Modelling Social-Technical Attacks with Timed Automata

Nicolas David *
University of Nantes/LINA, France
nicolas.david1@univ-nantes.fr

Alexandre David
Aalborg University, Denmark
adavid@cs.aau.dk

René Rydhof Hansen
Aalborg University, Denmark
rrh@cs.aau.dk

Kim G. Larsen
Aalborg University, Denmark
kgl@cs.aau.dk

Axel Legay
Inria, France
axel.legay@inria.fr

Mads Chr. Olesen
Aalborg University, Denmark
mchro@cs.aau.dk

Christian W. Probst
Technical University of Denmark
cwpr@dtu.dk

ABSTRACT

Attacks on a system often exploit vulnerabilities that arise from human behaviour or other human activity. Attacks of this type, so-called *socio-technical* attacks, cover everything from social engineering to insider attacks, and they can have a devastating impact on an unprepared organisation. In this paper we develop an approach towards modelling socio-technical systems in general and socio-technical attacks in particular, using *timed automata* and illustrate its application by a complex case study. Thanks to automated model checking and automata theory, we can automatically generate possible attacks in our model and perform analysis and simulation of both model and attack, revealing details about the specific interaction between attacker and victim. Using timed automata also allows for intuitive modelling of systems, in which quantities like time and cost can be easily added and analysed.

Categories and Subject Descriptors

D2.4 [**Software/Program Verification**]: Formal methods; Model checking; H.1.2 [**User/Machine Systems**]: Human factors

Keywords

Insider threats; timed automata; attack trees; attack generation

*The work for this paper was done during an internship at Aalborg University.

MIST'15 October 16, 2015, Denver, Colorado, USA

DOI: http://dx.doi.org/10.1145/2808783.2808787.

1. INTRODUCTION

When considering attacks on organisations, the "chain of security" is no stronger than the weakest link in the chain. While users are not the main source of vulnerabilities of organisations, often the weakest link involves some form of human behaviour or activity, e.g., executing a particular work flow or following a predictable routine. Such *socio-technical attacks* exploit human behaviour in conjunction with technology, and range from social engineering of the unwary to abuse of insider knowledge and access. Combined with the rapid adoption of information technology in all aspects of life, the risk and potential impact of such attacks are quickly exploding, and are likely to continue to increase rapidly as well in the coming years.

In recognition of this problem, several novel modelling formalisms have recently been developed that explicitly take human behaviour into account when modelling a system [3, 9, 10]. The goal of these formalisms is to enable easier and more direct modelling of socio-technical systems, and thereby also the potential socio-technical attacks against them. The advantage of having formal models of a system, is that they enable the full range of formal methods to be used in modelling, analysing, and verifying the security of a system. They are thus an important element in identifying threats against the modelled organisation. However, while these formalisms are excellent for modelling and formal (manual) analysis, they all lack automated tool support, e.g., for large scale modelling and (automated) analysis of attacks. This lack of tool support naturally limits the size and complexity of systems that can be usefully modelled and analysed.

The goal of the TREsPASS project [14] is to close this gap by developing mod- els and analytic processes that support risk assessment in complex organisations including human factors and physical infrastructure. The goal of this support is to simplify the identification of possible attacks and to provide qualified assess- ment and ranking of attacks based on the expected impact.

In this paper we show how this problem can be solved by using *timed automata* as an intermediate formalism for modelling socio-technical attacks and systems, e.g., by translating models from one of the modelling languages mentioned

above into a timed automata model. Using timed automata enables the application of state-of-the-art tools for analysis of the models, including the use of model checking for finding attacks against a system. In particular, we show how a real-life case study, developed and studied in the TREsPASS project [14] can be modelled and analysed using timed automata. We furthermore show how the *attack trees*, resulting from the extensive attack modelling of the case study carried out in the TREsPASS project, also can be re-cast using timed automata and thereby enable the use of model checking tools for automatically finding attacks against the model. Indeed, using the UPPAAL model checker [1], we succeed in finding a novel attack variation that was not previously considered by the attack tree analysis.

In addition to finding new attacks, model checking and related techniques can be used for many other kinds of analysis that are useful for evaluating the security of a system, e.g., the potential impact of an attack or the time it takes to perform an attack. Also potential locations of attackers at time t after an attack or observed event can be approximated by the techniques presented.

In summary, we consider the following to be the main contributions the paper: *(1)* a novel application of timed automata for modelling socio-technical attacks and systems; *(2)* a general technique for converting, or re-interpreting, attack trees as timed automata.

The rest of this paper is structured as follows. After discussing related work in the next section, we present the case study considered in Section 3. Our automata-based approach for attack modeling is introduced in Section 4 together with experimental results, followed by a discussion of more advanced modeling of victim and attacker actions in Section 5. Finally, Section 6 concludes this paper.

