The Attack Navigator

Christian W. Probst[1], Jan Willemson[2], and Wolter Pieters[3]

[1] Technical University of Denmark
cwpr@dtu.dk
[2] Cybernetica, Estonia
janwil@cyber.ee
[3] Delft University of Technology, Netherlands
w.pieters@tudelft.nl

Abstract. The need to assess security and take protection decisions is at least as old as our civilisation. However, the complexity and development speed of our interconnected technical systems have surpassed our capacity to imagine and evaluate risk scenarios. This holds in particular for risks that are caused by the strategic behaviour of adversaries. Therefore, technology-supported methods are needed to help us identify and manage these risks. In this paper, we describe the attack navigator: a graph-based approach to security risk assessment inspired by navigation systems. Based on maps of a socio-technical system, the attack navigator identifies routes to an attacker goal. The computed routes do not take specific attacker properties such as skill or resource into account. These can be added through attacker profiles, thus simplifying exploration of different attack scenarios. This enables defenders to explore the effectiveness of defense alternatives under different threat conditions.

1 Introduction

The need to assess security and take protection decisions is as old as our civilisation, and maybe even older. Looking around in nature, we see that animals try to build their lairs in safe places and that some plants grow prickles. These kinds of decisions are not taken in a conscious way, but are rather a result of a long evolutionary trial and error process.

What differentiates humans from other species is the highly complex technical environment we operate in. The speed of development of this environment exceeds the capabilities of natural evolution by several orders of magnitude, which means we cannot rely on evolution to develop safeguards. Instead, we need security assessment methods to identify potential threats, and to allow us to cope with the highly sophisticated attacks being enabled by our environment.

On the other hand, our perception of surroundings is still very much limited by what evolution has provided for us. Humans are averagely good at perceiving visual images, sounds, and smells, but not so much at grasping all the small details and implications of large infrastructures. Yet, in order to utilize such infrastructures efficiently, we need such abilities in one way or another.

Even if humans manage to collect adequate environmental data, their risk comprehension may be severely biased due to educational, cultural, psychological, political, and other reasons [1–3]. Hence, there is a clear need for tools that provide a visual, easy to comprehend overview of the environment, but at the same time being rational and unambiguous. The target of the TRE$_S$PASS project [4] is to achieve exactly that – assist humans in taking security decisions about large, complex infrastructures in a way that is easy to perceive given our limited capabilities.

In security risks, we deal with strategic attackers who plan their actions. This means that we must be able to "think thief", and predict possible attack scenarios by imagining attacker behaviour. The central innovation to achieve this goal is the introduction of the notion of *attack navigator map*. It can be seen as an effort to bridge the gap between complexity of real systems and limits of human perception by utilising a concept familiar to all of us, namely spatial navigation. This approach gives us several benefits:

– Moving towards an attacker's goal corresponds intuitively well to navigating through complex terrain, together with the need to take decisions, achieve subgoals, etc.
– Navigation optimisation is rather well studied and understood, as opposed to complex system security.
– Navigation can be handled on different levels of abstraction. There can be a bird-eye version for executive-level, grass-root version for technical level, and an arbitrary number of intermediate levels as needed.

All these aspects make navigation a good metaphor for studying security assessment of complex infrastructures and for communicating assessment results.

The remainder of this paper is organised as follows. In Section 2 we outline the main steps of the TRE$_S$PASS process that provides analysts with the toolset and methodology forming the basis of the attack navigator, which then is described in Section 3. Sections 4 and 5 explain how to move from a high-level abstract view of the environment (the satellite view) to a fine-grained system model (the map) and how to find routes (the attacks). Finally, Section 6 discusses how to select countermeasures based on TRE$_S$PASS analysis, and Section 7 draws some conclusions.

2 The TRE$_S$PASS Process

Of course it takes more than just a good metaphor to build a usable risk assessment system. In practice, the analyst needs a working toolset and methodology that would be able to support the navigation approach on various levels of abstraction. The main result of the TRE$_S$PASS project are the toolset and methodology that together support the TRE$_S$PASS process, which we describe in this section.

In order to achieve the navigation effect, one needs an analogue of a map to navigate on. In the real world, maps represent cities and streets, and to a certain extent artefacts such as points of interest. These maps are produced by geographers based on satellite images and inspection of the terrain under consideration. In the TRE$_{\text{S}}$PASS approach, the role of a map is played by the *system model*, a formal representation of the socio-technical environment to be analysed. System models contain a number of components from such environments:

– **Actors** represent human players or processes involved in the system;
– **Assets** can be either **items** or **data**;
– **Locations** represent where actors or items may be situated either physically or digitally;
– **Edges** describe possible relocation paths between locations;
– **Policies** describe access control and specify allowed actions, *e.g.*, get some data item from a location or move between locations; and
– **Processes** formalize certain state transition mechanisms, *e.g.*, computer programs or virtual machines.

