The Timed Decentralized Label Model

2nd of June, 2015

Martin Leth Pedersen

Michael Hedegaard Sørensen

Aalborg University
Department of Computer Science
Student report
Selma Lagerlöfs Vej 300
Telefon 99 40 99 40
Fax 99 40 97 98
http://www.cs.aau.dk

Title: The Timed Decentralized Label Model

Theme: Information flow security, access control, security models, smart meter security

Project period:
SW10/DAT10, 2/2/2015 - 2/6/2015

Project group:
des1012f15

Participants:
Martin Leth Pedersen
Michael Hedegaard Sørensen

Supervisor:
René Rydhof Hansen

Copies: 4

Pages: 65

No. of appendix pages: 0

Finished: 2/6/2015

Synopsis:

Information flow and access control are subjects in the area of computer security where extensive research has been performed. However, research in regards to time-based information flow and access control have received limited attention and as such no time-based security models based on information flow and access control have been developed. The Timed Decentralized Label Model (TDLM) that takes timed information flow and access control into account was developed. The TDLM is based on timed automata and as such it is formally defined via the use of these, however an implementation of the model is yet to be developed. As such further study is needed in regards to a complete implementation of the TDLM which perhaps could be a new programming language or an extension to the implementation of the model the TDLM is based on. UPPAAL could be used in the implementation to statically verify that security policies are enforced correctly.

RESUME

Information flow and access control are subjects in the area of computer security where extensive research has been performed. However, research in regards to time-based information flow and access control have received limited attention and as such no time-based security models based on information flow and access control have been developed. As a result of this, security in time-sensitive systems can be difficult to model as there is no well-documented way of doing so.

The Decentralized Label Model (DLM) was chosen as a basis for developing a security model that takes time into account, resulting in the Timed Decentralized Label Model (TDLM) which extends the DLM with time-based constructs such that security policies can be expressed in terms of who and when access to data are allowed.

The clock expressions introduced by the TDLM can be placed on principals in a security policy to restrict these principals' access to data based on time. This allows the TDLM to express security policies where principals are restricted in certain time intervals, for example a principal may only be allowed to access some data once every month which can be expressed via the use of clocks with certain clock parameters.

The TDLM also allows for clocks in the principal hierarchy which restricts acts-for relationships between principals based on time intervals resulting in the possibility of defining when certain principals trust other principals to act on their behalf.

The TDLM is subject to further study in the form of formal proofs of the components introduced in the DLM which still should hold for the TDLM. Furthermore, a complete implementation of the TDLM could be constructed perhaps in the form a new programming language or an extension to the implementation of the DLM that makes use of UPPAAL to statically verify that security policies are enforced correctly.

PREFACE

This report studies security models in regards to secure information flow and access control resulting in the construction of a timed extension to the well-known Decentralized Label Model to be able to model time-sensitive systems such as a smart meter system.

The report is written as a master's thesis by a Software Engineering student and a Computer Science student.

Chapter 1 introduces the problem area and narrows the scope of the report.

Chapter 2 presents the underlying theories needed to understand how to approach the problem.

Chapter 3 presents the extension to the Decentralized Label Model named the Timed Decentralized Label Model. The extension adds concepts to the Decentralized Label Model such that time can be expressed in policies and the principal hierarchy. Furthermore, the Decentralized Label Model's rules for complete safe relabeling is adapted to fit the extension.

Chapter 4 uses the Timed Decentralized Label Model in a concrete real-world example to illustrate the practical use of the model.

Chapter 6 concludes on the project.

We would like to thank Daniel Lux from Seluxit ApS for providing us with insight on how the Danish smart meter/smart grid system works. Furthermore, we would like to thank René Rydhof Hansen for his valuable supervision throughout the project.

CONTENTS

CHAPTER 1

INTRODUCTION

Extensive research has been performed in the area of information flow security and access control [15, 7, 10, 4, 5, 6, 8], however well-known security models such as Paralocks, the Decentralized Label Model (DLM), and Bell-LaPadula does not take time-based access to information into account. Research has been performed in the area of timed security protocols and the verification of these [9, 14], however this research are based on communication protocols and not on concrete security models. As a result of this, certain security systems are difficult to correctly model with the current available security models as these security systems are highly dependent on restricting access to information based on time, such that the entities in the systems only can access information at specific time intervals.

A smart meter system which is responsible for monitoring power consumption in households and reporting this to electrical companies, such that consumers can be billed accordingly cannot be modeled with the current security models, due to the fact that the system is highly dependent on being able to express that certain entities in the system only can gain access to data at specific time intervals. The need for expressing access to data at specific times is due to severe privacy issues for the consumers as the electrical companies can gain access to fine-grained power consumption data. This data could be used to derive the consumers' daily routines, which would be a breach of privacy [17]. Furthermore, if electrical consumption data is leaked to third parties, the data may be used to target advertisements, and plan criminal actions according to the consumers' behavioral patterns for example breaking-and-entering.

The smart meter system can also be used as a basis for a home automation platform as the system is able to retrieve real-time electrical prices, which can be used to determine when to turn on certain heavy power consuming devices such

as a dishwasher or a washing machine. However, this poses security issues if the smart meter platform is compromised as the intruder may be able to control devices connected to the smart meter, alter the electrical prices or billing information stored by the smart meter, or turn off the power to the household [3, 2].

In [22] we gave an overview of the problem in regards to time-based information flow control as well as an investigation of how this problem could be solved with focus on smart meter systems and the issues related to these. We performed an investigation of several well-known security models to determine which of these that would be most suitable for modeling smart meter systems. This lead to the preliminary conclusion that the DLM and Paralocks were the most promising candidates. Based on this investigation, we proposed two extensions to the DLM and Paralocks respectively which extended these well-known security models with time-based expressiveness. These extensions were based on a smart meter system as the one described above.

1.1 Narrowing the Scope: The Decentralized Label Model or Paralocks?

Before a further investigation of the ideas proposed in [22] can be performed, the research scope will have to be narrowed due to limited resources allocated to this project meaning that only one of the security models and their respective extension will be investigated. However, the results of an investigation of one security model may be applicable to the other to some extent.

In terms of comparing the DLM and Paralocks in regards to expressiveness, an investigation of this was done in [22] which resulted in the discovery that both extended models possesses the necessary expressiveness for modeling security systems that are time dependent. As such the choice of which model to further study is based on available documentation, estimated complexity, and implementation possibilities.

In regards to available documentation, the DLM is easier to comprehend than Paralocks and more research has been done using the DLM than Paralocks. We estimate the complexity of the extension and implementation of the DLM to be more accessible than the corresponding Paralocks extension/implementation because the DLM is more comprehend-able and as such the theoretical foundation is more intuitive to some extent. As such the DLM would be the focus of further study in regard to time-based information flow and access control.

CHAPTER 2

THEORIES

To be able to extend the DLM a deeper understanding of how the model works in detail is needed as well as an understanding of timed automata as they can be used to depict security policies involving time. In section 2.1 the DLM is explained and formally defined such that an extension to the model can be formulated. In section 2.2 timed automata are presented to be able to depict timed security policies.

2.1 The Decentralized Label Model

As the DLM is the core model studied in this report, a clear understanding of how the model is composed is essential. As the DLM is a decentralized model it supports computation in an environment with mutually distrusting entities where security policies are specified for each entity rather than decided by a central authority. The DLM assumes that users of a modeled system are external to said system meaning that programs running on the system are only leaking information if the information leaves the system. In addition to providing rules for information flow within a system, the DLM also specifies rules for handling release of information to external entities and rules for reading the information that is released [16].

Authority entities in the DLM are called *principals*, which can be users, groups or roles, where system processes has the capability to act on behalf of some set of principals. Principals can be authorized to *act-for* other principals, which means that if a principal a can act-for another principal a' then the principal a also gains the privileges of a', formally written as $a \succeq a'$. The *acts-for* relationship is both reflexive and transitive; meaning that if a principal a acts-for another principal a' then a would also be able to act-for a (reflexive), and if a' acts-for a'' then a would also be able to act-for a'' (transitive). These properties ensures that a hierarchy or

partial order of acts-for relationships can be formed as seen in Figure 2.1 - in the DLM called *the principal hierarchy*. The principal hierarchy may change over time, however removal of acts-for relationships does not occur frequently [16].

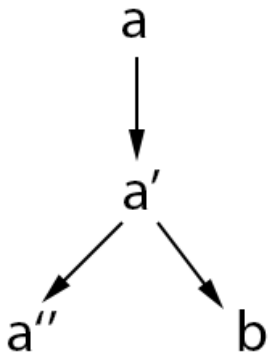

Figure 2.1: DLM principal hierarchy where a acts-for a' and a' acts-for a'' and b.

In the DLM data and programs can be associated with *labels* which expresses security policies with regards to principals participating in the system. A label is composed of one or more owners, where each owner permits a set of readers to observe the data protected by the policy. An example of a DLM label consisting of two security policies is $\{o_1 : r_1, r_2; o_2 : r_2, r_3\}$, where o_1 and o_2 are owners of the data and r_1, r_2, r_3 are readers. As this label is composed of more than one security policy, a principal that intents to gain access to information protected by this label can only do so if said principal can act-for at least one reader in each policy meaning that in this example only principals that acts-for r_2 can observe the data. If a policy contains no owner and an empty reader set then all principals in the system can gain access to the information [16]. Security labels are not defined on the actual data but rather on variables, input channels and output channels.

Data that are labeled with security policies can be relabeled by *restriction* or *declassification*. If a variable is labeled with a security policy an assignment to this variable may only occur if the relabeling that is performed is a *restriction*, meaning that the new label has more owners or less readers for the already existing reader sets. For example the relabeling from $\{o_1 : r_1, r_2\}$ to $\{o_1 : r_1\}$ is a restriction because less readers are permitted to observe the data. A restriction is considered *safe* as less principals are able to observe the data, however removing a reader r does not necessarily make the label more restrictive as there might be readers which r acts-for [16]. Relabeling can be safely done in the following ways:

- **Remove readers**: Readers can be removed from a label thus making it at least as restrictive as it originally was.
- **Add a policy**: A security policy can be safely added to a label as all policies associated with the label must be enforced.
- **Add a reader**: A reader can safely be added to a security policy's reader set if that reader can act-for an existing reader in the set. Formally a reader r can be added if another reader r' is already in the reader set and $r \succeq r'$.
- **Replace an owner**: An owner o of a security policy can safely be replace by another owner o' if $o' \succeq o$. The changed policy would then only allow processes to declassify the data if that process can act-for o' meaning that the security policy would be more restrictive.

Formally a relabeling from a label L_1 to another label L_2 is safe if there, through the relabeling rules above, is a rewrite sequence that transforms L_1 to L_2. The transformation is a restriction if L_1 is at most as restrictive as L_2 and L_2 is at least as restrictive as L_1, written as $L_1 \sqsubseteq L_2$. A formal definition of the $\sqsubseteq$ relationship requires that a function R which yields the implicitly allowed readers is defined in Definition 2.1.1, where I is the policy, $r(I)$ is the set of readers of I and p, p' are principals [16].

Definition 2.1.1: Safe Relabeling

$$R(I) = \{p \mid \exists_{p' \in r(I)}\ p \succeq p'\}$$

With the use of this function a formal definition of *the complete relabeling rule* can be defined as described in Definition 2.1.2, where $o(I)$ is the owner of the policy I and the equation describes the relationships $I \sqsubseteq J$ and $L_1 \sqsubseteq L_2$ where I, J are security policies and L_1, L_2 are labels [16]. For a proof that the complete relabeling rule is both sound and complete in regards to the formal semantics see [19]; meaning that the rule only allows safe relabelings and allows all safe relabelings.

Definition 2.1.2: The Complete Relabeling Rule

$$\begin{aligned} L_1 \sqsubseteq L_2 &\equiv \forall_{I \in L_1} \exists_{J \in L_2}\ I \sqsubseteq J \\ I \sqsubseteq J &\equiv o(J) \succeq o(I) \wedge R(J) \subseteq R(I) \\ &\equiv o(J) \succeq o(I) \wedge \forall_{p' \in r(J)} \exists_{p \in r(I)}\ p' \succeq p \end{aligned}$$

When executing a program within a system, values are often derived from other values, for example a new value may be derived by multiplying two other values. In

the DLM a derived value v must have a label that enforces the policies of the values used to derived v meaning that the label of v must be at least as restrictive as the combined label of the operands. Formally if we have two operands labeled with the labels L_1 and L_2 respectively the label for a derived value would be a join of these two which in terms is the union of the labels joined, as described in Definition 2.1.3.

Definition 2.1.3: Label Join

$$L_1 \sqcup L_2 = L_1 \cup L_2$$

In addition to the join operation, the DLM also defines a meet operation also referred to as the greatest lower bound of two labels. The greatest lower bound of two labels is the most restrictive label that can be relabeled to both of them [19]. Meet can be used to determine the labels of input automatically as join can be used for determining the labels of outputs automatically. The construction of the meet label of two labels A and B is the pairwise meets of all the policies contained in each label, however if there is no known acts-for relationship between the owners of the policies at compile-time then the meet is $\{\}$ because no other label can be relabeled to both policies. As an example consider the policies $Q = \{o : r_1, r_2\}$ and $P = \{o' : r'_1, r'_2\}$ which are in the labels A and B respectively. If o' can act-for or is equal to o then the meet of the policies Q and P would be $\{o : r_1, r_2, r'_1, r'_2\}$ else if o' is equivalent but not equal to o then the meet of the policies Q and P would be $\{o : r_1, r_2, r'_1, r'_2; o' : r'_1, r'_2, r_1, r_2\}$. Formally the meet operation can be defined as described in Definition 2.1.4.

Definition 2.1.4: Label Meet

$$A = \bigsqcup_i a_i$$
$$B = \bigsqcup_j b_j$$

$$A \sqcap B = \bigsqcup_{i,j} (a_i \sqcap b_j)$$

Relabeling can also be done by *declassification* where the restrictiveness of a label is reduced. This kind of relabeling can only be performed by owners of the policies or processes which can act-for these owners. When a process acts-for a set of principals it is executing with the authority of the principals that it acts-for, meaning that a program running with the authority of a principal can declassify data by adding readers to policies or removing policies entirely [16]. This means that a process may weaken or alter the policies that are owned by principals that the process can act-for. Formally, a process can declassify by relabeling the label L_1 to the label L_2

where the property $L_1 \sqsubset L_2 \sqcup L_A$, where L_A is the label containing the policy $\{p: \ \}$ for every principal p in the process' authority set. The declassification process can then be formally defined as an inference rule as described in Definition 2.1.5 [19].

Definition 2.1.5: Relabeling by declassification

$$\frac{L_A = \bigsqcup(p \text{ in current authority}) \ \{p: \ \} \qquad L_1 \sqsubset L_2 \sqcup L_A}{L_1 \text{ may be declassified to } L_2}$$

As the declassification inference rule makes use of relabeling by restriction, the rule for relabeling L_1 to L_2 defines that $\forall J \in L_1$ where J is policies there must be a corresponding policy K in L_2 that is at least as restrictive. For declassification this is achieved by having a corresponding policy in L_A for each principal in the process' authority set that is at least as restrictive, for the policies that are not owned by principals in the process' authority set a corresponding policy must be found in L_2 as the process cannot weaken these policies [19].

The DLM specifies input and output *channels* for obtaining information from outside systems or releasing information to outside systems. Input and output channels are labeled with labels as any other variable in a DLM system. The information received on the input channel would then be labeled with the same label associated with the input channel. Information sent to output channels must be labeled with labels that are at least as restrictive as the label for the output channel [16].

The security policies mentioned so far have all been *confidentiality* policies that only considers whom that can observe the data. However, the DLM can also specify *integrity* policies which considers whom that are able to modify the data protected by the policies. The integrity policies are quality guarantees provided by the owners of the policies that only the specified writers have modified the data. The syntactical notation would be the same as for confidentiality policies but instead of a reader set, a writer set would be associated with the policies. An integrity policy can also be relabeled in the same way that a confidentiality policy may be relabeled [16]. However, the relabeling rules for integrity labels are the inverse of those for confidentiality labels [19]:

- **A writer may be added to a policy**: Adding a writer to a policy is safe because additional writers serve as further warning of contamination.
- **Remove a policy**: A security policy can be safely removed from an integrity label as an integrity policy serves as an assurance that at most the principals

in the policy have affected the data. The removal of the policy restricts the principals that are allowed to modify the data.

- **Replace a writer**: An existing writer can safely be replaced in a security policy's writer set if that writer can be act-for a new writer. Formally a writer w' can be replaced by another writer w if $w' \succeq w$. This would in terms add more writers to the writer set as a policy that permits w' to write might not permit w to write.

- **A policy may be added if it is identical to an existing policy**: A policy I may be added to a label if that policy is identical to an existing policy J if $o(J) \succeq o(I)$.

- **Writers that acts for the owner of a policy may be removed**: As the most restrictive integrity policy is the one where only the owner is present, it is safe to remove writers that can act-for that owner.

In the DLM a writer policy is defined in a similar manner as a reader policy, for example the policy $\{o_1 : r_1, r_2\}$ is a reader policy and the policy $\{o_1 : w_1, w_2\}$ is a writer policy. However, if a security label contains both a confidentiality and an integrity policy, we will place an exclamation mark in front of the principals in the integrity policies for example $\{o_1 : r_1, r_2;\ !o_1, !r_1, !r_3\}$, where the owner o_1 permits r_1 and r_2 to read and r_1 and r_3 to write. A security policy can also be defined with an implicit owner as is the case in the following policy $\{o_1 : r_1, !r_2\}$, where o_1 may read and write to the data but r_1 may only read and r_2 may only write.