2. RELATED WORK

There are two major areas of related work: attack representation and (socio-technical) system models. The former is mainly concerned with representing possible attacks against a system, in a way that allows for efficient computation of relevant properties, e.g., cost or likelihood of an attack. The goal of the latter is to model entire systems, not only attacks, in order to find or generate (new) attacks, simulate attacks, e.g., to evaluate countermeasure, and also for computing relevant properties.

One popular approach to modelling attacks including socio-technical attacks, is that of *attack trees* [12, 13]. Attack trees are a formalism inspired by fault trees, well-known in safety engineering, and are used to succinctly, in a top-down manner describe a set of attacks or attack scenarios[1]. The top-down approach of attack trees, combined with the inherent compositionality, makes attack trees very convenient, not least for non-experts. From a formal methods perspective, the tree structure supports efficient algorithms for analysis of attack trees, at least for properties that naturally can be computed in a compositional manner. Nevertheless, attack trees have several limitations: *(1)* It is non-trivial to give a formal foundation for attack trees, cf. [8]. *(2)* It is non-trivial to integrate, and formalise the semantics of, certain important operators into the attack tree formalism, most notably sequencing. *(3)* Important factors in an attack are not (normally) modelled as part of the attack tree, e.g., victim actions and countermeasures, although so-called *attack-defense trees* attempt to integrate the latter. *(4)* Since trees do not allow for sharing of nodes or leafs, attack trees may contain significant redundancy (in [7] an approach is discussed using directed acyclic graphs to allow for sharing). *(5)* While simple quantities, like time and cost, can be easily added to attack trees, more complex quantities such as probabilities, e.g., likelihood of a successful attack, require additional assumptions on the model of attack trees.

In the socio-technical modeling approach we do not only represent the attacks, but also all the relevant parts of the entire system (ideally), including physical infrastructure, IT infrastructure, as well as actors, in particular human actors, within the system. Modelling the entire system also to capture *operational* models of attacks to be made. Such operational attack models can capture detailed aspects of an attack in terms its interaction with the attacked system and thus reveal properties both of the attack and the system that would otherwise be impossible or difficult to find.

Designing socio-technical modelling languages is a relatively new field; recent contributions include approaches such as ExASyM [10], Portunes [3], and ANKH [9]. The ExASyM modelling formalism was created with the goal of analysing security properties in the context of (human) workflows and physical infrastructure, e.g., would an intruder be able to access a restricted area during an emergency. In Portunes, the focus is on *connectivity* and, in particular, how connectivity can change over time. Finally, the ANKH (Actor-Network Hypergraph) model, also aims at modelling connectivity and interaction, but takes a more abstract approach using hypergraphs as the underlying formalism. All three languages support a graphical notation, making it easier for non-experts to design models, and all three also support various forms of analysis, e.g., through specialised static analysis or model checking. However, in order to apply such tools, the socio-technical models must first be converted into a formalism supported by the tool of interest, which may lead to loss of fidelity in the model.

Recent work on identifying attacks in system models aims at analysing and invalidating policies [4, 5, 6] or at analysing systems directly [15]. In contrast to these approaches, we identify attacks based on model checking the timed automata representing the system.

3. HOME PAYMENT SYSTEM: A CASE STUDY

In this section we describe the "IPTV case study that will be used as a running example throughout the paper[2]. The case study is based on an internal case study of the TREsPASS project. The technical details as well as the people and companies involved are confidential and we therefore present an anonymised and slightly redacted version of the original case study. However, all the important features have been retained and the work in this paper has also been performed on the original case study with similar results.

The case study concerns a system supporting primarily elderly and disabled people in performing online payments

[1]In fact there are many different and competing definitions of attack trees. They all share (only) the basic idea of a top-down attack goal recursively refined, either conjunctively or disjunctively, in a tree structure.

[2]Here IPTV refers to television service(s) provided over the IP protocol.

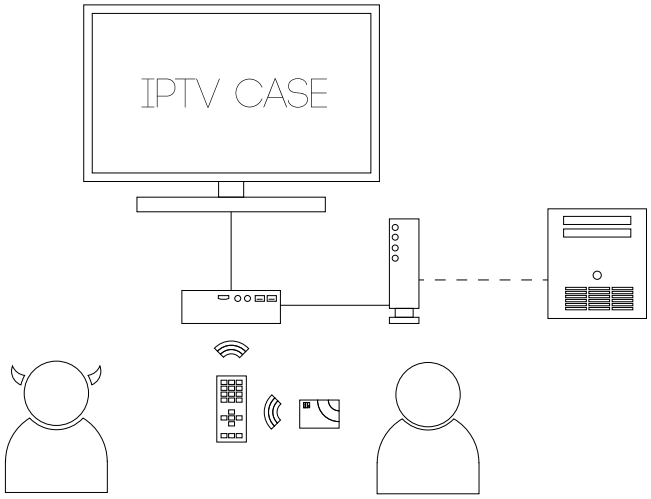

Figure 1: IPTV case study

and managing their accounts from their home. With the target demographic in mind, the system should be integrated into an existing device that is familiar and easy to use for the intended user groups, namely the television set. In practice this is accomplished by hooking up a small, dedicated computer to the TV and an enhanced remote control with a built-in card reader for authentication as illustrated in Figure 1.

