Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
Eindhoven Computer Science Security Group

Otto-von-Guericke Universität Magdeburg
Department of Computer Science
Institute of Technical and Business Information Systems
Reseach Group Multimedia and Security

Internship Thesis

Design and Implementation of a Support Tool for Attack Trees

Alexander Opel

Supervisors:

Dr. Sjouke Mauw (Technische Universiteit Eindhoven),
Andreas Lang (Otto-von-Guericke Universität Magdeburg)

Opel, Alexander:
Design and Implementation of a Support Tool for Attack Trees
Internship Thesis,
Otto-von-Guericke University
Magdeburg, March 2005.

Acknowledgements

Many thanks to all the people who contributed to the success of my internship and this thesis. My supervisor Sjouke Mauw who offered me a big scope to realize this project and so I had a free view on Attack Trees. Martijn Wolfs for his discussions about Attack Trees at the beginning of the project. My office mate Koos Gadellaa for his great support on programming issues and experienced talks. Thank you for the fun between the working sessions. My german fellow students Lennart Nacke and Maurice Hollmann for their support with LaTeX and their ICQ and Skype talks. Sandra Gebbensleben from the ITI Research Group on Multimedia and Security for her feedback. Also I want to thank all the other people of the Formal Methods group in Eindhoven. The daily lunch was always fun.

And of course thanks to my family especially my father who was nearly always daily attainable (the internet is a great invention). Thank you.

Alexander Opel, March 2005

Declaration

I declare hereby, that the present work was created independently and only with allowed appliances.

Eindhoven, March 11, 2005 Alexander Opel

Overview

Viruses, worms and trojans are only some threats which computer systems are exposed nowadays. To get an overview about all possible attacks to a system, Attack Trees could be used. Attack Trees were first introduced in the literature by Bruce Schneier in 1999 [1]. In the december redaction of the Dr. Dobb's Journal his article has been published. Thereby, he describes Attack Trees as follows. "Attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks. Basically, you represent attacks against a system in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes." An Attack Tree can represent each opportunity for an attack that might be used to compromise systems. In that way, it is possible to predict how and where intruders pose the greatest threats against a system [2]. Reasonable precautions for securing the system can be taken with these information.

However, this article offers only a rough overview of the possibilities which the Attack Trees can offer. In the literature, which deals with Attack Trees, often only a reference to this paper can be found. Further information, which go more deeply into the detail, are missing and not available respectively. The intention of this internship was to get a deeper view on Attack Trees, to find out where their origin can be found and what problems might occur with the Attack Trees described by Bruce Schneier and how they can be solved. In addition some new extensions have to be worked out which might make Attack Trees more useful and extend their function range.

The completeness of an Attack Tree depends on the overview the originator has of the whole system, and the possible events that might occur to the system. In small systems it is possible to keep a complete overview, but in a big environment the overview is rapidly lost [3]. In order to deal with this, the second goal of this internship was to implement a freeware tool, that offers the opportunity to represent an Attack Tree graphically. Thereby, it is easy to expand an Attack Tree, and if new attacks occur it is possible to intervene fast. Thus, the tool has to deal with the extension of an Attack Tree, too. Therefore a complete new program has to be designed. This covers the creation of an user-friendly graphical user interface, the necessary data structures and algorithms, and a file format for a later reuse of the Attack Trees.

At the end of this internship a software tool prototype was implemented, a XML-based file format was adapted and three new extensions were analysed and integrated into the program. Thereby some new cases arised and their possible solutions were worked out.

Parties involved

This internship was completed at the Eindhoven University of Technology, the Netherlands.

On the part of the TU/e this project was supervised by Sjouke Mauw, associate professor for the department of "Mathematics and Computer Science". He is member of the "Formal Methods" group and co-founder of the "Eindhoven Computer Science Security Group".

The supervisor from the University of Magdeburg for this project was Andreas Lang, associate for the "Department of Computer Science". He is member of the "Research Group Multimedia and Security" at the "Institute of Technical and Business Information Systems".

Tasks

The main goal of this internship was to design and implement a support tool for Attack Trees. For that purpose an overview of Attack Trees had to be worked out at the beginning. This included the study of literature, existing tools and the talk with people that are more experienced in the field of Attack Trees. Additionally, some new expedient extensions should be analysed. For this purpose the ideas of three extensions were given: The enhancement of the calculation formulas, the combination of same content in different tree parts and the possibility to add countermeasures directly in the Attack Tree.

The sub-goals were to get a definition of an Attack Tree – also for the new extensions – , to design the program, an implementation of the design and finally a test of the tool. The design of the program included the necessary data structures, the algorithms, the graphical user interface and the inputs and outputs of the program. The program ought to be written in a modular fashion, to make it easy extendable. In addition, the task was to improve the tool by an incremental approach of the design and the implementation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Nowadays the security of computer systems is a very important area in the information technology industry. By the increasing cross-linking of computer systems and the associated risks like trojans, viruses and Distributed Denial of Service (DDoS) attacks[1] this industry gains more significance. The accociated possible threats like the unintentional stealing of passwords, the destruction of data or the attempt to make computer networks unattainable can be life-threatening for a company. To deal with new security threats, computer companies spend much money. Before money is spent for security issues, the causes and the attack possibilities respectively have to be worked out. For this task the risk analysis can be consulted.

As a part of the risk analysis, the Attack Tree analysis offers possibilities to find out such attacks and causes – obvious threats as well as initially not regarded threats. Since these can produce harm to security relevant systems, the Attack Tree analysis helps to secure systems by finding preferably all attacks. The advantage of the Attack Trees is the easy understanding of this method and the possibility of receiving fast results. Thus, it is usable for both beginners and professionals. Additionally, it can be used for simple as well as harder problems.

Since it is easy to get familiar with possible eventualities, the Attack Tree method can also be used for brainstorming. However, in such cases the work with pencil and paper is not very suitable. Easily operated software can support the work with Attack Trees. Software tools give the opportunity to make changes shortly and show

[1] A denial-of-service attack (also, DoS attack) is an attack on a computer system or network that causes a loss of service to users, typically the loss of network connectivity and services by consuming the bandwidth of the victim network or overloading the computational resources of the victim system. Attempts to "flood" a network with bogus packets, thereby preventing legitimate network traffic, are the most common form of attack, often conducted by disrupting network connectivity with the use of multiple hosts in a distributed denial-of-service attack or DDoS. Such attacks can consume the resources of intervening systems and networks over which the attack is transmitted. [23]

results immediately. This supports the search for possible vulnerabilities.

Since Attack Trees were introduced in 1999 by Bruce Schneier [1], the risk analysing method of Attack Trees was not followed up intensively. Only in the last years it is mentioned in papers as a simple examination method for security relevant systems. But information which goes deeper into detail are available barely.

With the todays state of the art, only the company Amenaza Technologies Limited [22] offers a high sophisticated tool – Secur*IT*ree – for Attack Tree modeling. This tool supports also the creation and modification of Attack Trees. Also it has a large scale of analysis techniques. But Secur*IT*ree is an commercial program. Only a time-limited demo version is available for free. Thus no extensions can be made for new researches like it would be possible with open source software. Also the program has a closed file format. This prevents a further processing of Attack Trees with other resources or programs.

Thus this internship offers the possibility to get deeper into the field of Attack Trees by creating a new software tool. By the implementation of the Attack Tree method with new functionalities in a new tool, new experiences can be won, which can serve as basis for later research.

1.2 Expected Results

The expected result of this internship was a tool that can support the work with Attack Trees. For this reason the implemented tool should make it easy to create, manipulate and visualize Attack Trees. This included the support of standard editing functions like creating, deleting and changing parts of an Attack Tree. Also the new extensions mentioned in the *Task*-Section should be implemented and the experience with these new functions had to be analysed. Additionally, the tool should supported simulations of attackers profiles in order to examine possible danger scenarios. To inspect the tool, some tests had to be made. The results of the tests should help to improve the tool in next development stages. To support a later reuse of an Attack Tree, a XML-based file format had to be worked out as output of the program. Since XML is an open standard, it is easy for other users to work with created Attack Tree in further processings. Also the structured content and the easy processing of XML files is advantageously.

1.3 Thesis Structure

This internship thesis is divided into six chapters. The following Chapter two examines the origin of Attack Trees in the Risk Analysis. For this purpose similar techniques are described and Bruce Schneier's view on Attack Trees like he published in a paper [1] are explained. The next chapter shows how Attack Trees could be extended with additional functionality. Three extensions were analysed and why and/or how they can be used. Chapter five will show how the implemented Attack Tree software tool is designed and how the Attack Trees with the extension mentioned above are realized. This chapter also takes a look on the XML-based file format, how the Attack Trees are stored and how the extensions are integrated. In the next chapter the experiences with the software tool during some program tests are discussed. Thereby, the software tool was analysed in general, and the realisation of the extensions and the calculation process were examined. Possible problems are shown and their solutions are given. Finally, in the sixth chapter a conclusion is given and some suggestions for further development are made.

Chapter 2

Risk Analysis

This chapter explores the origin of Attack Trees. Similar techniques, which also have their roots in the risk analysis, are explained and the Attack Trees like Bruce Schneier described them are analysed.

Decisions are made every day. For each decision the risks must be balanced, which influences the choice. With the help of the experiences in the past the estimation of risks has been learned. The risk level decides whether the options, which are at the disposal, are accepted or rejected. Mostly, it is the goal to reduce or avoid the risks.

Often the risk structure of a given problem is very complex. Therefore it can happen that personal experiences are not sufficient in order to estimate a risk. In the past, statistics were used to face that problem. Statistics make it possible to foresee probabilities of a possibly arriving event in order to meet appropriate arrangements. In some situations, statistics of past events are not sufficient during the examination of a risk. This is usually the case with consciously accomplished, malicious attacks. Such attacks often cause large harm with relatively small exertion.

However, each decision can also be criticized – independently which option has been chosen. Was a possibility ignored or was a possibility neglected and/or regarded too intensively? To solve these questions the risk analysis can be used [15].

In system design and realisation there are some established procedures to discover errors and vulnerabilities and trace them back to their origin. But mostly the procedures are used in a safety context rather then in security. In the safety area usually events play a role, which are caused unintentionally or by higher force like environmental influences. These events are coined/shaped by a coincidental distribution.

In the security area events are provoked methodically and malicious which permits hardly a meaningful probability modeling. Therefore often only rough probability rasters are present. They permit only a qualitative statement. Qualitative security adapted analysis methods help to recognize and evaluate malicious influences of the environment on the system. For example, they can be used during the creation process of computer software or for analysis of security areas like financial institutions or strongrooms.

In every state of a development process, risk analysis can be used. Since vulnerabilities have often their origin in fundamental lacks, system requirements and rough specifications should be analyzed accurately. For example wrongly classified environmental and application conditions for a system, or missing relevant system functions, which are ignored in the specification and which have to be integrated in later phases.

These aspects can entail malfunctions. In order to avoid these malfunctions and to ensure safety/security, the risk analysis offers different analysis techniques.

The Preliminary Hazard Analysis (PHA) offers first evaluations and recognitions of system-critical ranges. It is used in later states of the requirement analysis and in early states of the development process. With the PHA critical areas can be found and first evaluations for further processings can be done. But this method isn't good formalized. Typically a brainstorming is used for obtaining results.

In order to evaluate potential weak points in the concept, formalized analysis techniques can be accomplished. They can be categorized in forward-arranged analysis techniques and backward-arranged analysis techniques. [16]

2.1 Forward-arranged Analysis Techniques

Starting point of the forward-arranged analysis is an elementary disturbance of a system, which emanate from user, environment or subsystem error. This analysis technique points out, how the disturbance spreads in the system. Whether it increases or decreases or it is noticed as externally observable failure. Typical forward-arranged analysis techniques are the *Failure Modes and Effects Analysis* and the *Event Tree Analysis*.

2.1.1 Failure Modes and Effects Analysis

Failure Modes and Effects Analysis, or FMEA, is a methodology for identifying the potential failure modes (e.g. a battery fails to provide adequate power, a switch fails to open or a motor fails to operate) that a product or process may encounter. The risks associated with these failure modes are assessed and prioritized according to their urgency afterwards. Thus, the prevention of the more urgent failure modes, i.e., the ones that are most likely to cause serious harm to a company, can be done. Generally, the outcome of the FMEA is a table, which documents each component's failure mode, it's cause, and the corresponding effects [24]. See Appendix A.2 for an example.

