Formal Modeling and Analysis with Humans in Infrastructures for IoT Health Care Systems

Florian Kammüller

Middlesex University London, f.kammueller@mdx.ac.uk

Abstract. In this paper, we integrate previously developed formal methods to model infrastructure, actors, and policies of human centric infrastructures in order to analyze security and privacy properties. A fruitful approach for discovering attacks on human centric infrastructure models is invalidation of global policies. Invalidating global policies by a complete exploration of the state space can be realized by modelchecking. To counter the state explosion problem inherent in modelchecking, Higher Order Logic (HOL) supported by the interactive theorem prover Isabelle can be used to emulate modelchecking. In addition, the Isabelle Insider framework supports modeling and analysis of human centric infrastructures including attack trees. In this paper, we investigate how Isabelle modelchecking might help to improve detection of attack traces and refinement of attack tree analysis. To this end, we use a case study from security and privacy of IoT devices in the health care sector as proposed in the CHIST-ERA project SUCCESS.

1 Introduction

The expressive power of HOL allows modeling the process of social explanation inspired by Max Weber into an Isabelle Insider Threat framework. We applied this framework to case studies from airplane safety and security [8], insider threats for the IoT [10], and for auction protocols [9]. The CHIST-ERA project SUCCESS [2] will employ the framework in combination with attack trees and the Behaviour Interaction Priority (BIP) component architecture model to develop security and privacy enhanced IoT solutions. A pilot case study from the health care sector, cost-effective IoT-based bio-marker monitoring for early Alzheimer's diagnosis, will enable us to investigate the feasibility of the approach.

The Isabelle Insider framework [14] is used as a basis for a formalisation of an architecture-level description of the infrastructure including human actors, their psychological disposition, and core privacy and security requirements integrated as logical predicates of local security and privacy policies. The modelchecking procedure advocated in the invalidation approach to Insider threat analysis [13] has meanwhile been incorporated into the Isabelle tool [7]. Its applicability has been demonstrated by means of an example on the analysis of an earlier IoT case study [10]. However, in this earlier IoT case study, we originally extended the Isabelle Insider framework by the concept of attack trees to refine known IoT Insider attack vectors. Attack trees allow refining known attack vectors into

sequences of state transitions explaining how the attack leads to a state in which the security property is violated. Thus the refined attack corresponds to a path in the state graph of the system model. Similarly, the process of modelchecking produces automatically a sequence of state transition – if the checked property does not hold in the model.

The question we investigate by means of an IoT health care case study is whether the concepts of modelchecking and attack tree refinement correspond. The extension by modelchecking [7] and the embedding of attack trees into the Isabelle Insider framework allow us to examine this correspondence using the mathematical rigour and automated proof support of Isabelle. The results provide important insights on how the methods of modelchecking and attack tree analysis can be fruitfully combined to enhance the verification of attacks on human centric infrastructure models and possibly even the discovery of yet unknown ones.

This paper begins by briefly reviewing the Isabelle Insider framework with a special emphasis on the extensions to modelchecking as well as attack trees. The running example of a simple health care scenario and its privacy and security risks is then introduced followed by the presentation of its formalisation in the Isabelle Insider framework. We reconsider the definition of state transition in modelchecking introducing an adaptation that explicitly shows the attack paths. This allows the transformation of attack traces found by modelchecking into the attack tree refinement process. We show how these processes relate. As illustrated by the case study, we can use the combination of modelchecking and attack trees to guide the attack tree refinement in finding and analysing the attacks in human centric scenarios.

2 Isabelle Insiders, Modelchecking and Attack Trees

In formal analysis of technical scenarios, the motivation of actors and the resulting behaviour of humans is often not considered because the complexity is beyond usual formalisms. The Isabelle Insider framework [14] provides expressiveness to model infrastructures, policies, and humans while keeping up the level of proof automation. In this section, we give a short introduction to this framework for modeling and analysing Insider attacks. We describe its extensions by attack trees and modelchecking. A detailed technical introduction to the framework is given in [14], the extensions are introduced in [10,7] and the Isabelle sources are available online [6].

2.1 Isabelle Insider framework

The Isabelle Insider framework [14] is based on a logical process of sociological explanation [3] inspired by Weber's *Grundmodell*, to explain Insider threats by moving between societal level (macro) and individual actor level (micro).

