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 report. 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 and proof reading the report. My office mate Koos Gadellaa for his great support on programming issues and experienced talks. Thank you for proof reading parts of this report and 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. Also I want to thank all the other people of the Formal Methods group. 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) - and my girlfriend Christin Meßner for her daily e-mail contact which always gave me the feeling to be not alone. Thank you.

Alexander Opel, March 2005
Contents

List of Figures v

List of Tables vii

Introduction ix

1 Project Description 1
   1.1 Problem Statement ................................. 1
   1.2 Goals ............................................. 1
   1.3 Expected Result .................................. 2
   1.4 Parties involved .................................. 2
   1.5 Planning .......................................... 2
   1.6 Risk Analysis ..................................... 3

2 Attack Trees 5
   2.1 Bruce Schneier’s Attack Trees ....................... 5
   2.2 Similar Techniques .................................. 7
   2.3 Tools .............................................. 8
   2.4 Extensions ......................................... 8
      2.4.1 AND/OR Formulas ............................... 8
      2.4.2 Confluent Branches ............................. 9
      2.4.3 Defense Nodes .................................. 9

3 Design and Implementation 11
3.1 Functionality ......................................................... 11
  3.1.1 Creating and Modifying Trees ............................... 11
  3.1.2 Add Values to the Tree ...................................... 14
  3.1.3 Calculation .................................................... 14
  3.1.4 Creating special Attacks .................................... 15
3.2 File Format: XML ................................................... 15
  3.2.1 The GraphML File Format .................................... 16
  3.2.2 My Extensions and Definitions ............................... 18
3.3 Program Structure .................................................. 20

4 Further Work .......................................................... 23
  4.1 Experienced Limitations and possible Solutions ............... 23
  4.2 Further Development .............................................. 25

Conclusion ............................................................... 27

Bibliography ............................................................. 29

Declaration ............................................................... 31

A XML Example .......................................................... 33
List of Figures

2.1 Simple Representation of an Attack Tree .......................... 5
2.2 Simple Example of an Attack Tree ................................. 6
2.3 Example with confluent branch .................................. 10
2.4 Example with a defense node ............................... 10

3.1 The Program with the continuous Example ......................... 12
3.2 Connection from a Node to a Leaf of a lower Layer .......... 13
3.3 Pop up Menus on a Leaf and a Node ............................. 13
3.4 Dialogs for Adding Value Types .............................. 14
3.5 Mask to set the Profile of an Attacker ....................... 15
3.6 Result of the XML Example .................................... 20
3.7 Java Structure of the Program ............................... 21

4.1 Node with mixed operation-type values ....................... 24
List of Tables

1.1 Time shift of the Internship .................................................. 3
2.1 Possible Operations on Nodes .................................................. 9
Introduction

The curriculum of the “computational visualistics” education includes an internship in the advanced study period. Since I am very interested in the field of IT-Security I decided to complete my internship in this domain. Additionally I wanted to go abroad to a country where English is also spoken. During the search of a suitable place, I came across at the Eindhoven Computer Science Security Group, where they offered me an assignment regarding Attack Trees. Since I was not familiar with Attack Trees, I created a small overview for myself, which aroused my interest. Also programming and working with technologies I never used before were the main arguments on my decision to accept the assignment.

This report is divided into four chapters. The first one gives an overview on the whole project. In this overview the project is explained with its goals and expected results and how the time frame of the internship was planned. Chapter two describes what is behind Attack Trees and how they work. Furthermore some similar techniques and tools are presented. At the end an explanation is given how to extend Attack Trees with further functionality. The third Chapter will show how the implemented program is designed and how the Attack Trees with the extension mentioned above are realized. Finally, in the fourth chapter some suggestions for further development are made and a conclusion is given.
Chapter 1

