Attribute Decoration of Attack–Defense Trees

Alessandra Bagnato[1], Barbara Kordy[2], Per Håkon Meland[3], Patrick Schweitzer[2]

[1]TXT e-solutions

alessandra.bagnato@txt.it

Corporate Research Division, TXT e-solutions, I–16100 Genoa, Italy

[2]University of Luxembourg,

{barbara.kordy,patrick.schweitzer}@uni.lu

SnT, University of Luxembourg, 6, rue Coudenhove–Kalergi, L–1359 Luxembourg, Luxembourg

[3]SINTEF

per.h.meland@sintef.no

SINTEF ICT, SP Andersens vei 15 B, N–7465 Trondheim, Norway

Abstract

Attack–defense trees can be used as part of threat and risk analysis for system development and maintenance. They are an extension of attack trees with defense measures. Moreover, tree nodes can be decorated with attributes, such as probability, impact and penalty, to increase the expressiveness of the model. Attribute values are typically assigned based on cognitive estimations and historically recorded events.

This paper presents a practical case study with attack–defense trees. First, we create an attack–defense tree for an RFID-based goods management system for a warehouse. Then, we explore how to use a rich set of attributes for attack and defense nodes and how to assign and aggregate values to obtain condensed information, such as performance indicators or other key security figures. We discuss different modeling choices and trade-offs. The case study led us to define concrete guidelines that can be used by software developers, security analysts and system owners when performing similar assessments.

1 Introduction

The security of any sufficiently valuable system is not static. To keep a system secure, it has to be protected against an increasing number of threats of growing complexity. As defenses are added to the system, more sophisticated attacks break these defensive measures anew. To cope with the resulting, intricate systems, a formal modeling and evaluation approach become indispensable.

One of the formal approaches to assess a system's security is the *attack–defense tree* (ADTree) methodology. ADTrees focus on the interaction between two types of players, attackers and defenders, while keeping the complexity of the formalism at a minimum [Kordy et al., 2011b]. They are a compromise between attack trees, which are too restrictive in their modeling capabilities, and petri-nets, where modeling is quite intricate and computationally complex. ADTrees retain the easily understandable tree structure and are therefore especially useful in an interdisciplinary work environment, where an intuitive understanding of the system is as important as formal foundations. ADTrees even allow a rough first assessment of a system's security purely based on the visual representation of the scenario, making it easy to spot missing or redundant defenses. The theoretical aspects of the ADTree methodology have already been extensively studied in [Kordy et al., 2010, 2011a,b].

The purpose of this paper is to present experiences and provide practical recommendations on the use of attributes in ADTrees. Attributes are the part of the ADTree formalism that allows quantitative analysis, something that is of great value for risk analysis either during planning, development or maintenance of a system. There are numerous security attributes to be found in the literature today, and through a case study we show how a selection of them can be applied, how values are assigned to nodes and how they are used for quantitative analysis. Knowing which attributes to choose and how to estimate their values is a non-trivial challenge and is addressed in detail. Attributes are used to answer questions such as: Is it possible to attack the system? How much would it cost to prevent one or all attacks or implement one or all defenses? How long does it take to secure the entire system? We are interested in extending these answerable questions to bivariate questions, i.e., questions where inputs from attackers and defenders are needed. This, for example, includes questions such as: Given a limited defense budget, can the defender at least defend against some attacks? How does the scenario change in case of a power outage?

The case study was based on an operational *Radio-Frequency Identification* (RFID) system for goods management in a warehouse, taking technical, physical and social engineering aspects into account. There were four players from both academia and industry involved, taking roles as defenders and attackers.

The rest of the paper is structured as follows. This sections continues with a summary of the theoretical foundations of ADTrees and concludes with a short literature review on related work. In Section 2, we review some of the attributes that can be found in the literature and elaborate on different calculation methods. In Section 3, we present the case study scenario and the corresponding ADTree. Section 4 shows the attribute decoration and calculation of values for the ADTree. The results of the case study are discussed in Section 5 and we conclude and synthesize our recommendations in Section 6.

1.1 ADTrees

ADTrees [Kordy et al., 2011a] were introduced by Kordy et al. and are an extension of attack trees [Schneier, 1999, Mauw and Oostdijk, 2005] with defense nodes. An ADTree is a node-labeled rooted tree describing the measures an attacker might take in order to attack a system and the defenses a defender can employ to protect the system. ADTrees allow the system modeler to repeatedly interleave attack and defense components. Consequently, an ADTree has nodes of two opposite types, *attack* nodes and *defense* nodes, and can be seen as a game between two players: an attacker and a defender. The *root* node of an ADTree represents the *main goal of the attacker.* Every node of an ADTree may have one or more *children* of the same type representing a *refinement* of the node's goal into sub-goals. The refinement of a node is either *disjunctive* or *conjunctive.* The goal of a disjunctively refined node is achieved when *at least one* of its refining children's goals is achieved. The goal of a conjunctively refined node is achieved when *all* of its refining children's goals are achieved. Nodes which do not have any children of the same type represent *basic actions.* Every node may also have one child of the opposite type, which represents a *countermeasure.* Thus, an attack node may have several children which refine the attack and one child which defends against the attack. A defense node in turn may have several children which refine the defense and one child, being an attack node, which counters the defense. We understand countermeasures in a general sense, i.e., countermeasure ranges from a complete prevention of the parent's goal over a possible prevention to a weak mitigation.

The semantics of an ADTree is the following. The attack tree, obtained from an ADTree by removing all subtrees rooted in defense nodes, represents how an attacker can attack a given system taking all existing defensive measures into account. The remaining parts of an ADTree, i.e., all defense nodes, their refinements, counterattacks, and so on, represent possible measures that can be put in place in order to defend against the original attack on the system, or attack the newly introduced defenses, and so on. This means that the existing defensive mechanisms are *not* explicitly depicted in an ADTree, whereas, the original attack tree already represents how they should be overcome.

We depict attack nodes by circles and defense nodes by rectangles. Refinement relations are indicated by solid edges between nodes, and a countermeasure is connected to the countered node using dotted edge. In addition, a conjunctive refinement of a node is depicted by an arc which connects the node's edges to its children of the same type. We illustrate the attack–defense language by giving an ADTree, depicted Figure 1. The root of the tree depicts the main goal of the attacker. It is disjunctively refined into two subgoals. The Disjunctive Subgoal 1 is countered by a countermeasure, whereas the Disjunctive Subgoal 2 is conjunctively refined into two subgoals. A more extensive ADTree, where the main goal is a Denial of Service (DoS) attack on an RFID-based management system, is detailed in Section 3.1 and is depicted in Figures 4–7.

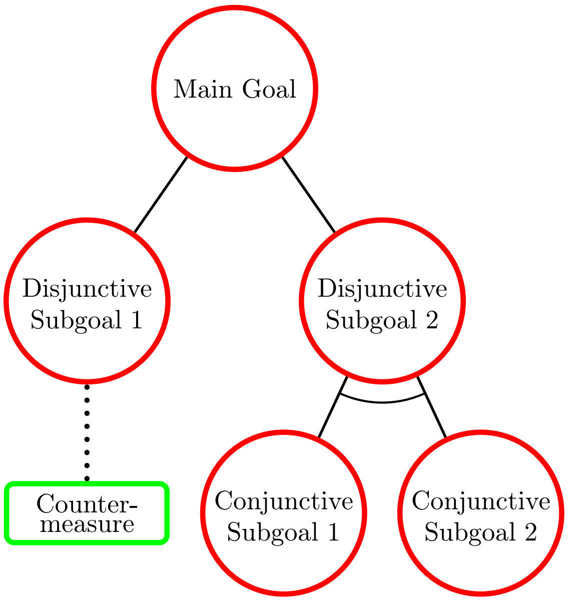

Figure 1: The remove tag subtree

1.2 Related Work

The literature on attack trees is abundant. Piètre-Cambacédès and Bouissou [Piètre-Cambacédès and Bouissou, 2010] have given a historical overview on graphical representations of computer attacks, such as fault trees [Vesely et al., 1981], threat trees [Amoroso, 1994] and privilege graphs [Dacier and Deswarte, 1994], and how these representations lead Schneier to coin the term attack tree [Schneier, 1999].

Many authors treat how to add different kinds of values to attack tree nodes. In Schneier's terms [Schneier, 1999] these values are called attributes. In 1999, he proposed how to analyze the costs and the success probability of an attack with the help of attack trees. Since then, many authors have followed in his footsteps proposing extensions to attack trees and attributes, as well as describing case studies. For instance, Amoroso [Amoroso, 1994], Mauw and Oostdijk [Mauw and Oostdijk, 2005], Buldas et al. [Buldas et al., 2006], Li et al. [Li et al., 2009], and Tanu and Arreymbi [Tanu and Arreymbi, 2010] demonstrate how an attack tree can be parametrized with different kinds of values and how to deduce aggregated results. Moore et al. [Moore et al., 2001] include attacks related to social engineering and physical entering of premises and Saini et al. [Saini et al., 2008] show examples from multiple systems. Baca and Petersen [Baca and Petersen, 2010] have extended attack trees with countermeasure graphs with an example from open-source application development, while Bistarelli et al. [Bistarelli et al., 2006], Edge et al. [Edge et al., 2006] and Roy et al. [Roy et al., 2011] have extended attack trees with a notion of defense nodes for the leaves of the trees.

There also exists a number of deeper studies and experience reports with attack tree based methods applied to real-life systems. Some notable examples are Henniger et al. [Henniger et al., 2009], who have conducted a study using attack trees and a variety of node attributes for vehicle communications systems, Abdulla et al. [Abdulla et al., 2010] with an analysis on the GSM radio network, and Tanu and Arreymbi [Tanu and Arreymbi, 2010] using vulnerability tree, fault tree and attack tree analysis on a mobile SCADA system for a multiple tank and pump facility. Byres et al. [Byres et al., 2004] treat another SCADA case study related to a MODBUS protocol for critical infrastructures. All these work show that attack tree based methodologies constitute a very useful tool for modeling threats and analyzing vulnerabilities of complex systems.

Finally, steps have been made to compare and combine attack tree based models with other modeling techniques in order to obtain a better and more complete way for representation and analysis of threats and vulnerabilities. For instance, Opdahl and Sindre [Opdahl and Sindre, 2009] compare misuse cases with attack trees. They use a class room experiment to gather experimental data and determine which method models the scenario more accurately. According to this study, using attack trees for finding threats is more effective than the use of misuse cases. Moreover, the authors conclude that the perception of a used technique is not correlated with the actual performance of that technique. A similar approach has been proposed by Diallo et al. in [Diallo et al., 2006], where a comparative evaluation of the common criteria, misuse cases and attack trees is presented. Based on the results obtained in these works, Tøndel et al. suggest in [Tøndel et al., 2010] to combine misuse cases and attack trees, in order to represent possible threats, attacks and mitigating countermeasures. This suggestion is made with reuse of models in mind, and is supported by an online repository of security models developed within the SHIELDS project [SHIELDS, 2008-2010]. The results and experiences from the related work have been taken into account when designing the formal treatment of attributes for ADTrees. As ADTrees are the only approach that is formal and allows for interleaved attacks and defenses, the ideas needed to either be formalized or extended to include a notion of interleaving.

2 Background of Attributes

In order to analyze an attack–defense scenario, we use quantitative measures called metrics or attributes. Some attributes, e.g., **costs**, **probability**, appear frequently in the context of attack trees, others can rarely be found. In this section, we provide background information about attributes and related calculation methods. For readability, we always write attribute names in bold font.

2.1 Attribute Overview

Attributes for Attack Trees. In [Schneier, 1999], Schneier proposes decorating leaf nodes of attack trees with the values expressing the component's **costs** in order to deduce the cheapest attack. A similar approach applies to metrics such as **probability**, **severity**, **impact**, and **consequence**. According to Schneier, all of these attributes can be combined or conditioned, in order to identify the cheapest low-risk attack, attacks that cost less than a given amount of money, or cheap, highly successful attacks where only medium skill is needed, etc.

Other researchers extend this catalog of attributes with further metrics. Edge et al. [Edge et al., 2006] introduce **protection costs**; Byres et al. [Byres et al., 2004] use **detectability** which illustrates how easily a defender can discover an attack and **difficulty** which specifies the needed skill of the attacker. As suggested by Henniger et al. [Henniger et al., 2009], **attack time** — which models an amount of time needed to perform an attack — may be considered independent of an attacker's **skill**, **costs** or **resources**. Attacks that a defender cannot really protect himself against can be annotated as **unmitigatable**. Instead of modeling constraints as additional child nodes of a conjunctive node, we can use attributes, such as **requires an insider** or **needs electricity**. These and similar attributes merely depict a choice between two options and can therefore be modeled with Boolean values, as suggested by Schneier [Schneier, 1999]. We subsume Boolean attributes under the keyword **special skill**. A formal treatment of attributes for attack trees, that guarantees compatibility with underlying semantics, was first described by Mauw and Oostdijk in [Mauw and Oostdijk, 2005].