Unlike in the real world, there is no satellite to provide pictures of the environment. The model creation is instead the result of a collection of processes that resemble the combination of satellite and geographer. Before the actual model creation can start, information about the system needs to be gathered. This happens in several parallel processes, both via a specially crafted user interface and automated data acquisition, *e.g.*, in case of large IT infrastructures.

When using a real map for navigation, the goal is to reach a certain location under certain constraints, *e.g.*, as fast as possible, as economical as possible, or without using freeways. Once a system model is built, the attack navigator needs an *attacker goal* to explore the ways to achieve this goal by moving through the model. The goal itself is stated as a policy violation, *e.g.*, illegitimate access to a data asset, and as such can serve as a trigger for an automated navigation procedure.

At this point, navigation through a system model and orienteering across a terrain start to differ. As mentioned above, finding one's way in nature or urban environment usually has a well-set optimisation goal, typically path length or time that it would take to follow this path.

Navigation through a system model is relatively less understood and the methods of along-the-path optimisation are much less mature than shortest path algorithms on terrain graphs. Hence, the output of an attack navigator has to contain more information and optimisation itself has to happen at a later stage.

In case of the current toolset implementation of TRE$_{\text{S}}$PASS, this output contains formal attack vector descriptions in the form of attack trees [5]. This is not the only possible option, but attack trees were chosen since they are rather well established and accepted in the risk assessment community [6]. Also, computational methods have been developed for various optimisation targets that can be stated for attack trees [7–10].

After the analysis of the attack trees has been finished, the results are displayed to the end user on a visual front-end. The user can then take decisions concerning overall security level, required additional controls and possible model updates. After the model has been updated, the analysis can be run again to study the effects of the changes on the security level.

3 The Attack Navigator

We now will look closer at the attack navigator itself. Car navigation systems are independent of the car they are used in, *i.e.*, properties of the car are often ignored since they typically are the same for each car. The navigator may have options to avoid, for example, unpaved roads in non-4WD cars but these options are not explicitly linked to types of cars.

In the attack navigator, the important properties that influence the possible attacks are properties of the attacker. Just as in car navigation systems, in many current models of security risk, these attacker properties are implicit. The risks and identified attacks by such methods are annotated with probability, time, and cost values, which are based on assumptions on the attacker that tries to perform the attack.

Threat agent modelling [11–13] aims at specifying explicit threat agents as a basis for security risk assessment, with properties such as skill, resources, and objectives. This may lead to profiles such as activists, terrorists, or spies all with specific properties.

The TRE$_{\text{S}}$PASS attack navigator concept takes an important step beyond current models of security risk by leveraging threat agents as attacker profiles. The attack navigator analysis uses a combination of a navigator map and an attacker profile to derive

- suitable goals for the attacker based on attacker motivation, and
- feasible routes to that goal and properties of these routes based on skill and resources from the attacker profile.

The attacker profiles also imply a link between attack navigators and security economics [14]. Both attackers and defenders have costs for their actions, and utility functions associated with the possible outcomes, but only a limited budget. The utility of attackers may be different based on their motivation, and this can be used in the analysis of attack trees [15]. The attack navigator aims at optimising defender investments, assuming that

- attackers optimise their investments as well,
- the defender moves before the attacker, and
- the attacker knows what the defender has done.

This amounts to a simple two-step game with minimax optimisation [16]. One can also consider attacker behaviour over time in order to get frequency metrics for risk analysis [17].

The similarity with economic models also means that there is quite a bit of uncertainty in the results of computations. The assumptions made may not always hold, and the available data is fragile. The claim of attack navigators is therefore not a precise prediction of what will happen, but rather a prediction of what is possible or likely, and to what extent countermeasures improve the situation. Even if results are not the exact numbers we would like to have, they can be useful for comparing options, or even as thinking tools for imagining possible attacks.

4 From Satellite View to Maps

An essential component of a navigator is the underlying map, on which routes are computed. As such they also form an important component of the attack navigator. Maps of the real world are created based on satellite images and the work by geographers. This approach is only partly feasible for creating maps of organisations: while the overall building structure can be assessed from the outside, elements such as access control policies or network and social structures cannot. These elements, however, form an essential part of attack navigator maps, since they can be enabling factors of attacks, *i.e.*, routes through the navigator map. Satellites are not the right tool for another reason: the organisations under scrutiny are typically rather small and consequently also only cover a limited area. If the attack navigator map covers a bigger area, this part of reality can usually be represented by parts of a real map.

