Security Modeling

Lecture Notes

Sjouke Mauw, Peter Ryan, Barbara Kordy, Patrick Schweitzer, Tim Muller

2011/2012

These are the lecture notes for the course *Security Modeling* given at the University of Luxembourg between September and December 2011.

Contents

1 Class Description **5**
- 1.1 Topics . . . 5
- 1.2 Outcomes . . . 5
- 1.3 Organization . . . 5
- 1.4 Students' Evaluation . . . 6
  - 1.4.1 Homework . . . 6
  - 1.4.2 Research Presentation and Report . . . 6
  - 1.4.3 Final Grade . . . 6

2 Attack–Defense Trees **7**
- 2.1 Attack Trees . . . 7
  - 2.1.1 Example . . . 7
- 2.2 Attack–Defense Trees . . . 8
  - 2.2.1 Running Example . . . 10
- 2.3 Attributes for Attack–Defense Trees . . . 14
  - 2.3.1 Bottom-up Evaluation of Attributes . . . 15
  - 2.3.2 Minimal Cost of the Attacker . . . 17
- 2.4 Literature for Interested Students . . . 19
  - 2.4.1 Attack Trees Origins . . . 19
  - 2.4.2 Formalization of Attack Trees . . . 19
  - 2.4.3 Formalization of Attack–Defense Trees . . . 19
  - 2.4.4 Attributes for Attack Trees . . . 19
  - 2.4.5 Case Studies and Experience Reports . . . 19
- 2.5 The ADTree Methodology for Performing Case Studies . . . 20
  - 2.5.1 Step 1: How to create a tree? . . . 20
  - 2.5.2 Step 2: Which attributes are interesting? . . . 20
  - 2.5.3 Step 3: Who should estimate attribute values for which nodes? . . . 21
  - 2.5.4 Step 4: How to estimate values? . . . 21
  - 2.5.5 Step 5: Combining Values . . . 22
  - 2.5.6 Step 6: Bottom-up Procedure . . . 23
  - 2.5.7 Summary . . . 24

Chapter 1

Class Description

1.1 Topics

The following topics will be presented during the class

1. Attack trees, attack-defense trees (2 weeks, BK, PS)
2. Trust models (subjective logics etc.) (2 weeks, TM, SM)
3. Information flow (1 week, PR)
4. E-voting (2 weeks, PR)
5. Seminar sessions (4 weeks, all)

1.2 Outcomes

After successful completion of this course the student should be able to:

- select a suitable modeling technique for a given problem
- apply a given modeling technique to a specific domain or problem
- assess the security of a given system through its modeling
- compare modeling techniques with respect to expressive power and suitability to a given domain
- describe new trends in security modeling
- identify the limitations of modeling techniques with respect to real-world security

1.3 Organization

Security Modeling is a third semester master course. The lectures will be spread over 13 weeks. They will take place on Wednesdays between 3:45pm and 5:15pm, at campus Kirchberg in B17. The lectures will start on September 21 and end on December 14. General information about the course can be found at `http://satoss.uni.lu/courses/securitymodeling/`. Dynamic information concerning the course will be available in the FSTC Moodle system.

1.4 Students' Evaluation

1.4.1 Homework

Each teacher will propose exercises to be solved at home. The students solve the homework and send it to the teacher before a given deadline. Homework will be graded. Solving proposed exercises on time is mandatory.

1.4.2 Research Presentation and Report

In the middle of the semester, the teachers will propose presentation topics. Each student will be asked to pick one topic and present it by giving a seminar. A research report, taking into account the feedback obtained during the seminar, will be written by the student. The deadline for submitting the report is the 1st of January 2012.

Attending all seminar sessions is mandatory.

1.4.3 Final Grade

The final grade will be calculated according to the formula:

$$50\% \text{ homework} + 50\% \text{ presentation and report.}$$

Chapter 2

Attack–Defense Trees

Security assessment of systems is a standard but suboptimal procedure due to its informal nature. While a formal approach would be desirable, but is out of reach, a systematic approach would be beneficial and feasible.

Attack–defense tree methodology constitutes one of several tools for security modeling and evaluation. In this chapter we give an overview of the attack–defense tree formalism. We start with the presentation of attack trees from which attack–defense trees take their origins. Then, we show how to extend attack trees by incorporating possible defensive measures and subsequent attack components. Finally, we focus on attributes' evaluation which serves for quantitative and qualitative analysis of attack–defense scenarios.

This chapter is mostly based on and reuses parts of [KMRS10] and [KPS11]. The running example is taken from an unpublished work by Bagnato, Kordy, Meland and Schweitzer *Attribute Decoration of Attack–Defense Trees*.

2.1 Attack Trees

Attack trees [Sch99, MO05] are a well-known methodology for assessing the security of complex systems.

An attack tree is a rooted tree representing an attack scenario.

The *root* of an attack tree depicts the main goal of an attacker, and the other nodes constitute *refinements* of this goal into sub-goals. Two kinds of refinements are possible: conjunctive and disjunctive.

- A *conjunctively* refined (sub-)goal is satisfied if all its children are fulfilled.
- A *disjunctively* refined (sub-)goal is satisfied when at least one of its children is fulfilled.

The *non-refined nodes*, i.e., leaves of an attack tree, represent so called *basic actions* which are used to build complex attacks.

Graphical Representation Nodes of an attack tree are represented with *red circles*. Moreover, in order to distinguish between disjunctively and conjunctively refined nodes, we connect the edges pointing to the children of a conjunctively refined node with an *arc*, as in Figure 2.2.

2.1.1 Example

The attack tree from Figure 2.1 represents a *Denial of Service* (DoS) attack on an RFID-based management system. The root node of the tree is called "RFID DoS Attack". It is a disjunctive node. This means that, in order to achieve the corresponding goal, an attacker has six options represented by the child nodes

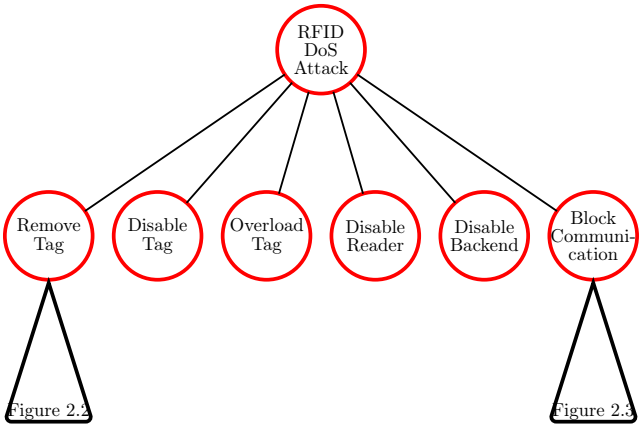

Figure 2.1: Attack tree: DoS attack on RFID management system

- "Remove Tag"
- "Disable Tag"
- "Overload Tag"
- "Disable Reader"
- "Disable Backend" and
- "Block Communication" between tags and readers.