Project Description

1.1 Problem Statement

The intention of this work is to build a tool for Attack Trees. When first introduced in the literature [1] Bruce Schneier describes them 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. So, one can predict how and where intruders pose the greatest threats against a system [2] and one is able to take reasonable precautions for securing the system with this information. However the completeness of the Attack Tree depends on the overview the originator has of the whole system, and (during the construction phase) the possible events that might occur to the system. In small systems it is possible to remain a complete overview, but in a big environment the overview is rapidly lost [3]. In order to deal with this, the Attack Tree should be easily expandable so that if new attacks occur it is possible to intervene fast. Furthermore the tool is supposed to be extended to support defenses. In these so called Defense Trees the user can attach a countermeasure at a leaf to stop the attack in this part of the tree. Normally each countermeasure gives rise to new attacks, which can be again repelled. But this increases the costs and consequently it makes the attack in this part of the tree more improbable. So the project describes how one can realize the Attack Trees and how one can deal with the problems which occur during the work with Attack Trees.

1.2 Goals

The main goal of this project is to design and implement a support tool for Attack Trees. For that purpose an overview of Attack Trees has to be worked out at the beginning. This includes the study of literature, existing tools and talking with more experienced
people in the field of Attack Trees.

The sub-goals are to get a definition of an Attack Tree, to get a design for the program (including data structures, algorithms, user interface and inputs/outputs), an implementation of the design and finally a test of the tool. In addition there is the possibility to improve the tool by an incremental approach of the design and the implementation.

Further the source code should be open, a manual and other support must be made available on a website.

1.3 Expected Result

The expected result of this project are the support tool and a report. On one hand, the report describes the concept of an attack tree, and on the other hand the report will contain the design and implementation of the support tool. The program ought to be written in a modular fashion, so that extensions are possible.

1.4 Parties involved

The student that will be working on this project is Alexander Opel. He will be supervised by a supervisor from the Eindhoven University of Technology.

The supervisor from the TU/e for this project is 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 is 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”.

1.5 Planning

Project start-date: October 11, 2004
Project end-date: March 11, 2004

An internship of the curriculum of the “computational visualistics” education takes 20 weeks to complete. See Table 1.1 for detailed plan.
### 1.6 Risk Analysis

This is an internship and things can go wrong.

**unclear goals / expectations** It is important to have clear goals and clear mutual expectations to make a project a success. Sections 1.3 and 1.2 should make the goal and expectations clear.

**more work than expected** It is always possible that the planning was not completely accurate and there is not enough time to finish the work at the end of the project. To solve this, the project could be divided into smaller parts that should be reached successively. So parts of the complete project are finished and can be used for further development by other parties.

**bad communication** A reasonable amount of projects that fail, fail because the communication is not good enough. The student and the supervisor will meet once a week at the start of the project to discuss the progress of the project. Later on in the project, the frequency of these meetings will probably go down.

<table>
<thead>
<tr>
<th>Week</th>
<th>Month</th>
<th>Days</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oct.</td>
<td>11.-15.</td>
<td>Project plan</td>
</tr>
<tr>
<td>2</td>
<td>18.-22.</td>
<td></td>
<td>study literature, tools; talk to experienced people</td>
</tr>
<tr>
<td>3</td>
<td>25.-29.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Nov.</td>
<td>01.-05.</td>
<td>definition, design, implementation, 1st test</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>08.-12.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>15.-19.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>22.-26.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Nov. / Dec.</td>
<td>29.-03.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>06.-10.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>13.-17.</td>
<td>incremental approach</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>20.-24.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Jan.</td>
<td>10.-14.</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>17.-21.</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>24.-28.</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Jan. / Feb.</td>
<td>31.-04.</td>
<td>start final report</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>07.-11.</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>14.-18.</td>
<td>make available: source, web page, manual, etc.</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>21.-25.</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Feb./Mar.</td>
<td>28.-04.</td>
<td>final report for revision</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>07.-11.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1: Time shift of the Internship
Chapter 2

Attack Trees

To provide a secure system, one needs to know where attackers might attack the system. Finding all the possible attacks is a complex process. But this is a necessary step. If the possible steps of an attack (attack paths) are known, countermeasures against those attacks can be formulated. And if one can put oneself in the position of an intruder one can improve these countermeasures. The knowledge of the intentions and skills of an attacker (the attacker profile) can help to understand how a system’s protection can be improved against such intruders.