4.1 Models of Reality

When creating maps as models of reality, one needs to abstract the real world by a concept that is suited for automated detection of routes. For real navigation systems, maps are stored as graphs with nodes connected by edges; both nodes and edges can have properties, *e.g.*, size of a city, size of a street, or whether it is open for traffic or not.

Models for attack navigators follow the same approach: organisations are abstracted to graphs, nodes in the graph represent locations in the organisation, and edges between nodes represent connectivity between these locations. The construction of attack navigator maps follows a different approach than for real

Table 1. An overview of components in the attack navigator map and the tools and processes to identify them.

Real world	Model component	Tool
Relevant area	Locations and edges	Maps
Computer networks	Assets and edges	Network exploration tools such as `nmap` to explore network infrastructure.
Human actors	Actors	Demographic surveys, personnel profiles
Physical access control	Policies and processes	Documents and interviews
Computer access control	Policies and processes	Documents, extraction tools, interviews
Software processes	Processes	Documents, extraction tools, interviews

maps, though. As mentioned above, satellites are not really applicable. They can, however, serve as a metaphor. Where satellite pictures give a view of the real world that needs to be interpreted to create a map, tools can be used to obtain a similar view of organisations.

For creating attack navigator maps, a collection of tools or processes are required to collect information about the different parts of an organisation and its surroundings as necessary for the map. Table 1 shows components of attack navigator maps and tools and processes to collect them. In general, whenever adding a new category to be represented in attack navigator maps, one will also need to add a new tool or process to collect the necessary information.

As shown in Table 1, quite a number of components are obtained through interviews or by running tools. This is where the *modeller*, the attack navigator map's equivalent of the geographer, becomes important. Like the geographer is in charge of assembling the map, and interpreting parts of the satellite image, the modeller is in charge of integrating the bits and pieces of infrastructure and data. Especially the interview parts require special attention, since extracting and interpreting the information obtained through interviews is difficult.

In the TRE$_{\text{S}}$PASS project, a set of tools for physical modelling have been developed [18] to structure the interview process; physical modelling enables employees to contribute to the map creation as domain experts with inside knowledge of their organisation and its policies, assets and values. Physical modelling provides a way to engage employees into the map creation, and to give them a creative process to provide input.

The attack navigator map is constructed around the mapping of locations together. The locations in the different infrastructures establish the connection points between the different layers of the organisation. Access control policies are associated with locations in the building layer and assets in the network layer. Locations in the network layer can coincide with locations in the building layer. Assets are located at other assets or at locations of the network or building layer. Attack navigator maps are structured using these co-locations.

Figure 1 shows a small example for a navigator map with different locations, actors, and assets. In the office there is a safe with a secret in it, and Bob has a key to open the safe. There is another key on the shelf in the reception. Alice wants to obtain the secret from the safe, but the safe has a policy that requires actors to have the matching key in order to open the safe and access its content. Accessing content is represented as input in system models.

4.2 Policies

Policies play an important role in attack navigator maps, since they describe how access to certain nodes is restricted, and what an actor in the model needs to fullfil to access the annotated location or asset. Examples include key cards or keys that are required to access a door. Besides these *local* policies, there also exist system-wide or *global* policies. Global policies identify the assets of an organisation that should be protected against attackers. For example, they might specify that a certain file type is not allowed to leave the organisation,

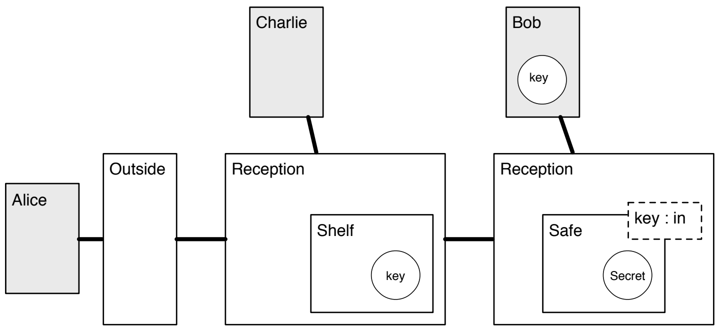

Fig. 1. Example for a small system model with several actors, locations, and assets.

or that a certain location may only be entered at certain times or with a set of credentials. Section 5 discusses how these global policies guide the computation of attacker routes.

4.3 Model Patterns

Like real maps, attack navigator maps tend to contain components that are similar to each other; they share the same structure, but might be different with respect to some properties. For creating maps, there exist standards of such patterns used by map editors.

For attack navigator maps, patterns are equally important since many elements occur repeatedly. To ease the modeller's task, model patterns are provided in a library. Model patterns are sub-graphs that can be put into the attack navigator map. When such a pattern is put into the map, it is instantiated and can be configured to match the element of the real world it represents.