This case study features many different security aspects that may be considered: from the strictly technical, such as how information is protected while stored or transmitted, to the socio-technical, covering security issues arising from the use of and interaction with the technology. In this paper we focus mostly on the socio-technical aspects, both to demonstrate the potential for formal modelling of such aspects, but also because that is where the technical skills required of a potential attacker are at the lowest and thus admits a much larger group of attackers.

Figure 1 shows an overview of the IPTV case study. There are two primary actors: the attacker (represented by a devil in the figure) and the victim (the IPTV owner/user). Under normal operation, the user would first open a session on the IPTV, using a standard, password-based authentication scheme. From this session, the user can then use different services, e.g., pay a bill or transfer money, by using a payment card with the concomitant PIN code. The payment card is read by a card reader built into the IPTV remote control on which the PIN code is also entered.

In the following we make a number of assumptions about the context for the case study to simplify treatment:

1. The card-holder has a functional IPTV in his/her house prior to the attack.
2. The IPTV security configuration ensures security for the communication of data between the different physical devices.
3. One Internet Service Provider (ISP) is used for all Internet access.
4. The source code of the software of the IPTV system is not freely available.
5. Firmware updates are not cryptographically encoded.
6. The IPTV set-top box uses a standard API.
7. The user can log on and off the IPTV system at will.

While these assumptions help delineate the scope of the case study, they are not critical and can be relaxed or modified to better capture a specific system.

Before considering actual attacks against the IPTV system, the goal or goals of the attacker must be decided upon. Since the IPTV system is designed to handle money and payments, one obvious choice goal for an attacker is, *how to "acquire" money using the IPTV system.* Within the TREsPASS project, domain experts have developed an attack tree to explore different attacker strategies for achieving this goal. We will not repeat the analysis and attack tree here, but instead focus on three of the attack strategies considered:

1. Steal payment card and related access codes; gain physical access to the IPTV system and transfer money.
2. Social engineer (threaten, blackmail, trick) the card-holder to transfer money.
3. Manipulate the hardware and/or software of the IPTV set-top box to allow remote access for money transfer or stealing critical information.

The first two attack strategies are not IPTV specific and can be applied to a wide range of targets, e.g., net banking systems in general. We have included them here to illustrate how such essentially non-technical, human-based attacks can be modelled and analysed in our approach.

The last of the above attacks can be implemented by the attacker gaining physical access to the IPTV set-top box and installing modified hardware and/or software. Alternatively, the attacker can use *social engineering* techniques to make the victim, i.e., the owner of the IPTV set-top box, install the modified hardware and/or software. Figure 2 shows a slightly modified part of an attack tree representation of these possible attacks, originally developed by domain experts within the TREsPASS project. As discussed in Section 2, attack trees are a top-down approach to exploring