The interpretation into a logic of explanation is formalized in Isabelle's Higher Order Logic. This Isabelle formalisation constitutes a tool for proving security

properties using the assistance of the semi-automated theorem prover [14]. Isabelle/HOL is an interactive proof assistant based on Higher Order Logic (HOL). Applications can be specified as so-called object-logics in HOL providing reasoning capabilities for examples but also for the analysis of the meta-theory. Examples reach from pure mathematics [11] to software engineering [5]. An object-logic contains new types, constants and definitions. These items reside in a theory file, *e.g.*, the file `Insider.thy` contains the object-logic for social explanation of Insider threats (see [14,6]). This Isabelle Insider framework is a *conservative extension* of HOL. This means that our object logic does not introduce new axioms and hence guarantees consistency.

The micro-level and macro-level of the sociological explanation give rise to a two-layered model in Isabelle, reflecting first the psychological disposition and motivation of actors and second the graph of the infrastructure where nodes are locations with actors associated to them. Security policies can be defined over the agents, their properties, and the infrastructure graph; properties can be proved mechanically with Isabelle.

In the Isabelle/HOL theory for Insiders, we express policies over actions `get`, `move`, `eval`, and `put` We abstract here from concrete data – actions have no parameters:

```
datatype action = get | move | eval | put
```

The human component is the *Actor* which is represented by an abstract type and a function that creates elements of that type from identities:

```
typedecl actor
type_synonym identity = string
consts Actor :: string ⇒ actor
```

Policies describe prerequisites for actions to be granted to actors given by pairs of predicates (conditions) and sets of (enabled) actions:

```
type_synonym policy = ((actor ⇒ bool) × action set)
```

We integrate policies with a graph into the infrastructure providing an organisational model where policies reside at locations and actors are adorned with additional predicates to specify their 'credentials', and a predicate over locations to encode attributes of infrastructure components:

```
datatype infrastructure = Infrastructure
    "igraph" "location ⇒ policy set" "actor ⇒ bool" "location ⇒ bool"
```

These local policies serve to provide a specification of the 'normal' behaviour of actors but are also the starting point for possible attacks on the organisation's infrastructure. The `enables` predicate specifies that an actor `a` can perform an action `a'`∈ `e` at location `l` in the infrastructure `I` if `a`'s credentials (stored in the tuple space `tspace I a`) imply the location policy's (stored in `delta I l`) condition `p` for `a`:

```
enables I l a a' ≡ ∃ (p,e) ∈ delta I l. a' ∈ e
                          ∧ (tspace I a ∧ lspace I l ⟶ p(a))
```

We demonstrate the application of the Isabelle Insider framework in Section 5.1 on our running example of an Insider case study from the health care sector.

2.2 Attack Trees

Attack Trees [19] are a graphical tree-based design language for the stepwise investigation and quantification of attacks. They have been integrated as an extension to the Isabelle Insider framework [10,18]. In this Isabelle framework, base attacks are defined as a datatype and attack sequences as lists over those:

```
datatype baseattack = None | Goto "location"
                    | Perform "action" | Credential "location"
type_synonym attackseq = baseattack list
```

The following definition `attree` defines the nodes of an attack tree. The simplest case of an attack tree node is a base attack. Attacks can also be combined as the "and" of other attacks. The third element of type `attree` is a `baseattack` (usually a `Perform action`) that represents this attack, while the first element is an attack sequence and the second element is the attribute, simply a "string":

```
datatype attree = BaseAttack "baseattack" ("𝒩 (_)") |
                  AndAttack "attackseq" "string" "baseattack" ("_ ⊕∧(-) _")
```

The functions `get_attseq` and `get_attack` are corresponding projections on attack trees returning the entire attack sequence or the final base attack (the root), respectively.

The main construction concept for attack trees is *refinement* defined by an inductive predicate `refines_to` syntactically represented as the infix operator $\sqsubseteq$. There are rules `trans` and `refl` making the refinement a preorder; the rule `refineI` shows how attack vectors can be integrated into the refinement process. We will investigate this rule in detail when integrating with modelchecking in Section 4.2 because this is where modelchecking and attack tree refinement complement each other nicely.

