ABSTRACT

Traditional system documentation focuses on the behaviour or functionality we would like the system or application to provide. However, it is equally important to document the undesirable behaviour; what happens when things goes wrong. Moreover, this documentation must be unambiguous and easy to read and understand for the different stakeholders involved. SINTEF has developed a graphical language, the CORAS language for security assessment, allowing undesirable behaviour to be documented in the form of threat scenarios. The CORAS language covers notions like asset, threat, risk and treatment. The objective of this report is to demonstrate the suitability of the CORAS language for modelling threats in relation to: Web Services, ASP.NET, SQL Server, Active Directory and SmartCards.
**Executive Summary**

Traditional system documentation focuses on the behaviour or functionality we would like the system or application to provide. However, it is equally important however, to document the undesirable behaviour; what happens when things goes wrong. Moreover, this documentation must be unambiguous and easy to read and understand for the stakeholders involved. SINTEF has developed a graphical language, the CORAS language for security assessment, allowing undesirable behaviour to be documented in the form of threat scenarios. The CORAS language covers notions like asset, threat, risk and treatment.

The CORAS language is an extension of the UML [6] specification language, the de facto modelling language for information systems. It is defined as a UML profile using the extension mechanisms provided in UML itself. The language is a part of the *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* [7] that became a recommended OMG standard in November 2003. The language was developed within the CORAS project, as one of several components in the CORAS framework for security assessment.

This report demonstrates the suitability of the CORAS language for modelling threats in relation to:
- Buffer overflow
- Web Services
- ASP.NET
- SQL Server
- Active Directory
- SmartCards

This report also compares the CORAS language with the threat modelling technique described by Howard and LeBlanc [4]. There are certainly similarities, but also some clear differences. In particular, we make the following observations:
- The CORAS language introduces some concepts and features making it more expressive than Howard & LeBlanc, e.g., unwanted incident, initiate and stakeholder.
- The CORAS language is based on familiar UML notation enhanced with intuitively understandable symbols representing the basic concepts of threat modelling. Both these features increase the readability.
- The CORAS language is more general than Howard & LeBlanc.
- The CORAS language is more flexible and provides additional and stronger structuring mechanisms.
- The basic terms and concepts are sufficiently similar for the two approaches to be combined without much effort.
- The threat models of Howard & LeBlanc would constitute a useful complement to the CORAS language.
Table of contents

Executive Summary ........................................................................................................................... 2

Table of contents ............................................................................................................................. 3

1 Introduction .................................................................................................................................. 5

2 Threat modelling with the CORAS language – a tutorial ............................................................ 7
   2.1 Informal specification ............................................................................................................... 7
   2.2 Assets ...................................................................................................................................... 7
   2.3 Threats and unwanted incidents ............................................................................................ 8
   2.4 Treatments ............................................................................................................................. 9
   2.5 Overview ................................................................................................................................ 10

3 Threat modelling for buffer overrun ........................................................................................... 11
   3.1 Informal Specification .............................................................................................................. 11
   3.2 Assets ...................................................................................................................................... 11
   3.3 Threats and unwanted incidents ............................................................................................ 12
   3.4 Treatments ............................................................................................................................. 12
   3.5 Overview ................................................................................................................................ 13

4 Threat modelling for Web Services ............................................................................................. 16
   4.1 Spoofing of client requests .................................................................................................... 16
      4.1.1 Informal specification ...................................................................................................... 16
      4.1.2 Assets ............................................................................................................................. 17
      4.1.3 Threats and unwanted incidents .................................................................................... 17
      4.1.4 Treatments ...................................................................................................................... 22
      4.1.5 Overview ........................................................................................................................ 22
      4.2 SOAP message replay ........................................................................................................ 23
         4.2.1 Informal specification .................................................................................................... 23
         4.2.2 Assets ............................................................................................................................. 24
         4.2.3 Threats and unwanted incidents .................................................................................... 25
         4.2.4 Treatments ...................................................................................................................... 26
         4.2.5 Overview ........................................................................................................................ 27
      4.3 Reverse engineering of client assemblies .......................................................................... 28
         4.3.1 Informal specification .................................................................................................... 28
         4.3.2 Assets ............................................................................................................................. 29
         4.3.3 Threats and unwanted incidents .................................................................................... 29
         4.3.4 Treatments ...................................................................................................................... 30
         4.3.5 Overview ........................................................................................................................ 31

5 Threat modelling for ASP.NET .................................................................................................... 33
   5.1 Uploading malicious program code via upload-input-field .................................................... 33
      5.1.1 Informal specification ...................................................................................................... 33
      5.1.2 Assets ............................................................................................................................. 34
      5.1.3 Threats and unwanted incidents .................................................................................... 35
      5.1.4 Treatments ...................................................................................................................... 36
      5.1.5 Overview ........................................................................................................................ 37

6 Threat modelling for SQL Server ............................................................................................... 39
   6.1 SQL Injection ......................................................................................................................... 39
      6.1.1 Informal specification ...................................................................................................... 39
      6.1.2 Assets ............................................................................................................................. 41
      6.1.3 Threats and unwanted incidents .................................................................................... 42
7  Threat modelling for Active Directory ............................................................... 49
7.1  Masquerade as a genuine user .................................................................. 49
7.1.1  Informal specification ........................................................................... 49
7.1.2  Assets ....................................................................................................... 50
7.1.3  Threats and unwanted incidents ............................................................ 51
7.1.4  Treatments ............................................................................................... 53
7.1.5  Overview .................................................................................................. 55
8  Threat modelling for SmartCards................................................................. 57
8.1  Token is no longer in the possession of the genuine holder ...................... 57
8.1.1  Informal specification ........................................................................... 57
8.1.2  Assets ....................................................................................................... 59
8.1.3  Threats and unwanted incidents ............................................................ 60
8.1.4  Treatments ............................................................................................... 60
8.1.5  Overview .................................................................................................. 61
9  Comparing CORAS to the Howard & LeBlanc approach ......................... 63
9.1  Evaluation criteria ...................................................................................... 63
9.2  Comparison of terms ................................................................................ 63
9.3  Comparison of models ............................................................................. 64
  9.3.1  System decomposition ......................................................................... 64
  9.3.2  Assets ....................................................................................................... 65
  9.3.3  Threats & unwanted incidents ............................................................... 65
  9.3.4  Risks ......................................................................................................... 65
  9.3.5  Treatments ............................................................................................... 65
  9.3.6  Overview .................................................................................................. 65
  9.3.7  Hacker questions .................................................................................... 66
9.4  Evaluation ................................................................................................... 66
10 Conclusions .................................................................................................. 67

References ...................................................................................................... 68

Appendix A  Terms and Definitions ................................................................. 69

Appendix B  The CORAS profile ..................................................................... 71
Risk Analysis Metamodel ............................................................................. 71
Context ........................................................................................................... 72
SWOT .............................................................................................................. 73
Unwanted incident ......................................................................................... 73
Threat agent .................................................................................................. 74
Risk ............................................................................................................... 75
Treatment ...................................................................................................... 76

Appendix C  Icons used in the CORAS profile ............................................. 78
Context ........................................................................................................... 78
SWOT .............................................................................................................. 78
Unwanted incident ......................................................................................... 78
Risk ............................................................................................................... 79
Treatment ...................................................................................................... 79
1 Introduction

A threat is by definition dangerous and therefore important. When it comes to human beings, many threats are reflected instinctively. Any snake, even the non lethal ones, scares most of us. This is because through evolution we have indirectly experienced many dangerous situations with snakes involved. When it comes to information systems we lack this experience. While a snake can scare us without carrying a note saying: “if you are a human being, I might be a threat to you”, the threats connected to information systems must be documented in a clear and understandable way for us to see them. Documenting threats in a clear and understandable way is what threat modelling is all about; tagging the snakes and spiders out there in the world of information systems.

Traditional system documentation focuses on the behaviour or functionality we would like the system or application to provide. However, it is equally important however, to document the undesirable behaviour; what happens when things goes wrong. Moreover, this documentation must be unambiguous and easy to read and understand for the stakeholders involved. SINTEF has developed a graphical language, the CORAS language for security assessment, allowing undesirable behaviour to be documented in the form of threat scenarios. The CORAS language covers notions like asset, threat, risk and treatment.

The objective of this report is to demonstrate the suitability of the CORAS language for modelling threats in relation to:

- Buffer overrun
- Web Services
- ASP.NET
- SQL Server
- Active Directory
- SmartCards

The CORAS language is an extension of the UML [6] specification language, the de facto modelling language for information systems. It is defined as a UML profile using the extension mechanisms provided in UML itself. The language is a part of the UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [7] that became a recommended OMG standard in November 2003. The language was developed within the CORAS project, as one of several components in the CORAS framework for security assessment. More information about CORAS and its follow up project SECURIS can be found at:

- http://coras.sourceforge.net, and
- http://www.sintef.no/eway/showArticle.asp?oid=33859, respectively.

The remainder of the report is structured as follows. Chapter 2 gives an example driven introduction to threat modelling using the CORAS language. The presentation is structured in the same five subsections as the specifications of threat scenarios in Chapter 3-8. The objective of the next six chapters is then to demonstrate the CORAS approach in relation to various technologies. Chapter 3 addresses the buffer overrun issue described in Chapter 5 of Howard & LeBlanc [4]. The following five Chapters are concerned with scenarios connected to the fictive e-tax application inspired by the European e-tax application that has been described in the Microsoft project DeSecA. The e-tax application allows citizens to declare their taxes via the Internet. The application uses an electronic ID card as a mean for authentication and as a mean for signing tax declarations electronically. Figure 1 provides an overview of the e-tax application.
Each of the Chapters 4 to 8 is devoted to one kind of technology, namely Web Services, ASP.NET, SQL Server, Active Directory and SmartCards, respectively. Chapter 9 relates the CORAS language to the modelling approach of Howard and LeBlanc [4]. Chapter 10 concludes and summarizes the work. This report includes three appendices: Appendix A, listing terms and definitions, Appendix B providing a more precise definition of the CORAS language in the form of a UML profile, and Appendix C presenting the graphical symbols of the CORAS language.
2 Threat modelling with the CORAS language – a tutorial

The CORAS profile supports threat modelling. The underlying assumption when modelling threats with the CORAS profile is that the treats are part of the behaviour of a computerized system and vulnerabilities are features of the system, even though these are negative or unwanted features. This means that basically the same modelling techniques may be used for modelling threats as for modelling the desired features and functionality of a system. For example; in the same way as use case diagrams often are used for modelling the wanted or required interactions and behaviour of a system, the CORAS profile offers specialized use case diagrams for modelling threats and unwanted behaviour. As with normal use cases, these specialized use cases may be specified in more detail by standard modelling techniques using interaction diagrams, work flow diagrams, structured text, etc.