Model patterns also include policies and processes, which represent access control restrictions and functionality at nodes in the model. For access control or for modelling, *e.g.*, network infrastructure, policies and processes can be combined to model quite complex scenarios. For example, role-based access control can be modelled by allowing different roles to output different messages to a location, where each message triggers a process that implements the assigned functionality.

5 From Maps to Routes

Once an organisation has been represented using a graphical model, the attack navigator can identify possible routes on the map for the attacker to reach a goal [19, 20]. In this section we discuss the different steps in doing so. After introducing the representation of attacker routes in the next part, we discuss the actual attack navigation and attack patterns, which can be used to extend identified attack in a similar way as the model patterns discussed in the previous section.

Like real navigation, attack navigation is white-box testing of a map. We assume that the attacker has perfect knowledge of the organisation and knows, *e.g.*, where assets are located, what the layout of the organisation is, or how employees can be social engineered. Scenarios with incomplete knowledge can be considered as well, *i.e.*, an attacker who needs to explore the organisation, but then the impact of attacks can be expected to be lower than for an attacker with perfect knowledge.

5.1 Attacker Routes

Before presenting the actual routing mechanism on attack navigator maps, we briefly discuss the representation of routes. In a navigation system, routes are series of coordinates, often with information about potential congestion on that part of the route. A navigation system assumes that its user is rational and will follow the suggested route. Only once deviations from that route are observed, it will start to recalculate a new route from the position where the user is at this point.

Attacker routes are computed slightly differently, and consequently need another representation. For attacker routes, we are interested in all possible attacks. As described above, the result of the attack navigator is the set of all attacks that are possible in the model, quantified by some property, and ranked accordingly. This is similar to the regular navigator: for navigation, only the shortest, fastest, or most economic route is displayed. Due the complexity of attacks, this selection is far from easy for the attack navigator; the result is therefore presented to a human defender who will dismiss impossible or negligible attacks.

To enable this selection process, attack trees [5,6] are the ideal representation, since they combine different possible attacks that lead to the same goal. The root of an attack tree represents this goal, and the sub trees represent sub-attacks that either need all to be fulfilled, or where one is sufficient to reach the goal. For representing attacker routes, the former would represent that several steps need to be taken, and the latter would represent different possible routes. We present examples for attack trees in Figure 5.

5.2 Attack Identification

Attack identification is the actual navigation on the attack navigator map. Like real navigation, it takes an attacker location and identifies a possible route from this location to the desired goal.

For the attack navigator map shown in Figure 1, the goal is clear: Alice wants to obtain the secret from the safe. Once the goal is identified, the paths to the goal (only one in the example) and the missing assets are identified. Alice lacks the key, which is available from Bob or from the Shelf. The upper part of Figure 5 shows part of the attack tree generated for this scenario.

Goal Identification: As discussed above, the goal in attack navigator maps is identified based on global policies of the modelled organisation. These policies represent a goal of the organisation that should not be violated. Examples include that employees should not send secret files by email, that in general secret files should not leave the organisation, or that the password file on a computer may not be read. In the attacker route, this goal would be the root node, and its children would represent different attacks that enable an attacker to reach this goal.

The result of the goal identification is an action, which the attacker tries to perform, or an asset, which the attacker tries to possess. An important observation is that the latter is a variant of the former; to possess an asset, the attacker needs to perform an action to obtain it. In the attack navigator, this is represented as inputting the asset.

Attack Paths: For each of the identified attacker goals, there may exist numerous paths to reach the goal location, where the goal action can be performed, or where the goal asset can be obtained. The attack navigation considers all these paths, since they may result in different impact or may otherwise have different properties that the defender deems important.

This property is essentially different from standard navigation, where it is a safe assumption that one can ignore routes that are too slow compared to the optimal routes at any given point during routing. Attacker routes are only evaluated in the next step and a defender might use different criteria for evaluating trees; as a result, there is no decision basis for ignoring attack routes or for evaluating them on the fly. One important evaluation criteria is an attack route's impact, which does not increase continuously, but may have discontinuous changes based, *e.g.*, on the assets obtained.

Every step in an attack path consists of a step in the model, be it moving from one location to another, or be it obtaining some asset—either the final one, or one that is needed to perform some other action. For example, if the attacker goal is to read the password file on a central server, then the root password of that machine is an asset that needs to be obtained.

Required Resources: These required resources are acquired on the fly. Whenever the attacker encounters an action in an attack path that requires an asset such as the password for the server machine, a new attack is spawned, at the end of which the attacker has obtained that asset. It is important to note that the routes always assume success, even though an attack might be prohibited. From the attacker's viewpoint the asset has been obtained, and the original attack can continue as planned. This should also be the defender's point of view—the interesting case is not a defeated attack, but a successful one.