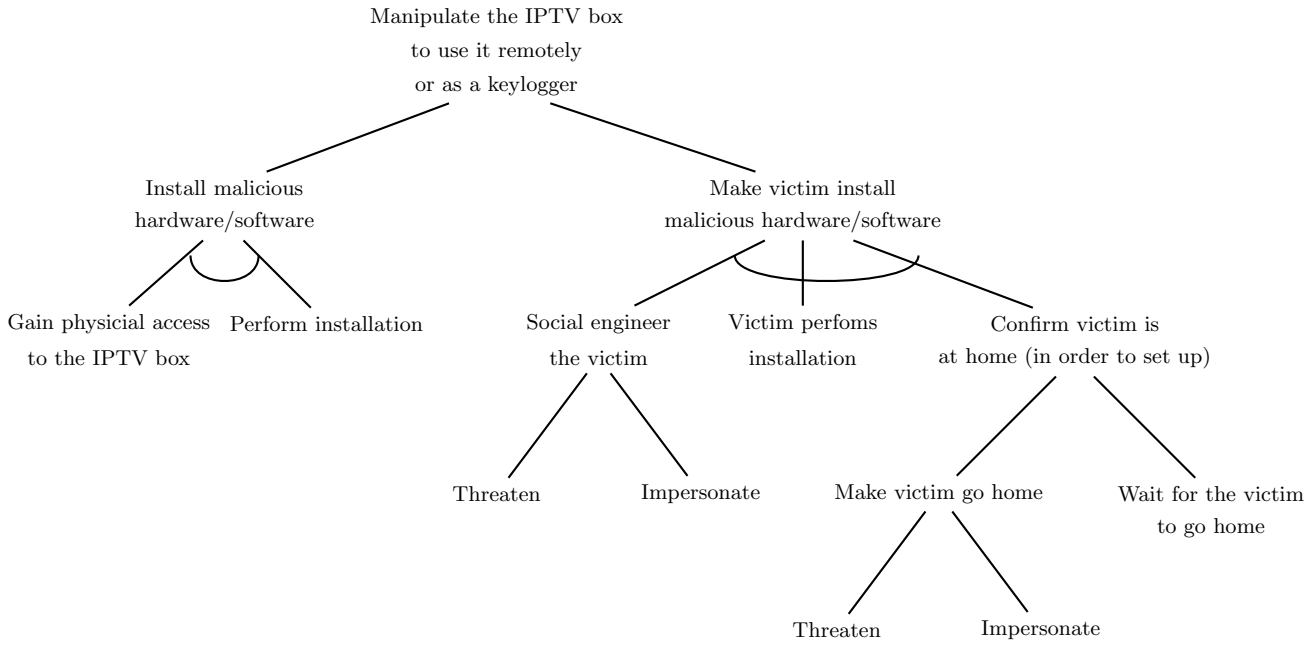

Figure 2: Subtree for attacking the IPTV system

possible attack strategies. The focus is on the attacker's primary goal, which is placed as the root node, and further refined into sub-goals. This refinement is continued recursively until reaching a sub-goal that corresponds to a basic action, which is then placed as a leaf node. The sub-goals can either be conjunctive or disjunctive with the former representing steps that are *all* necessary to reach the goal and the latter representing alternatives for reaching a particular sub-goal [11]. In Figure 2 conjunctive sub-goals are shown with an arc across the edges connecting to further sub-goals.

The attack strategies described in the preceding paragraph will all be modelled as timed automata using our approach. In this modelling, each of the mentioned attack sub-goals will be refined further and formalised. One of the benefits of using timed automata, rather than trees, is that some of the redundancy that is evident even in a simple attack tree such as the one shown in Figure 2, can be reduced or even completely avoided. This is important in several ways, for example, to ease understanding the attack tree. Most importantly, reducing redundancy also reduces the total size of the model, making automated analysis more feasible.

4. AN AUTOMATA-BASED APPROACH FOR ATTACK MODELING

In this section, we introduce our automata-based approach for modelling socio-technical systems. For lack of space, we do not give the formalisation in full detail but refer instead to [2].

The fundamental concept in our formalisation is called a *step*. Similar to the sub-goals in attack trees, steps allow to describe the different goals and sub-goals of an attack. While attack trees contain sub-trees for sub-goals, each step is formalised as a separate automaton describing the behaviour of that step. In contrast to the inherently dynamic nature of automata, nodes in attack trees are static.

The step-automata use locations to represent their current status, which can be *Idle*, *Ready*, *Chosen*, or *Done*. An example of this can be seen in Figure 3. Moving from *Idle* to *Ready* requires that some dependencies are satisfied. Boolean resources are used to realize dependencies between steps that must be enabled to switch from *Idle* to *Ready*. Once a step is successfully completed, new resources are generated *(set to 1)* or consumed *(reset to 0)*, updating the set of steps which are ready to be performed. Dealing with conjunctions and disjunctions within those dependencies is taken in account by using set resources that allow to perform a step.

In Figure 3, transitions of the first automaton are labelled with guards or updates linking the behaviour to those dependencies. As an example, *isReady()==1* tests if one set of resources is fully enabled, whereas *updateResources()* updates the corresponding resources. We use an encoding in UPPAAL to provide those data to automata: each step is automatically instantiated over a specification.