In this chapter the parts of the profile relevant for this report is presented by means of a small telemedicine example.\(^1\) The relevant parts for threat (and mitigation) modelling are asset diagrams, threat & unwanted incident diagrams, and treatment diagrams. Asset diagrams model the targets of the threats, threats and unwanted incident diagrams model the threats, and treatment diagrams model treatments (or mitigation techniques) to the treats. Appendix B provides the formal definitions of all concepts introduced in this chapter.

2.1 Informal specification

In this example we will model threats to a telemedicine system. The system uses a dedicated network for transmitting medical data, allowing a general practitioner and a cardiology expert to examine a patient together even though they are physically located at different sites. This involves retrieving the patient’s health record from a database and transmitting the relevant data to be provided to the application clients (presentation layer) running at both medical doctors’ computers.

Taking the patient’s view, the target of the threats is his/her medical data, more specifically the health record. There are two threats to this health record (1) input or wrong data, resulting in a health record containing misleading medical information; (2) illegal access to the data when transmitted over the network, with the result that the data get known by unwanted people which again may result in the data being used against the patient’s interests.

The obvious treatments to these threats are to (1) introduce some kind of validation mechanism for input of data into the health record, and (2) introduce protection mechanisms like encryption of the data being transmitted over the network.

2.2 Assets

In the CORAS profile, targets of threat are called assets, and the targets are modelled by asset diagrams. The asset diagram is shown in Figure 2.

\(^1\) This example is a subset of the example presented in [5]
Since we now look into the patient’s concerns we model the patient as a stakeholder, represented by the stick figure holding a stake. The target of the threats is represented as an asset, a stereotyped class with a stack of coins in the corner. The compartment in the middle of the asset class holds the value of the asset and in the bottom compartment the vulnerabilities opening for threats to this asset is listed.

Another stakeholder may also view the medical data as an asset, but possibly with another value. For this reason the target is modelled both as the asset “Personal patient data” and the entity “Health record”, “Personal patient data” is the target viewed as an asset from the viewpoint of the patient, modelled by the arrow tagged <<Ownership>>, while “Health record” is the target as a “neutral” entity of the system in question. The diagram also shows that there is a relationship between the asset and the entity. Showing the associated entities of the assets is optional. Threat and unwanted incident and treatment diagrams commonly only contain assets (i.e. not entities).

2.3 Threats and unwanted incidents
When using the CORAS profile, a threat is modelled using threat agents and threat scenarios, where the threat agents represent the active part of the threat and the threat scenarios model the behaviour of the threat agents. The profile has a number of predefined threat agent, such as <<Eavesdropper>> and <<Intruder>>, shown the diagram of Figure 3. The fact that they are represented by stick figures means that they are human threat agents. Each of the threat agents in the diagram is related to a threat scenario, represented by ovals with ignited bombs. The threat scenarios are again related to the asset they threat.

Event though a threat exists, does this not mean that something bad actually will happen. The actual threat may also be the result of interplay between a number of threats. We model this by the use of unwanted incidents, represented by ovals with warning signs. The <<include>> arrow is used for expressing that an unwanted incident includes a threat. Further, an unwanted incident
may lead to another unwanted incident forming a chain of events. This is modelled by the <<Initiate>> arrow. In the example the unwanted incident “Disclosure of patient data” includes the threat that an eavesdropper gets illegal access to the network, and may lead to the unwanted incident that the patient data is used against the patient’s interests. As we see, both unwanted incidents are related to the asset, meaning that the occurrence of these events have consequences for the asset.

Figure 3 Threat & unwanted incident diagram

Unwanted incidents may be evaluated as risks by assigning frequency and consequence values. This may be done in separate tables or in risk diagrams².

2.4 Treatments

As with threats and unwanted incidents, treatments are modelled by specialized use cases. Figure 4 shows the treatment diagram for the example case. An arrow from a treatment to an unwanted incident means that the treatment treats the unwanted incident. The tag on the arrow depicts what kind of treatment it is. There are four kinds of treatment arrows:

- Avoid (the unwanted incident),
- transfer (the unwanted incident to some other target),
- reduce likelihood (of the unwanted incident), and
- reduce consequence (of the unwanted incident).

² a part of the CORAS profile not presented in this report (see [5] for details)
A treatment may be evaluated with respect to its effectiveness as protection against risks. The results of such an evaluation may be shown in treatment evaluation diagrams (see [3]). This is outside the scope of this report.

2.5 Overview

A collection of UML diagrams specifies an underlying model. Each diagram in the collection provides a specific view on the model. The consequence of this is that there are no sharp boundaries between the threat & unwanted incident diagrams and the treatment diagrams, and that a model may be specified in any number of diagrams as long as they are mutually consistent. In the preceding example we could have modelled each threat and treatment in a separate diagram or merged the two diagrams modelling threats and treatments. This opens for the possibility of deriving “overview diagrams” by extracting and combining information from the threat & unwanted incident and treatment diagrams in cases when this is desirable.
3 Threat modelling for buffer overrun
This chapter addresses the buffer overrun problem described in Chapter 5 of Howard & LeBlanc [4].

3.1 Informal Specification
Problems with buffer overrun have been identified as far back as the 1960s. The reason that buffer overruns are still a problem today is because of poor coding practices, especially in connection with the use of the programming languages C and C++. Although there are different types of buffer overruns, the main problem is always related to pointers and memory addresses. Exploiting a buffer overrun is about filling a buffer with more data than it is designed for. In this way a pointer that was meant to return a value from a known address in the buffer returns the value from an uncontrolled address from outside the buffer. The value of this address can be used to redirect the running program to executing some malicious code.

3.2 Assets
The identification of assets is a key part of the context identification phase specified in the CORAS security analysis process. Assets represent the values in the system. Without any values there is nothing to loose. CORAS therefore starts a security analysis with the identification of assets. Assets are identified by stakeholders, which are people with different roles connected to development and use of the system. Figure 5 shows an example of an asset diagram, represented as a UML class diagram. As shown, the asset “Quality Level” is a “software” entity identified by a programmer and has got its asset value set to “high”. Further it has got identified a vulnerability connected to it.

![Figure 5 Asset diagram (Quality Level)](image-url)
3.3 Threats and unwanted incidents

After the assets have been identified, we need to specify how they possibly can miss their value. The threat diagram is a crucial diagram since it is the diagram that indicates the initial threat, but it is still only a threat, nothing has happened yet. After a threat has been identified it needs to be expanded to a concrete action towards the system reducing one or more of the assets value. This is what we call an unwanted incident. Figure 6 shows a threat and unwanted incident diagram, containing both threats and unwanted incidents in relation to assets and attackers. The threats and unwanted incidents are specified in this diagram as use cases, these may be specified further by means of sequence diagrams, activity diagrams or normal text. This specific diagram also contains a related and more specific unwanted incident that describes the use of Unicode and ANSI buffer size mismatches as direct threats to the WindowsNT asset.

![Figure 6 Unwanted Incident diagram](image)

3.4 Treatments

The main goal of treatments is to reduce either their consequence or frequency. In a treatment diagram, treatments are specified as a use case addressing the treatment to an unwanted incident. Just as with the threat and unwanted incident use cases, also the treatment use case can be specified in more detail with the help of sequence diagrams, activity diagrams and structured text.
3.5 Overview

An important aspect with the diagrams presented here is the requirement to provide a clear and easy to understand overview. This puts restrictions on how much information one diagram can contain. However, commonly unwanted incidents are related, in addition, there may be treatments addressing these unwanted incidents. To get an overview of the total picture it might therefore be worth while to include all related unwanted incidents and their associated treatments in one diagram. Figure 8 shows such an overview diagram.
The WindowsNT asset was already included in Figure 6 and is also included in Figure 8 again. In order to show how the CORAS profile opens for the specification of vulnerabilities connected to assets we take a closer look at the WindowsNT asset in Figure 9. One of the vulnerabilities that WindowsNT has in connection with the buffer overrun problem is related to the use of the function ‘GetServerVariable’. This vulnerability is therefore mentioned in the class diagram of the asset. A more detailed description with specific code describes the pitfall that introduces a buffer overrun when using this function.
Vulnerabilities:
- ANSI Unicode Strings
- Unicode functions deal with buffer sizes in wide characters, not byte sizes
  - GetServerVariable

// Vulnerable code that causes a Unicode bug

TCHAR wszcomputerName[256];
BOOL GetServerName(EXTENSION_CONTROL_BLOCK *pECB) {
    DWORD dwSize = sizeof(wszComputerName);
    char szComputerName[256];
    if (pECB -> GetServerVariable (pECB -> ConnID, 
        "SERVER_NAME", 
        szComputerName, 
        &dwSize)) {
        // Do something
    }
}
4 Threat modelling for Web Services

A web service is an XML based messaging interface to some computing resource. The web services protocol stack consists of:

- A transport layer protocol, typically HTTP.
- An XML based messaging layer protocol, typically SOAP.
- A service description layer protocol, typically WSDL.
- A service discovery layer protocol, typically UDDI.

In this document, the assumed web services communication model is SOAP over HTTP. Basic SOAP interactions are asynchronous and unidirectional, but can be combined to implement request/response processes, or even more sophisticated interactions.

SOAP messages are XML based messages for exchanging structured and typed information.

This document makes the assumption that WSDL is used to specify the public interface to a web service. A WSDL based description of a web service can include information on available services, typing information and address information.

