = t + 1)  & (next(seen) = 1)) |  -- seen H*
((obs = 2) & (act = r_e) & (D3_min <= t) & (t < D3_max) &
    (next(obs) = 4)&(next(t) = 0)&(next(seen) = seen)) | -- seen H* r_e
((obs = 2) & (act = r_e) & (t < D3_min) & (next(obs) = -1) &
    (next(t) = 0)  & (next(seen) = seen)) |   -- too soon
((obs = 2) & !((act in H)|(act = r_e)) & (t < D3_max) &
    (next(obs) = -1) & (next(t) = 0)  & (next(seen) = seen)) | -- fail
((obs = 2) & (t >= D3_max) & (next(obs) = -1) & (next(t) = 0)  &
    (next(seen) = seen)) |  -- too late
((obs = 4) & (next(obs) = obs)  & (next(t) = 0) &
    (next(seen) = seen)) -- final state
```

Fig. 6. Observer automaton

```
-- Npr({r_e, [18, 20), a_e, [180, 220), r_e}, 2, a_w)
SPEC
AG (obs = 4 -> seen = 1)
```

Fig. 7. SMV specification

```
-- Log with D3_min = 13

Iteration 0: Early evaluation of specifications
-- specification AG (obs = 4 -> seen = 1) is true

Iteration 1: Early evaluation of specifications
-- specification AG (obs = 4 -> seen = 1) is true

.......................

Iteration 381: Early evaluation of specifications
-- specification AG (obs = 4 -> seen = 1) is true

The verification is complete.

resources used:
user time: 2.51 s, system time: 0.04 s
BDD nodes allocated: 64184
Bytes allocated: 2293760
BDD nodes representing transition relation: 776 + 10
```

Fig. 8. A glimpse of the SMV output when the No-privacy Property holds

However for clarity we kept the description of processes separated.

The actual SMV code is obtained by concatenating Figs. 3, 4, 2, 5, 6, 7.

Fig. 8 sketches SMV output for our SMV program. In this case the property is verified.

Changing the value of `D3_min` in Fig. 3 to "`D3_min := 12`" the No-privacy Property does not hold any longer. The results of running SMV in this case are sketched in Fig. 9 where a counter example is given. Intuitively, by making the temporal window too large we weaken our constraints and thus we are no longer able to guarantee that on all computation sequences satisfying our temporal constraints the high action `a_w` occurs as we expect.

4 The *tSPA* process algebra approach

4.1 A simple model for describing real-time systems

In this section, we introduce the Timed Security Process Algebra [6] (*tSPA*).

Our time modeling approach is fairly simple: there is one special *tick* action to represent the elapsing of time while all the other actions are durationless [2]. This is reasonable if we choose a time unit such that the actual time of an action is negligible w.r.t. the time unit. 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 the two global synchronizations on action *tick* (that represents the elapsing of one time unit), all the processes proceed asynchronously by performing durationless actions. Moreover, another feature of the *tSPA* is the so-called *maximal progress assumption* according to which *tick* actions have

[2] These are the features of the so-called *fictitious clock* approach

```
-- Log with D3_min = 12

Iteration 0: Early evaluation of specifications
-- specification AG (obs = 4 -> seen = 1) is true

.........................

Iteration 33: Early evaluation of specifications
-- specification AG (obs = 4 -> seen = 1) is true