DEFINITION 1 (STEP). *Let $\mathcal{R}$ be the set of resources and $\mathcal{S}$ be the set of steps. A step s is a set (n, d, D, E, M) where :*

- *$n \in \mathbb{N}$ is the number of sets enabling the step;*
- *$d \in \mathbb{N}$ is the number of dependency couples used to describe the step;*
- *D is the set of dependency couples so that $|D| = d$ and $\forall e \in D, e = (r, w)$ where $r \in \mathbb{N}$ and $r < N\_RESOURCES$ is a resource needed and $w \in \mathbb{N}^*$ and $w \leq n$ is a set in which r is needed;*
- *E is the boolean array of length $N\_RESOURCES$ of enabled resources*

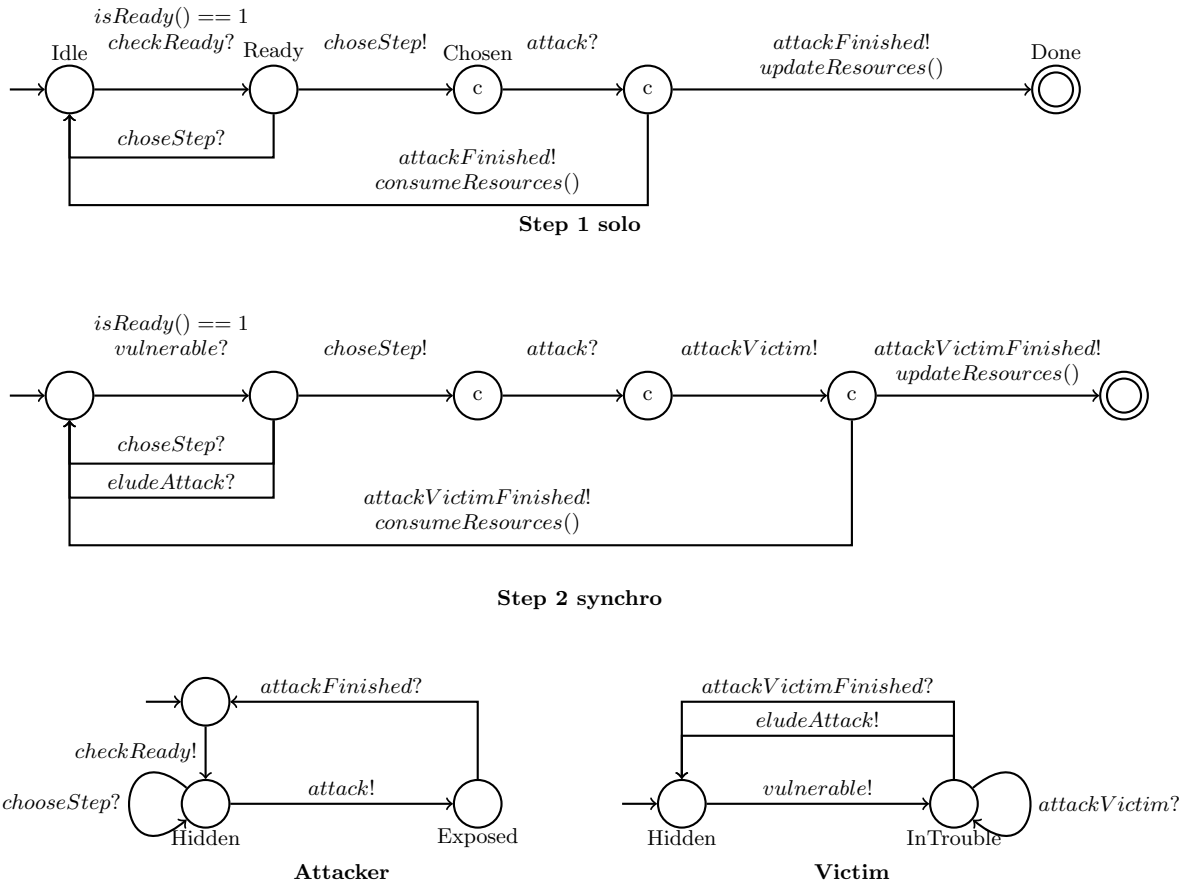

Figure 3: Network of attack composed of 2 steps

$\forall i < N\_RESOURCES, E[i] = 1 \Leftrightarrow s$ *enables* i *otherwise* $E[i] = 0$*; and*

- M *is the boolean array of length* $N\_RESOURCES$ *that masks the disabled resources*
  $\forall i < N\_RESOURCES, E[i] = 0 \Leftrightarrow s$ *disables* i *otherwise* $E[i] = 1$.

Step 1 and step 2 of Figure 3 illustrate the potential concurrency between steps. An attack scenario is defined a *sequence of steps* and can be represented by the network of timed automata of Figure 3.

Definition 2 (Scenario). *An attack scenario* p *is a sequence of steps* $s_0, s_1, ..., s_n$ *over* $\mathcal{S}$ *such that:*

- $\exists set_{j_0} \in 2^{\mathcal{R}}$ *a set of resources that enables* s_0 *such that* $\forall r \in set_{j_0}$, $status_{init}(r)$ *is true*
- $\forall s_i, \exists set_j$ *a set of resources such that* $\forall r \in set_j$,$status_{p,si-1}(r)$ *is true,*

where $status_{p,s}$ *is a function from* R *to* $\{false, true\}$ *and* p, s *is a scenario and a step of this scenario or the keyword init such that,* $status_{p,s}(r)$ *equals to the value of* r *once the path* p *achieved the step* s *if* $p, s \neq init$ *and to its initial value otherwise.*

To model the effect of an attacker, the network of timed automata can be extended to include an automaton depicting the activity of the attacker. In Figure 3, this extension is illustrated by the *Attacker* automaton. Each time the attacker reaches the hidden location, a new step can be chosen among the ready steps set.