"The FMEA is a proactive analysis tool, allowing engineers to anticipate failure modes even before they happen, or even before a new product or process is released. It also helps the engineer to prevent the negative effects of these failure modes from reaching the customer, primarily by eliminating their causes and increasing the chances of detecting them before they can do any damage." [17]

The Failure Modes and Effects Analysis is a qualitative and quantitative technique, which judges from well-known causes to unknown effects and is an inductive method accordingly. Since the FMEA needs the knowledge of the whole system, it is used late in the life cycle [16].

In 1949 the FMEA process was originally developed by the US military. Afterwards it was used for applications in space travel, chemical industry and car development. It is an integral part of any ISO 9000 compliant quality system [18].

2.1.2 Event Tree Analysis

The Event Tree Analysis, or ETA, also provides a qualitative and quantitative evaluation. The analysis is based on identification of the effects that a failure (of the initial event, i.e. the root of the tree) can produce. The ETA is based on binary logic, in which an event either has or has not occurred or a component has or has not failed. An Event Tree can be visualized as a tree structure. One node has only two children (yes or no, failed or not failed). Each path of the binary tree can be assigned with a probability of occurrence. At the end of each path the probability of the possible outcome can be calculated [6, 7].

Figure 2.1 shows a simplified Event Tree. The example[1] illustrates an Event Tree of a fire protection which is provided by a sprinkler system. If fire spreads, a detector will or will not detect the rise of the temperature. If the detector should fail, extensive damage would be the result. On the other hand the question would arise whether the fire alarm functions duly. No matter what the answer is, the fire could be brought under control by the fire sprinkler. However, the functional efficiency decides on the possible damage of the building. As Figure 2.1 shows, an event in the Event Tree has only two alternatives – a positive and a negative. Also only one branch of the Event Tree indicates that all subsystems have succeeded.

Event Trees also describe the path from well-known causes to unknown effects and they are also an inductive method accordingly. Towards the FMEA, the Event Tree Analysis can be used during each phase of the development process. The obvious disadvantage of the ETA is the exponential extension of the event combinations to the basis events. For n basis events there are 2^n event combinations. Thats why ETAs could only be used for small systems or small components of larger systems [19]. However, in literature this fact is not clear. In some sources the number of event combinations is less than 2^n. Thus, fewer event combinations are also possible

[1]taken from [7]

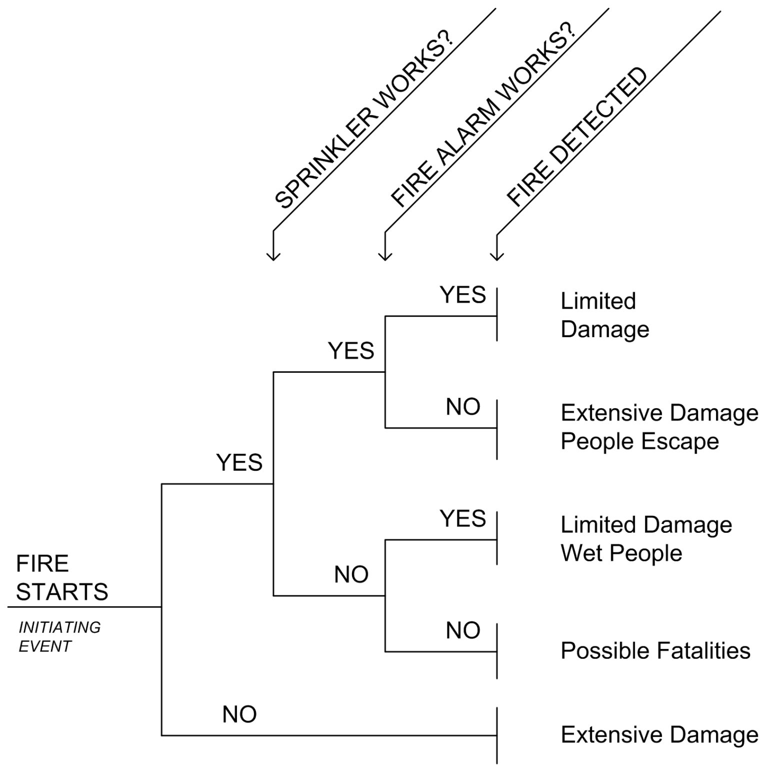


Figure 2.1: Simplified Event Tree

[7]. As in Figure 2.1 can be seen, the tree does not need to be divided further if no fire starts. It would not make sense to check, if fire is detected when no fire starts.

2.2 Backward-arranged Analysis Techniques

A backward-arranged analysis technique starts from an externally observed failure and explores the causes which may have produced the failure. These causes are reduced again to elementary causes until a detailed overview about the potential system weaknesses was created. With the knowledge of the weaknesses, suitable countermeasures can be met or the weaknesses can be rejected as irrelevant or negligible. A typical backward-arranged analysis technique is the *Fault Tree Analysis*. Particularly for the field of security another backward-arranged technique – the *Attack Tree Analysis* – is used.

2.2.1 Fault Tree Analysis

The goal of the Fault Tree Analysis, or FTA, is to determine possible combinations from causes which can lead to certain unwanted events. They are called Top Level Events, or TLEs. Unlike natural trees, Fault Trees grow downwards from above.

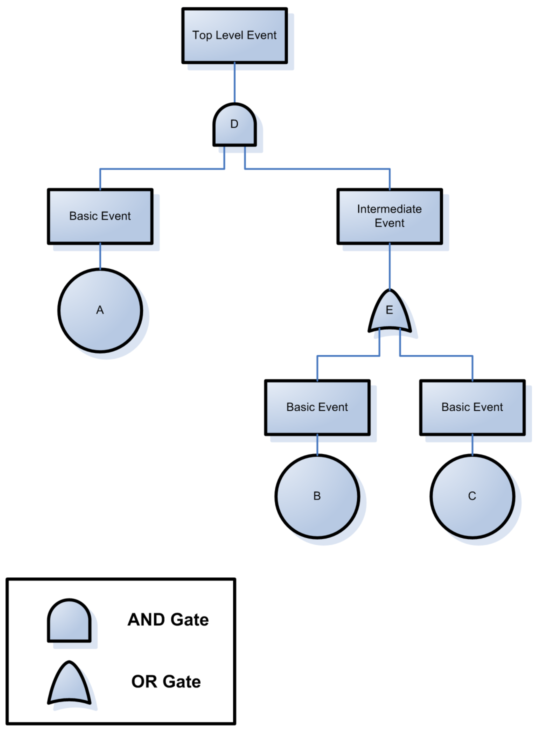


Figure 2.2: Simple Structure of a Fault Tree

Fault Trees consist thereby of several levels of events, which are so linked with one another that events of one level is the result of events of the underlying level. Thereby the events are linked by two different kinds of logical operators (gates) [16]. An AND gate represents a condition in which all the events shown below the gate must be present for the event shown above the gate to occur. This means that an event will only occur, if all of the direct underlying events occur simultaneously. An OR gate represents a situation in which any of the events shown below the gate will lead to the event shown above the gate. The event will occur only if one or any combination of the direct underlying events occures [20]. The further one pursues the nodes in the tree downward; the overview becomes the more precise. The lowest-level events in each branch of a Fault Tree - the leaf nodes of a tree - are referred to as Basic Events. These Basic Events are already precise enough. They do not need to refine further.

Figure 2.2 shows a simple structure of a Fault Tree. In this example[2] the Top Level Event (D) is only fulfilled if both events (A and E) of the underlying layer take place, since both are connected via an AND gate. In order to fulfill event E at least one of the Basic Events (B or C) at the lowest layer have to occur. Due to the fact that they are connected via an OR gate, it is already sufficient that one Basic Event succeeded.

[2]taken from [16]

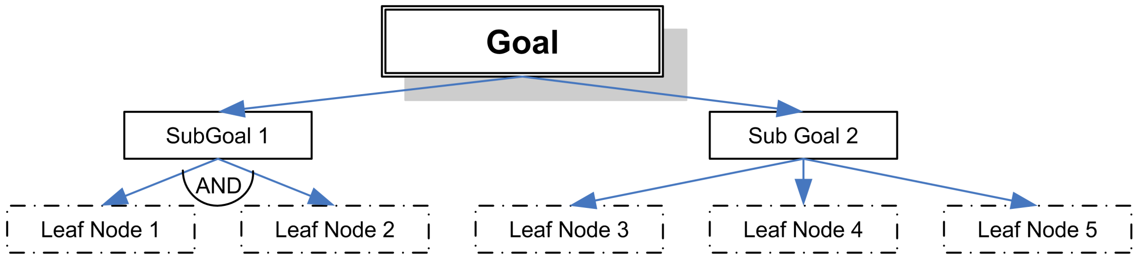


Figure 2.3: Simple Representation of an Attack Tree

This kind of Fault Tree describes how an event can occur, but not the chance of its occurrence. Therefore this simple Fault Tree model is reduced in its significance and needs to be extended.

Based on statistics of historical data, the probability of failure is given for the Basic Events which represents hardware, software and human failures [21]. With these probabilities it can be calculated how dependent components of this subsystem can be affected. Reliability statistics of individual components offer the opportunity to calculate the prospective reliability of the entire system.

The Fault Tree Analysis is a deductive technique which can produce a qualitative analysis of the system. With the extension of probabilities the Fault Tree Analysis can also be accomplished quantitative [16].

However, there are some events for which no statistics or historical data are available. These events are often produced by intelligent, hostile attackers. In order to be able to estimate risks, for which there are no statistics, the Event Tree technique must be extended. This extension is known as the concept of the Attack Trees.

2.2.2 Attack Tree Analysis

In december 1999 the Dr. Dobb's Journal published an article by Bruce Schneier about a method to describe the security of a system [1]. As the basis for his description he used varying attacks, how a system could be compromised. To illustrate a set of possible attacks with a certain goal, he formed a so called Attack Tree. In the root node of the tree the goal of an attack is shown. To achieve this goal there are different ways, described in child nodes of the root node. Each child node is thus a sub goal in the tree. The branches of the tree are divided until no further sub goals seems to be possible. A node without children is called a leaf node. Any path from a leaf node to the root node represents a possible attack against the goal of the root node.

In an Attack Tree there are two different type of nodes: AND nodes and OR nodes. At OR nodes a minimum of one sub goal has to be fulfilled to achieve the goal of the OR node. The sub goals are alternatives. At AND nodes each sub goal has to be achieved. Thus they form the steps to fulfill the goal. As one can see in

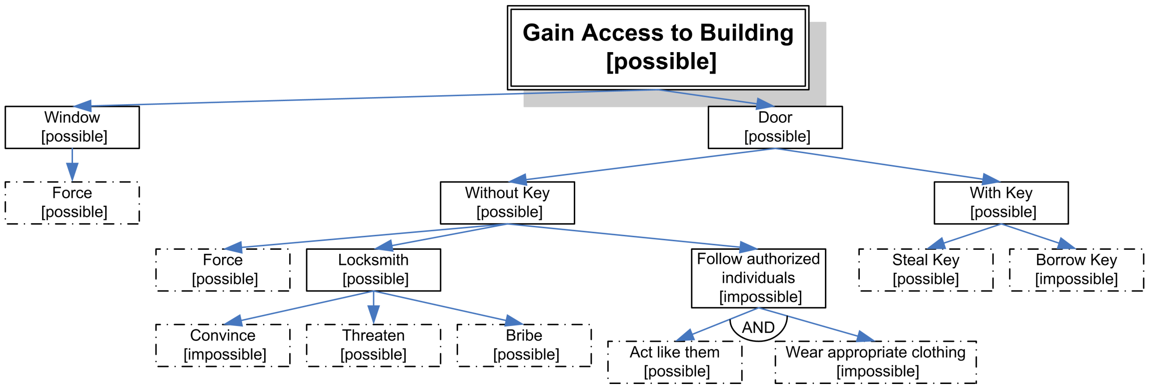


Figure 2.4: Simple Example of an Attack Tree

Figure 2.3 one has to achieve SubGoal 1 or SubGoal 2 for the root node goal. In order to achieve SubGoal 1, leaf node 1 and leaf node 2 have to be fulfilled. To achieve SubGoal 2, one of the remaining leaf nodes actions has to be fulfilled. Up to here the structure of an Attack Tree is similar to the Fault Tree structure mentioned in Section 2.2.1.

