MASTER

Secure processing of confidential digital information in a potentially insecure environment

Schrijvers, L.

Award date:
2005

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
MASTER’S THESIS

Secure processing of confidential digital information in a potentially insecure environment

by
L. Schrijvers

Supervisors:
Dr. S. Mauw (TUE)
Dr. ir. A.M. Bos (Chess iT)

Eindhoven, July 2005
A. Abstract

When working with secret data, the data needs to be handled with confidence. This means that operations on this data, like opening, changing, saving and sending have to be done at a confident way. Sending data can be done via a possibly unsafe channel.

Realising these operations in a confident way is tricky, because doing safe ‘calculations’ on an untrusted environment is hard. To do this, supporting resources must be used. Think of smartcards, dedicated crypto pcmcia cards and cryptographic storing mechanisms.

For parts of this task, standard solutions are available. Question is, for which of these parts are standard solutions known, available and realizable, and can these solutions be combined to a complete solution of the whole problem.

This thesis describes the investigation of this subject. A risk analysis of the situation is performed and extended using attack trees. This leads to approaches for implementing a secure environment. Examples of these are a secure software environment and an insecure environment which is not connected to an (external) network.

An attack tree tool has been developed to determine an appropriate approach for implementing a secure environment. From the attack tree analysis we learned that managing document security is only possible when security measures are taken simultaneously on all aspects.
## B. Table of Contents

### A. Abstract

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i</td>
</tr>
</tbody>
</table>

### B. Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Environment</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Security</td>
<td>1</td>
</tr>
<tr>
<td>1.2.1 Confidentiality</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2 Integrity</td>
<td>2</td>
</tr>
<tr>
<td>1.2.3 Availability</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Project aim</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Document set up</td>
<td>3</td>
</tr>
<tr>
<td>2. Existing research</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Trusted computing base</td>
<td>4</td>
</tr>
<tr>
<td>2.2.1 Introduction</td>
<td>4</td>
</tr>
<tr>
<td>2.2.2 TCB in practice</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Trusted computing group</td>
<td>5</td>
</tr>
<tr>
<td>2.3.1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2.3.2 Boot process</td>
<td>6</td>
</tr>
<tr>
<td>2.3.3 Software</td>
<td>6</td>
</tr>
<tr>
<td>2.3.4 TCG in practice</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Conclusion</td>
<td>7</td>
</tr>
<tr>
<td>3. Related problems and projects</td>
<td>8</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Financial Reader</td>
<td>8</td>
</tr>
<tr>
<td>3.3 Online voting</td>
<td>9</td>
</tr>
<tr>
<td>3.4 VKaart and documents</td>
<td>9</td>
</tr>
<tr>
<td>3.5 Medical systems</td>
<td>10</td>
</tr>
<tr>
<td>3.6 Conclusion</td>
<td>10</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Environment

Consider an office environment where people are working together on digital documents. Each person has a desk with a computer which he uses to work on the documents. When necessary, communication is used to send documents to each other. This can be done via a network.

Working on documents can be seen as a chain of operations on these documents. An operation can be any action, it does not necessarily need to change the document, as long as the document is the subject of the action or is involved with the action. Examples of this are opening, editing or sending the document.

The subject of this project is to investigate if we can make this chain of operations secure. This is interesting because if a secure environment can be created, it can be directly used in similar environments, like companies and other agencies. Before we can dive into this problem, we have to know what security is.

1.2 Security

To get an idea of what security is, we take a look at the definition of information systems security in the National Information Systems Security (INFOSEC) Glossary:

Protection of information systems against unauthorized access to or modification of information, whether in storage, processing or transit, and against the denial of service to authorized users or the provision of service to unauthorized users, including those measures necessary to detect, document, and counter such threats.
The definition of information systems security gives a number of areas of concern about protecting information systems. We can break this down into separate subjects:

- Unauthorized access of information
- Unauthorized modification of information
- Denial of service to authorized users
- Provision of service to unauthorized users
- Measures to detect, document and counter threats

Each of these subjects play an important role in securing information and can be expressed in one or more of the three security principles of the ‘CIA triad’: confidentiality, integrity and availability.

1.2.1 Confidentiality

Information must be kept confidential. This means that accessibility of information must be restricted to authorized users who have the privilege to see it.

1.2.2 Integrity

Integrity can be split into two parts: source integrity and data integrity. Source integrity means that the sender of the information is who he claims to be. Data integrity of information is compromised when the information has been modified before it is received. Data integrity can also be violated when that data is stored. This can be done either by accident or on purpose.

1.2.3 Availability

Normally, availability means that the information systems (e.g. a webserver) must be fast enough to serve the information that is requested by its users. Fast enough means that the information system responds within a certain time which is acceptable for the user. For fulfilling this task, the information system needs resources. If an attacker manages to decrease the amount of available resources, then service to a request is denied. This is called Denial of Service (DoS).
1.3 Project aim

Now that we know what security is, we know in what ways the security of information can be broken. Each of the actions in the chain of operation on the documents in the office environment, is a weak spot on which an attacker can break in.

For some of these weak spots, there may be solutions already. It is preferred to use these solutions as much as possible. In software terms this means that we have to look for Commercial Of The Shelf (COTS) software. But because the solution needs to be as cheap as possible, free software is preferred, and no custom hardware solutions may be used.

The aim of this project is to analyse the problem of secure document processing in a potentially unsecure environment and to provide a solution or parts of a solution to this problem. This will be done by investigating existing projects, analyzing the problem and discussing solution strategies with aid of a tool.

1.4 Document set up

The project starts with looking at existing research and projects. Other research can help by giving ideas for solutions. These projects are described in Chapter 2 and Chapter 3.

After this, a risk analysis of the situation is performed. Chapter 4 contains a description of the used method and the results of the risk analysis. This risk analysis helps to identify security problems which need to be solved. In Chapter 5, the risk analysis is extended with attack trees to get a more structured view of these problems.

When the problems have been identified, we can look for appropriate solutions. An attack tree tool has been developed to reason about these solutions. It is developed as part of this project, to help with finding and describing possible solution strategies. It can be used to evaluate existing solutions. The results from using the tool are documented in Chapter 6. The chapters after this describe the attack tree tool development in detail.
Chapter 2

Existing research

2.1 Introduction

To solve the problem described in the previous chapter, it is useful to look at existing research and techniques. This can be used as possible solution or idea for further research. First, a technique for creating a trusted part of an information system is described. After this, a project which uses this technique is discussed.

2.2 Trusted computing base

2.2.1 Introduction

A trusted computing base (TCB) is a part of a system which can be fully trusted. The TCB closely works together with the core of the Operating System: the kernel.

The TCB is often a part of the kernel of an Operating System, or it acts as a shell around the kernel. In the first case, the whole kernel cannot be fully trusted because it contains untrusted parts, e.g. third party device drivers. In the second case, the kernel is contained in the TCB and the TCB can contain more parts of the system, like a closed system with a kernel and a set of trusted system utilities. The two possibilities are depicted in Figure 2.2.1.

The basic idea is that the TCB works as a separation between the interface and software of a system. It guarantees that user input is well processed and output is correctly displayed. All input and output is first checked by the TCB before it proceeds its path through the system.

For using a system in a secure way, the user has to be convinced that he can trust the TCB. This means that the TCB has to be in a trustworthy state and cannot be tampered with so it leaves that state. If this goal cannot be reached, the TCB should
be tamper evident, which means that it can be proven whether an intruder has changed the system. If this is the case, the user can be warned that the system cannot be trusted anymore.

2.2.2 TCB in practice

In Microsoft Windows, the TCB works as a boundary between two parts of the system, with the difference that in one part normal security policies hold and in the other part, code can run without security checks. This is called the ‘Operating System Privilege’ and is given to kernel code, device drivers and user code (software applications) which runs under the local SYSTEM account. 

This technique can be used for secure document processing, by letting the Operating System only run secure code. An extension to the kernel can be developed which runs with the Operating System Privilege. The extension checks every program execution if it is allowed to run. This can be done via software signing. Every trusted application is signed and only software with a valid signature can be executed.

2.3 Trusted computing group

2.3.1 Introduction

The idea of Trusted Computing Based is used in a project of the Trusted Computing Group (TCG). TCG contains five different companies (Microsoft, Intel, IBM, HP and AMD) and can be seen as the successor of the Trusted Computing Platform Alliance.
(TCPA) and Next Generation Secure Computing Base (NGSCB). TCG is responsible for developing a new computer security standard.

The idea can be broken down into five different components. First, a chip which is contained in modern PC-hardware. This chip is called the Fritz-chip and works as a bridge between hardware and software. The second component is another hardware extension and is a special separated memory.

The software contains two components. The first is a special security kernel, called the Nexus kernel. Further more, every other software application which is used within TCG contain a special security part, which is called NCA.

The last component of TCG is a system of online servers which are used for security tasks, like certification and validation of data.

2.3.2 Boot process

During the boot process of a PC, the Fritz-chip determines whether the PC is in an approved state. An approved state can only be reached after certifying the system. This depends on the hardware of the PC and which software is installed. From this, a hash-value is calculated.

An approved hash-value gives access to a cryptographic Trusted Computing-key (TC-key), which is needed to decrypt Trusted Computing-software (TC-software) and data. If the hash-value is not approved, then there is no access to the TC-key. The boot process of the PC continues, but no TC-software can be started and no TC-data can be read.

2.3.3 Software

The Nexus security kernel of the Operating System (OS), checks if the present hardware and software of the PC can be used. This can be done by for example checking serial numbers against a blacklist.

If a TC-key is present, TC-software can be started and TC-data can be read. The TC-software contains the NCA security part, which protects its data. For example by defending itself against live-debuggers and other software which can break in or hijack other processes.

The software supplier provides software with a key and security policy. The Fritz-chip will only pass through the key to the software application when the system meets the security policy and the environment is trustworthy.

When the software application receives the key, it can be used to decrypt data. The security policy is used to control the usage of the data. Because the software supplier defines the security policy, the software supplier can restrict the usage of data.
2.3.4 TCG in practice

At this time, the project of TCG is not available for production environments yet. There are plans to integrate the link between the hardware and the operating system into the next version of Microsoft Windows (Longhorn) which is planned for release in 2006.

2.4 Conclusion

Although the products of TCG are not usable yet, the idea of building a secure environment using a trusted computing base can be useful to this project. TCG could be a solution to the problem of digital document handling because the ‘supplier’ of a document can restrict the access to the document via the key and security policy. To gather more knowledge about these kind of projects we will take a look at similar security related problems.
Chapter 3

Related problems and projects

3.1 Introduction

In the previous section, a security related technique and a project concept were explained. These can be used in other security related projects. When we take a look at related projects, we can investigate what the corresponding problems are.

The problems described below, are all security related and thus, require one or more of the three security principles. Each of the problems have one or more requirements in common with the problem of this project.

3.2 Financial Reader

Finreader (Financial Reader) is a device which is developed to make payments via internet. The finreader contains a display and a ‘Yes-button’. Because the interface between user and PC is transfered to the finreader, the payment and amount of money cannot be tampered with.

When the user wants to pay an amount of money, the PC sends the amount to the finreader, on which it is displayed. If this amount is correct, the user can confirm it with the ‘Yes-button’. After this, a hash is calculated from the payment instruction using the SHA1 function. The hash is encrypted and sent along with the payment instruction. This is done for checking the integrity of the payment. The payment instruction itself is not encrypted, because secrecy is not important.

In this example, integrity is the most important security principle which has to be met. The integrity of the amount of money and the integrity of the payment itself is important. This can be compared to the integrity of a digital document. If a user opens a document, he needs to be sure that the document which is displayed is the same document which he opened. Even so, when sending a document to someone else, the integrity of the
document may not be violated. However, in this example secrecy is not important, in contrast to the problem of digital document handling.

The Finreader is developed to provide a secure interface between the user and the hardware. For the problem of digital document handling, this would be usable. However, creating such a secure interface is hard to do without using custom hardware.

### 3.3 Online voting

There are a number of security problems with online voting via internet. The reason for this, is that there are the same security requirements as with normal voting, but the implementation of this is harder. This is because it is harder to verify the digital identity of a person. Furthermore, the channel over which the vote is sent is possible insecure.

A vote has to be kept secret and integer. Because of this, an encrypted hash of the vote is sent along with the encrypted vote. Because a vote has to be kept anonymous, possible feedback of the vote cannot reveal any information about the person or the vote, but the person has to know if the vote was succesful or not.

