Transforming graphical system models to graphical attack models

Marieta Georgieva Ivanova[1], Christian W. Probst[1],
René Rydhof Hansen[2], and Florian Kammüller[3]

[1] Technical University of Denmark
{mgiv, cwpr}@dtu.dk
[2] Aalborg University, Denmark
rrh@cs.aau.dk
[3] Middlesex University, UK
f.kammueller@mdx.ac.uk

Abstract. Manually identifying possible attacks on an organisation is a complex undertaking; many different factors must be considered, and the resulting attack scenarios can be complex and hard to maintain as the organisation changes. System models provide a systematic representation of organisations that helps in structuring attack identification and can integrate physical, virtual, and social components. These models form a solid basis for guiding the manual identification of attack scenarios. Their main benefit, however, is in the analytic generation of attacks. In this work we present a systematic approach to transforming graphical system models to graphical attack models in the form of attack trees. Based on an asset in the model, our transformations result in an attack tree that represents attacks by all possible actors in the model, after which the actor in question has obtained the asset.

1 Introduction

Organisations face a constant stream of attacks on their IT-infrastructure. Many of these attacks and the ways to prevent them are well understood. Traditional and well-established risk assessment methods can often identify these potential threats, but due to a technical focus, these approaches often abstract away the internal structure of an organisation and ignore human factors when modelling and assessing attacks. However, an increasing number of attacks do involve attack steps such as social engineering.

Attack trees [1,2] are a loosely defined, yet (or maybe therefore) widely used approach for documenting possible attacks in risk assessment [3]; they can describe attack goals and different ways of achieving these goals by means of the individual steps in an attack. The goal of the defender is then to inhibit one or more of the attack steps, thereby prohibiting the overall attack, or at least making it more difficult or expensive. While attacks trees for purely technical attacks may be constructed by automated means [4,5], for example by scanning networks

and identifying software versions, this is currently not possible for attacks exploiting the human factors. Actually, only few, if any, approaches to systematic risk assessment take such "human factor"-based attacks into consideration. The goal of the TRE$_S$PASS project [6] is to close this gap by developing models 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 assessment and ranking of attacks based on the expected impact.

In this work we present the fundamental approach to systematically transform graphical system models to graphical attack models in the form of attack trees. Since the transformation considers all relevant system components, the resulting attacks may include elements of human behaviour. These attacks can then be used as input to a traditional risk assessment process and thereby extend and support the brainstorming results. Our approach is applicable to a class of recent system models such as ExASyM [7] and Portunes [8], which have been used to model and analyse organisations for possible attacks [9]. These models contain both the physical infrastructure and information on actors, access rights, and policies; consequently, analysis of such models can include, for example, social engineering in the identified attacks.

The benefit of converting system models to attack models is a conceptually new view on qualitative security properties. The system model represents spatial connections on the different layers of an organisation, thus blurring potential attacks exploiting items not connected in the model, or not connected in the mental image of the modeller. Attack models represent connections between elements and actions that can be exploited to perform an attack.

Our transformations are independent of the underlying model. While we present them in the setting of the TRE$_S$PASS model, the general approach can be applied to any graphical system model. The transformations described in this work can be used as the core technique for policy invalidation [10, 11], where policies describe both access control to locations and data, as well as system-wide policies such as admissible actions and actor behaviour. We have implemented the transformations presented in this work in an attack tree generator for TRE$_S$PASS models. The example shown in Figure 9 has been generated with this tool.

The rest of this article is structured as follows. The next section gives an overview of graphical models for systems and attacks, followed by a description of the transformations for simple models in Section 3. These simple models do not consider mobility of data or other actors than the attacker. Mobility of data through processes is added in Section 4. Finally, Section 5 concludes the paper and discusses future work.

2 Graphical System Models and Attack Models

We start by introducing the main concepts in the system model and the attack models we consider. System models includes representations of both the physical

and the digital infrastructure of an organisation. This is similar to approaches such as ExASyM [7] and Portunes [8], which represent relevant elements as nodes in a graph, that form the natural basis for the application of our techniques. However, for the current work, we do not require a particular kind of representation: the only requirement is that the core concepts discussed later in this section can be extracted from the underlying model. Similarly, attack models represent possible attacks on the modelled organisation. For the approach in this paper, we essentially only require that attack goals can be divided into sub-goals that can be combined either *conjunctively* (must all be completed) or *disjunctively* (only one sub-goal need to be completed). This is very similar to attack trees [1, 2], and just as for these it would be interesting to allow more complex combinations at a later point.