Similar to the attack automata, the network of timed automata can also include an automaton representing victim behaviour, which marks a substantial increase in expressiveness compared to attack trees. In Figure 3 this component is the *Victim* automaton. Indeed, this automaton simulates a victim's activity, generating and consuming specific resources and creating new opportunities for the attacker. The victim's behaviour may for instance allow the attacker to bypass some steps in order to reach the goal faster. On the other hand, the victim can create pitfalls that would make the attack more difficult.

Victim and attacker behaviours are specified in the UPPAAL encoding. We could for instance imagine an action *lose the payment card* that would allow the attacker to find it without stealing it. This introduces a refinement in the description of steps: on the one hand, some steps can be performed by an attacker on its own: "*preparing a malware, enable a location...*". We call those steps *solo steps*; on the other hand, some steps require the victim to be vulnerable in order to be performed: *steal a password, make the victim do something...* We call those steps *synchro steps*. Synchro steps can be performed when the victim is in the *vulnerable state*. This explains why the two automata of Figure 3 *solo steps* wait for the attacker's signal *ready* whereas the *syn-*

Description	request	result	states explored	CPU time used (ms)
Standard model	E<> (status[20]==1)	true	1422	160
	E<> Step(3).Success	true	1691	210
	E<> Step(8).Success	true	1595	190
Refined model	E<> (status[20]==1)	true	7103	780
	E<> Step(8).Success	true	8002	850
	E<> Step(13).Success	true	45684756	1.614e+06

Table 1: Requests computed with an Intel(R) Core(TM) i7-4702MQ

chro steps wait for the victim's signal *vulnerable*. A *goal* is then defined by the possibility to reach a step or to generate a resource.

4.1 Finding Possible Attacks

Using the above modeling approach, it is now straightforward to use model checking to generate *all possible* attacks on a system. We are mainly interested in reasoning about the goal of the attack and about properties of (sets of) resources.

To validate our approach to attack generation, we have applied it to the case study as described in Section 3 (in the following called "standard model"), as well as a version of the case study where some of the attack goal were further refined (in the following called "refined model"). The refinement does not change the case study as such, but enables more specific attacks to be generated and thus more states to be covered by the model checker.

We do not discuss the generated model in more detail here, but note that in both variants a successful attack is represented by the same resource number "20". We use model checking to test if this resource can be enabled, and then generate the shortest trace to enable it. This is approach enabled by UPPAAL options that extract examples from model checking. Alternatively, we can try to find if one of the steps which enable resource "20" can be reached successfully, that is to say, reach the state *Done*. For the first model, the corresponding steps are Step 3 (representing *Make the victim use IPTV*) and Step 8 (*Use IPTV*), for the second (refined) model the corresponding steps are step 8 (*Make victim use service*) and Step 13 (*Use service*).

The queries to identify attacks are simple reachability requests, which provide results and examples of scenarios that fit the conditions. On the model for the case study, UPPAAL finds an interesting scenario that was not captured by the attack tree description of this case study shown in Figure 2, which was developed by domain experts: *victim turn on IPTV - attacker enable pub/house location - victim go to the pub - attacker steal the card and the pin - attacker use the card and the pin to use the service*. Such a scenario