Attributes for Defense Nodes. Traditionally, attack trees only consider attributes directly related to attacks or attackers. With ADTrees we can also cover attributes that quantify defenses and their behavior. Many attributes related to attacks can straightforwardly be extended to encompass defenses. For instance, **costs** can refer to **costs of performing an attack** or **costs of performing a defense**. Similarly **probability of success**, **probability of occurrence**, **required skill level**, **number of possible countermeasures** and many more can be adapted. Other attributes might only make sense for either attacker or defenders, such as **penalty** which a law-abiding defender would never have to worry about, or **response time** where a defender might be depended on the **response time of a third party**. The usefulness of distinguishing between attributes for attack and defense nodes becomes apparent, once we look at questions related to attributes such as **social costs** (addition of attacker's as well as defender's costs) or attributes where values for one player have a direct consequence on values for the other player, as in the case of **satisfiability** or **probability**. In addition, defense nodes and their associated attribute values allow us to answer bivariate questions, such as how much does a defender have to spend on defenses, if he knows that the attacker has a low **skill level**.

Meta-attributes. Attributes themselves are the main ingredient to perform a quantitative analysis with the help of ADTrees. However, when associating attribute values with nodes, we might still want to distinguish between the associated values, even if the values themselves are the same. For example, an RFID security expert would associate a medium probability of occurrence to a DoS attack and is very confident about that, whereas a person working in computer forensics would also associate a medium probability to the node, but would probably be less confident about it. This additional information, describing an attribute value we call a *meta-attribute*. An example is **confidence**, which indicates how certain a decorator is, when associating an attribute value with a given node. Another meta-attribute is **coverage**, which expresses the number of people who have associated an attribute value with a given node. Meta-attributes are suitable to be used in combination with any attribute. Using meta-attributes allows us to model the desired properties more accurately, and to improve their quantification. Meta-attributes constitute one of the novelties of the methodology used in the current case study, as they have not yet been mentioned in the context of attack trees nor attack–defense trees in the existing literature.

Value Domains. Attribute values can range over diversified types of mathematical domains. One can consider Boolean values, values from a nominal scale, e.g., `Low`, `Medium`, and `High`, real numbers or even discrete or continuous probability distributions. If data is available, the probability distribution could be estimated from histograms. Meta-attributes can also be quantified using the values from any of possible domains. For example, the pair $(4.23, 2)$ could be a possible value for **costs of attack** with the meta-attribute **confidence**, where the domain is modeled as the product space of the real numbers and a nominal scale from 1 to 5. Instead of a single value, it is also possible to associate sets of values to the nodes. For example, if we know that the attack costs are not `High`, we could associate the set $\{\texttt{Low}, \texttt{Medium}\}$ to the corresponding node.

2.2 General Calculation with Attributes

Attributes provide a powerful analysis tool for vulnerability scenarios. They help us estimate which attacks may happen with a high probability and which countermeasures should be applied. However, to get useful insights from the analysis, it is necessary to have accurate values associated with all the nodes of an ADTree. One possibility is to ask experts to provide the values. Another strategy is to involve several people, such as the system owner, developers and administrators, to perform the task. In any case, this process can be very time consuming, costly and highly error-prone, depending on the tree complexity and the number of attributes. Thus, numerous approaches have been proposed, allowing us to deduce values for one node, based on values already associated with other nodes, or to combine values for several attributes in order to deduce the value for another attribute. In this section, we give a brief overview of the calculation methods already present in the literature.

As pointed out by Schneier [Schneier, 1999] in the case of attack trees, the most intuitive calculation procedure on attack trees is the bottom-up approach. The idea is to only associate attribute values with the basic actions and then deduce the values corresponding to the refined nodes from the values associated with their children. The functions which are used to calculate the value for a parent node depend on the type of the corresponding refinement. The bottom-up approach presents several advantages. First of all, we only have to decide on values for a small amount of nodes, which reduces the time necessary for the attribution of values. Moreover, estimation of values for basic actions should be feasible in most cases, since such actions can be easily understood and quantified (if this is not the case, a node should be refined further). Finally, this approach is suitable for evaluation of a large number of attributes and it can be automated. A formal framework for the bottom-up approach for attribute evaluation on attack trees has been developed in [Mauw and Oostdijk, 2005]. This approach has been extended to ADTrees by Kordy et al. [Kordy et al., 2011a], where Example 5 illustrates the calculation procedure.

It is also possible to define a new attribute by combining several existing attributes. Such combinations allow us to estimate values corresponding to more complex properties, for which it would be difficult to provide values directly. As an example, Edge et al. [Edge et al., 2006] have defined a variant of a **risk** attribute for attack trees based on the formula **risk** = (**probability**/**costs**) $*$ **impact**. A similar approach has been used in [Jürgenson and Willemson, 2008], where the **costs**, **success probability**, **gain** and **penalty** attributes have been combined in order to define a new attribute called the **exact expected outcome** of the attacker. Henniger et al. [Henniger et al., 2009] combine the attributes **elapsed time**, **expertise**, **knowledge of system**, **window of opportunity** and **required equipment**, in order to deduce the **required attack potential**. Finally, Fung et al. [Fung et al., 2005] show how the **difficulty level** associated to the non-refined nodes can be used to estimate the **survivability** in the root node.

It is also important to observe how multiple attributes relate to each other and how the values for one attribute may influence the values for another one. For instance, the **costs** associated with a given attack component can be used to estimate the corresponding **probability of occurrence** value, i.e., to deduce how probable it is that an action will take place. This is of particular importance, when we have some specific knowledge about the attacker. For instance, if we know that the attacker has a limited budget we can deduce that, with a high probability, he will not perform actions which are more expensive than a certain threshold. Moreover, attribute values can be used to check soundness of a scenario. Rational reasoning lets us deduce that attack components, which require a higher investment than the potential gain of the attacker, do not have to be considered, because in such case the attack would be unprofitable for the attacker. Similarly, as pointed out by Herley [Herley, 2009], the **protection costs** of a threat should not be greater than the **benefit** gained by following a security advice. Indeed, such a protection would do more harm than good, from an economic point of view. Furthermore, if the **protection costs** is greater than the **impact** of an attack, economically speaking the protection causes more harm than the attack it addresses.

Moreover, in the case of ADTrees, the values associated with the components of one player may influence the values for the other player. For instance, if the **satisfiability** value of an action for the attacker is `True`, then it follows that the **satisfiability** of this action for the defender is `False`, and vice versa. A similar property holds for **probability**: if the attacker is successful at a node with probability p, then the defender is successful at this node with probability $1 - p$. However, such a relation does not exist for all attributes. For instance, when considered in isolation, the **attacker's costs** has no impact on the **defender's costs**.

Finally, properties quantified with Boolean values are well suited to model hypothetical scenarios. As an example, let us consider the attribute **electricity needed**, which evaluates to `True` at every component which requires electricity, e.g., cameras, and to `False` otherwise. Using this attribute, we can model what happens if we experience a power outage, by projecting the scenario on its part evaluated to `False`. This would allow us to check whether the existing defensive measures would also be sufficient in emergency situations.

3 An RFID-based Goods Management System

In this section, we describe the setting of RFID-based goods management system and the creation of a corresponding ADTree model. The system had already been subject of a threat assessment as a part of the EU-funded project SHIELDS [SHIELDS, 2008-2010]. Therefore, we already had attack trees and misuse case diagrams describing potential threats and countermeasures as an initial starting point. In order to extend this work and capture threats related to technical, physical and social engineering, we did a more thorough analysis of the conceptual design of the RFID-based goods management system, the physical layout of the warehouse where it is deployed and how people are involved in the work processes. We also made use of other relevant attack trees, such as Mirowski et al. [Mirowski et al., 2009], to supplement the creation.

In this case study we have chosen to provide detailed examples from the physical and social engineering world and illustrate the ADTree formalism on them. Obviously, this is not a restriction on the formalism, and the presented methodology can be readily applied in other fields, such as risk management or software engineering. We believe that it is easier to relate to these tangible examples than to fields that require expert knowledge to even understand the depicted scenario. In the following sections we draw up general purpose guidelines that can be easily modified to the field of application by adapting the set of suitable attributes as well as possibly changing their value domain.

3.1 Setting

In order to perform a realistic case study, we selected an already deployed and operational system named the *Warehouse Information Management System* (WIMS) with special focus on one of its components, the *Warehouse Loading Docks Management Application* (WaLDo). This system manages all incoming and outgoing goods to and from a warehouse, keeping track of orders, goods location, picking lists, shipping notifications, etc. The warehouse is a highly automated environment where all goods can be electronically identified using RFID tags. Figure 2 gives a high level overview of the system itself.

The WaLDo application controls all goods that cross the loading docks of the warehouse. The physical warehouse is equipped with RFID enabled loading docks. All RFID readers conform to the EPCGlobal specifications and are managed via an *Application Level Event* (ALE) service that provides a web service interface to upper layer applications like WaLDo. Additionally, the warehouse has an information management system able to interact with the company *Enterprise Resource Planning* (ERP) system, the integrated software application that manages the entire company information flow and resources, to process universal business language documents like *Picking Lists*, used to specify which material is to be shipped to whom, and *Advanced Shipping Notification* (ASN) documents, used to specify which goods are expected to be received.

In order to properly analyze potential threats, we also consider the environment in which the system operates. Figure 3 depicts the physical premises, the equipment and the workspaces inside the warehouse. The WaLDo application operates in a warehouse where eight employees are working. The size of the warehouse building is $500\,\mathrm{m}^2$. It contains RFID enabled forklifts, shelves for goods and three loading docks with RFID readers, which can only be opened from the inside. All goods pass in and out through the loading docks and are registered by the RFID readers. The building also has one room for computer servers, one administrative office, one security room containing two *Closed-Circuit Television* (CCTV) monitors and a fuse box, one bathroom, one corridor, a main entrance and an emergency exit. The warehouse is surrounded by a fence that encloses the entire area. The fence has two gates, one for trucks and one for employees, which can only be opened remotely from the security room. The area inside the fence has a parking place where trucks can wait before unloading their goods and where the employees can park their cars. The warehouse is equipped with a high-speed Internet connection and a wired LAN Ethernet. The Ethernet network connects the servers with the RFID readers of the loading dock. In total, there are seven surveillance cameras that are linked to external security services, monitoring both the inside and the outside of the warehouse. Cameras 1, 5 and 6 monitor the shelves within the

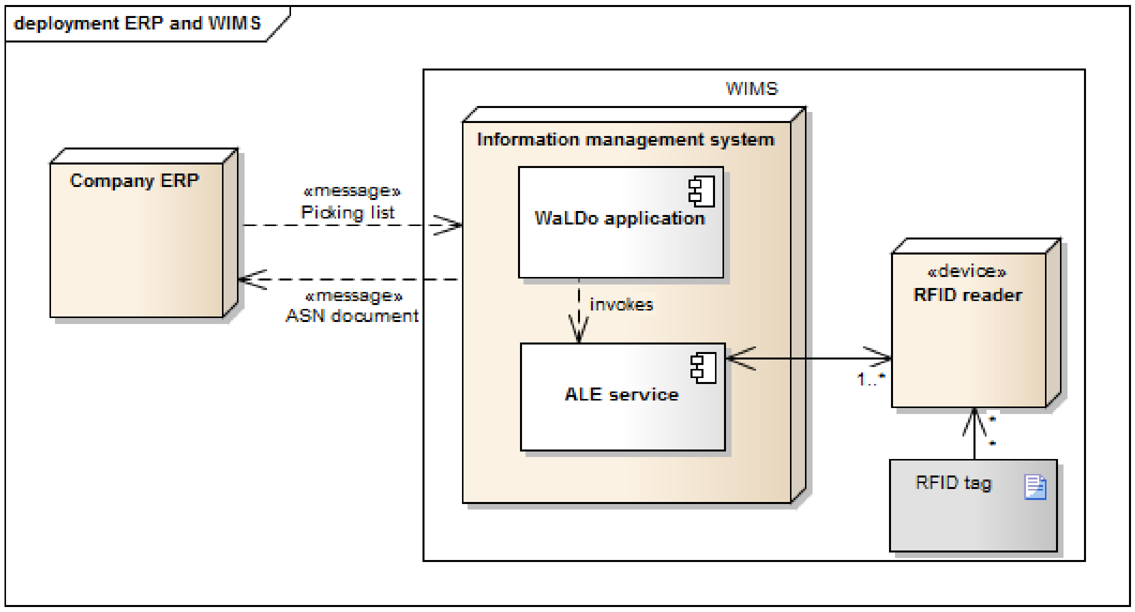

Figure 2: WIMS Deployment Diagram

WaLDo building, Camera 2 monitors the main entrance gates, Camera 3 monitors the parking areas, Camera 4 monitors the loading docks and Camera 7 monitors the warehouse's main door. Each day between 10 and 20 trucks deliver goods to or from the warehouse. The drivers load the goods on and of their trucks by accessing the warehouse through the docks. Though we are not specifying exactly what kind of goods are stored in the warehouse, we assume they are worth stealing (else the security assessment would be pointless).

3.2 The ADTree Model

The information given in the previous section served as a basis to create an ADTree that we then used for attribute decoration. First, an initial tree was suggested by one player. This tree was then independently examined by the other players who suggested improvements which were merged with the tree through several iterations. We limited the scope by focusing on one high-level attack–defense scenario, namely disrupting the RFID-based part of the system by preventing communication between a specific tag and a specific reader. Each player spent roughly three hours on the tree creation phase, later only minor refinements were necessary. We did not have an automated method of combining trees, however the trees were small enough to do a visual comparison in order to reveal missing or similar nodes.

The top goal node of the high level tree model, shown in Figure 4, is called "RFID DoS Attack". In order to achieve this goal, an attacker has six options. He can "remove the tag", "disable the tag", "overload the tag", "disable the reader", "disable the backend" or "block the communication" between all tags and all readers. Even though we initially refined all children, we chose to continue the case study by only refining the nodes "remove tag" and "block communication". The refinements are depicted in Figures 5 and 7. We deliberately chose to analyze an incomplete tree to reflect that, in most use cases, the modeling time is limited which invariably will lead to incomplete trees.