The use of dynamic discovery of web services in web applications is not yet widely used. Hence this document does not consider the service discovery layer.

4.1 Spoofing of client requests

4.1.1 Informal specification

Table 1 describes the indicated threat “spoofing the client request” using the template provided by Microsoft. Three different detailed scenarios connected to this threat are described. These scenarios are directly connected to the kind of authentication that has been used, which is specified as vulnerabilities of the web server as shown in the asset diagram in Figure 10.

<table>
<thead>
<tr>
<th>Name</th>
<th>Spoofing of client requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Id</td>
<td>WST101</td>
</tr>
<tr>
<td>Description</td>
<td>Weak or no authentication of client requests allows an attacker to send requests appearing to originate from a valid client.</td>
</tr>
<tr>
<td>Threat Target</td>
<td>Any resource or functionality offered by the web service</td>
</tr>
<tr>
<td>Threat Classification</td>
<td>S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attack Scenarios</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. [No authentication used]: attacker can freely impersonate any client</td>
<td></td>
</tr>
<tr>
<td>2. [Password authentication]: attacker learns password</td>
<td></td>
</tr>
<tr>
<td>2.1 Attacker sniffs password from network</td>
<td></td>
</tr>
<tr>
<td>2.2 Fishing attack</td>
<td></td>
</tr>
<tr>
<td>2.3 Dictionary attack</td>
<td></td>
</tr>
<tr>
<td>2.4 Attacker installs Trojan horse/key logger on client machine</td>
<td></td>
</tr>
<tr>
<td>3. [Initial authentication only]: attacker can hijack a session</td>
<td></td>
</tr>
</tbody>
</table>

| Countermeasures           | Use strong authentication, e.g. Windows authentication, client certificate authentication or one time passwords. |
4.1.2 Assets

Figure 10 shows the asset WebServer in an asset diagram. The asset is identified by an independent analyst and has been given the value “high”. Also three vulnerabilities are specified, which correspond to the three attack scenarios mentioned in Table 1.

![Asset Diagram](image)

**Figure 10 Asset diagram**

4.1.3 Threats and unwanted incidents

The three different scenarios of threat WST101 are split up in WST101.1, WST101.2 and WST101.3. For each of them a threat and unwanted incident diagram is developed, a sequence diagram specifies further details of these diagrams. The diagrams are given in Figure 11-Figure 16.

Figure 11 specifies an attacker exploiting the vulnerability of the WebServer of not using authentication, in order to impersonate a client.
Figure 11 Threat and unwanted incident diagram for WST101.1

In Figure 12 the threat scenario from Figure 11 is specified in more detail in a sequence diagram. A sequence diagram specifies communication between instances (in this case Attacker, WebServer, Valid User) in the form of messages sent between these instances. Since there is no form for authentication, the attacker can log inn as soon as he gets a valid user ID in hand.
When a password authentication is required the attacker needs to get a hold on this password in some sort of way. Figure 13 shows four possibilities for how this can be done.
The sequence diagram of Figure 14 shows three of the four attack scenarios mentioned in Figure 13. The outer alt-box consists of a loop-box, which again contains an alt-box. The alt-boxes specify alternative behaviour, while the loop-box specifies repetition. Thus, the second alternative specifies a dictionary attack where a password is tried until the right one logs in the attacker.

**Figure 14 Sequence diagram for WST101.2**

Figure 15 shows that an attacker can hijack a session when only initial authentication is required by the WebServer.
Figure 15 Threat and unwanted incident diagram for WST101.3

The sequence diagram in Figure 16 shows that hijacking of a user ID needs to take place while the actual user is logged in.

Figure 16 Sequence diagram for WST101.3
4.1.4 Treatments

Three treatments are identified that could reduce the chance for an attacker managing to spoof client requests. These treatments are specified in the treatment diagram of Figure 17.

Figure 17 Treatment diagram for WST101

4.1.5 Overview

Summarizing these results it is possible to capture them in one diagram. This diagram might be difficult to read without having seen the earlier ones, but provides a complete picture.
4.2 SOAP message replay

4.2.1 Informal specification

As mentioned earlier, we assume that web services communication is SOAP over HTML. Changing SOAP messages influences therefore directly the communication with the web server and is thus regarded as a serious threat. Changing a SOAP message requires the original message as well. How this message is recorded is not addressed in this document.

<table>
<thead>
<tr>
<th>Name</th>
<th>SOAP message replay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Id</td>
<td>WST201</td>
</tr>
<tr>
<td>Description</td>
<td>Attacker records and resends a SOAP message originally constructed by a real client.</td>
</tr>
<tr>
<td>Threat Target</td>
<td>With a replay attack, the attacker typically tries to duplicate an action performed by a regular client, or tries to bring the service in an inconsistent state.</td>
</tr>
<tr>
<td>Threat Classification</td>
<td>ST</td>
</tr>
</tbody>
</table>
For resending it, the attacker only needs the ability to open an http connection to the server. If connections to the server are only possible through authenticated SSL, the attack is harder: in that case the attacker needs to have privileges on one of the two endpoints:

1. Attacker controls client machine
   1.1 Attacker is owner of client machine
   1.2 Attacker has installed Trojan on client machine

1.2 Attacker controls web server machine

Use nonce’s or timestamps in messages to prevent replays
Data protection of the SOAP call
Data protection of the communication channel (SSL or IPSEC)
Good configuration and patch management on client and server machines

4.2.2 Assets
Three assets have been identified in connection with this threat. They are given in Figure 19. Having ranged both ‘WebServer’ and ‘ClientMachine’ as high, the SOAP message itself has got the value ‘low’. Even though it is the SOAP message that can harm either the WebServer or the ClientMachine, or both, the SOAP message itself does not represent a high value.

![Figure 19 Asset diagram for WST201](image-url)
4.2.3 Threats and unwanted incidents

The specification of the threats and unwanted incidents for this case are divided in two diagrams. The first diagram is shown in Figure 20, which specifies the situation where no SSL authentication is used and the attacker just can open an http connection.

![Figure 20 Threat and unwanted incident diagram for WST201.1](image)

The second alternative situation for this case is where the attacker depends on the installation of a Trojan horse, either on the client machine or on the WebServer, in order to be able to replay a SOAP message. This specification is shown in Figure 21.
4.2.4 Treatments

Some of the countermeasures to this threat have been indicated above and are specified in the treatment diagram of Figure 22. While ‘Data protection of SOAP call’ and ‘Dataprotection of communication channel’ are specified to reduce the likelihood of a SOAP message being replayed, the use of Nonce’s or timestamps are expected to remove the whole problem.
4.2.5 Overview

An overview diagram of the presented results is shown in Figure 23.
4.3 Reverse engineering of client assemblies

4.3.1 Informal specification

The reverse engineering of client assemblies is difficult and typically not an activity carried out by a novice hacker. However, the possible damage of a successful attack of this kind is correspondingly serious.

Client assemblies are small applications designed to communicate with the WebServer and are therefore trusted by the WebServer. Fooling the WebServer by adjusting a client application can open a variety of opportunities for gaining influence or even control over the WebServer.

Reverse engineering is about trying to redesign an application through analysing the functionality of this application. This requires a thorough understanding of how the WebServer behaves and what kind of communication possibilities it offers, as well as advanced skills in (re)designing the client application.

Table 3 Reverse engineering of client assemblies

<table>
<thead>
<tr>
<th>Name</th>
<th>Reverse engineering of client assemblies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Id</td>
<td>WST204</td>
</tr>
<tr>
<td>Description</td>
<td>A client analyzes and reverse engineers assemblies downloaded from the server, in order to enable goal driven modification of these assemblies, or to derive information about the web service.</td>
</tr>
<tr>
<td>Threat Target</td>
<td>The reverse engineering is typically a probing attack: gathering information that may enable new attacks, such as the goal driven modification. Modifying the assembly can target any of the resources or functionalities of the web service.</td>
</tr>
<tr>
<td>Threat Classification</td>
<td>ST</td>
</tr>
</tbody>
</table>
### Attack Scenarios

| First: Decomilation of downloaded assemblies to get readable and understandable code |
| Then: |
| 1. Assembly is modified to remove security related checks (for instance input validation) at the client side |
| 2. Call services that are not intended to be called directly, possibly with unexpected parameters, or in an incorrect order. |

| Countermeasures | NEVER perform security related validations at the client side |
| Obfuscate assemblies |
| Reconsider the rich client design |

### 4.3.2 Assets
The WebServer is the main asset identified. The value is set to high. The security validations carried out at the client side are mentioned as a vulnerability opening for reverse engineering of client assemblies.

#### Figure 24 Asset diagram for WST204

### 4.3.3 Threats and unwanted incidents
Figure 25 shows the modelling of the different steps that are necessary for the attacker to carry out a reverse engineering of a client assembly. An important step in this process is the testing of the reactions of the WebServer represented by the unwanted incident ‘call services not intended to be called directly’.
Figure 25 Threat and unwanted incident diagram for WST204

4.3.4 Treatments
As shown in Figure 26, there have been identified two treatments that remove the possibility of services being called directly that were not designed for that purpose. The treatment of ‘obfuscating assemblies’ just removes the likelihood of this happening.
4.3.5 Overview
The overview diagram of this case is shown in Figure 27.
Figure 27 Overview diagram for WST204
5 Threat modelling for ASP.NET

ASP.NET provides a set of components for developers to implement complex functionality in DLLs. It is scalable, in that it provides state services to manage session variables (cookies, session ids, temporary URLs) across multiple Web servers in a server farm. It is stable, in that it can detect application failures and recover from them. It addresses both “managed code” (conformant to ASP.NET), as well as “unmanaged code” (“native code”) to include „legacy“ applications. Following is a summary of the benefits of ASP.NET [2]:

- Simplified development: Rich object model that developers can use to reduce the amount of code they need to write.
- Language independence: ASP pages must be written with scripting that is interpreted rather than compiled. ASP.NET allows compiled languages to be used, providing better performance and cross language compatibility.
- Simplified deployment: With .NET components, deployment is as easy as copying a component assembly to its desired location.
- Cross client capability: ASP.NET provides rich server side components that can automatically produce output specifically targeted for each type of client. For example, writing one script that will render correctly in Internet Explorer 5.5, Netscape Navigator 4.7, a PDA and a mobile phone is very difficult.
- Web services: ASP.NET is well made for clients that understand HTTP and XML, the de facto language for inter device communication.
- Performance: ASP.NET pages are compiled whereas ASP pages are interpreted. When an ASP.NET page is first requested, it is compiled and cached, or saved in memory, by the .NET Common Language Runtime (CLR). This cached copy can then be re-used for each subsequent request for the page. After the first request, the code can run from a much faster, compiled version.