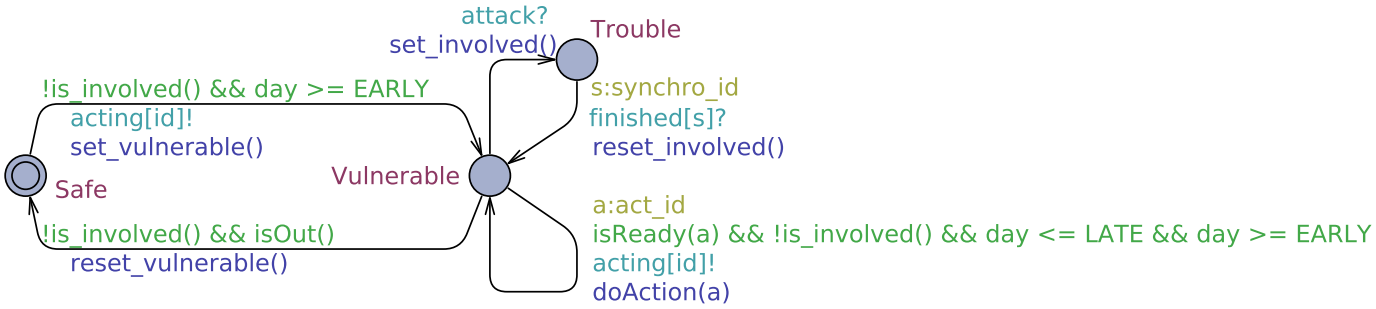

Figure 4: Victim template

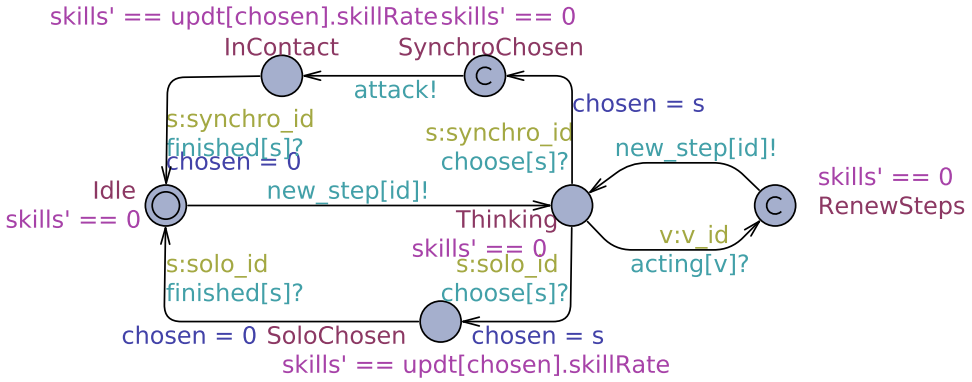

Figure 5: Attacker template

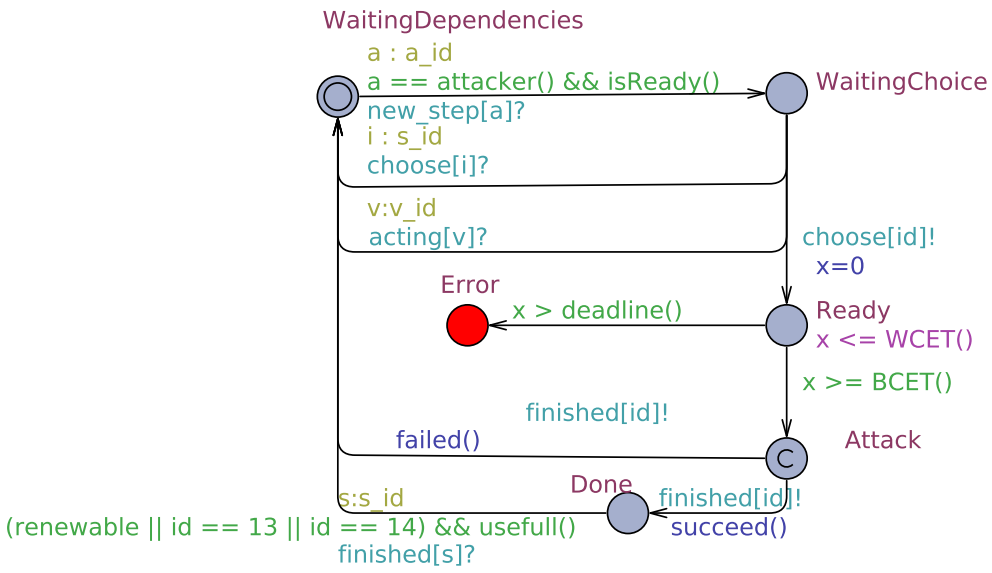

Figure 6: Step template

is only possible thanks to the influence of the victim and the use of dynamic resources to simulate the model creating opportunities of attack. From this scenario a simple counter measure can be proposed: *always log-off an account when you do not use it*. Such a counter measure would be more difficult to obtain with an attack tree, as we deduced it from a trace which reflects the opportunities that arise while the victim is acting.

Some statistics of our model checking experiment are shown in Table 1. Note that in the refined model the number of explored states increases rapidly. In order to deal with this *state space explosion* in general, reduction techniques have to be applied. We refer to [2] for further discussion and a solution.

5. BEYOND ATTACK GENERATION

In the previous sections we have introduced our approach to attack generation based on networks of timed automata. In this section, we use the diversity provided by automata-based modeling to present an improved model.

Since the actions of victims (as well as those of attackers) are explicitly represented in our approach, it is straightforward to extend the victim model to take more complex interactions into account, as illustrated by the template in Figure 4. Both the attacker and the "step" templates have to be updated correspondingly. These are shown in Figures 5 and 6, respectively.