The refinement of attack sequences allows the expansion of top level abstract attacks into longer sequences. Ultimately, we need to have a notion of when a sufficiently refined sequence of attacks is valid. This notion is provided by the final inductive predicate `is_and_attack_tree`. We will not focus on this here. For details, see [10] or the online formalisation [6]).

Intuitively, the process of refining corresponds to enlarging an attack tree as depicted in Figure 1.

2.3 Modelchecking

Modelchecking is often advertised as a 'push-button' technique in contrast to automated verification techniques, for example with Isabelle, where the user has to interact with the tool to verify properties. Thus it is in practice very successful mainly due to this full automation. The applications in the Isabelle Insider

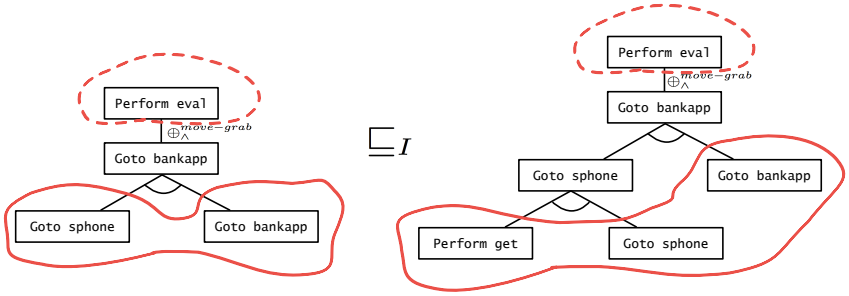

Fig. 1. Attack refinement for healthcare case study (see also Section 2.3).

framework that we construct are mostly performed by simple combinations of automatic proof procedures once the theorem and lemmas have been stated. The most well known problem of Modelchecking is the exponential growth of the number of states, the 'state explosion' common in most applications because of infinite data domains. Due to this restriction, models often oversimplify.

Another important advantage of modelchecking is the natural use of temporal logic to express system constraints, e.g., `M ⊢AG send -> AF ack` to express "on all paths in the model `M` it is the case that a `send` request is eventually followed by an acknowledgement `ack`".

Due to the expressiveness of HOL, Isabelle allows us to formalise within HOL the notion of Kripke structures, temporal logic, and formalise the semantics of modelchecking by directly encoding the fixpoint definitions for each of the CTL operators [7]. To realize this, a change of the state of the infrastructure needed to be incorporated into the Isabelle Insider framework. A relation on infrastructures is defined as an inductive predicate called `state_transition`. It introduces the syntactic infix notation `I →ᵢ I'` to denote that infrastructures `I` and `I'` are in this relation.

```
inductive state_transition ::
        [infrastructure, infrastructure] ⇒ bool ("_  →ᵢ _")
```

The definition of this inductive relation is given by a set of rules. To give an impression of this definition, we show here just the rule for the move action.

```
move: ⟦ G = graphI I; a @_G l; l ∈ nodes G; l' ∈ nodes G;
        a ∈ actors_graph(graphI I); enables I l (Actor a) move;
        I' = Infrastructure (move_graph_a a l l'
                             (graphI I))(delta I)(tspace I)(lspace I)
      ⟧ ⟹ I →ᵢ I'
```

3 Health Care Case Study in Isabelle Insider Framework

The case study we use as a running example in this paper is a simplified scenario from the context of the SUCCESS project for Security and Privacy of the IoT [2]. A central topic of this project for the pilot case study is to support security

and privacy when using cost effective methods based on the IoT for monitoring patients for the diagnosis of Alzheimer's disease. As a starting point for the design, analysis, and construction, we currently develop a case study of a small device for the analysis of blood samples that can be directly connected to a mobile phone. The analysis of this device can then be communicated by a dedicated app on the smart phone that sends the data to a server in the hospital.

In this simplified scenario, there are the patient and the carer within a room together with the smart phone.

We focus on the carer having access to the phone in order to support the patient in handling the special diagnosis device, the smart phone, and the app.

The insider threat scenario has a second banking app on the smart phone that needs the additional authentication of a "secret key": a small electronic device providing authentication codes for one time use as they are used by many banks for private online banking.