A possible solution for this problem is a closed network with vote-terminals. This solves the problem of a unsecure network. The votes of these terminals are collected and sent to a central processing unit. After the votes have been processed all secret information which is not needed anymore, is deleted.

Integrity and secrecy of the digital vote play an important role in the problem of online voting. This is comparable to how the digital documents need to be handled. The document must be kept secret and integer when it is being sent to another person. The identity of the sender is also important to verify the source integrity of the document.

### 3.4 VKaart and documents

The VKaart is a pcmcia card and is developed for on-the-fly encryption and decryption of documents. The documents are stored on a normal harddrive. The VKaart makes the process of storing documents in a secure way transparent for the user.

When a document is opened by the user, the VKaart reads the file from the harddisk and decrypts it. After this, it remains decrypted in the memory of the PC. So, if a hacker breaks into the PC at that moment, it can find a decrypted version of the document in the memory. When the document is closed, the document has to be removed from memory and virtual memory like temporary files and swap file.

The VKaart is typically developed for secrecy of data. It can be used for storing digital documents. However, when a digital document is openend, it still can be found in the memory of the computer in an unencrypted form. This gives an attacker the opportunity
of stealing the digital document when it has been opened by the user.

3.5 Medical systems

For medical inspection medical equipment is used. For instance a scanner which produces images that have to be processed by a doctor. For this reason, the scanner is connected to a network of workstations and medical equipment. The workstations in the network contain an installation of Microsoft Windows NT and additional medical software to process the images.

The installation of a workstation has to be certified before the scanner can be used. The certification guarantees that the workstation will display the images properly. This is of course needed for taking the right medical conclusions.

The problem is that after a software update or patch of the workstation, the whole system has to be certified again. This process takes a lot of time, so in the worst case when the workstation has been certified, new software updates are already available.

There are two main approaches for handling this problem. First, improve or redesign the system, so that patches and updates are not necessary anymore. Theoretical, this is a good approach, but very hard to realize with a closed operating system like Microsoft Windows with no source code available.

The second approach is to define a security policy with access restrictions, so that only authorized people can access the (still unsafe) system. This can be done by restricting network communication and physical access.

This problem of patching a workstation is similar to the problem of digital document handling. This is because for the best security, the workstations in the office environment also need to be updated regularly. This problem is hard to solve, because new vulnerabilities in software keep being discovered.

3.6 Conclusion

Each of the projects above have to ensure one or more security principles which are connected to the problem of digital document handling. The projects introduce a number of techniques which ensure these security principles.

These techniques can solve a part of the whole problem and can answer questions about how security is handled. These security related questions are also applicable to this project.

For using techniques to solve problems, we first need to know what the problems are. To achieve this goal, a risk analysis is performed to identify the weak spots in the environment.
Chapter 4

Project research

4.1 Introduction

A risk analysis is a useful technique for identifying problems in an environment. Different standard methods are available. In this project we use a custom method derived from the OCTAVE method. An extensive specification of OCTAVE is given in [13].

The OCTAVE method is designed for performing a risk analysis in a large organization. Therefore, some processes are too comprehensive to execute. To solve this problem, we have taken a subset of processes of OCTAVE, to create a custom risk analysis method. This results in a clear complete risk analysis.

First an introduction to OCTAVE is given. This is done by describing three different phases along with the corresponding processes they contain. After this, the custom version of the method is applied to the problem of this project. The description of this includes an explanation of the differences with the OCTAVE method.

4.2 Risk analysis

4.2.1 Introduction to OCTAVE

In this project, we use a custom version of the OCTAVE method for performing a risk analysis. The original method is widely used for IT projects in organizations. First, an introduction to OCTAVE is given.

Preparations: Phase 0

The first step is the preparation phase of the OCTAVE method. In this phase, the available budget is determined. To do this, senior management of the organization has
to support the risk analysis. After this, members of the analysis team are selected. These members perform the processes of the risk analysis. If the analysis team has been composed, it selects the other participants of the risk analysis. These other participants are selected from all levels of employees of the organization.

Phase 0 is not executed in this project.

Phase 1

Phase 1 of the OCTAVE method is used to gather information about the organization. It contains workshops for senior managers, middle managers and staff members. The information which is gathered in the first three processes of this phase, is used to perform Process 4. In this process, the gathered information is processed and analyzed. The result of this is used in Phase 2 and 3.

Summary of Phase 1

There are four processes. For each of the processes, the title, participants and points of interest or actions are described.

- Process 1: Identify Senior management knowledge
  - Participants: Senior managers
  - Points of interest:
    * Identify assets and relative priorities
    * Identify areas of concern
    * Identify security requirements for the most important assets
    * Capture knowledge of protection strategy practises and organizational vulnerabilities

- Process 2: Identify operational area management knowledge
  - Participants: Middle managers (operational area)
  - Points of interest:
    * Identify assets and relative priorities
    * Identify areas of concern
    * Identify security requirements for the most important assets
    * Capture knowledge of protection strategy practises and organizational vulnerabilities

- Process 3: Identify staff knowledge
  - Participants: Staff
– Points of interest:
  * Identify assets and relative priorities
  * Identify areas of concern
  * Identify security requirements for the most important assets
  * Capture knowledge of protection strategy practices and organizational vulnerabilities

• Process 4: Create threat profiles
  – Participants: Analysis team members
  – Activities:
    * Group assets, security requirements and areas of concern by organizational level
    * Select critical assets
    * Refine security requirements for critical assets
    * Identify threats to critical assets

Phase 2

The second phase of the OCTAVE method is used to identify vulnerabilities in the infrastructure of the organization. It uses the following definition of a vulnerability:

"Weakness in an information system, security practices and procedures, administrative controls, internal controls, implementation of technology, or physical layout that could be exploited to gain unauthorized access to information or to disrupt information processing."

When looking for vulnerabilities in IT-components, first weaknesses have to be targeted. The infrastructure is split in five different parts:

• Information systems
• Network services
• Architecture
• Operating systems
• Applications

Each of the five parts contains weakness which can lead to vulnerabilities. The OCTAVE method distinguishes three different categories of vulnerabilities. The categories are:
• Design vulnerabilities
• Implementation vulnerabilities
• Configuration vulnerabilities

There are different approaches available for finding weaknesses in IT-components. The OCTAVE method proposes three different approaches. These can be done by members of the IT-staff or by external technology experts.

The first approach is to look for vulnerabilities in a public list. For example, the Common Vulnerabilities and Exposures (CVE). The second step is to execute a number of tests for checking the IT components. To do this, the OCTAVE method provides a list of tests which can be used as a guideline, which is given below. The last approach is by running vulnerability evaluation tools. These tools can be used to automate common known security evaluation tests.

The OCTAVE IT-component security tests:

• Reviewing firewall configuration.
• Checking the security of public web servers.
• Performing a comprehensive review of all operating systems.
• Listing all system user accounts.
• Identifying known vulnerabilities in routers, switches, remote access servers, operating systems and specific services and applications.
• Identifying configuration errors.
• Looking for existing signs of intrusion (Trojan horses, backdoor programs, integrity checks of critical system files).
• Checking file ownership and permissions.
• Testing password usage and strength.

Summary of Phase 2

There are two processes. Both processes are given with their participants, which information they use, and what activities they have.

• Process 5: Identify key components
  – Participants: Analysis team and members of IT-staff
- Information: Documentation of infrastructure and network topology diagrams

- Activities:
  * Identify system of interest
  * Identify key classes of components
  * Identify infrastructure components to examine

- Process 6: Evaluate selected components
  - Participants: Analysis team and members of IT-staff
  - Information: Evaluation of software tools
  - Activities:
    * Run vulnerability evaluation tools on selected infrastructure components
    * Review technology vulnerabilities and summarize results

**Phase 3**

The OCTAVE method uses a qualitative way of reviewing assets and security risks. For each asset, a scenario is defined in which a risk to this asset comes to expression. In this way, the value of the impact of the risk can be determined.

After the risk analysis, a plan is written to reduce the risk in the organization. This is done via three steps:

- Implementation of new security practices
- Maintenance of existing security practices
- Fix found vulnerabilities

**Summary of Phase 3**

There are two processes. Both processes are given with their participants and activities.

- Process 7: Conduct risk analysis
  - Participants: Analysis team
  - Activities:
    * Identify the impact of threats to critical assets
    * Create risk evaluation criteria
    * Evaluate the impact of threats to critical assets

- Process 8: Develop protection strategy
– Workshop 1
  * Participants: Analysis team and members of the organization
  * Activities:
    · Consolidate protection strategy information
    · Create protection strategy
    · Create mitigation plans
    · Create action list
– Workshop 2
  * Participants: Analysis team and senior management
  * Activities:
    · Review risk information
    · Review and refine protection strategy, mitigation plans and action list
    · Create next steps

4.2.2 Applying OCTAVE in this project

For the risk analysis in this project, an custom derived version of the OCTAVE method is used. The original OCTAVE method contains 8 processes, some of which are not suitable for technical problems. Therefore, the original Processes 1 to 3 are combined into one process. This results in the following list of steps:

- Phase 1: Build asset-based threat profiles
  - Process 1: Identify knowledge
  - Process 2: Create threat profiles
- Phase 2: Identify Infrastructure Vulnerabilities
  - Process 3: Identify key components
  - Process 4: Evaluate selected components
- Phase 3: Develop security strategy and plans
  - Process 5: Conduct risk analysis
  - Process 6: Develop protection strategy

4.2.3 Phase 1: Build asset-based threat profiles

Process 1: Identify knowledge

The first process contains four different activities to gather information about the project.
1. **Identify assets and relative priorities:**
   Because the OCTAVE method follows an asset-based approach, the first activity is used to identify assets in the project. Assets are things that are most valuable to an organization and can be classified into five different categories:

   (a) Information (data)
   (b) Systems
   (c) Hardware
   (d) Software
   (e) People

For each category, the identified assets are given as well as a priority between square brackets. The priority indicates the ‘value’ of the specific asset. Possible priorities are: high, medium, low.

   (a) Information (data):
      • (Secret) Documents [High]
   (b) Systems:
      • Network infrastructure / Internet connectivity [Medium]
   (c) Hardware:
      • Workstation PCs [Medium]
   (d) Software:
      • Workstation software for an office-environment [Low]
        (Operating System, Office-suite, Web browser, Mail client)

2. **Identify areas of concern:**
   If the assets have been identified, the next step is to think of scenarios in which threats to the important assets occur. These scenarios do not have to be specified in detail, as this will be an activity of Process 2.

   For the assets identified in Process 1, the following scenarios can occur. The Risk of:

   • Documents being viewed by unauthorized people.
   • Documents being created by unauthorized people.
   • Documents being changed by unauthorized people.
   • Documents being deleted by unauthorized people.
   • Denial of access to the network or internet.
   • Accessing workstation PCs by unauthorized people (intruder).
   • Denial of access to workstation PCs because of hardware failure.
3. **Identify security requirements for the most important assets:**
   For each of the most important assets, security requirements have to be specified. This can be one or more of the security principles: confidentiality, integrity and availability.

   - **Documents**: Confidentiality, Integrity, Availability.
     The most important asset of the project is a secret document. All aspects of security have to hold here. The secret documents have to be handled with confidence and must always be integer and available without undue delay.
   - **Internet/Network connectivity**: Availability, Confidentiality.
     Network connectivity has to be available to send and receive documents. Because the internet is known to be insecure, some solution has to be found to make it more confident.
   - **Workstation PCs**: Confidentiality, Availability, Integrity.
     PCs have to be available to work with the documents, and they have to do their job properly.
   - **Workstation Software**: Integrity, Availability.
     Software has to be authentic when it is used. If it has been tampered with, it cannot be trusted anymore. For people to work with software, it has to be available. For example, availability of software can be violated by software license issues or file corruption.

4. **Capture knowledge of protection strategy practices and organizational vulnerabilities:**
   This activity defines the current practices which are used to protect the assets.

   - Users are trained to work properly and follow (existing) security policies.
   - Network of workstation PCs is guarded by a firewall.
   - Network infrastructure and workstation PCs are all up to date.