2.1 Bruce Schneier’s Attack Trees

In December 1999 the Dr. Dobb’s Journal published an article by Bruce Schneier about a method how to describe the security of a system [1]. As the basis for his description he used varying attacks, means of 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 he put the goal of an attack. 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 of each branch 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.

Figure 2.1: Simple Representation of an Attack Tree
In an Attack Tree there are two different types 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. They are steps to fulfill the goal. As one can see in Figure 2.1 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.

For further explanations we will take a running example. In Figure 2.2 one can see an example 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 was built, one can assign values to the leaf nodes. 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 2.2 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 one can calculate the values of the other nodes. 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 become also possible and impossible, if a minimum of one of the children is impossible.
Besides possible or impossible one can also assign other boolean values like special equipment needed or no special equipment et cetera. 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 one has to take the cheapest child value for an OR node and for the AND nodes one has to take the summation of all their child values.

The combination of both value types is also possible, so one can better describe a profile of an attacker. For instance, the case where the intruder wants to break into the house with no equipment and in the shortest time can be considered. To play around with the eventualities one learns much about the system’s security. With this knowledge one can secure the vulnerabilities of the system. One can also create different attacker profiles for persons one is worried about. So one can develop adequate countermeasures. For example, it could be better to put a key at a safe place than to buy a safer and much more expensive lock.

An undefined Case

In Bruce Schneier’s paper an undefined case arises. If one has more than one value of the same type in the nodes it is possible that the calculation isn’t unique and consequently undefined. For example, one has an attack goal with two OR-connected leaves. Each leaf has a value for the time and a value for the money. So, if one need 5 minutes and 3 euros for leaf one and 3 minutes and 5 euros for leaf two the goal becomes 3 minutes and 3 euros if one calculates each value separately. But if one creates the 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 one has to create a rank of the properties (see Section 4.1).

2.2 Similar Techniques

In the literature one can find similar approaches with the same methodology. All approaches introduced here are based on a tree with a root node and one or more leaf nodes.

Fault Trees: A Fault Tree is a graphic model that is used for the analysis that can be simply described as an analytical technique, whereby an undesired state of the system - the root of the tree - is specified (usually a state that is critical from a safety standpoint), and the system is then analysed in the context of its environment and operation to find all credible ways - the subtrees with nodes/leaves - that contribute to the undesired event. The fault events of the tree are linked by logical operators AND and OR. [4, 5]
Event Trees: The logic used in Event Trees is different to Fault Trees because 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 process is therefore the opposite of Fault Trees. Event Trees do not include decision points requiring the logical operator’s OR and AND. They are based on binary logic, in which an event either has or has not happened or a component has or has not failed (partitioning of the tree in each case in two subtrees). [6, 7]

Attack Graphs: Attack Graphs are a natural application of scenario graphs as outlined by Sheyner [8]. They are organized like Attack or Fault Trees. However, unlike Attack or Fault Trees, the possibilities of cyclic dependencies [9, page 4] or merged states exists [10].

2.3 Tools

For the project the study of different tools for Attack Trees was necessary to see how a program might look like and what the functions of a program might be.

SecurITree: The tool named SecurITree [11] from Avenaza is a commercial tool for building Attack Trees. One can build up a tree with nodes, which are AND or OR connected. Every node can be assigned attributes which describe the behaviour of the sub nodes. For example, the value of the sub goal of the nodes can be the minimum, maximum, product, average, sum, boolean NAND and boolean NOR of the sub nodes. An attribute can also be evaluated with percentage. After the assignment one can analyse the tree. On the one hand, one can trace a path through the tree with the greatest influence or the most critical path and on the other hand, for every attack scenario one can see the costs and the necessary steps through the tree.

TANAT: The tool TANAT [12] - Threat ANd Attack Tree Modeling plus Simulation - is part of the dissertation of Harald Görl. Since the program is in an early stadium and has no manual, information regarding this tool is scarce.

2.4 Extensions

To provide more functionality in the Attack Trees which Bruce Schneier described, the following extensions were added to the tool.