As determined before in this chapter, there are some threats where no statistics are available. Threats which originate from intelligent, hostile opponents. Nevertheless, there are some possible factors which can influence the behavior of the attacker. If an attack should be accomplished, an aggressor must have certain resources which are necessary in order to use the errors of the system. If the attacker has only limited resources available, his possible attacks are also restricted. Examples of such resources are the money which an aggressor needs to accomplish an attack, the technical skill level or the time which he needs to perform an attack. The Attack Tree Analysis deals with the limitation of resources to determine the probability of an attack more exactly.

For further explanations a running example will be taken. In Figure 2.4 an example can be seen for gaining access to a building. The intruder could gain the access by:

- breaking the window
- breaking the door
- convincing, threatening or bribing a locksmith
- acting like authorized individuals AND wearing appropriate clothings to follow these people into the building
- stealing the door key
- borrowing the door key to go inside the building with a key.

If a tree is built up, only values to the leaf nodes can be assigned, because these are the attack options for the system. The values of the AND and OR nodes are results of the values of their sub nodes. For example, as one can see in Figure 2.4 there are two different values a node can adopt. An action of a node can be *possible* or *impossible*. With the values in the leaf nodes, the values of the other nodes can be calculated. The value of an OR node is possible, if any of the children nodes is possible. The OR node is impossible, if all children are impossible. If all children of an AND node are possible, the AND node becomes also possible, consequently it gets impossible, if a minimum of one of the children is impossible. Besides *possible* or *impossible* other boolean values like *special equipment needed* or *no special equipment* et cetera can be assigned. The calculation of the node values takes place according to the same principle.

Furthermore it is possible to assign non-boolean values. For example the values of the leaves could be the costs of this special action or the time that is needed to perform the action. But with non-boolean values it is not possible to use the same calculation operations. To calculate the node values up to the root, Bruce Schneier defined that for an OR-node the cheapest child value represents the node value. For the AND nodes the summation of all child values (see Figure 2.5) gives the resulting outcome.

The combination of both value types is also possible. Thus the profile of an attacker can be described better. By reviewing an attacker profile the resources the attacker deals with are analysed – the money, time, technical skill level. For instance, the case where the intruder wants to break into the house with no equipment and in the shortest time can be considered. The attacker's profile could now be called *Shortest Attack with no Equipment*. The structured handling with the eventualities gives the possibility to learn much about the system's security. With this knowledge the vulnerabilities of the system can be secured. Also it is possible to create different attack profiles for different attack strategies or resources. In that way, adequate countermeasures can be developed. For example, it could be better to put a key at a safe place than to buy a safer and much more expensive lock.

Besides the graphical representation of a tree, a textual representation is also possi-

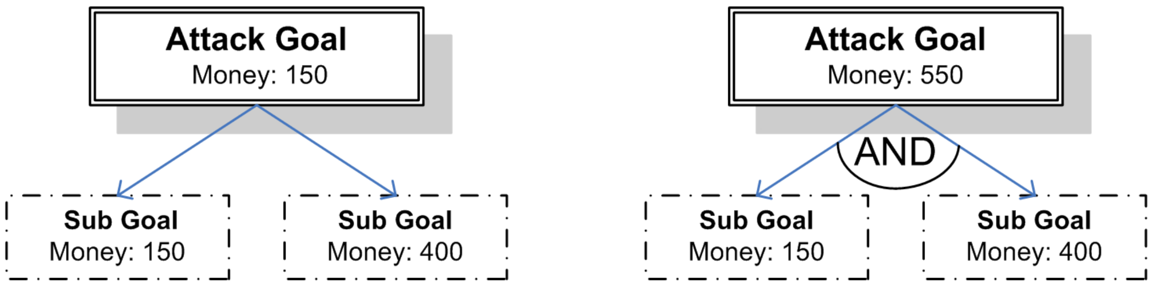


Figure 2.5: Simple Example of non-boolean values in OR and AND nodes

```
1 Goal: Gain Access To Building (OR)
  1.1 Window (OR)
    1.1.1 Force
  1.2 Door (OR)
    1.2.1 Without Key (OR)
      1.2.1.1 Force
      1.2.1.2 Locksmith (OR)
        1.2.1.2.1 Convince
        1.2.1.2.2 Threaten
        1.2.1.2.3 Bribe
      1.2.1.3 Follow Authorized Individuals (AND)
        1.2.1.3.1 Act Like Them
        1.2.1.3.2 Wear appropriate Clothing
    1.2.2 With Key (OR)
      1.2.2.1 Steal Key
      1.2.2.2 Borrow Key
```

Figure 2.6: Textual Representation of the running Example

ble. For this representation of the running example see Figure 2.6. This description serves as basis for the new developed tool mentioned in Chapter 4.

An undefined Case

In Bruce Schneier's paper an undefined case arises. If more than one value of the same type in the nodes is present, it is possible that the calculation isn't unique and consequently undefined. For example, an attack goal with two OR-connected leaves. Each leaf has a value for the time and a value for the money. So, if five minutes and three euros for leaf one were needed and three minutes and five euros for leaf two, the goal becomes three minutes and three euros if each value is calculated separately. But if an attack should be simulated where the attacker wants to break in as fast as he can and on the cheapest way (attack profile: *Cheapest Attack in shortest Time*), the values of the attack goal are undefined. Which values are chosen from which leaves? To solve this dilemma a rank of the properties (see Section 5.3) has to be established.

Chapter 3

Attack Tree Extensions

The Attack Tree Model offers some advantages in the work with possible threats. The hierarchical structure can simplify navigation and enable multiple experts to work on different branches of a tree in parallel. The concept of Attack Trees is easy to understand, allowing a wider range of persons to take advantage of an Attack Tree or to contribute to it. The simple visual representation of the possible threat scenarios makes it easy to get a general idea of the whole system. With this concept it is easy to expand the tree in order to react fast to new threats. Another advantage is the reuseability of trees or parts of trees. Often some different systems have similar vulnerabilities. Consequently the reuse of trees and parts, respectively, saves time and money. Nevertheless there are also some disadvantages. The creation of an complete Attack Tree is practically impossible. It always depends on the skills and knowledge of the creator. A realistic estimation of the resources at the leaf nodes is also a difficult matter. The context plays also a role during the estimation process and thereby the reuse of the estimation values is often not possible without adaption[11, 12].

Finally, the structure of the AND/OR nodes can reduce the functionality. Sometimes, there are good reasons for a node to be both AND and OR node. Also different parts of the tree can have the same origin. This leads to the fact that the tree is expanded unnecessarily.

During this internship, the Attack Tree concept by Bruce Schneier was extended. To deal with these structure and formula problems three extensions were developed. The first extension – the AND/OR Formulas – offers more possibilities to handle values in the tree. The second one – the Confluent Branches – can be used to reduce the size of the tree and can serve the clarity. The third extension – the Defense Nodes – shows a possibility to add a countermeasure to the tree.

3.1 Extension: AND/OR Formulas

Bruce Schneier only described two types of nodes - the AND nodes and the OR nodes. As illustrated in Figure 2.5 of Chapter 2 at least one sub goal has to be fulfilled if an OR node is present. In the case of an AND node all sub goals have to be fulfilled. In the way that a continuous value is present the calculation is either the *summation* of all children values (AND node) or the value becomes the *minimum* value of all children values (OR node). Thus, in the example the attack goal with the two OR connected sub goals becomes 150 for the value type *money* since the minimum value of the two children is taken. In the case of the AND connected sub goals the attack goal gets 550 for the value type *money* because the values of the sub nodes have to be added. If boolean values are present the node types correspond to the boolean operations. With the restriction to these four formulas (see Table 3.1) the functionality of an Attack Tree is limited.

operations for	**AND node**	**OR node**
non-boolean values	summation	minimum
boolean values	AND	OR

Table 3.1: Attack Tree operations by Bruce Schneier

In the extension, described in this Section, more operations to both types of nodes can be assigned. Besides the *summation* operation for AND nodes and the *minimum* operation for OR nodes a *maximum* operation, an *average* operation and a *product* operation for non-boolean values can also be used. Now all five operation can be assigned to both of the node types. Thus it is possible that the AND nodes become the minimum of all child nodes of a value type and the OR nodes become the summation of all their children nodes of a value type. For boolean values only operations which are commutative and associative can be used. Due to the fact that the order of nodes can differ, the operations have to have these properties. Thus the logical operations AND, OR, NAND, NOR and XOR are supported (see Section B.1 for the logical tables). As in the case of the non-boolean values all 5 operations to

	operations for boolean values		**operations for non-boolean values**
AND/OR nodes	AND	AND/OR nodes	Minimum
	OR		Maximum
	NAND		Summation
	NOR		Average
	XOR		Product

Table 3.2: Possible Operations on Nodes

operation for	**AND nodes**	**OR nodes**
value type 1 (e.g. money)	minimum	average
value type 2 (e.g. time)	summation	maximum
value type 3 (e.g. need for equipment)	AND	NOR

Table 3.3: Example for possible operations on different value types

both types of nodes can be assigned. Finally there are two types of nodes (AND and OR Nodes) in the tree. For each value type it is possible to assign one of the five operations to each node type (see Table 3.2 for the possible operations and Table 3.3 for an example).

Now one may ask if the differentiation between the operations in AND and OR nodes makes sense. The advantage in this extension is that for both node types operations whose results are affected by more than one value (for example: OR - summation, AND - average) are possible. Also it could be possible to add more node types. Consequently it might be better not to use AND and OR nodes any longer but node type one, node type two and even node type three etc. However, this order can cause confusion. Trees, whose formulas could be provided actually in a general manner, are obstructed by confusing formula allocations in easy interpreting. For example, Trees could be created, whose AND-nodes contain the OR-formula and the OR-nodes contain the AND-formula.

Another suggestion is that AND nodes only have the combining operations like *summation*, *average*, *product*, *AND* and *NAND* and the OR nodes have the *minimum*, *maximum*, *OR*, *NOR* and *XOR* operations (see Table 3.4). This case serves more the understanding but the combining possibilities decrease and the case of mixed operation types doesn't occur. Accordingly it wouldn't be possible to have for example values for the skill level and time in a node, where the operations are the maximum for the skill level and summation for the time (see Figure 3.1). The lack of this possibility would restrict the functionality of the Attack Tree significantly.

However, since the Attack Trees should be provided with more functionality, the

	AND combining formulas	**OR non-combining formulas**
non-boolean values	Summation Average Product	Minimum Maximum
boolean values	AND NAND	OR NOR XOR

Table 3.4: Possible combining/non-combining Operations on Nodes

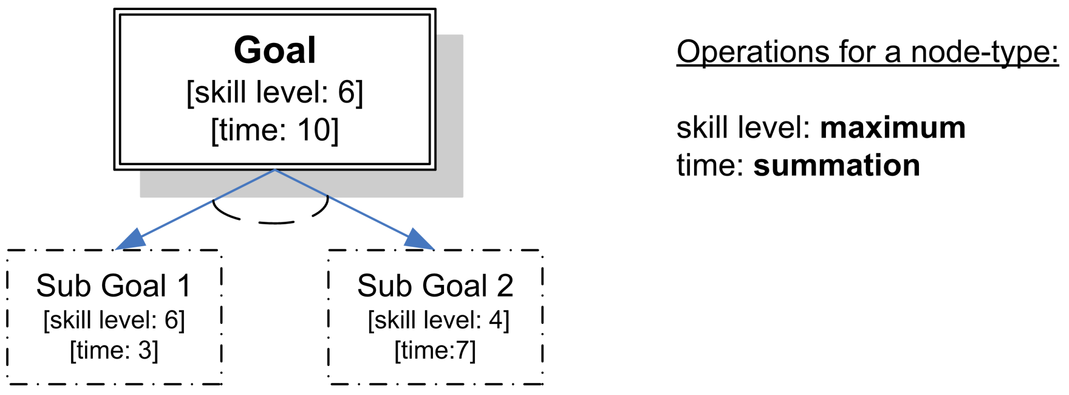


Figure 3.1: Node with mixed operation-type values

first case has been chosen. In this case – like also generally – the creator should know exactly, what makes sense. It might be wise to exclude some confusing combinations – like e.g. the boolean AND-formula in an OR-node – from the beginning.