**Process 2: Create threat profiles**

1. **Group assets, security requirements, and areas of concern by organizational level:**
   Process 2 is used to organize the gathered information. The table below is the result of this process. Each of the found assets is listed with the corresponding priority, areas of concern and security requirements.
<table>
<thead>
<tr>
<th>Asset</th>
<th>Priority</th>
<th>Area of concern</th>
<th>Security requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents</td>
<td>High</td>
<td>• Documents being viewed by unauthorized people.</td>
<td>C, I, A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Documents being changed by unauthorized people.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Documents being deleted by unauthorized people.</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>Medium</td>
<td>• Denial of access to the network or internet.</td>
<td>C, A</td>
</tr>
<tr>
<td>Workstation PCs</td>
<td>High</td>
<td>• Accessing workstation PCs by unauthorized people (intruder).</td>
<td>C, I, A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Denial of access to workstation PCs because of hardware failure.</td>
<td></td>
</tr>
<tr>
<td>Workstation software</td>
<td>Medium</td>
<td>• Denial of access to workstation PCs because of software problems.</td>
<td>I, A</td>
</tr>
</tbody>
</table>

2. **Select critical assets:**
   Critical assets are assets which are most important in the project. In this case the most important assets are the secret documents, which have a high priority and all security principles (confidentiality, integrity, availability) as security requirement.

3. **Refine security requirements for critical assets:**
   This step is skipped because there are no different data from different levels of the organization. This is the result of combining processes 1 to 3 of the original OCTAVE method into one single process.

4. **Identify threats to critical assets:**
   This step is skipped at this time, because it will be executed by a technique which is used to extend the OCTAVE method. More on this subject can be found in Chapter 5.
4.2.4 Phase 2: Identify Infrastructure Vulnerabilities

The result of the first phase is structured information about the operational requirements of the problem. The next step is to determine information technology requirements. This is done in two processes.

Process 3: Identify key components

1. **Identify system of interest:**
   The system of interest is the system which is most closely related to the critical asset and gives an intruder and normal users access to this asset.
   For each of the critical assets, we determine the system of interest:
   - **Workstation PCs:** The Workstation PCs are system assets and thus are the system of interest.
   - **Documents:** The documents can be accessed via the Workstation PCs and through the network infrastructure.

2. **Identify key classes of components:**
   The systems of interest can be broken down into components or can be components themselves. The components are classified into classes. These classes of components are:
   - Servers
   - Networking components
   - Security components
   - Desktop workstations
   - Home computers
   - Laptops
   - Storage devices
   - Wireless components
   - Other

   The key classes are desktop workstations and networking components.

3. **Identify infrastructure components to examine:**
   The most important components which have to be examined are the Workstation PCs, which have to be checked for technology vulnerabilities.
Process 4: Evaluate selected components

1. Run vulnerability evaluation tools on selected infrastructure components:

To examine the security of the selected infrastructure components, a few approaches are possible:

(a) Check against catalogs of vulnerabilities:

- US-CERT Vulnerability Notes Database\(^1\)
- Common Vulnerabilities and Exposures (CVE)\(^2\)
- SANS Top 20 Internet Security Vulnerabilities:\(^3\)

<table>
<thead>
<tr>
<th>Top Vulnerabilities to Windows</th>
<th>Top Vulnerabilities to Unix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Internet Information Services (IIS)</td>
<td>BIND Domain Name System</td>
</tr>
<tr>
<td>2. Microsoft SQL Server (MSSQL)</td>
<td>Remote Procedure Calls (RPC)</td>
</tr>
<tr>
<td>3. Windows Authentication</td>
<td>Apache Web Server</td>
</tr>
<tr>
<td>4. Internet Explorer (IE)</td>
<td>General UNIX Authentication Accounts with No Passwords or Weak Passwords</td>
</tr>
<tr>
<td>5. Windows Remote Access Services</td>
<td>Clear Text Services</td>
</tr>
<tr>
<td>6. Microsoft Data Access Components (MDAC)</td>
<td>Sendmail</td>
</tr>
<tr>
<td>7. Windows Scripting Host (WSH)</td>
<td>Simple Network Management Protocol (SNMP)</td>
</tr>
<tr>
<td>8. Microsoft Outlook and Outlook Express</td>
<td>Secure Shell (SSH)</td>
</tr>
<tr>
<td>10. Simple Network Management Protocol (SNMP)</td>
<td>Open Secure Sockets Layer (SSL)</td>
</tr>
</tbody>
</table>

(b) Follow security checklists:

- Defense Information Systems Agency (DISA) Windows 2000 Checklist\(^4\)
- Defense Information Systems Agency (DISA) Windows XP Checklist\(^5\)

(c) Run security tools:

- Microsoft Baseline Security Analyzer\(^6\)

\(^1\)http://www.kb.cert.org/vuls
\(^2\)http://www.cve.mitre.org
\(^3\)http://www.sans.org/top20
\(^4\)http://www.csrc.nist.gov/pcig/CHECKLISTS/win2k-checklist-022704.zip
\(^5\)http://www.csrc.nist.gov/pcig/CHECKLISTS/winxp-checklist-022704.zip
\(^6\)http://www.microsoft.com/technet/security/tools/mbsahome.mspx
4.2.5 Phase 3: Develop security strategy and plans

The last phase of the original OCTAVE method is not executed in this project. First we extend the risk analysis using a technique called ‘attack trees’. A attack tree software tool is used to aid in this. The result is a number of solution approaches for the project which will be discussed in Chapter 6.
Chapter 5

Attack trees

5.1 Introduction

An attack tree is a structured method of listing attacks and counter measures of a situation. This will be explained via an example\(^1\) attack tree which is depicted in Figure 5.1.

The graphical representation of an attack tree can grow very wide if it contains more levels. Therefore a text version of an attack tree can often be more clear for attack trees with many items. A textual representation of the example attack tree can be found in Section 5.3.

In Figure 5.1, can be seen that the attack tree contains three distinctive types of items: a root node at the top of the attack tree, nodes and leaves. For each of these three types of items, a description will be given.

5.1.1 Items of an attack tree

Root node

The root node describes the goal of the attacker. It is the base of the tree to which the rest of the tree is connected. An attack tree can only contain one root node.

The root node has a type which defines how its attack can be executed. This type can be an OR-type or AND-type. If the root node has the OR-type, then only one of the attacks of child-nodes need to be executed to reach its goal. But when the root node has the AND-type, then all of the attacks of its child-nodes need to be executed to reach its goal.

\(^1\)Taken from: http://www.schneier.com/paper-attacktrees-ddj-ft.html
The root node is connected to zero or more nodes and leaves. Each node represents a more detailed attack than the attack of its parent node. The nodes provide the level of detail of the attacks of the attack tree. Like the root node, a node has also an AND-type or OR-type, which defines in which way it depends on its child attacks.

### Leaves

A leaf represents an atomic attack which is not further split into more detailed attacks. A leaf can contain attributes which can be seen as a property of the corresponding attack. An attribute has a value of a certain type. Normally, all attributes in an attack tree are of the same value type.

#### 5.1.2 Attribute types

There can be different value types and an attribute of a certain type defines its value in a context. Examples of attributes:

- **Boolean type:**
  - Possible / Impossible
  - Easy / Not easy
– Expensive / Not expensive
– Intrusive / Non-intrusive
– Legal / Illegal
– Special equipment required / No special equipment required

• Continuous type:
  – Cost (money)
  – Cost (resources)
  – Time needed for the attack

• Fuzzy type:
  – Impact of an attack
  – Possibility of success of the attack
  – Possibility that the attack occurs

These three attribute types are supported by the developed attack tree tool. A detailed description of this can be found in Section 7.3.

The attributes can be used to draw different conclusions from the attack tree. For example, if a path from the root to a certain leaf exists in which all involved attributes have the value ‘possible’, then the attacker can reach its goal.

Because attributes can be combined, it is possible to look into the attack tree to find attacks which satisfy complex conditions. For example: give the cheapest attack which is possible and requires no special equipment.

Because of the structured view of attacks, attack trees can be used to determine and view the vulnerabilities of a system in a structured way. The use of attributes can determine the impact of a modification in the system and the effect of counter measures. Attacks can be compared and ordered.

If the number of attacks or the level of detail increases, the complexity of the structure of the attack tree can grow very large. To handle this complexity, certain improvements to the standard attack trees are needed.

5.2 Improvements

5.2.1 Subgoals with procedures

Simplification to the structure of an attack tree can be done by introducing subgoals. A subgoal is a part of the structure of an attack tree.
If there are several appearances of the same structure in an attack tree, it is useful to create a subgoal for it. Each of the appearances is substituted with a subgoal reference. This means that a new type node is introduced which refers to the subgoal.

The subgoal tree forms a new attack tree. The attack tree in combination with the attack trees the subgoals refer to are called a forest.

Besides reducing the size of the structure of the attack tree, subgoals are also very useful for re-using attack trees. If an information system contains a certain object, then it is very easy to include the attack tree of attacks on that object in the attack tree of the information system. Of course, this can only be done if the attack tree of the object is available. For this, a library with general attack trees can be built and extended with new attack trees.

This extension is supported by the attack tree tool.

### 5.2.2 Reduction after attribute substitution

In the following example, we use an attack tree in which all leaves have exactly one attribute with a boolean value. The meaning of this value is: possible/impossible.

The value of a node can be calculated via a recursive function which calculates the value depending on the value of its children. Because in this example the attributes have boolean values, the calculation is done via boolean logic (AND/OR calculation rules). In general, every context has its own set of calculation rules which can be used. More on this can be found in Section 7.3.

In this way, the value of all nodes including the root node can now be calculated resulting in an attack tree in which all items have a certain value.

A different ‘situation’ of the attack tree can be realised by changing the values of attributes. If the attribute of a leave is set to false, then the attack cannot be executed anymore because it has the value ‘impossible’. After this, the calculation process can be done again to update the whole attack tree with new values.

However, some changes of values can lead to a different structure of the attack tree if the tree is being reduced after the substitution of the value. This is done via reduction rules:

- **AND-node:**
  - If a child object evaluates to true, then that child can be deleted
  - If a child object evaluates to false, then the parent object can be substituted with a new leaf with value false

- **OR-node:**
  - If a child object evaluates to false, then that child can be deleted
– If a child object evaluates to true, then the parent object can be substituted with a new leaf with value true

This extension to attack trees is not supported by the attack tree tool.

5.3 Examples of attack trees

5.3.1 Simple example

This example is the textual representation of the attack tree ‘Open Safe’ as described in Section 5.1.

**Goal:** Open Safe. (OR)

1. Pick Lock.
2. Learn Combo. (OR)
   2.1. Find Written Combo.
   2.2. Get Combo From Target. (OR)
      2.2.1. Threaten.
      2.2.2. Blackmail.
   2.2.3. Eavesdrop. (AND)
      2.2.3.1. Listen to Conversation.
      2.2.3.2. Get Target To State Combo.
   2.2.4. Bribe.
3. Cut Open Safe.
4. Install Improperly.

5.3.2 Example with subgoals

This example is the textual representation of the attack tree ‘Document being viewed by unauthorized people’. The results of the risk analysis of Section 4.2.2 have been used to create this attack tree. It uses subgoals to re-use groups of attacks.

This example is used to reason about possible solutions for the problem described in Chapter 1. These solutions can be found in Chapter 6.
**Goal:** Document being viewed by unauthorized people. (OR)
- 1. **Subgoal:** D. Convince user to reveal document.
- 2. Read document when it is being entered into the computer. (OR)
  - 2.2. Read document when it is typed. (OR)
    - 2.2.1. Use keyboard-logging software to ‘record’ the document. (AND)
      - 2.2.1.1. **Subgoal:** A. Gain access to computer.
      - 2.2.1.2. Install keyboard-logging software.
    - 2.2.2. Visually monitor keyboard for keypresses.
      - 2.2.2.1. Watch user type (shoulder surfing).
      - 2.2.2.2. Install videocamera.
- 3. Read document when it is being stored on disk. (OR)
  - 3.1. Read document directly from hard drive. (AND)
    - 3.1.1. **Subgoal:** A. Gain access to computer.
    - 3.1.2. Get direct access to hard drive.
  - 3.2. Implant virus/trojan to expose document. (OR)
    - 3.2.1. Infect software which will be installed in the future.
    - 3.2.2. Install trojan on computer. (AND)
      - 3.2.2.1. **Subgoal:** A. Gain access to computer.
      - 3.2.2.2. Install trojan software.
    - 3.2.3. **Subgoal:** D. Convince user to install trojan.
- 4. Capture document when it is being sent from sender to recipient. (OR)
  - 4.1. Use network weaknesses to capture document. (OR)
    - 4.1.1. Abuse of routing tables to send the document to the attacker.
    - 4.1.2. **Subgoal:** E. Sniff document.
- 5. Read document when it is being displayed. (OR)
  - 5.1. Use electromagnetic snooping techniques.
  - 5.2. Visually monitor computer screen.

