NTNU
Norwegian University of
Science and Technology

Engineering secure software

Investigating the relationship between requirements and design

Amund Mortensen

Master of Science in Informatics
Submission date: June 2009
Supervisor: Guttorm Sindre, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem description

The need for integrating security concerns in mainstream software engineering has received increasing attention lately. Several approaches have emerged that try to include security threats in modeling techniques that were previously meant for general systems modeling rather than security modeling. On the requirements level, some examples are abuse cases and misuse cases, i* / Secure Tropos, and abuse frames, and there is also research into taxonomies of security-related requirements which can be used for instance as checklists in the specification phase. On the design level, UMLsec is an adaptation of UML targeted specifically towards security, and security patterns are design patterns typically addressing various security goals.

The objective of this project is to look into the transition between requirements and design in more detail: given requirements models in one or more of the above-mentioned languages, possibly supplemented with plain textual security requirements, what guidelines can be given for choosing an appropriate design, for instance in the form of security patterns? Two possible outcomes can be identified: (1) a systematic description of the guidelines, and (2) if time allows, the implementation of the guidelines in a tool providing semi-automatic assistance to the software engineer in the selection and elaboration of a design given certain security requirements.

Preface

This thesis is the result of one year of work, and concludes the requirements for a Master's degree in informatics at the Norwegian University of Science and Technology (NTNU). The assignment was given by the department of Computer and Information Science in cooperation with SINTEF. The supervisors of the thesis were Per Håkon Meland from SINTEF and Guttorm Sindre from NTNU.

The pursuit of a master's degree has been a journey – a chapter in life that soon is over. Thorough these years of education, I've experienced and learned a lot and I would not trade it for anything. I would like to thanks NTNU for the wonderful time, and the opportunity to take parts of my master degree in Singapore where I had an incredible time and meet many lovely people.

I would like to thank my supervisors for their valuable input and sharing of knowledge. Without them, this master thesis would not be the same.

I would also like to thank my friends and family for their help and support during my education. A special thanks to the guys at Gribb for all their help and contribution, thanks for a wonderful time!

Last, but not least, a special thanks to my cohabitant Mari Harby for believing in me and having the patient and tolerance thorough these years.

Abstract

Software security costs the society great deal of money each year. Considerable amount of research has been conducted on the subject. The conclusion has been that it values to start early and doing it right the first time.

As it values to focus on security at an early stage, we want to investigate the transition between requirement and design in more detail. How can information from the requirement phase be used more substantially to produce guidelines when designing the system.

Security requirements are one way to define what security measures a system should have. As poor requirement engineering costs a great deal of money, it values to do it right the first time, and create requirements with high quality.

Pattern is one way to transfer expert knowledge. In this master thesis, we are investigating the theory that patterns could be used when eliciting security requirements by utilizing reusable requirements. Also, to employ patterns that way, a mapping to Security Design Patterns (SDPs) could be created. The mapping could be used as guidelines later in the design phase.

To test the theory, we implemented a tool as a web application. We conducted an experiment to investigate the hypotheses and how it performed with users in a controlled environment.

From the experiment we concluded that the theory is plausible, but more research have to be conducted to investigate how useful it is.

Keywords: Software security, Requirement and design phase, Reusable security requirements, Patterns, Security requirements, Security design, Tool.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we introduce this master thesis and presents the goal with the research, and the approach to accomplish them.

Thorough this thesis some terminology specific to software security are used. A short description of the most used once are listed in Appendix A and could be employed as a reference when necessary. Also, Appendix B includes a list of the acronyms used.

First an introduction of the motivation with this thesis is introduced in Section 1.1. Next, in Section 1.2 our goal with the research is presented. Further, in Section 1.3 our approach to accomplish the goals is outlined. At last, in Section 1.4 we present the outline of the rest of the report.

1.1 Background and Motivation

Earlier, security was thought of as a network problem where firewalls and system administration were the solution. Still when it came clear that this was not enough, software security was treated as an aftertouch with low priority. The result was patched up systems with high maintenance costs [27].

Software security is a hot topic these days as the problem has grown with time [46]. Figure 1.1 lists the costs of various computer security incidents in the resent years. From the figure, a peak from 2000 to 2003 can be seen. Since then, the costs appear to have been lowered and gotten more stable [46].