The subtrees rooted in "Remove Tag" and "Block Communication" nodes are depicted in Figures 2.2 and 2.3, respectively. According to the subtree from Figure 2.3, 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. 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 around the reader or tag.

The "Shield Tag" node is a conjunctive node. It means, that both its children

- being in the "Vicinity of Tag" and
- using a "Faraday Cage"

have to be fulfilled in order to satisfy the goal represented by the node. 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.

A detailed description of the entire scenario is given in Section 2.2.1.

2.2 Attack–Defense Trees

Attack–defense trees (ADTrees) [KMRS10] are attack trees extended with defense nodes.

An ADTree is a rooted tree representing an attack–defense scenario.

In other words, ADTrees represent scenarios involving

- actions of an attacker trying to compromise a system
- and counteractions of a defender trying to protect the system.

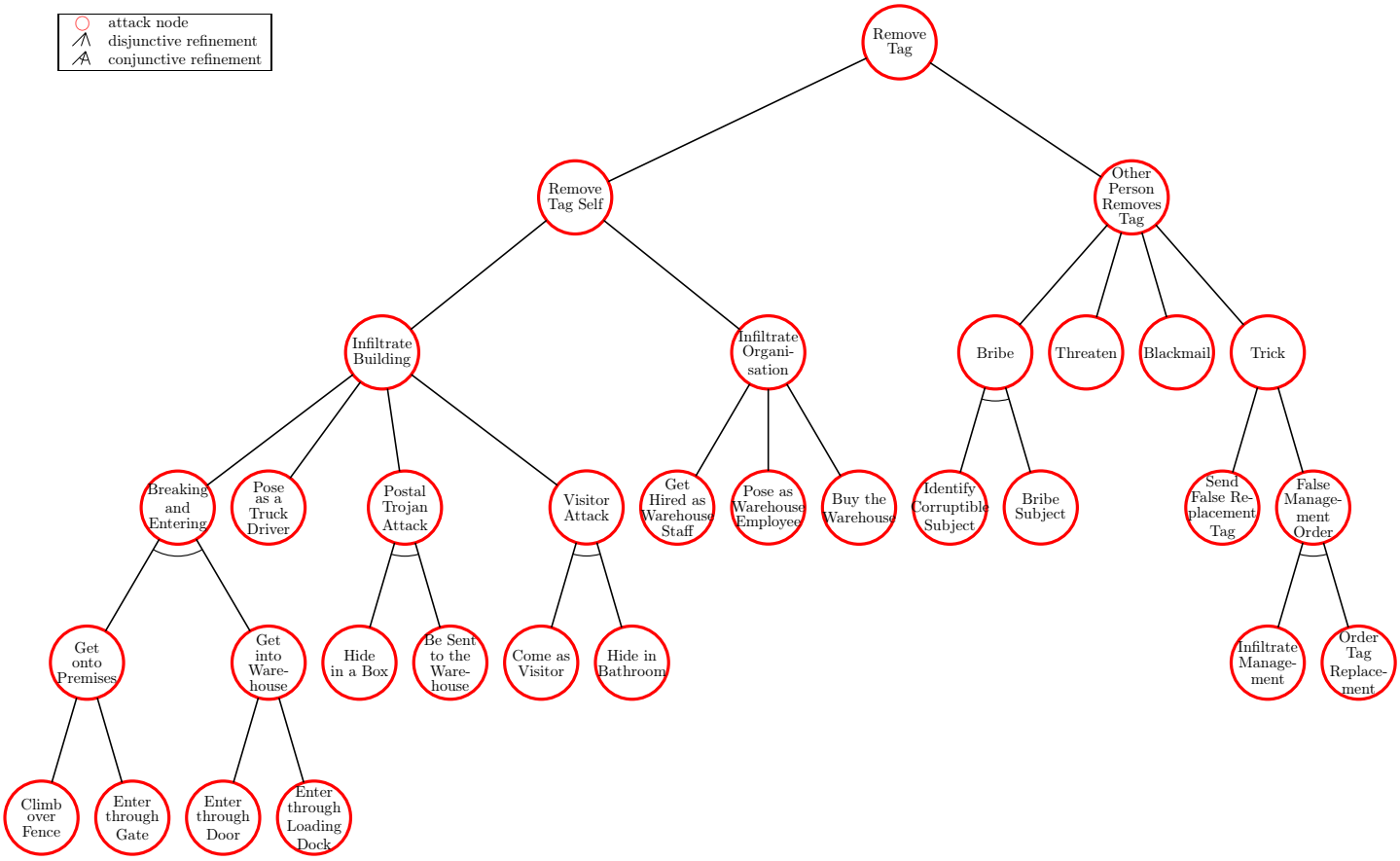

Figure 2.2: Attack tree: how to remove a tag

Consequently, an attack–defense tree can be seen as a game between two players: an *attacker* and a *defender*. Each node of an attack–defense tree depicts a (sub-)goal of one of the players, and the root node represents the main goal of an attacker or of a defender, depending on the modeler's perspective. Therefore, instead of talking about attacker and defender, we rather refer to them as proponent and opponent:

- by *proponent* we mean the player related to the root node,
- and by *opponent* we mean the other player.

As in the case of attack trees, every node of an attack–defense tree can be refined in a conjunctive or a disjunctive way. The refinement is modeled using child nodes of the same *type* (proponent or opponent) as the type of the parent node.

In addition to the refining children, *each node of an attack–defense tree may have one child of the opposite type*. Such a child then represents a *countermeasure* that can be applied to counter or mitigate the (sub-)goal represented by its parent. Finally, every node without any child of the same type is called a *non-refined node* and represents a basic action. Contrary to attack trees, such a node does not have to be a leaf, because it can still have a child of the opposite type.

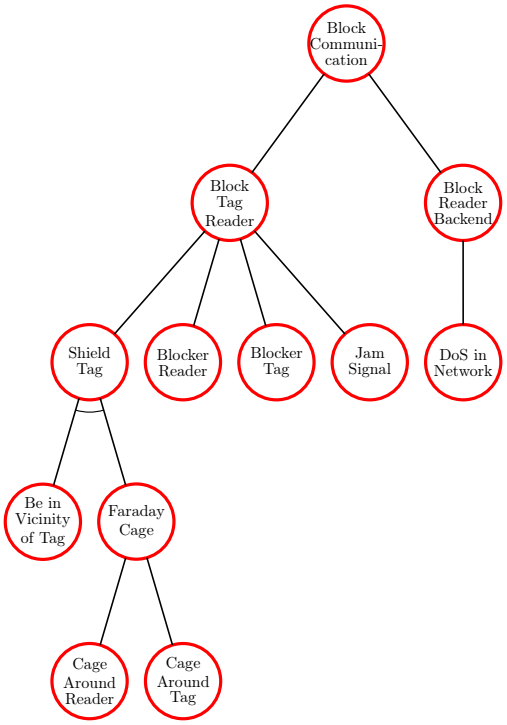

Figure 2.3: Attack tree: block communication between a tag and a reader

Graphical Representation In attack–defense trees, we depict attack nodes by *red circles* and defense nodes by *green rectangles*. Refinement relations are indicated by *solid edges* between nodes, and a countermeasure is connected to the countered node using *dotted edge*. As in the case of attack trees, an *arc* connects the edges going from a conjunctively refined node to its children of the same type.