2.1 Graphical System Models

We consider *nodes* as the central element in our graphical model of an organisation. We differentiate between nodes representing

- **Locations** in the organisation, for example, rooms, access control points, network components, computers, etc. Nodes representing locations that are physically or logically connected in the organisation, are linked by directed edges in the graph.
- **Actors** in the modelled organisation.
- **Processes** modelling information sharing or policies.
- **Items** modelling tangible assets in the modelled organisation, for example, access cards, harddrives, etc.

Additionally, nodes can store *items* and *data*; in contrast to items, that are represented by nodes, data is represented by an (abstract) name and includes, *e.g.*, pins, passwords, and other intangible assets. All elements in the model provide a unique identifier that can be used to refer to the element and to obtain, for example, information on its concrete type, model, or other relevant properties.

A location in the modelled organisation may belong to several *domains*, *e.g.*, it can be (physically) part of the building and also be present (virtually) on the network. Nodes in the model can also belong to different domains, which limit the operations that can be performed on a node and limiting where processes can move; human actors, for example, are restricted to nodes in the physical domain, and computer processes are restricted to nodes in the virtual domain.

Assets are used for modelling any kind of item or data that is relevant in the modelled organisation. In addition, assets can be annotated with extra information, *e.g.*, a probability representing how likely it is to lose a particular piece of data.

Nodes that represent processes or actors can *move* around in the model, *i.e.*, be associated to changing locations; actors are allowed to store both items and data, while processes can only store data. Assets stored at either of these nodes move around with the node.

To represent a wide variety of processes and the possible behaviour of actors, we assume that a number of simple **actions** can be performed on a target, which can be any location in the model, including physical locations or actors.

To constrain mobility of processes and actors, as well as to constrain actions, we assume a policy mechanism in the model, consisting of

- **Policies** that regulate access to locations and assets. Policies consist of required credentials and enabled actions, representing what an actor needs to provide in order to enable the actions in a policy, and what actions are enabled if an actor provides the required credentials, respectively.
- **Credentials** are data, items, or an identity that the actor or process performing an action needs to have, or predicates.

Predicates as credentials express that the actor must possess a certain attribute. In the example shown in Figure 1, an actor must be trusted by Alice in order to be allowed to move to the location *Door*. We also assume policies to support variables to relate credentials to each other, or to restrict actions based on the credentials provided. In the example shown in Figure 1, the policy at the ATM requires the actor to present a card with a pin X and the matching pin.

As stated above, both the ExASyM [7] and Portunes [8] modelling languages fulfil the above requirements for using our approach, as does any Klaim-like models [12] in general. While Klaim models process mobility by processes moving from node to node, we request processes to reside in special nodes that move around with the process. We choose this abstraction to make the modelling of (movement of) actors and assets carried by actors more intuitive and natural; mapping "standard" Klaim-like models to this abstraction is straightforward. In Figure 1, for example, the node representing the actor Alice has a pin code and a card. The card in turn contains information about the owner and the pin code for the card.

In the work described here, we only consider the pure transformation of graphical system models to graphical attack models. An essential next step in risk assessment is to valuate the risk and impact of an attack, for example, by annotating the attack model with metrics and performing analyses on them [13]. This mapping can be achieved by associating the elements' identifiers with relevant metrics. These metrics can represent any quantitative knowledge about components, for example, likelihood, time, price, impact, or probability distributions. The latter could describe behaviour of actors or timing distributions. For the transformation described in this article these metrics are irrelevant, but they can be evaluated on the generated attack trees.

Containment. Items as described above are an important concept in our abstract model, since they can represent containment. Containment represents for example the fact that a workstation contains a harddrive that contains a file. In the model underlying our transformations we would represent the workstation as an item with a location; this location in turn would contain an item representing the harddrive; this item's location would contain data representing the (intangible) file.

We interpret containment as being transitive: if item a contains item b, and item b contains the data d, then we say a contains d transitively, and b contains d directly.

2.2 Graphical Attack Model

Attack trees [1, 2] are widely used by various security analysis techniques; they support an easily accessible tree-like structure that can be visualised and understood by non-experts. At the same time, they can be subjected to formal analysis and structured treatment due to their tree-structure. Even though standard attack trees represent sub-goals that must be completed in a specific sequence, they have a hierarchical structure: the root node represents the attacker's goal, which is further refined by defining sub-goals. As mentioned above, the sub-goals can be represented as sub-trees in the overall attack tree, where sub-trees, *i.e.*, sub-goals, are combined conjunctively or disjunctively.