2.2 Timed Automata

In computer science finite-state machines [18] are well known theoretical computation models used for describing logical program behavior based on transitions and states that a program can be in. An extension of the finite-state machine with time is called a timed automaton [1], which adds a finite set of real-valued clocks to the finite-state machine. The clocks are increased at the same rate and can be reset during a transition if need be. The primary use of the clocks are to set up guards that can prevent transitions from being executed or prevent the program from being in a certain state [1].

A timed automaton is a tuple $T = (\Sigma, S, S_0, C, E)$ consisting of the following components [1]:

- Σ is the input alphabet accepted by the automaton.

- S is the set of possible finite states that the automaton can be in.

- S_0 is the set of start states which is a subset of S; $S_0 \subseteq S$
- C is a finite set of clocks
- E is the set of possible transitions in the automaton, formally defined as $E \subseteq S \times S \times [\Sigma \cup \{\epsilon\}] \times 2^C \times \Phi(C)$

[1] An edge in the timed automaton would then be defined as $\langle s, s', \sigma, \lambda, \delta \rangle$, which represents a transition from a program state s to another program state s' on the input σ. λ would then be the set of clocks that should be reset with the transition, and δ is the enabling condition (the guard). The automaton starts in one of the start states with all clocks set to zero. The clocks then increases to reflect time elapsed and a transition may be taken when the guards of an edge is satisfied.

An example of a system that is ideally modeled with a timed automaton is a simple smart meter system consisting of a smart meter and an electrical company. The electrical company should be restricted to only being able to read the smart meter data at certain time intervals. In Figure 2.2 this scenario is presented with a timed automaton constructed in UPPAAL [21]. UPPAAL is used to verify timed automata models, such as the model presented in Figure 2.2. The automaton consists of two locations *smd* and *ec*, where *smd* is the smart meter data and *ec* is the electrical company. The system can take the transition from *ec* to *ec* non-deterministically while waiting for the clock x to be larger than 90. When the clock x is larger than or equal to 90 the transition guarded by the expression $x >= 90$ can be taken thus modeling a read of smart meter data by going to the location *smd*, where the only transition that can be taken resets the clock variable to zero and leads back to *ec*. This simple example models an electrical company that can read smart meter data of a single customer every 90 days.

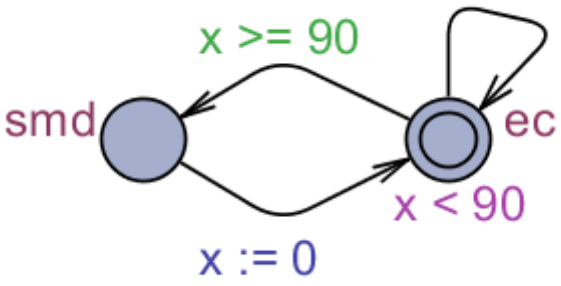

Figure 2.2: Simple smart meter example where an electrical company reads smart meter data every 90 days.

CHAPTER 3

THE TIMED DECENTRALIZED LABEL MODEL

Based on the DLM and timed automata, an extension to the DLM in regards to time is presented along with scenarios that describes the expressiveness of the extension by defining security policies that are dependent on time. In Section 3.1 the Timed Decentralized Label Model (TDLM) is presented. In Section 3.1.1 the new constructs added to the TDLM along with a syntax in Extended Backus-Naur Form are presented and explained. The TDLM is formally defined in Section 3.1.1 and in Section 3.1.3 the constructs are used in concrete examples to clarify their meaning and expressiveness. In Section 3.1.4 an explanation of how time can be trusted in the TDLM is presented. In Section 3.1.5 the principal hierarchy known from the DLM is further extended with the use of constructs from the TDLM. The complete safe relabeling rule from the DLM is modified to cohere to the new constructs added in the TDLM in Section 3.2. In Section 3.3 several scenarios based on timed automata are formulated to describe the expressiveness of the TDLM.

3.1 DLM Extension

In [22] we proposed an extension to the DLM that takes time into consideration when defining security policies. In this section we will clarify how the extension could look syntactically and how it is supposed to work.

The main idea is to extend the DLM with clock variables to represent incrementable time where the system is responsible for incrementing clock variables with the authority of the clock owners. Incrementable clock variables are integers representing arbitrary values defined by the owner of the clock variable and may be increased according to the rules the owner of the clock variable has defined. In addition to this, the value of clock variables would have to be normalized for the each

system that makes use of the TDLM to make clock comparisons unified between all policies in a system. The clock variables could for example be normalized to strictly model nanoseconds which would provide a more general way of depicting real-time even though clock variables does not reflect real-time.

As in the DLM it would be the system that is responsible for enforcing that clock variables are correctly incremented and access to data labeled with time-based security policies are restricted in the proper manner. The principal hierarchy and the acts-for relationship defined by the DLM would still behave in the same manner even though the model is extended with time, however in Section 3.1.5 we propose an extension to the principal hierarchy that can be used to restrict acts-for relationships.

The DLM does not clearly define how the principal hierarchy works at run-time or how the acts-for relationships are enforced or the rules for changing these relationships. As such, we make the same assumption that is made in regards to the DLM that the principal hierarchy may change over time and that acts-for relationships are correctly enforced when inspecting a security policy.

The DLM extension is henceforth referred to as the TDLM and extends the regular DLM with time, which is done by adding clock variables that can be used to restrict policies based on time. Clock variables are defined by the principal that creates the security policy or a principal that can act-for it, and can be used to restrict an entire owner's reader set or specific readers.

3.1.1 TDLM Constructs

To clarify the new constructs that the TDLM adds to the DLM, an explanation of the isolated constructs are presented to give an unambiguous understanding of what each individual construct means before a detailed and exemplified explanation is given. The semantics for the TDLM constructs are as follows:

- **Declaring clock variables**: Clock variables are used to restrict access to data based on time and are declared within a set of parentheses () which is placed after a principal (owner or reader/writer) to restrict that principal's access, after another clock variable to indicate who that can see and alter the clock value, and in the principal hierarchy to restrict acts-for relationships. If a clock is placed on an owner of a policy then all readers or writers associated with this owner is restricted by the clock, however if a clock is placed on a reader or writer then only that specific reader or writer is restricted by it. A clock variable is identified by its unique name which can be any combination of alphabetic characters and the value of a clock variable can be any positive integer.

- **Comparing clock values**: Clock variables can be compared to other clock variables and constant integers with the use of the binary comparison operators

defined in Table 3.1. If that comparison evaluates to true then the clock variables allows access to the entity it is associated with. The comparison is defined within the parenthesis along with the declaration of the clock variable for example ($x > 10$). The value the clock variable is compared to is called the comparison value.

- **Multiple comparisons**: Multiple clock comparisons can be performed within the same parenthesis by separating them with the logical && and || operators, which evaluates as expected for example ($x > 10$ && $x < 15$) would evaluate to true when x is between ten and 15. A statement of clock comparisons placed within a single set of parenthesis is called a clock expression and must evaluate to true before the associated principal(s) can gain access to the data.

- **Parameterized clock variables**: Clock variables can be parameterized with the use of square parenthesis [] placed after the name of the clock variable. However, if no parameters should be defined on a clock variable then the square parenthesis may be omitted. Within the square parenthesis three optional parameters can be declared. Parameters are separated with a semicolon ; and are identified in the following order: Upper Limit ; Event ; Reset Value. If only one parameter is declared and that parameter's identifier starts with a question mark then it is an event otherwise it would be an upper limit. A parameterized clock variable would then be defined as ($x[15; ?event; 1] > 10$).

- **Upper limit**: An upper limit is a constant integer used to define when a clock variable should be reset. Upon reaching the upper limit the value of the clock variable will instantly get reset.

- **Reset value**: The reset value is a constant integer used to define the value of a clock variable when it resets. If omitted from a clock variable the reset value is always zero.

- **Events**: Events are defined by a unique alphabetic name starting with a question mark ? and can be placed as a parameter on a clock variable, which indicates that when the event is triggered then the value of the clock variable is reset.

- **Event trigger**: An event trigger can be specified on any principal with the use of square parenthesis [] placed immediately after a principal's identifier. An event trigger starts with the symbol $*$ and should have the exact same name as the event that should be triggered when said principal successfully reads or writes to some data. Several event triggers can be placed on the same principal in the same security policy by separating the event triggers with a comma, for example $p[*event1, *event2]$ would trigger $event1$ and $event2$ when

p reads the data. An event can only be triggered from within the system by those principals that have an event trigger defined on them in a given security policy.

$<$	Less than
$>$	Greater than
$==$	Equal to
$<=$	Less than or equal to
$>=$	Greater than or equal to
$!=$	Not equal

Table 3.1: Operators that are allowed when comparing clock variables.

To give an overview of the TDLM syntax consider Definitions 3.1.1, 3.1.2, and 3.1.3 where a syntax in Extended Backus-Naur Form (EBNF) [12] is presented.

Definition 3.1.1: TDLM Syntax in EBNF - General

$$
\begin{aligned}
< \text{digit} > &= \text{"0"} \mid \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \text{"4"} \mid \text{"5"} \mid \text{"6"} \\
&\quad \mid \text{"7"} \mid \text{"8"} \mid \text{"9"} \ ; \\
< \text{letter} > &= \text{"}a\text{"} \mid \text{"}b\text{"} \mid \text{"}c\text{"} \mid \text{"}d\text{"} \mid \text{"}e\text{"} \mid \text{"}f\text{"} \mid \text{"}g\text{"} \\
&\quad \mid \text{"}h\text{"} \mid \text{"}i\text{"} \mid \text{"}j\text{"} \mid \text{"}k\text{"} \mid \text{"}l\text{"} \mid \text{"}m\text{"} \mid \text{"}n\text{"} \\
&\quad \mid \text{"}o\text{"} \mid \text{"}p\text{"} \mid \text{"}q\text{"} \mid \text{"}r\text{"} \mid \text{"}s\text{"} \mid \text{"}t\text{"} \mid \text{"}u\text{"} \\
&\quad \mid \text{"}v\text{"} \mid \text{"}w\text{"} \mid \text{"}x\text{"} \mid \text{"}y\text{"} \mid \text{"}z\text{"} \mid \text{"}A\text{"} \mid \text{"}B\text{"} \\
&\quad \mid \text{"}C\text{"} \mid \text{"}D\text{"} \mid \text{"}E\text{"} \mid \text{"}F\text{"} \mid \text{"}G\text{"} \mid \text{"}H\text{"} \mid \text{"}I\text{"} \\
&\quad \mid \text{"}J\text{"} \mid \text{"}K\text{"} \mid \text{"}L\text{"} \mid \text{"}M\text{"} \mid \text{"}N\text{"} \mid \text{"}O\text{"} \mid \text{"}P\text{"} \\
&\quad \mid \text{"}Q\text{"} \mid \text{"}R\text{"} \mid \text{"}S\text{"} \mid \text{"}T\text{"} \mid \text{"}U\text{"} \mid \text{"}V\text{"} \mid \text{"}W\text{"} \\
&\quad \mid \text{"}X\text{"} \mid \text{"}Y\text{"} \mid \text{"}Z\text{"} \ ; \\
< \text{string} > &= \ < \text{letter} > \ , \ \{< \text{letter} >\} \ ; \\
< \text{operators} > &= \text{" < "} \mid \text{" > "} \mid \text{" == "} \mid \text{" <= "} \mid \text{" >= "} \mid \text{"!="} \ ; \\
< \text{integer} > &= \ < \text{digit} > \ , \ \{< \text{digit} >\} \ ; \\
< \text{logical-operators} > &= \text{"\&\&"} \mid \text{"||"} \ ;
\end{aligned}
$$

Definition 3.1.2: TDLM Syntax in EBNF - Clocks

$$
\begin{aligned}
&< \text{upper-lim} > = \ < \text{integer} > \ ; \\
&< \text{reset} > = \ < \text{integer} > \ ; \\
&< \text{event} > = \text{"?"} \ , \ < \text{string} > \ | \ \text{"?"} \ , \ < \text{string} >^{<\text{label}>}; \\
&< \text{clock-par} > = \text{"["} < \text{upper-lim} > \text{"]"} \ | \ \text{"["} < \text{event} > \text{"]"} \ | \\
&\qquad \text{"["} < \text{upper-lim} > \ , \ \text{";"}, < \text{reset} > \text{"]"} \ | \\
&\qquad \text{"["} < \text{event} > \ , \ \text{";"} \ , \ < \text{reset} > \text{"]"} \ | \\
&\qquad \text{"["} < \text{upper-lim} > \ , \ \text{";"} \ , \ < \text{event} > \ , \ \text{";"} \ , \ < \text{reset} > \text{"]"} \ | \\
&\qquad \text{"["} < \text{upper-lim} > \ , \ \text{";"} \ , \ < \text{event} > \text{"]"} \ ; \\
&< \text{clock} > = \ < \text{string} > \ , \ [^{<\text{label}>}] \ | \\
&\qquad < \text{string} > \ , \ < \text{clock-par} > \ , \ [^{<\text{label}>}] \ ; \\
&< \text{clock-com} > = \ < \text{clock} > \ , \ < \text{operators} > \ , \ (< \text{integer} > \ | \ < \text{clock} >) \ ; \\
&< \text{clock-exp} > = \text{"("} \ , \ < \text{clock-com} > \ , \ \text{")"} \ | \\
&\qquad \text{"("} \ , \ < \text{clock-com} > \ , \ \{< \text{logical-operators} > \ , \\
&\qquad < \text{clock-com} >\} \ , \ \text{")"} \ ; \\
&< \text{event-trig} > = \text{"}*\text{"} \ , \ < \text{string} > \ \{\text{","} \ , \ \text{"}*\text{"} \ , \ < \textit{string} >\} \ ;
\end{aligned}
$$

Definition 3.1.3: TDLM Syntax in EBNF - Label

$$
\begin{aligned}
&< \text{principal-par} > = \ [< \text{clock-exp} >] \ , \ [< \text{event-trig} >] \ ; \\
&< \text{principal} > = \ < \text{string} > \ , \ < \text{principal-par} > \ | \\
&\qquad \text{"!"} \ , \ < \text{string} > \ , \ < \text{principal-par} > \ | \\
&\qquad \text{"!!"} \ , \ < \text{string} > \ , \ < \text{principal-par} > \ ; \\
&< \text{policy} > = \ < \text{principal} > \ , \ \text{" : "} \ | \\
&\qquad < \text{principal} > \ , \ \text{" : "} \ , \ < \text{principal} > \ , \\
&\qquad \{\text{","} \ , \ < \text{principal} >\} \ ; \\
&< \text{label} > = \text{"\{"} \ , \ \text{"\}"} \ | \\
&\qquad \text{"\{"} \ , \ < \text{policy} > \ , \ \{\text{";"} \ , \ < \text{policy} >\} \ , \ \text{"\}"} \ ;
\end{aligned}
$$

In regards to parameterized clock variables, an upper limit was introduced to explicitly define when a clock should reset to its original value and thereby create a recurring time period where principals can observe the data protected by the security policy containing a clock variable rather than just being able to specify a single nonrecurring time period. Reset values were introduced to provide flexibility in regards to further specifying the time period where principals can observe data and to allow systems that require clock variables to not be zero. Events were introduced to provide a mechanism for specifying that principals have a single one time access to data before the clock is reset and thus limit how many times principals can read or write to data in the time period where the policy allows so.

Formal Definition

Formally a clock Υ can be defined as described in Definition 3.1.4, where c is the clock variable, α is the upper limit, β is the reset event, and γ is the reset value. As mentioned above, clock expressions Φ can be placed on any principal in a security policy and is formally defined as described in Definition 3.1.5, where Υ is either a clock or a constant signed integer, υ is a comparison symbol from Table 3.1, τ is the logical && or ||, and the last Φ represents another clock expression or nothing if τ is omitted.

Definition 3.1.4: TDLM Clock

$$\Upsilon = c[\ \alpha;\ ?\beta^{\Lambda};\ \gamma\ ]$$

Definition 3.1.5: TDLM Clock Expression

$$\Phi = (\Upsilon \upsilon \Upsilon)\ \tau\ \Phi$$

A TDLM security policy Γ can formally be defined as described in Definition 3.1.6, where (Φ) is an optional clock expression that can be placed on any principal in the policy, $[*\beta]$ is an optional event trigger, and rw are readers or writers of the data owned by the owner o. A security label Λ in the TDLM consists of one or more security policies and can be formally defined as described in Definition 3.1.7, where Γ is a security policy and m describes the total number of policies present in the label.

Definition 3.1.6: TDLM Security Policy

$$\Gamma = o(\Phi)[*\beta] : rw_1(\Phi)[*\beta], \ldots, rw_n(\Phi)[*\beta]$$

Definition 3.1.7: TDLM Security Label

$$\Lambda = \{\Gamma_1; \ldots; \Gamma_m\}$$

3.1.2 From Policies to Timed Automata

Security policies can be constructed based on the TDLM constructs described in Section 3.1.1 to express which principals that can access protected data and when they can access this data. As an example consider the policy in Equation 3.1 where all constructs in the TDLM is used to express a policy where o and r may access data protected by this policy when the clock variable x is larger than ten and y is larger than 15, x is reset to five when reaching the value 20 or when r successfully reads the data protected by the policy.

$$\{o\ (x[20; ?reset; 5] > 10 \&\& y > 15) : r[*reset]\} \tag{3.1}$$

The behavior of security policies can be expressed via one or more timed automata where each principal present in the security policy have their own automaton describing their access restrictions, however if several principals are allowed to access the data under the same conditions then a single timed automaton might describe the access restrictions for multiple principals. A timed automaton that describes the access restriction for a principal has its start location labeled with the name of the principal that the automaton describes as depicted in Figure 3.1, where rw_i is the principal that the timed automaton depicts the access possibility of. The location named *data* is used to describe that when the timed automaton is in this location then the principal is allowed to observe the data but as the location is committed (marked with a C) then an edge going away from this location must be taken immediately to enforce the access restriction. The access restriction must be placed on an ingoing edge to the *data* location thus modeling that the data is protected by some time constraints Φ. If a policy contains events β that should be triggered by certain principals then the timed automaton that describes the access behavior of these principals must trigger the event on the edge going away from the *data* location thus modeling reset only on successful read/write.