5.1 Uploading malicious program code via upload-input-field

5.1.1 Informal specification

In this case a piece of program is uploaded via de input field of a web application. This program can for example be executed by the WebServer if it is saved in the right place.

Table 4 Uploading malicious program code via upload-input-field

<table>
<thead>
<tr>
<th>Name</th>
<th>Uploading malicious program code via upload-input-field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Id</td>
<td>AST103</td>
</tr>
<tr>
<td>Description</td>
<td>Some code file containing malicious code is uploaded using an upload form.</td>
</tr>
<tr>
<td></td>
<td>- The attacker gets to know the save folder on the server and tries to execute his code</td>
</tr>
<tr>
<td></td>
<td>- The uploaded file with included code is processed by the server, thus the code is executed (as VBA in Word documents could be).</td>
</tr>
<tr>
<td>Threat Target</td>
<td>Uploading a script to get access to the server (=&gt; Authenticity, Availability, Confidentiality, Accountability)</td>
</tr>
<tr>
<td>Threat Classification</td>
<td>STRIDE</td>
</tr>
</tbody>
</table>
| Attack Scenarios | 1 Attacker creates and loads up code file
| | 1.1 file is accepted and stored
| | 1.1.1 store folder is open to execution by http request
| | 1.1.1.1 ✋ attacker successful guesses folder location
| | 1.1.2 file is inserted into generated html code
| | 1.1.2.1 file contained includable program code
| | 1.1.2.1.1 ✋ code generates input form inside the html code output by the server. Data inserted into this form would be send to attackers server
| | 1.1.2.1.2 ✋ code opens connection to server
| | 1.1.3 ✋ file is processed by server
| Complexity: | html form with attackers server as recipient: 2
| | complex script code opening a connection to server: 4
| Damage: | 4-5

| Countermeasures | - Deny execution of uploaded files
| | - Restrict procession of file (included code will not be executed)
| | - Check upload files not to be code
| | - Secure save folder (no [direct] access via http)
| | - Deny file types with possibly included code (or allow files from a positive file type list)

### 5.1.2 Assets
Again the WebServer is the main identified asset.
5.1.3 Threats and unwanted incidents

The three unwanted incidents shown in Figure 29 are all unwanted situations that can appear when malicious code is uploaded to the WebServer. This may happen if the WebServer unconditionally executes programs in, for example, the save folder.
5.1.4 Treatments

Three treatments have been identified for the three unwanted incidents.
Figure 30 Treatment diagram for AST103

5.1.5 Overview
The overall picture is shown in Figure 31.
Figure 31 Overview diagram for AST103
6 Threat modelling for SQL Server

SQL Server is a relational database management system. SQL Server was designed for client/server use and is accessed by applications using SQL. It runs on Windows NT version 4.0 or higher and is compliant with the ANSI SQL 92 standard. SQL Server supports symmetric multiprocessing hardware, ODBC, OLE DB, and major open standard communications protocols. It has Internet integration, data replication, transaction management and data warehousing features.

The main role of SQL Server in a web architecture is to store and manage the data that the authorized web applications need to access. To be able to access data from a database, a user must pass through two stages of authentication, one at the SQL Server level and the other at the database level. These two stages are implemented using logins names and user accounts respectively.

6.1 SQL Injection

6.1.1 Informal specification

In addition to plain text, the description below includes an attack tree. This tree shows a more detailed specification of what is required in order for SQL injection to happen.

Table 5 SQL Injection

<table>
<thead>
<tr>
<th>Name</th>
<th>SQL Injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Id</td>
<td>DBT001</td>
</tr>
<tr>
<td>Description</td>
<td>It is a technique which exploits vulnerabilities in input validation to run arbitrary commands in the database. It can occur when the application uses input to construct dynamic SQL statements to access the database. It can also occur if the code uses stored procedures that are passed strings that contain unfiltered user input. The issue is magnified if the application uses an over privileged account to connect to the database. In this instance it is also possible to use the database server to run operating system commands and potentially compromise other servers, in addition to being able to retrieve, manipulate, and destroy data.</td>
</tr>
<tr>
<td>Threat Target</td>
<td>Applications that incorporate non validated user input into database queries. Particularly susceptible is code that constructs dynamic SQL statements with unfiltered user input.</td>
</tr>
<tr>
<td>Threat Classification</td>
<td>Classification of the threat based on STRIDE:</td>
</tr>
<tr>
<td>Spoofing of Identify</td>
<td>Y/N</td>
</tr>
<tr>
<td>Tampering with Data</td>
<td>Y</td>
</tr>
<tr>
<td>Repudiation</td>
<td>N</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Y</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>N</td>
</tr>
<tr>
<td>Elevation of Privileges</td>
<td>Y</td>
</tr>
</tbody>
</table>
### Attack Scenarios

#### Threat 1: SQL Injection

1.1 Weak input validation
1.2 Dynamic SQL statement construction
1.3 Over-privileged database logins

#### Scenario 1.3.1: Compromise Database

**Damage Potential**
The attacker can tamper the data stored in the database, or even destroy it, depending on the privileges granted to the application accessing the database server.

**Reproducibility**
The attack can be easily repeated until the proper countermeasures are taken.

**Exploitability**
A basic knowledge of SQL is needed to perform the attack.

**Affected Users**
An unauthorized data modification performed through a high privileged account can affect all the database users.

**Discoverability**
The attack can be discovered searching the log file for malformed queries. This can be quite a demanding task depending on the size of the log file.

#### Scenario 1.3.2: Information Disclosure

**Damage Potential**
The attacker can view users’ data and collect information about the database structure, but he can not modify them.

**Reproducibility**
Same as Scenario 1.3.1

**Exploitability**
Same as Scenario 1.3.1

**Affected Users**
Same as Scenario 1.3.1

---

**Risk rated with the DREAD method (0-10)**

Scenario 1.3.1

**Damage Potential**
10

**Reproducibility**
10

**Exploitability**
6

**Affected Users**
10

**Discoverability**
6

Scenario 1.3.2

**Damage Potential**
6

**Reproducibility**
9

**Exploitability**
8

**Affected Users**
10
### Countermeasures

<table>
<thead>
<tr>
<th>Discoverability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same as Scenario 1.3.1</td>
</tr>
</tbody>
</table>

#### Preventive Countermeasures
- Perform thorough input validation. The application should validate its input prior to sending a request to the database.
- Use type safe SQL parameters for data access. These parameters can be used with stored procedures or dynamically constructed SQL command strings. Perform parameters type and length checks and also ensure that injected code is treated as literal data, not executable statements in the database.
- Use an account that has restricted permissions in the database. Ideally, you should only grant execute permissions to selected stored procedures in the database and provide no direct table access.

#### Detective Countermeasures
- Log all the requests sent to the database and search the log file for malformed expression.

#### Corrective Countermeasures
- Restore the data manipulated by the injected code.

### 6.1.2 Assets
In this case the SQL server is the asset that is threatened. One of the main functions of a database is to store information without changing it unintentionally. SQL injection can lead to both changing or deleting or even disclosure of the information.

![Asset Diagram](image)

**Figure 32 Asset diagram for DBT001**
6.1.3 Threats and unwanted incidents

DBT001 is divided into three different cases specifying three different situations connected to SQL injection. Figure 33 shows that an SQL database server with a weak input validation can open for arbitrary commands to be run in the database. Another unwanted incident is shown in Figure 34, it shows a database that can be accessed through dynamic SQL statements constructed from the input of a web application.

Figure 33 Threat and unwanted incident diagram for DBT001.1
Figure 34 Threat and unwanted incident diagram for DBT001.2

A special form for SQL injection is represented by strings passed to the SQL server that contains unfiltered user input. This threat depends on whether the SQL database server uses stored procedures. This is indicated in Figure 35 as being a vulnerability of the server.
6.1.4 Treatments
Indicated treatments for the three variants of SQL injection described above are shown in Figure 36-Figure 38.
Figure 36 Treatment diagram for DBT001.1

- **Unwanted Incident**: Run arbitrary commands in the database
  - **Treatment**: Perform throughput validation
  - **Reduce Likelihood**: Use type safe SQL parameters for data access

Figure 37 Treatment diagram for DBT001.2

- **Unwanted Incident**: Accessing the database through dynamic SQL statements constructed from the input
  - **Treatment**: Perform throughput validation
  - **Reduce Likelihood**: Use type safe SQL parameters for data access
6.1.5 Overview
The three overview diagrams for DBT001 are shown in Figure 39-Figure 41.
Figure 39 Overview diagram for DBT001.1

Figure 40 Overview diagram for DBT001.2
Figure 41 Overview diagram for DBT001.3
7 Threat modelling for Active Directory

Active Directory is an essential and inseparable part of the Windows 2000 network architecture that improves the domain architecture of the Windows NT® 4.0 operating system in order to provide a directory service designed for distributed networking environments. Active Directory lets organizations efficiently share and manage information about network resources and users. In addition, Active Directory acts as the central authority for network security, letting the operating system readily verify a user's identity and control his or her access to network resources. Equally important, Active Directory acts as an integration point for bringing systems together and consolidating management tasks.

7.1 Masquerade as a genuine user

7.1.1 Informal specification

Just as in section 6.1, this specification also includes a graphical attack tree. Masquerading a genuine user can be done by either eavesdropping the network that the user is using or directly accessing the Activity Directory containing the personal information of the user. These two situations are modelled here, the last one is further separated into two cases.