3.2 Extension: Confluent Branches

During the construction phase it might be possible that some different parts of the tree system have the same threat origin. In this case it should be possible to work on this threat at only one position. This possibility has been implemented as a new extension: the Confluent Branches. This means that a node or leaf could have more than one parent. In that way the Attack Tree becomes a graph. This extension serves the clarity in an Attack Tree. Without this functionality two different creation teams could generate two different tree branches although it is the same threat. Even if the tree has the same structure in both branch parts the Confluent Branches functionality can save memory in the Attack Tree data and file structure.

In the running example (Figure 3.2) the door can be opened by using force and also the windows can be broken by using force. Both tree branches can be joined and only one node for this action is necessary in the tree. Of course, it is possible that more than two parts can be joined.

However, some things have to be considered. The respective tree parts, which are joined, must request also the same resources. The tree structure leads to the same action but it could be possible that the resources for the actions are different. Like if is the case with the extension before, the creator has to take a look at the sense of each combination. This is also required if the tree is expanded at a part of a tree which is the result of confluent branches.

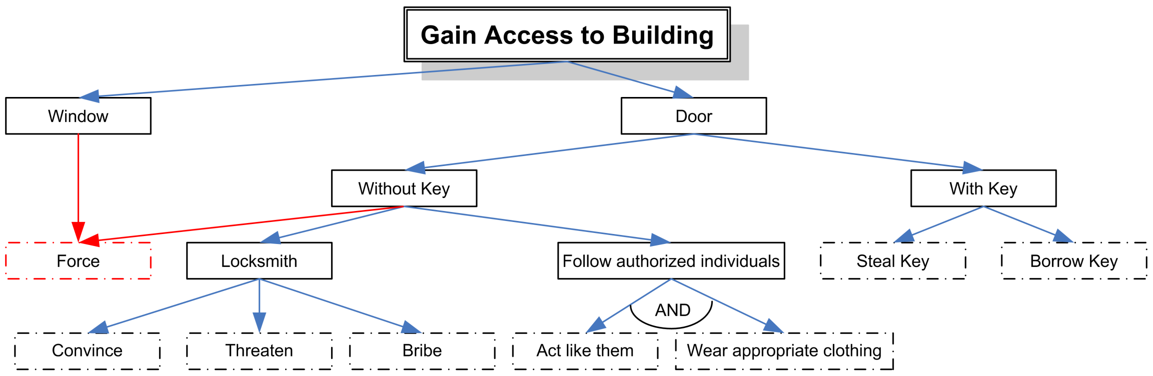


Figure 3.2: Example with confluent branch

3.3 Extension: Defense Nodes

If a profile of an attacker is present, it is possible to put oneself in the position of such an attacker and his path through the tree can be traced. For example, if the attacker has only 20.000 euros for an attack available he uses the cheapest attack against the system to maximize his proceeds (e.g. the outcome of this attack goal is only 20.000 euros worth). To be sure that the intruder can't take the cheapest path an adequate countermeasure – a Defense Node – could be added. Consequently the costs grows and this attack isn't worthwhile anymore for this attack profile.

With this new extension the influence of a countermeasure to the system can be evaluated. For the first time it is possible to work during the development process of a system with countermeasures against threats the system's developer is worried about. By adding Defense Nodes it is possible to observe changes of the security situation in the entire system. Thus, it is possible to identify the possibly next attack targets and secure the system at these positions by adding appropriate countermeasures.

In the example (see Figure 3.3) a break-in through the windows should be eliminated. If secure materials like shatterproof glass is used the window can't be broken. Finally, the break-in through the windows isn't possible anymore. Since the *Force* node isn't only connected to the window branch the defense node has also influence to the door branch of this Attack Tree.

This Defense Nodes extension is a simple version to add a countermeasure. A disadvantage of this kind of Defense Nodes is that only defenses in leaf nodes have an influence to the Attack Tree system. If new attacks against these defenses arises and are added, the functionality of the Defense Node is lost. In order to work around this the functionality could be further extended that countermeasures have a relative influence on threats. This means, that for example a non-boolean value could be increased by 10 percent as a result of a defense. Consequently, it might be possible to add a defense inside the tree rather than leaf nodes. For practical application this enhancement needs more research, thats why it is not implemented at the moment.

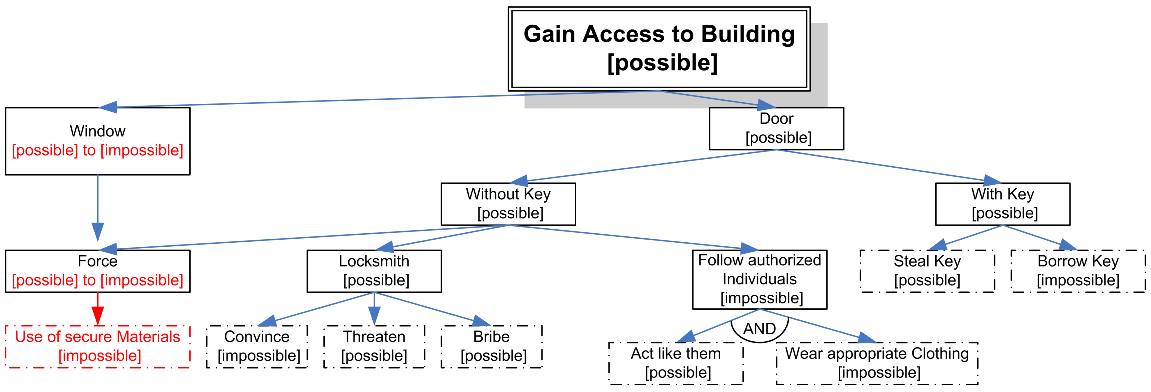

Gain Access to Building [possible]
Window [possible] to [impossible]
Door [possible]
Without Key [possible]
With Key [possible]
Force [possible] to [impossible]
Locksmith [possible]
Follow authorized Individuals [impossible]
Steal Key [possible]
Borrow Key [impossible]
Use of secure Materials [impossible]
Convince [impossible]
Threaten [possible]
Bribe [possible]
AND
Act like them [possible]
Wear appropriate Clothing [impossible]

Figure 3.3: Example with a defense node

With these extensions the functional range of Attack Trees can be enhanced. Now, it is possible to combine resources in much more versions than Bruce Schneier described in his paper. The first extension offers the user to create an individual Attack Tree which is adapted to his demands. To serve the clarity in an Attack Tree the second extension was developed. This extension can be used to reduce the complexity of an Attack Tree. Tree branches don't have to be created twice, if their content is the same. And for working with countermeasures the last extension was developed. Thus it is possible to track the changes of an countermeasure in an Attack Tree and the success/failure-rate of an countermeasure can be analysed.

The next chapter shows how these extension are implemented in a new software tool called *Support Tool for Attack Trees*. Also the functions of the tool will be described, which are necessary for an meanful work with Attack Trees.

Chapter 4

Design and Implementation

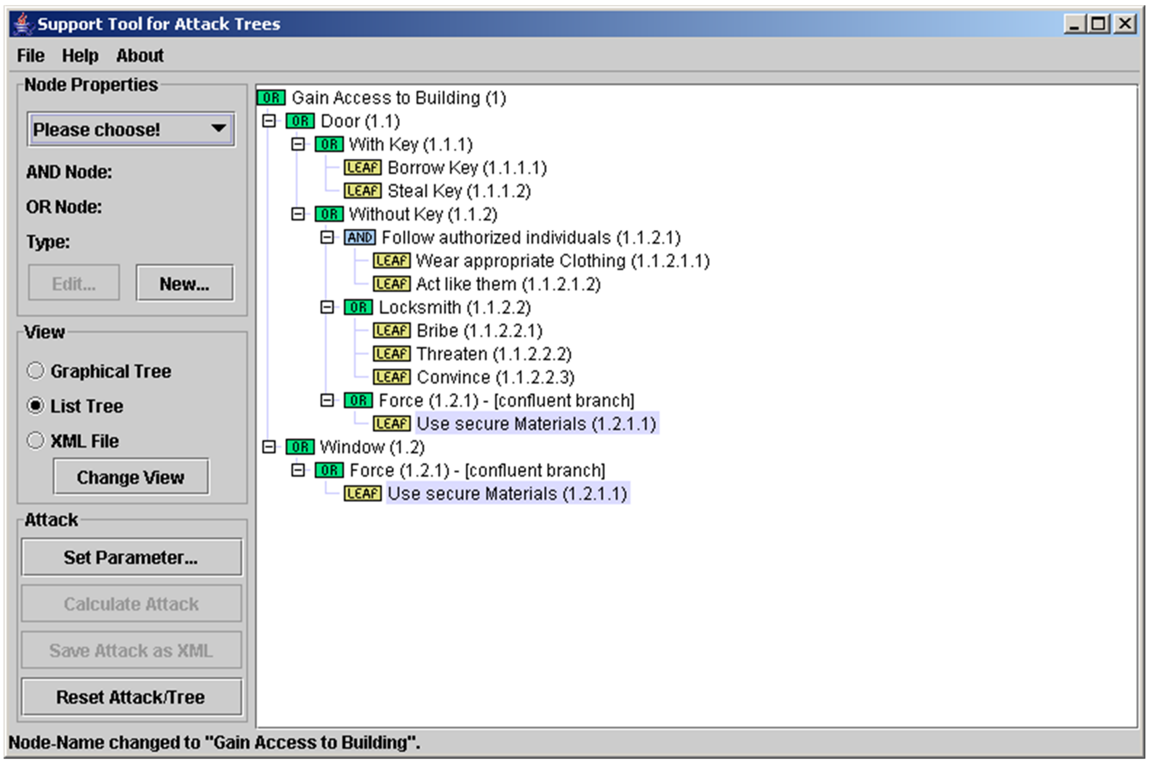
The Attack Trees and the new extensions are based on simple concepts. The shown examples can be accomplished easily by a person with paper and pencil. But if the situations become more complex, the operations become fast unmanageable. Also possibility to try different attack profiles would be reduced drastically, as soon as the expenditure becomes consciously.

With the help of a software tool, that is in the position to accomplish these operations with one mouse-click, complex situations can be simplified. So the creator does not have to keep all parameters in the eye any more during the work with Attack Trees.

Since there is no non-commercial product for Attack Trees available, a main part of this internship was to create a freeware. The decision to implement the tool in Java was easy, because there is for almost every operating system a Java Runtime available. Besides the extensions to Attack Trees described in Chapter 3 the program should also have an open file format. For any extensions at a later time and an easy understanding a XML-compatible format was selected and extended. Consequently an exchange with other programs and tools is possible.

4.1 Functionality

Because the program should have the same graphical user interface on every system the Java Swing library was used. Since a user might work with the program on different operating systems an uniform user interface and behavior is advantageous. Accordingly, there is no need for an adaption phase. This has also the advantage that different operating systems don't need different source code for the graphical user interface. Figure 4.1 shows the tool with the running example. The user interface is divided into two main parts. On the left side are three option panels. The first panel offers the set up of the value types and the according node operations. The second panel offers the opportunity to change the view of the Attack Tree. Thereby, three options are available. The different views affect the Attack Tree representation in the right part of the main frame. In order to obtain results quickly the Java JTree functionality was used to display the Attack Tree in the *List Tree View*. This JTree representation can be compared with the textual representation of an Attack Tree mentioned in Section 2.2.2. Also the JTree functionality enables the use of mouse interactions to manipulate an Attack Tree. The possibility of changing the view to XML is also given. Therefore it is possible to see how an Attack Tree is represented in the external file. Since this view isn't intended to create and modify Attack Trees it is primarily meanful for the developer. Nevertheless it is possible for others to understand the file format that is used to represent an Attack Tree. An option for displaying the Attack Tree in a graphical view is also prepared but not implemented yet. A graphical view approaches the human feeling of a tree. The comprehension of this method corresponds more the method of creating Attack Trees by pencil and paper. This method will be probably the easily most understandable method. However, the implementation timeframe was limited and for obtaining a working software tool the *List Tree View* was chosen. The last box of the left side of the tool offers the possibility to create an attack profile with its necessary resources.