In addition to this, for each clock variable in a security policy a timed automaton describes how this clock variable is incremented and who that can increment it. A timed automaton that describes a clock variable consists of one location with up to three cyclic edges that describe incrementing the value of the clock c, resetting the clock value upon reaching an upper limit α, and resetting the clock value when an event β is triggered. The clock value is reset to the reset value γ when an event is triggered or an upper limit is reached. When an upper limit is present on a clock

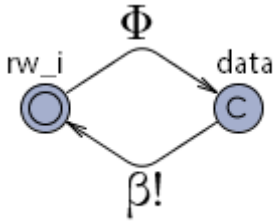

Figure 3.1: General timed automaton that describes the access restrictions for the principal rw_i restricted by the clock expression Φ and triggering the event β on successful read/writes.

variable then a guard on the incrementation edge must be placed to force a reset when reaching the upper limit. Figure 3.2 depicts a timed automaton that describes the behavior of the clock c which can be incremented by the principal *co*.

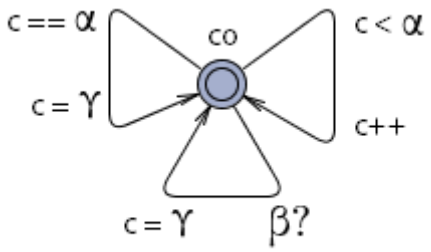

Figure 3.2: General timed automaton that describes a clock c that can be incremented by the principal *co*.

As an example consider the timed automata in Figure 3.3 which describes the policy in Equation 3.1. The first timed automaton labeled 1. describes the access restrictions for the owner o which is restricted by the guard placed on the edge from the initial location to the *data* location meaning that the owner o may only observe the data when the conditions for x and y are met. When o has successfully gained access to the data (the timed automaton is in the location named *data*) the timed automaton immediately goes to the initial location such that the restrictions placed on the data can be enforced correctly.

Since o and r are restricted by the same clocks they should be modeled by the same timed automata, however r triggers an event when successfully reading the data and thus the access behavior for r must be modeled by a different timed automaton. The timed automaton labeled 2. models the access behavior of r and is equivalent to the timed automaton modeling o except for the event ($reset!$) that is triggered

when r has successfully gained access to the data. Events are expressed by channel synchronization meaning that when a timed automaton takes an edge marked with an event trigger such as $reset!$ then all the corresponding timed automata in the system take any available matching edges marked with the same event name such as $reset?$.

The timed automaton labeled 3. models the behavior of the clock variable x, which can be incremented by the principal o (indicated by the name of the start location) when the value of x is below 20 but upon reaching the value 20, o must reset the clock to five before the clock can be incremented again thus modeling the upper limit and reset value of x described by the policy. The edge marked with $reset?$ models the event that is triggered by r and resets the clock to its reset value when the event is triggered.

The timed automaton labeled 4. models the behavior of the clock variable y which also can be incremented by the principal o but this timed automaton contains only one edge as there is no upper limit, reset value or event associated with y.

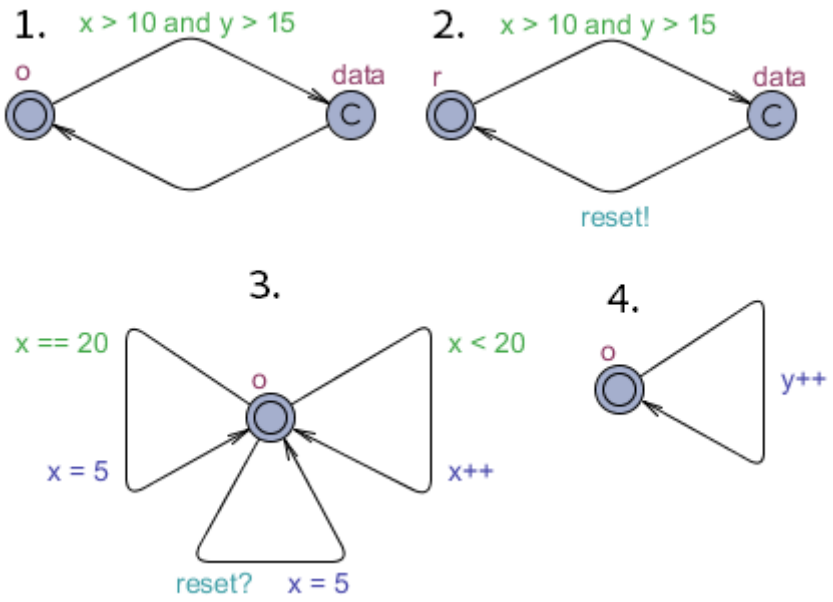

Figure 3.3: Timed automata describing the security policy $\{o(x[20; ?reset; 5] > 10 \&\& y > 15) : r[*reset]\}$

3.1.3 Clarifying TDLM Constructs

As mention earlier the TDLM extends the DLM with clock variables used to restrict security policies for example the extended label in Equation 3.2 specifies that the

entire reader set for o_1 only can access the data when the clock variable x is above two and that the specific reader r_5 only can access the data when the clock variable y is above five. The other readers r_3, r_4 have unrestricted access to the data. If another principal p_1 acts-for r_5 then p_1 would be restricted under the same condition that r_5 is restricted. The behavior of this security label is modeled with three timed automata as depicted in Figure 3.4. Since this security label is composed of multiple security policies a timed automaton cannot be constructed separately for each principal as a principal should be able to act-for another principal in each security policy in the security label before he can access the data. To model this a single timed automaton is constructed which models all the different cases, so to be in a location labeled with the names of principals then the principal must be able to act-for one of the principals in the label name. This means that for a principal to access the data in this case then he must be able to act-for one of the principals in the each location towards the *data* location.

$$\{o_1(x > 2) : r_1, r_2;\ o_2 : r_3, r_4, r_5(y > 5)\} \tag{3.2}$$

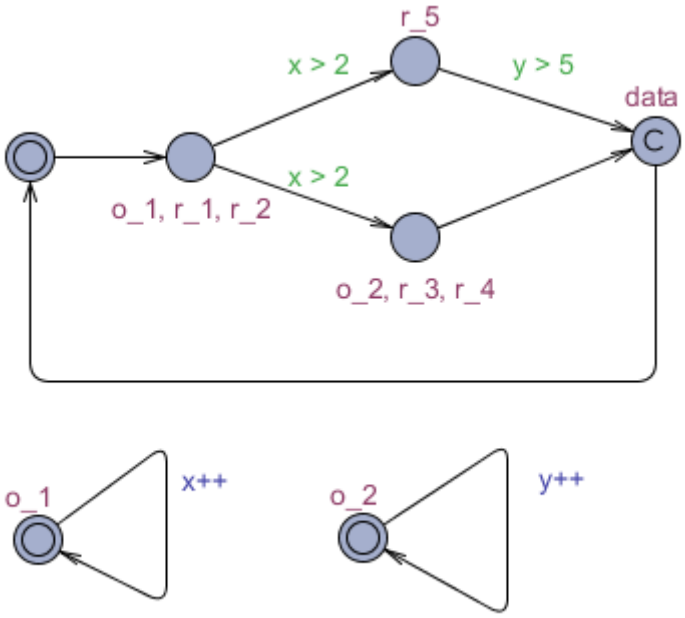

Figure 3.4: The security label in Equation 3.2 modeled as three timed automata.

Clock variables can also be combined to form complex logical expressions that must be true for the restricted principals to access the data, for example the label in Equation 3.3 expresses that before r_6, r_7 can access the data then the logical clock expression must be true, which is modeled by the timed automata in Figure 3.5.

$$\{o_3(z > 10 \;\&\&\; p < 15 \;||\; k == 12) : r_6, r_7\} \tag{3.3}$$

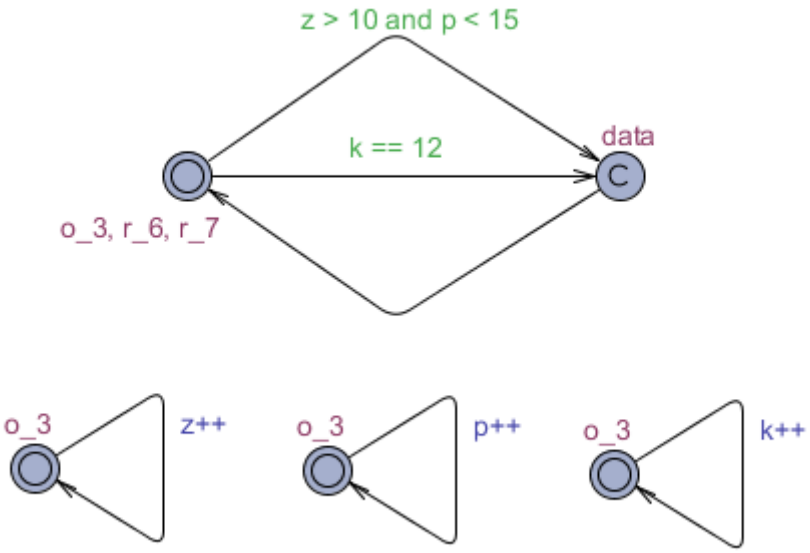

Figure 3.5: The security label in Equation 3.3 modeled as timed automata.

In addition to defining when principals should be able to access the data, it is also possible to specify when a clock variable should be reset. Consider the label in Equation 3.4 where a clock variable is defined to be reset to the value three when it reaches the value ten, meaning that the principal r_8 would be able to access the data when the clock variable q is larger than five and less or equal to ten. The reset value can be omitted which would result in the clock variable being reset to zero. In Figure 3.6 two timed automata models this scenario, where the principals can access the data when the value of q is larger than five and less or equal to ten indicating that the clock resets when reaching the value ten.

$$\{o_4(q[10;3] > 5) : r_8\} \tag{3.4}$$

Security policies involving clock variables can be defined such that the data protected by the policy only can be accessed in an infinitely small amount of time meaning that no principal actually can access the data. For example the policy in Equation 3.5, the clock is reset as soon as it reaches the value where the data can be accessed resulting in no principals being able to gain information about the data in a practical scenario, but theoretically there could be an infinite small amount of time where the principal could observe the information as modeled with the timed automata in Figure 3.7.

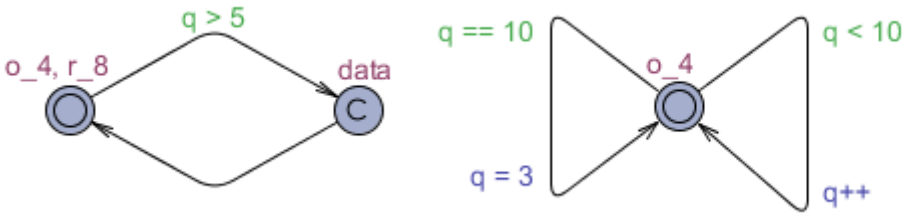

Figure 3.6: The security label in Equation 3.4 modeled as timed automata.

$$\{o_5(x[90] >= 90) : r_9\} \tag{3.5}$$

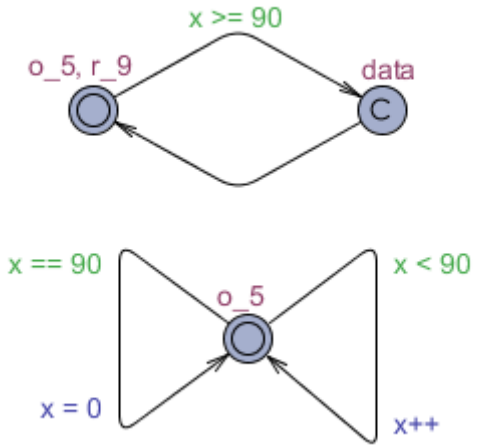

Figure 3.7: The security label in Equation 3.5 modeled as timed automata.

Furthermore, the extension to the DLM has the possibility of defining events that can be used to reset clock variables. An event can be defined as a part of the clock variable's parameters but must be placed in between the upper limit and the reset value such that these can be clearly distinguished as both are optional. In addition to this, the event is defined by placing a question mark in front of an alphabetic string and for the event to be triggered a star symbol is placed in front of a matching string on for example a reader or writer. This means that the specific reader or writer triggers the event when they successfully read or write to the data thus resetting the clock variable. In practice, this means that the particular reader or writer may only access the data once before the clock value is reset. As an example

consider Equation 3.6 where the clock variable y is specified to have an upper limit of 15, a reset value of one, and an event called $?reset$, which means that the clock is reset to one when reaching the value 15 or when the event is triggered. Furthermore, the reader r_7 is specified to trigger the event via the event trigger $*reset$ so when r_7 successfully reads the data then the clock variable is reset. However, if the data is not read successfully the event will not be triggered for example if the reader tries to read when the clock condition is not met. This system behavior is depicted in Figure 3.8.

$$\{o_6(y[15; ?reset; 1] > 10) : r_7[*reset]\} \tag{3.6}$$

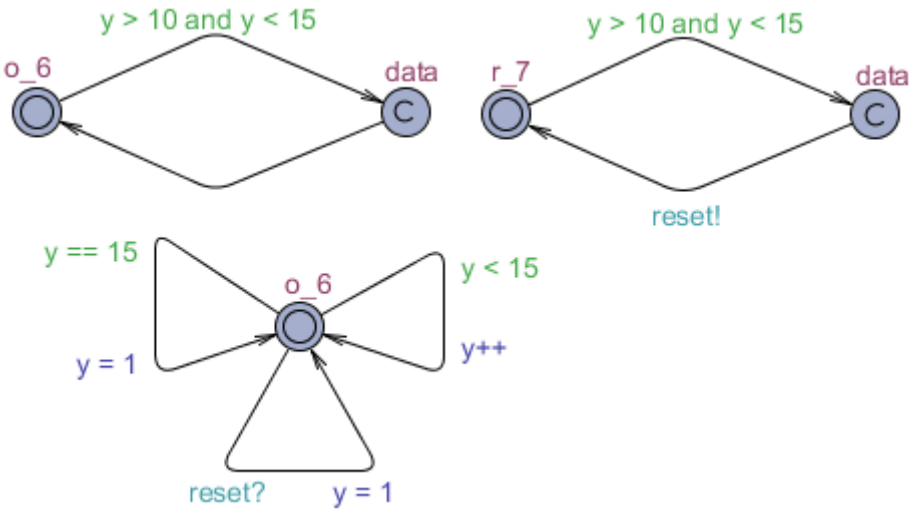

Figure 3.8: The security label in Equation 3.6 modeled with three timed automata.

In addition to this, it is possible to specify that an event trigger is applied to all principals for a given security policy by placing the event trigger on the owner of the policy as depicted in Equation 3.7. Figure 3.9 models this behavior with two timed automata that is responsible for resetting the clock value y to one when the event $reset?$ is triggered by either o_6 or r_7 accessing the data when y is greater than ten.

$$\{o_6(y[?reset; 1] > 10)[*reset] : r_7\} \tag{3.7}$$

Furthermore, events can be triggered from security policies even though they were not defined in these policies, which means that events names are unique in regards to the system they are defined in. This also indicates that the principal who owns the clock variable data gives permission such that the defined event can

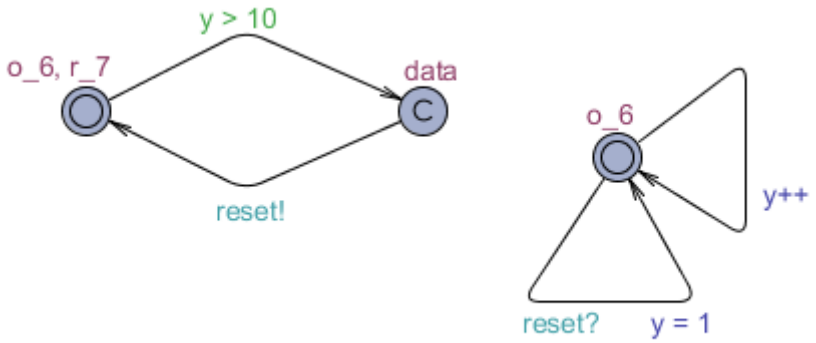

Figure 3.9: The security label in Equation 3.7 modeled with two timed automata.

reset the clock even though it is triggered from a different security policy. However, security policies may also be defined on events to indicate which principals that are allowed to trigger the given event no matter from which policy the event is triggered but if no security label is defined on an event then all principals may trigger it. An example of an event with a security policy where only r_8 is allowed to trigger the event is defined in Equation 3.8. Figure 3.10 depicts this behavior, however the timed automaton that is responsible for resetting the clock value is annotated with the names of the principals that can trigger the event. As such, the event has its own timed automaton instead of being an edge in the clock automaton as usual to model that o_6 and r_8 may trigger the event but it is only o_6 that can increment the clock value of y.

$$\{o_6(y[?reset^{\{!o_6:!r_8\}}; 1] > 10) : r_8[*reset]\} \tag{3.8}$$

In regards to integrity policies, potential writers are defined to have a default write behavior of non-destructive writes (append) and the owner of the integrity policy may perform both appends and destructive writes, which means that any principal who can act-for the owner of an integrity policy may also perform both append and destructive write operations. In addition to this, principals can be defined to be allowed to perform destructive writes by explicitly placing two exclamation marks !! in front of the principal that should be allowed to do so. An example of an integrity policy with a principal that can perform destructive writes is defined in Equation 3.9, where r_8 may perform destructive writes but r_9 may only append.

$$\{!o_7 :!!r_8, !r_9\} \tag{3.9}$$

Clocks in integrity policies ensure that principals that are restricted by the clocks

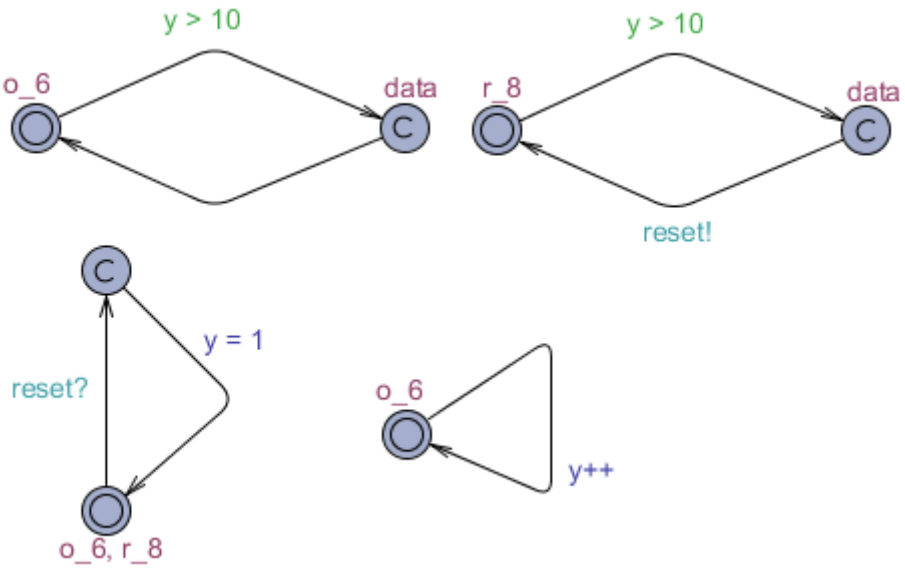

Figure 3.10: The security label in Equation 3.8 modeled with four timed automata.

are only allowed to alter the data protected by the integrity policy at the defined time intervals. Time in integrity policies does not provide the same degree of utility as time in confidentiality policies, however in some cases time in integrity policies might be useful for example to identify in which time period changes happened at run-time such that appropriate measures can be taken. As an example consider the integrity label $\{!o :!w(x[15] > 5)\}$ where the principal w only can alter the data when x is between five and 15 meaning that errors resulting from w's write operations only can occur in the specified time interval.