Assuming that the carer finds this device in the room of the patient, he can steal this necessary credential and use it to get onto the banking app. Thereby he can get money from the patient's account without consent.

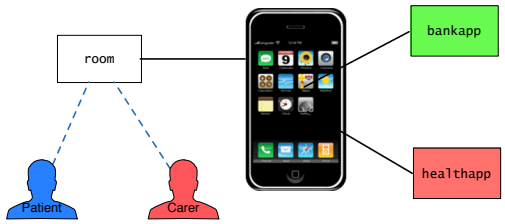

Fig. 2. Health care scenario: carer and patient in the room may use smartphone apps.

4 Combining Modelchecking and Attack Trees

We now show the interaction of the modelchecking and attack tree approaches by introducing the necessary extensions to the Isabelle Insider framework while highlighting the adaptations that manifest the combination. We use the health care case study introduced in the previous section to show how this combination enables the analysis of the Insider risk given by the carer.

4.1 Relation between State Transition and Attack Sequences

Modelchecking introduces the concept of state transition explicitly into the Isabelle Insider model. The relation $\rightarrow_i$ (see Section 2.3) provides a transition between different states of the infrastructure that can evolve into each other through changing actions taken by actors. By contrast, in the attack tree world, we have not explicitly introduced an effect on the infrastructure's state but we have equally investigated and refined attacks as sequences of actions eventually mapping those actions onto sequences of base attacks.

The main clue to combine modelchecking and attack tree analysis is intuitively described as using the Kripke models as the *models* for the attack tree analysis. More precisely, the sequences of attack steps that are eventually found through the process of refining an attack, need to be checked against sequences of transitions possible in the Kripke structure that consists of the graph of infrastructure's state changes.

Technically, this transformation needs a slight transformation between sequences of steps of the infrastructure's state changing relation $\rightarrow_i$ and that same relation but with the actions leading to the exact same state changes annotated at the transitions. Those annotations then naturally correspond to the paths that determine the way through the Kripke structure. They can be one-to-one translated into attack vectors.

Formally, we simply define a relation very similar to $\rightarrow_i$ but with an additional parameter added as a superscript after the arrow.

```
inductive state_step ::
        [infrastructure, action, infrastructure] ⇒ bool ("_  →^(-)  _")
```

For the definition of this inductive relation we show here again just the rule for the move action which is nearly identical to before just adding the action.

```
move: ⟦ G = graphI I; a @_G l; l ∈ nodes G; l' ∈ nodes G;
        a ∈ actors_graph(graphI I); enables I l (Actor a) move;
        I' = Infrastructure (move_graph_a a l l'
                             (graphI I))(delta I)(tspace I)(lspace I)
      ⟧ ⟹ I →^move I'
```

We define an iterator relation `state_step_list` over the `state_step` that enables collecting the action sequences over state transition paths.

```
inductive state_step_list ::
  [infrastructure, action list, infrastructure] ⇒ bool ("_  →^(-)  _")
where
  state_step_list_empty: I →^[] I |
  state_step_list_step : ⟦ I →^[a] I'; I' →^l I'' ⟧ ⟹ I →^a#l I''
```

Note, how in Isabelle overloading of the operator $\rightarrow^{(-)}$ can be neatly applied.

With this extended relation on states of an infrastructure we can now trace the modelchecking action sequences. Finally, a simple translation of attack sequences from the attack tree model to action sequences can simply be formalised by first defining a translation of base attacks to actions.

```
primrec transform :: baseattack ⇒ action
where
  transform_move:    transform (Goto l') = move |
  transform_get:     transform (Credential l') = get |
  transform_perform: transform (Perform a) = a
```

From this we define a function `transf` for transforming sequences of attacks.

```
primrec transf :: attackseq ⇒ action list
where
 transf_empty : transf [] = [] |
 transf_step:   transf (ba#l) = (transform ba)#(transf l)
```