Table 6 Masquerade as a genuine user

<table>
<thead>
<tr>
<th>Name</th>
<th>Masquerade as a genuine user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Id</td>
<td>ADT001</td>
</tr>
<tr>
<td>Description</td>
<td>An attacker attempts to access the Active Directory by capturing a user’s credentials and then masquerading as the user</td>
</tr>
<tr>
<td>Threat Target</td>
<td>Vulnerabilities in the network, vulnerabilities in the Active Directory Information Base and vulnerabilities in the application gateway.</td>
</tr>
</tbody>
</table>

![Attack Scenario Diagram]

- **Threat 1 Masquerade**
  - **OR**
    - **AND**
      - **1.1 Eavesdrop the network**
      - **1.2 Access AD**
      - **1.2.1 Retrieve user DN from AD**
      - **OR**
        - **1.2.1 Dictionary Attack**
        - **1.2.2 Modify PW Attribute in AD**
      - **1.2.2 Discover Password**

1.1 Complexity 3 (Sniffer programs e.g. TCPDUMP are readily available from the Internet. However the attacker has to be on the same network as the victim).

1.2.1 Complexity 0. (AD is designed to give out this sort of information, often to the public).

1.2.2.1 Complexity 2 (Dictionary attack programs are available from the Internet).

1.2.2.2 Complexity 2-4 (If the client interface allows modification operations then it could be relatively easy, otherwise the attacker might need to use the standard LDAP protocol and discover how to access AD in this way).

All Potential Damage 5 (depending upon which user account is cracked) except 1.2.1 which has a Potential Damage of 1.

**Countermeasures**

1.1 Don’t require the (username and) password to be sent unprotected over the network. Use SSL on the connection, or HMAC, or even client certificates.

1.2.1 Use usernames instead of DNs for authentication, or if clients don’t need to know user DNs, then hide them from the interface.

1.2.2.1. Have the application count the number of failed login attempts, write them to audit trails, then disable the account when the number is exceeded and notify the administrator.

1.2.2.2 Don’t provide a modification capability at the client interface. However, if this is required, then carefully validate all modification operations and trap this one.

### 7.1.2 Assets

Two main assets are affected when personal information of a genuine user is used by an unauthorized person: the database that contains the information and the user that owns the information. This is shown in the asset diagram of Figure 42.

![Asset diagram for ADT001](image)

**Figure 42 Asset diagram for ADT001**
7.1.3 Threats and unwanted incidents

The threats and unwanted incidents for this case are presented through three diagrams. The first diagram specifies how the network between the user and the Active Directory is used to get unauthorized access to the AD. The user is not a part of the diagram in Figure 43 since the specific user information is not mentioned here. This is only a general description of how eavesdropping the network can lead to unauthorized access to the Active Directory.

![Diagram](attachment:diagram.png)

**Figure 43 Threat and unwanted incident diagram for ADT001.1**

The diagram in Figure 44 shows how the user information is retrieved through direct access to the Active Directory (AD). The difference between Figure 44 and Figure 45 lies in the way the AD is treated. In the case of ADT001.2, a dictionary attack is used to gain access to the AD, while in the case of ADT001.3 information is directly changed in the AD.
Figure 44 Threat and unwanted incident diagram for ADT001.2

Figure 45 Threat and unwanted incident diagram for ADT001.3
7.1.4 Treatments

The three following treatment diagrams correspond to the threat and unwanted incidents shown in subsection 7.1.3.

Figure 46 Treatment diagram for ADT001.1
Count the number of failed login attempts, disable the account if number is exceeded.

Use usernames instead of DNs for authentication.

Remove modification capability at client interface.

Figure 47 Treatment diagram for ADT001.2

Figure 48 Treatment diagram for ADT001.3
7.1.5 Overview

Figure 49-Figure 51 provide the overview diagrams summarizing ADT001.

![Diagram](image)

**Figure 49 Overview diagram for ADT001.1**
Figure 50 Overview diagram for ADT001.2

Figure 51 Overview diagram for ADT001.3
8 Threat modelling for SmartCards

The threat modelling for security tokens mainly focuses on electronic identity (EID) cards and smartcards used for digital signatures.

An EID-card can be used

- To obtain strong authentication of the cardholder, e.g., in the SSL related context of client authentication for websites. This client authentication is accomplished by having the client sign data which is specified by the underlying challenge response protocol. The token producing the digital signature only produces this signature after some form of cardholder verification, e.g., using a personal identification number (PIN). After the successful authentication of the client, specific services may be available to the client (cf. authorization).
- To generate non repudiation signatures. These signatures, in combination with a qualified certificate, are equivalent to handwritten signatures. The production of these signatures by the token also requires the verification of the PIN of the cardholder. Note that the generation and verification of non repudiation signatures are a very complicated matter (more legally than technically).
- To obtain some information on the cardholder (such as address, social security number, date of birth, sex, …). This information can usually be acquired without any cardholder verification.

Figure 1 gives an architectural overview of the entities involved when using smartcards in a web application. The smartcard may contain user data (e.g., e-business card, home address, birth information, photograph, …), secret information (private keys to sign or decrypt information) and reference data (e.g., a genuine copy of the root certificate of the cardholder’s certification authority). SmartCards communicate with an application on the user PC through the smartcard reader, and the user has to authenticate himself/herself to its smartcard using a PIN. This PIN can be entered on the smartcard reader, on the keyboard of the user PC, or it may have been cached on the user PC for convenience reasons. If an attacker can inject a Trojan horse in the user machine and this Trojan is run with, or gains, sufficient privileges, it can control different software components.

Figure 52 Architectural overview of SmartCards in a web environment

8.1 Token is no longer in the possession of the genuine holder

8.1.1 Informal specification

There is no conceptual difference between a software token and a hardware token. Both are used for identification. One might even argument that hardware tokens do not exist, because every
hardware token contains a software token. However, for a user the difference is clear. A hardware token needs a place in a bag or a pocket, and it can be missed or destroyed, while a software token is stored on a PC or a Palm. This difference is used in the modelling of the threat of missing a token through the specification of different assets for both hardware and software assets.

Table 7 Token is no longer in the possession of the genuine holder

<table>
<thead>
<tr>
<th>Name</th>
<th>Token is no longer in the possession of the genuine holder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Id</td>
<td>SCT001</td>
</tr>
<tr>
<td>Description</td>
<td>It is not certain whether an attacker has obtained the token, or whether the token will remain inactive. This type of threat is seen as a denial of service attack: the legitimate user can no longer use the token’s services. The goal of the attacker is not to use the services of the token him/herself, but to make the token unavailable to the legitimate user.</td>
</tr>
<tr>
<td>Threat Target</td>
<td>• Both hardware and software tokens</td>
</tr>
<tr>
<td></td>
<td>• (Service) availability</td>
</tr>
<tr>
<td>Threat Classification</td>
<td>Classification of the threat based on STRIDE:</td>
</tr>
<tr>
<td></td>
<td>Spoofing of Identify</td>
</tr>
<tr>
<td></td>
<td>Tampering with Data</td>
</tr>
<tr>
<td></td>
<td>Repudiation</td>
</tr>
<tr>
<td></td>
<td>Information Disclosure</td>
</tr>
<tr>
<td></td>
<td>Denial of Service</td>
</tr>
<tr>
<td></td>
<td>Elevation of Privileges</td>
</tr>
<tr>
<td>Attack Scenarios</td>
<td>1.1 The genuine token holder loses the token (involuntary).</td>
</tr>
<tr>
<td></td>
<td>1.2 An attacker obtains (e.g., finds or steals) the token.</td>
</tr>
<tr>
<td>The risk scaled with the DREAD method:</td>
<td>(0-10)</td>
</tr>
<tr>
<td>Damage Potential</td>
<td>Threat 1.1: If the user loses the token, it should be assumed that the token is stolen (threat 1.2). 8 8</td>
</tr>
<tr>
<td></td>
<td>Threat 1.2: Apart from being unavailable, the token should be considered to be under attack; the attacker can tamper with the token at will and other threats will apply (especially SCT005).</td>
</tr>
<tr>
<td>Reproducibility</td>
<td>Threat 1.1: This threat is completely out of control of the attacker, but it is quite likely to happen. 1 6</td>
</tr>
<tr>
<td></td>
<td>Threat 1.2: Unless procedures for physical security are used, the attacker can perform this attack in a medium scale.</td>
</tr>
<tr>
<td>Exploitability</td>
<td>Threat 1.1: This happens involuntary: there is no attacker involved in losing the token. 1 7</td>
</tr>
<tr>
<td></td>
<td>Threat 1.2: It is not hard to steal a token.</td>
</tr>
<tr>
<td>Affected Users</td>
<td>Threat 1.1: Only the genuine user of the token is affected. 2</td>
</tr>
<tr>
<td></td>
<td>Threat 1.2: Idem. 2</td>
</tr>
</tbody>
</table>
Discoverability

Threat 1.1: This “attack” requires no technical knowledge and is known to everybody.
Threat 1.2: Idem.

<table>
<thead>
<tr>
<th>Countermeasures</th>
<th>Indication of the countermeasures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preventive Countermeasures</td>
<td>Educate the user to protect the token so that the token will not easily be lost or cannot easily be stolen.</td>
</tr>
<tr>
<td>Detective Countermeasures</td>
<td>If the token is stolen, then the user should be able to detect this as soon as possible.</td>
</tr>
<tr>
<td>Corrective Countermeasures</td>
<td>If this incident happens, the genuine token holder should have the token revoked so that a potential attacker cannot abuse it any longer.</td>
</tr>
</tbody>
</table>

8.1.2 Assets

Of the three assets shown in Figure 53, service availability is ranged as ‘high’, while both software token and hardware token are ranged as ‘low’. Even though service availability depends on the user having a valid token, the scope is regarded much wider since service availability affects all possible users of the service.

Notice that a software token can be copied and thus stolen without having the owner losing it. The first time the owner might notice that the software token has been stolen is when the service is unavailable while he uses the token.