**Subgoal:** A. Gain access to computer. (OR)
- A.1. **Subgoal:** B. Gain local access to computer.
- A.2. **Subgoal:** C. Gain remote access to computer.

**Subgoal:** B. Gain local access to computer. (AND)
- B.1. Get physical access to location.
- B.2. Get physical access to computer.
- B.3. Gain access to computer. (OR)
  - B.3.1. Guess password.
  - B.3.2. **Subgoal:** D. Convince user to reveal password.
**Subgoal**: C. Gain remote access to computer. (OR)
  - C.1. Remotely login to computer. (OR)
    - C.1.1. Guess username and password.
    - **C.1.2. Subgoal**: E. Sniff username and password.
    - C.1.3. Hijack existing session.
  - C.2. Exploit network based application/protocol. (OR)
    - C.2.1. Exploit network daemon.
    - C.2.2. Exploit userland software.
    - C.2.3. Exploit OS level networking.

**Subgoal**: D. Convince user to... (OR)
  - D.1. Bribe user.
  - D.2. Blackmail user.
  - D.3. Threaten user.
  - D.4. Fool user.

**Subgoal**: E. Sniff message. (OR)
  - E.1. Gain local network access to the network segment. (AND)
    - E.1.2. Install sniffing software at server.
  - E.2. Tap physical network medium.
  - E.3. Redirect traffic through a compromised host.

### 5.4 Available tools

#### 5.4.1 SecurITree

SecurITree is a commercial tool developed by Avenaza\(^2\). An attack tree can be built with AND-nodes and OR-nodes. Items of the attack tree can contain attributes. A function is used to calculate the value of a node out of the values of its children. This function can be specified by the user.

#### 5.4.2 TANAT

TANAT\(^3\) (Threat ANd Attack Tree Modeling plus Simulation) is an open source tool for attack tree modelling. It can give a graphical representation of attack trees and has an export function to other data formats. It supports attributes. Unfortunately, it was not completely working at the time of this project.

\(^2\)http://www.amenaza.com
\(^3\)http://www13.informatik.tu-muenchen.de:8080/tanat
5.4.3 Conclusion

There are not very many tools available for creating attack trees, which provide the functionality that is needed for this project. Therefore, an attack tree tool with special requirements has been developed as part of this project. The requirements for the attack tree tool are listed in Chapter 8.

The attack tree tool is used to analyze the problem of secure document handling and reason about possible solutions for it. It has been developed using a software engineering documentation standard. The description of the development process of this attack tree tool starts at Chapter 7.
Chapter 6

Solutions

6.1 Introduction

To investigate possible solution strategies, we use the attack tree ‘Document being viewed by unauthorized people’ from Chapter 5. In this attack tree, all leaves have a fuzzy attribute ‘Degree of concern’ which describes how much attention the attack requires. When an attribute has the lowest possible value in the fuzzy domain ‘Very Low’, this means that it does not need attention anymore.

Because it is easier to work with only one attack tree and thus one file, we expand the attack tree by substituting each subgoal reference by its corresponding attack tree. The result is a fully expanded version with repeating blocks of attacks.

To illustrate how the attack tree tool is used to discuss the solution strategies, screenshots will be used. Because the fully expanded attack tree of the problem is very large, some irrelevant parts of the attack tree have been collapsed in the screenshots. For the begin situation, this is depicted in Figure 6.1.

We use an incremental approach for describing solutions. This means that first a solution proposal is done. This is a description of how attacks in the attack tree can be blocked. From the behaviour of the two node types, it follows how attacks can be blocked. In case of an AND-node, only one sub-attack has to be blocked to disable the node and in case of an OR-node, all sub-attacks have to be blocked.

The solution proposals have a number of approaches. Each of the approaches of the solution strategies has an ‘attribute profile’, which describes how attribute values in the attack tree are changed. After the values have been changed in the attack tree tool, the attack tree is updated and the tool shows the resulting situation. Each step of applying solutions and updating the attack tree results in the reduction of amount of problems.

For solving the most problems, we take the approach which blocks the most attacks. This does not need to be the best solution though, because it does not take other variables of
Figure 6.1: Starting situation
the situation into account. For example, the functionality of the office environment can be strongly decreased until it has no use anymore. In practise, we aim at the best balance between security and all other variables, like costs of the solution, costs of impact of an attack and remaining functionality.

6.2 Software policy

The first solution strategy describes how the operating system and additional software is being handled. For this, different approaches are possible which are listed below. Each of these approaches is listed with a description and an attribute profile.

1. Software installation policy; user has restricted software installation rights:
   Becase the user has limited rights for installing software, the chance for executing malicious code is reduced, but still exists. If less software is installed, the chance that an attacker uses a software exploit to reach its goal is also reduced.
   
   **Attribute profile:** Value of items concerning trojans/viruses is reduced. Value of items concerning software exploits is reduced.

2. Software installation policy; user has no software installation rights:
   In this case, the user has no rights for installing software. This means that the IT-staff decides which software is installed on the workstation PCs. If this is done properly, we do not have to worry about malicious software anymore.
   
   **Attribute profile:** Value of items concerning trojans/viruses becomes Very Low. Value of items concerning software exploits is reduced.

3. Only trusted software can be executed (e.g. via software signing):
   To have total control of what code may be executed, a technique called software signing can be used. For each program, a signature is calculated, and only software with a known trusted signature can be executed. This solves the problem of trojans and viruses. Software exploits are harder to execute, but are still possible.
   
   **Attribute profile:** Value of items concerning trojans/viruses becomes Very Low. Value of items concerning software exploits is reduced.

4. Use strict operating system update policy:
   When the operating system is regularly updated, chances of an attacker breaking in via an exploit in the operating system get reduced. The IT-staff has to define an operating system update policy and follow it strictly.
   
   **Attribute profile:** Value of items concerning exploiting parts of the operating system is reduced.
Figure 6.2: Changes of software policy
For this solution, we use approach 3. The result of applying the corresponding attribute profile to the attack tree is depicted in Figure 6.2.

6.3 Network connectivity

Network connectivity defines how the attacker can communicate with the workstation PC. We distinguish two types of network connection, local and remote network. Communication via local network can only be done if the attacker has access to the local area network. Remote network means that the communication is done via an Internet connection.

1. **No connection**: In this case, the workstation PC has no network connection at all. There is no network communication from the local area network possible and the workstation is not reachable from the Internet. This solves all problems which require local or remote network.

   **Attribute profile**: Value of items requiring local network becomes Very Low. Value of items requiring remote network become Very Low.

2. **Only connection with Local Area Network**: In the case of only a network connection to the local area network, the workstation PC is not reachable from the Internet. This solves all problems which require remote netork.

   **Attribute profile**: Value of items requiring remote network becomes Very Low.

3. **Router/firewall filters the connection with Internet**: The chance that an attacker breaks in into the workstation PC is reduced when a firewall filters the network communication. However, there is still a chance that a break-in attempt succeeds.

   **Attribute profile**: Value of items requiring remote network is reduced.

For this solution, we use approach 1. The result of applying the corresponding attribute profile to the attack tree is depicted in Figure 6.3.

6.4 Physical access

The attacker can use physical access to the workstation PC, to directly view or retrieve the secret document. For this method, no communication is required. Physical access can be solved by putting the workstation PC into a separate guarded office.

1. **No physical access to workstation PC**: When no physical access to the workstation PC is possible, the attacker cannot get local access anymore. The result is that attacks which require physical access to the workstation PC are blocked.
Figure 6.3: Changes of network connectivity
Figure 6.4: Changes of physical access
Attribute profile: Value of items requiring physical access to the workstation PC becomes Very Low.

2. No physical access to total environment: In addition to the previous solution, if an attacker is blocked from the total environment, all attacks requiring physical access are blocked.

Attribute profile: Value of items requiring physical access becomes Very Low.

For this solution, we use approach 2. The result of applying the corresponding attribute profile to the attack tree is depicted in Figure 6.4.

6.5 Remaining problems

As can be seen in Figure 6.5, there is one remaining problem. The user can reveal the document. It is difficult to solve this problem because there is a difference between controlling information systems and controlling people. However, people can be trained how to securely handle secret documents if they know what the methods of an attacker are.

For example, an attacker can use ‘social engineering’ to fool the user and retrieve the document. If the user is well informed, he can take into account that this can happen and be distrustful about any form of communication with external parties.

6.6 Conclusion

We have used the attack tree tool to reason about possible solution strategies and combined these strategies to reduce the number of problems. Because the attack tree tool provides a structured view of the situation, it is easier to see what the problems are and which problems need more attention.

Of course, the attack tree tool cannot solve problems. You still have to search for solutions yourself. But it is easier to see how an attack can be blocked or what the result of a solution is.

For the problem of digital document handling, the conclusion is that the problem can be solved, but every aspect of the problem has to be taken into account. The result is, that the user and the workstation pc are physically and digitally seperated from the original office environment.

As said before, in practise this would not be an acceptable solution, because the functionality of the environment is heavily decreased. Therefore, it is desirable to find a balance between functionality and security. For the problem of digital document handling, this is very hard.
Figure 6.5: Remaining problem
Chapter 7

KATT

7.1 Introduction

7.1.1 Purpose

The next chapters describe the software development trajectory of KATT, the KATI Attack Tree Tool. It consists of the specification of requirements, a design and general guidelines. The requirements specify the demands of the customer with respect to KATT. In a later stage these chapters will be used as a reference for verification. It is a contract between the customer and the developer. It is to be read by all active participants in this project, namely: initiators, the customer, user representatives, project managers, the author and reviewers of this document.

7.1.2 Scope

The function of KATT is to offer a way of manipulating attack trees, which can be changing the structure of the attack tree or changing values of items.

The user can add, edit and remove attack trees or parts of it. A part of an attack tree is a node (or leaf, which is a node without children), edges between nodes and attributes.

7.1.3 Overview

In Chapter 7 the environment of the tool is discussed and the important requirements and constraints on the development of the tool are specified.

Chapter 8 contains the specific capability and constraint (user) requirements, as well as the functional and non-functional (software) requirements. These requirements are given with their corresponding priorities and were defined according to the directions of
Chapter 9 gives a breakdown of the tool into components with a detailed description of each component. This chapter can be used for reference in future development.

The tenth and last chapter describes a number of general project management topics, which can be seen as global guidelines for the project.

7.1.4 List of definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>The customer in this project, Bert Bos from Chess iT.</td>
</tr>
<tr>
<td>end-user</td>
<td>User of KATT.</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency.</td>
</tr>
<tr>
<td>KATT</td>
<td>KATI Attack Tree Tool.</td>
</tr>
<tr>
<td>KATI</td>
<td>KATT Alternating Tree Items.</td>
</tr>
<tr>
<td>PD</td>
<td>Project document.</td>
</tr>
<tr>
<td>project manager</td>
<td>Leon Schrijvers, Eindhoven University of Technology.</td>
</tr>
<tr>
<td>senior project manager</td>
<td>Sjouke Mauw, Eindhoven University of Technology.</td>
</tr>
<tr>
<td>SR</td>
<td>Software requirements.</td>
</tr>
<tr>
<td>tool</td>
<td>KATT.</td>
</tr>
<tr>
<td>tree document</td>
<td>File which contains a description of an attack tree.</td>
</tr>
<tr>
<td>UR</td>
<td>User requirements.</td>
</tr>
</tbody>
</table>

7.1.5 List of references


7.2 General description

7.2.1 Product perspective

KATT is an application for manipulating attack trees. This can be done by performing operations on the whole attack tree or parts of it.

The end-user controls the software via the User Interface (UI) component. This is a part of the frontend. An extensive description of this is given in Chapter 9. The frontend internally communicates with the backend which contains the actual data of the attack
An attack tree is a structure of nodes, edges and attributes. Edges are used to connect the nodes of the tree. A leaf is a special type of node which has no children. A leaf can contain one or more attributes, which have a name and a value. This can be seen in Figure 7.2.