View: [✓ List Tree]

```
OR Gain Access to Building (1)
⊟ OR Door (1.1)
    ⊟ OR With Key (1.1.1)
        LEAF Borrow Key (1.1.1.1)
        LEAF Steal Key (1.1.1.2)
    ⊟ OR Without Key (1.1.2)
        ⊟ AND Follow authorized individuals (1.1.2.1)
            LEAF Wear appropriate Clothing (1.1.2.1.1)
            LEAF Act like them (1.1.2.1.2)
        ⊟ OR Locksmith (1.1.2.2)
            LEAF Bribe (1.1.2.2.1)
            LEAF Threaten (1.1.2.2.2)
            LEAF Convince (1.1.2.2.3)
        ⊟ OR Force (1.2.1) - [confluent branch]
            LEAF Use secure Materials (1.2.1.1)
⊟ OR Window (1.2)
    ⊟ OR Force (1.2.1) - [confluent branch]
        LEAF Use secure Materials (1.2.1.1)
```

Node-Name changed to "Gain Access to Building".

Figure 4.1: The Program with the continuous Example

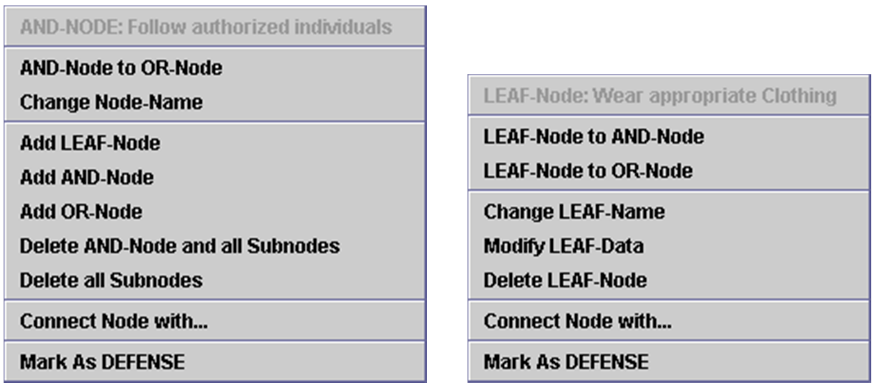


Figure 4.2: Pop up Menus on a Leaf Node and an inner Node

4.1.1 Creating and Modifying Trees

Due to the lack of time and the easy adaption of the textual representation using the Java JTree functionality the List Tree View was chosen for building and changing the appearance of an Attack Tree. At each position of the tree changes are possible to create individual Attack Trees. This is also necessary to modify parts of the Attack Tree in later states of the development process. Thus, the edit functions were realized with mouse accessible pop up menus. Because the edit functions differ in LEAF nodes and AND/OR nodes two kinds of pop up menus were created (see Figure 4.2). With this differentiation input errors of the users are eliminated. Since the pop up menus depend on the current underlying node only this node is affected by changes of the user. Outside of the tree listing no pop up menus are available.

If a tree is developed, the creator has to have always the ability to change the structure of the tree. If new threats arise fast changes can be made and the tree is adapted to the new circumstances. In order to comply with this some standard editing functions were implemented. One is, that the creator is always able to change the type of nodes. It is possible that an AND node can be changed to an OR node or vice versa. Since new nodes can't be added to LEAF nodes, a LEAF node can be changed to an AND or OR node. Now, expansions at these positions with AND, OR and LEAF nodes in the tree are possible. Consequently it is possible to change an AND/OR node to a LEAF node to finalize a branch of an Attack Tree. Of course, the renaming of node names is an implemented function too.

To implement the Confluent Branches extension mentioned in Section 3.2 it has to be differentiated between LEAF-nodes and AND/OR nodes. AND/OR nodes can be connected to every other node or leaf in the tree – except the nodes in the same layer or children and parents of the node (since a connection is already present). A special case is the connection between a node and a leaf in a lower layer[1]. Usually the direction of a connection is established from a lower to a higher layer but in

[1]Lower levels are closer to the top of the tree and higher levels are further accordingly

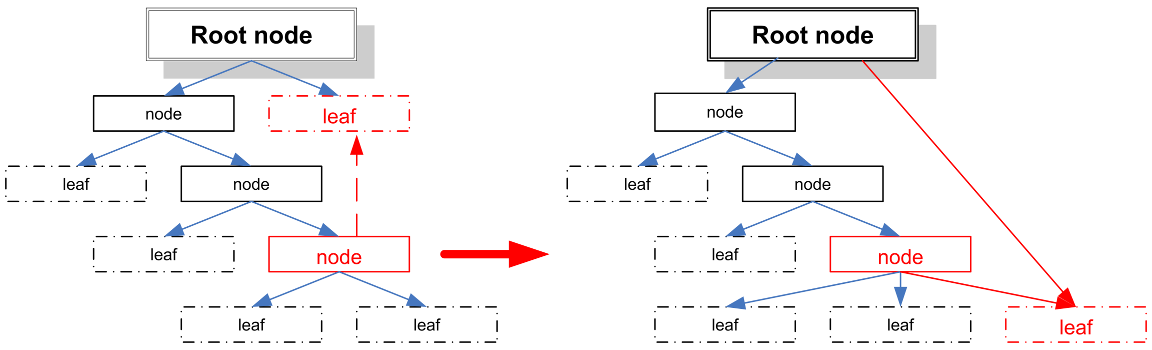


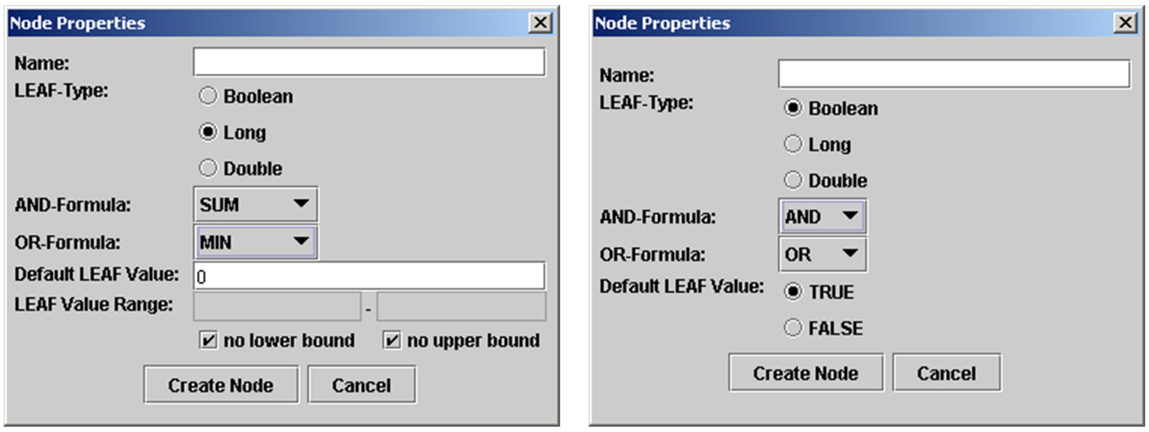
Figure 4.3: Connection from a Node to a Leaf of a lower Layer

this case the lower leaf node is pulled down in the tree and the leaf becomes a child of the node (see Figure 4.3). Since it is not possible to display graphs in a JTree the confluent branches are split and displayed at each part of the tree. This is accentuated in the JTree through the notice *[confluent branch]* after the name of the leaf and node respectively. If a graphical view is implemented later this problem is solved. Internally the program handles the confluent branches as one unit. If the creator changes something at one position of a confluent branch in the *List Tree View* the changes are automatically realized at the other parts. Thus, it doesn't matter at which position of the confluent branch changes are made.

In opposition to the AND/OR nodes, connections from a LEAF node are only possible to nodes of a lower layer. Of course except their parents, since these connections are already present. Since LEAF nodes are the last elements of a branch they cannot be connected to nodes or leaves at a higher level. If a connection to a node at a higher level should be established the order has to be reversed and the case mentioned in the paragraph above arises (Figure 4.3).

To complete the construction functionality the pop up menus support the deletion of parts of the tree. Since modifying a tree is necessary in the development phase this standard editing function was implemented. Besides the removal of a single LEAF node it is possible to delete whole branches of the tree. For this reason it has to be differentiated between the deletion functions. On the one hand only the sub nodes are deleted, and on the other hand all sub nodes and additionally the node which is the starting point of the removal are deleted. In both cases the deletion of the sub nodes and leaves is done recursively. If a sub node is part of a confluent branch the program only deletes the connection between the nodes. Thus, the other parts of the tree still can use the old joint nodes and leaves. This function is consequently suitable for the disintegration of confluent branches. The advantage of the second deletion function is the saving of two additional steps for the deletion of the starting node. Without this function the sub nodes and leaves have to be deleted first and than the starting node has to be converted into a leaf which has to be deleted afterwards.

To realize the Defense Node extension the program was enhanced to highlight


Node Properties

Name:

LEAF-Type: Boolean [✓ Long] Double

AND-Formula: SUM

OR-Formula: MIN

Default LEAF Value: 0

LEAF Value Range: -

[✓ no lower bound] [✓ no upper bound]

Node Properties

Name:

LEAF-Type: [✓ Boolean] Long Double

AND-Formula: AND

OR-Formula: OR

Default LEAF Value: [✓ TRUE] FALSE

Figure 4.4: Dialogs for Adding Value Types

nodes and leaves in a different color in the Java JTree. This accentuation was done to help the users to recognize fast the different kind of nodes. Thereby, these nodes and leaves are internally marked as defenses. To create a *Defense LEAF* a leaf is created and provided with resources, which correspond to a defense. Afterwards, the leaf is marked as a defense. The creator has to examine again, which values and markings make sense.

During the creation process of an Attack Tree it may happen that a node is highlighted in red. This behavior indicates that this branch of the tree has no leaf node and thus there is no calculation possible in this part of the tree.

To set up resources in the LEAF nodes their types have to be defined before. How this is managed will be demonstrated in the next section.

4.1.2 Add Values to the Tree

To modify values in the leaves of the Attack Tree the type of the values has to be defined first. For example, the cost for the actions, the time which is needed for an action, whether one needs special equipment for an action, the skill level which is needed can be added and so on. The number of possible properties is not limited. The more properties are defined the better an attack profile can be adjusted to an attacker later.

The value type can adopt boolean and continuous values (see Figure 4.4). As AND and OR formulas the operations mentioned in Section 3.1 are available. Furthermore a default value for each type has to be assigned. This value is automatically set for the leaf nodes in the construction phase until it is changed by the user. Thus, values are always present in the tree and the automatic calculation can take place without restrictions. Without this default value the automatic calculation process returns errors, because input data for the calculation is missing. For an flexible behavior new value types can be added at a later time without affection of the calculation process. However, it is unfavorably that the creator does not know at a later time, which leaves he already provided with new, updated values. A possible solution shows Section 6.2.

If a non-boolean value type is chosen two alternatives are available. The values can adopt the LONG data type or the DOUBLE data type like they defined in Java. Since INTEGER is part of the LONG data type and FLOAT is part of the DOUBLE data type only this two options are available. These data types were selected, in order to ensure a certain accuracy of the values. In the non-boolean cases a lower and upper bound can also be set for values. Thus, the possible input values can be restricted. For example, the costs for the actions in a leaf can be only set between 100 and 200 euros. Both bounds are optional properties. Only one bound is also allowed, accordingly.

4.1.3 Automatic Calculation in an Attack Tree

At the point where a value type is set, the calculation in the tree is done automatically. Thereby, each value type is regarded independently. If the mouse is moved over the AND/OR nodes in the JTree the calculated values can be seen in a tool tip. The values in the LEAF nodes can also be reviewed in a tool tip. If all values would be shown directly in the tree listing, the overview is lost fast and the tree would be enlarged unnecessarily.

Now it is possible to know what resources are necessary to achieve the goal in the root node or the subgoals in the AND/OR nodes. All branches in the tree have to end with at least one leaf node for an successful calculation in the entire Attack Tree. If more than one value type for the nodes is present the values in the AND/OR nodes can originate from different sub nodes or leaves. The program chooses the best value depending upon the AND/OR Formula for each value type (Money, Time, Equipment,...). In that way it is possible, that, for example, for an AND/OR node the money is taken from another child than for the time.

This automatic calculation offers an general overview of the necessary resources in the nodes. Since values can originate from different nodes the values shown in the tool tip are only suited if the values are regarded individually.