To physically remove the RFID tag an attacker can either remove the tag himself, or he can convince someone else to remove the tag. In the first case, he either can "infiltrate the building" or he has to "infiltrate the organization" and thereby gain legitimate access. Infiltrating the building can be achieved by "breaking and entering", as detailed in Figure 6, by "posing as a truck driver", by executing a "postal Trojan attack" or by staging a "visitor attack". A postal Trojan attack can be achieved when the attacker "hides in a box" and this box is sent to the warehouse. The

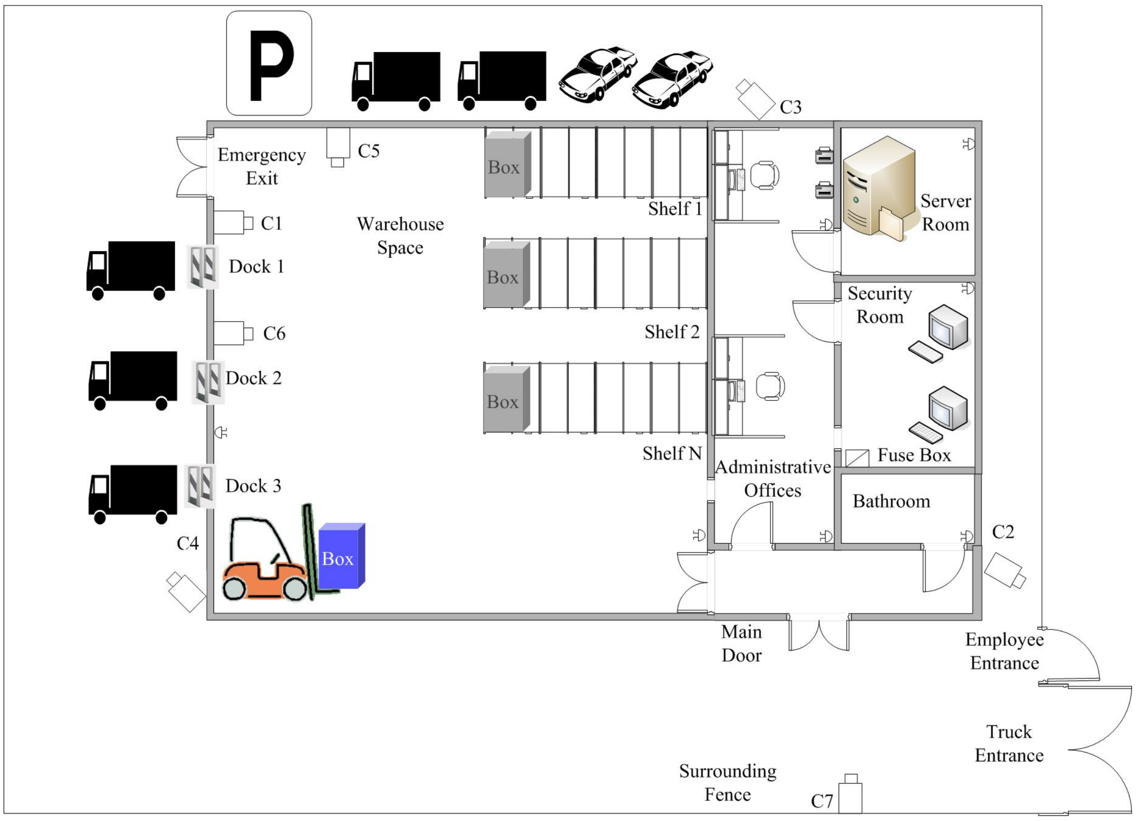

Figure 3: Floor plan

owner of the warehouse could defend against Trojan mail by employing a “sniffer dog” that can detect humans in the incoming goods. The attacker, in turn, could confuse the dog using decoy rats or pepper spray. If the attacker decides to execute a “visitor attack” he can “come as visitor” during daytime and “hide in the bathroom” until everyone else has gone home. The defender could anticipate such an attack and “track visitors” on the warehouse premises. Tracking the visitors can be accomplished by “escorting the visitors”, by requiring visitors to “register in a visitor’s log”, by using a more supervised attended visitor’s log, or by installing “presence detectors on the premises”. A visitor could chose to overcome the defense “register in a visitor log” by “faking a log entry”. In that case, the warehouse owner should switch to “register in an attended visitor’s log”. If the attacker decides that he wants to infiltrate the organization, he can try to “get hired as warehouse staff”, “pose as warehouse employee”, or simply “buy the warehouse” (we deliberately added some extreme nodes to the tree to try and provoke some extreme attribute values). The defender could protect himself against any infiltration by performing “background checks” on everyone he works with.

If the attacker chooses to convince someone else to removed the tag, he can “bribe”, “threaten”, “blackmail” or “trick” this person. In the first case, he has to “identify a corruptible subject” and then he has to actually “bribe the subject”. The warehouse owner could defend against bribery by “thwarting the employees” from receiving bribes, by providing mandatory “security awareness trainings” or ”threatening to fire the employees” in case of infringement. Provided the attacker wants to trick another person into removing the tag, he can either “send false replacement tags” or he can place a “false management order” to replace the tags. Fake orders can be done by “infiltrating the management” and “ordering replacement tags”. A defender can prevent this kind of trickery by mandatory “security awareness training courses”. Last, a defender could prevent any kind of removal of the RFID tag by using a “stronger adhesive”, i.e., attaching the tag in a way

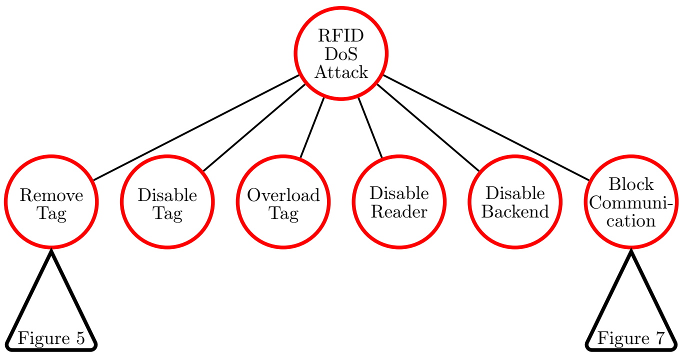

Figure 4: An ADTree for RFID DoS attack

that it cannot be removed.

If an attacker decides to remove the tag himself by breaking and entering he must ”get onto the premises” and “get into the warehouse”, undetected by the installed cameras. To get onto the premises, an attacker can “climb over the fence” or he can “enter through the gate” for employees undetected by Camera 2. To prevent attackers from climbing over the fence, the defender could install “barbed wire” on the fence. An attacker, in turn, could circumvent the barbs by guarding against them, which he could achieve by either throwing a “carpet over the barbs” or by “wearing protective clothing”. The attacker also has to get into the warehouse. He can accomplish that by “entering through the door” undetected by Camera 7 or “entering through the loading dock” undetected by Camera 4. The defender could prevent an attacker from entering through the main door by monitoring the door with biometric sensors. Another defensive measure would be to install and monitor the premises with additional security cameras. These new cameras would monitor the parts of the property not yet covered, but could be rendered useless if an attacker disables them. Disabling could be done by shooting a strong laser at the cameras or by “video looping the camera” feed. Alternatively, guards patrolling the premises could protect against this kind of threat.

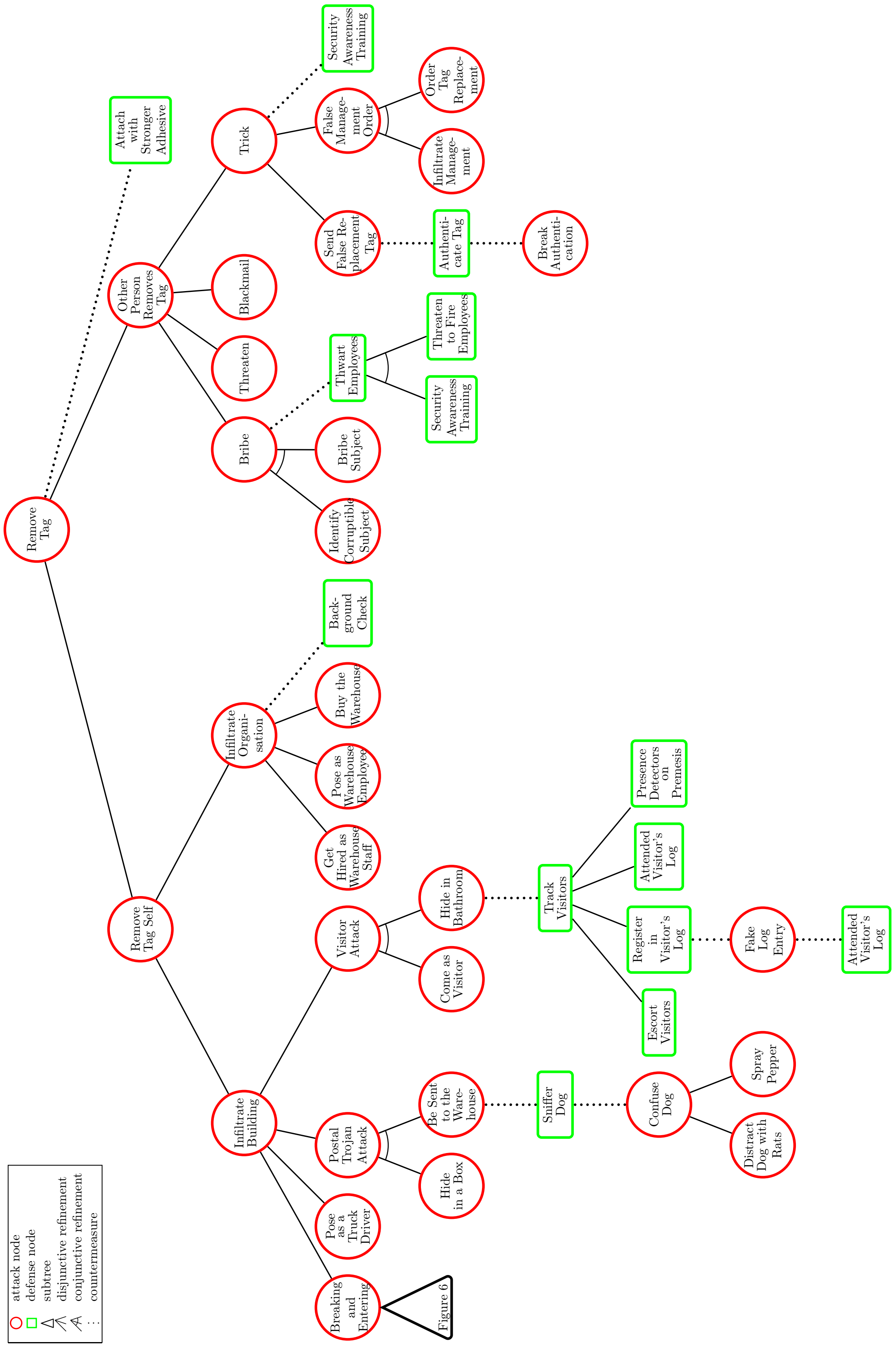

Figure 5: The remove tag subtree

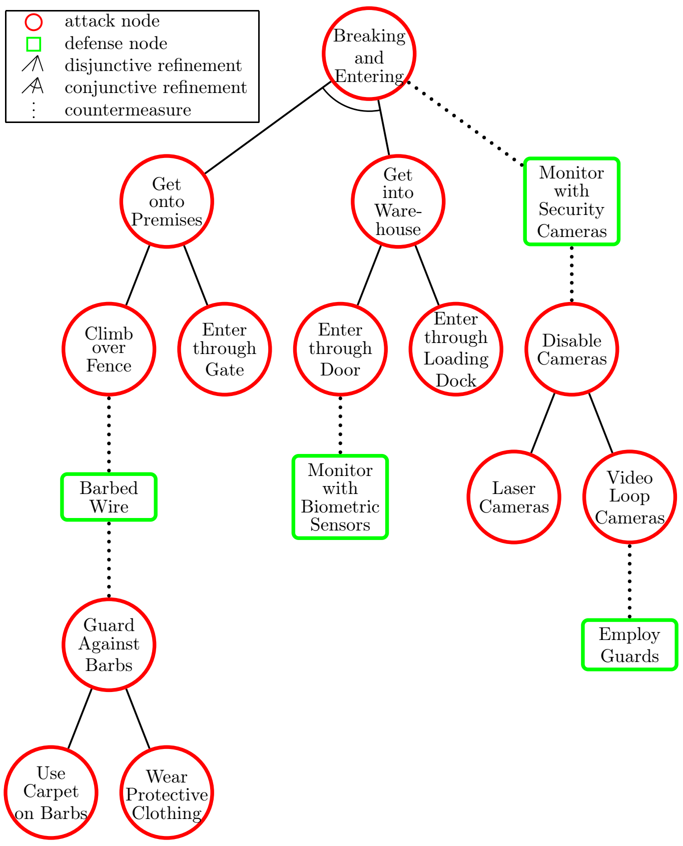

Figure 6: The breaking and entering subtree

Blocking communication can be done by blocking the communication between the tag and the reader or by blocking communication between the reader and the backend, as depicted in Figure 7. To do the former, there exist several options: It is possible to "shield the tag", to use a malicious reader that constantly requests information from the tag and this way "blocks the tag", to use a different tag that "blocks the reader", or to "jam all signals". Shielding a tag can be achieved by "being in the vicinity of the tag" and by using a "Faraday cage". An obvious defense against attackers being in the vicinity of the tags would be to increase the "security of the warehouse". A Faraday cage can be installed around the reader or around the tag. To prevent attackers from jamming the signal, the defender could "isolate the entire warehouse network", which could be achieved by "securing the warehouse" or "encasing the entire warehouse inside a Faraday cage". If the attacker decides to block the communication between the reader and the backend, he can achieve it by evoking "DoS in the wired network".