2.4.1 AND/OR Formulas

Bruce Schneier only described two types of nodes - the AND nodes and the OR nodes. In the way that a continuous value is present the calculation is either the summation of
all children values (AND node) or the values becomes the minimum value of all children values (OR node). Or if boolean values are present the node types correspond to the boolean operations.

In the extensions one can assign more operations to both types of nodes. Besides the summation operation for AND nodes and the minimum operation for OR nodes one can also use a maximum operation, an average operation and a product operation for non-boolean values. Now all five operation can be assigned to both of the node types. So it is possible that the AND nodes become the minimum of all child nodes an the OR nodes become the summation of all their children nodes. For boolean values one can only use operations which are commutative and associative. Due to the fact that the order of nodes can differ, the operations have to have these properties. So the program supports the logical operations AND, OR, NAND, NOR and XOR. As in the case of the non-boolean values one can assign all 5 operations to both types of nodes. Finally one has two types of nodes (AND and OR Nodes) in the tree and for each value type it is possible to assign one of the five operations to each node type (see Table 2.1).

<table>
<thead>
<tr>
<th>boolean values</th>
<th>non-boolean values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Minimum</td>
</tr>
<tr>
<td>OR</td>
<td>Maximum</td>
</tr>
<tr>
<td>NAND</td>
<td>Summation</td>
</tr>
<tr>
<td>NOR</td>
<td>Average</td>
</tr>
<tr>
<td>XOR</td>
<td>Product</td>
</tr>
</tbody>
</table>

Table 2.1: Possible Operations on Nodes

2.4.2 Confluent Branches

A further extension is the possibility of confluent branches. This means that a node or leaf could have more than one parent. So the Attack Tree becomes a graph. In the running example (Figure 2.3) one can see that the door can be opened by using force and also the windows can be broken by using force. So only one node for this action is necessary in the tree.

2.4.3 Defense Nodes

If one has a profile of an attacker one can put oneself in the position of such an attacker and one can trace his path through the tree. So for example if he has only 20,000 euros for an attack available he uses the cheapest attack against our 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 one could add an adequate countermeasure. Consequently the costs grows and this attack isn’t worthwhile anymore for this attack profile.
In the example (see Figure 2.4) a break-in through the windows should be eliminated. So if one use secure materials like shatterproof glass 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.
Chapter 3
Design and Implementation

One main goal of this project was the usability of the program. Since there is a Java Runtime available for almost every operating system the decision to write the tool in Java was easy. Besides the extensions to Attack Trees described in Section 2.4 the program should also have an open file format. So an exchange with other programs and tools is possible.

3.1 Functionality

Because the program should have the same graphical user interface on every system the Java Swing library was used. In order to obtain quick possible results the JTree functionality to display the Attack Tree was used. This enables the use of mouse interactions on the JTree to manipulate an Attack Tree. Also one has the possibility of changing the view to XML so one can see how the Attack Tree is represented in a external file. An option for displaying the Attack Tree in a graphical view is also prepared but not implemented yet.

3.1.1 Creating and Modifying Trees

For building and changing the appearance of a Tree an editor was implemented. Because the edit functions differ from leaf nodes and AND/OR nodes two pop up menus were created. With the pop up menu on AND/OR nodes one has the following options available. Some of them are self-explanatory.

- change the type of a node: AND node to OR node or vice versa
- change the name of the node
- add a LEAF node or AND node or OR node
3.1. Functionality

Figure 3.1: The Program with the continuous Example

- **delete only the sub nodes**
  Deletion of nodes or leaves is done recursively. If a sub node is part of a confluent branch the program only deletes the connection between the nodes. So the other parts of the tree can still use the old joint nodes and leaves.

- **delete the node with all sub nodes**
  Additionally to the item above the node is also deleted.

- **connect with other nodes**
  For the confluent branch functionality an AND/OR node can connected to every other node or leaf in the tree - except the nodes in the same layer or children and parents of the node. A special case is the connection between a node and a leaf in a lower layer. Normally the direction of a connection is established from a lower to a higher layer but in this case the lower leaf node is pulled down in the tree and the leaf becomes a child of the node (see Figure 3.2). Since one cannot 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.