4.1.4 Creating Attacks

If the profile of an attacker is known or a profile should be tested, a special attack from the intruders point of view can be created. In a mask (see Figure 4.5) the properties of the attacker can be specified, for example that the attacker has special equipment and 1000 euros available (the attack should cost below 1000 euros). The dialog contains a panel for each value type, where the user can specify the attributes of an attack profile. Besides *true* or *false* for boolean value types there is the possibility to define non-boolean values more detailed. For this purpose an upper and a lower

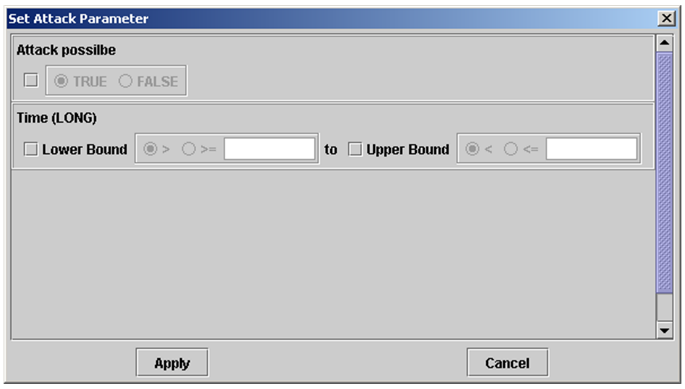


Figure 4.5: Mask to set the Profile of an Attacker

limit can be specified. Additionally, it can be specified whether the borders are also included into the profile or not. That allows to adapt profiles in every detail to the respective needs.

To set a profile at least one option has to be activated. All matching nodes are highlighted in the JTree listing in a light green color. Now the program uses all light green highlighted nodes and leaves to check if at least one highlighted path from the root node to a leaf node is possible. If this applies an attack is possible for this special profile of an attacker. If at least one path is available the functionality to recompute the tree only with the matching paths exist. Thus, this offers the Attack Tree creator to receive a view which is only directed toward the specified attack. In this case all nodes which are not a part of an attack path are cut off. Afterwards the nodes in the tree are recalculated and displayed in a new JTree. In this new tree nodes and leaves can also be added. Because this tree is meant to show special attacks matching to a profile all new added leaves and nodes are automatically set as defense nodes and defense leaves respectively. According to this only leaf nodes can be changed and new defenses can be added at this position of the tree. This view also provides the functionality to save this part of the tree. If the complete tree should be displayed again, both trees are combined. All new defense nodes and leaves are part of the original tree, too.

With this functionality the user has the opportunity to check every possible attack profile. Thereby, no limitations are present for the creation of an attack profile. Thus, the user can be prepared for new attackers within shortest time. Subsequently, countermeasures can be seized immediately if necessary.

The next chapter analyses special cases of attack profiles and the calculation process. Potential effects and solutions will be shown.

4.2 File Format: XML

Since there should be the possibility to use the Attack Trees in other programs and tools, an open and easy reviewable representation is advantageous. Today the Extensible Markup Language (XML) [13] is used for most different purposes in many different application areas. "The XML is a W3C[2]-recommended general-purpose markup language for creating special-purpose markup languages. Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet."[26] For XML-data exists the possibility to create a Document Type Definition (DTD) for a validation process. With the DTD an own definition for a file format for Attack Trees is feasible. This has the advantage that different programs can use the same validated file to work with Attack Trees. Since XML is very popular there are many libraries for different programming languages available.

4.2.1 The GraphML File Format

Because Attack Trees were extended with the Confluent Branches functionality during this internship the tree obtains properties of a graph. Consequently a XML based graph file format was required and it turned out that GraphML has the best requirements for the program. GraphML is described as follows on the GraphML homepage: "GraphML is a comprehensive and easy-to-use file format for graphs. It consists of a language core to describe the structural properties of a graph and a flexible extension mechanism to add application-specific data."[14]

The convenience of this XML file format is the independent declaration of nodes and edges. This allows extensions of the graph in arbitrary places of the file format. For this purpose each node and each edge are indicated individually in the graph. To avoid mistakes while building up a graph, unique identifier can be assigned as attributes to the nodes and edges. Contrary to the other elements of the graph definition the unique identifiers are not optional for the node elements. This is necessary, since an edge must be determined clearly by its two end points, which correspond to the unique identifiers of the nodes in the same graph. A graph could contain directed and undirected edges. To avoid the declaration of each edge, a global default value has to be set. If several edges should differ from default value, they can be assigned with an addional attribute for the edge direction.

To define properties of the graph *key* elements have to be declaired inside the XML structure. Like the other elements the properties definition has also an unique identifier. Furthermore an unique name of the property and its type have to be set. Thereby value types like *boolean*, *int*, *long*, *float*, *double* or *string* can be adopted. They are defined like the data types in Java. Finally, a domain can be specified. Accordingly it can be indicated whether the properties refer to the nodes, edges or