We do not require any further properties for the target of our transformations. In principle the transformation could embed additional information into the attack tree; for example, we currently assume an implicit left to right order in sub-goals of conjunctive nodes.

2.3 Running Example

The running example in this paper is based on a case study in the TRE_{S}PASS project [6] based on an actor Alice, who receives some kind of service, *e.g.*, care-taking, provided by an actor Charlie. Charlie's employer has a company policy that forbids him to accept money from Alice. Figure 1 shows a graphical representation of the example scenario, consisting of Alice's home, a bank with an ATM, and a bank computer. Alice owns a card and a concomitant pin code to obtain money from an ATM, and a password to initiate transfers from her workstation via the bank computer. Some of the nodes are labelled with policies in dashed boxes; for example the money at the ATM requires a card with a pin code, as well as that very pin code in order to obtain money (modelled as input).

Figure 1 shows a graphical representation of the model of our running example. The locations, represented by small rectangles, are connected through directed edges. Actors are represented as rectangles with a location, *e.g.*, Alice is at home and Charlie is in the city. Both actor nodes and location nodes can contain data and items represented as circles. In our example, Alice has a card that contains a pin code and Alice also has (knows) the pin code for her card. Actor nodes can also represent processes running on the corresponding locations. The processes at the workstation and the bank computer represent the required functionality for transferring money; they initiate transfers from Alice's home (P_{WS}), and check credentials for transfers (P_C).

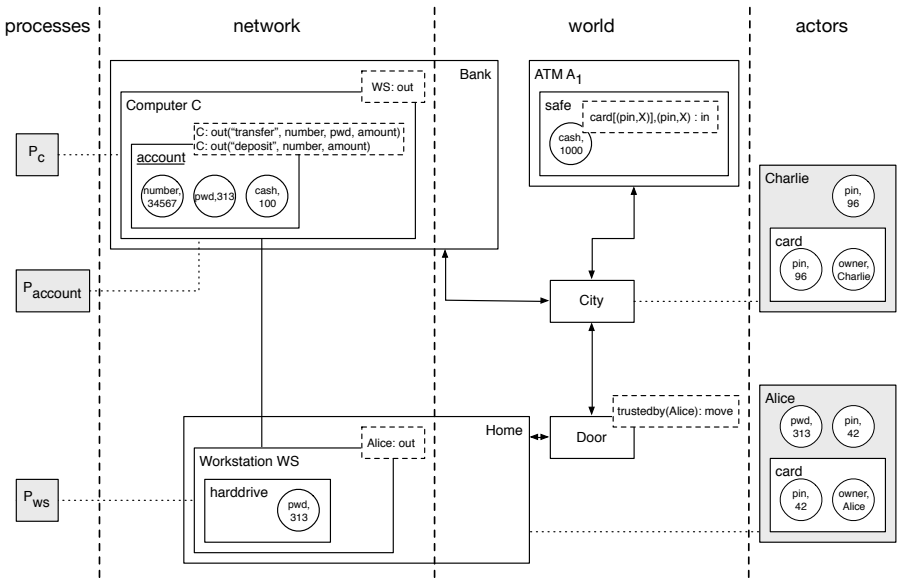

Fig. 1. Graphical representation of the example system. The white rectangles represent locations or items, the gray rectangles represent processes and actors; actors contain the items or data owned by the actor. The round nodes represent data. Solid lines represent the physical connections between locations, and dotted lines represent the present location of actors and processes. The dashed rectangles in the upper right part of some nodes represent the policies assigned to these nodes.

3 Transforming Models without Asset Mobility

The class of attacks we generate from graphical system models address attackers trying to reach a certain location or to obtain an asset. We mainly deal with confidentiality and integrity properties. We are currently working on extending this class to include attacks that aim at, *e.g.*, starting a process as part of a distributed denial-of-service attack. We expect to be able to generate these attacks with similar transformations. In this section we consider assets in the modelled organisation to be immobile. This restriction, which will be lifted in the next section, simplifies the first presentation of transformations.