2.2.1 Running Example

In order to present a realistic example, 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 2.4 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 process universal business language documents like *Picking Lists* (PL), 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 2.5 depicts the physical premises and the equipment inside the warehouse.

The WaLDo application operates in a warehouse where eight employees are working. The size of the warehouse building is 500m^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

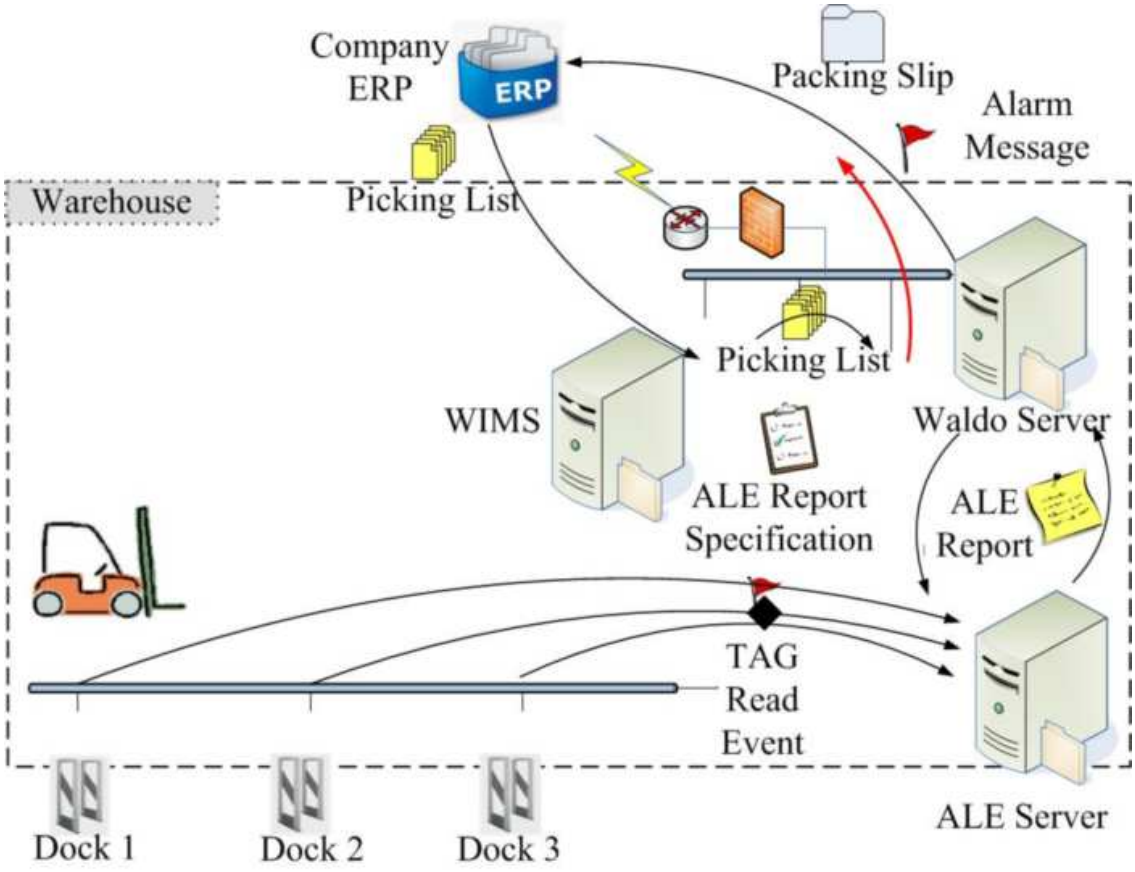

Figure 2.4: An RFID system scheme

pass in and out through the loading docs 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. Moreover, intrusion alarms are placed on the fence and the fence gates must 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 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 entrance. Each day between 10 and 20 trucks deliver goods to or from the warehouse. The drivers load the goods on and of their trucks and have warehouse access through the docs.

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).

The description given above was used to create an ADTree presented in Figures 2.6, 2.7, 2.8 and 2.9. This ADTree models an attack–defense scenario extending the attack scenario considered in the previous section with possible defensive measures. The obtained scenario is detailed below.