As in the DLM relabeling is done through rewrite sequences that transforms a label L_1 to L_2. Relabeling should still be a restriction to ensure that it is safe, meaning that the relationship $L_1 \sqsubseteq L_2$ should hold. Clock expressions would then have to be at least as restrictive in L_2, meaning that new clock variables could be added or the existing variable could be made more restrictive. In addition to this, declassification should still work as described in the DLM where principals or processes can declassify data if and only if they have the authority of the data owners. Declassification would then remove or lessen the restrictions of clock expressions.

3.1.4 Trusting Clocks

The system with the authority of the clock variable creator is responsible for making sure that incrementable clocks are increased according to the security policies defined by the owner of the clock. The clock variables are considered untrustworthy as they

are defined exclusively by owners of security policies and principals who can act-for the owner meaning that they can only be trusted in the policy where they are defined. Furthermore, clocks can only be incremented by the principal that defined them or the system acting for the defining principal. Implicitly, this means that a clock c defined by the principal p would have the security policy $\{!p : \ \}$. This is the default behavior of the system if no labels are defined on the clock variables.

In addition to relying on the default behavior, security policies (both confidentiality and integrity) can also be defined on clock variables to describe who can observe the value of the clock and who can modify the clock value. As such it can be defined how trustworthy a clock is by explicitly defining who have influence in regards to the clock value, making it possible for principals that make use of the specific clock to determine if they trust the value of the clock. An example of a security label that contains a clock with a security label could be the label defined in Equation 3.10, where o_1 is the owner of the data security label and the clock security label, r_1 may read the data and the value of the clock x, however r_2 cannot observe the value of the clock.

Even though r_2 cannot observe the actual value of the clock he may be able to infer the value by observing a pattern in regards to how the data protected by the security policy is handled, for example if the data is read in the same interval each time then r_2 can infer when the clock is reset and what value the clock must have before the data can be observed. However, the ability to infer clock values is an acceptable consequence of trying to make clock variables confidential because the trust of clock variables are more dependent on the integrity of the clock value than the confidentiality. If a principal can infer the value of the clock but not modify it, the clock value can still be trusted as it is only the specified principals that may update the value. However, if a system requires that clock variables are confidential and cannot be inferred then some other methods must be used to ensure this property holds, but this is out of scope and will not be further investigated.

$$\{o_1(x[10]^{\{o_1:r_1\}} > 5) : r_1, r_2\} \tag{3.10}$$

Furthermore, the security policies that can be defined on clocks can also contain other clocks such that it is possible to express that a clock is only trustworthy in certain time periods. This form of security labeling clocks forms a recursive evaluation process as the clocks used to describe security policies can also contain clocks and so on. As an example consider the security label in Equation 3.11, where the addition of the clock y in the clock x's security label further restricts the possibility of observing the clock value of x.

$$\{o_1(x[10]^{\{o_1(y[20]>10):r_1\}} > 5) : r_1, r_2\} \tag{3.11}$$

Theoretically, security labeling clocks makes the TDLM more expressive and

secure as it can explicitly be defined if a clock is trustworthy in a certain context or not. However, practically this would be challenging to implement and enforce due to the recursive behavior of clock labeling.

Formally labeling clocks with a security label requires a change in the formal definition of a clock Υ which is redefined as described in Definition 3.1.8, where a security label Λ now can be placed on a clock.

Definition 3.1.8: TDLM Clock

$$\Upsilon = c[\ \alpha;\ ?\beta;\ \gamma\ ]^{\Lambda}$$

3.1.5 Clocks in the Principal Hierarchy

To further increase the flexibility of the TDLM, clock variables can also be placed directly on the acts-for relationships in the principal hierarchy to express that a principal has a limited acts-for relationship to another principal. Specifically, this means that a principal can be defined to act-for another principal in a given time period, which may be relevant to model certain systems. For example if a principal p_1 can act-for another principal p_2 but the principal p_2 only trusts p_1 when a certain amount of time has passed for example if p_2 is responsible for performing certain critical system updates and does not want any other principals to be able to interfere whilst the update is being performed. In this case a clock variable is placed on the acts-for relationship as depicted in Figure 3.11, where p_1 may act-for p_2 when the clock variable x is larger than ten and p_2 is the only principal that may update the value of the clock x.

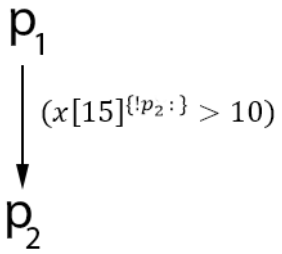

Figure 3.11: Principal hierarchy with a time restricted acts-for relationship.

Another example is when several principals are involved in acts-for relationships as depicted in Figure 3.12, where a principal p_1 may act-for another principal p_2 which can act-for a third principal p_3 and a forth principal p_4. A time restriction is placed on the acts-for relationship between p_1 and p_2, and between p_2 and p_4. This

means that for p_1 to be capable of acting for p_4 then both clock variables x and y must allow so, and for p_1 to act-for p_3 the clock variable x must be satisfied. The event that resets the clock variable y can be triggered from other security policies.

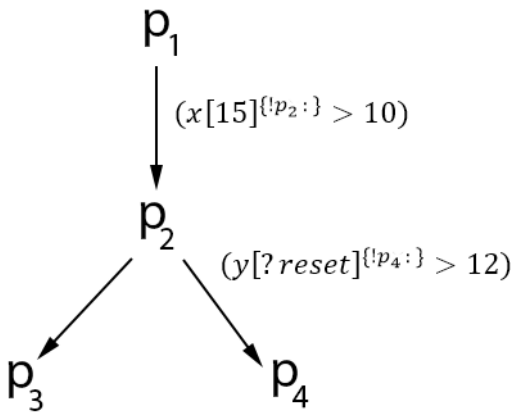

Figure 3.12: Principal hierarchy with several time restricted acts-for relationships that propagates.

The expressiveness of clocks in the principal hierarchy is equivalent to the clocks in security policies meaning that security policies can be placed on the clock variables and events can be expressed as well.

3.2 Safe Relabeling in the TDLM

As the TDLM extends the DLM, the possibility of relabeling data can be performed either by restriction or declassification. Restriction is said to be a safe relabeling as it restricts the principals that can observe or alter the data protected by the security policy.

3.2.1 Confidentiality Policies

As with the DLM safe relabeling for confidentiality labels can be done for the TDLM in the following manner:

- **Remove readers**: As is the case for the DLM, readers can be freely remove even though it has an associated clock variable because this would still make the label at least as restrictive.

- **Add policy**: Adding a security policy to a label would also be a safe relabeling as it would only make the label more restrictive.

- **Add a reader**: A reader r can be added to a security policy's reader set if that reader can act-for a reader r' that is already present in the reader set such that $r \succeq r'$. However, in the TDLM clock variable that are present on r' would also have to be associated with r.

- **Replace an owner**: As in the DLM, an owner o may be replaced by another owner o' if $o' \succeq o$, which would restrict the policy by only allowing processes to declassify the data with the authority of o' instead of o. In the TDLM the clock variables associated with o should be associated with o' as well.

- **Add clock variables**: A clock variable that is not already associated with a principal p can be added to that principal safely as it would cause the security policy to be at least as restrictive as before.

- **Access time**: The comparison value in a clock expression may be altered safely if it reduces the window of opportunity for observing the data. Specifically, this means that if a clock variable x with an upper limit of 15 allows access for a principal when the value is above ten then increasing the comparison value to 12 would reduce the window of opportunity. Formally, the policy $\{o : p(x[15] > 10)\}$ can be safely relabeled to $\{o : p(x[15] > 12)\}$ as this would make the label more restrictive because the principal p has less time access the data. Another example is the policy $\{o : p(x[15] < 10)\}$ which can be safely relabeled to $\{o : p(x[15] < 5)\}$ as this relabeling reduces the time p can access the data from nine time units to four time units.

The DLM defines a safe relabeling to be a rewrite sequence that via the safe relabeling rules transforms a label L_1 to the label L_2 where L_1 is at most as restrictive as L_2 and L_2 is at least as restrictive as L_1, written formally as $L_1 \sqsubseteq L_2$. In the TDLM this definition should still be upheld when performing safe relabeling for confidentiality labels. To describe the formal definition for safe relabeling in the TDLM we still need the function $R(I)$ presented in Definition 2.1.1 which yields the implicitly allowed readers. However, in the TDLM clocks can also be placed on acts-for relationships which would result in a different implicit reader set based on the current values of clocks. A function which yields clocks associated with a principal for a policy I is needed to define the safe relabeling rule for the TDLM. Definition 3.2.1 describes this function where p is a principal in the policy I, $c(I)$ is the set of clocks present in the policy I, and c is the clock associated with p. $C(p)$ would then be the set of clocks associated with the principal p.

Definition 3.2.1: Safe Relabeling: Clock Function

$$C(p) = \{c \mid \forall_{c(I) \in p}\ c\}$$

Furthermore, a definition for the concept of a clock restriction is presented in Definition 3.2.3 which is dependent on the function $T(c)$ defined in Definition 3.2.2 which yields the available time units t that a principal restricted by the clock c can access some data protected by the clock. The clock restriction states that for a clock to be at least as restrictive then the window of opportunity (time units where clock allows access) must be equal to or less than the original clock.

Definition 3.2.2: Safe Relabeling: Time Function

$$T(c) = \{t \mid \text{time units } t \text{ where clock } c \text{ allows access}\}$$

Definition 3.2.3: Safe Relabeling: Clock Restriction

$$C(p) \sqsubseteq C(p') = \forall_{c' \in C(p')} \exists_{c \in C(p)} T(c') <= T(c)$$

The safe relabeling rule for the TDLM extends the safe relabeling rule for the DLM by adding two new rules regarding clock variables and how these may be safe relabeled. The complete safe relabeling rule for the TDLM is defined in Definition 3.2.4, where the first three statements are equal to the ones presented for the DLM but two additional statements are added. The first states that for all clocks associated with a principal p in the reader set for the policy I there exist clocks associated with the principal p' in the policy J such that the clocks associated with p' is equivalent to those of p but with the possibility of additional clocks not already associated with p' being added. The second statement says that the set of clocks associated with the principal p' in J must be at least as restrictive as those associated with p in the policy I, meaning that the window of opportunity for observing the data in $C(p')$ should be less than or equal to the ones present in $C(p)$.

Definition 3.2.4: The Complete Relabeling Rule for the TDLM

$$\begin{aligned}
L_1 \sqsubseteq L_2 &\equiv \forall_{I \in L_1} \exists_{J \in L_2}\ I \sqsubseteq J \\
I \sqsubseteq J &\equiv o(J) \succeq o(I) \wedge R(J) \subseteq R(I) \\
&\equiv o(J) \succeq o(I) \wedge \forall_{p' \in r(J)} \exists_{p \in r(I)}\ p' \succeq p \\
&\equiv o(J) \succeq o(I) \wedge \forall_{p \in r(I)} \exists_{p' \in r(J)}\ C(p) \subseteq C(p') \\
&\equiv o(J) \succeq o(I) \wedge \forall_{p \in r(I)} \exists_{p' \in r(J)}\ C(p) \sqsubseteq C(p')
\end{aligned}$$

As an example of all the safe relabeling rules consider the security policy $\{o_1(x[15] > 10) : r_1, r_2\}$ with the principal hierarchy depicted in Figure 3.13 which corresponds to the timed automata in Figure 3.14.

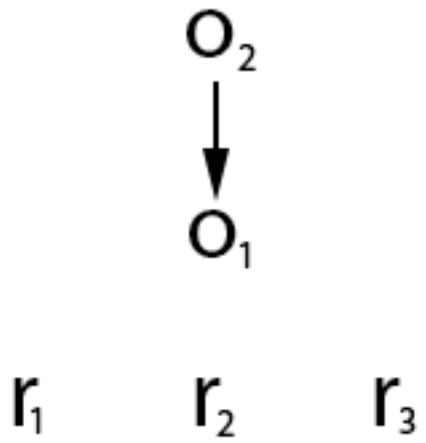

Figure 3.13: Principal hierarchy for safe relabeling rules example.

The first relabeling rule for confidentiality is that a reader may be removed such as removing r_2 which would result in the policy $\{o_1(x[15] > 10) : r_1\}$ and the timed automata that describes the principals access to the data would be refined to only have o_1 and r_1 as principals in its start location. In regards to adding a policy to a security label this would be the equivalent of adding the required timed automata to the existing system of timed automata, which means that if a principal is able to gain access to the data then there should exist a system state where all timed automata in the system is in the *data* location at the same time. In regards to adding a reader to the reader set, which may only be done if the new reader can act-for an existing reader, the label on start location of the timed automaton that models principals' access to the data would be refined to reflect the new reader set. If an owner is replaced than all timed automata in the system should be refined to reflect that there is a new owner in this example if o_1 is replaced with o_2 then all labels in the timed automata should refine their labels such that o_2 is placed instead of o_1.

If a clock variable is added to a specific reader in a policy then a timed automaton must be created to reflect that principal's access to the data as it would be different from the other existing principals' access rights. However, there are certain cases where a new timed automaton is not needed such as if a new clock variable is added to an owner's clock expression for example if the clock variable y is added to the

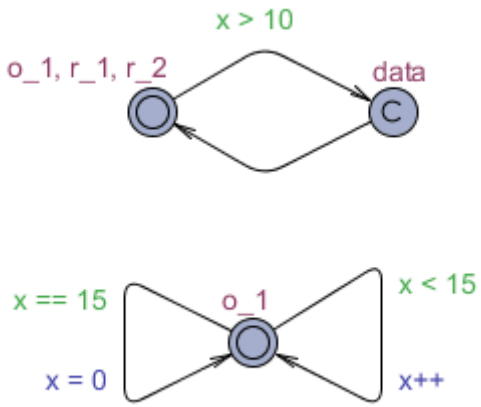

Figure 3.14: Timed automata expressing the security policy $\{o_1(x[15] > 10) : r_1, r_2\}$.

policy $\{o_1(x[15] > 10) : r_1\}$ resulting in the relabeled policy $\{o_1(x[15] > 10 \;\&\&\; y > 12) : r_1\}$ then the timed automaton that models access restriction could be refined to the timed automaton depicted in Figure 3.15, where the guard on the edge to the data has been refined to include the clock variable y and a new timed automata has been added to model the increase of the clock value for y. Finally, the comparison value for a clock variable may be changed if it reduces the window of opportunity for accessing the data for example if the comparison value of x is changed to 11 then the window of opportunity for accessing the data would shorten.

3.2.2 Integrity Policies

In regards to integrity labels a different set of safe relabeling rules need to be defined, as integrity policies are quality guarantees provided by the owners of the policies. Integrity rules are concerned with how trustworthy data are an as such the safe relabeling rules allows policy changes that warns of contamination or increases the trustworthiness. The relabeling rules for integrity labels can generally be expressed as the inverse of the safe relabeling rules for confidentiality labels and is defined in the TDLM as follows:

- **A writer may be added to a policy**: As in the DLM, a writer can be safely added to the writer set of an integrity policy as more writers serves as a warning of data contamination, which in terms means that as the more writer that are added the less trustworthy the data becomes.
- **Remove a policy**: A security policy may be removed from an integrity policy

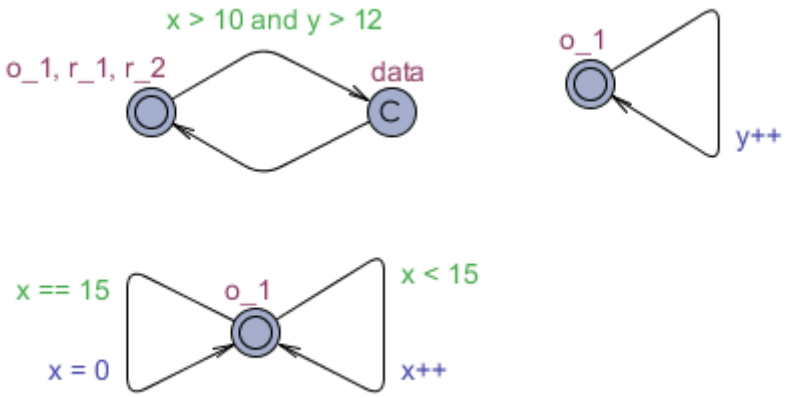

Figure 3.15: A refined timed automata expressing the relabeled security policy $\{o_1(x[15] > 10 \;\&\&\; y > 12) : r_1\}$.

as this would make the data more trustworthy as less principals are allowed to modify the data.