Attack generation assumes an asset in the system, which an attacker should not be able to obtain. For every possible actor in the system, the goal of the transformation is then to generate an attack that results in the actor having obtained this asset. The overall transformation is a generalised version of policy invalidation [10, 11]:

1. Starting from the goal asset and the attacking actor,
2. the transformation identifies all paths to the asset,
3. and for every path, identifies the credentials that the actor is lacking;

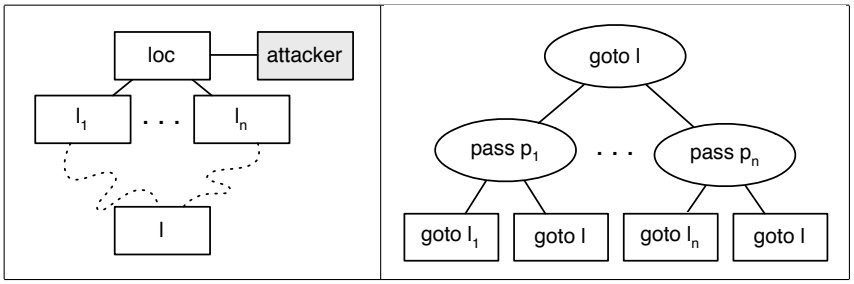

Fig. 2. Transforming a location. Any credential c_i that the attacker is lacking is obtained before performing action a at the location *loc*.

4. for each missing credential, a new transformation is started recursively;
5. after obtaining all necessary credentials, the actor can reach the location of the goal asset, and perform an action to obtain it.

In the following, we present for each of the model elements discussed in the previous section, how they are transformed into an attack representation. For each transformation we show the part of the system model that triggers the translation as well as the generated part of the attack model. For the system models we use the same graphical representation as shown in Section 2.3 and Figure 1. For attack models we use a special notation that represents parts of the attack as circles, and invocations of the transformation as rectangles.

3.1 Locations

A location is transformed into a disjunction of all possible paths from the locations already reached by the attacker to the location in question. Whenever traversing a path requires new credentials due to some policy, we recursively invoke the attack transformation, which ensures that the attacker obtains the necessary credentials to pass the path.

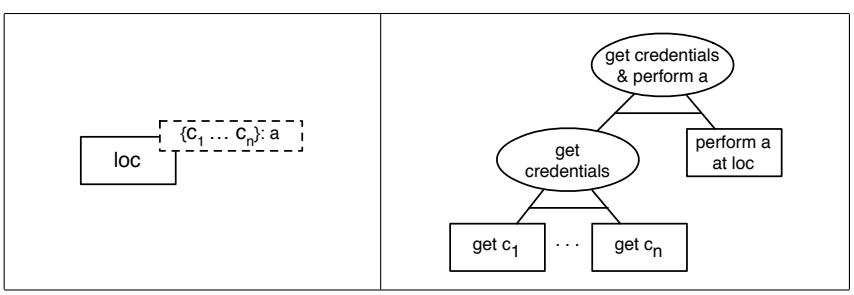

Fig. 3. Transforming a policy. If the attacker lacks any credential to perform action a at the location *loc*, the transformation creates an attack that obtains that credential.

The transformation pattern is shown in Figure 2. For every possible path we first generate one step to the first node of the path, followed by a recursive invocation of the transformation for going to the target location.

3.2 Policies

If the transformation at any point needs to create an action that is prohibited by a policy, for which the attacker does not have all credentials, a new transformation is started to obtain this credential, resulting in a new attack representation. The transformation pattern is shown in Figure 3.

As mentioned above, many system models support predicates as credentials, for example, to express that the actor must possess a certain attribute. In the example shown in Figure 1, an actor must be trusted by Alice in order to be allowed to move to the location *Door*. Often, such a predicate is not a credential that can be obtained, as for trust. In this case, the transformation generates a social engineering action to "obtain" the predicate in question.

The variables in policies can be factored out before performing this transformation by identifying all sets of assets that fullfil a policy. For the example shown in Figure 1 and the location ATM, the possible sets of assets are the card and the pin at Alice or at Charlie.

In the following we assume that the transformation generates all necessary steps for obtaining assets before performing the transformations described. In the resulting attack representation, the root node of the attack representation for obtaining the necessary credentials will be to the left of the root node for performing the following actions, expressing an ordering as described above.