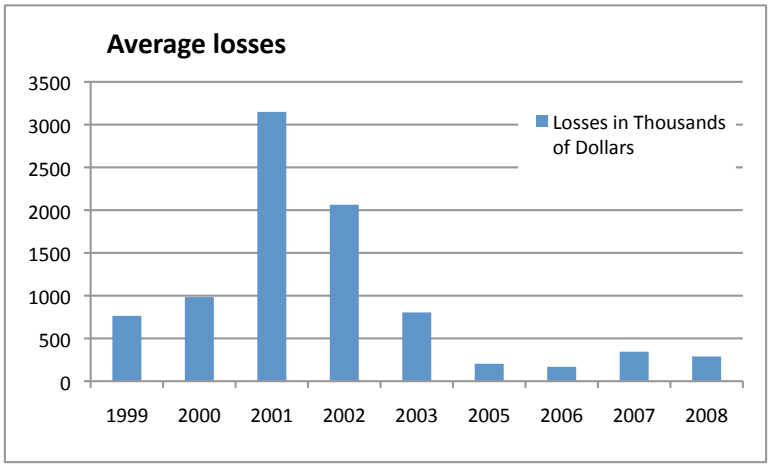


Figure 1.1: Average losses caused by computer security incidents in the resent years [46].

Because the damage costs look more stable these days, it is tempting to conclude that we have learned and are taking enough measures as it is. Still according to the survey by the Computer Security Institute, the total cost of computer security incidents in 2008 were as high as 289 000 US dollars per respondent [46]. The number is frightenly high, and we could save the society a great deal if we could lower it.

The survey by the Computer Security Institute lists four categories that have the highest incidence reports [46]. The four categories are viruses, insider abuse, laptop theft and unauthorized access to systems. It has been brought to light that few developers follow security coding practices to produce more secure code, which implies that the number of flaws grows in the same phase as the code base [27, 33]. Some of the categories listed by the survey are software related. This indicates that by creating more secure software, we could lower the number of incidents in these categories.

How can we create more secure software? For starters, it pays of to do things right the first time [35]. Security awareness is also important. A major factor in the security unawareness problem is a lack of focus in the students education. Howard [27] has identified that the education has more focus on preventive mechanisms, than looking at "security as threats".

Software is both costly and difficult to correct, which also indicates that we should start as early as possible to incorporate security into the development cycle. Studies have shown that reworking requirements, design and coding defects on most software development projects costs 40 to 50 percent of total project effort [36]. Also, as much as 25 to 40 percent of the budget goes to

requirement defects [36]. Mead and Stehney [38] have also identified that requirement problems are a significant cause of why projects fails.

Also, a gap has been identified between the security requirement and design phase. Several approaches that were previously meant as a general modeling techniques have been altered to include security information. The problem is that developers are primarily concerned with building a system that works. The non-functional security requirements are not taken seriously, and the security experts and developers are only getting frustrated at each other. The problem lies in the transition between the security requirement and design phase, where the security information gathered in the first phase is either lost, or the information is difficult to utilize [15, 42].

As developers and security experts have different goals and prioritization, we need a way to transfer knowledge between security experts and developers in an easy fashion. Patterns are a way of dealing with reusable solutions to a common problem, which means that novices can benefit from know-how and skills of experts [26, 51]. Also, the structure of a pattern is already known by the developers. Patterns could be used to bridge the knowledge gap between security experts and developers and make recommendations in the design phase [15, 42].

1.2 Goal

From the problem description outlined at the start of this master thesis, the objective of this research is to look into the transition between the requirement and design phase in more detail. The goal of doing this, is to look at what guidelines can be given for choosing an appropriate design, for instance in form of security patterns.

When conducting the pre study, we discovered the potential of using patterns. Not only could patterns be used in the design phase, where they first grew popular in the software industry [26], but also represent security requirements [57, 58].

As more research were conducted, we got further insight into the topic and got more and more eager to try out the following theory; that patterns could be used when eliciting security requirements and also have a mapping to Security Design Patterns (SDPs) to be used as guidelines in the design phase. We did not find other research using security requirements like this, other than work that had recognized the potential of reusable requirements that is outlined in chapter 3.

Investigating the assumptions would also provide information on how security requirements could be considered as patterns, and how they could be utilized in the elicitation process. Because of this, we added it as one of the goals with this thesis.