- **Replace a writer**: As in the DLM, a new writer w can be replace an existing writer w' if that writer can be acted-for by the existing writer $w' \succeq w$. This would allow more principals to modify the data and thus serve as a warning of contamination.
- **A policy may be added if it is identical to an existing policy**: A policy I may be safely added to an integrity label if it is identical to an existing policy J such that $o(J) \succeq o(I)$.
- **Writers that acts for the owner of a policy may be removed**: Principals that can act-for the owner of the integrity policy may be removed as they can already modify the data through the acts-for relationship with the owner.
- **Removal of clock expressions**: Clock expressions may be removed from security policies to warn of potential contamination as principals would be allowed to modify the data at any time.

3.2.3 Declassification

Declassification is a type of relabeling where the restrictiveness of the label is reduced and can only be performed by processes or principal which can act-for the owner of the security policy they want to declassify. This also means that if a label contains

more than one security policy then the process or principal may only change the policies where they have the authority of the owners. In the TDLM declassification should work in the same manner as it does in the DLM, and since clock variables only can be defined by owners or principals who can act-for these owners then processes or principals that have the authority to declassify a label also have the authority to change clock variables. Concretely this means that a process which has the authority of a policy owner may increase the time window where the data can be observed or changed for example the policy $\{o(x > 10) : r\}$ may be declassified to $\{o(x > 5) : r\}$ as this increases the time windows where readers associated with o may access the data protected by this security policy. This also means that clock variables may be removed entirely which corresponds to the largest possible increase in the time windows where the data may be accessed. However, comparison operators cannot be changed via declassification and as such the only parts of a clock expression that are allowed to change are upper limit, events, reset value, and comparison value. The inference rule defined by the DLM $L_1 \sqsubset L_2 \sqcup L_A$ would then also hold for the TDLM as L_A contains the policy $\{p : \}$ for every principal in the declassifying process' or principal's authority set.

3.3 Time-based Scenarios

To explore the capabilities of the TDLM, several security modeling scenarios which are only possible to model with the use of time are presented. These scenarios are used to depict the expressiveness of the TDLM and to serve as templates for constructing timed automata for common security policies.

3.3.1 Access to a System After Specified Time

A security modeling scenario that requires time-based access control is when a principal should only be able to access certain data after a specified amount of time has passed. An example smart meter security policy that enforces this constraint in the TDLM would be $\{c : e\ (x[91] >= 90)\}$, depicted as three timed automata in Figure 3.16, where c is the consumer, e is the electrical company, and x is a clock variable that specifies that e may only access the data protected by this label when x is equal to 90 or above. However, x is reset to zero when reaching the value 91 meaning that e may only access the data when x is equal to 90. This example depicts an electrical company accessing a consumer's smart meter data every 90 days which is a necessary constraint in regards to protecting the consumer's privacy as more fine grained data could be used to derive the consumer's daily routines [17], and as such time is required to model this scenario.

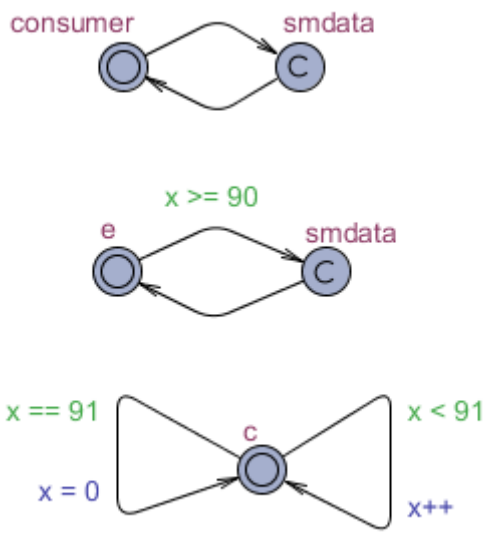

Figure 3.16: An example smart meter security policy $\{c : e\ (x[91] >= 90)\}$

Another security modeling scenario that models access to a system after a specified time is when an owner of some data specifies a time constraint on the entire reader set combined with a time constraint on a single principal. For example consider the label $\{o_1\ (y > 10) : r_1\ (z < 15), r_2\}$ where a time constraint on the entire reader set and a time constraint on the single principal r_1 are present as depicted with five timed automata in Figure 3.17. In this scenario both time constraints would be in effect meaning that r_1's access to the data is more restricted than r_2's access, meaning that r_2 would be able to access the data when y is between above 10 but r_1's time constraint should be viewed as a combined logical expression, which would in this case be $(y > 10\ \&\&\ z < 15)$.

3.3.2 Change Access Rights Based on Time

Principals' access rights are not modeled directly with security labels but are enforced via the principal hierarchy, which may change on run-time meaning that principals may lose or gain the possibility to access certain data when changes in the hierarchy occurs. Hierarchy changes may occur when certain events happen. However, label restriction may be used to depict change to access rights for specific data, for example if a variable v_1 is labeled with a security policy $\{o_1 : r_1, r_2\ (x >= 5), r_3\ (y > 15)\}$ and another variable v_2 is labeled with a security policy $\{o_1 : r_1, r_2\ (x > 5)\}$ and v_2 is assigned the data stored in v_1 then a join

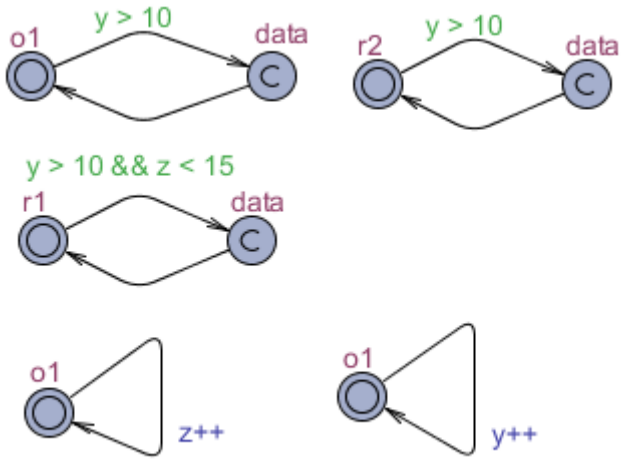

Figure 3.17: An example smart meter security policy $\{o_1\ (y > 10) : r_1\ (z < 15), r_2\}$

of the two labels would occur resulting in a policy $\{o_1 : r_1, r_2\ (x > 5)\}$ which is a restriction of access to the data originally stored in v_1. This could be viewed as a restriction to access rights as the clock variable would be further restricted and the possible reader set would also be further restricted.

Another example of label joining would be if a variable v_3 stores data labeled with the security policy $\{o_1\ ((x > 15\ \&\&\ y > 10)\ ||\ z < 5) : r_1\}$ and another variable v_4 stores data labeled with the security policy $\{o_1\ (p < 30\ \&\&\ x < 10) : r_1\}$ which would result in the joined label $\{o_1\ (((x > 15\ \&\&\ y > 10)\ ||\ z < 5)\ \&\&\ (p < 30\ \&\&\ x < 10)) : r_1\}$ where time restrictions would be combined with the logical && resulting in a combined time constraint. Six timed automata can be used to depict this scenario as shown in Figure 3.18.

3.3.3 Access to a System For Specified Time

A security modeling scenario that models access to a system for a specified amount of time is when clocks are reset at specific times or a combination of clock constraints only allows the readers or writes of the security policy to access the data for a limited amount of time. For example consider the label $\{o_1(x[10] > 5) : r_1\}$ where the reader set of o_1 only would be able to access the data for four time units because the clock is reset to zero when reaching 10 and the data can only be accessed when the clock variable x is above five. The access to the data labeled with this policy would be reoccurring as the clock variable is reset meaning that the principal r_1 can access the data more than once. This is depicted with two timed automata as shown in Figure 3.19.

Another example would be the policy $\{o_1(x > 10 \;\&\&\; x =< 15) : r_1\}$ where the access to the data is restricted to five time units by using the same clock variable, however the clock variable is never reset in this policy meaning that r_1 would only be able to access the data when x is between 11 and 15 and when x is above 15 then the data can never be accessed again unless some other event or policy resets x.

A final scenario is the policy $\{o_1(x > 10 \;\&\&\; y < 20) : r_1\}$ where the access to the data is restricted by two clock variables resulting in an access time of nine time units. This is also a non-reoccurring event as the clock variables are never reset, however if only y is reset to zero then the access to the data could reoccur while maintaining the original constraint.

3.3.4 Time Reset Based on Events

Another scenario is one time access to data which can be done via the use of events which resets a clock variable such that principals no longer can access the data and thus enforces a one time access to data within a time limit. For example, consider the policy $\{o(x[?reset] > 10) : r_1[*reset]\}$ where the owner o can access the data when x is above ten and so can r_1 but when r_1 successfully reads the data then the clock x is reset thus enforcing a one time access for r_1 every ten or more time units. The timed automata in Figure 3.20 depicts this behavior via the use of channel synchronization such that when r_1 successfully reads the data the *reset* channel is synchronized causing the clock x to be reset.

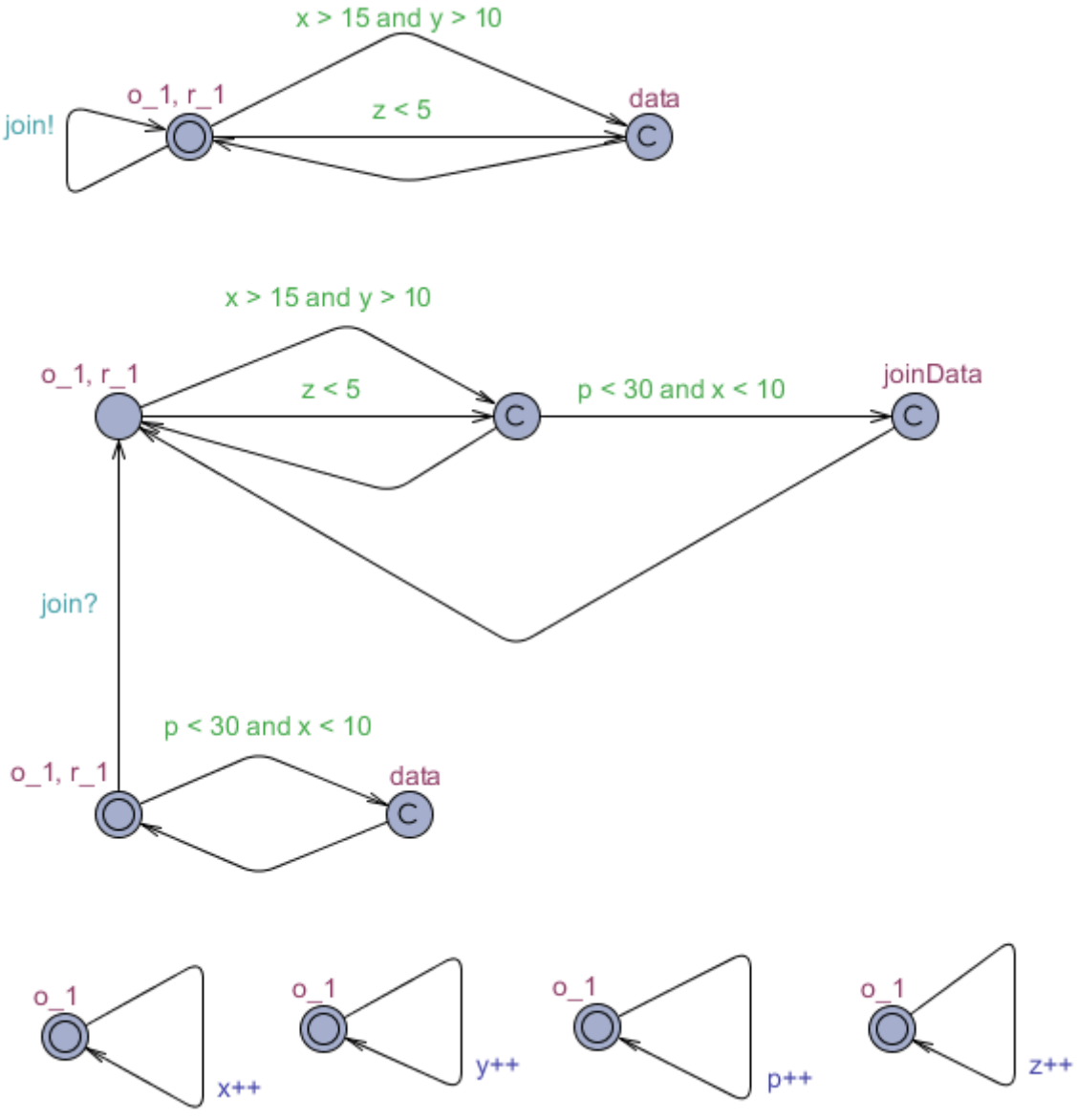

Figure 3.18: An example smart meter security policy $\{o_1\ (((x > 15\ \&\&\ y > 10)\ ||\ z < 5)\ \&\&\ (p < 30\ \&\&\ x < 10)) : r_1\}$

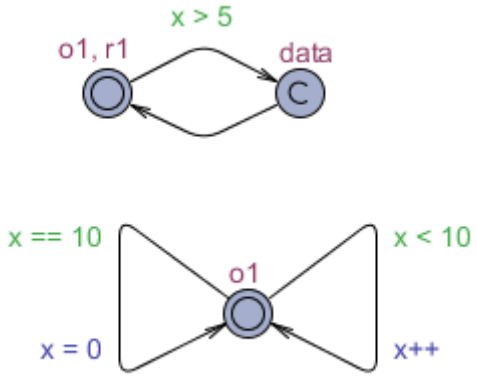

Figure 3.19: An example smart meter security policy $\{o_1(x[10] > 5) : r_1\}$

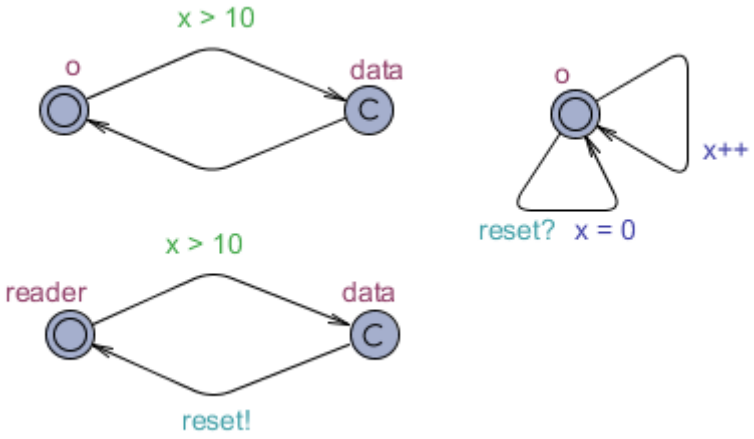

Figure 3.20: An example smart meter security policy $\{o(x[?reset] > 10) : r_1[*reset]\}$

CHAPTER 4

CASE STUDY

A study of how a real world system can be modeled with the TDLM is presented to show how the TDLM can be applied to an actual system that is dependent on secure information flow and access control. In Section 4.1 the smart meter system that the case study is based on is presented and clarified. In Section 4.2 several concerns in regards to privacy and confidentiality in the smart meter system is explored and in Section 4.3 the requirements for how access control should be handled is investigated to obtain knowledge of how to model the system's security. Section 4.4 describes, based on the previous investigations, how the smart meter system could be modeled with the TDLM.

4.1 Case Study: Smart Meter System

To explore the capabilities of the TDLM, a case study of a real world scenario is presented where most of the expressiveness of the TDLM is explored in detail.

The case study involves a smart meter system which consists of multiple users (household owners and residents), smart meters, electrical companies, distribution companies, smart meter manufactures, third parties, a data hub and a government entity as depicted in Figure 4.1. A definition of each entity in the smart meter system is as follows:

- **Users**: Two types of users exist in the smart meter system - household owners and residents. Household owners own one or more households, which each have a smart meter installed, and may also be residents in the households they own. Residents are permitted to live in a household by the household owner.

- **Smart meters**: A smart meter is responsible for collecting electricity consumption data for the household it is installed in. Furthermore, it also serves as a platform for other devices to connect to and communicate with for example for doing home automation.
- **Electrical companies**: An electrical company is responsible for delivering electricity to one or more customers and billing the customers according to their electricity consumption.
- **Distribution companies**: A distribution company is responsible for maintaining the power distribution grid and keeping track of which electrical companies users are associated with. Furthermore, it is responsible for processing raw smart meter data and making this data available to electrical companies via a data hub.
- **Data hub**: The data hub serves as a storage center for processed smart meter data which electrical companies or third parties can gain access to when appropriate.
- **Smart meter manufactures**: Smart meter manufactures are responsible for producing the smart meters and updating the firmware if need be.
- **Third parties**: Third parties are entities that might have an interest in the data collected by the smart meters such as research companies.
- **Government**: The government is interested in obtaining power consumption reports such that they can optimize the smart grid.