3.3 Data

Data represents intangible assets, such as passwords or pins. For obtaining data, a conjunction is generated where the first element is to reach the location of the data. Once the attacker has reached a location that contains the goal data, an action in the attack representation will be generated that depends on the kind of location that contains the data:

- If the data is contained in a location, then a simple in action will be generated; or
- If the data is contained at an actor, then a social engineering action will be generated.

If the goal data is contained in an item i, the transformation generates the conjunction of several actions:

- Obtain the item and then obtain the data from the item; or
- Obtain the data from the item directly.

The difference between the two options is that the first option represents the case that the attacker obtains the containing item itself and then obtains the

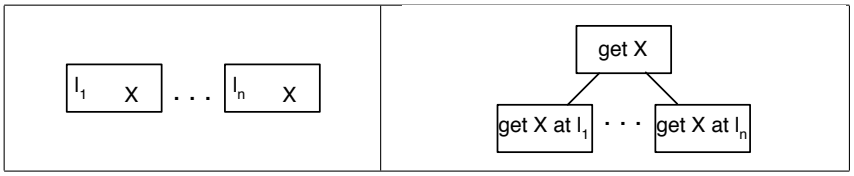

Fig. 4. Items and data may be available from different locations. For each of these locations, the transformation generates a separate attack path to obtain the asset. The transformation will generate attacks to obtain all necessary credentials, and then input the asset.

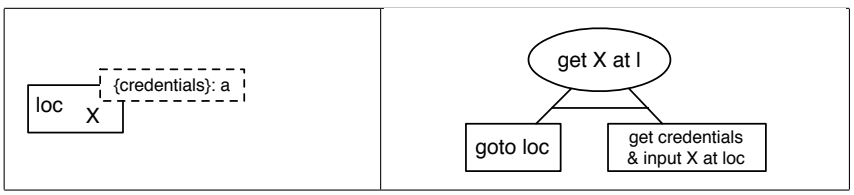

Fig. 5. To obtain an asset from a location, the transformation generates the necessary attack to go to the asset's location, then obtains the credentials, and finally performs the necessary **in** action.

data, while the second option represents the case that the attacker removes the data or item in place.

For the example of the workstation mentioned before this would mean that the attacker either steals the harddrive containing the file, or that he extracts the file from the harddrive.

3.4 Items

Items represent tangible assets, such as the aforementioned workstation, harddisk, or an access card. Just as for data, we generate a conjunction that first contains a node that represents reaching the location of an item. Once the attacker has reached a location that contains the goal item, an action in the attack representation will be generated that depends on the kind of location that contains the item:

- If the item is contained in a location, then a simple in action will be generated;
- If the item is contained at an actor, then a disjunction of a social engineering action or an in action will be generated, where the latter represents an attempt of stealing the item.

If the goal item is contained in another item, the transformation generates the conjunction of several actions:

- Obtain the item and then obtain the goal item from the item; or

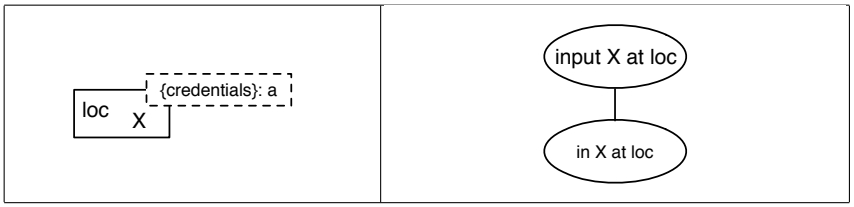

Fig. 6. To obtain an asset that is directly contained at a location, the transformation simply generates an **in** action. Note that the necessary credentials have been obtained before invoking this transformation.

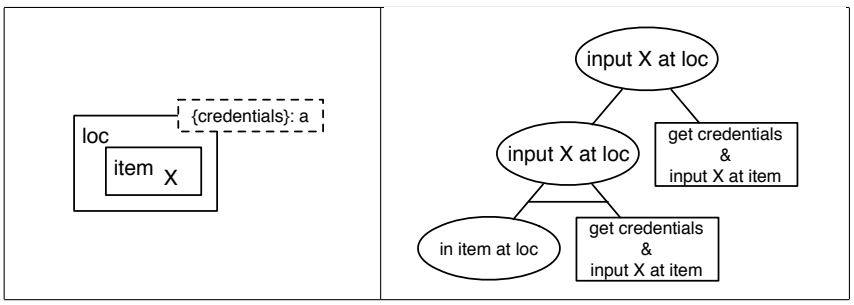

Fig. 7. To obtain an asset that is transitively contained at a location, the transformation first obtains the item containing the asset and then recursively invokes the transformation.