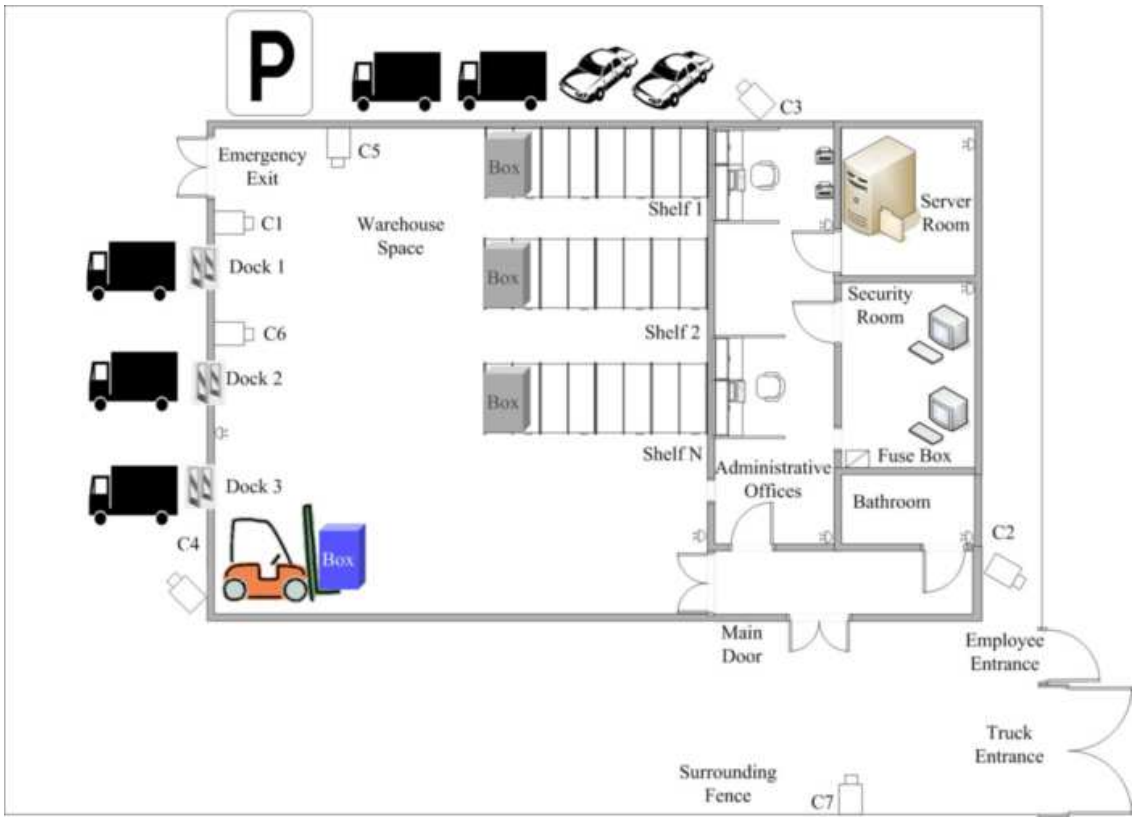

Figure 2.5: Floor plan

Description of the ADTree The top goal node of the high level tree model shown in Figure 2.6 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. We chose to only refine the nodes "Remove Tag" and "Block Communication". The corresponding subtrees are depicted in Figures 2.7 and 2.9. We deliberately chose to analyze an incomplete tree to reflect that in most use cases, the modeling time is limited which invariably leads to incomplete trees.

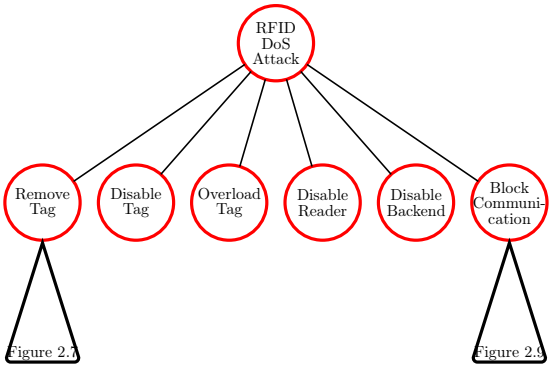

Figure 2.6: ADTree: RFID DoS attack

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 legit access. Infiltrating the building can

be achieved by “breaking and entering”, as detailed in Figure 2.8, 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 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”, 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 remove the tag, he can “bribe”, “threaten”, “blackmail” or “trick” this person. In 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 that it cannot be removed.

If an attacker decides to remove the tag himself by breaking and entering he must ”get onto the premises” undetected and he must “get into the warehouse” undetected. To get onto the premises, an attacker can “climb over the fence” or he can “enter through the gate” for employees. 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” or “entering through the loading dock”. 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 laster at the cameras or by “video looping the camera” feed. Alternatively, guards patrolling the premises could protect against this kind of threat.

As we have already mentioned in Section 2.1.1, 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. 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.

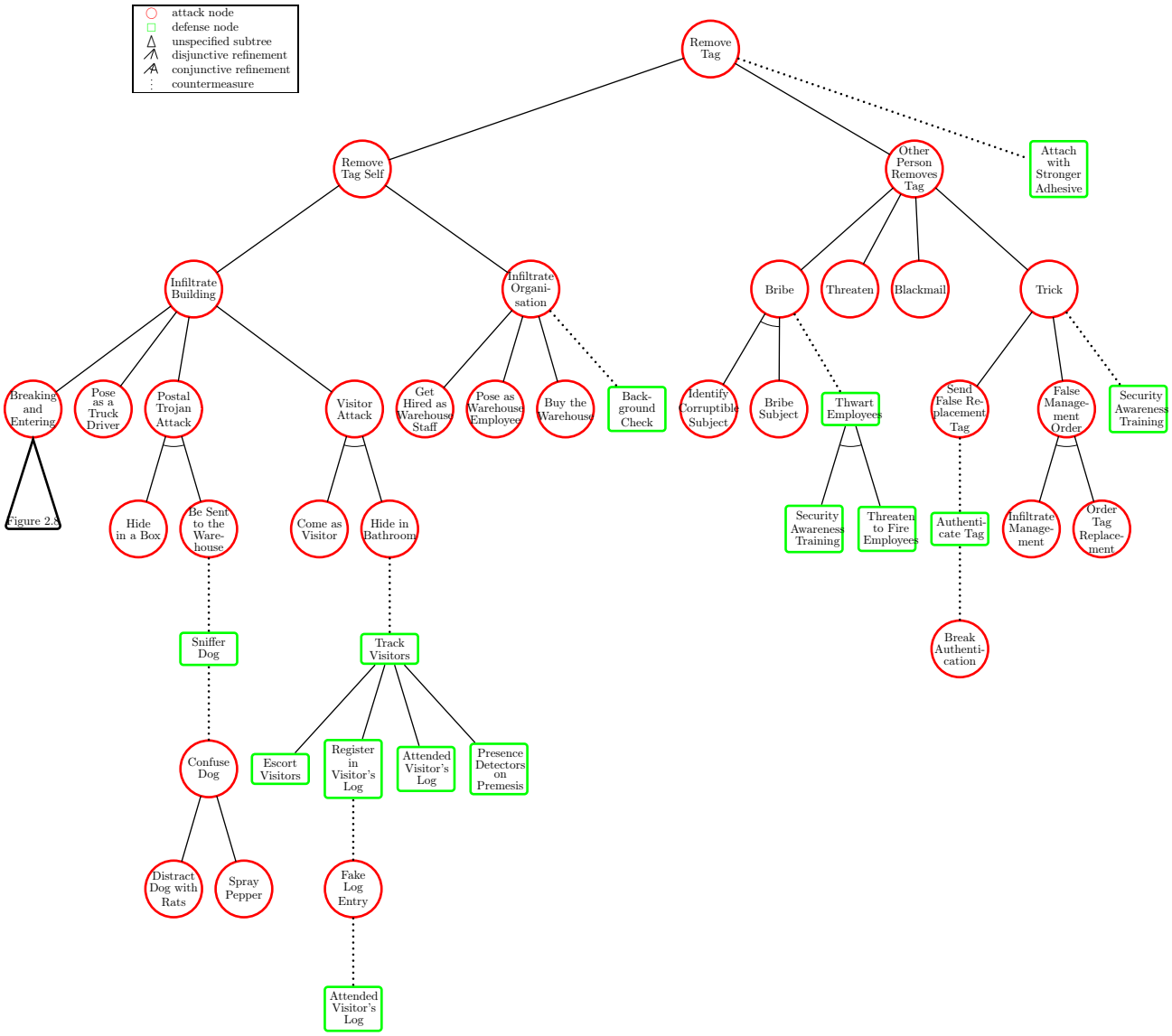

Figure 2.7: The social engineering subtree

2.3 Attributes for Attack–Defense Trees

A quantitative analysis of an attack–defense scenario represented by ADTrees is done with the help of attributes.