The users can monitor their own power consumption by directly communicating with the smart meter in their household such that they can identify how power can be saved. However, the household owner is capable of granting and revoking access to the smart meter for residents living in the household. The distribution companies request data from the smart meters they are associated with and perform necessary processing of the data in order to protect the privacy of the users. The processed data is delivered to a data hub which enforces which entities that can gain access to the data, when they can gain access to the data, for how long they may access the data, and the granularity of the data that may be accessed. The government and potential third parties can access the data they need through the data hub, however the permissions of each entity may vary for example a research company may be able to access more fine-grained data than the government. The electrical companies deliver electricity to the users and bill them according to billing data obtained from the data hub and as such an electrical company is implicitly associated with one or more smart meters/users.

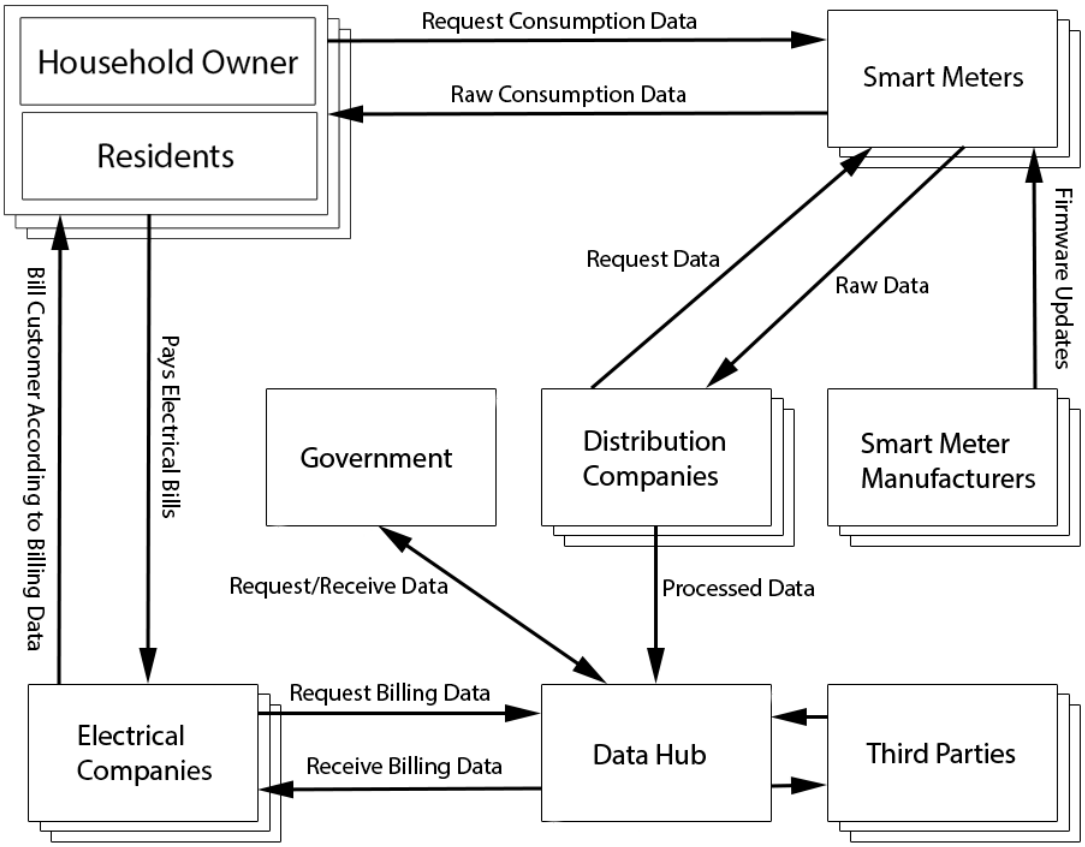

Figure 4.1: Overview of the smart meter system with communication outlined.

4.2 Smart Meter Privacy Concerns

A smart meter system makes it possible to observe fine-grained electricity consumption data of users associated with a smart meter. The system allows users to observe their own power consumption down to the minute which opens up the possibility of optimizing their usage patterns to lower overall power consumption [17]. In addition to this, the smart meter system provides users with a platform for home automation which can further lower their power consumption by automatically turning of unused devices or starting devices based on electricity prices [3]. Electricity prices are publicly available through the electrical companies and the power market, which in northern Europe is Nord Pool. However, the price the customer pays is dependent on which deal the customer has agreed to with their electrical company that regulates the electricity prices via Nord Pool or based on prices from the power plants [11]. This means that the smart meters can obtain the current electricity prices

directly from the electrical companies.

However, the possibility of reading fine-grained power consumption data also posses several privacy issues due to the possibility of deriving personal behavior patterns from the power consumption data, for example it is possible to figure out if a person is home during their sick leave or if they left late for work by observing when they consume power [17]. As such it would be favorable to regulate whom that has access to this data and in how fine a granularity the data can be extracted in relation to the entity that wants to observe the data.

Furthermore, the smart meter system contains certain strategic vulnerabilities that can be exploited by malicious third parties to alter the properties of the system such as changing the billing information, electrical prices, and consumption data [3]. In regards to communication outside the smart meter system, additional security vulnerabilities exist in regards to protecting the data collected by the smart meters, as a third party might perform man-in-the-middle attacks or retrieve the data for malicious use.

4.3 Access Rights

As there are several privacy and security issues in regards to smart meters, it should *not* be possible for all entities in- and outside the system to obtain data collected by smart meters or data transferred internally between entities. The smart meter collects data about the user's electrical consumption and as such the data collected by the smart meter are directly related to the user, meaning that there is no need to restrict the user's access to the data in any form. The user is able to observe and analyze the data collected by the smart meter at any time via for example a web interface.

As the smart meter data reveals private information about the user, the data should not be available in its original form to any other entity than the user. This means that the electrical company associated with the smart meter should not be able to observe the data at their convenience but rather a restriction should be placed on their capability to view the data. This restriction could for example be that the electrical company only can observe the data once every quarter which is enough for billing purposes. In addition to this, the data observed by the electrical company should not be the same fine-grained data that the user can observe but rather a summation of the total power consumption for the last quarter.

Access rights for each entity that have an interest in the smart meter data can be enforced by the data hub as all entities are required to fetch smart meter data through the data hub. Different access rights and data observation capabilities (how fine-grained data that can be accessed) can be giving to each entity according to their intentions in regards to the smart meter data. The distribution companies are

responsible for converting the raw smart meter data into data with a granularity of 15 minutes but the data hub is responsible for further processing the data as needed. The data hub is responsible for converting the data according to which entity that requests access to smart meter data such that each user's privacy is protected, for example the data hub may change the granularity of the data from 15 minute intervals to 24 hour intervals before sending the data depending on the entity that requests it. In addition to this, the data hub is responsible for only providing entities with data that are at least two weeks old such that the privacy of the users that the data involves are further protected. The reason for not providing entities with the newest data is that the most recent data may reveal what the user is doing at the current moment but if the data is a couple of weeks old then this would not be an issue.

In regards to the government entity, which is interested in obtaining power consumption reports to be able to regulate and optimize the power grid, they should not be able to obtain data directly from the smart meters as this would violate the individual user's privacy but instead they can obtain power consumption report from the data hub. This could be done by only providing the government with total power consumption for specific geographical sections for example total power consumption for a street, city district, city, or province. The data hub is responsible for performing this alteration of the data before sending it to the government.

If it is necessary to provide smart meter data to outside entities the data would need to be declassified with the authority of the corresponding smart meter and user, which means that per default no outside entity has access to any data residing within the smart meter system. However, if the outside entity enforces the security label placed on the variable and output channel then the data can be transferred to a third party without any need for declassification indicating that the system trusts the third party to enforce the security policies.

4.4 Smart Meter System Modeled with the TDLM

The smart meter system cannot be described by the DLM alone as this model lacks the possibility of defining time-based security policies, which are crucial in regards to modeling smart meter security. However, the TDLM extends the DLM with the required components for describing a smart meter system in regards to secure information flow and access control.

The principal hierarchy in the smart meter system would consist of all the users, the smart meters associated with those users, the electrical companies associated with users, distribution companies associated with smart meters, the data hub, smart meter manufactures and the government. Each user is associated directly with a smart meter in an acts-for relationship, meaning each smart meter can act-

for their associated user. Figure 4.2 depicts that a smart meter $\sum_{i=1}^{n} s_i$ can act-for their associated user $\sum_{i=1}^{n} u_i$, where i represents the specific user and smart meter and n represents the maximum amount of users and smart meters. Specifically, this means that a smart meter s_1 can act-for the user u_1. The electrical company $\sum_{j=1}^{m} e_j$ can be associated with many smart meters and as such j represents the specific electrical company and m the maximum amount of electrical companies. This means that a smart meter s_1 and s_2 both can be associated with an electrical company e_1. Each smart meter, and thereby users associated with this smart meter, is affiliated with a distribution company $\sum_{k=1}^{p} d_k$ which owns the smart meter and as such it can act-for the smart meter. The relationships between smart meters and distribution companies are that a single distribution company can be associated with one or more smart meters. As previously mentioned, users consists of both residents and household owners, however household owners can act-for residents meaning that the residents trust the household owners to update and maintain their personal information. The government g, the data hub dh, and the smart meter manufactures $\sum_{x=1}^{l} m_x$ are not in any acts-for relationships. Furthermore, it is assumed that no potential third party can act-for any of the other entities in the system.

The smart meter data can be divided into three segments; the first containing the personal information about the user that is associated with the smart meter at the current time, the second containing the real-time electrical readings recorded by the smart meter, and the third being the smart meter firmware which controls how the smart meter behaves. The first part of the data is owned by the user as it is sensitive information about them, but the electrical company that is associated with the user at the current time is allowed to read the data for billing purposes. In addition to this, the data should have an integrity policy which expresses that the user is the owner of the data but trusts the electrical company to change the data if needed, for example if the user moves to a different household. This gives the first part of the data the security label in Equation 4.1, where u_i corresponds to a user and e_j corresponds to the electrical company associated with the user.

$$\{u_i : e_j;\ !u_i :!!e_j\} \tag{4.1}$$

The second part of the smart meter data is the actual electrical readings that are saved by the smart meter every minute. As this data posses privacy concerns in regards to the user's behavioral patterns, the data should not be accessible for the electrical company associated with the smart meter at all times. To enforce this, the TDLM introduces time-based security policies such that it can be defined that the

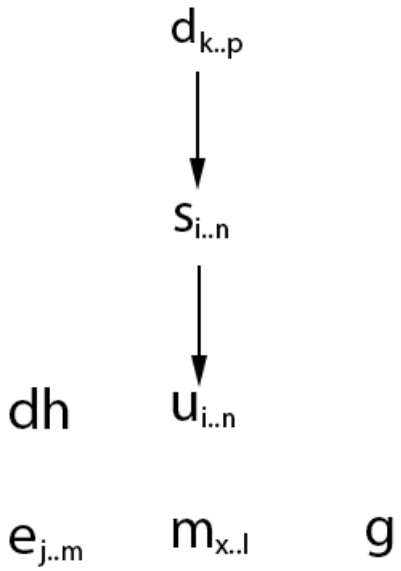

Figure 4.2: Smart meter system principal hierarchy. Only acts-for relationship is between distribution companies and smart meters, smart meters and users.

electrical company only should have access to the data once every quarter, which is the requirement for quarterly billing. The owner of the electrical readings is the smart meter and because the user should be able to monitor their own consumption, the user has unrestricted read access to the data. However, the electrical company associated with the smart meter is restricted by a clock variable indicating allowed access once when at least 90 days have passed. Equation 4.2 expresses these concerns in a security label, where the smart meter s_i is the only principal that can change the value of the clock x, which should be incremented once every day, and it is also the only principal that can alter the smart meter readings. In addition to this, the distribution company has unrestricted access to the data as it can act-for the smart meter such that it can read and process the data before making it available at the data hub. However, this means that the user trusts the distribution company to not modify the data in a harmful way as the distribution company has the power to do so. Furthermore, the electrical company is allowed to read the smart meter data once before the clock is reset, which is expressed by the event *reset*. The behavior of this security label is depicted in Figure 4.3 by four timed automata.

$$\{s_i : u_i, e_j(x[?reset : 1]^{\{!s_i:\}} > 90)[*reset];\ !s_i :\ \} \tag{4.2}$$

Furthermore, before the data is made available at the data hub a declassification of the data is performed to add the data hub as reader and writer such that

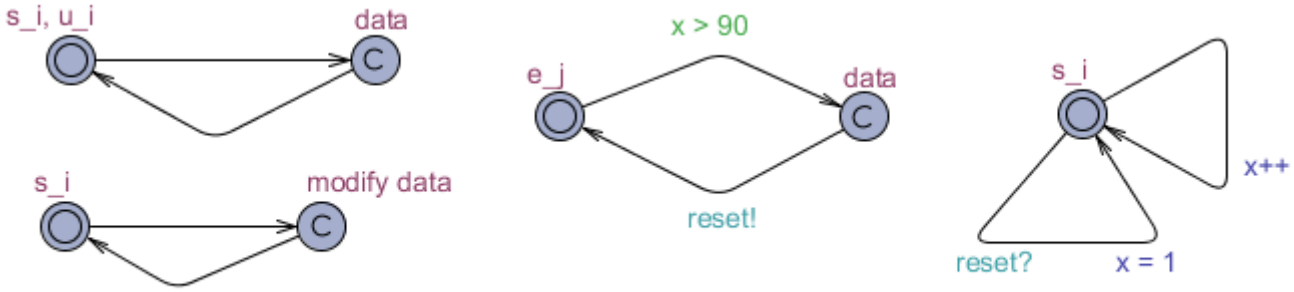

Figure 4.3: The security policies in Equation 4.2 modeled with four timed automata.

it can modify the data according to which principal that requests access to the data. Equation 4.3 describes the declassified label with the data hub dh added as both reader and writer. The distribution company is responsible for performing this declassification which can be done on the smart meter data without the user's authority as the distribution company can act-for the smart meter. The data hub is then capable of performing the necessary alteration to the data depending on which entity that requests access to it. The security label on the personal information linked to the smart meter data will not be altered, meaning that the identity of the user associated with the data is only known to the user and the electrical company associated with the user.

$$\{s_i : u_i, dh, e_j(x[?reset : 1]^{\{!s_i:\ \}} > 90)[*reset];\ !s_i :!!dh\} \tag{4.3}$$

The third part of the smart meter data is the smart meter firmware which is the software that dictates how the smart meters behave and what functionality that the smart meters have. The smart meter manufactures are responsible for maintaining and updating the smart meter software as needed and therefore they are the only principal that can change the smart meter firmware, as described by Equation 4.4 where m_x is the smart meter manufacturer.

$$\{!m_x :\ \} \tag{4.4}$$

The user associated with the smart meter can monitor their own power consumption in real-time via an output channel. As the user is the only principal that should be able to do this, the output channel is labeled with the security policy in Equation 4.5.

$$\{u_i :\ \} \tag{4.5}$$

The government can obtain a report from the data hub that contains total power consumption for a geographic location. The data used to fabricate this report comes from the smart meter data, which the government is not permitted to read and as such a declassification with the authority of the smart meters (and hence the users) are needed for passing the altered data on to the government. The data hub is responsible for altering the smart meter data so that it masks the users' behavioral patterns before giving the report to the government. As the data in the report is confidential it is declassified with the authority of the users, which means that the report will only contain the data of the users that have agreed to have their data declassified, so that the data hub is the owner of the data, the government is the sole reader, and the data hub is the only principals who can alter the integrity of the report as depicted in Equation 4.6.

$$\{dh : g;\ !dh :\ \} \tag{4.6}$$

Figure 4.4 gives and overview of the information flow and security labels that are present in the smart meter system. The arrows indicate information flow; the users can change and observe their personal information stored on their smart meter, and obtain electrical readings from their smart meter which can be viewed via an output channel. The electrical companies can obtain electrical readings from the smart meters they are associated with through the data hub, and change personal information about users they are associated with. Finally, the arrow between reports and government indicates that the government can obtain information from reports generated by the data hub when requested. The lines indicate associations, which means that an entity is associated with another entity in some way.

4.4.1 Security Label Changes

Two distinct events can occur in the smart meter system that forces a change to the security labels; a user moves from one household to another and a user changes electrical company.

When a user moves from one household to another the security label placed on both parts of the smart meter data would have to be changed to account for the change in smart meter ownership. The first step of the move process is that the distribution company declassifies the smart meter readings with the authority of the smart meter/user to be able to remove the time constraint such that a final billing can occur based on the user's electricity consumption since last billing period. This means that the electrical company can gain extraordinary access to the smart meter readings for this sole purpose. After the necessary data readings have occurred, the electrical company clears the personal information and the smart meter resets the electrical readings and the distribution company places a new security labels

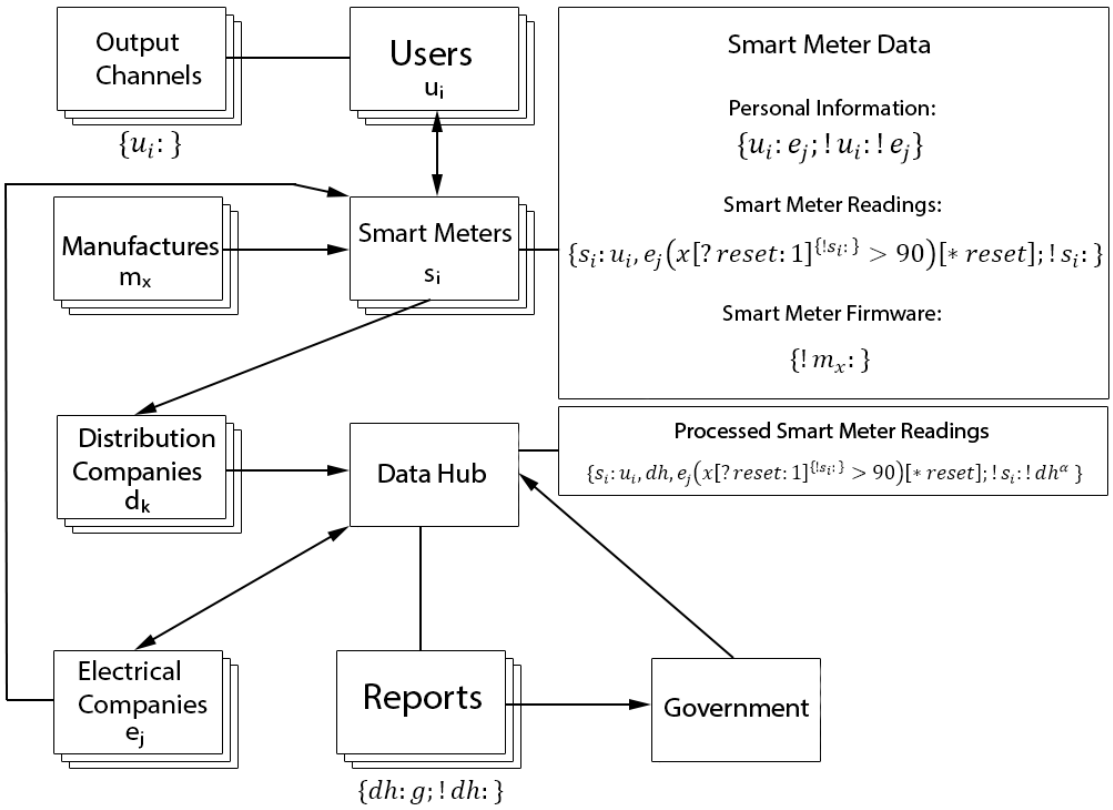

Figure 4.4: Smart meter system with security labels as per the definition of TDLM. Arrows depict information flow.

containing the new user associated with the smart meter. The same process occurs at both households to account for all affected smart meters. In addition to this, all user passed permissions would have to be revoked such that unintended residents does not have access to the new residents' electrical readings and personal information.