![Figure 53 Three identified assets for SCT001](image-url)
8.1.3 Threats and unwanted incidents

The left side of the diagram in Figure 54 addresses the hardware token and the right side the software token. Service availability is only associated to the software token. Although the service can be unreachable without a hardware token it is not necessarily unavailable. The software token can be in place while the service is unavailable, if the software token is broken or being used by an unauthorized user.

![Diagram]

Figure 54 Threat and unwanted incident diagram for SCT001

8.1.4 Treatments

Two treatments have been identified connected to the loss of soft- and hardware tokens. In order to prevent the user from losing a hardware token he/she needs to be educated. At some point in time a hardware token might become as important, as as or maybe even more important than a credit card. The treatment of revoking the token is an ‘after’ treatment, meant to reduce possible loss after the token has been stolen.
8.1.5 Overview
The overview diagram is shown in Figure 56.
<<ThreatScenario>>
Token kept in a place where it can easily be lost

<<UnwantedIncident>>
Token Missed

<<Asset>>
Hardware Token

<<ThreatScenario>>
Attacker steals token

<<Asset>>
Service availability

<<UnwantedIncident>>
Token stolen

<<Asset>>
Software Token

<<Treatment>>
Revvoke token

<<ReduceConsequence>>

<<Treatment>>
Educate user

<<ReduceLikelihood>>

<<include>>

Figure 56 Overview diagram for SCT001
9 Comparing CORAS to the Howard & LeBlanc approach

In Chapter 4 of [4] Howard & LeBlanc present an approach to threat modelling. In this chapter we investigate how their approach relates to threat modelling with the CORAS language.

9.1 Evaluation criteria

One of the main objectives of this report is to “present ideas for simplifying threat modelling for a designer”. The answer to this objective was to apply the CORAS language on modelling threats identified by the approach of Howard & LeBlanc. Fulfilment of the requirement for “simplifying threat modelling” is assessed by comparing the two approaches with respect to the following evaluation criteria:

1. Structure: To what degree is structuring of and navigation in threat model documents supported?
2. Readability: Is a threat model document easily read and understood?
3. Usability: How much training is needed for creating (good) threat model documents?
4. Generality: How general is the approach? To what degree is unforeseen threat expressible?
5. Expressiveness: Is all information about threats that we want to capture possible to express?
6. Unambiguity: Do the terms and concepts have a clear meaning? To what degree can a threat be unambiguously modelled?
7. Simplicity: Is the complexity of the threat models sensible with respect to the threats that are modelled, or do they introduce unnecessary complexity making the models more complex than necessary?

9.2 Comparison of terms

The two approaches use basically the same terms, but there are some differences. In Table 8 the relations between terms used in the two approaches are explored.

Table 8 Terms and definitions

<table>
<thead>
<tr>
<th>CORAS</th>
<th>Howard &amp; LeBlanc</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>component</td>
<td>Entity in CORAS is more general than component in that it may represent more abstract things than just system components.</td>
</tr>
<tr>
<td>asset</td>
<td>asset/treat target</td>
<td>Assets in the CORAS approach are “things that have value for a stakeholder”, while in Howard &amp; LeBlanc they are the part or components of the system and their value is implicit. Hence, the CORAS assets are more general.</td>
</tr>
<tr>
<td>stakeholder</td>
<td>user</td>
<td>Howard &amp; LeBlanc do not explicitly state the stakeholders, but various groups of users seem to be the stakeholders in mind.</td>
</tr>
<tr>
<td>threat</td>
<td>threat</td>
<td>Threats in Howard &amp; LeBlanc are more general than in CORAS, see unwanted incident.</td>
</tr>
<tr>
<td>threat agent</td>
<td>attacker/hacker</td>
<td>In CORAS, attacker/hacker is a sub category of threat agent.</td>
</tr>
<tr>
<td>threat scenario</td>
<td>attack scenario</td>
<td>Threat scenarios are more general than attack scenarios, since they may be related to any threat agent, while attack scenarios are related to attackers.</td>
</tr>
<tr>
<td>vulnerability</td>
<td>vulnerability</td>
<td>Basically the same.</td>
</tr>
<tr>
<td>unwanted incident</td>
<td>threat</td>
<td>In Howard &amp; LeBlanc there is no distinction between threat and unwanted incident. The term</td>
</tr>
</tbody>
</table>
As we see from this comparison, there should be no overwhelming problems with using the terms of the two approaches interchangeably since the Howard & LeBlanc terms mean the same as or is specializations of the CORAS terms. One exception is threat which is more general in Howard & LeBlanc and therefore should be treated with some care. In CORAS a threat is a combination of threat scenario and threat agent, so a possible solution could be to say that CORAS threat does not correspond to something specific in Howard & LeBlanc and that CORAS unwanted incident correspond to Howard & LeBlanc threat.

### 9.3 Comparison of models

In the approach of Howard & LeBlanc, a threat model consists of:

- A decomposition of the system/application into key components using a graphical modelling language such as data flow diagrams or UML.
- For each threat a table listing:
  1. the asset/threat target,
  2. a classification of the threat according to STRIDE,
  3. attack scenarios presented in a threat tree,
  4. the risk related to the threat classified according to DREAD,
  5. a list of countermeasures/mitigation techniques.
- A list of hacker questions.

A threat tree (also called attack tree) is a specialization of a fault tree. A threat tree decomposes a threat into its basic steps, and may have a graphical or textual representation. The branches of a threat tree may be and-branches or or-branches, and the paths of the tree represent attack scenarios. A threat tree may be enhanced with mitigations tied to the leaf nodes.

A threat model in the CORAS profile consists of:

- An asset diagram.
- A threat & unwanted incident diagram.
- (A risk diagram.)³
- A treatment diagram
- (A treatment evaluation diagram.)³
- Overview diagram

In the following we describe similarities and differences of the two approaches.

#### 9.3.1 System decomposition

Howard & LeBlanc put emphasis on decomposing the system/application into its key components using a graphical modelling language like data flow diagrams or UML. There is no reason the CORAS language could not be used together with a UML system model being an extension to

---

³ Not treated in this report
UML. In this case the identified key components should have status as entities in the CORAS asset diagrams.

9.3.2 Assets
In particular, there are two ways the CORAS language is more explicit and general than the approach of Howard & LeBlanc: (1) In CORAS there is a distinction between entity and asset which means that the parts of the system considered to have value is explicitly stated, whereas in Howard & LeBlanc all identified components are implicitly considered assets. (2) In CORAS the stakeholders are modelled explicitly due to the observation that not all parts of a system need have value for all stakeholders. In Howard & LeBlanc there is no concept of stakeholder leaving it open who the threat model is constructed for; the users of the system, the developers, the company?

Another difference is that, in the CORAS language, vulnerabilities are modelled as part of the assets. In Howard & LeBlanc it is not specified how vulnerabilities should be documented, even though it seems as they may be used as nodes in the threat trees.

9.3.3 Threats & unwanted incidents
As shown in Section 9.2, in Howard & LeBlanc there is no distinction between threats and unwanted incidents. This means that both what are modelled as threats and as unwanted incidents in the CORAS language would be modelled as threats in the approach of Howard & LeBlanc. Since Howard & LeBlanc does not provide any explicit means for relating threats (except possibly clever use of threat trees) inclusion of threats and chains of events should be hard to express without resorting to ad hoc modelling.

Another difference between the two approaches is the level of which the threats (and unwanted incidents) are modelled. While the CORAS language focuses on the relations among threats and unwanted incidents and their relations to assets, the focus of Howard & LeBlanc is on the details of the threats. These ways of modelling threats (and unwanted incidents) are complementary rather than excluding each other. In the CORAS language threats and unwanted incidents are modelled by means of specialized use cases, and it is common practice to specify the details of use cases with other modelling techniques such as sequence diagrams, work flow diagrams and structured text. There is no reason the textual descriptions of threats used by Howard & LeBlanc, with or without threat trees, could be used as a way of specifying the details of threats and unwanted incidents modelled in the CORAS language.

9.3.4 Risks
We do not go deep into the discussion on representation of risks, but make the observation that DREAD is not a conflicting approach to the way risks are represented in the CORAS language, even though the standard way of calculating risks in CORAS is by consequence and frequency values.

9.3.5 Treatments
In Howard & LeBlanc treatments or mitigation techniques are listed in the textual descriptions of threats or attached to the leaf nodes of threat trees. The CORAS language is more flexible in that treatments are modelled by means of specialized use cases that may be related to any unwanted incidents and may have detailed specifications in the same way as threats and unwanted incidents.

9.3.6 Overview
Howard & LeBlanc has no equivalent of the possibility of making overviews that the CORAS language exhibits. Obviously it is possible to make all sorts of tables, queries and hyperlinks in
order to obtain overview and navigability in a threat document as long as an appropriate tool is at hand. The same holds for the CORAS language.

9.3.7 Hacker questions
The CORAS language has no equivalent of hacker questions, but there is no reason such questions could not be attached to threat agents as extra information.

9.4 Evaluation
In the following we compare the two approaches with respect to the evaluation criteria specified in Section 9.1.

1. Structure: The CORAS language has structuring mechanisms inherit through UML, while Howard & LeBlanc only have the structure of each threat description and unique threat identifiers. All ad hoc structuring mechanisms that may be allied to Howard & LeBlanc may also be applied to CORAS.

2. Readability: For a developer familiar with UML or similar modelling languages the CORAS language should be fairly easy to interpret. The use of stereotypes increases the readability by having easily understood icons representing the main concepts in the threat models. In Howard & LeBlanc the readability depends on the modeller’s ability to make good textual descriptions.

3. Usability: It is hard to see any difference in this respect, assuming the creators of the threat models is familiar with making system models/specifications in structured text and graphical modelling languages. As with structure this also depends on the tools available.

4. Generality: The approach of Howard & LeBlanc is practical oriented towards protection from attackers/hackers exploiting vulnerabilities of software. The CORAS language, however, is designed to model threats in general and is not restricted to intentional attacks, and may be specialized in any direction.

5. Expressiveness: For several reasons, the CORAS language may be seen as having more expressive power than Howard & LeBlanc: (1) Howard & LeBlanc have no explicit stakeholders. (2) The models of Howard & LeBlanc only implicit describe vulnerabilities. (2) Howard & LeBlanc have no way of relating threat/unwanted incidents. (3) The CORAS language is more general than Howard & LeBlanc in that not only threats originating in an attacker are expressible.