4.2 Improving Attack Refinement

This relative simple adaptation of the modelchecking state transitions to action sequences paired with the transformation from attack traces has a simplifying as well as unifying effect: the attack tree approach necessitated the explicit definition of "attack vectors" that could be used to replace an abstract attack node by a sequence of attacks (see Section 2.2). This was manifested by the rule `refineI` which required a predefined list of attack vectors.

```
⟦ P ∈ attack_vectors; P I s l a;
   sublist_rep l a (get_attseq A) = (get_attseq A');
   get_attack A = get_attack A' ⟧ ⟹ A ⊑_I A'
```

An example of an attack vector that had to be replaced for `P` and provided as premise to the above rule is the example `UI_AV7` of *unintentional Insider attack vectors* [10].

```
⟦ enables I l a move; enables (add_credential I a s) l a get ⟧
⟹ UI_AV7 I s
   (get_attackseq ([Goto l, Perform get] ⊕_∧^move−intercept Credential l))
   (Credential l)
```

Previously, such attack vectors had to be defined as inductive rules in an axiomatic fashion. Now, the attack vectors can be *inferred* from the modelchecking process. The new rule in the attack refinement definition is `refineIMC`.

```
⟦ I →^l' I'; transf l = l';
   sublist_rep l a (get_attseq A) = (get_attseq A');
   get_attack A = get_attack A' ⟧ ⟹ A ⊑_I A'
```

An application can be seen in the following section.

5 Analysing Carer Attack

5.1 Health Care Case Study in Isabelle Insider Framework

We only model two identities, `Patient` and `Carer` representing a patient and his carer. We define the health care scenario in the locale `scenarioHealthcare`. The syntax `fixes` and `defines` are keywords of locales that we drop together with the types for clarity of the exposition from now on. The double quotes `''s''` represent strings in Isabelle/HOL. The global policy is 'no one except the patient can use the bank app':

```
fixes global_policy :: [infrastructure, identity] ⇒ bool
defines  global_policy I a ≡  a ≠ ''Patient'' ⟶
                              ¬(enables I bankapp (Actor a) eval)
```

The graph representing the infrastructure of the health care case study has the following locations: (0) smart phone, (1) room, (2) bank app, and (3) health app: In order to define the infrastructure, we first define the graph representing the scenario's locations and the positions of its actors. The actors patient and carer are both initially in room. The graph is given as a set of nodes of locations and the actors residing at certain locations are specified by a function associating lists of nodes with the locations.

```
ex_graph ≡  Lgraph
             {(room, sphone), (sphone, healthapp), (sphone, bankapp)}
             (λ x. if x = room then [''Patient'', ''Carer''] else [])
```

In the following definition of local policies for each node in the office scenario, we additionally include the parameter `G` for the graph. The predicate $@_G$ checks whether an actor is at a given location in the graph G.

```
local_policies G  ≡
 (λ x.  if x = room then {(λ y. True,{get, put, move}) }
    else (if x = sphone then
      {((λ y. has (y, ''PIN'')), {put,get,eval,move}), (λ y. True, {})}
          else (if x = healthapp then
                   {((λ y. (∃ n. (n @_G sphone) ∧ Actor n = y)),
                   {get,put,eval,move})}
                else (if x = bankapp then
                        {((λ y. (∃ n. (n @_G sphone) ∧ Actor n = y ∧
                        has (y, ''skey''))), {get,put,eval,move})}
                      else {})))
```

In this policy, any actor can move to the room and when in possession of the PIN can move onto the sphone and do all actions there. The following restrictions are placed on the two other locations.

`healthapp`: to `move` onto the `healthapp` and perform any action at this location, an actor must be at the position `sphone` already;
`bankapp`: to `move` onto the `bankapp` and perform any action at this location, an actor must be at the position `sphone` already and in possession of the `skey`.

The possession of credentials like PINs or the skey is assigned in the infrastructure as well as the roles that actors can have. We define this assignment as a predicate over actors being true for actors that have these credentials. For the health care scenario, the credentials express that the actors `Patient` and `Carer` possess the PIN for the sphone but `Patient` also has the `skey`.

```
ex_creds ≡ (λ x. if x = Actor ''Patient'' then
                       has (x,''PIN'') ∧ has (x, ''skey'')
                 else (if x = Actor ''Carer'' then
                       has (x, ''PIN'') else True))
```

The graph and credentials are put into the infrastructure `hc_scenario`.

```
hc_scenario ≡ Infrastructure
                ex_graph (local_policies ex_graph) ex_creds ex_locs
```