A user can also change electrical company as often as they want, which means that the security labels placed on the smart meter data would have to change. The first part of the process is identical to moving household, where the distribution company declassifies the electrical readings with the authority of the smart meter to be able to give the electrical company extraordinary access to the smart meter readings such that they can bill the user one last time. After this process the security label is changed to reflect that a new electrical company is associated with the smart meter. In regards to the first part of the smart meter data, which contains personal information about the user, the user would have to declassify the data and place a new security label reflecting the change.

In regards to the data hub creating consumption reports for the government, the data used for the reports would have to be declassified with the authority of the users (smart meters) meaning that if a user does not want to participate in electrical consumption statistics they can deny the data hub the possibility of declassifying their data thereby excluding them from the report.

4.4.2 User Passed Permissions

Users are capable of granting permission to other principals in regards to observing smart meter readings, which for example could be a husband that owns a property with a smart meter installed that gives his wife permission to observe the smart meter readings. This would be done by establishing an acts-for relationship between the principal that needs access to the smart meter data and the current owner of the data, which would indicate that the principal that is granted permission only has access to the data as long as the permission given principal has access.

However, if a resident moves to a new household the relationship between the resident and the current smart meter would be invalidated but the acts-for relationships established by the resident would still be in place, which means that the principals that were given permission would have access to the smart meter data in the resident's new household which may not be intended. As such, a clock variable can be placed on all new acts-for relationship created by the resident that gives the resident for example two days where the resident can decide which acts-for relationships to keep. This means that the other principals would not be able to act-for the resident in a given time period, which can be achieved via the clock expression in Equation 4.7, where the event $user\_i\_moves$ will reset the clock variable y to zero giving the resident two days where no resident defined acts-for relationships are active. The event will then occur when the last electricity reading is performed by the resident's current electrical company, meaning that the distribution company is responsible for declassifying the smart meter readings and placing the event in the security policy.

$$\{(y[?user\_i\_moves]^{\{!u_i:\}} > 2)\} \tag{4.7}$$

The security label for the smart meter data that is declassified when the resident moves to a new household would be the label defined in Equation 4.8, where the clock restriction on the electrical company is removed such that they can read the remaining data needed for the final billing and the event $user\_i\_moves$ is triggered causing the clocks in the principal hierarchy relating to the resident to be reset.

$$\{s_i : u_i, e_j[*user\_i\_moves];\ !s_i :\ \} \tag{4.8}$$

CHAPTER 5

EVALUATION

An evaluation of the TDLM is performed to identify which parts of the model that can be improved and how it may be done. Section 5.1 explores vulnerabilities that might exist in the TDLM and presents possible solutions to the identified vulnerabilities. Specifically, section 5.1.1 explores the problems associated with the principal hierarchy from the DLM. Section 5.1.2 studies window of opportunity exploits that might occur according to how policies are understood and implemented. An analysis is performed to visualize the impact on the window of opportunity according to clock parameters. Section 5.1.3 presents a complex security policy and explains the complications associated with complex security policies. Section 5.1.4 explains that the TDLM constructs are largely dependent on how they are implemented. Section 5.2 lists possible improvements and future work that can be done on the TDLM to provide a more complete solution.

5.1 Attacking the TDLM

To evaluate different security aspects of the TDLM, an investigation of the areas where we think that the TDLM could be vulnerable in regards to security loop holes and exploits is performed.

5.1.1 The Principal Hierarchy

In the TDLM the principal hierarchy works in the same manner as in the DLM except for the addition of clocks that restricts acts-for relationships. The addition of time-based restrictions in the principal hierarchy makes it more clearly defined and flexible as a certain set of rules for how acts-for relationships can described are

now present. However, the principal hierarchy is still loosely defined as there are no rules for changing relationships at run-time or defined who that can add these acts-for relationships.

As such the principal hierarchy is a vulnerability in both the DLM and TDLM because it could be subject to exploitation by a malicious third party that for example some how is able to add an acts-for relationship that makes him capable of observing or modifying restricted data. The likelihood of this happening is unknown at the moment as there are no clear rules for how the principal hierarchy works. However, the weakness of the principal hierarchy would be largely dependent on the implementation as there is no formal definition of how the hierarchy should behave and as such the vulnerability lies on the implementation.

To remove all doubt about the vulnerability of the principal hierarchy a certain set of rules should be defined that describe who that can add acts-for relationships to the principal hierarchy, when they can add acts-for relationships, who and when that can alter acts-for relationships at run-time, and which entity that enforces the principal hierarchy. However, these definitions of the principal hierarchy is out of scope for this report.

5.1.2 Window of Opportunity

Small windows of opportunity for performing non intended actions might exist in the TDLM when permissions (acts-for relationships) are altered in the principal hierarchy and when clocks are being reset.

When a system change occurs that requires the principal hierarchy to change, the revocation of acts-for relationships might be slow to propagate throughout the hierarchy compared to the change of the system and as such there could be acts-for relationships that are still present in the changed system state, which could cause a principal to gain access to restricted data that he was not suppose to access in the new system state. As an example consider the case study in Chapter 4 where a resident moves from a household to a new household which requires a system change, however as described in Section 4.4.2 a small window of opportunity exists where the resident still can observe data associated with his old household. To close this window of opportunity clocks which reset on certain events can be placed in the principal hierarchy, for example the clock $(x[?event] > 10)$ could be placed on an acts-for relationship to indicate that when the event occurs then the relationship is invalidated for ten days, which gives the system and the principal time to react to an eventual system change. Another solution to this problem could be to define that time cannot pass and access to any data is prohibited when changes in the principal hierarchy occurs. However, this solution requires that the principal hierarchy is formally defined in regards to how it should behave during changes to acts-for relationships.

Another scenario is a security policy with a clock that resets precisely when principals can start to observe the data $\{o : r(x[10] >= 10)\}$. In such a policy there might be an infinitely small amount of time where the principal r can observe the data even though the meaning of the policy might be that r never should be able to access the data. The reason for this window of opportunity comes down to how the reset event is handled, for example if the clock variable is reset an infinitesimally small amount of time before reaching ten then there is no window of opportunity but if it is reset an infinite small amount of time after reaching ten then the window persists. As such the window of opportunity in regards to security policies is dependent on how resetting clock variables are implemented and in which programming language it is implemented.

Analysis

To analyze the window of opportunity UPPAAL is used to verify a simple security policy $\{c : reader(x[15] >= 15)\}$ which is modeled in UPPAAL by two timed automata. Figure 5.1 depicts this scenario where the timed automaton P models the principal *reader*'s access rights and the timed automaton C models the behavior of the clock x. However, in UPPAAL the incrementation of clocks are handled by UPPAAL and as such the edge that increments the clock is left out. Four verification properties are checked via UPPAAL to verify the access possibilities within the modeled system:

- $A[]$ P.data imply $x >= 15$: States that for all possible system states when in the location *data* in the automaton P then the value of the clock x is larger than or equal to 15. This property is verified to be true which is the intended behavior of the system.
- $E <>$ P.data and $x == 14$: States that there exists a system state where the automaton P is in the location *data* and the value of the clock x is 14. This property is verified to be false which is the intended behavior of the system, as the security policy states that x should be larger than or equal to 15 before *reader* can access the data.
- $E <>$ P.data and $x == 15$: States that there exists a system state where the automaton P is in the location *data* and the value of the clock x is 15. This property is verified to be true which is the intended behavior of the system.
- $E <>$ P.data and $x == 16$: States that there exists a system state where the automaton P is in the location *data* and the value of the clock x is 16. This property is verified to be false which is the intended behavior of the system, as the security policy states that x should be reset to zero after reaching the value 15 meaning that x never reaches the value 16.

The verification done by UPPAAL shows that the system only allows access to the data according to the intended security properties defined in the policy meaning that the window of opportunity in this case is limited to access when the value of the clock x is exactly 15 giving the *reader* one time unit to access the data.

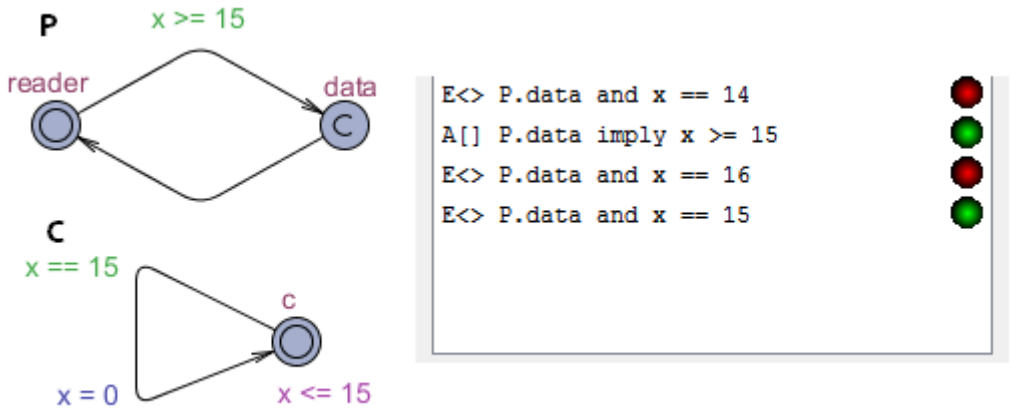

Figure 5.1: UPPAAL window of opportunity analysis where upper limit is set to 15.

Figure 5.2 depicts that the upper limit for the clock x has been altered to 16 which should expand the window of opportunity to allow the *reader* two time units to observe the data. According to the verifications done by UPPAAL the data can now be accessed when x is 15 and 16 but as the clock value never reaches 17 due to the new upper limit no system states exists where the *reader* can access the data when the clock value is 17 or above. The consequence of increasing the upper limit on the clock x from 15 to 16 effectively increases the window of opportunity from one time unit to two time units as *reader* now can gain access to the data when the clock value is 15 and 16.

Figure 5.3 depicts a scenario where the upper limit has been set to 14 which tighten the window of opportunity to zero time units as the clock value is reset before allowing the *reader* to observe the data. The property of the system being in the location *data* when x is larger than or equal to 15 still holds, however there does not exist a system state where the automaton is in the location *data* when the value of x is 14, 15, 16, or 17 meaning that the *reader* never can observe the data according to the verification performed by UPPAAL.

To show that changing the comparison value also affects the window of opportunity in the same manner that the upper limits affects it, consider the security policy $\{c : reader(y[10] >= 10)\}$ which is modeled and verified in UPPAAL as depicted in Figure 5.4 where the principal *reader* only can gain access to the data when the value of y is ten giving *reader* one time unit to observe the data. However, when modifying

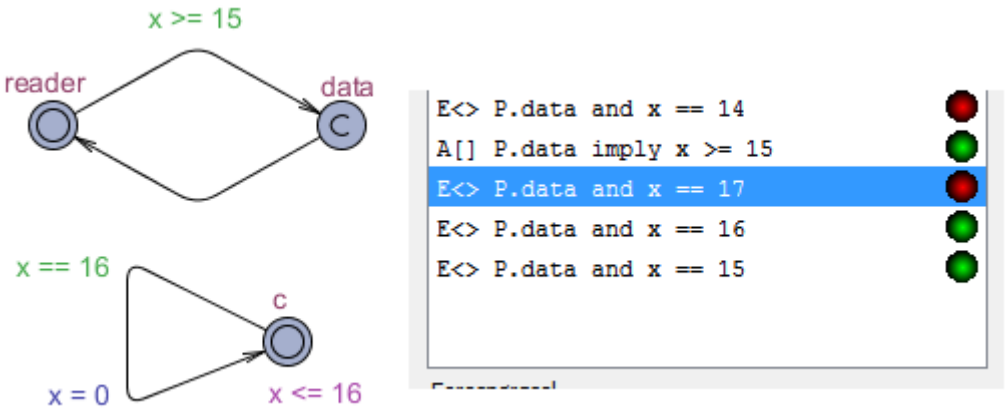

Figure 5.2: UPPAAL window of opportunity analysis where upper limit is set to 16.

the comparison value to nine yielding the security policy $\{c : reader(y[10] >= 9)\}$, *reader* now has two time units to observe the data as depicted in Figure 5.5. Finally, Figure 5.6 depicts when changing the comparison value to 11 yielding the security policy $\{c : reader(y[10] >= 11)\}$, *reader* now has zero time units to observe the data. The window of opportunity is as such expanded and tightened according to the comparison value.

If a security policy with an equals comparison $\{c : reader(y[10] == 10)\}$ is modified in regards to the comparison value or upper limit then if and when the window of opportunity occurs would change but the window would not expand as where the case with the other scenarios above. For example if the comparison value is changed to 11 or the upper limit is set to nine then the window of opportunity would disappear as the data would be inaccessible for the principal *reader*.

A preliminary conclusion of this analysis is that the window of opportunity is either expanded or tighten when modifying the upper limit or comparison value for a clock variable meaning that caution should be taken when altering how clocks are compared and reset as significant changes to how principals can observe data occurs when changing these values.

If an implementation of the TDLM was to be constructed where policies are verified by turning them into timed automata that are statically checked in UPPAAL then window of opportunity analysis could be performed on these timed automata to give the programmer of a system an idea of when and for how long each principal has access to the data protected by the policies.

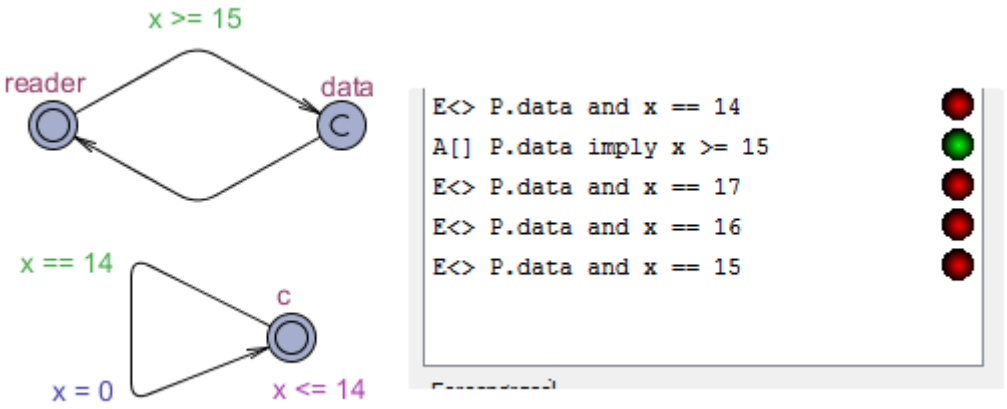

Figure 5.3: UPPAAL window of opportunity analysis where upper limit is set to 14.