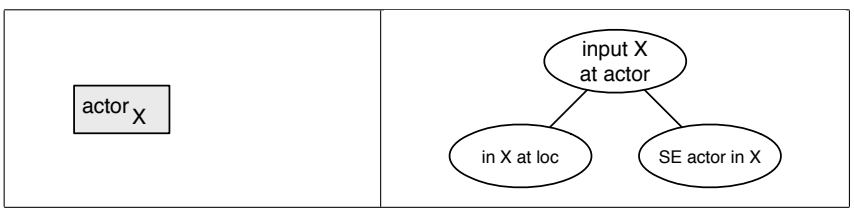

Fig. 8. Obtaining an asset from an actor is almost the same as for locations; the only difference is that assets can be obtained by social engineering. The transformation generates a special social engineering action, which is not further defined. Refining this action depends on the context of the action such as, *e.g.*, the involved actors; this is left to later phases that consume the generated attack.

– Obtain the goal item from the item directly.

The difference between the two options in the generated disjunctions is that the first option represents the case that the attacker obtains the containing item itself, while the second option represents the case that the attacker removes the data or item in place.

For the example of the workstation mentioned before this would mean that the attacker either steals the workstation containing the harddrive, or that he extracts the harddrive from the workstation.

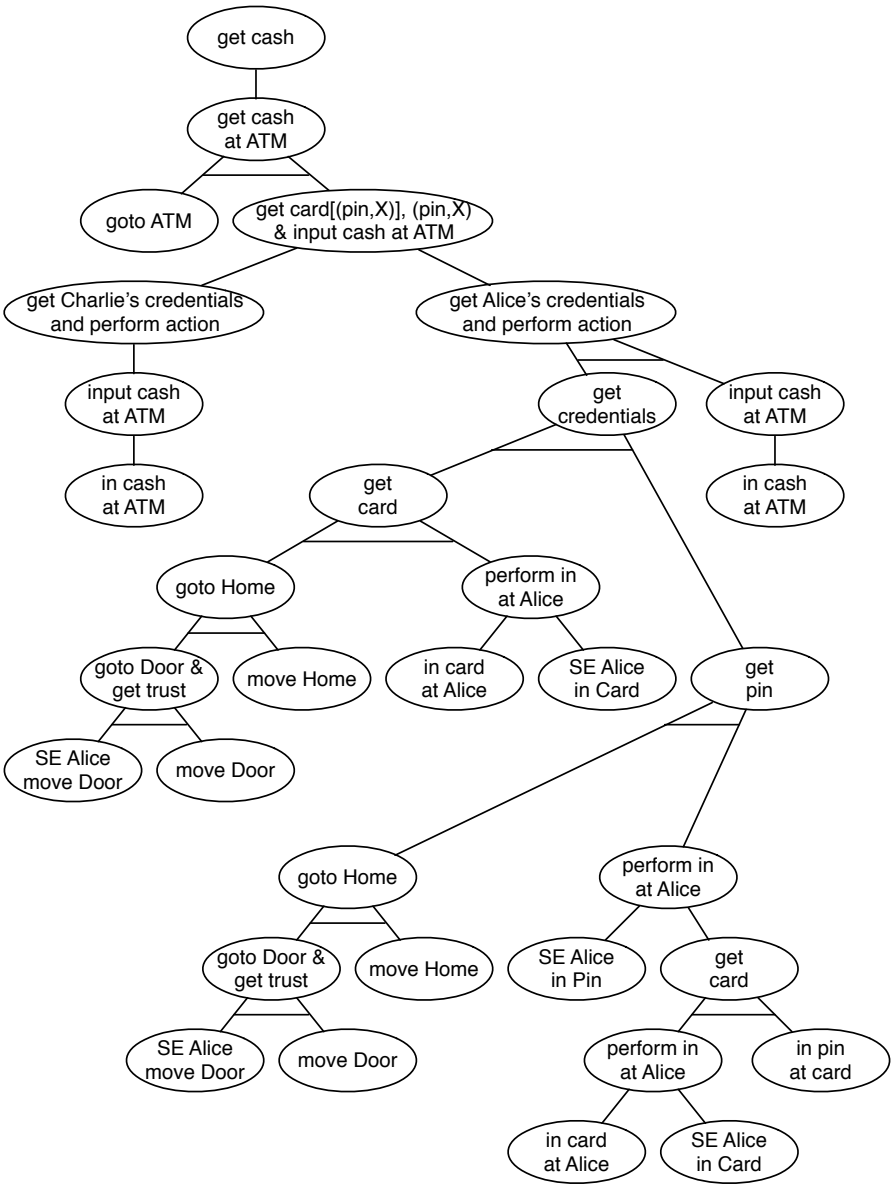

Fig. 9. Result of transforming the example from Figure 1 using *cash* as the goal asset and Charlie as an attacker.