5.2 Modelchecking Supported Attack Tree Analysis

As a setup for the state analysis, we introduce the following definitions to denote changes to the infrastructure. A first step towards critical states is that the carer gets onto the smart phone. We first define the changed infrastructure graph.

```
ex_graph' ≡  Lgraph
             {(room, sphone), (sphone, healthapp), (sphone, bankapp)}
             (λ x. if x = room then [''Patient''] else
                 (λ x. if x = sphone then [''Carer''] else []))
```

The dangerous state has a graph in which the actor `Carer` is on the `bankapp`.

```
ex_graph'' ≡  Lgraph
              {(room, sphone), (sphone, healthapp), (sphone, bankapp)}
              (λ x. if x = room then [''Patient''] else
                 (λ x. if x = bankapp then [''Carer''] else []))
```

The critical state of the credentials is where the carer has the `skey` as well.

```
ex_creds' ≡ (λ x. if x = Actor ''Patient'' then
                       has (x,''PIN'') ∧ has (x, ''skey'')
                  else (if x = Actor ''Carer'' then
                        has (x, ''PIN'') ∧ has (x, ''skey'')
                        else True))
```

We use these changed state components to define a series of infrastructure states.

```
hc_scenario'  ≡ Infrastructure
                ex_graph (local_policies ex_graph) ex_creds' ex_locs
hc_scenario'' ≡ Infrastructure
                ex_graph'(local_policies ex_graph') ex_creds' ex_locs
hc_scenario'''≡ Infrastructure
                ex_graph''(local_policies ex_graph'') ex_creds' ex_locs
```

We next look at the abstract attack that we want to analyse before we see how the modelchecking setup supports the analysis.
The abstract attack is stated as `([Goto bankapp]` $\oplus_{\wedge}^{move-grab}$ `Perform eval)`. The following refinement encodes a logical explanation of how this attack can happen by the carer taking the `skey`, getting on the phone, on the `bankapp` and then evaluating.

```
([Goto bankapp] ⊕_∧^(move−grab) Perform eval)
⊑_hc_scenario
([Perform get, Goto sphone, Goto bankapp)] ⊕_∧^(move−grab) Perform eval)
```

This refinement is proved by applying the rule `refineI` (see Section 4.2). In fact, this attack could be *found* by applying `refineI` and using interactive proof with the modelchecking extension of the Isabelle Insider framework to instantiate the higher order parameter `?l` in the following resulting subgoal.

$$\texttt{hc\_scenario} \rightarrow^{\texttt{transf(?l)}} \texttt{hc\_scenario'''}$$

This proof results in instantiating the variable `?l` to the required attack sequence `[Perform get, Goto sphone, Goto bankapp]`.

So far, we have used the combination of a slightly adapted notion of the state transition from the modelchecking approach to build a model for attack refinement of attack trees. We can further use the correspondence between modelchecking and attack trees to find attacks. To properly employ modelchecking, we first define the Kripke structure for the health case scenario representing the state graph of all infrastructure states reachable from the initial state.

$$\texttt{hc\_states} \equiv \texttt{\{ I. hc\_scenario} \rightarrow_i^* \texttt{ I \}}$$
$$\texttt{hc\_Kripke} \equiv \texttt{Kripke hc\_states \{hc\_scenario\}}$$

Following the modelchecking approach embedded into the Isabelle Insider framework [7], we may use branching time logic CTL to express that the global policy (see Section 5.1) holds for all paths globally.

$$\texttt{hc\_Kripke} \vdash \mathbf{AG}\ \texttt{\{x. global\_policy x ''Carer''\}}$$

Trying to prove this must fail. However, using instead the idea of invalidation [12] we can prove the negated global policy.

$$\texttt{hc\_Kripke} \vdash \mathbf{EF}\ \texttt{\{x.} \neg\ \texttt{global\_policy x ''Carer''\}}$$

The interactive proof of this `EF` property means proving the theorem

$$\texttt{hc\_Kripke} \vdash \mathbf{EF}\ \texttt{\{x. enables x bankapp (Actor ''Carer'') eval\}}$$

This results in establishing a trace `l` that goes from the initial state `hc_scenario` to a state `I` such that `enables I bankapp (Actor ''Carer'') eval`. This `I` is for example `hc_scenario'''` and the action path `get, move, move` is a side product of this proof. Together with the states on this path the `transf` function delivers the required attack path `[Perform get, Goto sphone, Goto bankapp]`.