The following is a summary of the research goals:

- Investigate the transition between security requirements and design in more detail.
- Study how security patterns could be used when eliciting security requirements.
- Consider if a mapping between Security Requirement Patterns (SRPs) and SDPs could be used as guidelines in the design phase.

1.3 Approach

The approach to explore the goals are described thorough this master thesis. Briefly outlined, we investigated the transition between the security requirement and design phase in the pre study. The most important parts are recapitulated in chapter 3 – Literature Review.

The approach for the last two goals were to create a tool to bring the theory into life and investigate it by doing an experiment. As an approach to find more information about the usefulness of a tool, and answer the goals with the thesis, we created the hypotheses listed in Table 1.1.

To make a distinction between the theory behind the research goals and the hypotheses in Table 1.1, we have tried to refere to them as theory and hypotheses respectively thorough the thesis.

The tool was basically a web application with easy access in mind. The thought was by creating a web application, we could more easily get users to participate testing the tool, we could reach a broad user group, and the tool could be demonstrated to the public. As the research process went further, we saw the need of having a more controlled experiment.

To test usefulness and the theory implemented in the tool, we carried out an experiment. The aim of the experiment was to make users test the tool with fictitious cases and compare the results with how they performed with and without the tool. We got twelve participants to do two different cases, one using the tool and the other one with a manual method. The data were gathered by using questionnaires and the results from the tasks. The data

Table 1.1: The hypotheses about the usefulness and validity of the theory.

	Using data from the questionnaire
H1	It seems easier and quicker to find security requirements and Security Design Patterns (SDPs) with a tool.
H2	The quality seems better when finding security requirements and SDPs with a tool.
H3	Users have a higher confident when using a tool to find security requirements and SDPs.
	Using data from the tasks
H4	Security requirements found with the tool has better quality.
H5	Security requirements found with the tool has better security coverage.
H6	SDPs found with the tool have better security coverage.

were analyzed to see if there was any difference using the tool compared to do it manually.

A live demonstration of the tool can be found at "http://myrequirements.idi.ntnu.no" and the source code is available at "http://amundmo.github.com/myRequirements/" under the GNU General Public License (GPL) license.

1.4 Outline of the Document

This master thesis is divided into chapters that presents the content that is outlined thorough the document. Each chapter presents some area of the research and starts with a short introduction and a summary of the content.

The chapters are as follows:

Chapter 1 (this chapter) introduces the thesis. It summarizes background and motivation of conducting the research and the goals and approaches we had.

Chapter 2 describes the theory of conducting research and discusses how the research process was conduced accordingly. The chapter also describes how we developed the tool, conducted the experiment, gathered data and analyzed the results.

Chapter 3 outlines the most important literature in the security field that has some relevance to the research. The chapter presents how software development should be conducted with security in mind. Then it presents information about security requirements and security modeling methods. At last the chapter gives a description of security patterns.

Chapter 4 presents the tool we implemented as an approach to satisfy the goals. The chapter first present the requirements for the tool, then outline the functionality of the tool and how the requirements are fulfilled. At last a summary considering the completeness of the tool is made.

Chapter 5 gives an overview of the results that were gathered from the empirical evaluation. Participants background, security coverage, data quality and participants experience are presented.

Chapter 6 evaluates and discusses the findings from the results presented in chapter 5. The chapter first discuss each of the results accordingly, then present the results of the hypotheses, and at last discuss the knowledge gained from the results.

Chapter 7 concludes the master thesis and suggests further work.

References presents the bibliography list.

Appendices lists the supplementary material of the thesis. The material includes acronyms and terminology used in the thesis, as well as the assignment and questionnaires from the experiment. At last the data populated in the tool is presented as it describes how the data were mapped and categorized.

Chapter 2

Research Method

Research is an intellectual process of investigation to gain more knowledge. Generally, research follow a well-defined structure with a set of standard activities. A research method is the structure that defines how to do the research, and is a valuable way of knowing that we are doing it right. If the research method can be doubted, so can the result [44].

In this chapter, we first, in Section 2.1 give an overview over the different theories that apply when doing research. Afterwards, in section 2.2 we describe how we conducted the research process.

2.1 Theory

In this section we present the theoretical backbone of research. First in Section 2.1.1, we outline how the research process should be conducted. Afterwards, in Section 2.1.2 we outline some of the philosophical paradigms. Then, in Section 2.1.3, we go through some of the strategies that can be used. At last, in Section 2.1.4, we present data collection and analysis methods.