Iteration 34: Early evaluation of specifications
-- specification AG (obs = 4 -> seen = 1) is false
-- as demonstrated by the following execution sequence
```

state 1.1:	state 1.2:	state 1.11:	state 1.20:	state 1.25:	state 1.31:
D3_max = 220	Au = 2	t = 8	t = 17	Ac = 2	Au = 7
D3_min = 12	act = r_e	state 1.12:	state 1.21:	x = 1	act = nop
D2_max = 22	state 1.3:	t = 9	Au = 4	act = nop	t = 9
D2_min = 18	act = nop	state 1.13:	act = a_e	t = 3	state 1.32:
L = a_e,r_e	obs = 1	t = 10	t = 18	state 1.26:	z = 1
H = a_c,r_c,	state 1.4:	state 1.14:	state 1.22:	x = 2	t = 10
a_w,r_w,nop	t = 1	t = 11	z = 1	t = 4	state 1.33:
Au = 1	state 1.5:	state 1.15:	act = nop	state 1.27:	z = 2
Ac = 1	t = 2	t = 12	obs = 2	x = 3	t = 11
Aw = 1	state 1.6:	state 1.16:	t = 0	t = 5	state 1.34:
z = 0	t = 3	t = 13	state 1.23:	state 1.28:	Au = 6
x = 0	state 1.7:	state 1.17:	z = 2	x = 4	z = 0
y = 0	t = 4	t = 14	t = 1	t = 6	act = r_e
act = nop	state 1.8:	state 1.18:	state 1.24:	state 1.29:	t = 12
seen = 0	t = 5	t = 15	Au = 3	x = 5	state 1.35:
obs = 0	state 1.9:	state 1.19:	z = 0	t = 7	Au = 1
t = 0	t = 6	t = 16	act = r_c	state 1.30:	act = a_e
	state 1.10:		t = 2	Ac = 1	obs = 4
	t = 7			x = 0	t = 0
				act = a_c	
				t = 8	

```
No more specifications left.