[2]World Wide Web Consortium

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <key id="d0" for="node" attr.name="weight" attr.type="double"/>
  <key id="d1" for="edge" attr.name="color" attr.type="string">
      <default>blue</default>
  </key>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="d0">2.0</data>
    </node>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <edge id="e0" source="n0" target="n1">
      <data key="d1">yellow</data>
    </edge>
    <edge id="e1" directed="true" source="n0" target="n2"/>
    <edge id="e2" source="n1" target="n3"/>
    <edge id="e3" source="n2" target="n3"/>
  </graph>
</graphml>
```

Figure 4.6: GraphML example with all properties

the graph. Like the edges the properties can also be set up with default values. Besides them the default values for the properties are optional. If an edge, node or graph element should be specified with an property an element has to be nested inside the structure element. Thereby a reference to the property identifier must be established, too.

Figure 4.6 shows all properties together in one simple example. As there can be seen the example consists of four nodes which are connected – up to one – with undirected edges. Also two properties are defined. One for the nodes and one for the edges. Thus, for the nodes a weight is described which accepts data values of the type *double*. The property for the edges describe the color. Besides the data type *string* the property has a default value which is set to *blue*. Inside the graph structure definition one edge exists which varies from the default value. Accordingly edge *e0* is set to *yellow*. Since no default value is available for the nodes only node *n0* has a value. Figure 4.7 shows a schematic representation of this XML example.

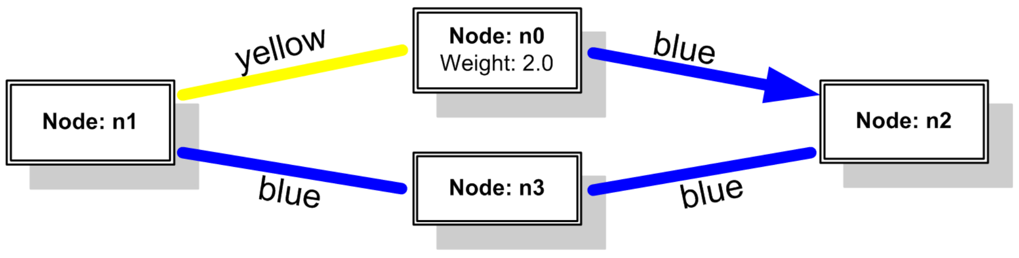


Figure 4.7: Schematic Representation of the XML Example

4.2.2 Extensions and Definitions

The GraphML format provides a good basis for the XML representation of Attack Trees. To integrate the new extensions mentioned in Chapter 3 the GraphML format has to be extended to support these new improvements. Also some definitions for the input data have to be made and some features of the program have to be supported.

Every Attack Tree representation has at least three property definitions. They are necessary for the structure of an Attack Tree. The first one always describes the type of a node. Thus AND nodes, OR nodes and LEAF nodes can be differentiated. The second property is the definition for the node name. Through this the names for the nodes and leaves are assignable. The last must-have property describes the kind of nodes and leaves. It defines if a node is an attack node or a defense node. By default the nodes are defined as attack nodes.

If new values for the actions are defined – like the costs, time or the need of equipment – a new property is defined and it is extended by 5 sub elements. Besides the sub element *default*, the elements *and*, *or*, *lowerbound* and *upperbound* are necessary. The values of the *and* and *or* sub elements describe the formulas in the AND and OR nodes. In opposition to the original GraphML definition, the data types are restricted and can only adopt three values: boolean, long and double. This was adapted to the program-specific peculiarities mentioned in Section 4.1.2. If long or double is chosen the XML-file also provides the possibility for bordering the input values. Thus only values between the lowerbound and upperbound are possible. If one or both bounds are not set, the value NULL is written into the XML-file. Note that if borders are present, the default value has to be in the range of the bounds. In the boolean case both bounds are set to NULL. Also as default values only TRUE or FALSE are accepted. Of course only the logical operations mentioned in section 3.1 are possible.

Since the first two properties have always no default values a node element has at least two data sub elements. The one with the value type and the one with the node name. If leaves with values which differ from the default values are present in the properties the node elements have further data sub elements with the corresponding data values. To prevent errors, all elements are provided with references to the unique identifiers.

Appendix A.1 shows a complete XML example of two AND connected leaves, which contain one boolean and one non-boolean value. So all possibilities can be seen.

Chapter 5

Test of the Tool

To evaluate the program some general and special tests were done. Thereby the tool was examined in general. The realisation of the new extensions mentioned in Chapter 3 and the calculation process have been analysed. Thus arisen problems and limitations and their possible solutions were pointed out.

5.1 General

First the general handling of the program was analysed. This contains the starting process of the program and the possibilities to load and save Attack Trees. In addition the creation process of an Attack Tree was observed. These investigations were made in order to check the usability of the software tool. For this purpose all program-specific functionalities were tested, whereby this did not only happened in an arranged order.

Since the program needs different libraries which are not part of the Java Runtime, the program is delivered in a Java Archive (JAR-file). Into this Java Archive all necessary libraries were included. Consequently it is possible that every user – no matter if beginner, advanced or professional – can access the program with only one command. The only requirement is a proper installed Java Runtime in Version 1.4 or above.

Immediately after a Attack Tree document is created, the functionality to save the document is offered. This can be done in every phase of the work with the tool. If the program or a document is closed or a new file should be created, the program offers to save the old document. Consequently it cannot happen that a document is closed by mistake, without the possibility to save the document first.

If a new document is created or an old file is opened the user can change the structure of the tree in the *List Tree View*. Thereby changes are converted intermediately. Also theses changes are transferred intermediately in the XML code. In that way it is possible for the user to pursue at each time the appropriate XML structure.

In addition only the view must be changed.

To change the structure of the tree the two pop up menus are available. They offer the functionality described in Section 4.1 and they adapt to the existing conditions. For the two types of nodes, their functionality differs. Only the allowed functions are visible. Thus the possibility of accomplishing invalid actions does not exist. Consequently only regular Attack Tree could be created and the user has to be unconcerned about it.

5.2 Extensions

As structural extension the functionality of Confluent Branches were implemented (see Section 3.2). The user has the ability to connect different nodes. With the notice "Confluent Branch" and the unique numbering in the *List Tree View* the Confluent Branches are clearly identified. Also the user has the functionality to annihilate the connection by using the standard deletion functions in the corresponding branches of the tree. In order to prevent errors during the creation of connections only meaningful and/or regular connection possibilities are offered to the user. Thus the user possesses control of the entire creation process and all input errors of the user are prevented.

To combine node values with different calculation procedures the program supports much more formulas as Bruce Schneier described in his paper. Therewith at any time the formulas can be changed in the program. This ability is available at any point during the work with the Attack Tree tool. To create more exact predications about the vulnerabilities or more exact attack profiles, the program offers the possibility to add unlimited resources and the corresponding formulas to the nodes. Like the formulas, new resources can be added anytime. Also the deletion of resources is available at any time during the work with the tool. With this extension the user also possesses complete control about this functionality of the program.

The Defense Nodes extension mentioned in Section 3.3 was realized by colorizing the corresponding nodes in the *List Tree View* of the tool. Thus the user always recognize the special type of a node. In order that Defense Nodes are effective, the user has to have always the knowledge of what is necessary for defenses (what values are at least necessary for a defense). Without this knowledge it might be possible that some Defense Nodes are useless in the tree and they are effectless for prevention of an attack accordingly.

5.3 Calculation Process

As mentioned in Section 4.1.4 the highlighting is done for nodes and leaves where the values are calculated separately. This means that nodes could have values from different sub nodes and sub leaves respectively. If a profile of an attacker is created, all values of the matching nodes of the profile should originate from only one node if

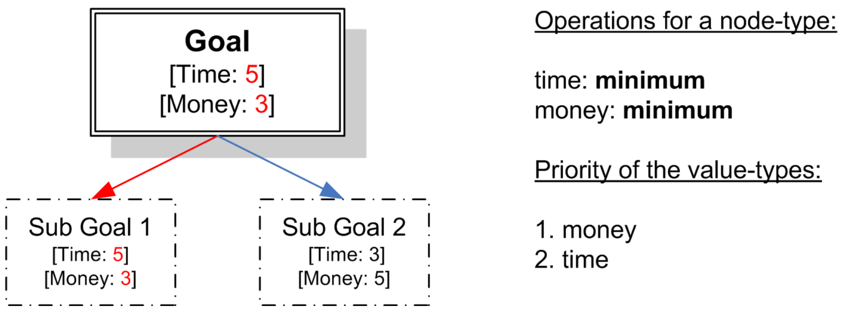


Figure 5.1: Solution of the undefined case

operations are present where only one value determines the result. To find matching nodes and leaves, it has to be differed between operations where all sub values have influence and operations where only one value has an influence.

In the case that only one value determines the result of an operation (minimum, maximum, OR, NOR, XOR) the undefined case mentioned in Section 2.2.2 could occur. The described example in that section has two OR-connected leaves where the minimum operation is set for both values. Leaf one has the value five for the time and value three for the money and leaf two has the value three for the time and value five for the money. With the attack profile *Cheapest Attack in shortest Time* it is not clear which leaf values determine the value of the root node. To solve this case an ordering in the profile has to be established in the tool. This has to be done always if more than one non-boolean value is present. Accordingly the money might be set to a higher priority as the time in the example and the root node now receives the values of leaf one (money: 3, time: 5 – see Figure 5.1). The ordering function also eventuates if more than one profile matching sub element is available. If a profile like *Money below 6 and Time below 6* is present both leaves are matching to the profile in the example. Due the fact that the money has a higher priority, leaf one is taken for the values in the root node. Finally a node does not receive values from different sub elements anymore.

In the case that more than one value determine the result of an operation (summation, average, product, AND, NAND) all sub nodes of a node have to match to the attackers profile. If that is the case the node can also be tested if it matches to the profile. Otherwise, if at least one node does not match, the whole branch of the tree has no influence on the attack anymore. This problem is evident in the current state of the tool. The highlighting process that marks matching nodes for an attack profile, works with no respect to the formulas in the nodes. This leads to the problem that the tool could find a path from a leaf node to the root even though a formula is present where not all sub nodes match. In Figure 5.2 the problem is demonstrated. The value of the two AND-connected sub nodes is calculated by the summation formula. With the attack profile *Value One greater than 5* the program highlights sub goal 1 and the root node, since the highlighting process is done with

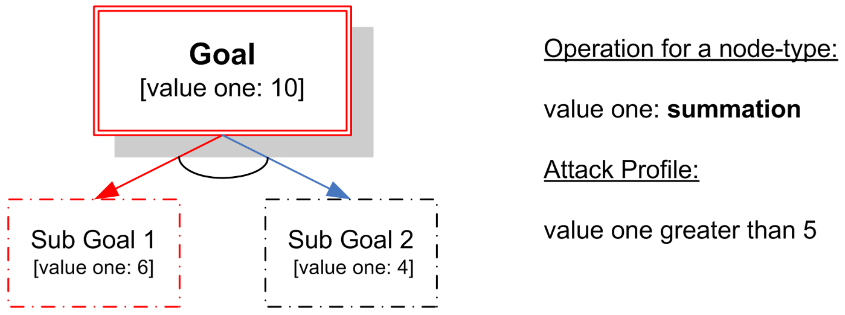


Figure 5.2: Wrong highlighting of the tool

no respect to the formulas. But in this case the highlighting of the goal node mustn't eventuate. This has to be revised.

The following remarks refer to the example mentioned in Section 3.1. Thereby values for the skill level and time are present in a node, and the operations are the maximum for the skill level and summation for the time (see figure 5.3).

In this example the creator of the tree should know which operations for the non-combining value (skill level) make sense. To achieve the root goal, sub goal one and sub goal two have to be fulfilled. The time which is needed to fulfill the goal is the summation of both children (it becomes 10 minutes) and the skill level takes the highest value of the children (skill level 6). It would not be useful, if the skill level is connected via a minimum operation. If only one sub goal needs equipment it would be illogical too, if there is no need of any equipment to fulfill the goal.

If a minimum of one combining operation is present in a node, it has also influence on attack profiles which depend on the values with non-combining operations. Thus, the profile *Minimum Skill Level* highlights sub node one and the root node in the example, since at least a skill level of six is needed to fulfill the goal.

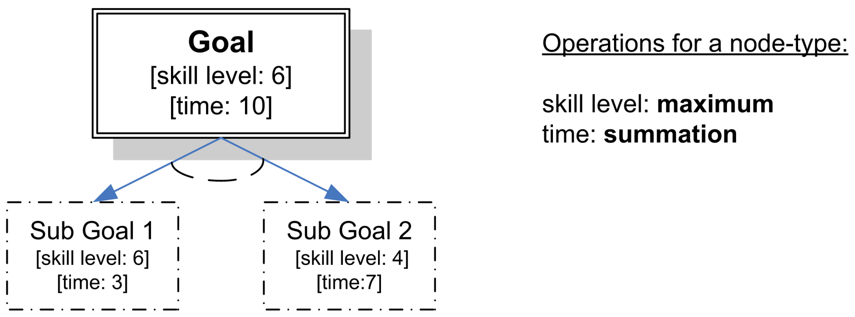


Figure 5.3: Node with mixed operation-type values

Chapter 6

Conclusion and Further Development

6.1 Conclusion

This internship was intended to get a deeper look into the field of Attack Trees. Today, only one software tool for the work with Attack Trees is available. Since the software tool is a commercial one, it is not suited for academic research. It is not possible to change parts of the software or to extend the program with further functionality. For this reason a new software tool was developed and implemented which is open source and easily to extend.

Starting from the work of Attack Trees by Bruce Schneider, the internship was intended to examine new possible extensions to Attack Trees. For that purpose three extensions were analysed and implemented into the software tool. The first one – the AND/OR Formulas – offers the possibility to connect nodes with additional different calculation formulas as Bruce Schneider described in his paper about Attack Trees [1]. The Confluent Branches extension enhances the Attack Trees in a structural way. Equal attacks, which are used in different branches of an Attack Tree, can be connected. Consequently, changes at a node influence the connected nodes too, what simplifies the working process. Finally, an extension – the Defense Nodes – was investigated to add countermeasures directly in the Attack Tree.

To work with Attack Trees the software tool needed standard editing functions. For this reason functions for adding, editing and deleting nodes and leaves were implemented. Attack Trees can be simply created and modified. For a later reuse of the Attack Trees a XML-based file format was developed and implemented. A XML-based file format was preferred, because it is easy to understand and later extensions are simply manageable. With the implementation of the three new developed extensions the goals pointed out at the beginning of this internship were achieved.

To verify the behavior of the software tool some test were made. It turned out that the new extensions produce some ambiguous constellations. Some of these cases weren't considered at the beginning of the internship. Consequently, some problems arised with the calculation of attacker simulations. This is not unusual during the development of a new software. However, the causes of these problems were found fast. Due to the lack of time during the implementation phase of the internship these problems were not fixed. But this document offers a good basis for solving these restrictions. Section 6.2 explains how the tool can be enhanced with further functionality and how it can be improved.

Nevertheless, a software tool offers some essential advantages in relation to the pencil and paper method. Creating and editing of Attack Trees can be accomplished substantially faster. If a big Attack Tree of a system is examined a software tool is needed to keep the overview. The automatic calculation process of different resources makes the effects caused by changes retraceable.

With Attack Trees the possibility exists to accelerate the process of securing a system. An Attack Tree software tool supports the creation of complex trees. During the creation process the user learns much about a system and one can find out possible vulnerabilities. However, the creator has to know what is going on in the system and how the system works. This is the basis for the creation of useful Attack Trees. With the potential to simulate an attack, it is possible to react fast and to add appropriate countermeasures. Also attack paths can be found which are not the obvious ones. For example, to prevent an attacker of reading a PGP encrypted message of a computer system it could be more effective to be sure that no key logger is installed than to increase the key length of the PGP algorithm [1]. A software tool can give support in finding these possible attacks and paths, respectively.

Finally, this internship shows that Attack Trees offer a large potential. As on the basis of the extensions can be seen, still more opportunities are behind the Attack Tree method. However, as the AND/OR Formulas extension shows, more functionality could implicate larger problems. The initially very easily understandable and manageable Attack Trees become more complex by the extensions. Accordingly, the user must always know whats behind the extensions and how to deal with them.

6.2 Further Development

Developing a software tool is always a process. During the work some new ideas and possible changes of parts of the program can arise. This was the case in the programming phase of this internship. Due to the lack of time they were not be implemented. Following some general ideas for a further development are listed.

- Revise of the calculation process to solve the experienced limitations mentioned in Chapter 5. Before new enhancements could be added the problems found during the test period of the internship have to be solved. This is the first and most important step for the further development of the software tool.

- As described in Section 4.1.2 it is not possible to differentiate leaves which are already updated with new values and leaves with the default value at first sight. In order to call attention to this, the tool tips could be customized. For example, a note can be added behind the value, if the default value is taken.

- Due to the lack of time there is no validation process of the input data in the input dialogs. To prevent errors by entering false values this should be one of the next steps within the development process of the software tool. Accordingly, the borders (upper and lower bound) and data types must be considered.

- The attack profile mask offers an opportunity to enhance the software tool.
  - First, a rank has to be established within the attack profile criterias. This is necessary to solve the undefined case mentioned in Section 2.2.2 and Section 5.3.
  - If a lower and/or upper bound is set for a non-boolean value type these bound could be set as default value in the attack profile mask.
  - Besides constraining a value type in the attack profile with a lower bound and/or upper bound it might be convenient to use the minimum or the maximum as attack criteria. Because the options of a lower bound and upper bound can't simulate a minimum or maximum attack path.
  - Finally, a consideration might be worth to connect different attack criteria not only via AND. For example, it could be possible to connect two criterias as alternatives. If criteria 1 does not apply then use critera 2.

- Since the Attack Trees are saved in XML a validation of these files is possible. For example, with an adapted DTD file (Document Type Definition) the software tool could check a file before it is loaded into the program. This is a further step to prevent input errors.

- In the actual version it is not possible to move parts of the tree directly to another branch. This can only be done by misusing the Confluent Branches extension. A better and preferred solution is a drag and drop functionality in the JTree view.

- As mentioned in Section 3.3 a different implementation of the Defense Nodes extension might be possible. They could have relative influence to a node (e.g. increase the value of a node by 40). If new attacks are added to these new Defense Nodes the influence of these defenses is present anyway.

- As mentioned in Section 3.1 it might be possible to add more than two node types (more than AND nodes and OR nodes). Since the formulas can be used for both node types a third or fourth formula combination might be useful. For this purpose the node types have to be named in a different manner and additional nodes types should be added dynamically.

Besides these corrections and enhancements the software tool could be graded up with additional functionality.

- For each leaf the path(s) to the root node could be displayed. For this reason all nodes that are no longer required, can be cut off and the values can be re-calculated. To consider are nodes which have values with combined operations like AND, SUM, etc.
- After an attack profile is established and the attack is done a summarisation could be given. The amount of possible attacks could be counted just like the amount of the necessary steps for each attack path.
- An import function of Attack Trees could be implemented. The software tool should be able to add the imported tree at any position in the Attack Tree. However, the program has to check the value types and nodes types of both trees for a proper import.
- Finally, the implementation of the *Graphical View* should be realized. This view enables a better overview of an Attack Tree, because it is easier to conceive the structure of an Attack Tree in a graphical view. Of course, all functions of the *List Tree View* need to be implemented.
- The *Graphical View* offers to extend the software tool with further export functions. For example, the tree could be saved in a graphical file format like JPG or SVG.

If the corrections were made, the Attack Tree software tool can be used for practical application. Until then, it should be accounted as a prototype. With the information of this paper the restrictions, which arised during the development of the software tool, could be handled. Afterwards the software tool can be equipped with additional functionality, which will improve the work with the tool. Since there were some inquiries concerning the software from different educational institutions, one can expect an extended version of it in the near future.

Bibliography

[1] Bruce Schneier, Attack Trees, Dr Dobb's Journal, December 1999
http://www.schneier.com/paper-attacktrees-ddj-ft.html

[2] CUUG Meeting; Tuesday, February 22, 2000;
http://www.cuug.ab.ca/CUUGsite99mar/past-meetings/meetings.99-00.html

[3] Philip E. Varner, A Security Analysis of VoteHere, May 11, 2001
http://www.cs.virginia.edu/~pev5b/writing/academic/thesis/

[4] Fault Tree Handbook, U.S. Nuclear Regulatory Commission, January 1981
http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf

[5] ARES Coporation, Risk Techniques, Fault Trees
http://www.arescorporation.com/ares/rm/rm_services_riskt_faulttrees.html

[6] Canadian Handbook on Health Impact Assessment, Chapter E: Methodologies for technological risk Assessments (continued)
http://www.hc-sc.gc.ca/hecs-sesc/ehas/publications/canadian_handbook/volume3/chapter_e_part3.htm

[7] IEE – Health and Safety Briefing 26b – Quantified Risk Assessment Techniques – Part 2, Event Tree Analysis
http://www.iee.org/Policy/Areas/Health/hsb26b.pdf

[8] Oley Scheyner; Scenario Graphs and Attack Graphs; PhD thesis; Carnegie Mellon University; April 14, 2004
http://www.milena.org/thesis/sg-ag.pdf

[9] Ramkumar Chinchani, Anusha Iyer, Hung Ngo, Shambhu Upadhyaya; A Target-Centric Formal Model For Insider Threat and More; University at Buffalo; October 12, 2004
http://www.cse.buffalo.edu/tech-reports/2004-16.pdf

[10] Saket Kaushik; Network Vulnerability Analysis Tool
http://www.isse.gmu.edu/~skaushik/nva/

[11] Jan Steffan, Markus Schumacher; Collaborative Attack Modeling; Darmstadt University of Technology; 2002
http://www.ito.tu-darmstadt.de/publs/pdf/sac2002.pdf

[12] Prof. Dr. Felix Gärtner; Verlässliche Verteilte Systeme 1, Angewandte IT-Robustheit und IT-Sicherheit, Vorlesung im Wintersemester 2004/2005
http://www-i4.informatik.rwth-aachen.de/lufg/teaching/ws2004/dds/08security-metrics_23.11.2004.pdf

[13] Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation 04 February 2004
http://www.w3c.org/TR/2004/REC-xml-20040204/

[14] The GraphML File Format
http://graphml.graphdrawing.org/

[15] Understanding Risk Through Attack Tree Analysis; Amenaza Technologies Limited; 2003
http://www.amenaza.com

[16] Virtuelles Software Engineering Kompetenzzentrum; Risikoanalyse
http://www.software-kompetenz.de

[17] Failure Modes and Effects Analysis (FMEA); Semicon FarEast; 2004
http://www.semiconfareast.com/fmea.htm

[18] Failure Mode and Effect Analysis; Wikipedia; 2005
http://en.wikipedia.org/wiki/FMEA

[19] Martin Fränzle; Eingebettete Systeme I – Scriptum zur Vorlesung im Wintersemester 2001/2002; Carl von Ossietzky Universit¨at Oldenburg
http://ca.informatik.uni-oldenburg.de/~fraenzle/ES-I-WS0102/skript-ES-I.pdf

[20] Fault Tree Analysis; Texas Workers' Compensation Commision, Workers Health & Safety Division, Safefy Education & Training Programs
http://www.twcc.state.tx.us/information/videoresources/stp_fault_tree.pdf

[21] Using Fault Trees to Identify Potential Faults in Critical Systems; Relex Software Corporation; 2005
http://www.relexsoftware.com/resources/art/art_fta.asp

[22] Secure*IT*ree – Attack Tree-based modeling software; Amenaza Technologies Limited; 2003
http://www.amenaza.com

[23] Denial-of-service attack; Wikipedia; 2005
http://en.wikipedia.org/wiki/Ddos

[24] RISK MANAGEMENT: HAZARD ANALYSIS AND FMEA; mdi Consultants INSIGHT REPORT No. 11 Vol. 4; 2001
http://www.mdiconsultants.com/Section_NI/Insights/insight_V4_is11.htm

[25] Process Analysis Tools: Failure Modes and Effects Analysis (FMEA); American Society For Quality
http://www.asq.org/learn-about-quality/process-analysis-tools/overview/fmea.html

[26] XML; Wikipedia; 2005
http://en.wikipedia.org/wiki/XML

Appendix A

Examples

A.1 XML Example

Example referring from section Section 4.2.2:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<graphml>
   <key id="0" for="node" attr.name="type" attr.type="String" />
   <key id="1" for="node" attr.name="name" attr.type="String" />
   <key id="2" for="node" attr.name="attacktype" attr.type="String">
       <default>attack</default>
   </key>
   <key id="3" for="node" attr.name="Cost of Attack" attr.type="long">
       <default>20</default>
       <and>SUM</and>
       <or>MIN</or>
       <lowerbound>5</lowerbound>
       <upperbound>NULL</upperbound>
   </key>
   <key id="4" for="node" attr.name="No Equipment" attr.type="boolean">
       <default>FALSE</default>
       <and>AND</and>
       <or>OR</or>
       <lowerbound>NULL</lowerbound>
       <upperbound>NULL</upperbound>
   </key>
   <graph id="0" edgedefault="directed">
       <node id="0">
           <data key="0">AND</data>
           <data key="1">Goal for Attack</data>
       </node>
       <node id="1">
           <data key="0">LEAF</data>
           <data key="1">Attack Leaf</data>
           <data key="3">50</data>
       </node>
```

```
        <node id="2">
            <data key="0">LEAF</data>
            <data key="1">Defense Leaf</data>
            <data key="2">defense</data>
            <data key="4">TRUE</data>
            <data key="3">10</data>
        </node>
        <edge source="0" target="1" />
        <edge source="0" target="2" />
    </graph>
</graphml>
```