Moving Assets: Finally, attacker routes can differ significantly from normal routes through the fact that the goal asset in attacker models can move or be moved, resulting in novel attacks. In a regular navigation system this would mean that the goal could be moved, resulting in a shorter, faster, or longer trip.

While this is not possible for real goals, it is a common attack strategy in attacker maps: The attack consists in making the asset move, and then finding attacks to all those locations that the asset can reach. The means of making an asset move differ depending on the kind of asset. Data usually moves through processes, which are triggered by the attacker; assets usually move with actors, which an attacker must social engineer.

An example for an attack that made the data move is a cloud service administrator who attached a network sniffer to the local network in the server room, and then made a virtual machine migrate from one server to another; as a result, the administrator had a copy of the network traffic that he could playback to obtain a copy of the virtual machine.

5.3 Detailedness of Models

One general issue with maps and routes, both for real maps and attack navigator maps, is the level of detail in the maps. In both cases, if the maps are too detailed, it is very difficult to identify a close-to-optimal route; if the maps however are too imprecise, the routes are not realistic either, and may lack important information needed to follow the route.

In attack navigator maps, the level of detail relates to how detailed the identified attacks are. Coming back to the cloud administrator example, modelling the bits and bytes of the virtual machine and the OSI network stack is likely too much detail. On the other hand, in a system that models only the two servers not including the network infrastructure, it will not be possible to identify the attack at all.

The level of detail is therefore an important design criteria when designing (attack navigator) maps. A good guiding principle is to include only those elements that are essential for the functionality of the overall system, but exclude internal workings of the system. The modelling work in the TRE$_{\mathrm{S}}$PASS project has shown that it is better to exclude some details and to rely on attack patterns to add possible attack steps to the generated attack route.

5.4 Attack Patterns

To deal with detailedness of models, and the resulting detailedness of attacks, we introduce attack patterns, which are similar to the model patterns discussed in the previous section. For too detailed models it is difficult to deal with the resulting overly detailed attack trees. For models with too few details, this is equally difficult. However, it is easier to add "standard" attack pattens to an attack tree, than it is to remove superfluous nodes.

Attack patterns identify typical approaches to performing an attack. Since they are used to extend the attacker routes or attack trees introduced earlier, attack patterns are represented as sub trees as well.

```
label match {
  case IN attacker item container:
    // get type attacker from attacker profile
    // get type item from knowledge base
    // get type container from knowledge base
    // insert APL attacks that allow to extract item from container
  case MAKE attacker actor action:
    // get type attacker from attacker profile
    // get type actor from attacker profile
    // insert APL attacks based on types and action
  //...
}
```

Fig. 2. Code for the expansion of general attack trees in a context-unaware fashion. The expansion algorithm iterates over all leaf nodes and matches leaf node labels against the known cases. If a leaf node label matches a pattern in the attack pattern library, it is inserted into the general attack tree. Figure 5 illustrates this process.

Attack patterns are applied by inspecting the actions in an attack tree, and by exploring whether a certain action realisations of this action are known. The overall structure of this exploration is shown in Figure 2: The expansion algorithm iterates over all leaf nodes and matches the action at this leaf (represented as leaf node labels) against the known cases. If a leaf node label matches a pattern in the attack pattern library, it is inserted into the general attack tree.

This approach has a number of benefits beyond it contributing to clearing out models and keeping them free of clutter. Attack pattern libraries can be shared between organisations to disseminate findings about possible attacks. Once an attack pattern is available in the attack navigator, whenever a matching action working on matching types of assets or actors is found, the pattern will be instantiated.

Two attack patterns are shown in Figure 3 and 4. The pattern in Figure 3 replaces obtaining an item from an actor with either stealing the item or social engineering the actor to give it to the attacker. The root of the pattern specifies the action and the types of the arguments for the actor A obtaining an item I from an actor C, represented as A inputing I from C:

$$IN\ A\ item : I\ actor : C$$

This information is crucial for applying the pattern, also because these arguments A (attacker), I (item), and C (actor) occur again in the attack pattern, and must be replaced with the matching values from the attack tree.

The pattern in Figure 4 is a bit more complicated; it describes that A makes B perform some action for him. As before, the root of the pattern is replaced with nodes that represent different alternatives in the attack. It should be noted that later phases may discard some of the generated attacks since they might be infeasible.

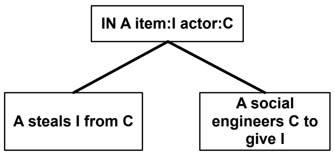

Fig. 3. An attack patterns that replaces the action of obtaining (inputing) an item from an actor with two attacks, one stealing the item from the actor, and the other one social engineering the actor to hand over the item.