4 Attribute Case Study

Having established our ADTree, we are ready to focus on the novelty of the case study: the decoration of the ADTree with attributes and the corresponding values. Figure 8 gives an overview on how this case study was performed. As explained in Section 3, the *ADTree creation* activity was done through several iterations. This was followed by an *attribute decoration* activity, which consisted in selecting a relevant set of attributes, choosing players and estimating attribute values. Section 4.1 explains this part in more detail, while Section 4.2 describes the objective observations we recorded. The next step was to prepare the attribute values for calculation, as explained in Section 4.3. Finally, the bottom-up algorithm was applied in order to derive the minimal **costs** of the attacker in our scenario. This step as well as the corresponding findings are presented in

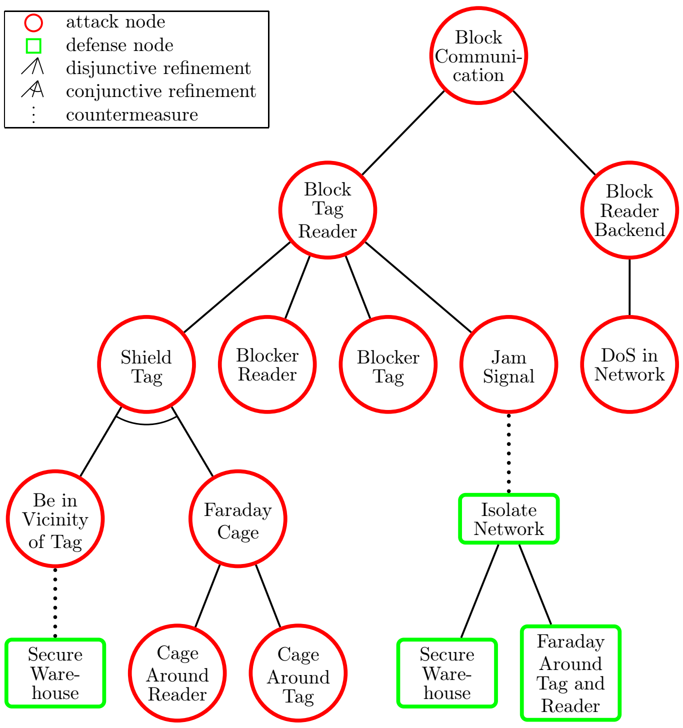

Figure 7: The block communication subtree

Section 4.4.

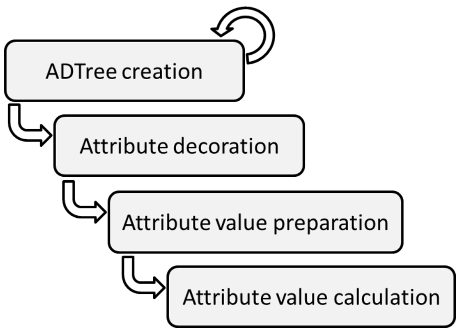

Figure 8: The ADTree case study process

4.1 ADTree Decoration with Attributes

In this case study, we have experimented with a game-based approach for the decoration of the ADTree. We chose a set of attributes that we felt were useful and possible to provide accurate values for (in other studies a different set might be more suitable). The selected attributes with their detailed descriptions, example references and the corresponding value domains can be found in Table 1. In many cases there are variants of the attribute names in the literature, e.g., we have used the term **impact** to cover **impact, severity, consequence, damage, criticality**, as well as **seriousness**. Most of the chosen attributes can be applied to both attack and defense nodes. We also decided to make use of the meta-attribute **confidence**, with the domain $\{1, \ldots, 5\}$, where 1 represents total lack of confidence and 5 very high confidence.

There were two players taking roles as attackers, and two as defender. The players estimated the attribute values for the nodes of the ADTree. For this purpose, we created an empty table with 9 columns for the attributes and 79 rows for all nodes. Our intention for the table was to prevent illegible values, that would have occurred if the players had estimated all values directly on the ADTree printed on a sheet of paper. All players were allowed to use the labeled ADTree, the warehouse and the system description given in Section 3.1, the attribute description from Table 1 and the empty table. The values were estimated independently over a period of one week, but we did not set a specific time limit. Neither did we require to estimate values for every node and attribute, we rather suggested to apply a best effort strategy.

Table 1: Decoration attributes for the RFID case study

Attribute	Description	Values
Costs [Schneier, 1999, Buldas et al., 2006, Tanu and Arreymbi, 2010, Baca and Petersen, 2010, Saini et al., 2008, Mauw and Oostdijk, 2005, Yager, 2006, Abdulla et al., 2010, Roy et al., 2011, Byres et al., 2004, Amenaza, 2001, Wang et al., 2011, Edge et al., 2006]	The amount of real money needed to finance the attack or defend against it (depending whether it is the attacker's of defender's point of view), referring to e.g., equipment or software costs, educational expenses, development costs or size of a bribe.	`Cheap` (C): Any attacker or defender can afford this without thinking twice. `Average` (A): The costs of the attack will fend off most attackers without a steady income. Defenders will typically do a cost-benefit analysis before the expenses can be justified. `Expensive` (E): The attacker will need substantial funding in order to perform the attack. It is unlikely the defender will invest in this. Optional: Real number value.
Detectability [Tanu and Arreymbi, 2010, Byres et al., 2004, Amenaza, 2001]	The chance that the defender will notice the attack during its execution or the attacker will notice the defense mechanism.	`Easy` (E): Any attacker or defender with a clear state of mind will detect that something was out of the ordinary right away. `Possible` (P): Some attackers or defenders are able to detect this defense or attack. `Difficult` (D): Very few attackers or defenders are qualified to notice that something is wrong before it is too late.
Difficulty [Byres et al., 2004, Fung et al., 2005, Tanu and Arreymbi, 2010, Henniger et al., 2009, Amenaza, 2001, Mauw and Oostdijk, 2005, Abdulla et al., 2010, Amoroso, 1994, Wang et al., 2011]	The technical or social skill level needed for the attacker or defender to succeed.	`Trivial` (T): Little technical skill required. `Moderate` (M): Average cyber hacking or defense skills required. `Difficult` (D): Demands a high degree of technical expertise, the attacker is a professional con artist. `Unlikely` (U): Beyond the known capability of today's best attackers or defenders.

Table 1: Decoration attributes for the RFID case study

Attribute	Description	Values
Impact [Schneier, 1999, Tanu and Arreymbi, 2010, Henniger et al., 2009, Li et al., 2009, Saini et al., 2008, Mauw and Oostdijk, 2005, Amoroso, 1994, Abdulla et al., 2010, Roy et al., 2011, Edge et al., 2006, Wang et al., 2011]	The severity or consequence from the system owner's point of view. Can refer to loss of money, but also other less tangible resources such as loss of reputation.	`Low` (L): The system owner will not care or notice. `Moderate` (M): Acceptable but unwanted loss. `High` (H): Unacceptable loss, must be avoided. `Extreme` (E): Will terminate business. Optional: Real number value.
Penalty [Buldas et al., 2006, Jürgenson and Willemson, 2008, Wang et al., 2011]	The consequences for the attacker given that the attack fails, for instance, a fine, jail sentence or being blacklisted. Here, we do not consider any penalty of a successful attack.	`Low` (L): The attacker will not care. `Medium` (M): The attacker will think twice before performing the attack. `High` (H): Very few attackers will take the risk. Optional: Real number value of a fine or years in prison.
Profit [Amoroso, 1994, Jürgenson and Willemson, 2008, Bistarelli et al., 2006, Roy et al., 2011]	The economic profit or gain the attacker will receive should the attack succeed. This value does not include costs of attack.	`None` (N): A successful attack does not lead to any direct income. `Marginal` (M): Economic gain is not enough by itself to justify the attack. `Lucrative` (L): The attacker can obtain a substantial profit. Optional: Real number value.
Probability [Schneier, 1999, Buldas et al., 2006, Henniger et al., 2009, Li et al., 2009, Manikas et al., 2011, Yager, 2006, Abdulla et al., 2010, Roy et al., 2011, Byres et al., 2004, Edge et al., 2006, Wang et al., 2011]	The assumed chance that the attack or defense will succeed. Could be based on heuristics of similar attacks or cognitive estimations.	`Unlikely` (U): Below 5%. `Low` (L): Between 5% and 25%. `Medium` (M): Between 25% and 75%. `High` (H): More than 75%. `Certain` (C): Close to 100%. Optional: Specific percentage value.
Special skill [Mauw and Oostdijk, 2005, Abdulla et al., 2010, Schneier, 1999]	A specified skill or property the attacker or defender will need in order to succeed. This is orthogonal to the **difficulty** attribute. Examples are access to insiders or need of electricity.	Binary value: `True` (T): A special skill is required. `False` (F): No special skill is required.

Table 1: Decoration attributes for the RFID case study

Attribute	Description	Values
Time [Henniger et al., 2009, Schneier, 1999, Wang et al., 2011]	For the attacker this is the time needed to perform the attack, independent of **difficulty** and **costs of attack**. For the defender this is the time needed until the defense is effective.	`Quick` (Q): The attack or defense can be performed in an instance. `Medium` (M): The attacker or defender will need to be patient and wait for a while. `Slow` (S): The attack or defense takes really long time to complete. Optional: Real number in terms of minutes.

4.2 Observations from the Attribute Decoration

The entire ADTree consisted of 79 nodes, where 59 were attack nodes and 20 were defense nodes. An extract of the resulting estimated values is shown in Table 2, where each line for an attack node represents a player who had been given the role of an attacker; similarly for lines representing defense nodes. A dash indicates a conscious decision not to estimate a value and an empty field indicates that the player was not considering estimating a value. The letters [D] and [B], inserted in front of the node labels, indicate that the node was a defensive node and a basic action, respectively.

None of the players used a real number values for any of the attributes, instead all four players used the ranged values given in Table 1. In Table 2, the first letter of these ranges has been used to indicate the value, and the following number gives the confidence value.

The two players who were initially given the role of attackers spent approximately 90 min and 120 min, the players with the role of the defender spent 40 min and 90 min.

We would like to mention that two of the players consulted the ADTree to reexamine the context of the nodes, whereas the two other players estimated the values without taking existing values of similar nodes or values of the parents and children into account.

The percentage of the nodes, where both players have given values, ranges between 0 % and 93 %, for different attributes. For five attributes (**detectability**, **impact**, **penalty**, **profit** and **probability**) either an attacker or a defender did not estimate a value. For the **special skill** attribute, we discovered that the attackers had not considered the same special skill. One defender estimated values only for basic actions and not for refined nodes.

In the remainder of this section, we elaborate on the difficulties we encountered while estimating the values. We have grouped the difficulties into *attribute related*, *node related* and *methodological* problems. They are based on our analysis of the data and the players' own evaluation.

Table 2: Extract of raw-data of the case study. Each row represents the estimates of one player. The first letter of every field represents an attribute value, as abbreviated in Table 1 and the second represents the confidence level on a scale from 1 to 5.

Name of the Node	**Costs**	**Det**	**Diff**	**Imp**	**Pen**	**Prof**	**Prob**	**Skill**	**Time**
Breaking and Entering	A,3	P,3	M,3	M,3	M,3	-	-	F,3	Q,3
	-		M,4		-		M,4	F,4	Q,4
Get onto Premises	C,4	P,4	T,4	M,4	M,4	-	-	F,4	Q,4
	C,4		T,4		L,4		H,4	F,4	Q,4
Get into Warehouse	C,4	P,4	T,4	M,4	M,4	-	-	F,4	Q,4
	C,3		M,3		M,3		H,3	F,3	Q,3
[D,B] Monitor with	E,4	P,4	T,4		H,4			F,4	S,4
Security Cameras	E,4	E,4	M,4				M,3	T,4	Q,3

Table 2: Extract of raw-data of the case study.