An attribute expresses a particular property of an attack–defense scenario,

e.g., the **minimal cost of an attack** or the **expected impact of a defensive measure**. Attributes on attack trees are used to answer questions such as: Is it possible to attack the system? How much would an attack cost? How long an attacker has to prepare his attack? Using ADTree methodology, we can extend the above questions to bivariate questions, i.e., questions, where inputs from attackers and defenders are needed and possibly have constraints. This may include questions such as: Given a limited defense budget, can the defender at least defend against some attacks? How long does it take to secure the entire system? How does the scenario change in case of a power outage?

In this section, we first present one of the most often used approaches for evaluation of attributes

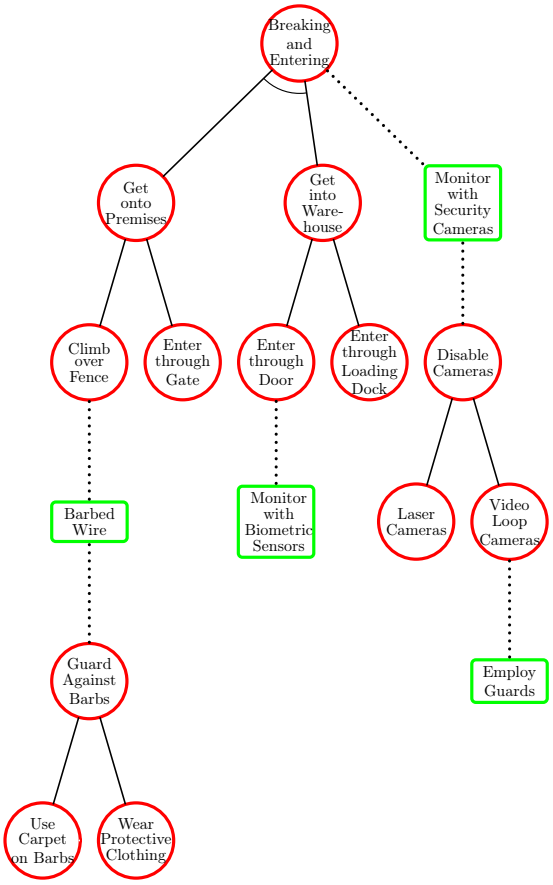

Figure 2.8: The breaking and entering subtree

on attack trees and attack–defense trees. Then, we illustrate how to use this approach to calculate the **minimal cost of the attacker** on our running example from Section 2.2.1.

2.3.1 Bottom-up Evaluation of Attributes

The most often used approach for attributes evaluation on attack trees is the bottom-up procedure. It was originally sketched by Schneier in [Sch99] and then formalized by Mauw and Oostdijk in [MO05]. The work of Kordy et al. [KMRS10] extends it from attack trees to attack–defense trees.

The bottom-up evaluation works as follows.

1. First, all the non-refined nodes are quantified with values.
2. Then, the values corresponding to all subtrees, and in particular for the entire tree, are calculated, using functions which depend on the type (proponent/opponent, disjunctive/conjunctive) of the root of the subtree and the considered attribute.

The bottom-up algorithm is a recursive procedure: *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.*

Therefore, in order to use the bottom-up procedure, we have to define an *attribute domain* which is a 7-tuple of the form $(D, f^{\mathrm{p}}_{\vee}, f^{\mathrm{p}}_{\wedge}, f^{\mathrm{o}}_{\vee}, f^{\mathrm{o}}_{\wedge}, f^{\mathrm{p}}_{\mathrm{c}}, f^{\mathrm{o}}_{\mathrm{c}})$, where

- D is the set of values used to quantify a considered property,

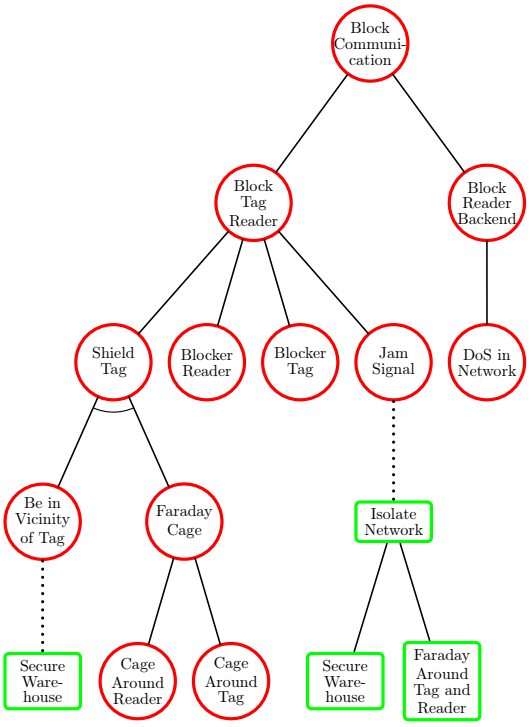

Figure 2.9: The block communication subtree

- $f_{\vee}^{\mathrm{p}}$, $f_{\wedge}^{\mathrm{p}}$, $f_{\vee}^{\mathrm{o}}$, $f_{\wedge}^{\mathrm{o}}$, $f_{\mathrm{c}}^{\mathrm{p}}$, $f_{\mathrm{c}}^{\mathrm{o}}$ are operators on D which correspond to proponent's disjunctively refined node, proponent's conjunctively refined node, opponent's disjunctively refined node, opponent's conjunctively refined node, proponent's countered node and opponent's countered node, respectively. These functions are used to quantify all the subtrees of a considered tree, as well as the tree itself.

Quantification of subtrees works as follows:

Subtrees rooted in a node which is refined but not countered

- An `attribute value` calculated for a subtree rooted in a disjunctively refined proponent (resp. opponent) node results from the application of the function $f_{\vee}^{\mathrm{p}}$ (resp. $f_{\vee}^{\mathrm{o}}$) to the `attribute values` calculated for its refining subtrees.
- An `attribute value` calculated for a subtree rooted in a conjunctively refined proponent (resp. opponent) node results from the application of the function $f_{\wedge}^{\mathrm{p}}$ (resp. $f_{\wedge}^{\mathrm{o}}$) to the `attribute values` calculated for its refining subtrees.

Subtrees rooted in a node which is not refined but countered An `attribute value` calculated for a subtree rooted in a proponent (resp. opponent) node results from the application of the function $f_{\mathrm{c}}^{\mathrm{p}}$ (resp. $f_{\mathrm{c}}^{\mathrm{o}}$) to the initial value for the non-refined root node and the `attribute value` calculated for the countering subtree.