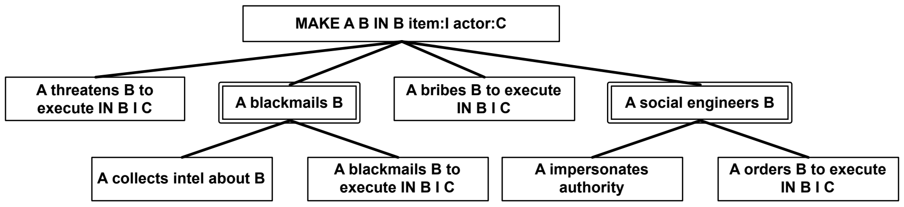
MAKE A B IN B item:I actor:C
A threatens B to execute IN B I C
A blackmails B
A bribes B to execute IN B I C
A social engineers B
A collects intel about B
A blackmails B to execute IN B I C
A impersonates authority
A orders B to execute IN B I C

Fig. 4. An attack pattern that replaces social engineering an actor A to obtain (input) an item from another actor B. The alternatives inserted are threatening, blackmailing, bribing, and social engineering actor A to perform the action.

Social Engineering: A typical example for attack steps that should be added through attack patterns, not through adding more details to the model, is social engineering. Social engineering is an important factor of attacking organisations through exploiting the knowledge and the access rights of employees or insiders. Social engineering usually requires creating a pretext, which is part of bringing the victim into a situation where it either is not aware of contributing to an attack, or where it has sufficient reason to believe to do the right thing.

Due to its dependency on human behaviour, social engineering is difficult to deal with in formal methods. Since the choice of pretext, for example applying authority, depends heavily on the victim, this kind of attack is best dealt with through attack patterns. The patterns shown in Figures 3 and 4 introduce social engineering nodes, where the attacker social engineers another actor to perform an action.

5.5 Attacker and Actor Profiles

The success of both attackers and defenders depends on the type of actor and the skills considered. In the attack navigator, different profiles are considered based on threat agent modelling [11–13], which provides skills, resources, and objectives of actors. The attack navigator analysis uses these profiles to identify attacks and countermeasures on a system model, and to predict the likelihood of success and impact of the attack.

Actor profiles separate the planning of a route from its assessment: routes in the attack navigator are *all* possible attacks with respect to the model. Not all

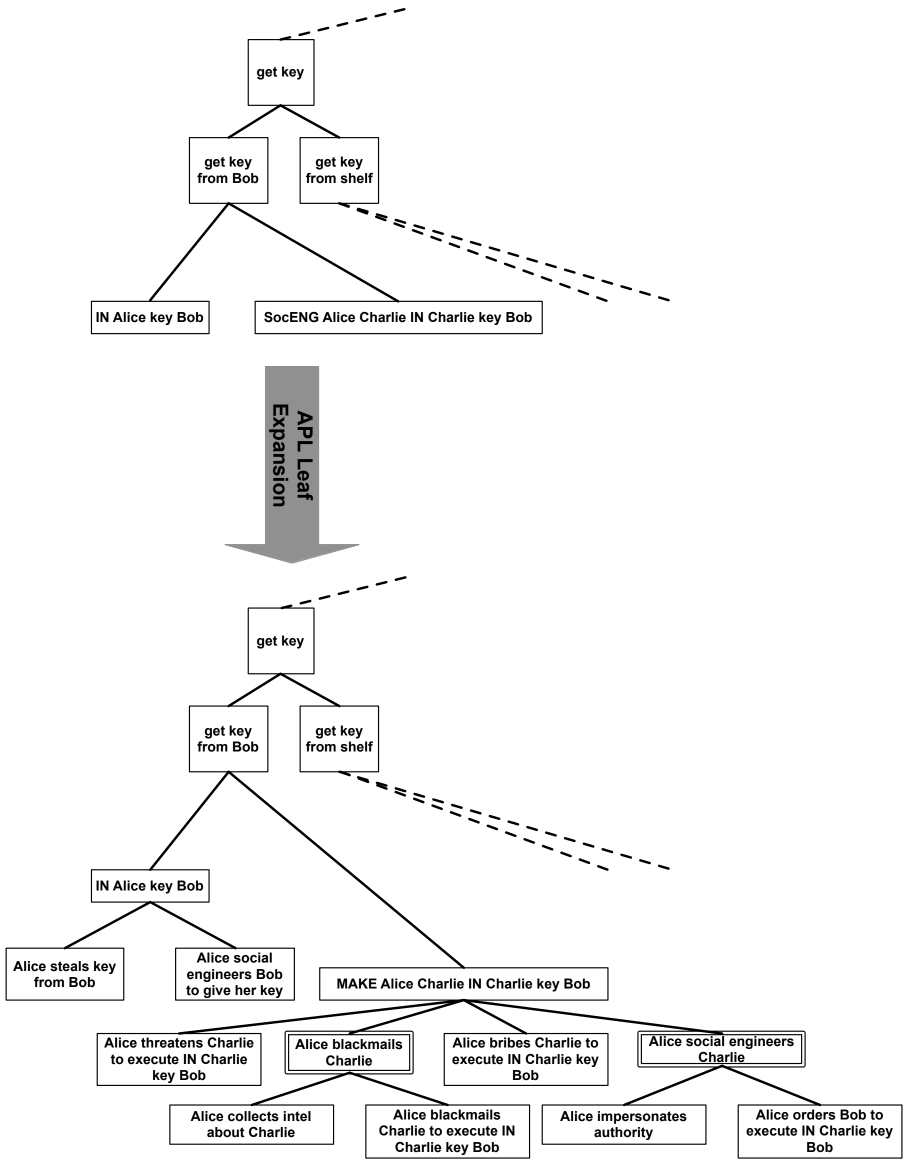
get key
get key from Bob
get key from shelf
IN Alice key Bob
SocENG Alice Charlie IN Charlie key Bob
APL Leaf Expansion
get key
get key from Bob
get key from shelf
IN Alice key Bob
Alice steals key from Bob
Alice social engineers Bob to give her key
MAKE Alice Charlie IN Charlie key Bob
Alice threatens Charlie to execute IN Charlie key Bob
Alice blackmails Charlie
Alice bribes Charlie to execute IN Charlie key Bob
Alice social engineers Charlie
Alice collects intel about Charlie
Alice blackmails Charlie to execute IN Charlie key Bob
Alice impersonates authority
Alice orders Bob to execute IN Charlie key Bob