Name of the Node	**Costs**	**Det**	**Diff**	**Imp**	**Pen**	**Prof**	**Prob**	**Skill**	**Time**
[B] Climb over Fence	C,5 C,4	D,5	T,5 T,4	L,5	L,5 L,4	-	M,5 H,4	F,5 F,4	Q,5 Q,4
[B] Enter through Gate	C,5 C,4	E,5	T,5 M,4	L,5	L,5 M,4	-	H,5 M,4	F,5 F,4	Q,5 Q,4
[B] Enter through Door	C,5 C,4	E,5	T,5 M,4	M,5	M,5 M,4	-	M,5 M,4	F,5 F,4	Q,5 Q,4
[B] Enter through Loading Dock	C,5 C,4	E,5	T,5 M,4	M,5	M,5 M,4	-	L,5 M,4	F,5 F,4	Q,5 Q,4
Disable Cameras	A,3 C,3	E,3	T,3 M,3	L,3	M,3 L,3	-	M,3 -	F,3 F,3	Q,3 M,3
[D,B] Barbed Wire	E,4 C,4	D,4 E,4	D,4 T,4		H,4		 M,3	T,4 T,3	S,4 Q,2
[D,B] Monitor with Biometric Sensors	E,4 E,4	D,4 E,3	D,4 M,3		H,4		 M,3	T,4 T,3	S,4 Q,2
[B] Laser Cameras	A,3 A,2	E,3	M,3 M,2	L,3	M,3 L,2	-	L,3 M,2	F,3 F,2	Q,3 Q,2
[B] Video Loop Cameras	A,2 A,2	D,2	D,3 M,2	L,2	M,2 L,2	-	U,2 L,2	T,2 F,2	M,2 M,2
Guard Against Barbs	C,4 A,4	-	T,4 T,4	L,4	L,4 L,4	-	H,4 H,4	F,4 F,4	Q,4 Q,4
[D,B] Employ Guards	A,4 E,4	P,4 E,4	T,4 T,4		H,4		 M,3	F,4 F,4	M,4 M,2
[B] Use Carpet on Barbs	C,5 C,4	E,5	T,5 T,4	L,5	L,5 L,4	-	C,5 H,4	F,5 F,4	Q,5 Q,4
[B] Wear Protective Clothing	A,5 A,4	E,5	T,5 T,4	L,5	L,5 L,4	-	H,5 H,4	F,5 F,4	Q,5 Q,4
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Nodes with less than two values	11	65	6	79	24	79	47	9	12
Attack nodes with two values	54	0	59	0	55	0	32	56	53
Defense nodes with two values	14	14	14	0	0	0	0	14	14

Attribute Related Problems. Several problems, encountered while estimating values, are worth mentioning. First of all, we learned that the players tended to estimate values by using their own interpretation of the attributes and not the one provided in Table 1. This happened when the description given in Table 1 was not immediately understandable or when it was contradictory to the player's belief of what the attribute represented. Apparently, we forgot to mention that *only* the attribute description given in the table should be used. For example, the attribute **costs of attack** or **costs of defense** to "disable cameras" could be understood from two perspectives. From an attacker's point of view, it could mean how much the attacker has to pay to disable the cameras. From a defender's point of view, however, the **costs** of this attack could be the money the defender has to pay to get the cameras running again after the attack.

Table 1 describes **penalty** and **profit** only from the perspective of an attacker that performs an attack. Slightly different, the **impact** attribute describes consequences from a system's owner or defender's point of view, the scale given in Table 1 however indicates attacks. When attackers only estimate values for attack nodes and defenders only for defense nodes, both descriptions led to nodes which did not have any values.

Even when Table 1 did not make a reference to only the defender or the attacker, some players believed that a certain attribute only concerns the other player's actions. As a result, the **detectability** attribute was interpreted by an attacker as detectability of the defense, and

probability was interpreted by a defender as the probability that the attack succeeds. This again led to nodes without estimated values.

A third source of missing values was the description of the attribute itself. This occurred for the attributes **profit** and **special skill**. More concretely, the **profit** attribute is missing information on what can be gained with the attack. For example, we should have mentioned whether we plan to steal valuables from the warehouse or to annoy people that work there. Furthermore, the special skill description allows for a use of a Boolean domain, but does not specify an attribute such as whether or not **insider knowledge**, **electricity**, or certain **technical skills** are required. Table 1 offers an explanation for the different attribute value categories. However in some cases, the given category explanation was not precise enough, or only the keyword classifying the category was taken into account. For example, depending on the node, the value `Medium` for the attribute **time** can be understood as anything ranging from minutes to weeks. Indeed, consider the nodes "hide in bathroom" and "get hired": *being patient for a while* could mean *a couple of days* when we were hiring someone, but it would mean *a couple of minutes* if we were hiding in the toilet.

For some attributes, e.g., **costs**, **probability**, **time**, it felt easier to estimate the more refined nodes, while for other attributes, e.g., **penalty**, **profit**, it was easier to estimate the less refined nodes. As a result, some players chose to only estimate attribute values they were moderately sure about. When uncertain about an attribute value, they favored not to estimate over estimate a value with low confidence.

The description and the use of the meta-attribute confidence should have been described more clearly. One of the players chose to only use one confidence level per node, with the intention to save time at the expense of less accurate values. Other players selected a different confidence level for every estimated attribute value. All players concluded that the confidence level scale should be reduced to fewer (e.g., three) options.

Node Related Problems. Issues directly concerning attributes and their specifications are not the only reasons why associating values with the nodes of an ADTree was a difficult task. Several problems were in fact related to the nodes themselves. Associating attribute values with the nodes helped us to realize that the user's understanding of the presented scenario is in many cases incomplete or even incorrect. One of the main problems was that node labels were often not self-explanatory and may lead to confusion. We used simple labels in order to be able to graphically represent the nodes. Thus, the labels were often short, e.g., "false management order", imprecise, e.g., "blocker reader", or did not contain verbs, e.g., "barbed wire". This implied that, without looking at the context in which the nodes had been used, i.e., parent, sibling and child nodes as well as the corresponding main goal, it was impossible to estimate the related attribute values. As an example, we can consider the "enter through door" node. Without taking its parent node "get into warehouse" into account, it is impossible to estimate the corresponding values for the **time** and **difficulty** attributes, because the values may differ depending on which door we are interested in: the warehouse door, the bathroom door or the administrative office door.

Another issue is whether attribute values should be assigned to the non-refined nodes only or also to the refined ones? This problem is related to the meaning of refined nodes. In fact, some of them represent understandable attacks or defenses, e.g., "get onto premises" or "block tag reader", others play the role of dummy placeholders, e.g., "trick". In the first case, it was possible to associate values with such a refined node. In the latter case, they were only used to connect several options that could be attacked or defended in the same way. In such situations, the attribution was more problematic and could not be performed without taking the corresponding child nodes into account.

Finally, in order to quantify some of the considered properties, additional information may be required. Such information does not have to be related to the considered ADTree structure. For instance, it is hard to decide how long it takes to employ guards, without knowing what kind of goods are stored in the warehouse. If the goods are not very valuable, anybody could be hired as a guard. Thus, the execution **time** of "employ guards" would be `Medium`. However, if the goods are expensive, sensitive or dangerous for the environment, the guards have to be chosen more carefully.

In this case, the selection process would take longer, because it would have to be accompanied by additional measures such as background checks. Thus, the corresponding attribute value would then be `Slow`.

Methodological Problems. The case study was performed by four people: two of them played a role of the attacker and two of the defender. However, it was not explicitly specified to which nodes the players should assign the values. This led to inconsistencies in the gathered data. Each of the players provided values for nodes related to his or her role, but one of the attacker players also estimated some values for the defender nodes. So a question arises: Which player should provide values for which nodes?

This issue is closely related to the problem of what the knowledge of the player is. Should we assume that the attacker and the defender are only able to estimate values for their own actions, or should the game give them the freedom to assign values to the adversary's nodes as well? Furthermore, should the players only take the part of the scenario corresponding to their role into account, or can they base their decisions on knowledge about the other player too?

Another issue is how to assign values to the nodes that have the same labels. Nodes such as "secure warehouse" or "attended visitor's log", were mentioned twice in the initial table, because they appear twice in Figure 5. On the one hand, the values assigned to two different occurrences of the same action might be different in the case when the context is taken into account. On the other hand, if the nodes are handled independently of the tree, it would be more reasonable to associate the same values with similarly labeled nodes.

We also identified an issue concerning the role of the players in the creation phase of the considered ADTree. As we were the creators of the tree, we had a good understanding of the tree. However, in general players might be asked to estimate values for trees which they have not seen before.

4.3 Preparation of Attribute Values

The attribute values have been estimated by four players. As a result we have obtained several values for a given attribute and a given node, as described in Table 2. In practice, we would like to have a single value that can give us some indication about the aspects we are interested in. In this section, we show how a single value can be derived from the raw data gathered in the previous step of the case study.

We start from the data provided by the players, as given in Table 2. If every player has estimated the same value, we can immediately use it as input to produce an indicator for the scenario. In practice, this perfect world scenario seldom occurs. There are several reasons why the values were not identical. First, some players may have opted not to estimate a value at all. Second, the understanding of the node may have been different for each player, as we pointed out in Section 4.2. Finally, it might also happen that no player has estimated a value for a node. Therefore, in a non-perfect world scenario, we need a method to choose one representative value for each node.

Provided all players had estimated a value for a given node, we automated the process of combining the values by using the following procedure. First, we transformed the attribute values into natural numbers. E.g., the **costs** attribute values `Cheap`, `Average` and `Expensive`, were transformed into $1, 2$ and 3, respectively. We let n be the number of independently gathered pairs $(\text{attribute}, \text{confidence})$, i.e., in our case n was equal to 2. Then, we used the *formula*

$$(\text{rnd}\left(\frac{\sum \text{attribute} \cdot \text{confidence}}{\sum \text{confidence}}\right), \left\lfloor \frac{\sum \text{confidence}}{n} \right\rfloor), \tag{1}$$

where $\lfloor \cdot \rfloor$ symbolizes rounding down to the nearest integer and rnd regular rounding. We chose to round the first component to use the best estimate. This rounding also allows us to transform the first component back into the original categories. We chose to round the second component down, to reflect risk averseness. The results of the application of the formula are the non-bold values in Table 3.

To illustrate the application of Formula (1), we combine the **difficulty** attribute values estimated for the "Enter through Gate" node. From Table 2 we can see that one of the players estimated the **difficulty** of entering through the gate as being `Trivial` (T) and his confidence level in this value was 5. The other player found the considered action as `Moderate` (M) in **difficulty** and his confidence level in this value was 4. The value domain for the **difficulty** attribute, as defined in Table 1, contains four values: `Trivial` (T), `Moderate` (M), `Difficult` (D) and `Unlikely` (U), which we transform into $1, 2, 3$ and 4, respectively. Thus, we use $(1, 5)$ and $(2, 4)$ as inputs for Formula (1). We obtain:

$$(\mathrm{rnd}\left(\frac{1 \cdot 5 + 2 \cdot 4}{5 + 4}\right), \left\lfloor \frac{5 + 4}{2} \right\rfloor) = (1, 4).$$

This means that, after combining the values estimated by the players, we obtain that the **difficulty** of "entering through the gate" is `Trivial` (T) (because of the 1), of which we are certain with a confidence level of 4. The **cost**, **difficulty**, and **time** values, as well as the corresponding confidence levels, obtained for the remaining nodes by applying Formula (1), are depicted as non-bold pairs in Table 3. We defer a discussion of other possible methods to combine values to Section 5.2.

In our case study, we encountered several exceptional cases, when using the formula was either impossible, because, for example, not a single value was given, or doubtful, because the values differed substantially. These critical cases we classified and discussed at a *consensus meeting.*

The main goal of the consensus meeting was to obtain values that can then be used as unbiased input for further calculations on the ADTree. For example, we reduced the unbalance that would be introduced if we considered nodes with four times the same value equal to those where only one of the players had estimated this value. Since for several attributes we did not have enough data, we focused only on the attributes **costs**, **difficulty**, and **time** at the consensus meeting.

We first ensured that the nodes were correctly classified, i.e., that the players had not mistakenly estimated wrong value. Whenever mistakes were discovered, they were corrected, and the nodes were reassigned to the correct category, before the actual conflict resolution started. We also uncovered one inconsistency where the scenario and the tree were not matching. To repair this mistake, we corrected the tree. To obtain agreed values, the nodes were analyzed in context, i.e., we looked at the parents and the children of the nodes, but without considering any values assigned to these nodes. More concretely, we identified the following categories and resolved the problems in the following way

- *Nodes where no one had estimated a value*: We opted to discuss the value and eventually assign a single value. The players who had taken the opposite role commented on plausible values, then we selected one with which we all agreed.
- *Nodes where not every player had estimated a value*: We also decided on a single value at the consensus meeting. Concretely, the player who had not given a value, first explained why he had not done so, then the player who had given a value explained his choice. Then, a consensus was formed.
- *Nodes that had non-neighboring values*: The player with the lower value explained his choice, then the player with the higher value explained his. After that, the involved players agreed on one of the given values, or a compromise was chosen. Whenever a compromise was chosen, we lowered the confidence value.
- *Nodes where all given values have low confidence levels*: We also planned to discuss these uncertain values, but we ran out of time, skipped this step and applied Formula (1).

The final results of the consensus meeting are given as pairs in **bold font** in Table 3. Instead of the allocated hour, we spend two hours discussing the values. Out of the 3×79 possible values, there were 188 cases for which we applied the Formula (1), 8 cases without any assigned value, 24 cases to which we had assigned only one value and 17 cases where the values diverged significantly.

Table 3: The table after the consensus meeting. The first letter of every field represents an attribute value, as abbreviated in Table 1 and the second represents the confidence level on a scale from 1 to 5. The letters preceding the node names denote basic actions [B] and defensive actions [D].