Subtrees rooted in a node which is refined and countered In this case, first the `attribute value` 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 function $f_{\mathrm{c}}^{\mathrm{p}}$ or $f_{\mathrm{c}}^{\mathrm{o}}$ are applied to such previously calculated `attribute value` for the refining part and `attribute value` for the countering part.

Note that, since disjunctively and conjunctively refined nodes can have an arbitrary number

of children, the functions $f_{\vee}^{\mathrm{p}},\ f_{\wedge}^{\mathrm{p}},\ f_{\vee}^{\mathrm{o}},\ f_{\wedge}^{\mathrm{o}}$ are unranked[1]. Functions $f_{\mathrm{c}}^{\mathrm{p}},\ f_{\mathrm{c}}^{\mathrm{o}}$, in turn, are binary because each node can have one countermeasure only.

In Section 2.3.2, we show how to apply the bottom-up procedure to the calculation of the **minimal cost of the attacker**.

2.3.2 Minimal Cost of the Attacker

We are interested in calculating the **minimal cost** of a successful attack in the RFID warehouse scenario. We consider a situation where the attacker does not have any precise information on how the defender decides to protect the warehouse. Thus, we assume that *all possible defenses illustrated on the ADTree are in place and that they are fully functional*, i.e., a defense attached to an attack node foils the corresponding attack component, unless the defense itself is rendered useless by a counterattack.

In order to make our analysis more precise, we use pairs of the form (`cost value`, confidence value) to quantify the nodes and subtrees. The following cost values are considered: `Cheap` < `Average` < `Expensive` < `Extreme`, with the following meaning:

- `Cheap`: Any attacker can afford this without thinking twice.
- `Average`: The costs of the attack will fend of most attackers without a steady income.
- `Expensive`: The attacker will need substantial funding in order to perform the attack.
- `Extreme`: Extremely high cost that the attacker is not able to afford.

The confidence value represents how confident we are in the corresponding `cost value`. We can chose one of the three confidence levels Low, Medium and High, with the following intuitive meaning:

- Low: Confidence below 25%
- Medium: Confidence between 25% and 75%
- High: Confidence above 75%.

We start by initializing the values at the non-refined nodes of the tree.

- In the case of the attacker's non-refined nodes, the pair assigned to a node represents the `cost value` of the corresponding component, and the confidence level tells us how confident we are in the assigned `cost value`. For instance, if we assign the pair (`Average`, `Low`) to an attack node, this means that, in our opinion, the cost of performing the action represented by the node is `Average`, but our confidence in this value is below 25%.
- All non-refined nodes of the defender are initialized with the pair (`Extreme`, `High`). This indicates that we are fully confident that it is extremely expensive (and thus impossible) for the attacker to be successful at a defender's action.

Such initial values allow us to express the **cost** of the considered scenario from the point of view of the attacker.

Next, we describe how we calculate the **minimal attacker's cost** for all subtrees.

[1] *An unranked function* F with domain D and range R denotes a family of functions $(F_k)_{k\in\mathbb{N}}$, where $F_k\colon D^k \to R$, for $k > 0$.

Subtrees rooted in a node which is refined but not countered

- The `cost value` calculated for a subtree rooted in a disjunctively refined attack (resp. defense) node is defined as the minimum (resp. maximum) of the `cost values` calculated for its refining subtrees.

- The `cost value` calculated for a subtree rooted in a conjunctively refined attack (resp. defense) node is defined as the maximum (resp. minimum) of the `cost values` calculated for its refining subtrees.

The maximal confidence level corresponding to the chosen cost value is propagated.

Subtrees rooted in a node which is not refined but countered The `cost value` 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 `cost value` calculated for the countering subtree. The maximal confidence level corresponding to the chosen cost value 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 `cost value` for the refining part.

Some words of explanation might be useful. In this framework, we chose the minimal value for an attacker disjunctive subtree, because we are interested in the **minimal cost**. Thus, we suppose that the attacker always performs the least expensive option. Moreover, we assume that performing several actions belonging to the same **cost** category is not more expensive than performing one of such actions. Therefore, we chose the maximal **cost** in the case of a conjunctive subtree with the attacker's root.

In order to successfully disable a disjunctively refined defensive countermeasure, the attacker has to disable *all* corresponding refining options. Therefore, the operator used in this case is maximum. 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 choses the least expensive solution. Thus, the operator used in this case is minimum.

We would like to remark that the functions used to calculate attribute values depend on the considered attribute and additional assumptions. Thus, if we would be interested in calculation of the defender maximal **cost**, 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 that they are fully functional.

With the assumption that all the possible defenses are present and fully functional, the real minimal **cost** 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 **cost** of the corresponding counterattacks will not be taken into account in the final **cost** of the attacker. However, by taking the described approach we use a safe solution, in the sense that

- the calculated **minimal cost** will not be lower than the actual minimal cost, i.e., the **minimal cost** 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.

2.4 Literature for Interested Students

2.4.1 Attack Trees Origins

The literature on attack trees is abundant. Piètre-Cambacédès and Bouissou [PCB10] have given a historical overview on graphical representations of computer attacks, such as fault trees [VGRH81], threat trees [Amo94] and privilege graphs [DD94], and how these representations lead Schneier to coin the term attack tree [Sch99].

2.4.2 Formalization of Attack Trees

In 2005, the work of Mauw and Oostdijk [MO05] augmented attack trees with semantics, providing a solid, formal and methodological framework for security assessment. This work also formalizes the bottom-up approach for attribute evaluation on attack trees.

2.4.3 Formalization of Attack–Defense Trees

Attack–defense trees where formally defined in [KMRS10]. This work introduces several useful semantics for attack-defense trees and extends the bottom-up approach for attribute evaluation from attack trees to attack–defense trees.

The authors of [KPS11] show that, for a large class of semantics, extending attack trees with defensive measures was not done at the expense of computational complexity.

2.4.4 Attributes for Attack Trees

Many authors treat how to add different kind of values to attack trees and related models. Here we recommend two papers which propose how to define a new attribute by combining several existing attributes. Edge et al. [EDRM06] have defined a variant of a **risk** attribute for attack trees based on the formula $\textbf{risk} = (\textbf{probability}/\textbf{costs}) * \textbf{impact}$. A similar approach has been used in [JW08], 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. These combination methods allow us to estimate values corresponding to more complex properties, for which it would be difficult to provide values directly.

2.4.5 Case Studies and Experience Reports

Examples of deeper studies and experience reports with attack tree methods applied to real-life system can be found in Henniger et al. [HAF+09], who have conducted a study using attack trees and a variety of node attributes for vehicle communications systems, Abdulla et al. [ACK10] with an analysis on the GSM radio network, and Tanu and Arreymbi [TA10] using vulnerability tree, fault tree and attack tree analysis on a mobile SCADA system for a multiple tank and pump facility.

2.5 The ADTree Methodology for Performing Case Studies

The bigger picture: Theoretically we understand the methodology, but can we apply it already?