In the new victim model, the transition from *WaitingChoice* to *WaitingDependencies* labeled by *acting[v]?* means that we have to verify dependencies each time that the victim performs an action which may change the set of enabled resources.

The behaviour of the victim has been abstracted: we created a set of resources depicting the status of the victim. When the victim becomes *vulnerable*, the corresponding resource is set to 1. This resource is tested when former synchro steps are verifying their dependencies. Therefore, we can keep one global template for *solo steps* and *synchro steps*. In any case, we need to specify if a step involves the victim or not. This is realised with the resource called *involve*. Indeed, when the victim is *involved* in a step, the victim must not be able to act. This is a simple way to create semaphore over the resources, without involving a process that would increase the state space and complicate the model. We also define a subset of *consumable* resources over which the victim has control. Last, we define some physical locations for the victim.

This new template comes with several assumptions: the victim can only act over a limited set of resources and the victim's actions are atomic. In the initial state, the victim is not involved and not vulnerable.

As future work, our approach lends itself easily to further extension and experiments involving, e.g., statistical model checking to introduce stochastic behaviour and statistical estimates of attacker and victim behaviour.

6. CONCLUSION

We have presented a timed-automata based model to deal with socio-technical attack problems such as insider threats. Based on a simple encoding, non-experts would be able to specify and simulate a complex system thanks to this kind of tool. The systematic analysis provided by UPPAAL allows finding interesting reachability results, which confirm the strength of model-checking in the field of attack study. Sequential modeling of attack considering victim's behaviour is especially relevant for dealing with insider threats, where the weakness of the system comes both from the system and the user.

Therefore we emphasize a simple but relevant countermeasure based on generated traces. Moreover using automata opens perspectives toward a use of real time simulation in such problems, but also prices or probabilities which represent relevant future work areas to face the challenges raised by socio-technical attack study. As future work, we plan to study how *statistical model checking*, using the UPPAAL SMC tool, allows for more nuanced models of both attackers and victims through weighted choices for the ac-

tors of the model. We also are investigating the relation of our approach with policy invalidation [4, 5, 6].

7. ACKNOWLEDGMENTS

Part of the research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (TREsPASS). This publication reflects only the authors' views and the Union is not liable for any use that may be made of the information contained herein.

References

[1] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[2] N. David. TREsPASS project. Master's thesis, Ecole Centrale de Nantes (IRCCYN), France, 2014. Completed as part of internship at Aalborg University.

[3] T. Dimkov, W. Pieters, and P. H. Hartel. Portunes: representing attack scenarios spanning through the physical, digital and social domain. In *Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10)*, volume 6186 of *Lecture Notes in Computer Science*, pages 112–129. Springer.

[4] M. G. Ivanova, C. W. Probst, R. R. Hansen, and F. Kammüller. Attack tree generation by policy invalidation. In *9th International Conference on Information Security Theory and Practice (WISTP)*. Springer, 2015.

[5] F. Kammüller and C. W. Probst. Invalidating policies using structural information. In *2nd International IEEE Workshop on Research on Insider Threats (WRIT'13)*. IEEE, 2013.

[6] F. Kammüller and C. W. Probst. Combining generated data models with formal invalidation for insider threat analysis. In *3rd International IEEE Workshop on Research on Insider Threats (WRIT'14)*. IEEE, 2014.

[7] B. Kordy, L. Pietre-Cambacedes, and P. Schweitzer. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. CoRR/arXiv:1303.7387, 2013.

[8] S. Mauw and M. Oostdijk. Foundations of attack trees. In *Proceedings of the International Conference on Information Security and Cryptology (ICISC 2005)*, volume 3935 of *Lecture Notes in Computer Science*, pages 186–198. Springer, 2005.

[9] W. Pieters. Representing humans in system security models: An actor-network approach. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 2(1):75–92, 2011.

[10] C. W. Probst and R. R. Hansen. An extensible analysable system model. *Information Security Technical Report*, 13(4):235–246, Nov. 2008.

[11] X. Qin and W. Lee. Attack plan recognition and prediction using causal networks. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, pages 370–379, Dec. 2004.

[12] C. Salter, O. S. Saydjari, B. Schneier, and J. Wallner. Toward a secure system engineering methodology. In *Proceedings of the 1998 New Security Paradigms Workshop (NSPW'98)*, pages 2–10.

[13] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, Dec. 1999.

[14] The TREsPASS Consortium. Project web page. Available at `http://www.trespass-project.eu`.

[15] R. Vigo, F. Nielson, and H. R. Nielson. Automated generation of attack trees. In *Proceedings of the 27th Computer Security Foundations Symposium (CSF)*, pages 337–350. IEEE, 2014.