Name of the Node	**Costs**	**Diff**	**Time**
Breaking and Entering	**A,3**	M,3	Q,3
Get onto Premises	C,4	T,4	Q,4
Get into Warehouse	C,3	T,3	Q,3
[D,B] Monitor with Security Cameras	E,4	M,4	**S,4**
[B] Climb over Fence	C,4	T,4	Q,4
[B] Enter through Gate	C,4	T,4	Q,4
[B] Enter through Door	C,4	T,4	Q,4
[B] Enter through Loading Dock	C,4	T,4	Q,4
Disable Cameras	A,3	M,3	Q,3
[D,B] Barbed Wire	**A,4**	**T,4**	**S,4**
[D,B] Monitor with Biometric Sensors	E,4	D,3	**S,5**
[B] Laser Cameras	A,2	M,2	Q,2
[B] Video Loop Cameras	A,2	D,2	M,2
Guard Against Barbs	A,4	T,4	Q,4
[D,B] Employ Guards	E,4	T,4	M,3
[B] Use Carpet on Barbs	C,4	T,4	Q,4
[B] Wear Protective Clothing	A,4	T,4	Q,4

4.4 Bottom-up Calculation of Attribute Values

As already mentioned in Section 4.2, assigning relevant values to all nodes of an ADTree may be difficult or even impossible. Fortunately, the ADTree methodology allows us to automate the calculation of values on ADTrees with the help of the bottom-up procedure. To use the bottom-up evaluation, we first have to initialize values at all non-refined nodes of the tree. Then, the values for all subtrees, and in particular for the entire tree, are calculated, using functions which depend on the type of the root of the subtree and the considered attribute. In this section, we first show how to calculate, in the bottom-up way, the values for the minimal **costs** of the attacker. Then, we use the obtained values to analyze the warehouse scenario.

In the case study, we were interested in calculating the minimal **costs** of a successful attack in the RFID warehouse scenario. We considered the situation where the attacker did not have any precise information on how the defender will decide to protect the warehouse. Thus, for this calculation, we assumed that all possible defenses illustrated on the ADTree were in place and that they were fully functional, i.e., a defense attached to an attack node defeats the corresponding attack component, unless the defense itself is rendered useless by a counterattack.

We started by initializing the values for the non-refined nodes of the tree. In the case of the attacker's non-refined nodes, we used the pairs (costs, confidence) from Table 3 as initial values. As the defender's **costs** do not have influence on the attacker's **costs**, we did not use the values from Table 3 in the case of the defender's non-refined nodes. Instead, we introduced an additional cost value `Infinite`, denoted by X, which represents infinite **costs** and we assumed the attacker is not able to afford it. The non-refined nodes of the defender were initialized with the pair $(X, 5)$. This indicates that we are fully confident that it is infinitely expensive (and thus impossible) for the attacker to be successful at a defender's action. Such initial values allow us to express the **costs** of the considered scenario from the point of view of the attacker.

Since we were interested in the minimal **costs**, we have to know how to compare different values. Thus, we considered the following linear order `Cheap` < `Average` < `Expensive` < `Infinite`. Now, we are ready to describe how we calculated the minimal attacker's **costs** for all subtrees. The bottom-up procedure is recursive, i.e., we start from the leaves and we calculate the value for every subtree rooted in a parent node based on the values previously calculated for the subtrees rooted in its child nodes.

In this framework, we chose the minimal value for an attacker disjunctive subtree, because we were interested in the minimal **costs**. Thus, we supposed that the attacker always performs the least expensive option. Moreover, we assumed that performing several actions belonging to the same **costs** category is not more expensive than performing one of such actions. Therefore, we chose the maximal **costs** in the case of a conjunctive subtree with the attacker's root. Conversely, in order to successfully disable a disjunctively refined defensive countermeasure, the attacker has to disable all corresponding refining options. Therefore, we used the maximum operator in this case. On the other hand, to successfully disable a conjunctively refined defensive countermeasure, it is sufficient for the attacker to disable only one of the corresponding refining actions. Here again, according to our assumption, the attacker chooses the least expensive solution. Thus, the operator used in this case is minimum. Finally, we always propagated the maximal confidence level, corresponding to the chosen **cost** value. This allowed us to express how certain we can be about this value.

The three paragraphs below summarize which operators are used for calculation of **costs** values for all possible subtrees.

Subtrees rooted in a node which is refined but not countered

- The **costs** calculated for a subtree rooted in a disjunctively refined attack (resp. defense) node is defined as the minimum (resp. maximum) of the **costs** calculated for its refining subtrees.
- The **costs** calculated for a subtree rooted in a conjunctively refined attack (resp. defense) node is defined as the maximum (resp. minimum) of the **costs** calculated for its refining subtrees.

The maximal confidence level corresponding to the chosen **costs** is propagated.

Subtrees rooted in a node which is not refined but countered The **costs** calculated for a subtree rooted in an attack (resp. defense) node is defined as the maximum (resp. minimum) of the initial value for the non-refined root node and the value calculated for the countering subtree. The maximal confidence level corresponding to the chosen **costs** is propagated.

Subtrees rooted in a node which is refined and countered In this case, first a pair corresponding to a refining part of the tree is calculated, as in the case of a subtree rooted in a refined but not countered node. Then, the functions for a subtree rooted in a non-refined but countered node are used, where the initial value for the root is replaced with the calculated value for the refining part.

We would like to remark that the functions used to calculate values depend on the considered attribute and additional assumptions. Thus, if we would be interested in calculation of the defender maximal **costs**, for instance, the used functions would have to be redefined accordingly. It is easy to see that the functions presented in this section are also suitable for calculation of the minimal **difficulty level** of the attacker and the minimal **time of an attack**, under the assumption that all defensive components are present and fully functional.

With the assumption that all the possible defenses are present and fully functional, the real minimal **costs** of a successful attack can be lower than the one obtained using our calculation. Indeed, in reality, the defender may decide not to implement some of the defenses and thus the **costs** of the corresponding counterattacks will not be taken into account for the final **costs** of the attacker. However, by taking the described approach we use a safe solution, in the sense that

- the calculated minimal **costs** will not be lower than the actual minimal **costs**, i.e., the minimal **costs** will not be underestimated,
- and the resulting set of attack components that have to be executed in order to achieve the cheapest attack forms a successful attack which, according to our scenario, the defender cannot defend against.

In the rest of this section, we use the calculated values for minimal **costs**, minimal **difficulty** and minimal **time** to analyze the RFID-based warehouse scenario. These attributes serve as representative examples. Our calculation shows that, to achieve an attack with the minimal **costs**, an attacker needs to spend `Average` amount of money. We have a confidence level 3 in this value. The corresponding attack consists of "disabling the backend". For the attack of minimal **difficulty**, the attacker should also "disable the backend". Its **difficulty** is `Medium` of which we are confident with level 3. The **time** it takes to perform the fastest attack is `Quick` of which we are confident with level 2. To achieve the fastest attack, an attacker should "disable the RFID tag". We observe that for all three attributes, the optimal attack option is something which we chose not to refine, see Figure 4. To be able to give a more insightful example, we look at the subtree rooted in the node "breaking and entering". The values resulting from the bottom-up approach for the **costs** attribute are depicted in Figure 9.

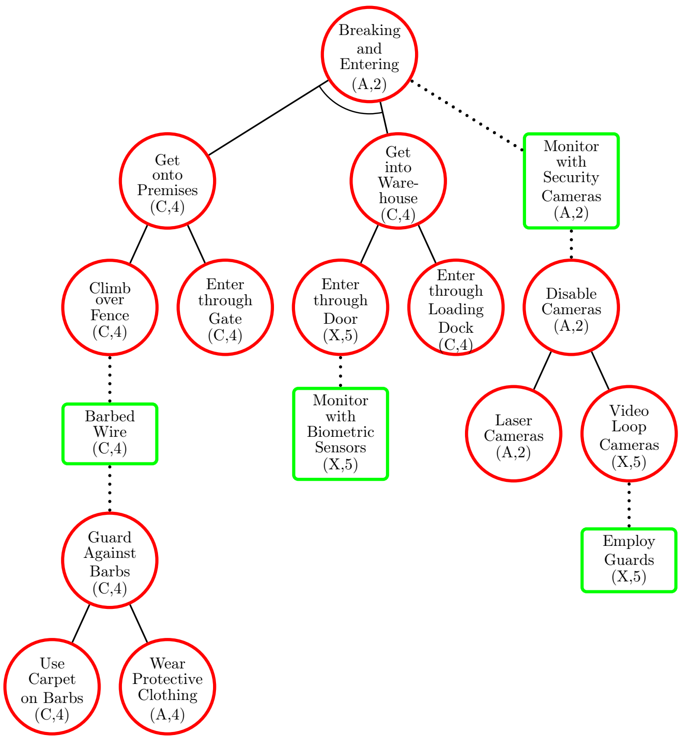
Breaking and Entering (A,2)
Get onto Premises (C,4)
Get into Ware-house (C,4)
Monitor with Security Cameras (A,2)
Climb over Fence (C,4)
Enter through Gate (C,4)
Enter through Door (X,5)
Enter through Loading Dock (C,4)
Disable Cameras (A,2)
Barbed Wire (C,4)
Monitor with Biometric Sensors (X,5)
Laser Cameras (A,2)
Video Loop Cameras (X,5)
Guard Against Barbs (C,4)
Employ Guards (X,5)
Use Carpet on Barbs (C,4)
Wear Protective Clothing (A,4)

Figure 9: The breaking and entering subtree with **costs** calculated in the bottom-up way.

From the bottom-up calculation presented in Figure 9, we deduce that an attacker can "break and enter" when he spends an `Average` amount of money, and we are confident with level 2 about that. To perform the attack, an attacker has two options: either he has to "use a carpet on the barbs", "climb over the fence", "enter trough the loading dock" and "laser the cameras", or he has to "enter through the gate", "enter through the loading dock" and "laser the cameras".

Using the bottom-up approach, we computed the minimal **costs** of an attacker for every subtree. This is the information that is depicted in every node. Comparing these values with the ones gathered in Table 3 seems like a natural consequence. In Section 5.3 we elaborate on this and other questions which illustrate the differences between attack trees and ADTrees.

5 Discussion

While performing this case study we encountered numerous design choices concerning the ADTree methodology. Some options were outright inadmissible, some easy to pinpoint, whereas for others the multiplicity of possible solutions proved the versatility of the ADTree methodology. In Section 5.1 we discuss issues related to estimation of values by players. Then, in Section 5.2, we present the alternative we were faced with when fusing several values into one final value that we then use as initial assignment in the bottom-up algorithm. Section 5.3 elaborates on the choices and problems we had during the actual bottom-up calculation. Section 5.4 contains a general discussion about the usefulness and the benefits of the entire ADTree methodology and elaborate on the four conflicting modeling goals. Finally, Section 5.5 shows the hindsight guidelines we have learned based on our specific case study.

5.1 Lessons Learned from ADTree Decoration

Some attributes like **penalty**, **profit** or **impact** were only estimated by either attackers or defenders. Assigning a specific role to a player initially seemed like a good idea, it caused the players to minimize their work, such that (with a small exception) attackers only estimated values for attack nodes and defenders for defense nodes. This reduces the number of estimated values by a half, but it is doubtful that the quality was twice as good. So, the first lesson here would be to have a clear understanding of what kind of players you have available and how to assign them. If you have a specialists within certain domains available, make sure to exploit that. For example, a janitor could estimate nodes related to physical building security, i.e., nodes depicted in Figure 6, whereas a psychologist might be better suited to estimate values for nodes related to social engineering, i.e., nodes depicted in Figure 5. A system administrator will in most cases know more about historical attacks than a software developer, and so on.

Another alternative would be to let specialists estimate values for all nodes but only for the attributes related to their field of expertise. For instance, accountants would be better suited to provide **costs** values and technical personnel could take care of deciding whether **electricity is needed** to perform the considered actions.

If the players are not specialized, e.g. you do not have that kind of people available or can afford to hire a seemingly trustworthy black hat hacker, we believe that a random assignment of nodes to be estimated is hardly justifiable. In such a case, it might be helpful to assign roles according to the node types to different players to transform them into attackers and defenders. We do not have sufficient evidence to recommend whether the players should then estimate only one type of nodes (i.e., attack node or defense nodes) or both, i.e., whether they should predict the strategy of their opponents and reflect that in the node values belonging to their role, or whether they should influence the results by estimating values for all non-refined nodes. The best data from non-specialists would probably be obtained from having the players play as both defenders and attackers, however we do suspect that having a friendly competition between two opposing teams would serve as a good motivation.

Even though all players were involved in the creation of the tree, there were cases where some of the players did not completely agree with its structure when estimating the values. Here, we feel it is best to run with a dual strategy. First, the player should nonetheless provide a value, but assign a very low confidence. Second, we should introduce a new attribute called **disagree with node** with a binary value range. Any node that has been flagged with this attribute should then be discussed at the consensus meeting. It is important to remember that other players may feel differently about the model and too many structural changes will make other values in the tree insignificant. To avoid such complications and especially repetitive work, it is important that the model of the tree is sufficiently accurate before any values are assigned.

To make full use of meta-attributes, they should always be estimated on a per attribute value basis. In particular, not all attribute values for a given node should be assigned with the same confidence level. This increases the time it takes to assign values, but it also increases the accuracy of any calculation. Furthermore, a $3-$valued confidence scale should be enough. As for the attribute values, each confidence level should be clearly explained. Otherwise distinguishing between different confidence levels is somewhat arbitrary.

If only the bottom-up approach is used during the analysis phase, estimating values for refined nodes is questionable. If an action is already sufficiently comprehensible that a reliable value could be suggested, it would not need to be refined anymore. Hence, from the fundamental modeling idea behind ADTrees, assigning values to intermediate nodes is an inherent contradiction.

On the one hand, node labels are very important because they help the players to understand the scenario without reading the scenario description in detail. Node labels that are too short may lead to confusion. On the other hand, the labels should be concise because if they are too long and detailed they are difficult to display and reduce the possibility of reusability. Therefore, to satisfy both criteria, we propose to always use a noun and a verb as node labels.