3.5 Triggering the Transformation

In general the transformation will be triggered by a certain asset being off-limit for an attacker. The transformation iterates over a specified set of actors available in the system model, and generates for each of these actors all possible attacks for how they can obtain the asset. The triggering transformation for an asset X is get X. While transforming the system model into an attack model, the transformation keeps track of the attacker, the location reached, and the assets obtained. The attacker may already possess assets before starting the transformation; this is specified in the system model.

3.6 Transforming the Example

We will now sketch the transformation of the example system discussed in Section 2.3 and shown in Figure 1. We assume that the goal asset of the attacker is *cash*, which is available from locations *ATM A1* and *Computer*. We will only described data mobility in the next section, so for now we concentrate on the "physical" cash available at the ATM location.

As discussed above, the transformation considers all possible actors and starts with the get *cash* action, which in turn will result in a get *cash* at *ATM* transformation (Figure 4). This results in a conjunction of going to the ATM, getting the credentials, and inputting the asset at that location, since the goal asset is directly contained in the ATM.

The credentials at the *cash* asset require a card with a matching pin. In the example system, both Charlie and Alice own matching assets, so the transformation generates two possible attacks, one using Charlie's card, another using Alice's card. Clearly, the first transformation result does not necessarily represent an attack; generating such unwanted artefacts can either prohibited by restricting permissible actors in the policy,[4] or it can be dealt with in later phases that work on the generated attacks.

For the first possible attack, Charlie would use his own card and pin; this does not require further credentials. For the second possible attack, Charlie needs to obtain the pin and the card from Alice. Alice's location is *Home*, and to pass the path to this location, Charlie must fullfil the predicate *trustedby(Alice)*. This results in an action *social engineer Alice move Door*, which could in a later phase, for example, be translated into a forceful entrance or pretending to be somebody who Alice trusts or is likely to let in her home. Once the location *Home* has been reached, Charlie has several options for obtaining the card and the pin:

- Social Engineer Alice to give him the card and the pin;
- Input card from Alice (stealing); and
- Input the pin from the card (skimming).

[4] In this case, the owner of the card would not be allowed to be the actor performing the action.

The generated attack takes account for all combinations hereof; some parts of the tree can be pruned or simplified in a later phase similar to [4]. Once the card and the pin have been obtained, Charlie moves to the location *ATM* and inputs the asset *cash*.

The resulting attack model is shown in Figure 9. Not surprisingly, the transformation result contains identical sub-trees due to identical assets to obtain or identical patterns being transformed. Similar to the actions for obtaining items, these could be simplified by a followup pass.

4 Adding Data Mobility

So far we have assumed assets to be at static locations. This assumption simplifies both the transformations for attack generation and the structure of the generated attacks; instead of having to consider all the locations that an asset can reach by means of actors or processes, we only have to consider the locations where data is available in the model. We now discuss how to loosen this restriction.

In Section 3.3 and Section 3.4 the transformations described assume that the data is available from a number of locations in the model, either directly or transitively. The main transformation starting the generation of sub-attacks is shown in Figure 4. When adding data mobility, we are interested in which *other* locations the assets are able to reach, either by means of processes (for virtual assets) or by means of actors (for real-world assets).

The transformation for data mobility works reverse to the transformations we have presented in the previous section. Before being able to generate an attack, we need to perform three steps:

1. Identify who is able to move the asset;
2. Identify how to trigger the movement; and
3. Identify which locations the asset can reach.

The result of these steps is an attack that triggers the movement, and a set of locations that the asset can reach; these locations can then be used as input to the transformation shown in Figure 4.

The main task lies in identifying who can trigger the movement and how. Beyond these steps, adding data mobility does not add to the transformation, but to the complexity of the generated attack model.