Because an attack tree can be broken down into objects, manipulating an attack tree means performing actions on those objects. In the table below, a list of objects with their corresponding actions is given.
Figure 7.2: Tree breakdown into objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>Load</td>
</tr>
<tr>
<td></td>
<td>Save</td>
</tr>
<tr>
<td></td>
<td>Export</td>
</tr>
<tr>
<td></td>
<td>Print</td>
</tr>
<tr>
<td></td>
<td>Select</td>
</tr>
<tr>
<td></td>
<td>Create</td>
</tr>
<tr>
<td></td>
<td>Remove</td>
</tr>
<tr>
<td></td>
<td>Rename</td>
</tr>
<tr>
<td></td>
<td>Cut</td>
</tr>
<tr>
<td></td>
<td>Copy</td>
</tr>
<tr>
<td></td>
<td>Paste</td>
</tr>
<tr>
<td></td>
<td>Reduce</td>
</tr>
<tr>
<td></td>
<td>Create node/leaf/subgoal</td>
</tr>
<tr>
<td>Node</td>
<td>Select</td>
</tr>
<tr>
<td></td>
<td>Remove</td>
</tr>
<tr>
<td></td>
<td>Rename</td>
</tr>
<tr>
<td></td>
<td>Cut</td>
</tr>
<tr>
<td></td>
<td>Copy</td>
</tr>
<tr>
<td></td>
<td>Paste</td>
</tr>
<tr>
<td></td>
<td>Create node/leaf/subgoal</td>
</tr>
<tr>
<td>Attribute</td>
<td>Reset</td>
</tr>
<tr>
<td></td>
<td>Rename</td>
</tr>
<tr>
<td></td>
<td>Change value</td>
</tr>
</tbody>
</table>
For each of the actions in the table, a short description will be given.

- **Load tree.**
  A tree document of the TML file format can be loaded from disk.

- **Save tree.**
  A tree document of the TML file format can be saved to disk.

- **Export tree.**
  A tree can be exported to disk. The possible file format which can be used is plain text.

- **Print tree.**
  The current tree is printed.

- **Select tree.**
  This selects the current tree. After this, other actions can be performed on the tree.

- **Create tree.**
  In an empty tree document, a new tree is created.

- **Remove tree.**
  The current tree is deleted. The result is an empty tree document.

- **Rename tree.**
  The name of the goal of the attack tree can be renamed via this action.

- **Cut tree.**
  This places a copy of the current attack tree into the clipboard and deletes the tree from the tree document.

- **Copy tree.**
  This places a copy of the current attack tree into the clipboard.

- **Paste tree.**
  An attack tree can be pasted into an empty tree document.

- **Reduce tree.**
  This action reduces the attack tree via the current valid computation rules.

- **Create node/leaf/subgoal of tree.**
  This creates a new node/leaf/subgoal.

- **Select node.**
  This selects a node. After this, other actions can be performed on this node.

- **Create node/leaf/subgoal.**
  A new node/leaf/subgoal is created.
• **Remove node.**
  The selected node is removed from the attack tree.

• **Rename node.**
  The selected node can be renamed via this action.

• **Cut node.**
  This places a copy of the selected node with its children into the clipboard and deletes the node from the attack tree.

• **Copy node.**
  This places a copy of the selected node with its children into the clipboard.

• **Paste node.**
  A node and its children can be pasted into the tree.

• **Reset attribute.**
  The attribute is resetted to a default name/value pair.

• **Rename attribute.**
  The name of the attribute can be renamed via this action.

• **Change value of attribute.**
  The value of the attribute can be change via this action.

### 7.2.3 General constraints

The project management has made it obligatory that

- development will be done according to [ESA],
- the software must be developed according to the object-oriented paradigm.

Development will be done in such a way that the customer and senior project management are able to maintain the software. This makes further development possible. For this reason, the complete source code and documentation of the software will be delivered to the customer and senior project management at the end of the project.

For maintainability, all of the comments and naming in the source code, the user interface and the product documents are in English.

The programming language used to develop the software must be one of the languages which supports the object-oriented paradigm and will be one which the developer is familiar with.
7.2.4 Model description

The logical model is constructed using the Object Modelling Technique (OMT) for the Software Requirements phase as described on [SE1]. This technique uses the Unified Modelling Language (UML) to describe its models. The high level model was constructed using elements from the UML but does not adhere to UML constraints. It is intended to provide an overview of the largest components and the relations between them. The logical model itself adheres to the UML fully.

7.3 Calculation rules

7.3.1 Attack tree structure

In an attack tree, a leaf can have an attribute with a value. This value can be used to perform calculations within the attack tree. These calculations have to follow certain rules, which are called calculation rules.

A leaf item in an attack tree always has a parent node. A node is an object which can contain children objects. In this case, the leaf is one of the children of the node. Because a node itself does not have a direct value, the value is calculated from the values of its children.

There are two types of nodes, an AND-node and an OR-node. These nodes can have a variable number of children, thus the node type can be seen as a function with a variable number of parameters.

7.3.2 Input domains

The root node of an attack tree defines the type of attribute values. This type is called an input domain. So if an attribute value is assigned, it has to match the current input domain.

In this project, we distinguish three types of input domains. These are: boolean values, fuzzy values and integer values. A calculation rule can now be seen as a mapping of a node type to a function with respect to the input domain and context. In the next section, for each of the input domains a description of the context and corresponding calculation rules are given.

7.3.3 Rules

To understand what a value means, a context for the corresponding domain has to be defined. After this, it is clear how the functions can be used to calculate values.
**Boolean**

Context: Is an attack possible?
Values of domain: True/False.

- $AND(a, b, ..., n) := a \land b \land ... \land n$
- $OR(a, b, ..., n) := a \lor b \lor ... \lor n$

For boolean values, the defined functions are the same as the node types. Thus an AND-node uses a boolean and ($\land$) function and an OR-node uses a boolean or ($\lor$) function.

**Fuzzy**

Context: Possibility that an attack is performed.
Values of domain: Very Low/Low/Medium/High/Very High.

- $AND(a, b, ..., n) := \text{min}(a, b, ..., n)$
- $OR(a, b, ..., n) := \text{max}(a, b, ..., n)$

For fuzzy values, an AND-node is mapped to the $\text{min}()$ function. In the defined context (possibility), this is because the possibility that more than one (sub-)attacks are performed is always less than the possibility that one single attack is performed.

The OR-node uses the $\text{max}()$ function which gives the value of the attack with the highest possibility. This can be seen as the ‘worst case’ scenario.

**Integer**

Context: Costs of performing an attack.
Values of domain: Integer values.

- $AND(a, b, ..., n) := a + b + ... + n$
- $OR(a, b, ..., n) := \text{min}(a, b, ..., n)$

In the case of integer values, the AND-node uses the sum ($+$) function, because for achieving a goal, all sub-attacks have to be performed. So the total cost is the sum of the costs of all sub-attacks.

For the OR-node, the result is the sub-attack with the least cost. This is also a ‘worst case’ scenario, when the attacker will choose for the cheapest attack.
Chapter 8

Specific requirements

8.1 Introduction

In this chapter all requirements of and constraints on KATT are listed. The software will adhere to these requirements. A priority (denoted with ‘Pri’ in the table headers below) has been given to each requirement to indicate its importance. The priorities indicate the order in which the requirements will be fulfilled. In case of lack of time, requirements may be skipped after consulting the client. A ‘1’ indicates the highest priority, ‘3’ the lowest. Priority 1 means that the requirement is essential, ‘3’ means that the requirement is ‘nice to have’.

Every requirement has a source (denoted with ‘Src’ in the table headers below). This is the name of a document, person, or group from where the requirement originated.

BEBO Bert Bos at Chess iT, Best
LESC Leon Schrijvers at Eindhoven University of Technology

8.2 Capability requirements

KATT provides functionality for manipulating an attack tree. An attack tree is a hierarchical structure of nodes (connected through edges) and attributes. A leaf (a node without children) can contain an attributes. Each of these structure types have specific requirements which are discussed below.
### Tree related requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCARA00</td>
<td>A tree document can contain zero or one attack tree.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARA01</td>
<td>A tree has zero or one root node.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARA02</td>
<td>Only one tree item object can be selected at at time.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARA03</td>
<td>A new tree can be created.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARA04</td>
<td>An existing tree can be deleted.</td>
<td>2</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARA05</td>
<td>The root node of a tree contains a impact property.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARA06</td>
<td>The root node of a tree contains a reference to a ‘colorfile’ which is a mapping of attribute values to colors.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARA07</td>
<td>The root node of a tree contains a attribute type property, which defines the type of the values of attributes in the tree.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARA08</td>
<td>The value of a root node is calculated via the values of its children via calculation rules specified in Section 7.3.</td>
<td>2</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARA09</td>
<td>An attack tree can be automatically reduced via calculation rules specified in Section 7.3.</td>
<td>3</td>
<td>BEBO</td>
</tr>
</tbody>
</table>
### Node related requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCARB00</td>
<td>A node has a title.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB01</td>
<td>A node has a type.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB02</td>
<td>A node has a parent object.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARB03</td>
<td>A node can be a normal node, a leaf (node without children) or a subgoal.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARB04</td>
<td>A leaf has one attribute.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB05</td>
<td>A subgoal has a reference to another attack tree (file).</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB06</td>
<td>A subgoal has a title of the attack tree it references to.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB07</td>
<td>The title of a node (root, node, leaf, subgoal) can be changed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB08</td>
<td>The type of a node (root, node) can be changed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB09</td>
<td>The reference of a subgoal can be changed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB10</td>
<td>The title of the reference of a subgoal can be changed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB11</td>
<td>New nodes can be created.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB12</td>
<td>An existing node (root, node, leaf, subgoal) can be deleted.</td>
<td>2</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB13</td>
<td>The value of a node is calculated via the values of its children via calculation rules specified in Section 7.3.</td>
<td>2</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARB14</td>
<td>The value of a leaf is the value of the attribute which the leaf contains.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARB15</td>
<td>The value of a subgoal node is equivalent to the value of the root node of the attack tree it references to.</td>
<td>2</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARB16</td>
<td>A subgoal node cannot reference to itself or another file which results in a cycle of references.</td>
<td>1</td>
<td>LESC</td>
</tr>
</tbody>
</table>

### Attribute related requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCARCC0</td>
<td>An attribute has a name.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARCC1</td>
<td>An attribute has a value.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARCC2</td>
<td>The value of the attribute is of the same type as the attribute type of the attack tree.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARCC3</td>
<td>The name of an attribute can be changed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARCC4</td>
<td>The value of an attribute can be changed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
</tbody>
</table>

---

50
User interface related requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCARD00</td>
<td>The application has a treeview component in which the attack tree is displayed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARD01</td>
<td>The application has a root editor component in which the properties of the root node can be changed.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARD02</td>
<td>The application has a node editor component in which the properties of a node can be changed.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARD03</td>
<td>The application has a leaf editor component in which the properties of a leaf can be changed.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARD04</td>
<td>The application has a subgoal editor component in which the properties of a subgoal can be changed.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCARD05</td>
<td>An item in the treeview component can contain an image which displays the value of the corresponding item.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARD06</td>
<td>General cut/copy/paste actions can be performed on all items of an attack tree.</td>
<td>3</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARD07</td>
<td>An existing tree document can be opened from disk.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARD08</td>
<td>A tree document can be saved to disk.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARD09</td>
<td>A tree document can be exported to disk.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCARD10</td>
<td>A tree document can be printed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
</tbody>
</table>

Runtime related requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCARE00</td>
<td>If and operation that is invoked by the user fails, the user must be notified.</td>
<td>1</td>
<td>LESC</td>
</tr>
</tbody>
</table>

8.3 Constraint requirements

Interface requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCORA00</td>
<td>KATT will be programmed in: Borland Delphi 7 (Object pascal).</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>UCORA01</td>
<td>End users must have a Windows operating system installed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCORA02</td>
<td>End users must have a plain text viewer installed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCORA03</td>
<td>Application installation must be done via a setup program.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>UCORA04</td>
<td>For application installation no administrator rights must be needed.</td>
<td>1</td>
<td>BEBO</td>
</tr>
</tbody>
</table>
Security requirements

Not applicable.

Quality requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCORB00</td>
<td>If an operation is performed the user will be notified about the progress.</td>
<td>3</td>
<td>BEBO</td>
</tr>
</tbody>
</table>

8.4 Non-functional requirements

8.4.1 Performance requirements

Not applicable.

8.4.2 Operational requirements

Prototype

The prototype is explained via usecases, which describe the usage of basic functionality. First is explained how to build a new attack tree from scratch. The second use case describes how the attribute type of an attack tree can be changed. The last use case describes the usage of subgoals with multiple instances of KATT. All use cases use the example attack tree ‘Open Safe’, which is described in Section 5.3.