5.1.3 Security Label Complexity

Complex security systems can be expressed with the use of the TDLM but the complexity of the security labels and the principal hierarchy may reach a point where it is hard to comprehend the actual meaning of the policies. As an example consider the security label in Equation 5.1 and the principal hierarchy in Figure 5.7 this simple security label becomes rather complex as it is combined with a principal hierarchy where the acts-for relationships are dependent on the value of the clocks placed on them.

$$\{p_3 : p2(f[20] < 5)[*reset]\} \tag{5.1}$$

To illustrate the complexity of this label the different restrictions for each principal are explicitly described below:

- p_3: The principal p_3 is the owner of the security policy defined in the security label in Equation 5.1 and as such there is no restrictions for p_3 when accessing the data protected with the label.
- p_2: The principal p_2 is restricted by either the clock expression $(f[20] < 5)$, which states that he may only access the data when this expression is true, or the clock expression placed in the principal hierarchy which states that he may act-for p_3 and thus freely access the data if the expression $(y[25] >= 10)$ is true. This effectively results in the combined clock restriction $(f[20] < 5 \;||\; y[25] >= 10)$ for the principal p_2.
- p_1: The principal p_1 may act-for p_2 to gain access to the data when the clock expression $(x[15] >= 5)$ evaluates to true, however this also requires that the

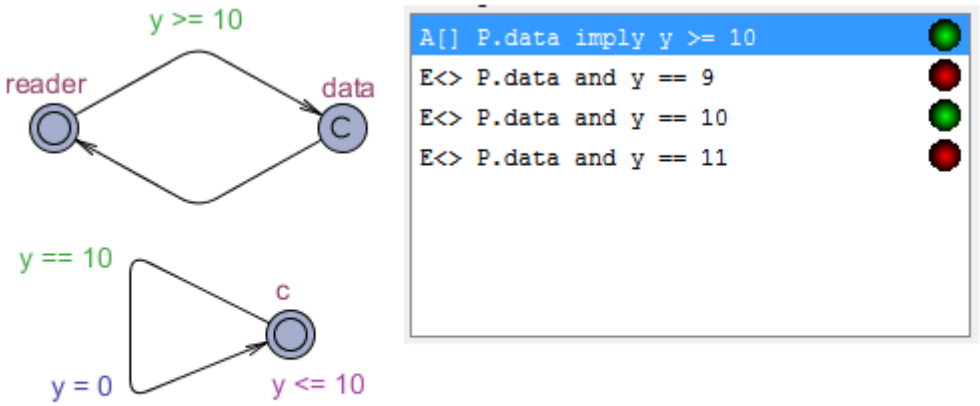

Figure 5.4: UPPAAL window of opportunity analysis where comparison value is ten.

clock expression placed on the relationship between p_2 and p_3 evaluates to true or the clock expression placed directly on p_2 in the security label is true. Furthermore, p_1 may act-for p_3 directly if the clock expression $(z[?reset] >= 35)$ is true but when p_2 successfully reads the data then the value of z would be reset making this acts-for relationship dependent on the actions of p_2. The explicit restrictions for p_1 would then be $(x[15] >= 5 \;\&\&\; (f[20] < 5 \;||\; y[25] >= 10) \;||\; z[?reset] >= 35)$.

As illustrated, the complexity of a rather simple security label may quickly escalate if combined with other TDLM constructs and as such the creator of security policies within a system should be cautious not to define security policies that may seem to express the intended behavior but expresses a different behavior when combined with other constructs.

To make sure that a policy actually enforces the security properties that were intended, UPPAAL could be used to perform a static analysis of the timed automata constructed based on the policies within a system written in an implementation of TDLM that allows this. This would provide a tool for the programmer to verify that the policies defined for the system actually corresponds to the intended security properties of the system.

5.1.4 Implementation

The TDLM provides a way of defining what security means within a system, but even though the model may be considered secure it is the implementation of the

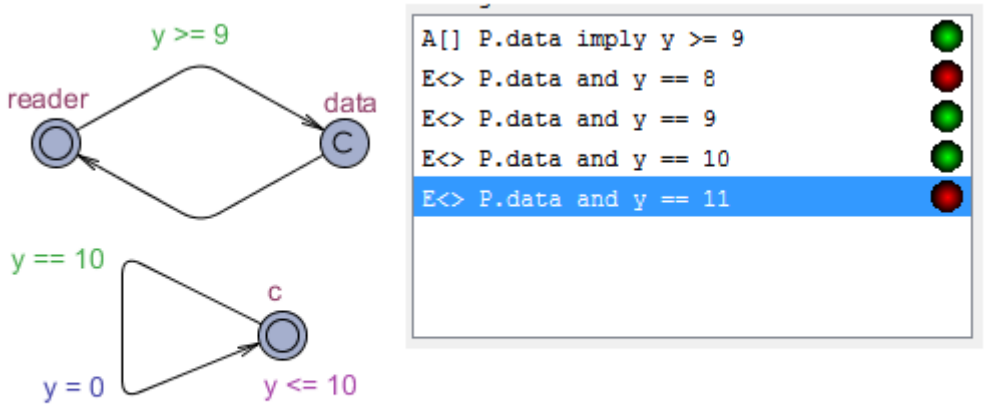

Figure 5.5: UPPAAL window of opportunity analysis where comparison value is nine.

TDLM that would be the target of a security violation attack. As a consequence of this, an implementation of the TDLM would need to be strict in regards to the implementation of the individual TDLM constructs and the way the constructs interact. A crucial part of the TDLM implementation would be the clock variables and how they are incremented and reset. Resetting a clock may be done in different ways which would impact how security policies are enforced, for example a clock may be reset just before reaching a certain value, exactly when reaching the value, or just after reaching the value. Concretely, consider the clock expression $(x[10] >= 10)$ where the meaning of the security policy changes according to when the implementation handles the reset upon reaching the upper limit. The three ways of handling reset impacts the security policy in the following way:

- **Before**: If the value of the clock variable x is reset an infinitely small amount of time before reaching the value ten then the clock variable blocks any principals associated with it from accessing the data.

- **Equal**: If x is reset exactly when reaching ten then then principals associated with the clock have an infinitely small amount of time to access the data, which may result in the principals being able to access the data or not depending on the concrete implementation.

- **After**: If x is reset an infinitely small amount of time after reaching ten then the principals have a small time window to access the data reliably.

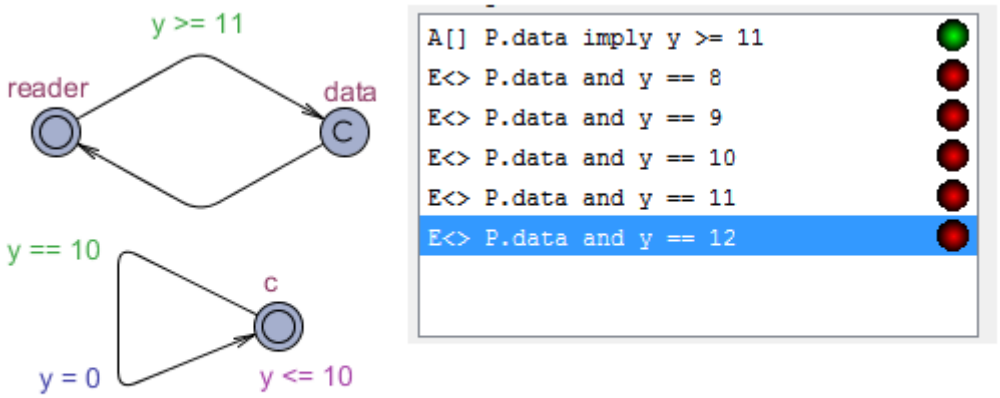

Figure 5.6: UPPAAL window of opportunity analysis where comparison value is 11.

5.2 Improvements and Future Work

Some aspects of the TDLM have been left unstudied as other areas have been the focus of this report. As such there are still several improvements and possible future work that can be done in regards to the TDLM.

A proof for the DLM constructs that are present in the TDLM would have to be constructed such as a formal proof that the join, meet, declassification, and complete safe relabeling rule still are sound and complete in regards to the TDLM even though additional constructs have been added to the model. In addition to this, the syntax for the TDLM could be improved to make it easier to use and understand and a formal syntax for the how to describe the principal hierarchy syntactically other than with a graph could be constructed. Ensuring confidentiality of clocks may prove difficult as the value of a clock may be inferred according to how often certain principals access data protected by the clock. In some systems the confidentiality of clocks may be required and as such a method of ensuring clock confidentiality should be further researched as the TDLM accepts that clock confidentiality does not adhere to the rule of noninterference [13].

Furthermore, the principal hierarchy and its acts-for relationships are not clearly defined in either the DLM or the TDLM and as such a formal definition of how the principal hierarchy is supposed to behave and how acts-for relationships may be defined and changed at run-time could be constructed. In addition to this, the TDLM introduces time in to acts-for relationships which means that a formal definition of how time in acts-for relationships can be defined is needed and a proof of that it does not interfere with the current definition of the principal hierarchy.

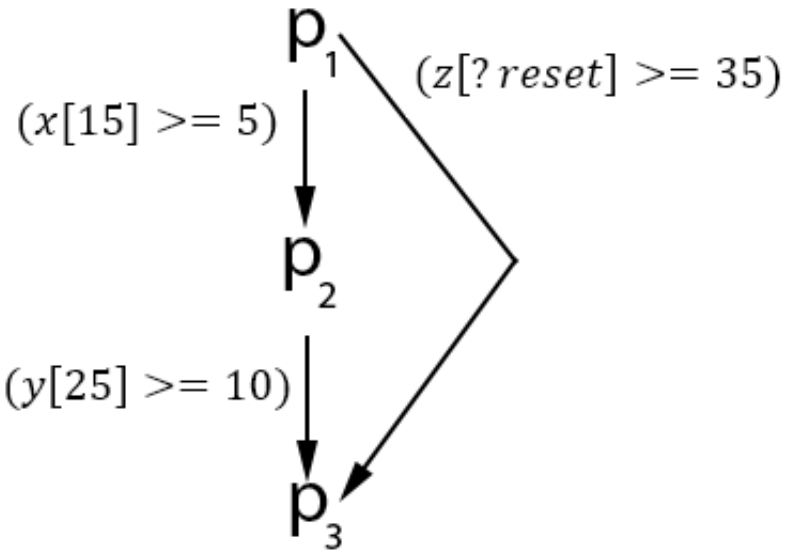

Figure 5.7: Principal hierarchy with time restrictions on all the acts-for relationships.

Finally, a complete implementation of the TDLM could be constructed such that it can be used in practice to ensure that systems developed with the TDLM implementation are secure in regards to the policies that are defined for each system. An implementation could for example be a new programming language that has the TDLM constructs built-in or an extension to the current implementation of the DLM, JIF [20, 23]. Furthermore, an implementation of the TDLM could utilize UPPAAL to statically check that security policies are correctly enforced in systems developed with the TDLM implementation by constructing timed automata based on the security policies defined in a system. In addition to this, UPPAAL could also be used to perform window of opportunity analysis when compiling to give the programmer an overview of when an for how long principals can access data, which can be used to asses whether the policies in a system are correctly defined.

CHAPTER 6

CONCLUSION

Information flow and access control are subjects in the area of computer security where extensive research has been performed, however none of the current security models [15, 7, 10, 4, 5, 6, 8] that have been developed take time into account when modeling systems. However, research in regards to time-based information flow and access control have been performed resulting in time-based security protocols and verification of these [9, 14] but currently no time-based security model that is based on information flow and access control has been developed. As a result of this, security in time-sensitive systems can be difficult to model as there is no well-documented and structured way of doing so.

Based on our previous research in regards to time-based information flow and access control [22], the Decentralized Label Model (DLM) was chosen as a basis for developing a security model that takes time into account. Security policies constructed via the DLM can be expressed as timed automata and as such these topics were extensively investigated to build a knowledge base for extending the DLM based on timed automata. The resulting extension to the DLM was the Timed Decentralized Label Model (TDLM) which extends the DLM with time-based constructs such that security policies can be expressed in terms of who and when access to data are allowed.

As the TDLM extends the DLM with new constructs a syntax was defined with the use of an Extended Backus-Naur Form that clearly defines how the syntax can be put together. One of the constructs introduced by the TDLM was clock expressions which can be placed on principals in a security policy to restrict their access to data based on time. More specifically, clock expressions consist of clock variables which are associated with clock parameters. Clock parameters consists of an upper limit, which resets the clock variable when a certain value is reached, a reset value, which

sets the value of the clock when reset, and events, which resets the clock when a principal reads or writes to data containing the event trigger. This allows the TDLM to express security policies where principals are restricted in certain time intervals for example a principal may only be allowed to access some data once every month which can be expressed via the use of clocks with certain clock parameters.

In addition to restricting principals' access to data in security policies, the TDLM also introduces clocks in the principal hierarchy which allows acts-for relationships between principals to be restricted based on time intervals resulting in the possibility of defining when certain principals trust other principals to act on their behalf. This allows for even more complex systems to be model with the TDLM as the principal hierarchy becomes more flexible than the one defined in the DLM.

The TDLM introduces incrementable time which is controlled by the principals that own the data protected by security policies. The clocks within security policies are incremented by the owner of the clock which is by default the owner of the policy, however principals that can act-for the owner of a policy may also define a clock in the policy. As a result of this, the TDLM introduces security labels on each clock such that it can be explicitly expressed who are allowed to increment and observe the clock which in terms means that principals can decide if a clock is trustworthy be observing who that may have incremented it.

The semantics of the TDLM is expressed via timed automata and a series of templates for how certain scenarios should be modeled semantically were presented, to give the users of the TDLM a foundation for understanding how security policies are enforced. The TDLM was used to model a real-world smart meter case to illustrate its use in time-sensitive systems as such a scenario could not be fully modeled with the use of DLM or any other well-known information flow and access control models. The case study involved making use of most of the added TDLM constructs and was a base for showing how the constructs could be used in real-world systems.

An evaluation of how the TDLM could be further improved was performed to identify vulnerabilities in the core model. The principal hierarchy was identified to be subject for further study and improvement as the DLM does not provide a clear and formal definition of how the principal hierarchy works in terms of adding and changing acts-for relationships, and as the TDLM makes use of the principal hierarchy from the DLM this problems persists in the extended model. A solution to the problem is a formal definition of the principal hierarchy along with a definition of the TDLM's additions to the hierarchy.

A window of opportunity analysis was performed to identify exactly how changing clock parameters in security policies affected the intended behavior of the policies. This resulted in the preliminary conclusion that clock parameters alter the window of opportunity for when principals may access data and caution should be

taken to ensure that security policies are defined correctly as even slight changes to clock parameters greatly impacts the meaning of the policy. In addition to this, it is possible to define rather complex security labels in the TDLM that may be incalculable for the user to comprehend. As such an automatic and generalized way of providing the users with information about the meaning of their defined security policies could be added to an implementation of the TDLM, such that defined policies are converted into timed automata which can be verified in UPPAAL to ensure correctness of the defined security policies and in terms correctness of the system implemented with the use of the TDLM implementation.

In regards to further study of the TDLM, formal proofs that the join, meet, declassification, and complete safe relabeling rule still holds for the TDLM should be constructed to verify the correctness of the model. Furthermore, a complete implementation of the TDLM could be constructed perhaps in the form a new programming language or an extension to the implementation of the DLM that makes use of UPPAAL to statically verify that security policies are enforced correctly.

BIBLIOGRAPHY

[1] Rajeev Alur and D. L. Dill. Automata for modeling real-time systems. *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*, 1990.

[2] Ross Anderson and Shailendra Fuloria. On the security economics of electricity metering. *Workshop on the Economics of Information Security (WEIS 2010)*, 2010.

[3] Ross Anderson and Shailendra Fuloria. Smart meter security: a survey. 2011. URL `https://www.cl.cam.ac.uk/~rja14/Papers/JSAC-draft.pdf`. Last accessed: 5/12-2014.

[4] D.E. Bell and L. J. La Padula. Secure computer system: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, The MITRE Corporation, Box 208, Bedford, MA 01730, March 1976. URL `http://csrc.nist.gov/publications/history/bell76.pdf`.

[5] K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, The MITRE Corporation, Box 208, Bedford, MA 01730, April 1977. URL `http://www.dtic.mil/dtic/tr/fulltext/u2/a039324.pdf`.

[6] David F. C. Brewer and Michael J. Nash. The chinese wall security policy. *Symposium on Security and Privacy, 1989. Proceedings., 1989 IEEE*, 1989.

[7] Niklas Broberg and David Sands. Paralocks – role-based information flow control and beyond. *Symposium on Principles of Programming Languages (POPL'10)*, 2010.

[8] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. *1987 IEEE, Symposium on Security and Privacy*, 1987.

[9] R. Corin, S. Etalle, P.H. Hartel, and A. Mader. Timed model checking of security protocols. *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, 2004.

[10] Dorothy E. Denning and Peter J. Denning. Certification of program for secure information flow. *Communications of the ACM, July 1977, Volume 20, Number 7*, 1977.

[11] Energinet.dk. Roller. URL `http://www.energinet.dk/DA/El/Engrosmarked/Viden-om-engrosmarkedet/Sider/Roller.aspx`. Visited: 15/4 2015.

[12] International Organization for Standardization. *Information technology - Syntactic metalanguage - Extended BNF*. International Organization for Standardization, 1st edition, December 1996. URL `http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf`. Last accessed: 25/5-2015.

[13] J. A. Goguen and Mesquer J. Security policies and security models. *Proceedings of the 1982 IEEE Symposium on Privacy and Security*, 1982.

[14] Gizela Jakubowska and Wojciech Penczek. Is your security protocol on time? *International Symposium on Fundamentals of Software Engineering*, 2007.

[15] Andrew C. Meyers and Barbara Liskov. Complete, safe information flow with decentralized labels. *Proceedings of the IEEE Symposium on Security and Privacy, May 1998*, 1998.

[16] Andrew C. Meyers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology (TOSEM), Oct. 2000*, 2000.

[17] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet, and David Irwin. Private memoirs of a smart meter. *BuildSys 2010*, 2010.

[18] Edward F. Moore, editor. *Sequential Machines: Selected Papers*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1964. ISBN B0000CMBI0.

[19] Andrew C. Myers. Mostly-static decentralized information flow control. Technical report, Massachusetts Institute of Technology, January 1999. URL `http://www.cs.cornell.edu/andru/release/tr783.pdf`.

[20] Andrew C. Myers. Jflow: Practical mostly-static information flow control. *Proceedings of the 26th ACM Symposium on Principles of Programming Languages (POPL)*, 1999.

[21] Department of Information Technology at Uppsala University (UPP) in Sweden and the Department of Computer Science at Aalborg University (AAL) in Denmark. Uppaal. URL `http://uppaal.org`. Visited: 12/2 2015.

[22] Michael H. Sørensen and Martin Leth Pedersen. Smart meter security and time based information flow control. Technical report, Department of Computer Science, Aalborg University, January 2015. URL `http://aauprojekt.mikdev.dk/report.pdf`.

[23] Danfeng Zhang, Owen Arden, Jed Liu, K. Vikram, Stephen Chong, Andrew Myers, Nate Nystorm, Lantian Zheng, and Steve Zdancewic. Jif reference manual, December 2014. URL `http://www.cs.cornell.edu/jif/doc/jif-3.3.0/manual.html`. Last accessed: 18/12-2014.