During the game, players raised the question whether nodes should be considered without their context, e.g., neighboring nodes and previously assigned values to similar nodes. We believe that if the values had been assigned inside the actual ADTree, and not in a separate table, one would have to consider the context. If the context is taken into account, the node labels might be easier understandable (due to the additional information the parent node gives). We are aware that assigning values without context can be used to detect inconsistencies and random assignments. This however, we would pay for by less accurate values, because the node by itself is then less descriptive. Estimating a value without context might not even be possible, because a bias that might have been introduced during the creation of the tree. The player may simply remember the context. Even though refined nodes should not hold less information than the children, repeating the label of the parent clutters the label of the child nodes.

5.2 Lessons Learned from Attribute Preparation

Due to the different background or knowledge of the players, the estimated values will rarely be the same. Furthermore, some players may chose not to insert a value at all. For these reasons, we end up with heterogeneous data that needs to be homogenized. In Section 4.3 we describe one possible option to proceed by using Formula (1) and to discuss the remaining values at a consensus meeting. The formula consists of a weighted average for the attribute value and an estimation of the confidence value, which reflects risk averseness. Instead of applying this formula, it is possible to for example, choose the average, the median, the used or the lowest value for the attribute value and an average, a reduced average, the lowest or possibly a new level of confidence as the confidence value. The desired method may vary, depending on the scenario and the attribute to be calculated.

When using any formula, it is, in general, preferable to have as many input values as possible, since this increases the significance of the result. However, the meaningfulness of the values may depend on the actual values that were estimated. To differentiate the inputs, we classify the input values into six categories. Then, for each of the following six categories we can choose a different approach of how to combine the estimated values in to one, e.g., using an average or a minimum value, using ranges as values or deciding on the final value at the consensus meeting. The proposed categories are:

- Category 1: Nodes with as many attribute values as players.
- Category 2: Nodes where at least one player has not estimated a value.
- Category 3: Nodes where all estimated values have a low confidence.
- Category 4: Nodes where the values diverge significantly.
- Category 5: Nodes where the **disagree** flag is set.
- Category 6: Nodes where no player has estimated a value.

The categories are ordered according to a descending scale of automation. Whereas for Category 1 it is entirely reasonable to combine the input values automatically into a single value, this is not even possible for Category 6. A decreasing automatic treatment is equivalent to an increased necessity for a consensus meeting. We again observe the conflict between modeling time and modeling accuracy. Holding a consensus meeting, will result in an improvement of most of the nodes from Category 5 or 6, because either the model was actually wrong or the model was not described clearly enough. Categories 3 and 4 only vaguely depict a design option, since the terms *low confidence* and *diverge significantly* would need to be defined, in more detail. Nonetheless, even defining low confidence as only values with confidence level of 1 or 2 and *diverge significantly* as are not neighboring in the natural order, already improved the model. For example for the attribute attack **time**, the node "postal Trojan attack" was put in Category 4. While reviewing this node at the consensus meeting, we discovered that one of estimated values was mistakenly given. Another example of model improvement was the node "hide in bathroom", where the divergent values started a discussion which led to the insight that an attack component was missing.

A different classification we have to consider is the domain of the attributes. When estimating values for non-refined nodes, naturally the question arises of why we only have three or four different possible values for each attribute. From a theoretic point of view, it is entirely possible to use real numbers, intervals or even discrete probability functions as value domains. However, the more detailed the information is a person has to estimate, the less likely he is inclined to provide a value. Using a more fine-grained scale to achieve more exact results is counterproductive, if the number of people, who estimates a value, decreases. Furthermore, increasing the graining of the scale may make it more difficult to distinguish between values.

5.3 Lessons Learned from Calculation

Comparing the values from Table 3 with the values calculated using the bottom up approach shows that the countermeasures are usually disregarded when we try to assign values to nodes on an intuitive basis. Therefore, we should not perform such a comparison. Figure 9 shows, for instance, that video looping cameras is infinitely expensive and thus impossible when guards are employed. This is contradicted by the estimated **costs** value mentioned in Table 3, which is `Average` with confidence level of 2. A similar negligence of countermeasures and subsequent counterattacks occurs if we consider attack trees instead of ADTrees. If we remove all subtrees rooted in defense nodes from Figure 9, we do not model that an attacker should worry about possible defenses, such as "barbed wire" or "monitor with security cameras". Then, the **costs** value of the cheapest scenario would be `Cheap`, with the confidence level of 4. The corresponding attack would be to enter through the gate and the loading dock, undetected.

Since ADTrees allow us to combine information about the attacker and the defender, the ADTree formalism allows to answer questions that depend on both players. We can, for instance, compute the minimal **difficulty** of an attack, assuming that the budget of the defender is limited to `Average`. Using Table 3 and Figure 9 we see that, in this case, monitoring with biometric sensors as well as with security cameras would be too expensive for the defender. Hence, there would be four possible attack scenarios:

- using the carpet on the barbs, climbing over the fence and entering through the door of the main building,
- using the carpet on the barbs, climbing over the fence and entering through the loading dock,
- entering through the gate and the door of the main building
- entering through the gate and the loading dock.

Similarly, it is possible to compute all combined attributes mentioned in Section 2.1, provided we know the values given in Table 3.

In the minimal **costs** calculation performed in Section 4.4, we have chosen to use the functions minimum and maximum. It is possible to redefine the used functions, in order to express more accurately how costly a combination of actions from different categories is, for instance, that performing a sequence of actions which are `Cheap` and `Average` is actually `Expensive`. However, in this case study we are more interested in the proof of concept rather than in precise computations, thus we use simple functions. It is clear that when our assumptions change, we have to redefine the used functions accordingly. For instance, the function minimum cannot be used any more for a disjunctively refined node, if we assume that the attacker is able to implement several among existing options and not only the cheapest one.

The minimal **costs** calculation, performed in Section 4.4, can be made more precise with the help of attributes expressed using Boolean values. Such attributes are well suited to reason about hypothetical scenarios. If we are sure that some of the hypothetically possible attacks or defenses do not occur, we can model this by pruning the tree. Pruning nicely fits in the framework of the bottom up propagation, when we use Boolean values. Let us, for instance, consider the attribute **is electricity needed**. We can prune the tree to simulate what happens if there is a power outage. Since power outage affects the attacker as well as the defender, ADTrees are the formalism we want to employ. In this case, pruning the tree is not done by simply cutting off defenses, rather we cut of the subtree rooted in the nodes which need electricity and all parents until we either hit a node of the other player or a disjunctively refined node. Pruning can also be used to reason about parts of the scenario that satisfy a certain property, like, for instance, their **costs** is lower than a certain threshold. In such cases, one player can prune the tree according to his knowledge and assumptions about the other player, to get a better overview of a realistic scenario.

5.4 Lessons Learned from the Methodology in General

The attack tree obtained from our ADTree by removing all subtrees rooted in defense nodes depicts the actual attack scenario, all other nodes describe hypothetical defenses and attacks that may or may not be in place. Therefore, the security cameras from the floor plan, see Figure 3, are not explicitly modeled as defense nodes in our ADTree. The cameras mentioned in Figure 6 are additional cameras that could be put into place. Modeling this way clearly distinguishes between the considered scenario and hypothetical attacks and defenses. When modeling an actual scenario this approach might be appropriate, whereas when we want to store possible attacks and defenses in a library it is preferable that all defenses, including the already existing ones, are depicted in the ADTree. Consequently, when using libraries as the starting point for an ADTree, we have to adapt the tree to the scenario. Moreover, we can include information we have about the attacker/defender, by adjusting the tree to the considered situation before actually starting a bottom-up calculation. For example, if we know that the defender only has a limited budget, we could remove any attack that is too expensive and then start the bottom-up calculation on the pruned tree.

Since an incorrect or missing value anywhere in the tree can affect the resulting value for an attribute, this indicates that the level of node refinement is crucial. To avoid biased results, the level of refinement should roughly be the same for all branches. For us this means that for most of the non-refined nodes we have the same intuitive level of understanding.

The level of refinement may be influenced by who created the tree and how it was created. First, the players can be given the tree and act as independent security experts or they can have created the tree (even described the corresponding scenario). Second, tree creation can start from templates available in security repositories, as suggested by Meland et al. [Meland et al., 2010] or from an empty sheet of paper. In either combination, there is a trade-off between time and creativity.

An observation that we found interesting is that some of the attributes, such as **skill, attack costs** and **insider required**, tell us a lot about the requirements for the attacker. This would actually allow us to generate attacker profiles based on specific projections of the tree. Having such profiles would be of benefit for the defender in order to identify potentially harmful candidates.

There are four conflicting modeling goals we would like to emphasize: time, reusability, accuracy and simplicity, which all have implications on the complexity of the analysis. In modeling, *time* is always a concern. According to our experience, companies spend between one hour and one week on threat modeling before the implementation starts. From a theoretic point of view this might not be enough, but unless we see a paradigm shift in security modeling, time is always a scare resource. The amount of time (and therefore money) spent always has to be justified by either allowing the analysis to be highly reusable or require a high degree of accuracy.

If we spend a lot of time modeling, we prefer our analysis to be *reusable*. For graphic security modeling, libraries immediately come to mind. For this reason, the SHIELDS project [SHIELDS, 2008-2010] developed an online library for (among others) attack trees. This library could be extended to also include attack–defense trees. Whereas a repository for the structure of attack scenarios already exists, there has not yet been an attempt to also store node values together with the structural information. The degree of reusability might not be as high for actual values. Therefore, instead of storing concrete values, it might be preferable to store ranges of admissible values which serve as possible and not actual values. The more values for different attributes are retrievable, the more likely some information will be reusable. Using stored values may again conflict with other modeling goals, such as a fast scenario analysis (the stored node values most likely still have to be adapted) and, unless a computer tool is used, the visual appeal of the ADTrees is diminished, because the tree feels cluttered.

A third conflicting modeling goal is the *accuracy* of the model and the values. It is necessary to find an acceptable compromise between the required time and necessary accuracy. Also, more accurate ADTrees and values reduce the reusability of the ADTree. In general, we can say that the coarser the value, the more raw data we get, because more people feel comfortable with actually providing the value. The finer the value, the more precise the result will be, but if the values are too fine, only experts might be able to estimate values. A coarse value range for a **costs** attribute would, for example, be `Low`, `Medium` and `High`, a fine grade would be if the value was given as a real number expressing a monetary value, e.g., in €.

As a last modeling goal, we want the methodology of ADTrees to be easily *understandable*. We use a simple tree structure which is a main advantage over the generalized petri-net approach. But we do not only want the relation between the basic actions depicted in an easy way, we also want non-experts to be able to make full use of the ADTree methodology. Therefore, we also want a common user, developer, administrator or system owner to be able to estimate values for basic actions. By doing this, we benefit from a larger resource pool of potential attribute assigners, which might reduce the costs, because we do not need to hire an expert for tree creation and providing values. This however, might have implications on the accuracy of the values.

5.5 Hindsight Guidelines for the Warehouse Case Study

Earlier in this section, we have elaborated on possible methodology design choices that typically occur in case studies such as ours. The "right" choices depend on the actual scenario, the security relevant questions to be answered, the modeling goals, the client, and last but not least, the people performing the case study. None of them should be treated in isolation. In Table 4, we take this discussion into account and list numerous design choices for the presented RFID case study. The bold options indicate which of the choices we would select with hindsight, but are of course not necessarily the right choices for other system settings.

Table 4: Work flow – Exemplary guidelines for the use of ADTrees for our case study

Step	Task	Design choices
1	**Create ADTree for scenario**	- Create tree from root node on/**adapt tree from existing templates**. - **Use**/do not use incomplete trees. - Continuously improve trees/**freeze tree structure at some time**. - Use **concise noun and verb**/detailed textual description as node label. - **Security expert**/system owner/random person creates tree. - Use **same** level of detail for refinements/limit number of nodes. - Allow/**disallow** pruning. - **Assume**/do not assume players are the creator of the tree.
2	**Choose and describe attributes**	- Use attribute description given in **Table 1**/provide new descriptions. - Select attribute domains: **discrete**/real numbers/fuzzy sets/intervals/probability measures. - **Allow**/disallow (**disagree with node** attribute). - **Always**/sometimes use meta-attribute confidence.
3	**Choose who estimates what**	- Who estimates: attackers/defenders/**specialists**/random people. - Which nodes: according to role of player/to **background**/depending on attribute/all nodes.
4	**Estimate values**	- Evaluate meta-attributes for all attributes **separately**/together. - Consider nodes **in**/without context. - Allow/**disallow** different values for repetitive nodes. - **Do not estimate**/estimate values for intermediate nodes.
5	**Combine values**	- Apply standard combining procedure for **Categories 1–4**/for other categories. - Use **Formula** (1)/something else as standard procedure. - Use averaging/minimization/majority/**consensus meeting** for alternative categories. - **Restrict**/do not restrict time in case of consensus meetings.
6	**Calculate values**	- Use **predefined**/other functions from software tool or literature. - Compare/**do not compare** with intermediate values.

6 Conclusion and Future Work

In this paper, we have looked at the use of attributes for attack–defense trees (ADTrees). After explaining the ADTree formalism and giving an overview of typical attributes for attack trees found in the literature, we have described a case study for an RFID-based system managing goods in a warehouse. An ADTree depicting possible attacks on the considered system and the subsequent countermeasures was created. Then, a set of suitable attributes was selected and their values were determined. The results of the attribute evaluation can be used as a part of a redesign process or risk analysis in order to improve the security of the system.

The main contributions from this case study are practical experiences and user feedback. Taking the lessons learned during the case study into account, we have extended the original case study process graph depicted in Figure 8. Below we present the resulting six steps guideline which suggests a work flow and lists possible design choices that we recommend for applying the ADTree methodology when performing case studies.