resources used:
user time: 0.88 s, system time: 0.05 s
BDD nodes allocated: 18427
Bytes allocated: 1638400
BDD nodes representing transition relation: 758 + 10
```

Fig. 9. A glimpse of SMV output when the No-privacy Property does not hold

lower priority w.r.t. internal τ actions: a communication (or an internal activity τ) 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. Finally, *tSPA* offers a construct, called the *idling* operator, for delaying the execution of the currently executable actions of its argument process.

We formally introduce the syntax and semantics of our timed language *tSPA*. We have a set $\mathcal{L}$, ranged over by l, of visible actions. $\mathcal{L}$ is $I \cup O$ where $I = \{a, b, c \ldots\}$ is the set of input action and $O = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$ of output actions. A special action τ models an internal computation, i.e. it is not visible by an external observer. We also have a complementation function $(^{-}) : \mathcal{L} \mapsto \mathcal{L}$, such that $\forall l \in \mathcal{L} : \overline{\overline{l}} = l$. To reflect different levels of secrecy the set $\mathcal{L}$ of visible actions is partitioned into two sets *ActH* (or simply H)

$$\frac{}{\alpha.E \xrightarrow{\alpha} E} \qquad \frac{E_1 \xrightarrow{a} E_1'}{E_1 + E_2 \xrightarrow{a} E_1'} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 + E_2 \xrightarrow{a} E_2'} \qquad \frac{E_1 \xrightarrow{tick} E_1' \quad E_2 \xrightarrow{tick} E_2'}{E_1 + E_2 \xrightarrow{tick} E_1' + E_2'}$$

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 \| E_2 \xrightarrow{a} E_1' \| E_2} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 \| E_2 \xrightarrow{a} E_1 \| E_2'} \qquad \frac{E_1 \xrightarrow{l} E_1' \quad E_2 \xrightarrow{\overline{l}} E_2'}{E_1 \| E_2 \xrightarrow{\tau} E_1' \| E_2'}$$

$$\frac{E_1 \xrightarrow{tick} E_1' \quad E_2 \xrightarrow{tick} E_2' \quad \forall l \in \mathcal{L} \quad \neg(E_1 \xrightarrow{l} \wedge E_2 \xrightarrow{\overline{l}})}{E_1 \| E_2 \xrightarrow{tick} E_1' \| E_2'}$$

$$\frac{Z = E \quad E \xrightarrow{\alpha} E'}{Z \xrightarrow{\alpha} E'} \qquad \frac{E_1 \xrightarrow{\alpha} E_1'}{E_1 \backslash F \xrightarrow{\alpha} E_1' \backslash F} (\alpha \notin F \cup \overline{F})$$

$$\frac{E \not\xrightarrow{tick} \quad E \not\xrightarrow{\tau}}{\iota(E) \xrightarrow{tick} \iota(E)} \qquad \frac{E \xrightarrow{tick} E'}{\iota(E) \xrightarrow{tick} \iota(E')} \qquad \frac{E \xrightarrow{a} E'}{\iota(E) \xrightarrow{a} E'}$$

Fig. 10. Operational semantics for timed *SPA*, where l ranges over $\mathcal{L}$, a ranges over $\mathcal{L} \cup \{\tau\}$ and α over Act.

and $ActL$, closed by complementation function. Let $tick$ be the special action used to model time elapsing and let $Act = \mathcal{L} \cup \{\tau\} \cup \{tick\}$, ranged over by $\alpha, \beta, \ldots$, while $\mathcal{L} \cup \{\tau\}$ is ranged over by $a, b \ldots$.

The syntax for *tSPA* terms is the following:

$$E ::= \mathbf{0} \mid \alpha.E \mid E_1 + E_2 \mid E_1 \| E_2 \mid E \backslash F \mid \iota(E) \mid Z$$

where $\alpha \in Act$, $F \subseteq \mathcal{L}$ 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 in the scope of some prefix operator $\alpha.E'$.)

The formal behaviour of Timed *SPA* terms is described by means of the *labelled transition system* (LTS, for short) $< Proc^t, Act, \{\xrightarrow{\alpha}\}_{\alpha \in Act} >$, where $\xrightarrow{\alpha}_{\alpha \in Act}$ is the least relation between Timed *SPA* terms induced by axioms and inference rules of Figure 10. Let us consider the following relations between *SPA* terms: $E \stackrel{\tau}{\Longrightarrow} E'$ (or $E \Longrightarrow E'$) if $E \xrightarrow{\tau}_* E'$, and for $\alpha \neq \tau$, $E \stackrel{\alpha}{\Longrightarrow} E'$ if $E \xrightarrow{\tau}_* \xrightarrow{\alpha} \xrightarrow{\tau}_* E'$ (where $\xrightarrow{\tau}_*$ is the reflexive and transitive closure of the $\xrightarrow{\tau}$ relation). Let $Der(E)$ be the set of derivatives of E, i.e. the set of processes that can be reached through the transition relations. Let $Sort(E) = \{\alpha | \alpha \in Act, \exists E' \in Der(E), E' \xrightarrow{\alpha}\}$.

The equivalence relation we consider is the timed version of observation equivalence. This equivalence permits to abstract, to some extent, from the internal behaviour of the systems, represented by the internal actions.

A relation $\mathcal{R} \subseteq Proc^t \times Proc^t$ is a timed weak simulation iff for every $(p, q) \in \mathcal{R}$ we have:

- if $p \xrightarrow{a} p'$ then there exists q' s.t. $q \stackrel{a}{\Longrightarrow} q'$ and $(p', q') \in \mathcal{R}$,