Practice: The ADTree methodology for performing case studies can be summarized in the following 6-steps:

- Step 1: Creating the tree.
- Step 2: Selecting attributes.
- Step 3: Choosing who estimates the attribute values.
- Step 4: Estimating values.
- Step 5: Combining values.
- Step 6: Performing the bottom-up procedure.

In all six steps, numerous design choices have to be made. The choices depend on the concrete modeling purpose, the scenario, the client and last but not least the person executing the analysis.

2.5.1 Step 1: How to create a tree?

- Which node labels to choose? (E.g., a CEO (chief executive officer) might be unfamiliar with some technical terms.)
- When to stop refining nodes (E.g., we stop refining the nodes, when we have an intuitive grasp for all basic action.) Note that the more detailed the tree is, the less reusable it becomes.
- Do we make use of libraries? (This might hinder creativity.)
- Do we design a tree once and freeze its structure, or do we allow changes to the tree at all times? (Estimates might have to be redone.)

Important: Study the system, make use of the available material and think about for whom we are analyzing the scenario.

2.5.2 Step 2: Which attributes are interesting?

- Choose attributes.
  Here is a non-exhaustive list of possible attributes:
  Costs, detectability, difficulty, impact, penalty, profit, probability of success, probability of occurrence, special skill, time, severity, consequence, number of necessary specialists, number of working hours, any binary question like: Is this action satisfiable? / Is electricity needed? / Do you disagree with this node?, etc.

Important: When selecting attributes, always provide a *objective, written* description of the attribute to ensure comparability between different people who estimate values. For each attribute define and describe a corresponding value domain.

- Choose meta-attributes.

Meta-attributes are properties which describe or detail attribute values and can only be used in combination with an attribute.

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, detailing 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. To summarize, using meta-attributes allows us to model the desired properties more accurately, and to improve their quantification.

Important: Always provide a value domain and a written description for meta-attributes, as has been done for attributes.

2.5.3 Step 3: Who should estimate attribute values for which nodes?

First select suitable people who are supposed to estimate:

- Specialists (security experts, system owners).
- Attackers, defenders, both attackers and defenders.
- Students in a security class.
- Everyone in a pedestrian zone.

Then select the nodes and the attributes which the selected people are supposed to estimate. The match-up of people and attributes can be done

- according to the role of the estimators: (attackers: attack nodes, defenders: defense nodes).
- according to the specialization of the estimators (psychologists: social engineering, security experts: digital attacks).
- depending on attribute (accountants: costs, is electricity needed: technical personnel).
- for all nodes, for intermediate nodes, non-refined nodes.

2.5.4 Step 4: How to estimate values?

- In context, that means that the tree must be available.
- Are different values for nodes with the same label allowed? (This could uncover that someone assigns values randomly.)
- Set a time limit for the estimation.

As a result of this step, we obtain values. These values, however are very different and often not immediately comparable. In the case study ten students were supposed to each fill in 57 values. For 35 nodes out of 570 nodes $(6,1\%)$ one or more student did not provide a value. These 35 missing values were spread over 19 different nodes. For 6 nodes, there were values were filled in, even though they were refined nodes, which were supposed to stay empty.

2.5.5 Step 5: Combining Values

Having to combine values is a typical problem which arises when the same data is collected from several sources: We have up to n values for every non-refined node, but we only want one which is comparable to the values assigned to other nodes. There are two cases that can occur:

- Every one has filled in the same value; very unlikely. Example: Buy Warehouse: 10x(E,3).
- The values differ; very likely. Example, every other node in the case study.

In the first case, we have an obvious suitable representative, in the latter case we are looking for an automated procedure to select a suitable representative. Selecting a such a suitable representative automatically could be done by means of:

- a formula (2.1).
- a majority selection.
- an average computation. (It is not always clear how to compute an average, e.g. in the case of binary or discrete variables without a given order.)
- a minimization. (For this we also need to define an order.)

Or, in case we cannot use an automated procedure, or do not know how to use an automated procedure, we could

- decide on a representative at a consensus meeting.

When using a formula, we would first transform discrete categories into numbers. In our case study we use the following bijection:

$$b\colon \{\mathrm{C}, \mathrm{A}, \mathrm{E}\} \to \{1, 2, 3\},$$

where C$\mapsto$1, A$\mapsto$2, E$\mapsto$3. Then we use the following formula to compute a representative:

$$(\mathrm{rnd}\left(\frac{\sum \text{attribute} \cdot \text{confidence}}{\sum \text{confidence}}\right), \left\lfloor \frac{\sum \text{confidence}}{n} \right\rfloor). \tag{2.1}$$

Here,
- rnd: regular rounding (best estimate).
- $\lfloor\cdot\rfloor$: rounding down to the nearest integer (round down to reflect risk averseness).

Example: Attend visitor's log. The ten estimated value are: 4x(C,3), 3x(A,2), 1x(E,1), 1x(A,1), 1x(C,1).

We compute the first component:

$$\mathrm{rnd}(\frac{4\times1\times3+3\times2\times2+1\times3\times1+1\times2\times1+1\times1\times1}{4\times3+3\times2+1\times1+1\times1+1\times1}) = \mathrm{rnd}(\frac{30}{21}) = 1.$$

Then we compute the second component:

$$\left\lfloor \frac{4\times3+3\times2+1\times1+1\times1+1\times1}{10} \right\rfloor = \left\lfloor \frac{21}{10} \right\rfloor) = 2.$$

Using b^{-1}, we transform the first component of the result $(1, 2)$ back into the original domain and obtain (C,2).

Depending on the comparability of the node values, we might not want to treat all nodes in the same way. To handle this, we partition the nodes into categories: (To guarantee a partition, nodes that could be placed in several categories are placed in the highest category.)

- Category 1: Nodes with as many attributes as players
  Example: Attend visitor's log.
- Category 2: Nodes where at least one player has not estimated a value.
  Example: Mostly non-refined nodes that are countered, which are countered.
- Category 3: Nodes where all estimated values have a low confidence.
  Example: Be sent to the warehouse: 5x(C,2), 1x(A,2), 1x(A,1), 3x ϵ.
- Category 4: Nodes where the values diverge significantly.
  Example: Presence detectors on premises. 8 different values of 9 possible.
- Category 5: Nodes where the **disagree** flag is set.
  (Not possible in class case study.)
- Category 6: Nodes where no player has estimated a value.
  (Did not occur in class case study.)

Categories 1–2 we can treat with an automated procedure: Formula (2.1), the other categories should be treated differently, for example a representative should be selected at a consensus meeting.