6 Conclusions

Summarizing, we have considered the benefits of relating earlier extensions to the Isabelle Insider framework to modelchecking and to attack trees and illustrated the benefits on a health care case study of an Insider attack.

Clearly relevant to this work are the underlying framework and its extensions [14,10,9,7] but also the related experiments with the invalidation approach for Insider threat analysis using classic implementation techniques like static

analysis and implementation in Java [18] or probabilistic modeling and analysis [1].

We believe that the combination of modelchecking and attack trees is novel at least in the way we tie these concepts up at the foundational level. Considering the simplicity of this pragmatically driven approach and the relative ease with which we arrived at convincing results, it seems a fruitful prospect to further explore this combination. Beyond the mere finding of attack vectors in proofs, the expressivity of Higher Order Logic will allow developing meta-theory that in turn can be used for the transfer between modelchecking and attack tree analysis.

There are excellent foundations available for attack trees based on graph theory [15]. They provide a very good understanding of the formalism, various extensions (like attack-defense trees [16] and differentiations of the operators (like sequential conjunction (SAND) versus parallel conjunction [4]) and are amply documented in the literature. These theories for attack trees provide a thorough foundation for the formalism and its semantics. The main problem that adds complexity to the semantical models is the abstractness of the descriptions in the nodes. This leads to a variety of approaches to the semantics, e.g. propositional semantics, multiset semantics, and equational semantics for ADtrees [16]. The theoretical foundations allow comparison of different semantics, and provide a theoretical framework to develop evaluation algorithms for the quantification of attacks.

Surprisingly, the use of an automated proof assistant, like Isabelle, has not been considered despite its potential of providing a theory and mechanised analysis of attacks simultaneously. The essential attack tree mechanism of tree refinement is relatively simple. The complexity in the theories available in the literature is caused by the attempt to incorporate semantics to the attack nodes and relate the trees to actual scenarios. This is why we consider the formalisation of a foundation of attack trees in the interactive prover Isabelle since it supports logical modeling and definitions of datatypes very akin to algebraic specification but directly supported by semi-automated analysis and proof tools. There have already been attempts at formalising attack trees for specific application domains in the interactive theorem prover Isabelle [10] (for IoT Insider attacks). They are also based on the Isabelle Insider framework but support only the use of axiomatized "attack vectors" derived from real Insider attacks. It is necessary to assume these attack vectors to provide the semantics of attack tree refinement. Clearly, in state based systems, attacks correspond to paths of attack steps. Hence, it is quite obvious to use a modelchecking approach to analyse attack trees. In fact, implementations like the ADTool [17] use modelchecking based on the guarded command language *gal* to analyse scenarios expressed as graphs. Surprisingly, again, it has not been considered to use a logical framework powerful enough to emulate modelchecking to augment this natural approach to modeling and analysing attack trees. The modelchecking approach brings the additional advantage of exploring and thus finding possibilities of attack refinements necessary for the attack tree development. If embedded within an inter-

active theorem prover, the integration of the formalism can be applied to case studies and meta-theory can be proved with the additional support and safety guarantees of a proof assistant.

The presented foundation of attack trees in Isabelle is consistent with the existing foundations [15,16,4] but instead of providing an on paper mathematical foundation it provides a direct formalisation in Higher Order Logic in the proof assistant. This enables the application of the resulting framework to case studies and does not necessitate a separate implementation of the mathematical foundation in a dedicated tool. Clearly, the Isabelle framework is less efficient and the application to case studies requires user interaction. However, the formalisation in Isabelle supports not only the application of the formalised theory but furthermore the consistent development of meta-theorems. In addition, dedicated proof automation by additional proof of supporting lemmas is straightforward and even code generation is possible for executable parts of the formalisation.

Again in comparison to the existing foundation [15,16,4], the presented attack tree framework is restricted. For example, it does not yet support disjunctive attacks nor attack-defense trees, i.e., the integration of defenses within the attack tree. We are convinced that this is a straightforward future development and will be provided in due course.