Fig. 5. The expansion of a part of a general attack tree. The patterns (Figure 3 and 4) may have holes, which are filled with attributes from the leaf node that is expanded. The boxes with a double frame represent conjunctive nodes, requiring all child nodes to be executed.

of these attacks are feasible for all attackers, but they are still attacks. For car navigation this would mean to show all possible paths from the starting point to the goal, but routes that require a 4WD car would not be feasible for all cars.

Real navigation cannot consider all routes, since it requires the driver to decide, which of the many possible routes is the best with respect to an optimization goal. For attack navigation it is the reverse: a single route or attack out of many is not useful; considering *all* attacks enables the tools to identify countermeasures that disable as many attacks as possible with a certain effort, and it also enables analysis of which kind of attacker to watch out for.

6 Countermeasures

A risk assessment would be useless if it would not come with a way to incorporate countermeasure effect analysis. There are two major ways in which the TRE$_{\text{S}}$PASS methodology supports this.

The first approach is generic and can in principle be applied to any risk assessment framework. It uses the framework as a black box which takes some inputs (in the case of TRE$_{\text{S}}$PASS, the system model) and gives some output (in our case, prioritized attack vectors). Assuming the end user is able to change the model and run the analysis again, we obtain a full operational loop with human involvement, where the user is expected to interpret the analysis results and actively participate in the model development.

Even though TRE$_{\text{S}}$PASS aims at automating the risk analysis process, we do not think that full automation is possible or even needed. Again coming back to the terrain navigation analogue – the human is not expected to follow GPS blindly. In fact, several cases have been reported when people being overconfident in the GPS reading have ended up in serious accidents [21, 22]. And even if the model, *i.e.*, the map, used by the GPS device is correct, the user may still have optimization preferences the device is unaware of.

In some sense, the situation is even better with the attack navigator. Here the user has more options than just selecting between the routes offered by a machine. The user can actually change the map by implementing additional controls, increasing efficiency of the existing ones, etc. All these changes would hopefully change the risk landscape, and running the analysis tool again on an updated map is the prime way of verifying this.

As mentioned in Section 2, attack trees are not the only possible attack description language that can be used in TRE$_{\text{S}}$PASS. Attack-defence trees by Kordy *et al.* [23] are an alternative approach to countermeasure selection. In principle, this formalism allows for integrating countermeasures into the risk assessment process on a lower level than the generic model update approach described above. It is possible already at the attack generation stage to also generate certain defence nodes into the tree or to obtain those from standard libraries. The option of changing the model and running the analysis again of course remains, so the attack-defence tree approach is potentially more flexible than the one based on classical attack trees. However, since attack-defence trees are considerably more recent and accordingly less studied, the current version of the TRE$_S$PASS toolset (as of 2015) does not yet support this.

7 Conclusions

The navigation metaphor is a new approach to security assessment of complex systems that aims at being more accessible to a human end user than other computer-assisted frameworks. However, no metaphor can make the inherent challenges of risk assessment to go away, it can only try to present them on the level where human decisions can be made more intuitively.