2.5.6 Step 6: Bottom-up Procedure

Now, as in the homework use a reasonable attribute domain. For the question "What are the minimal **costs** of an attacker, provided all defenses are in place", we use the following:

- Value domain:
  $\mathbb{T} = \{\texttt{Cheap}, \texttt{Average}, \texttt{Expensive}\} \cup \{\texttt{eXtreme}\}$,
  $\mathbb{S} = \{1, 2, 3\}$.
  Choose a category order, when discrete domains are used and when order has not already been chosen in Step 3. We need to extend the value domain to be able to express that defense nodes can never be satisfied by the attacker.
- Six functions: If the value domain of an attribute domain has two dimensions, the range of the six functions also has two dimensions:

$$
\begin{aligned}
&D = \mathbb{T} \times \mathbb{S},\\
&f^{\mathrm{p}}_{\vee}((c_1, x_1), (c_2, x_2)) = (\min\{c_1, c_2\}, \max\{x_j \mid c_j = \min\{c_1, c_2\}\}),\\
&f^{\mathrm{p}}_{\wedge}((c_1, x_1), (c_2, x_2)) = (\max\{c_1, c_2\}, \max\{x_j \mid c_j = \max\{c_1, c_2\}\}),\\
&f^{\mathrm{o}}_{\vee}((c_1, x_1), (c_2, x_2)) = f^{\mathrm{p}}_{\wedge}((c_1, x_1), (c_2, x_2)),\\
&f^{\mathrm{o}}_{\wedge}((c_1, x_1), (c_2, x_2)) = f^{\mathrm{p}}_{\vee}((c_1, x_1), (c_2, x_2)),\\
&f^{\mathrm{p}}_{\mathrm{c}}((c_1, x_1), (c_2, x_2)) = f^{\mathrm{p}}_{\wedge}((c_1, x_1), (c_2, x_2)),\\
&f^{\mathrm{o}}_{\mathrm{c}}((c_1, x_1), (c_2, x_2)) = f^{\mathrm{p}}_{\vee}((c_1, x_1), (c_2, x_2)).
\end{aligned}
$$

The first four functions extend to the k-ary case straightforwardly. The six functions above express in mathematical notation, what is written in Section 2.3.1.

Finally, there are some things that also need to be observed.

- Not all properties may be computable with the bottom-up procedure.
- In general we overestimate values, because we usually assume that all defenses are in place.
- We need not specify how many `Cheap` actions make up an `Average`. (We actually assume that even if we execute 10 actions which are `Cheap`, the result is still `Cheap`.)
- In *ADTree* we do not compare estimated values of refined nodes with corresponding computed values.

2.5.7 Summary

Table 2.1: 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 template. - Use /do not use incomplete trees. - Continuously improve trees/**freeze tree structure at some time**. - Use concise noun and verb as node label/detailed textual description. - 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 some attributes given in Section 2.5.2 /provide new attributes. - Select attribute domains: discrete/reals/fuzzy sets/intervals/ probability measures. - Select number of discrete categories. - Allow/disallow (**disagree with node** attribute). - Always/sometimes use meta-attribute confidence.
3	**Choose who estimates what**	- Who estimates: attackers/defenders/specialists/random person. - 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 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 (2.1)/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.

Bibliography

[ACK10] Abdulla, Parosh A. ; Cederberg, Jonathan ; Kaati, Lisa: Analyzing the Security in the GSM Radio Network Using Attack Jungles. In: Margaria, Tiziana (Hrsg.) ; Steffen, Bernhard (Hrsg.): *ISoLA (1)* Bd. 6415, Springer, 2010 (LNCS). – ISBN 978–3–642–16557–3, S. 60–74

[Amo94] Amoroso, Edward G.: *Fundamentals of Computer Security Technology*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1994 `http://portal.acm.org/citation.cfm?id=179237#`. – ISBN 0–13–108929–3

[DD94] Dacier, Marc ; Deswarte, Yves: Privilege Graph: an Extension to the Typed Access Matrix Model. In: *ESORICS* Bd. 875, 1994 (LNCS), 319-334

[EDRM06] Edge, Kenneth S. ; Dalton II, George C. ; Raines, Richard A. ; Mills, Robert F.: Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security. In: *Proceedings of the 2006 IEEE conference on Military communications*. Piscataway, NJ, USA : IEEE Press, 2006 (MILCOM'06). – ISBN 1–4244–0618–8, 953-959

[HAF+09] Henniger, Olaf ; Apvrille, Ludovic ; Fuchs, Andreas ; Roudier, Yves ; Ruddle, Alastair ; Weyl, Benjamin: Security requirements for automotive on-board networks. In: *9th International Conference on Intelligent Transport Systems Telecommunications,(ITST)*. Lille, 2009, S. 641–646

[JW08] Jürgenson, Aivo ; Willemson, Jan: Computing Exact Outcomes of Multi-parameter Attack Trees. In: Meersman, Robert (Hrsg.) ; Tari, Zahir (Hrsg.): *OTM Conferences (2)* Bd. 5332, Springer, 2008 (LNCS). – ISBN 978–3–540–88872–7, S. 1036–1051

[KMRS10] Kordy, Barbara ; Mauw, Sjouke ; Radomirovic, Sasa ; Schweitzer, Patrick: Foundations of Attack–Defense Trees. In: Degano, Pierpaolo (Hrsg.) ; Etalle, Sandro (Hrsg.) ; Guttman, Joshua D. (Hrsg.): *FAST* Bd. 6561, Springer, 2010 (LNCS). – ISBN 978–3–642–19750–5, S. 80–95

[KPS11] Kordy, Barbara ; Pouly, Marc ; Schweitzer, Patrick: Computational Aspects of Attack–Defense Trees. In: *Security & Intelligent Information Systems*, Springer, 2011. – to appear

[MO05] Mauw, Sjouke ; Oostdijk, Martijn: Foundations of Attack Trees. In: Won, Dongho (Hrsg.) ; Kim, Seungjoo (Hrsg.): *ICISC* Bd. 3935, Springer, 2005 (LNCS). – ISBN 3–540–33354–1, 186-198

[PCB10] Piètre-Cambacédès, Ludovic ; Bouissou, Marc: Beyond Attack Trees: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP). In: *European Dependable Computing Conference*. Los Alamitos, CA, USA : IEEE Computer Society, 2010. – ISBN 978–0–7695–4007–8, S. 199–208

[Sch99] Schneier, Bruce: Attack Trees. In: *Dr. Dobb's Journal of Software Tools* 24 (1999), Nr. 12, 21-29. http://www.ddj.com/security/184414879

[TA10] Tanu, Eedee ; Arreymbi, Johnnes: 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)*, University of East London, School of Computing, Information Technology and Engineering, 2010. – ISBN 978–0–9564747–0–4, 228-242

[VGRH81] Vesely, W. E. ; Goldberg, F. F. ; Roberts, N.H ; Haasl, D.F.: Fault Tree Handbook / U.S. Regulatory Commission. Version: 1981. http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf. 1981 (NUREG-0492). – Forschungsbericht