In Figure A.1 the result can be seen. Since a defense node was added the costs of the attack increase by 10 and for an intruder who has no equipment and only 50 euros available the attack is not possible. Note that for programming all IDs have to be positive integer values. Also all non-boolean values have to be positive.

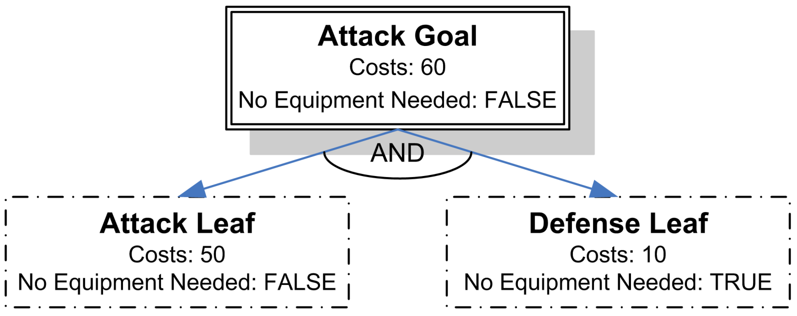
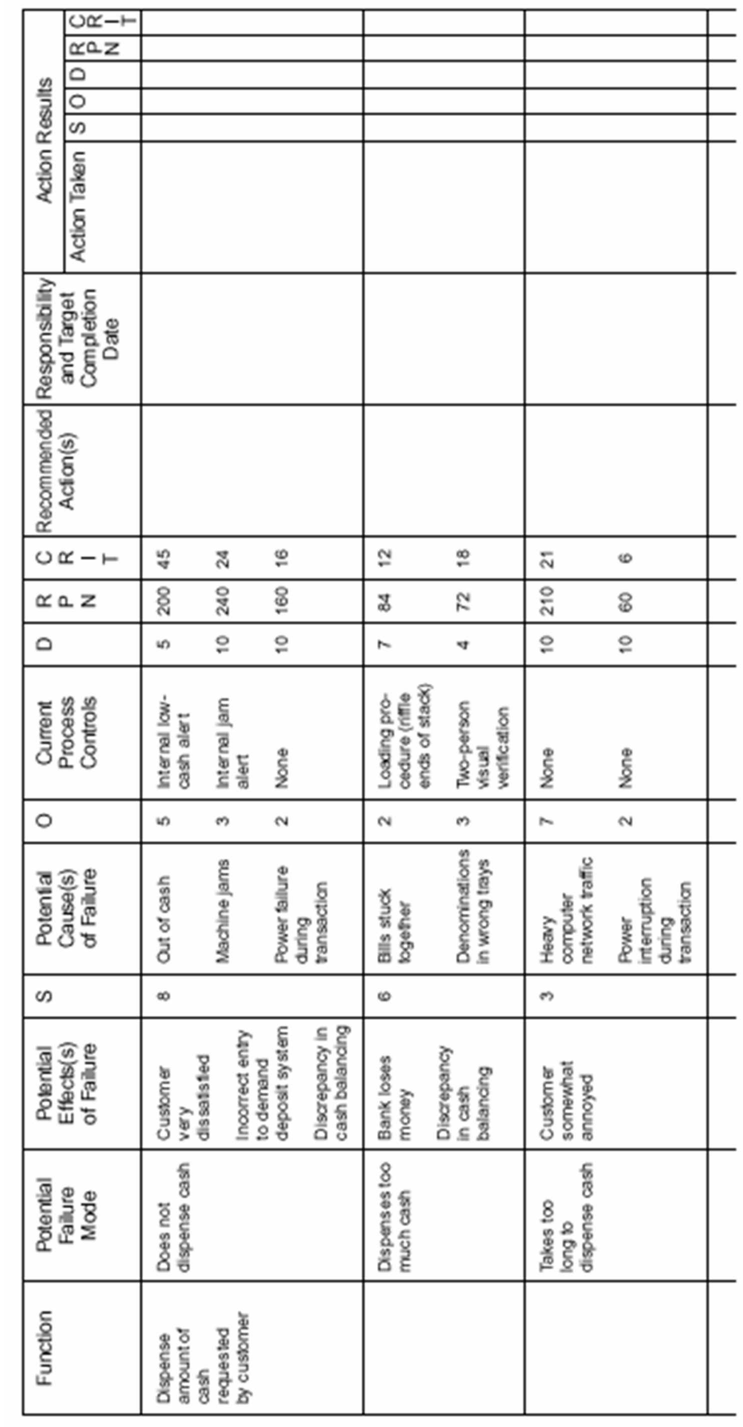

Attack Goal
Costs: 60
No Equipment Needed: FALSE
AND
Attack Leaf
Costs: 50
No Equipment Needed: FALSE
Defense Leaf
Costs: 10
No Equipment Needed: TRUE

Figure A.1: Result of the XML Example

A.2 FMEA Table Example



Function	Potential Failure Mode	Potential Effects(s) of Failure	S	Potential Cause(s) of Failure	O	Current Process Controls	D	RPN	CRIT	Recommended Action(s)	Responsibility and Target Completion Date	Action Results					
												Action Taken	S	O	D	RPN	CRIT
Dispense amount of cash requested by customer	Does not dispense cash	Customer very dissatisfied	8	Out of cash	5	Internal low-cash alert	5	200	45								
		Incorrect entry to demand deposit system		Machine jams	3	Internal jam alert	10	240	24								
		Discrepancy in cash balancing		Power failure during transaction	2	None	10	160	16								
	Dispenses too much cash	Bank loses money	6	Bills stuck together	2	Loading procedure (riffle ends of stack)	7	84	12								
		Discrepancy in cash balancing		Denominations in wrong trays	3	Two-person visual verification	4	72	18								
	Takes too long to dispense cash	Customer somewhat annoyed	3	Heavy computer network traffic	7	None	10	210	21								
				Power interruption during transaction	2	None	10	60	6								

Figure A.2: Example of a FMEA Table

This example is taken from [25].

Appendix B

Logical Operators

B.1 Tables

Input 1	Input 2	Output
false	false	false
false	true	false
true	false	false
true	true	true

Table B.1: Table for the AND Operator

Input 1	Input 2	Output
false	false	false
false	true	true
true	false	true
true	true	true

Table B.2: Table for the OR Operator

Input 1	Input 2	Output
false	false	true
false	true	true
true	false	true
true	true	false

Table B.3: Table for the NAND Operator

Input 1	Input 2	Output
false	false	true
false	true	false
true	false	false
true	true	false

Table B.4: Table for the NOR Operator

Input 1	Input 2	Output
false	false	false
false	true	true
true	false	true
true	true	false

Table B.5: Table for the XOR Operator