Use case: Building a tree from scratch

This use case describes how the end-user can build an attack tree from scratch with KATT.
1. Start the tool and begin with an empty screen:

![Screenshot of an empty screen]

2. Create a new tree document via the menu option ‘File | New’:

![Screenshot of a tree document]

---

53
3. Click on the root node and change its properties using the ‘Root Editor’:

4. Click on the root node, and use the menu option ‘Insert | Leaf’ to add a new leaf item:
5. Click on the created leaf, and use the ‘Leaf Editor’ to change its properties:

6. Click on the root node, and use the menu option ‘Insert | Child Node’ to add a new node item:
7. Click on the created child node, and use the ‘Node Editor’ to change its properties:

8. Create two more leaf items so that the first level of depth is created:
9. Finish the attack tree via the ‘Insert’ menu and the editors:

10. Save the attack tree via the menu option ‘File | Save As...’ with the filename ‘opensafe.xml’:
Use case: Changing the attribute type

1. Open the previously created tree with the filename ‘opensafe.xml’ via the menu option ‘File | Open...’:

2. Click on the root node, and change the attribute type to ‘Fuzzy’ and the color file to a fuzzy color file (in this case ‘c:\tree\fuzzy.txt’) via the ‘Root Editor’:
3. Use the ‘Leaf Editor’ to change the values of leaves:

Use case: Using subgoals

1. Open the previously created tree with the filename ‘opensafe.xml’ via the menu option ‘File | Open...’: 
2. Open a new instance of KATT and create a new tree document via the menu option ‘File | New’, and change the properties of the root node via the ‘Root Editor’:

![Image of KATT root editor]

3. Click on the root node and add a new leaf item via the menu option ‘Insert | Leaf’, and change the properties of the created leaf item via the ‘Leaf Editor’:

![Image of KATT leaf editor]

4. Click on the root node and add a new subgoal item via the menu option ‘Insert | Subgoal’, and change the properties of the created subgoal item via the ‘Subgoal Editor’:

![Image of KATT subgoal editor]
5. Switch to the first instance of KATT and change the values of the leafs to False so the value of the root node will change. Save the attack tree via the menu option ‘File | Save’:

6. Switch back to the second instance of KATT and use the menu option ‘Edit | Refresh’ to refresh the treeview component. The value of the subgoal item has
8.4.3 Resource requirements

Not applicable.

8.4.4 Verification requirements

Not applicable.

8.4.5 Acceptance-testing requirements

Not applicable.

8.4.6 Portability requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPORA00</td>
<td>The application is designed for the Microsoft Windows operating system.</td>
<td>1</td>
<td>BEBO</td>
</tr>
</tbody>
</table>
8.4.7 Maintainability requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMARA00</td>
<td>The product is programmed using Borland Delphi 7.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>SMARA01</td>
<td>The design of the product is modular.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>SMARA02</td>
<td>The development is maintained via Concurrent Versioning System (CVS).</td>
<td>1</td>
<td>BEBO</td>
</tr>
</tbody>
</table>

8.4.8 Reliability requirements

Not applicable.

8.4.9 Security requirements

Not applicable.

8.4.10 Safety requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSARA00</td>
<td>Supported input document types are: Tree Modelling Language (TML) files.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>SSARA01</td>
<td>Supported output document types are: Tree Modelling Language (TML) and plain text files.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>SSARA02</td>
<td>The usage of documents of an supported type should not lead to unexpected behavior.</td>
<td>1</td>
<td>LESC</td>
</tr>
<tr>
<td>SSARA03</td>
<td>The usage of documents of an unsupported type may lead to unexpected behavior and is on the users’ own responsibility.</td>
<td>1</td>
<td>LESC</td>
</tr>
</tbody>
</table>

8.4.11 Other quality requirements

Not applicable.

8.4.12 Documentation requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement description</th>
<th>Pri</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDORA00</td>
<td>The thesis contains a chapter which describes the requirements phase of the project.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>SDORA01</td>
<td>The thesis contains a chapter which describes the software design and implementation phase of the project.</td>
<td>1</td>
<td>BEBO</td>
</tr>
<tr>
<td>SDORA02</td>
<td>The thesis contains a chapter which describes general topics about project management.</td>
<td>1</td>
<td>BEBO</td>
</tr>
</tbody>
</table>
Chapter 9

Detailed Design

9.1 High level model

Because KATT does not depend on external components, the high level model can be explained via the figure of the product perspective in Section 7.2.1. The user has an interactive relation with the tool via the User Interface (UI). This UI is part of the frontend, and is used to receive and respond to actions performed by the user. These actions are processed by the backend which changes the internal data. Communication between the frontend and backend is done via the TBase class which is the base of the program. This is depicted in Figure 9.1.

9.2 Components

9.2.1 Internal data

The KATT classes are divided into two parts. The first part is the internal data in which the attack tree is stored. This data structure is changed when the user performs actions on the attack tree. The internal data can be transformed to other formats which are called views. For example, a view can be an instance of the user interface part TUITreeView or an export to a text file.

The classes of internal data are depicted in Figure 9.2.1. A detailed description of each class can be found in Appendix A.

9.2.2 Frontend and backend

The second part of the KATT classes is the frontend and backend with TBase as controlling class. The frontend is the part which provides the user interface and functionality
which is used by the user interface. For example, the TFilter class is controlled by the user interface, to set a filter on the data represented by the user interface.

The classes of the frontend and backend are depicted in Figure 9.2.2. A detailed description of each class can be found in Appendix A.
Figure 9.2: Internal data
Figure 9.3: Frontend and backend
Chapter 10

Project Management

10.1 Design standards

The decomposition in software components of KATT is done in Chapter 9. The design closely follows guidelines of the Unified Modelling Language (UML).

10.2 Documentation standards

Source code is documented in this document. For each of the software components, a component description is given in Appendix A. A component description consists of a list of fields with type and a list of methods. A method is given without list of parameters or possible return type. Each item contains an explanation of how the item is being used in the component.

10.3 Naming conventions

All function names should start with a uppercase letter. If a function name contains more than one word then all words should start with an uppercase letter. In functions, parameter names always start with a capital ‘A’. For example:

\texttt{ExportTML(AFileName: TFileName; AString: string);}

Variable names start with a lowercase letter. If a variable name contains more than one word then all words, except for the first one, should start with an uppercase letter. Local variable names start with ‘my’ and only loop variables can have a single letter as name (mostly ‘i’).
Constants are named with all uppercase letters and start with a underscore (_).
Classes and other types start with a capital ‘T’. For example:

```pascal
TTMLInterface = class(TXMLDocument)
```

All commenting and naming is done in English.

## 10.4 Coding standards

A standard tabstop of two spaces is used. Braces (or begin/end blocks) does not require a new line, but is recommended when it increases readability of the source code.

## 10.5 Software development tools

All components of KATT are written using Borland Delphi 7. For source code control, TortoiseCVS 1.6.0 is used to connect to a CVS server.
Chapter 11

Conclusion

11.1 Introduction

In this project, we have analyzed the problem of digital document handling. A risk analysis was made to identify weak spots in the chain of operations on digital documents and attack trees were used to structure the attacks on these weak spots. To help evaluating the situation and possible solutions to the attacks, an attack tree tool was developed and used.

11.2 Attack tree tool

11.2.1 Results

The attack tree tool has proved itself being a valuable tool for analyzing security problems. It has been used to reason easily about the solution proposals and evaluate the situation. The attack tree tool gives a clear structured view of the problems along with the corresponding attribute values. Because of this, attacks which need attention can be found at a glance.

11.2.2 Future research

In the time that the attack tree tool was used, several issues came around. First, the input domains of the attribute values along with the corresponding calculation rules have been hardcoded in the tool. This was done because only the supported types were required and it saved time of the developing process. If the user can define its own custom attribute types and functions, it will be a valuable extension to the attack tree tool.
Another issue was, that there was a restriction in the number of attributes a leaf can contain. Again, the reason of this was for keeping the developing time of the attack tree tool short. It can also be an extension that a leaf can contain more than one attribute, which leads to the ability of using more complex calculation rules and filter expressions.

11.3 Conclusion

Creating an secure interface between the user and the workstation PC is very hard to do. As can be seen from the attack tree, there are various ways for an attacker to intercept the document when a normal workstation PC is used. Because no custom hardware could be used, other ways to secure the documents were needed.

It is hard to find a complete solution for the problem of digital document handling, because all aspects of security have to be accounted for. The combination of a number of extreme approaches of solution strategies lead to a great reduction of the number of problems. However, the result was that the functionality of the office environment was heavily decreased. In practise, it will be a matter of finding the right balance between security and functionality.
Chapter 12
Bibliography

[1] Redrawing the security perimeter of a trusted system
   Sterne, D.F.; Benson, G.S.; Tajalli, H.;
   June 1994, pp. 162-174

[2] Structuring trust in a large general purpose operating system
   Parker, T.A.;
   1988, pp. 152-158

[3] Building reliable secure computing systems out of unreliable insecure components
   Dobson, J.E.; Randell, B.;

[4] 20 years of operating systems security
   Gligor, V.D.;
   1999, pp. 108-110

[5] Some weaknesses of the TCB model
   Blakley, B.; Kienzle, D.M.;
   pp. 3-5

[6] On models for a trusted application system
   Payne, C.N.; Froscher, J.N.; McDermott, J.P.;
   Computer Security Applications Conference, 1990, Proceedings of the Sixth Annual,
   3-7 Dec. 1990, pp. 58-67

[7] Retrofitting and developing applications for a trusted computing base
   Gambel, D.; Walter, S.;
1988, pp. 212-215

[8] Is The Trusted Computing Base Concept Fundamentally Flawed?
McLean, J.;
pp. 2-2

[9] A multi-level secure message switch with minimal TCB: architectural outline and
security analysis
Lipper, E.H.; Melamed, B.; Morris, R.J.T.; Zave, P.;
1988, pp. 242-249

[10] Rehosting existing command and control systems into a multilevel secure environment
Doncaster, S.; Endsley, M.; Factor, G.;
Computer Security Applications Conference, 1990, Proceedings of the Sixth Annual,
3-7 Dec. 1990, pp. 434-445

Code
Catuogno, L.; Visconti, I.;
Dipartimento di Informatica ed Applicazioni, Università di Salerno

Department of Defense (DoD);
DoD 5200.28-STD, Dec. 1985

Alberts, C.J.; Dorofee, A.J.;
Carnegie Mellon University, 2001

Alberts, C.J.; Dorofee, A.J.;
Addison Wesley, 2003

Preneel, B.; Rijmen, V.;
Springer, 1998

[16] Computer Network Security
Gorodetsky, V.; Popyack, L.; Skornin, V.; Second International Workshop on
Mathematical Methods, Models, and Architectures for Computer Network Security

[17] Computer Security
Gollmann, D.; Wiley, 1999
Appendix A

Class descriptions

Internal data

TTree

Parent class:

• TObject

Fields:

• FRoot: TRoot
• FNodes: TObjectList
• FLastID: TID

A non-empty tree always contains a root and zero or more nodes. The root is stored into the FRoot field and FNodes contains a list of TSuperNode derivatives, which can be nodes, leaves or subgoals. The FLastID field stores the last used TreeID which is used to connect items within the tree object.

Methods:

• Create:
  General constructor.

• SetRoot(ARoot: TRoot):
  Property method for setting the FRoot field.

• GetRoot: TRoot:
  Property method for getting the FRoot field.
• **GetNode**(AID: TID): TSuperNode:
  Gets a node from FNodes by absolute index.

• **GetNodeByID**(AID: TID): TSuperNode:
  Gets the node from FNodes which is identified by the corresponding TID.

• **NodeCount**: integer:
  Returns the number of items of FNodes.

• **AddNode**(AID: TID; ATitle: TTitle; AParent: TParent; AType: TType):
  Adds a node to FNodes.

• **AddLeaf**(AID: TID; ATitle: TTitle; AParent: TParent; AAttribute: TAttribute):
  Adds a leaf to FNodes.

• **AddSubgoal**(AID: TID; ATitle: TTitle; AParent: TParent; AReference: TReference; AFile: TFileName):
  Adds a subgoal to FNodes.

• **AddChildToParent**(AChild: TID; AParent: TParent):
  Subscribes a child to its corresponding parent node.

• **ResetAttributes**(AAttributeType: TAttrType):
  Resets the attributes of all leaf nodes of the tree.

• **ToTML**: string:
  Returns a string which contains the tree structure in TML form.

• **SetLastID**(AID: TID):
  Sets the last used TID after opening a file.

• **GenerateNewID**: TID:
  Returns a new unused TID for creating a data elements.