- **mark/unmark the node as a Defense Node**
  If one marks a node or leaf as a defense node the node is highlighted in the JTree in a different color.
Figure 3.2: Connection from a Node to a Leaf of a lower Layer

The pop up menu on leaf nodes looks a little bit different. The following options are available:

- change the LEAF node to an AND or OR node
- change the LEAF name
- modify the values of the LEAF
- delete the LEAF node
- connect the LEAF with other nodes
  Towards the AND/OR nodes connections from a leaf is only possible to nodes of a lower layer - except their parents.
- mark/unmark the LEAF as a defense node

During one build an Attack Tree it could happen that a node is highlighted in red. This means that this branch of the tree has no leaf node and so there is no calculation possible in this part of the tree.

Figure 3.3: Pop up Menus on a Leaf and a Node
3.1.2 Add Values to the Tree

To modify values in the leaves of the Attack Tree one has to define the kind of values first. For example, one can add 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 and so on. The number of possible properties is not limited. The value type can adopt boolean and continues values. As AND and OR formulas the operations mentioned in Section 2.4.1 are available. Furthermore one has to assign a default value for each type. This value is automatically set for the leaf nodes until one changes it via the pop up menu option Modify Leaf Data. If one chooses a non-boolean value type 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 one has only this two options available. With these two options one can also set a lower and upper bound for values. So it is possible to say that, for example, the costs for the actions in a leaf can only be set between 100 and 200 euros. It is also possible to use only one bound. Notice that the bound option is optional for the value type definition.

![Figure 3.4: Dialogs for Adding Value Types](image)

3.1.3 Calculation

At the point where a value type is set the calculation in the tree is done automatically. One can move the mouse over the nodes in the JTree and one can see the calculated values in a tool tip. Now one know what is necessary to achieve the goal in the root node or the subgoals in the AND/OR nodes. Notice that all branches in the tree have to end with at least one leaf node. Also one have to notice if one has more than one value for the nodes that the values in the AND/OR nodes can originate from different sub nodes or leaves. The program chooses for each value type (Money, Time, Equipment,...) the best value depending upon the AND/OR Formula. So it is possible, that for example for an AND/OR node the money is taken from a different child as for the time.
3.1.4 Creating special Attacks

If one knows the profile of an attacker one can create a special attack from his point of view. In a mask (see Figure 3.5) one can specify the properties of the intruder, for example that he has special equipment and 1000 euros available (the attack should cost below 1000 euros). To set a mask one has to activate at least one option. All matching nodes are highlighted in the JTree in a light green color. One has to notice that the highlighting is done for nodes and leaves where the values are calculated separately.

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 this applies the possibility to recompute the tree only with the matching paths exist. 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 view one can also add nodes and leaves. Because this view 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. So one can only change the leaves and add there nodes at this position of the tree. This view also provide the possibility to save this part of the tree. If one goes back to the original view both trees are combined. So all new defense nodes and leaves are in the original tree, too.

For a better solution for the highlighting process and the calculation with respect to the operation types please see Section 4.1

3.2 File Format: XML

As a must-have the use of an open file format for the representation of the tree was necessary for the program. Today the Extensible Markup Language (XML) [13] is used everywhere. The advantages of XML are that it is an recommendation of the World Wide
Web Consortium (W3C), it is possible to create a Document Type Definition (DTD) for the validation and it is an open standard. With the DTD it is possible to create an own definition for attack trees. This has the advantage that different programs can use the same validated file to act with Attack Trees. Since XML is very popular there are many libraries for different programming languages available.