5 Conclusion

In this article we have presented a systematic approach for transforming graphic system models into graphical attack models. Graphical models in general have the advantage of easing understanding by non-technical personelle. This is a significant advantage especially when communicating the risk of attacks on an organisation. While the techniques discussed in this work especially target IT security attacks, the techniques are applicable to any kind of attacks and risks.

Especially the support for social engineering attacks, though only at a very abstract level, enables handling of a wide class of attacks involving physical, virtual, and social layers of organisations. As recent events have shown, this class of attacks will become ever more important.

Our techniques help identifying and communicating attacks faced by organisation by enhancing traditional risk assessment methods that often abstract away the internal structure of an organisation and ignore human factors when modelling and assessing attacks. The attacks we identify consider all relevant system components, including elements of human behaviour, and can be used as input to a traditional risk assessment process.

Our approach is generally applicable to graphical system models and graphical attack models; examples for instances of such models include system models, *e.g.*, ExASyM [7] and Portunes [8], and attack models such as attack trees and attack-defence trees [1, 2].

As discussed in Section 3, we are currently working on extending the class of generated attacks to include attacks that aim at, *e.g.*, starting a process as part of a distributed denial-of-service attack. Another extension of our approach aims at considering the environment in which the system under attack is used. This environment influences, *e.g.*, the value of data or assets, either for the organisation or the attacker. Finally, we are exploring the relation of our approach to transformations of UMLsec models to sequence diagrams representing attacks [14].

Acknowledgment

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 (TRE$_S$PASS). 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. Schneier, B.: Attack Trees: Modeling Security Threats. Dr. Dobb's Journal of Software Tools **24**(12) (1999) 21–29
2. Kordy, B., Pièvre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: Don't miss the forest for the attack trees. Computer Science Review **13-14** (2014) 1 – 38
3. Pinchinat, S., Acher, M., Vojtisek, D.: Towards synthesis of attack trees for supporting computer-aided risk analysis. In Canal, C., Idani, A., eds.: Software Engineering and Formal Methods. Volume 8938 of Lecture Notes in Computer Science. Springer International Publishing (2015) 363–375
4. Vigo, R., Nielson, F., Nielson, H.R.: Automated generation of attack trees. In: Proceedings of the 27th Computer Security Foundations Symposium (CSF), IEEE (2014) 337–350

5. Hong, J.B., Kim, D.S., Takaoka, T.: Scalable attack representation model using logic reduction techniques. In: Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2013. (July 2013) 404–411
6. The TREsPASS Consortium: Project web page. Available at `http://www.trespass-project.eu`
7. Probst, C.W., Hansen, R.R.: An extensible analysable system model. Information Security Technical Report **13**(4) (November 2008) 235–246
8. Dimkov, T., Pieters, W., Hartel, P.: Portunes: Representing attack scenarios spanning through the physical, digital and social domain. In Armando, A., Lowe, G., eds.: Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security. Volume 6186 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 112–129
9. Probst, C.W., Hansen, R.R., Nielson, F.: Where can an insider attack? In Dimitrakos, T., Martinelli, F., Ryan, P.Y., Schneider, S., eds.: Formal Aspects in Security and Trust. Volume 4691 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 127–142
10. Kammüller, F., Probst, C.W.: Invalidating policies using structural information. In: Proceedings of the 2nd International IEEE Workshop on Research on Insider Threats (WRIT'13), IEEE (2013) Co-located with IEEE CS Security and Privacy 2013.
11. Kammüller, F., Probst, C.W.: Combining generated data models with formal invalidation for insider threat analysis. In: Proceedings of the 3rd International IEEE Workshop on Research on Insider Threats (WRIT'14), IEEE (2014) Co-located with IEEE CS Security and Privacy 2014.
12. de Nicola, R., Ferrari, G.L., Pugliese, R.: KLAIM: A kernel language for agents interaction and mobility. IEEE Transactions on Software Engineering **24**(5) (May 1998) 315–330
13. Aslanyan, Z., Nielson, F.: Pareto efficient solutions of attack-defence trees. In Focardi, R., Myers, A., eds.: Principles of Security and Trust. Volume 9036 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2015) 95–114
14. Jürjens, J., Wimmel, G.: Security modelling for electronic commerce: The common electronic purse specifications. In: Towards The E-Society: E-Commerce, E-Business, and E-Government, The First IFIP Conference on E-Commerce, E-Business, E-Government (I3E 2001). (2001) 489–505