6. Unambiguity: The CORAS language has an abstract syntax defined by its meta model, and all terms and concepts are given definitions based on common standards for security and risk management [1]. The terms in Howard & LeBlanc are defined (p. 87), but the definitions are not as explicit stated as in CORAS. The apparent fit between the terms of CORAS and of Howard & LeBlanc indicates that this is not a big problem; however, there seems to be a bit of confusion in Howard & LeBlanc with respect to threat vs. unwanted incident vs. vulnerability. Also, the lack of stakeholders in Howard & LeBlanc may obscure the purpose of the threat models; who is it written for?

7. Simplicity: The CORAS language introduces concepts and relations that are not present in Howard & LeBlanc, e.g., unwanted incident and initiate, but this must be seen as a trade off with respect to the expressiveness of the language, and the apparent confusion with respect to some of the terms in Howard & LeBlanc can not be said to increase the simplicity. It may also be argued that the generality of the CORAS language makes it less simple, but the possibility of making domain specific specializations (possibly in combination with domain specific guidelines) of the language solves this problem.
10 Conclusions

The overall goal of the Microsoft threat modelling initiative has been to model the threats inherent in the different technologies used for the design of web applications. The idea is to ship these technologies not only with documentation on how to use them but also with threat models documenting how they can be misused. For this to be feasible, the guiding threat models must be easily read and understood. This report demonstrates the suitability of the CORAS language for this purpose in relation to Buffer overflow, Web Services, ASP.NET, SQL Server, Active Directory and SmartCards.

Comparing the CORAS language with the threat modelling technique described by Howard & LeBlanc we see a lot of similarities, but also some clear differences. In particular, we make the following observations:

- The CORAS language introduces some concepts and features making it more expressive than Howard & LeBlanc, e.g., unwanted incident, initiate and stakeholder.
- The CORAS language is based on familiar UML notation enhanced with intuitively understandable symbols representing the basic concepts of threat modelling. Both these features increase the readability.
- The CORAS language is more general than Howard & LeBlanc.
- The CORAS language is more flexible and has more and stronger structuring mechanisms.
- The basic terms and concepts are sufficiently similar for the two approaches to be combined without too much effort.
- The threat models of Howard & LeBlanc would constitute a useful complement to the CORAS language.
References

## Appendix A  Terms and Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Directory</td>
<td>Active Directory lets organizations efficiently share and manage information about network resources and users.</td>
</tr>
<tr>
<td>Activity diagram</td>
<td>State machine like UML diagram intended to model computations and workflows.</td>
</tr>
<tr>
<td>ASP.NET</td>
<td>The framework for building server based Web applications. Microsoft's next generation of the ASP (Active Server Pages) language.</td>
</tr>
<tr>
<td>Asset</td>
<td>Assets in the CORAS approach are “what have value for a stakeholder”, while in H&amp;L they are the part or components of the system and their value is implicit.</td>
</tr>
<tr>
<td>Authentication</td>
<td>Verification of the identity of a user or computer for security purposes.</td>
</tr>
<tr>
<td>Class diagram</td>
<td>UML diagram, graphic presentation of the static model elements such as classes, types and their contents and relationships.</td>
</tr>
<tr>
<td>CORAS</td>
<td>Name of the EU project from which the development of the CORAS profile and the CORAS security analysis was initiated [3].</td>
</tr>
<tr>
<td>CORAS profile</td>
<td>UML profile for security assessment developed from the CORAS project [7].</td>
</tr>
<tr>
<td>CORAS security analysis process</td>
<td>Security analysis process as initially defined in the CORAS project [3].</td>
</tr>
<tr>
<td>DREAD</td>
<td>Method to range threats used for calculating risks defined by Howard &amp; LeBlanc [4]. Acronym from the terms: Damage potential, Reproducibility, Exploitability, Affected users, Discoverability.</td>
</tr>
<tr>
<td>Eavesdropping</td>
<td>Surreptitious interception of information sent over a network by an entity for which the information is not intended.</td>
</tr>
<tr>
<td>Masquerade</td>
<td>Claim to be a different entity.</td>
</tr>
<tr>
<td>Reverse Engineering</td>
<td>The procedure of carefully dismantling and inspecting an existing product to look for engineering or design features that can be incorporated into one's own product.</td>
</tr>
<tr>
<td>Risk</td>
<td>The chance of something happening that will have an impact upon objectives. It is measured in terms of consequence and likelihood.</td>
</tr>
<tr>
<td>Sequence diagram</td>
<td>UML diagram, showing communication between instances through messages arranged in a time sequence.</td>
</tr>
<tr>
<td>Smart Card</td>
<td>Credit card sized plastic card with a special integrated circuit (IC) chip imbedded in the surface of the card. The IC chip stores information in an electronic form. There are two principal kinds of smart cards, microprocessor and memory.</td>
</tr>
<tr>
<td>SOAP message</td>
<td>SOAP (Simple Object Access Protocol) messages are XML based messages for exchanging structured and typed information.</td>
</tr>
<tr>
<td>Spoofing</td>
<td>Pretending to be someone else. The deliberate inducement of a user or a resource to take an incorrect action. Attempt to gain access to an AIS by pretending to be an authorized user. Impersonating, masquerading, and mimicking are forms of spoofing.</td>
</tr>
<tr>
<td>SQL Server</td>
<td>Microsoft’s enterprise level database product, providing a database to run dynamic web sites.</td>
</tr>
<tr>
<td>Stakeholder</td>
<td>Those people and organizations who may affect, be affected by, or perceive themselves to be affected by, a decision or activity.</td>
</tr>
</tbody>
</table>
**STRIDE**  

**Threat**  
A potential cause of an unwanted incident which may result in harm to the assets of an information system.

**Treatment**  
Selection and implementation of appropriate options for dealing with risks.

**UDDI**  
Universal Description, Discovery and Integration protocol - A directory model for web services. UDDI is a specification for maintaining standardized directories of information about web services, recording their capabilities, location and requirements in a universally recognized format.

**UML**  
Unified Modelling Language - an Object Management Group (OMG) standard for graphical modelling of software artefacts [6].

**Unwanted incident**  
An incident doing harm to the assets of an information system.

**Use case diagram**  
UML diagram, diagram that shows the relationships among actors and use cases within a system.

**Vulnerability**  
A weakness, design, or implementation error that can open for a threat to cause an unwanted incident, compromising the security of the system.

**Web Services**  
A Web Service is application or business logic that is accessible using standard Internet protocols. Web Services combine the best aspects of component based development and the World Wide Web. Like components, Web Services represent black box functionality that can be used and reused without regard to how the service is implemented.

**WebServer**  
A computer program that serves the requested files which form web pages to the user's browser (client). A web server can also refer to the computer that runs the server software and holds the files for one or more web sites.

**WSDL**  
Web Services Description Language - a XML based language that describes an interface of a web service together with information how to call the web service and where to find it.

**XML**  
eXtensible Markup Language - the universal format for structured documents and data on the Web. XML is an industry standard protocol administered by the World Wide Web Consortium (W3C) [8].
Appendix B  The CORAS profile

This Appendix presents the UML profile that defines the CORAS language. The aim of model based risk assessment is to integrate established risk analysis methods, like HazOp, FTA and FMEA, with UML modelling in a framework for conducting risk assessments of dependable IT systems.

An important motivation for the metamodel is the practical use of UML to support risk management in general, and risk assessment in particular. In model based risk assessment, UML models are used for three different purposes:

- *To describe the target of evaluation at the right level of abstraction.* A proper analysis of technical system documentation is not sufficient; a clear understanding of system usage and its role in the surrounding organization or enterprise is just as important. UML allows these various aspects to be documented in a uniform manner.

- *To facilitate communication and interaction between different groups of stakeholders involved in a risk analysis.* One major challenge when performing a risk analysis is to establish a common understanding of the target of evaluation, threats, vulnerabilities and risks among the stakeholders participating in the analysis. This motivates a UML profile aiming to facilitate improved communication during risk analyses, by making the UML diagrams easier to understand for non experts, and at the same time preserving the well definedness of UML.

- *To document risk analysis results and the assumptions on which these results depend to support reuse and maintenance.* Risk analyses are costly and time consuming and should not be initiated from scratch each time we assess a new or modified system. Documenting analyses using UML supports reuse of analysis documentation, both for systems that undergo maintenance and for new systems, if similar systems have been assessed before.

Risk Analysis Metamodel

The metamodel is divided into five submodels (Figure 57) that support different stages of a risk analysis. A risk analysis always starts with identifying the context of the analysis. A strengths, weaknesses, opportunities and threats (SWOT) analysis may be part of this. After the context has been established, the reminder of a risk analysis can be divided into identification and documentation of unwanted incidents, risks and treatments. This process is in accordance with the risk management process of [1].

The unwanted incident model is concerned with organizing and documenting the threats and vulnerabilities that open for incidents that may harm the system. The risk model quantifies unwanted incidents with respect to the reductions of asset value that they may cause. The treatment model supports documenting ways of treating the system and quantifying the effect of treatments with respect to reducing the potential harm of risks.
Figure 57 Submodels in the Risk Analysis Metamodel

Context
This submodel (Figure 58) defines the context of a risk analysis. The context consists of the stakeholders and assets of the system under analysis, which all further analysis is based on.

Figure 58 Context submodel
A risk analysis is asset driven, which means that analysis is carried out relative to the identified assets. In the general case, an asset may be anything that stakeholders of the system under analysis find to have value.

Each asset may only be related to one stakeholder and should have an unambiguous value assigned by one stakeholder. If two stakeholders view the same entity as an asset, the entity should be documented as two different assets related to the same entity. Two assets are per definition different if valued by different stakeholders. Both the values and the reasons for the valuing may be different.