• **CalculateNodeBoolValue**(AID: TID): TAttrBoolValue:
  Returns the boolean value of an item.

• **CalculateNodeFuzzyValue**(AID: TID): TAttrFuzzyValue:
  Returns the fuzzy value of an item.

• **CalculateNodeIntValue**(AID: TID): TAttrIntValue:
  Returns the integer value of an item.

• **DeleteNode**(AID: TID):
  Deletes an item from the tree.

• **DeleteChildren**(AID: TID):
  Deletes all children from a node.

• **MakeValidParent**(AID: TID):
  Converts an item to a node which can act as a parent.
TRoot

Parent class:

- TObject

Fields:

- FID: TID
- FTitle: TTitle
- FType: TType
- FAttributeType: TAttrType;
- FImpact: TImpact;
- FColorFile: TFileName;
- FChildren: TArrID;
- FChildCount: integer;

A TRoot instance can be identified via its FID field. It has a title, which is the goal description of the attack tree. The FType field specifies the type of the root node; e.g for boolean values: and- or or-type. FAttributeType is used to store the domain of the value of the attributes. FImpact defines the impact of the attack. The FColorFile field holds the filename of the used file with color definitions. FChildren and FChildCount are used to keep track of the children items of the root.

Methods:

- Create(AID: TID; ATitle: TTitle; AType: TType; AImpact: TImpact; AAttributeType: TAttrType; AColorFile: string): General constructor.
- GetID: TID:
  Property method for getting the FID field.
- GetTitle: TTitle:
  Property method for getting the FTitle field.
- GetType: TType:
  Property method for getting the FType field.
- GetImpact: TImpact:
  Property method for getting the FImpact field.
• **GetColorfile**: string:
  Property method for getting the FColorFile field.

• **GetAttributeType**: TAttrType:
  Property method for getting the FAttributeType field.

• **ToString**: string:
  Returns a string which contains the roots information.

• **ToTML**: string:
  Returns a string which contains the root in TML form.

• **Update**(ATitle: TTitle; AType: TType; AImpact: TImpact; AAttributeType: TAttrType; AColorFile: string):
  Updates the properties of the root with new values.

• **SetColorFile**(AColorFile: string):
  Property method for setting the FColorFile field.

• **setAttributeType**(AAttributeType: TAttrType):
  Property method for setting the FAttributeType field.

• **AddChild**(AID: TID):
  Adds a child with the correct TID to FChildren.

• **DelChild**(AID: TID):
  Deletes a child with the correct TID from FChildren.

• **GetChildren**: TArrID:
  Property method for getting the FChildren field.

• **GetChildCount**: integer:
  Property method for getting the FChildCount field.

**TSuperNode**

*Parent class:*

• TObject

*Fields:*

• **FID**: TID
• **FTitle**: TTitle
• **FParent**: TParent
The TSuperNode class is an abstract class for creating nodes. As different node types still have common properties, the abstract TSuperNode class has fields for these properties. The fields are FID as identifier, FTitle for the name of the node and FParent which is an index for the parent node in the node list of the tree.

Methods:

- **ToString**: string:
  Returns a string which contains the nodes information.

- **ToTML**: string:
  Returns a string which contains the node in TML form.

- **GetNodeType**: TNodeType:
  Returns which type of node an descendant instance is.

- **GetID**: TID:
  Property method for getting the FID field.

- **GetTitle**: TTitle:
  Property method for getting the FTitle field.

- **GetParent**: TParent:
  Property method for getting the FParent field.

TNode

Parent class:

- **TSuperNode

Fields:

- **FType**: TType
- **FChildren**: TArrID
- **FChildCount**: integer

As in the TRoot class, the FType field of the TNode class defines the type of the node. FChildren and FChildCount are used to keep track of the children items of the node.

Methods:

- **Create**(AID: TID; ATitle: TTitle; AParent: TParent):
  General constructor.
• **SetType** (AType: TType):
  Property method for setting the FType field.

• **GetType**: TType:
  Property method for getting the FType field.

• **ToString**: string:
  Returns a string which contains the nodes information.

• **ToTML**: string:
  Returns a string which contains the node in TML form.

• **GetNodeType**: TNodeType:
  Returns node as type of TSuperNode descendant.

• **AddChild**(AID: TID):
  Adds a child with the correct TID to FChildren.

• **DelChild**(AID: TID):
  Deletes a child with the correct TID from FChildren.

• **Update**(ATitle: TTitle; AType: TType):
  Updates the properties of the node with new title and type.

• **GetChildCount**: integer:
  Property method for getting the FChildren field.

• **GetChildren**: TArrID:
  Property method for getting the FChildCount field.

**TLeaf**

*Parent class:*

• **TSuperNode**

*Fields:*

• **FAttribute**: TAttribute

The TLeaf class has a field FAttributes which is a list of attributes the leaf contains.

*Methods:*

• **Create**(AID: TID; ATitle: TTitle; AParent: TParent; AAttribute: TAttribute):
  General constructor.
• **ToString**: string:
  Returns a string which contains the leaf's information.

• **ToTML**: string:
  Returns a string which contains the leaf in TML form.

• **GetNodeType**: TNodeType:
  Returns leaf as type of TSuperNode descendant.

• **GetAttribute**: TAttribute:
  Property method for getting the FAttribute field.

• **Update**(ATitle: TTitle):
  Updates the properties of the leaf with new title.

• **ResetAttribute**(AAttributeType: TAttrType):
  Resets the attribute to an empty attribute of a given type.

**TSubgoal**

*Parent class:*

  • **TSuperNode**

*Fields:*

  • **FReference**: TReference
  
  • **FFile**: TFilename

The TSubgoal class has a field FReference which is a description of a tree which the subgoal references to, and a filename which is the file the subgoal references to.

*Methods:*

  • **Create**(AID: TID; ATitle: TTitle; AParent: TParent):
    General constructor.

  • **SetReference**(AReference: TReference):
    Property method for setting the FReference field.

  • **SetFile**(AFilename: TFilename):
    Property method for setting the FFile field.

  • **ToString**: string:
    Returns a string which contains the subgoals information.
• **ToTML**: string:
  Returns a string which contains the subgoal in TML form.

• **GetNodeType**: TNodeType:
  Returns subgoal as type of TSuperNode descendant.

• **GetReference**: TReference:
  Property method for getting the FReference field.

• **GetFile**: TFileName:
  Property method for getting the FFile field.

• **Update**(ATitle: TTitle; AReference: TReference; AFilename: TFilename):
  Updates the properties of the subgoal with new title, reference and filename.

**TAttribute**

*Parent class:*

• TObject

*Fields:*

• **FID**: TAttrID
• **FName**: TAttrName

Each instance of TAttribute has an identifier which is contained in the FID field and a name which is contained in the FName field.

*Methods:*

• **ToString**: string:
  Returns a string which contains the attributes information.

• **ToTML**: string:
  Returns a string which contains the attribute in TML form.

• **ValueToString**: string:
  Returns a string which contains the value of the attribute.

• **GetName**: TAttrName:
  Property method for getting the FName field.

• **GetID**: TAttrID:
  Property method for getting the FID field.

• **Destroy**:
  General destructor.
TAttributeBool

Parent class:

- TAttribute

Fields:

- FValue: TAttrBoolValue

The FValue field contains a boolean value of the attribute.

Methods:

- Create(AID: TAttrID; AName: TAttrName; AValue: TAttrBoolValue): General constructor.
- GetValue: TAttrBoolValue: Property method for getting the FValue field.
- ValueToString: string: Returns a string which contains the value of the attribute.
- ToString: string: Returns a string which contains the attributes information.
- ToTML: string: Returns a string which contains the attribute in TML form.
- Update(AName: TAttrName; AValue: TAttrBoolValue): Updates the properties of the attribute with new name and value.

TAttributeFuzzy

Parent class:

- TAttribute

Fields:

- FValue: TAttrFuzzyValue

The FValue field contains a fuzzy value of the attribute.

Methods:
• **Create**(AID: TAttrID; AName: TAttrName; AValue: TAttrFuzzyValue): General constructor.

• **GetValue**: TAttrFuzzyValue:
  Property method for getting the FValue field.

• **ValueToString**: string:
  Returns a string which contains the value of the attribute.

• **ToString**: string:
  Returns a string which contains the attributes information.

• **ToTML**: string:
  Returns a string which contains the attribute in TML form.

• **Update**(AName: TAttrName; AValue: TAttrFuzzyValue):
  Updates the properties of the attribute with new name and value.

---

TAttributeInt

*Parent class:*

• **TAttribute**

*Fields:*

• **FValue**: TAttrIntValue

The FValue field contains a integer value of the attribute.

*Methods:*

• **Create**(AID: TAttrID; AName: TAttrName; AValue: TAttrIntValue):
  General constructor.

• **GetValue**: TAttrIntValue:
  Property method for getting the FValue field.

• **ValueToString**: string:
  Returns a string which contains the value of the attribute.

• **ToString**: string:
  Returns a string which contains the attributes information.

• **ToTML**: string:
  Returns a string which contains the attribute in TML form.

• **Update**(AName: TAttrName; AValue: TAttrIntValue):
  Updates the properties of the attribute with new name and value.
TSuperRules

*Parent class:*

- TObject

*Fields:*

- FInputLen: integer

The TSuperRules class is an abstract class for creating calculation rules. The common property for different calculation rules is FInputLen, which contains the number of values which has been added to the calculation rule.

*Methods:*

- Create: General constructor.

TBoolRules

*Parent class:*

- TSuperRules

*Fields:*

- FInput: array of TAttrBoolValue

The FInput field is an array with boolean values which are used to calculate the result value.

*Methods:*

- ruleAnd: TAttrBoolValue: Calculates and returns the value after applying the proper calculation rule.

- ruleOr: TAttrBoolValue: Calculates and returns the value after applying the proper calculation rule.

- ruleAndToString: string: Returns a string which contains the result of the calculation rule.

- ruleOrToString: string: Returns a string which contains the result of the calculation rule.
• **Add**(*AValue: TAttrBoolValue*):
  Adds a value to the FInput field.

• **Create**:
  General constructor.

**TFuzzyRules**

*Parent class:*

• **TSuperRules**

*Fields:*

• **FInput**: array of TAttrFuzzyValue

The FInput field is an array with fuzzy values which are used to calculate the result value.

*Methods:*

• **ruleAnd**: TAttrFuzzyValue:
  Calculates and returns the value after applying the proper calculation rule.

• **ruleOr**: TAttrFuzzyValue:
  Calculates and returns the value after applying the proper calculation rule.

• **ruleAndToString**: string:
  Returns a string which contains the result of the calculation rule.

• **ruleOrToString**: string:
  Returns a string which contains the result of the calculation rule.

• **Add**(*AValue: TAttrFuzzyValue*):
  Adds a value to the FInput field.

• **Create**:
  General constructor.

**TIntRules**

*Parent class:*

• **TSuperRules**
Fields:

- **FInput**: array of TAttrIntValue

The FInput field is an array with integer values which are used to calculate the result value.

Methods:

- **ruleAnd**: TAttrIntValue:
  Calculates and returns the value after applying the proper calculation rule.

- **ruleOr**: TAttrIntValue:
  Calculates and returns the value after applying the proper calculation rule.

- **ruleAndToString**: string:
  Returns a string which contains the result of the calculation rule.

- **ruleOrToString**: string:
  Returns a string which contains the result of the calculation rule.

- **Add(AValue: TAttrIntValue)**:
  Adds a value to the FInput field.

- **Create**:
  General constructor.

**Frontend and backend**

**TBase**

*Parent class:*

- TForm

*Fields:*

- **FTMLInterface**: TTMLInterface
- **FUserInterface**: TUserInterface
- **FTree**: TTree
The TBase class is represented by the main form and works as a controlling class which connects the frontend with the backend. It contains a field FTMLInterface which provides file handling functionality and a field FUserInterface, which handles the user interface part of the application. The FTree field contains the tree structure on which actions are performed.

Methods:

- **SetFilename**(AFilename: TFileName):
  Uses the filename of the active document to set the caption of the program and gives the filename to the FTMLInterface field.

**TTMLInterface**

*Parent class:*

- **TXMLDocument**

*Fields:*

- **FFileName**: TFileName
- **FXMLDocument**: TXMLDocument
- **FTree**: TTree
- **FLastID**: TID

The TMLInterface class has a field FFileName which specifies the file (on disk) on which file handling routines will be performed. If a file is loaded, it is places into FXMLDocument, which is a XML structure. After that, it is parsed and transformed into the internal data structure field FTree. The FLastID field contains the last TID which is used in the tree that has been parsed.