1. *Create an ADTree for the scenario.* An ADTree is created using all available information

and support tools. The attack tree, obtained from an ADTree by ignoring all defense nodes and the corresponding subtrees, depicts the main attack scenario. All other nodes describe hypothetical defenses and counterattacks.

- People with different knowledge and relationship to the system, e.g., developers, security experts, system owner and end users, should be involved in the tree creation.
- Different material, such as system specifications, floor plans, blueprints, work descriptions, attack tree libraries and attack patterns, should be used to create the tree.
- The creation of the tree should be an iterative process which should end when there is mutual agreement between the involved parties. Modifying the tree after Step 3 should be avoided.
- Node labels should contain a verb and a noun and concisely represent an attack or defense action.

2. *Choose and describe attributes:* Relevant attributes and meta-attributes are chosen, based on the security questions to be answered.

   - A clear, written description of chosen attributes and meta-attributes should be provided.
   - The description of each attribute and meta-attribute should include a domain which is used to quantify the values.
   - In the case of discrete domains, a written definition for each introduced category, such as `small`, `medium`, `big`, should be provided.
   - A user should be allowed to express whether he disagrees with a part of the tree, e.g., by including the **disagree with node** attribute in the list of attributes.

3. *Choose who estimates attribute values:* Decide which and how many people estimate which values. Optimize the number of people with respect to the available resources.

   - In order to avoid errors and take into account different perspectives, more than one player should estimate attribute values.
   - Each player should obtain clear, written instructions detailing which values to estimate. It is not necessary that each player estimates the values for all nodes and/or all attributes, however it should be mandatory that he provides the values he is assigned to estimate.

4. *Value estimation:* The players selected in Step 3 estimate the values of the attributes chosen in Step 2 with the help of the support material identified in Step 1.

   - The values should be estimated *only* based on the attribute and meta-attribute descriptions provided in Step 2.
   - When the bottom-up approach is used, the values should only be estimated for non-refined nodes.
   - The **confidence** meta-attribute should express a user's confidence in the provided attribute value. It should therefore be given for each estimated attribute value separately.
   - The attribute values should be estimated based on the node's context in the tree.

5. *Value combination:* When the attribute estimates from different people diverge, a combined value needs to be obtained. This value should be the best representative for all input values.

   - Nodes should be partitioned into categories, depending on clear objective criteria, such as percentage of **coverage**.
   - The best way of deriving the representatives should be selected independently for each category, e.g., use a suitable formula, average or decide at a consensus meeting.

- In case a consensus meeting is called for, its duration should be limited.

6. *Value calculation:* If the bottom-up approach is to be applied, suitable functions need to be chosen in order to calculate values for all the subtrees of a considered tree.

    - The used functions should be in accordance with the attribute descriptions provided in Step 2.
    - Scientific papers discussing attribute evaluation and existing attack tree tools can be consulted in order to define the appropriate functions.
    - Estimated values of refined subtrees should not be compared with values resulting from the bottom-up algorithm, unless the tree does not contain any defense nodes.

When performing the case study, it became apparent that a software tool supporting the security analysis using attributes on ADTrees would be of great value. Such a tool is currently under development at the University of Luxembourg. Its main objective is to facilitate the work with the ADTree formalism by allowing to answer questions pertaining to security aspects based on realistic models. In particular, such a tool should lend support during input of values, show different tree views that focus on specific parts of a scenario, prevent repetitive tasks, lend support while computing values, be able to generate attack scenarios.

In the future, we intend to carry out another case study using the ADTree methodology. This will help us to further substantiate the ADTree formalism. We hope that it will allow us to expand our recommendations on modeling choices, fine-tune the attribute descriptions, lead to more insights about which attribute domains to choose in which case and test our software tool.

Another line of research that we foresee is to consider the use of the ADTree methodology in diagnostics and forensics. More explicitly, we would like to look at the question whether ADTrees can be usefully employed once an attack has occurred in order to reconstruct what happened.

Acknowledgments

We would like to thank Ton van Deursen and Domenico Rotondi for their invaluable insights in RFID technology and their help to create the ADTree used in this case study. B. Kordy and P. Schweitzer were supported by grants from the National Research Fund, Luxembourg, with No. C08/IS/26 and PHD-09-167, respectively. P. H. Meland and A. Bagnato have received funding leading to these research results from the European Union Seventh Framework Programme (FP7/2007-2013) under grant No. 215995 and 257930, and 215995, respectively.

References

Parosh Aziz Abdulla, Jonathan Cederberg, and Lisa Kaati. Analyzing the Security in the GSM Radio Network Using Attack Jungles. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA (1)*, volume 6415 of *LNCS*, pages 60–74. Springer, 2010. ISBN 978-3-642-16557-3.

Amenaza. SecurITree, 2001. URL `GermanLink:http://www.m-privacy.de/produkte/securitree/`. `http://www.amenaza.com/`.

Edward G. Amoroso. *Fundamentals of Computer Security Technology*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994. ISBN 0-13-108929-3. URL `http://portal.acm.org/citation.cfm?id=179237#`.

Dejan Baca and Kai Petersen. Prioritizing Countermeasures through the Countermeasure Method for Software Security (CM-Sec). In Muhammad Ali Babar, Matias Vierimaa, and Markku Oivo, editors, *PROFES*, volume 6156 of *LNBIP*, pages 176–190. Springer, 2010. ISBN 978-3-642-13791-4.

Stefano Bistarelli, Marco Dall'Aglio, and Pamela Peretti. Strategic Games on Defense Trees. In Theodosis Dimitrakos, Fabio Martinelli, Peter Y. A. Ryan, and Steve A. Schneider, editors, *FAST*, volume 4691 of *LNCS*, pages 1–15. Springer, 2006. ISBN 978-3-540-75226-4. URL `http://www.springerlink.com/content/83115122h9007685/`.

Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. Rational Choice of Security Measures Via Multi-parameter Attack Trees. In Javier Lopez, editor, *Critical Information Infrastructures Security*, volume 4347 of *LNCS*, pages 235–248. Springer, 2006. URL `http://dx.doi.org/10.1007/11962977_19`. 10.1007/11962977_19.

E.J. Byres, M. Franz, and D. Miller. The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems. In *International Infrastructure Survivability Workshop. (IISW'04)*, 2004.

Marc Dacier and Yves Deswarte. Privilege graph: an extension to the typed access matrix model. In *ESORICS*, volume 875 of *LNCS*, pages 319–334, 1994. doi: 10.1007/3-540-58618-0_72. URL `http://dx.doi.org/10.1007/3-540-58618-0_72`.

Mamadou H. Diallo, Jose Romero-Mariona, Susan Elliot Sim, Thomas A. Alspaugh, and Debra J. Richardson. A comparative evaluation of three approaches to specifying security requirements. In *12th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06)*, June 2006.

Kenneth S. Edge, George C. Dalton II, Richard A. Raines, and Robert F. Mills. Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security. In *Proceedings of the 2006 IEEE Military Communications Conference*, MILCOM'06, pages 953–959, Piscataway, NJ, USA, 2006. IEEE Press. ISBN 1-4244-0618-8. URL `http://portal.acm.org/citation.cfm?id=1896579.1896724`.

Casey Fung, Yi-Liang Chen, Xinyu Wang, J. Lee, R. Tarquini, M. Anderson, and R. Linger. Survivability analysis of distributed systems using attack tree methodology. In *Proceedings of the 2005 IEEE Military Communications Conference*, volume 1, pages 583–589, October 2005. doi: 10.1109/MILCOM.2005.1605745.

Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Weyl. Security requirements for automotive on-board networks. In *9th International Conference on Intelligent Transport Systems Telecommunications,(ITST)*, pages 641–646, Lille, 2009. doi: 10.1109/ITST.2009.5399279.

Cormac Herley. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proceedings of the 2009 New security paradigms workshop*, NSPW '09, pages 133–144, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-845-2. doi: http://doi.acm.org/10.1145/1719030.1719050. URL `http://doi.acm.org/10.1145/1719030.1719050`.

Aivo Jürgenson and Jan Willemson. Computing Exact Outcomes of Multi-parameter Attack Trees. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (2)*, volume 5332 of *LNCS*, pages 1036–1051. Springer, 2008. ISBN 978-3-540-88872-7. doi: 10.1007/978-3-540-88873-4_8.

Barbara Kordy, Sjouke Mauw, Matthijs Melissen, and Patrick Schweitzer. Attack–Defense Trees and Two-Player Binary Zero-Sum Extensive Form Games Are Equivalent. In Tansu Alpcan, Levente Buttyán, and John S. Baras, editors, *GameSec*, volume 6442 of *LNCS*, pages 245–256. Springer, 2010. ISBN 978-3-642-17196-3.

Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of Attack–Defense Trees. In Sandro Etalle Joshua Guttman Pierpaolo Degano, editor, *FAST*, volume 6561 of *LNCS*, pages 80–95. Springer, 2011a. URL `http://satoss.uni.lu/members/barbara/papers/adt.pdf`.

Barbara Kordy, Marc Pouly, and Patrick Schweitzer. Computational Aspects of Attack–Defense Trees. In *Security & Intelligent Information Systems*, volume 7053 of *LNCS*, pages 103–116. Springer, 2011b.

Xiaohong Li, Ran Liu, Zhiyong Feng, and Ke He. Threat modeling-oriented attack path evaluating algorithm. *Transactions of Tianjin University*, 15(3):162–167, 2009. doi: 10.1007/s12209-009-0029-y.

Theodore W. Manikas, Mitchell A. Thornton, and David Y. Feinstein. Using Multiple-Valued Logic Decision Diagrams to Model System Threat Probabilities, 2011. to appear.

Sjouke Mauw and Martijn Oostdijk. Foundations of Attack Trees. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *LNCS*, pages 186–198. Springer, 2005. ISBN 3-540-33354-1.

Per Håkon Meland, Inger Anne Tøndel, and Jostein Jensen. Idea: Reusability of threat models - two approaches with an experimental evaluation. In Fabio Massacci, Dan Wallach, and Nicola Zannone, editors, *Engineering Secure Software and Systems*, volume 5965 of *LNCS*, pages 114–122. Springer, 2010. ISBN 978-3-642-11746-6. URL `http://dx.doi.org/10.1007/978-3-642-11747-3_9`.

Luke Mirowski, Jacqueline Hartnett, and Raymond Williams. An RFID Attacker Behavior Taxonomy. *IEEE Pervasive Computing*, 8:79–84, 2009. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2009.68.

Andrew P. Moore, Robert J. Ellison, and Richard C. Linger. Attack Modeling for Information Security and Survivability, 2001.

Andreas L. Opdahl and Guttorm Sindre. Experimental comparison of attack trees and misuse cases for security threat identification. *Information & Software Technology*, 51(5):916–932, 2009. doi: doi:10.1016/j.infsof.2008.05.013.

Ludovic Piètre-Cambacédès and Marc Bouissou. Beyond Attack Trees: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP). In *European Dependable Computing Conference*, pages 199–208, Los Alamitos, CA, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4007-8. doi: http://doi.ieeecomputersociety.org/10.1109/EDCC.2010.32.

Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, 2011. ISSN 1939-0122. doi: 10.1002/sec.299. URL `http://dx.doi.org/10.1002/sec.299`. `http://dx.doi.org/10.1002/sec.299`.

Vineet Saini, Qiang Duan, and Vamsi Paruchuri. Threat Modeling Using Attack Trees. *J. Computing Small Colleges*, 23(4):124–131, 2008. ISSN 1937-4771. URL `http://portal.acm.org/citation.cfm?id=1352100`.

Bruce Schneier. Attack Trees. *Dr. Dobb's Journal of Software Tools*, 24(12):21–29, 1999.

SHIELDS. FP7 project, grant agreement 215995, 2008-2010. `http://www.shields-project.eu/`.

Eedee Tanu and Johnnes Arreymbi. An examination of the security implications of the supervisory control and data acquisition (SCADA) system in a mobile networked environment: An augmented vulnerability tree approach. In *5th Annual Conference on Advances in Computing and Technology, (AC&T)*, pages 228–242. University of East London, School of Computing, Information Technology and Engineering, 2010. ISBN 978-0-9564747-0-4. URL `http://hdl.handle.net/10552/994`.

Inger Anne Tøndel, Jostein Jensen, and Lillian Røstad. Combining Misuse Cases with Attack Trees and Security Activity Models. In *International Conference on Availability, Reliability and Security*, pages 438–445, Los Alamitos, CA, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-3965-2. doi: http://doi.ieeecomputersociety.org/10.1109/ARES.2010.101.

W. E. Vesely, F. F. Goldberg, N.H Roberts, and D.F. Haasl. Fault Tree Handbook. Technical Report NUREG-0492, U.S. Regulatory Commission, 1981. URL `http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf`.

Jie Wang, John N. Whitley, Raphael C.-W. Phan, and David J. Parish. Unified Parametrizable Attack Tree. *International Journal for Information Security Research*, 1(1):20–26, 2011. URL `http://www.infonomics-society.org/IJISR/Unified%20Parametrizable%20Attack%20%Tree.pdf`.

Ronald R. Yager. OWA trees and their role in security modeling using attack trees. *Information Sciences*, 176(20):2933–2959, 2006. ISSN 0020-0255. doi: DOI:10.1016/j.ins.2005.08.004. URL `http://www.sciencedirect.com/science/article/B6V0C-4H511HB-1/2/4431b03877e8b6d03677146af7b9fe4e`.