The TRE$_S$PASS project has been building a toolset supporting such a workflow since 2012. We have published key innovations in for example making attacker profiles explicit, quantitative analysis, and visualisation of maps and paths. Our practical and theoretical developments open up for many new and interesting research questions in the area of attack navigation and graphical models for security, for example:

– What is the correct abstraction level for a system models and maps that would be humanly comprehensible and at the same time would allow formal analysis?
– Are there additional opportunities for using the properties of attacker profiles in security analysis? Can we use more advanced calculations or statistics?
– Are the current TRE$_S$PASS model components generalisable enough to perform realistic security assessments on a wide class of systems, or are extensions needed for different types of systems?
– How can we share attack patterns and what are the requirements on the pattern sharing authorisation infrastructure?

Acknowledgment

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. Fischhoff, B.: Risk perception and communication unplugged: Twenty years of process. Risk analysis **15**(2) (1995) 137–145
2. Jasanoff, S.: The political science of risk perception. Reliability Engineering & System Safety **59**(1) (1998) 91–99
3. Weinstein, N.D.: What Does It Mean to Understand a Risk? Evaluating Risk Comprehension. Journal of the National Cancer Institute Monographs (25) (1999) 15–20

4. The TRE$_S$PASS Consortium: Project webpage. Available at `https://www.trespass-project.eu`. Last visited October 31, 2015.
5. Schneier, B.: Attack Trees: Modeling Security Threats. Dr. Dobb's Journal (December 1999)
6. Kordy, B., Piètre-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
7. Jürgenson, A., Willemson, J.: Computing Exact Outcomes of Multi-parameter Attack Trees. In Meersman, R., Tari, Z., eds.: OTM Conferences (2). Volume 5332 of Lecture Notes in Computer Science., Springer (2008) 1036–1051
8. Jürgenson, A., Willemson, J.: Serial Model for Attack Tree Computations. In Lee, D., Hong, S., eds.: ICISC. Volume 5984 of Lecture Notes in Computer Science., Springer (2009) 118–128
9. Jürgenson, A., Willemson, J.: On Fast and Approximate Attack Tree Computations. In Kwak, J., Deng, R.H., Won, Y., Wang, G., eds.: ISPEC. Volume 6047 of Lecture Notes in Computer Science., Springer (2010) 56–66
10. Arnold, F., Hermanns, H., Pulungan, R., Stoelinga, M.: Time-dependent analysis of attacks. In Abadi, M., Kremer, S., eds.: Principles of Security and Trust. Volume 8414 of Lecture Notes in Computer Science. Springer (2014) 285–305
11. Casey, T.: Threat agent library helps identify information security risks. Intel White Paper (2007)
12. Casey, T., Koeberl, P., Vishik, C.: Threat agents: A necessary component of threat analysis. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research. CSIIRW '10, New York, NY, USA, ACM (2010) 56:1–56:4
13. Rosenquist, M.: Prioritizing Information Security Risks with Threat Agent Risk Assessment. Intel White Paper (2010)
14. Pieters, W., Barendse, J., Ford, M., Heath, C.P., Probst, C.W.: The navigation metaphor in security economics. Accepted for Security & Privacy Magazine (2016)
15. Van Holsteijn, R.: The motivation of attackers in attack tree analysis. Master's thesis, TU Delft (2015)
16. Cox Jr., L.A.: Game theory and risk analysis. Risk Analysis **29**(8) (2009) 1062–1068
17. Pieters, W., Davarynejad, M.: Calculating adversarial risk from attack trees: Control strength and probabilistic attackers. In Garcia-Alfaro, J., Herrera-Joancomart, J., Lupu, E., Posegga, J., Aldini, A., Martinelli, F., Suri, N., eds.: Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance. Volume 8872 of Lecture Notes in Computer Science., Springer International Publishing (2015) 201–215
18. The TRE$_S$PASS Consortium: Final requirements for visualisation processes and tools (2015) Deliverable D4.1.2.
19. Kammüller, F., Probst, C.W.: Invalidating policies using structural information. In: 2nd International IEEE Workshop on Research on Insider Threats (WRIT'13), IEEE (2013) Co-located with IEEE CS Security and Privacy 2013.
20. Kammüller, F., Probst, C.W.: 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) Co-located with IEEE CS Security and Privacy 2014.
21. Holley, P.: Driver follows GPS off demolished bridge, killing wife, police say. Available from `https://www.washingtonpost.com/news/morning-mix/wp/2015/03/31/driver` (2015) Last visited October 15, 2015.

22. Knudson, T.: 'Death by GPS' in desert. Available from `http://www.sacbee.com/entertainment/living/travel/article2573180.html` (2011) Last visited October 15, 2015.
23. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Attack–defense trees. Journal of Logic and Computation **24**(1) (2014) 55–87