### 3.2.1 The GraphML File Format

Because Attack Trees were extended with the confluent branch functionality the tree obtain properties of a graph. So a XML based graph file format was required and it turned out that GraphML has the best requirements for the program. “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] In the following example one can see a simple GraphML document consists of the root element *graphml* an the sub elements *graph, node, edge*. This example only has the definition of the nodes and edges, no values are present.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
    http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <edge source="n0" target="n1"/>
    <edge source="n0" target="n2"/>
    <edge source="n1" target="n3"/>
    <edge source="n2" target="n3"/>
  </graph>
</graphml>
```

The graph is defined by the *graph* element. Inside this element the nodes and edges are declared. The *graph* element as well as the *node* element and the *edge* element have an unique identifier. They are specified by the XML-Attribute *id*. For both *node* and *edge* elements the identifier is optional.

Since a graph could contain directed and undirected edges one has to declare a default value inside the *graph* element as XML-Attribute *edgedefault*. This attribute can adopt the values *directed* and *undirected*. If an edge has no XML-Attribute for the direction type the default value is taken. To specify a different value for the direction than the default value one has to set the Attribute *directed* in the *edge* element. Thereby one can
assign the values \textit{true} for a directed edge and \textit{false} for an undirected edge. Also an edge needs two endpoints which are defined by the Attributes \textit{source} and \textit{target}. The values of the endpoint attributes have to be the identifier of nodes in the same graph.

<edge id="e1" directed="true" source="n0" target="n2"/>

To define properties of the graph one has to define \textit{key} elements which have an \textit{identifier}, a \textit{name}, a \textit{type} and a \textit{domain}. Like in the other elements the identifier is set by the XML-Attribute \textit{id}. The name of a property is defined by the attribute \textit{attr.name} and it has to be unique. The type of a property can be either \textit{boolean}, \textit{int}, \textit{long}, \textit{float}, \textit{double} or \textit{string} and is specified by the XML-Attribute \textit{attr.type}. They are defined like the data types in Java. The domain specifies which element adopt the property defined in the \textit{key} element. Possible values are \textit{graph}, \textit{node}, \textit{edge} and \textit{all}. The following example shows a property for the color of an edge. The color is specified as a string value.

<key id="d1" for="edge" attr.name="color" attr.type="string"/>

In addition it is possible so define a default value. To set the color blue as default the example is extended to:

<key id="d1" for="edge" attr.name="color" attr.type="string">
  <default>blue</default>
</key>

If one wants to set a value for a property set by the \textit{key} element one has to define a \textit{data} element nested inside the element for the graph element. In the next example one can see that the edge has now the color yellow:

<edge id="e1" directed="true" source="n0" target="n2">
  <data key="d1">yellow</data>
</edge>

And finally all properties together in one example:

<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
  http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <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>
  <node id="n0">
  </node>
</graphml>
As one can see edge $e0$ has the color yellow whereas the other edges are blue, because a default value is set for the color. Since there is no default value for the weight some nodes have no value. They are undefined. In this example this is the case for node $n1$, $n2$ and $n3$.

### 3.2.2 My Extensions and Definitions

The GraphML format provides a good basis for the XML representation of the Attack Tree. To integrate the extensions mentioned in sections 2.4 into the XML file the GraphML format has to be extended. Also some definitions for the input data have to be made.

Every tree representation has at least three key elements nested in the *graphml* element. The key element with id=0 always describes the type of a node. Only the values AND, OR or LEAF are possible.

```xml
<key id="0" for="node" attr.name="type" attr.type="String" />
```

The second key element is the definition for the node name. Thus the names for the nodes and leaf are made possible.

```xml
<key id="1" for="node" attr.name="name" attr.type="String" />
```

The last must-have key element describes the kind of nodes and leaf. The key with id=2 define if a node is an attack node or a defense node. For this value the default value *attack* is set. The other possibility is the value *defense*.

```xml
<key id="2" for="node" attr.name="attacktype" attr.type="String">
  <default>attack</default>
</key>
```
If one defines new values for the actions - like the costs, time or the need of equipment - the key element is extended by 5 sub elements. Besides the sub element default one has the elements and, or, lowerbound and upperbound. The values of the and and or elements describe the formulas in the AND and OR nodes. In the attribute attr.type of the key element three values are possible: boolean, long and double. 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 put into the XML-file. Notice that if borders are present, the default value has to be in the range of the bounds.