- if $p \xrightarrow{tick} p'$ then there exists q' s.t. $q \stackrel{tick}{\Longrightarrow} q'$ 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 with $\approx_t$ the union of all the timed weak bisimulations.

Example 4.1 *We may easily model time-out constructs in tSPA. Assume*

$t_1 \leq t_2$ and define a process

$$Time\_out(t_1, t_2, A, B) = tick^{t_1}.\iota(A) + tick^{t_2}.\tau.B$$

which may decide to act as the process A only for a certain amount of time, say between t_1 and t_2 units of time, and that after the elapsing of t_2 units of time, if no choice has been previously performed, it behaves as B.

In fact, by looking at the semantics rules, we see that $Time\_out(t_1, t_2, A, B)$ may perform a sequence of t_1 tick actions. Then, the system may perform other $t_2 - t_1$ tick, unless A resolves the choice by performing an action. Possibly, after t_2 units of time, through the τ action the process is forced to act as B.

4.2 Security properties in a real-time setting

In this section we present some information flow security properties, e.g. see [6]. In particular, these are the rephrasing in a real-time setting of the *non interference* theory proposed in [5].

The central property is the so called *Non deducibility on Composition* (*NDC*, for short). Its underlying idea is that the low level view of the system behaviour must be invariant w.r.t. the composition of the system with any high user. This means that the low level users cannot tell anything about the high level activity since, for them, the system is always the same. Indeed, there is no possibility of establishing a communication (i.e. sending information).

Let $\mathcal{E}_H^t$ be the set of all potential high level users, i.e. processes E s.t. $sort(E) \subseteq H \cup \{tick\} \cup \{\tau\}$ [3] . Then, timed Bisimulation Non-Deducibility on Compositions (*tBNDC* for short) is defined as follows:

$E \in tBNDC$ if and only if $\forall \Pi \in \mathcal{E}_H^t$ we have $(E \| \Pi) \backslash H \approx_t E \backslash H$.

Due to the presence of the universal quantification *tBNDC* is not very easy to be checked. Thus, other properties which can be checked more efficiently have been defined in our real-time setting (see [6]). Here, we present only the *tBSNNI* property which is an approximation of *tBNDC*, since only one high level process is considered. This process is the so called Top_H^{tick}, i.e:

$$Top_H^{tick} = \sum_{a \in H \cup \overline{H}} a.Top_H^{tick}$$

Note that this property can be checked in polynomial time in the size of finite systems E.

[3] Actually, the definition of $\mathcal{E}_H^t$ is slightly more complex. We refer the interested reader to [6] for a full discussion.

4.3 An example

We formalize in our framework the example of [3]. This shows how some information in the cache of web browsers might be leaked to the external world. In particular, it could be possible to detect whether or not a web browser has recently accessed a particular web page. In this context, the high level component is the cache. The low level processes must not deduce anything about the cache. Thus, the low level view of the system must be the same for whatever cache is present. Unfortunately, this is not the case.

Consider the following description of the system under investigation:

$$\begin{aligned} B &= \iota(\overline{r_e}.\iota(a_e.Time\_out(1,2,Cache,Web))) \\ Cache &= cached_w.B \\ Web &= Time\_out(100,250,B,B) \end{aligned}$$

The system B works as follows. We assume that there is an applet in the system which requests the web page. The system allows the applet to perform a request (denoted by the action r_e) to a site e, which may answer, by performing action a_e. Then, the applet requests a particular web page w, i.e. the web page it wants to know whether or not it is in the cache. We have two different possibilities depending on the presence of this page in the cache. If so, the system may again perform a request to the site e, in at most 2 units of time; otherwise, the system will access the web and the request to the web site e cannot be performed before 100 units of time. Therefore, we can simply check that the cache may be exploited to leak some information since the low level view of the system depends on the status of the cache.

Let the set of high actions H be $\{cached_w\}$. We can show that $B \setminus H \not\approx_t (B \| Top_H^{tick}) \setminus H$.

In particular, note that $(B \| Top_H^{tick}) \setminus H) \stackrel{\overline{r_e}}{\Longrightarrow}\stackrel{a_e}{\Longrightarrow}\stackrel{tick}{\Longrightarrow}\stackrel{\overline{r_e}}{\Longrightarrow}$, while $B \setminus H$ is not able to perform such a computation. As a matter of fact, $B \setminus H$ represents a system where no $cached_w$ action is possible. This models an empty cache. While $(B \| Top_H^{tick})$ represents a system whose cache has the page w, since Top_H^{tick} is able to perform $\overline{cached_w}$. Indeed, we have that B is not $tBSNNI$ and consequently that B is not $tBNDC$.