*Methods:*

- **HandleRoot**(AID: TID; ATitle: TTitle; AType: string; AImpact: TImpact; AAttributeType: string; AColorFile: string):
  Helper function which uses the parsed information to create a new root object.

- **HandleNode**(AID: TID; ATitle: TTitle; AParent: TID; AType: string):
  Helper function which uses the parsed information to create a new node object.

- **HandleLeaf**(AID: TID; ATitle: TTitle; AParent: TID; AAttribute: TAttribute):
  Helper function which uses the parsed information to create a new leaf object.
• **HandleSubgoal** (AID: TID; ATitle: TTitle; AParent: TID; AReference: TReference; AFilename: TFileName):  
  Helper function which uses the parsed information to create a new subgoal object.

• **Create**  
  General constructor.

• **OpenTree** (AFilename: TFileName): TTree:  
  Helper function for returning a tree from a parsed file.

• **LoadTML** (AFileName: TFileName):  
  Loads the TML file into the FXMLDocument.

• **ParseTML**:  
  Parses the TML information into tree items and builds the tree.

• **SaveTML** (ATMLString: string):  
  Saves a string with TML information to a file.

• **ExportTML** (AFileName: TFileName; AString: string):  
  Exports a string with TML information to a file.

• **GetTree**: TTree:  
  Property method for getting the FTree field.

• **SetFilename** (AFileName: TFileName):  
  Property method for setting the FFileName field.

• **GetFilename**: TFileName:  
  Property method for getting the FFileName field.

• **GetLastID**: TID:  
  Property method for getting the FLastID field.

• **GetBoolValueFromTree** (AFilename: TFileName): TAttrBoolValue:  
  Returns the boolean value of the root of the specified tree.

• **GetFuzzyValueFromTree** (AFilename: TFileName): TAttrFuzzyValue:  
  Returns the fuzzy value of the root of the specified tree.

• **GetIntValueFromTree** (AFilename: TFileName): TAttrIntValue:  
  Returns the integer value of the root of the specified tree.

**TUUserInterface**

*Parent class:*

• ** TObject**
Fields:

- **FTree**: TTree
- **FUITreeView**: TUITreeView
- **FUIRootEditor**: TUIRootEditor
- **FUINodeEditor**: TUINodeEditor
- **FUILeafEditor**: TUILeafEditor
- **FUISubgoalEditor**: TUISubgoalEditor
- **FUIFilterEditor**: TUIFilterEditor
- **FPcEditors**: TPageControl
- **FSelectedItemID**: TID
- **FSelectedRootID**: TID
- **FSelectedNodeID**: TID
- **FSelectedLeafID**: TID
- **FSelectedSubgoalID**: TID

The TUserInterface class is a wrapper class for different user interface classes. All these items have to work with attack tree data, which is contained in the FTree field. The FUITreeView field contains a user interface component treeview, which can be used to view the internal tree data. The field FUIFilterEditor is used to edit a filter for a filtered display of the treeview. The four editor fields FUIRootEditor, FUINodeEditor, FUILeafEditor and FUISubgoalEditor are used to edit selected items of the attack tree. Which item is selected is contained in the general FSelectedItemID field and the four specific fields FSelectedRootID, FSelectedNodeID, FSelectedLeafID and FSelectedSubgoalID. FPcEditors acts as a parent control for the editors.

Methods:

- **FUITreeViewChange(Sender: TObject; aNode: TTreeNode)**: Event handler for the OnChange event of the treeview.
- **FRootEditorOK(Sender: TObject)**: Event handler for the OnClick event of the OK button of the Root Editor.
- **FNodeEditorOK(Sender: TObject)**: Event handler for the OnClick event of the OK button of the Node Editor.
- **FLeafEditorOK**(Sender: TObject):
  Event handler for the OnClick event of the OK button of the Leaf Editor.

- **FSubgoalEditorOK**(Sender: TObject):
  Event handler for the OnClick event of the OK button of the Subgoal Editor.

- **FFilterEditorOK**(Sender: TObject):
  Event handler for the OnClick event of the OK button of the Filter Editor.

- **FFilterEditorClear**(Sender: TObject):
  Event handler for the OnClick event of the Clear button of the Filter Editor.

- **SetRootEditor**(ARoot: TRoot):
  Sets the user interface of the root editor for the selected root.

- **SetNodeEditor**(ANode: TNode):
  Sets the user interface of the node editor for the selected node.

- **SetLeafEditor**(ALeaf: TLeaf):
  Sets the user interface of the leaf editor for the selected leaf.

- **SetSubgoalEditor**(ASubgoal: TSubgoal):
  Sets the user interface of the subgoal editor for the selected subgoal.

- **Create**(ATreeViewPanel: TPanel; AEditorPanel: TPanel; APCEditor: TPageControl; APopupMenu: TPopupMenu): General constructor.

- **SetTree**(ATree: TTree): Property method for setting the FTree field.

- **Update**:
  Refreshes the treeview.

- **GetExportString**: string:
  Returns a string with a plain text representation of the treeview.

- **GetSelectedItemID**: TID:
  Property method for getting the FSelectedItemID field.

- **SetColorfile**(AFilename: TFileName):
  Property method for setting the FColorfile field.

---

**TUITreeView**

*Parent class:*

- **TTreeView**

*Fields:*

---

90
• **FTree**: TTree
• **FTreeIDs**: array of TTreeNode
• **FTreeIDLen**: integer
• **FImageList**: TImageList
• **FColors**: TColors
• **FColorFile**: TFileName
• **FFilter**: TFilter
• **FFilterActive**: boolean

The TUITreeView class get its data from the internal tree data structure which is placed into the FTree field. To build the treeview, two helper fields FTreeIDs and FTreeIDLen are used which contain TIDs of nodes of the tree. FImageList contains images which can be placed for nodes in the attack tree. The images are determined by the color of an item. For this, the fields FColors and FColorFile are used. The fields FFilter and FFilterActive are used for displaying a filtered view of the treeview.

**Methods:**

- **AddUIRoot**(ATitle: TTitle):
  Helper function to add a root item to the treeview.
- **AddUITreeItem**(AID: TID; ATitle: TTitle; AParent: TID):
  Helper function to add a tree item to the treeview.
- **Create**(AParent: TWinControl):
  General constructor.
- **SetTree**(ATree: TTree):
  Property method for setting the FTree field.
- **View**:
  Builds the treeview from scratch.
- **RefreshAll**:
  Clears the treeview and builds it from scratch.
- **RefreshRoot**:
  Refreshes the root of the treeview with a new text and image index.
- **RefreshItem**(AID: TID):
  Refreshes an item of the treeview with a new text and image index.
• **ToString**: string:
  Returns a string which contains a plain text version of the treeview.

• **UpdateImageIndex**(AID: TID):
  Gets the color index for an tree item and updates the image index.

• **SetColorfile**(AFilename: TFileName):
  Property method for setting the FColorFile field.

• **GetColorIndex**(AID: TID): integer:
  Property method for getting the FColorFile field.

• **SelectItem**(AID: TID):
  Makes an item in the treeview selected.

• **SetFilter**(AFilter: TFilter):
  Property method for setting the FFilter field.

• **ClearFilter**:
  Makes the usage of a filter inactive and clears the current filter in the FFilter field.

**TFilter**

*Parent class:*

* **TObject**

**Fields:**

• **FValueType**: TAttrType
• **FBoolValue**: TAttrBoolValue
• **FFuzzyValue**: TAttrFuzzyValue
• **FIntValue**: TAttrIntValue
• **FOrder**: TFilterOrder

The FValueType field defines the domain of the value the object contains. One of the three value fields FBoolValue, FFuzzyValue and FIntValue contain the value, depending of the value type. The FOrder field specifies how the actual value will be compared to the filter value.

**Methods:**

---

92
• **Create:**
  General constructor.

• **SetBoolValue**(AValue: TAttrBoolValue):
  Property method for setting the FBoolValue field. Sets also the FValueType to bool.

• **SetFuzzyValue**(AValue: TAttrFuzzyValue):
  Property method for setting the FFuzzyValue field. Sets also the FValueType to fuzzy.

• **SetIntValue**(AValue: TAttrIntValue):
  Property method for setting the FIntValue field. Sets also the FValueType to int.

• **SetOrder**(AOrder: TFilterOrder):
  Property method for setting the FOrder field.

• **GetBoolValue**: TAttrBoolValue:
  Property method for getting the FBoolValue field.

• **GetFuzzyValue**: TAttrFuzzyValue:
  Property method for getting the FFuzzyValue field.

• **GetIntValue**: TAttrIntValue:
  Property method for getting the FIntValue field.

• **GetOrder**: TFilterOrder:
  Property method for getting the FOrder field.

• **GetValueType**: TAttrType:
  Property method for getting the FValueType field.

**TColors**

*Parent class:*

• **TObject**

*Fields:*

• **FFilename**: TFileName;
• **FColors**: TColorArray;
• **FColorCount**: integer;
The FileName contains the filename of the color file in which the color data is specified. If the file has been read and parsed, the FColor array is build and contains the color data. The FColorCount field defines the number of items in the FColors array.

Methods:

- **Create**: General constructor.

- **SetFilename**(AFilename: TFileName): Property method for setting the FileName field.

- **ParseFile**: Reads and parses the specified file and builds the FColors array.

- **BoolValueToColorImage**(AValue: TAttrBoolValue): string: Returns the filename of the image which corresponds with the specified boolean value.

- **BoolValueToImageIndex**(AValue: TAttrBoolValue): integer: Returns the index of the image which corresponds with the specified boolean value.

- **FuzzyValueToColorImage**(AValue: TAttrFuzzyValue): string: Returns the filename of the image which corresponds with the specified fuzzy value.

- **FuzzyValueToImageIndex**(AValue: TAttrFuzzyValue): integer: Returns the index of the image which corresponds with the specified fuzzy value.

- **IntValueToColorImage**(AValue: TAttrIntValue): string: Returns the filename of the image which corresponds with the specified integer value.

- **IntValueToImageIndex**(AValue: TAttrIntValue): integer: Returns the index of the image which corresponds with the specified integer value.

- **GetColorCount**: integer: Property method for getting the FColorCount field.

- **GetColorImage**(AIndex: integer): string: Returns the filename of the image with a given index.
**TSuperColor**

*Parent class:*

- TObject

*Fields:*

- **FColorImage**: string;

  The FColorImage field contains the filename of an image.

*Methods:*

- **Create**(AString: string; AColorImage: string):
  General constructor.

- **GetColorImage**: string:
  Property method for getting the FColorImage field.

**TBoolColor**

*Parent class:*

- TSuperColor

*Fields:*

- **FValue**: TAttrBoolValue;

  The FValue field contains the boolean value of the object.

*Methods:*

- **Create**(AString: string; AColorImage: string):
  General constructor.

- **GetValue**: TAttrBoolValue:
  Property method for getting the FValue field.

- **GetColorImage**: string:
  Property method for getting the FColorImage field.
TFuzzyColor

*Parent class:*

- TSuperColor

*Fields:*

- **FValue**: TAttrFuzzyValue;

The FValue field contains the fuzzy value of the object.

*Methods:*

- **Create**(AString: string; AColorImage: string): General constructor.
- **GetValue**: TAttrFuzzyValue: Property method for getting the FValue field.
- **GetColorImage**: string: Property method for getting the FColorImage field.

TIntColor

*Parent class:*

- TSuperColor

*Fields:*

- **FValue**: TAttrIntValue;

The FValue field contains the integer value of the object.

*Methods:*

- **Create**(AString: string; AColorImage: string): General constructor.
- **GetValue**: TAttrIntValue: Property method for getting the FValue field.
- **GetColorImage**: string: Property method for getting the FColorImage field.
**TUIRootEditor**

*Parent class:*

- TFrame

*Fields:*

- No fields.

*Methods:*

- No methods.

**TUINodeEditor**

*Parent class:*

- TFrame

*Fields:*

- No fields.

*Methods:*

- No methods.

**TUILeafEditor**

*Parent class:*

- TFrame

*Fields:*

- No fields.

*Methods:*

- No methods.
**TUISubgoalEditor**

*Parent class:*

- TFrame

*Fields:*

- No fields.

*Methods:*

- No methods.

**TUIFilterEditor**

*Parent class:*

- TFrame

*Fields:*

- No fields.

*Methods:*

- No methods.