```
<key id="3" for="node" attr.name="Costs for Attack" attr.type="long">
  <default>20</default>
  <and>SUM</and>
  <or>MIN</or>
  <lowerbound>5</lowerbound>
  <upperbound>NULL</upperbound>
</key>
```

In the boolean case both bounds are set to NULL. Also as default only TRUE or FALSE are accepted. Of course only the logical operations mentioned in section 2.4.1 are possible.

```
<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>
```

Since we have two key elements with no default value (id=0, id=1) a node element has at least two data sub elements. The one with the value type and the one with the node name.

```
<node id="0">
  <data key="0">AND</data>
  <data key="1">Attack Goal</data>
</node>
```

If one has leaves with values which differ from the default value set in the key element the node element has further data sub elements. Notice that only LEAF nodes have values.

```
<node id="1">
```
In the example above the LEAF Attack Leaf is not a defense node, it has the costs of 50 and it don’t need equipment. With the following Defense-LEAF and EDGEs we complete the small example.

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

In Figure 3.6 one can see the result. Since we add a defense node the costs of the attack increase and for an intruder who has no equipment and only 50 euros available the attack is not possible. Notice that for programming all IDs have to be positive integer values. Also all non-boolean values have to be positive. See Appendix A for complete XML code.

![Figure 3.6: Result of the XML Example](image)

### 3.3 Program Structure

The program was built in a modular fashion (see figure 3.7). So the main program (`AttackTree.java`) is divided into three parts. The menu, the main area and a status bar. The file menu contains the possibility to open and save the AttackTree XML files. To get the filename for saving and opening the two files `FileOpenDialog.java` and `FileSaveDialog.java` are taken. For filtering the file extensions in the dialogs the `ExtensionFilter.java`
Figure 3.7: Java Structure of the Program

is used. The file `StartCard.java` contains the content of the main area. On the left side of this area are three boxes. From the first box one can set up the value types and the according node operations (`NodeProperties.java`). In the second box one can change the view of the Attack Tree. The “List Tree View” and the “XML View” are already implemented. And the last box contains the possibility to define the attack properties in a dialog (`MaskProperties.java`) and the possibility to perform an attack. Each view in the content area has its own file. The `ListTreeView.java` file displays the JTree, the `XMLView.java` file displays the content of the XML file and later it is possible to implement a graphical view of the tree in the `GraphicalTreeView.java` file. For customizing the JTree with specials colors and icons the file `CustomTreeCellRenderer.java` was written.

For easy handling the data of the tree some data structures were implemented in different files.

- `GraphMLKey.java` This data structure contains all data of a key-element of the XML file.
- `Node.java` This data structure contains all data of a single node.
• *NodeData.java* This data structure contains all data of value in a node.

• *AttackData.java* This data structure contains all date of a value type which is used for an attack.

Also a utilities class (*Utilities.java*) contains functions which are used by different classes.
Chapter 4

Further Work

4.1 Experienced Limitations and possible Solutions

As mentioned in Section 3.1.4 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 one creates a profile of an attacker all values of the matching nodes of the profile should originate from only one node if operations are present where only one value determines the result. To find matching nodes and leaves one has to differ between operations where all sub values have influence and operations where only one value has an influence.

In the case that only one value determine the result of an operation (minimum, maximum, OR, NOR, XOR) the undefined case mentioned in section 2.1 could occur. The described example has two OR-connected leaves where the minimum operation is set for both values. Leaf one has the value 5 for the time and value 3 for the money and leaf two has the value 3 for the time and value 5 for the money. With the attack profile Cheapest Attack in shortest Time it is not clear which leaf values are taken for the root node. To solve this case an ordering in the profile has to be established. This has to be done in the case if more than one non-boolean value is present. So in the example one can set the money to a higher priority as the time and the root node now receives the values of leaf two (money: 3, time: 5). The ordering function also eventuate if more than one profile matching sub element is available. Thus it is meant that if a profile like Money below 6 and Time below 6 is present both leaves are matching to the profile in the example and leaf two is taken for the values in the root node due the fact that the money has a higher priority. So a node does not receive values from different sub elements anymore.

In the case that more than one value determines the result of an operation (summation, average, product, AND, NAND) all sub nodes of a node have to match to the attackers profile. If this applies 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.