5 Conclusions

We have analyzed a time-dependent web attack with three different approaches. All of them are able to detect the attack.

The three approaches adopt different communication models: HL-Timed Automata and SMV assume synchronous parallel composition, tSPA Process Algebra assumes asynchronous parallel composition plus synchronous communication. To describe processes on the network the asynchronous parallel

composition is more natural, and moreover tSPA has ad hoc constructs for the purpose. On the other hand, HL-Timed Automata and SVM permit to describe the communication structure. To prove properties, in all the approaches, one must consider a sequential version of the system, and the cost of this reduction is exponential size on the number of components.

The three approaches consider three different time models. In HL-Timed Automata one has dense time domain and clocks which can be reset, SVM is a discrete time version of HL-Timed Automata, and tSPA has a discrete time domain and a tick transition. On the one hand, tSPA and SVM have the same time-expressiveness, but SVM is more succinct with respect to tSPA. In fact to simulate n discrete clocks a process in tSPA must be exponential on n. On the other hand HL-Timed Automata model is more time-expressive than SVM and tSPA. Now, in the framework of networks sometimes the discrete time approximation is too restrictive and one needs dense time, 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.

We observe also that in tSPA we have considered a non interference property, namely timed bisimulation non deducibility on composition. Its underlying idea is that no high level activity can be detected by only observing the low level behavior. The notion of observation is characterized through a process equivalence. The non interference property is based on the fact that the system can be composed with processes that are able to synchronize with its high and low actions. On the contrary, in the case of both HL-Timed Automata and SMV we have considered a different non interference property, which is based on the fact that the system can be composed with an environment that is able to synchronize only with its low actions.

The tBNDC property considered by tSPA seems to be more restrictive than the No-privacy Property defined for HL-Timed Automata and SVM, and so it might be used to discover other kinds of attack with respect to the No-privacy Property. On the other hand, since this property is more selective, it could be possible to define with it more flexible information flow properties for the systems under investigation.

We note that although the formal frames of the three approaches presented here appear to be quite different, as a matter of fact the *formalization effort* (i.e. the time needed to define the model to be analyzed) seems to be essentially the same for all the three approaches.

Finally, we note that it was not the goal of this paper to address *scalability* issues.

References

[1] Alur, R., and D.L. Dill: *A theory of timed automata.* Theoretical Computer Science **126** (1994), 183–235.

[2] Burch, J.R., E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang: *Symbolic model checking:* 10^{20} *states and beyond.* Information and Computation **98**, 1992.

[3] Felten, E.W., and M.A. Schneider: Timing attacks on Web privacy. Proc. 7^{th} ACM Conference on Computer and Communications Security, 25–32, 2000.

[4] Focardi, R., and R. Gorrieri: *The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties.* IEEE Transactions on Software Engineering **23(9)** (1997), 550–571.

[5] Focardi, R., and R. Gorrieri: *A classification of security properties for process algebras.* Journal of Computer Security **3** (1995), 5–33.

[6] Focardi, R., R. Gorrieri, and F. Martinelli: Information flow analysis in a discrete-time process algebra. Proc. 13^{th} Computer Security Foundation Workshop, IEEE Computer Society Press, 2000.

[7] Lanotte, R., A. Maggiolo-Schettini, and S. Tini: *Privacy in real-time systems.* Proc. MTCS'01. Electronic Notes in Theoretical Computer Science **52.3** (2001).

[8] McMillan, K.L.: *Symbolic model checking.* Kluwer Academic Publishers, Massachusets, 1993.

[9] Volpano, D., and G. Smith: *Confinement properties for programming languages.* SIGACT News **29** (1998), 33–42.

[10] Smith, G., and D. Volpano: Secure information flow in a multi-threaded imperative language. Proc. ACM Symposium on Principles of Programming Languages, 1998, 355–364.

[11] URL: `http://www.cs.cmu.edu/~modelcheck/`.