Acknowledgement

Part of the research leading to these results has received funding from the European Union (CHIST-ERA 2015) under grant agreement no. 102112 (SUCCESS). 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. T. Chen, F. Kammüller, I. Nemli, and C. W. Probst. A probabilistic analysis framework for malicious insider threats. *Human Aspects of Information Security, Privacy, and Trust, HAS 2015, HCII 2015.* LNCS **9190**: 178–189. Springer, 2015.
2. CHIST-ERA. Success: Secure accessibility for the internet of things, 2016. http://www.chistera.eu/projects/success.
3. C. G. Hempel and P. Oppenheim. Studies in the logic of explanation. *Philosophy of Science*, 15:135–175, April 1948.
4. R. Jhawar, B. Kordy, S. Mauw, S. Radomirovic, R. Trujillo-Rasua. Attack Trees with Sequential Conjunction. *30th IFIP TC 11 International Conference on ICT Systems Security and Privacy Protection (IFIP SEC'15)*, IFIP Advances in Information and Communication Technology, **455**:339–353, Springer, 2015.
5. L. Henrio, F. Kammüller, and M. Rivera. An asynchronous distributed component model and its semantics. In *Formal Methods for Components and Objects*, volume 5751 of *LNCS*, pages 159–179. Springer, 2009.
6. F. Kammüller. Isabelle insider framework with attack trees and modelchecking, 2016. Available from `https://www.dropbox.com/sh/rx8d09pf31cv8bd/AAALKtaP8HMX642fi04Og4NLa?dl=0`.

7. F. Kammüller. Isabelle modelchecking for insider threats. *Data Privacy Management, DPM'16, co-located with ESORICS'16*, LNCS. Springer, 2016.
8. F. Kammüller and M. Kerber. Investigating airplane safety and security against insider threats using logical modeling. In *IEEE Security and Privacy Workshops, Workshop on Research in Insider Threats, WRIT'16*. IEEE, 2016.
9. F. Kammüller, M. Kerber, and C. W. Probst. Towards formal analysis of insider threats for auctions. *8th ACM CCS International Workshop on Managing Insider Security Threats, MIST'16*. ACM, 2016.
10. F. Kammüller, J. R. C. Nurse, and C. W. Probst. Attack tree analysis for insider threats on the iot using isabelle. *Human Aspects of Information Security, Privacy, and Trust - HAS 2015, HCII 2016, Toronto*, LNCS. Springer, 2016.
11. F. Kammüller and L. C. Paulson. A formal proof of sylow's theorem. *Journal of Automated Reasoning*, 23(3):235–264, 1999.
12. F. Kammüller and C. W. Probst. Invalidating policies using structural information. *IEEE Security and Privacy Workshops, Workshop on Research in Insider Threats, WRIT'13*. IEEE, 2013.
13. F. Kammüller and C. W. Probst. Combining generated data models with formal invalidation for insider threat analysis. *IEEE Security and Privacy Workshops, Workshop on Research in Insider Threats, WRIT'14*. IEEE, 2014.
14. F. Kammüller and C. W. Probst. Modeling and verification of insider threats using logical analysis. *IEEE Systems Journal, Special issue on Insider Threats to Information Security, Digital Espionage, and Counter Intelligence*, 2016.
15. B. Kordy, L. Pitre-Cambacds, P. Schweitzer. Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review* **13-14**: 1–38, 2014.
16. B. Kordy, S. Mauw, S. Radomirovic, P. Schweitzer. Attack-defense trees. *Journal of Logic and Computation*, **24**(1): 55–87. Oxford Journals, 2014.
17. B. Kordy, P. Kordy, S. Mauw, P. Schweitzer, ADTool: Security Analysis with Attack-Defense Trees (Extended Version). *CoRR* **abs/1305.6829**, `http://arxiv.org/abs/1305.6829`, 2013.
18. C. W. Probst, F. Kammüller, and R. R. Hansen. Formal modelling and analysis of socio-technical systems. *Semantics, Logics, and Calculi, (Nielsens' Festschrift)*. LNCS **9650**, Springer 2016.
19. B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2004.