Also the question occurs if the extension that there is no difference between the operations in AND and OR nodes mentioned in Section 2.4.1 makes sense. The advantage in this extension is that for both node types one can have operations whose results are affected by more than one value (for example: OR - summation, AND - average). Also it could be possible to add more node types. So it might be better not to speak of AND and OR nodes any longer but of node type one, node type two and node type three. The other way round it would be better if 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. But in this case the combining possibilities decrease and the case of mixed operation types doesn’t occur. So 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 4.1). The lack of this possibility would restrict much the functionality of the Attack Tree.

In the case of example 4.1 the creator of the tree should know which operations for the non-combining value (skill level) make sense. To achieve the goal one has to fulfill sub goal one and sub goal two. The time which is needed to fulfill the goal is the summation of both children (it becomes 10 minutes) and the skill level becomes the highest value of the children (skill level 6). It wouldn’t make sense if the skill level is connected via a minimum operation. Also if only one sub goal needs equipment it wouldn’t make sense if to fulfill the goal one does not need any equipment.

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. So the profile Minimum Skill Level highlights sub node one and the root node in the example, since one needs at least a skill level of 6 to fulfill the goal.

Also in this case no order is necessary for the non-combining values because all children have to be fulfilled and the parent node gets the values which are at least necessary to achieve the goal. So it does not matter from which children the values are taken.
4.2 Further Development

During the process of this internship some new ideas have risen for the program which could not be implemented. Some of them need major changes of the program.

- Revise of the calculation process to solve the experienced limitations mentioned in Section 4.1.
- Expand the program to support more than two different node types (more than AND nodes and OR nodes). One might add new node types dynamically.
- Show for each leaf the path(s) to the root node and recalculate the values. One has to watch out for nodes which have values with combined operations.
- Possible changes in the mask to create an attack profile.
  - If one set up a lower bound and/or a upper bound one might put them as default values into the mask.
  - The possibility to use the minimum or the maximum as attack criteria.
  - Different attack criteria might be alternatives and not only connected via AND.
- Import of Attack Trees and add them at nodes in a new tree. The program has to check the value types and nodes types of both trees for the import.
- Add a validation for the XML file. For example a DTD file. It can be used to check a file before opening in the program.
- The input data has to be validated.
- A different implementation of the defense nodes. They could be relative to a node (e.g. increase the value of a node by 40). If one add new attacks to this defense nodes the values of these attack nodes mustn’t destroy the defenses.
- Implementation of a graphical view with all functions of the JTree view.
- Drag and drop functionality in the JTree view and in the graphical view.
Conclusion

With Attack Trees one has the possibility to accelerate the process of securing a system. It is possible to create complex trees. During the creation process one learns much about a system and one can find out where possible vulnerabilities are. The creator has to know what is going on in the system and how the system works. Only then it is possible to create a useful Attack Tree. With the potential to simulate an attack one can react fast and one can add appropriate countermeasures. Also one can find attack paths 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 tool can support to find these possible attacks and paths respectively. If one has a big Attack Tree of a big system one needs a tool to keep the necessary overview and with a tool one has the possibility to test different attack scenarios without much effort.
Bibliography


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

   http://www.cs.virginia.edu/~pev5b/writing/academic/thesis/


   http://www.arescorporation.com/ares/rm/rm_services_riskt_faulttrees.html

    technological risk Assessments (continued)
    volume3/chapter_e_part3.htm

[7] IEE - Health and Safety Briefing 26 - 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

[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

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

[12] TANAT - Threat ANd Attack Tree Modeling plus Simulation; Dipl.-Inform. Harald Görl
   http://www13.informatik.tu-muenchen.de:8080/tanat/

   http://www.w3c.org/TR/2004/REC-xml-20040204/

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

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

Eindhoven, March 11, 2005          Alexander Opel
Appendix A

XML Example

Complete example from Section 3.2.2:

```xml
<?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>
  </node>
</graphml>
```
<graphml>
  <graph bidirectional="true">
    <node id="0">
      <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>
    <node id="1">
      <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>
  </graph>
</graphml>