Below the concepts of the models are described:
Stakeholder. A person or organization that has interests in the assessed system.
Policy. A rule or regulation defined by a stakeholder, related to the system under analysis. A policy could relate to security aspects like confidentiality, integrity, availability, non repudiation, accountability, authenticity and reliability, and should provide directions for the analysis.
RiskEvaluationCriterion. A criterion that identified risks are evaluated against in order to decide whether the risk is acceptable or not.
Asset. A part or feature of the system that has value for one of the stakeholders, for example the quality level of a service.

Entity. A physical or abstract part or feature of the system under analysis that becomes an asset when assigned value by a stakeholder, for example a service provided by the system.

Asset Value. The value assigned to an asset by a stakeholder.

Value Definition. Definition of value types for various values used in a risk analysis, such as asset value.

SWOT

Strengths, weaknesses, opportunities and threats (SWOT) analysis is a part of establishing the context of a risk analysis. However, SWOT is used for pointing out general directions of the analysis, and its results are only indirectly used in the further analysis. For this reason the concepts of the submodel for SWOT, shown in Figure 59, are not strongly connected to the rest of the metamodel.

Figure 59 SWOT submodel

A SWOT analysis is concerned with identifying the strategic context of the organization carrying out a risk analysis. The elements of the SWOT model are described below:

Enterprise Asset. Asset of the organization from a strategic point of view.

Enterprise Strength. A strategic strength of the organization.

Enterprise Weakness. A strategic weakness of the organization.

Enterprise Opportunity. A strategic opportunity of the organization.

Enterprise Threat. Something that threatens the strategic position of the organization.

Unwanted incident

Identification and documentation of unwanted incidents is concerned with exploring the threats and vulnerabilities of the system under analysis, and how threats and vulnerabilities may combine and lead to potential incidents that can harm the system.

In the metamodel of Figure 60, the notion of threat is decomposed into a threat agent that initiates one or more threat scenarios.
Figure 60 Unwanted incident submodel

The concepts of Figure 60 are described below:

**ThreatAgent.** A potential cause of an unwanted incident, which may result in harm to a system or organization and its assets. Threat agents may be external, (e.g., hackers or viruses) or internal (e.g., system failures or disloyal employees).

**ThreatScenario.** A description of how a threat may lead to an unwanted incident.

**Vulnerability.** A weakness with respect to an asset or group of assets that can be exploited by one or more threats.

**UnwantedIncident.** An undesired event that may reduce the value of an asset.

**Initiate.** An unwanted incident may lead to another unwanted incident. Initiate is a relation for modelling that between an unwanted incident acts as an initiator of another unwanted incident.

**Threat agent**

The treat agent submodel is presented in Figure 61. This model defines a number of specialisations of the concept of treat agent. The purpose of this model is to support modelling of the variety of threat agent known from literature.

Figure 61 Treat agent submodel

**HumanThreat.** A person that, with or without intent, potentially may act in a way that leads to unwanted incidents.

**Attacker.** An intelligent threat agent that carries out an assault on system security, i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system.
Intruder. An entity that gains or attempts to gain access to a system or system resource without having authorisation to do so.

Eavesdropper. A person that does a passive wiretapping done secretly, i.e., without the knowledge of the originator or the intended recipients of the communication.

Man-in-the-middle. A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data in order to masquerade as one or more of the entities involved in a communication association.

Insider. An entity inside the security perimeter, i.e., an entity that is authorised to access system resources but uses them in a way not approved by those who granted the authorisation.

System Threat. A part of the system under assessment that potentially may act in a way that leads to unwanted incidents.

Hardware Failure. An error in hardware that may constitute a threat.

Software Failure. An error in software that may constitute a threat.

Malicious Software. Software made with the intent of harming computerised systems.

Virus. A hidden, self-replicating section of computer software, usually malicious logic, that propagates by infecting – i.e., inserting a copy of itself into and becoming part of – another program.

Worm. A computer program that can run independently, can propagate a complete working version of itself onto other hosts on a network, and may consume computer resources destructively.

Zombie. A program that secretly takes over another Internet attached computer and uses that computer to launch attacks that are difficult to trace to the zombie’s creator.

Trojan Horse. A computer program that appears to have a useful function, but also has a hidden and potentially malicious function that evades security mechanisms, sometimes by exploiting legitimate authorisations of a system entity that invokes the program.

Logical Bomb. Malicious logic that activates when specified conditions are met. Usually intended to cause denial of service or otherwise damage system resources.

Trap Door. A hidden computer flaw known to an intruder, or a hidden computer mechanism (usually software) installed by an intruder, who can activate the trap door to gain access to the computer without being blocked by security services or mechanisms.

Risk
A risk is an unwanted incident that has been assigned consequence and frequency values. These values are used for calculating a risk value, which represent loss of asset value of the asset the risk is related to. Risks that in some way are related or similar may be categorized into risk themes. A risk theme is itself assigned a risk value based on the risks it contains and is treated like a singular risk with respect to evaluation and treatment.

Risk values are evaluated by risk evaluation criteria defined in the context of the risk analysis. A risk evaluation criterion states which risk values are acceptable, and which are not – implying the need for treatment.
Figure 62 Risk submodel

The concepts of the submodel of Figure 62 are described in the following:

**AbstractRisk.** The common properties of risks and risk themes, such as risk value.

**Risk**. An unwanted incident that has been assigned a consequence value, a frequency value, and a resulting risk value. Threats, vulnerabilities and unwanted incidents may go to several assets, but since a risk may reduce the value of an asset, a risk is only related to one particular asset.

**RiskTheme.** A categorization of similar risks, assigned its own risk value.

**RiskRelationship.** The relation between risks or risk themes.

**RiskEvaluation.** The assignment of a risk or a risk theme to the unwanted incident it evaluates with respect to risk value.

**Consequence.** The consequence of an unwanted incident happening, relative to an asset.

**Frequency.** The probability of an unwanted incident occurring within a given period of time.

**RiskValue.** A value assigned to a risk, reflecting the loss of asset value that the risk represents.

**Treatment**

The treatment model (Figure 63) is concerned with documenting and evaluating ways of providing treatments to the system under analysis in order to reduce the value of risks. A treatment may apply to several unwanted incidents. However, when a treatment’s capability to reduce risk value is assessed, this is with respect to a single risk or risk theme.
Figure 63 Treatment submodel

The concepts of Figure 63 are described below:

*Treatment*. Ways of reducing the risk value of a risk or risk theme.

*TreatmentEffect*. A treatment’s capability to reduce the risk value of a particular risk.

*TreatmentEvaluation*. The assignment of a treatment effect to the treatment it evaluates.

*RiskReduction*. The value of a treatment effect, i.e., the concrete reduction of a value of a risk.

*TreatmentOption*. Main classes of providing treatment, and hence the relation between a treatment and the unwanted incident it applies to. The options are *avoid, reduce consequence, reduce likelihood, and transfer (of risk)* [1].
Appendix C  Icons used in the CORAS profile
This appendix presents a list of icons that are used when modelling threats with the CORAS profile.

Context
Table 9 presents icons for the context subprofile.

Table 9 Icons for the Context Subprofile

<table>
<thead>
<tr>
<th>Icon</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Stakeholder&gt;&gt;</td>
<td>![Image]</td>
</tr>
<tr>
<td>&lt;&lt;Asset&gt;&gt;</td>
<td>![Image]</td>
</tr>
<tr>
<td>&lt;&lt;Entity&gt;&gt;</td>
<td>![Image]</td>
</tr>
<tr>
<td>&lt;&lt;Ownership&gt;&gt;</td>
<td>![Image]</td>
</tr>
</tbody>
</table>

SWOT
Table 10 presents icons for the SWOT subprofile.

Table 10 Icons for the SWOT Subprofile

<table>
<thead>
<tr>
<th>Icon</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;EnterpriseAsset&gt;&gt;</td>
<td>![Image]</td>
</tr>
<tr>
<td>&lt;&lt;EnterpriseStrength&gt;&gt;</td>
<td>![Image]</td>
</tr>
<tr>
<td>&lt;&lt;EnterpriseWeakness&gt;&gt;</td>
<td>![Image]</td>
</tr>
<tr>
<td>&lt;&lt;EnterpriseOpportunity&gt;&gt;</td>
<td>![Image]</td>
</tr>
<tr>
<td>&lt;&lt;EnterpriseThreat&gt;&gt;</td>
<td>![Image]</td>
</tr>
</tbody>
</table>

Unwanted incident
Table 11 presents icons for the unwanted incident subprofile.
Table 11 Icons for the Unwanted incident Subprofile

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;ThreatAgent&gt;&gt;</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>&lt;&lt;ThreatScenario&gt;&gt;</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>&lt;&lt;Vulnerability&gt;&gt;</td>
<td>No icon</td>
</tr>
<tr>
<td>&lt;&lt;UnwantedIncident&gt;&gt;</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>&lt;&lt;Initiate&gt;&gt;</td>
<td>No icon</td>
</tr>
</tbody>
</table>

**Risk**

Table 12 presents icons for the risk subprofile.

Table 12 Icons for the Risk Subprofile

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Risk&gt;&gt;</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>&lt;&lt;RiskTheme&gt;&gt;</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>&lt;&lt;RiskThemeRelationship&gt;&gt;</td>
<td>No icon</td>
</tr>
<tr>
<td>&lt;&lt;RiskEvaluation&gt;&gt;</td>
<td>No icon</td>
</tr>
</tbody>
</table>

**Treatment**

Table 13 presents icons for the treatment subprofile.

Table 13 Icons for the Treatment Subprofile

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Treatment&gt;&gt;</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>&lt;&lt;TreatmentEffect&gt;&gt;</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>&lt;&lt;Avoid&gt;&gt;</td>
<td>No icon</td>
</tr>
<tr>
<td>&lt;&lt;ReduceConsequence&gt;&gt;</td>
<td>No icon</td>
</tr>
<tr>
<td>&lt;&lt;ReduceLikelihood&gt;&gt;</td>
<td>No icon</td>
</tr>
<tr>
<td>&lt;&lt;Transfer&gt;&gt;</td>
<td>No icon</td>
</tr>
<tr>
<td>&lt;&lt;TreatmentEvaluation&gt;&gt;</td>
<td>No icon</td>
</tr>
</tbody>
</table>