2.1.1 The Research Process

The research process is the series of steps that make up the research from start to finish. Oates[44] has created a model of the research process, which is described in Figure 2.1. The model describes the different stages that we will go through while doing research. The main steps in the model are experience and motivation, research questions, literature review and conceptual framework, strategies, data generation methods and data analysis. The

model also explains different methods that could be used in the individual phases answering the research questions.

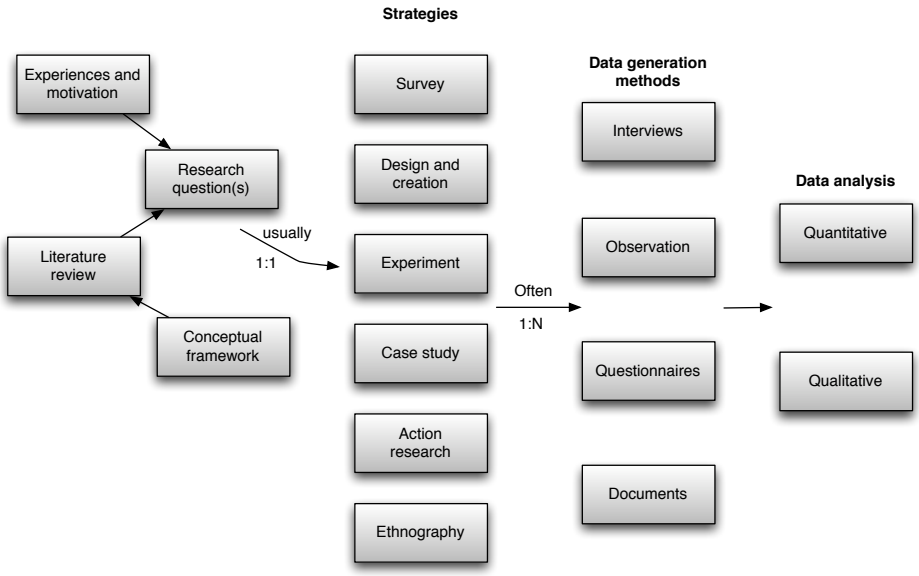


Figure 2.1: Model of the research process [44]

Alternative to the model in Figure 2.1 exists. One of them are the conceptualize, operationalize, generalize model [44]. The focus in this model is to use the three activities; conceptualize, operationalize and generalize to plan and conduct the research [44]. Another interesting fact is that there can be drawn an analogy to the Software Development Life Cycle (SDLC) described in Section 3.1 [44]. Both systems are used to plan and conduct a series of steps to create something new. System development creates new software, while research creates new knowledge.

2.1.2 Philosophical Paradigms

A philosophical paradigm is the fundamental view of the researcher; how he interprets and acquires knowledge about the world. The mostly known and used paradigm, is the *positivism*. This is the paradigm that the researcher most likely is using if nothing else is stated. It has been very successful of finding out the nature of the world we live in, but it has limitation when trying to study the social world. The social world includes people, how they behave and act, and things they can modify [44].

Other paradigms that have got much attention are the *interpretivism* and

constructivism. The constructivism paradigm is not as widely used as the others. The paradigm argues that the knowledge that we got are generated by our experience, and should therefore be treated that way. Interpretive studies does not use a hypothesis that need to be proved or disproven, but try to identify, explore and explain the social factors. This has the major difference that e.g. reliability is treated differently. In positivist research the argument is centered on repeatability, that an experiment can be repeated and get the same result. In interpretivism, it is argued that individuals construct the case being studied, so the same situation is unlikely to occur in a repeated study [44, 25].

The major differences between paradigms are their interpretation of:

- The different view about the nature of the world(ontology).
- How to acquire knowledge(epistemology).

Other paradigms that have been drawn forward by Hevner et al. [25], are the behavioral-science and design-science paradigm. In contrast to the philosophical paradigms that is discussing where "the truth" comes from. The behavioral-science paradigm seeks to find "what is true", while the design-science paradigm seeks to create "what is effective". Both paradigms are argued to be needed to ensure the relevance and effectiveness of information systems [25].

What philosophical paradigm to choose is important, because it is the fundamental view of the researcher; how he interprets and acquires knowledge about the world. If the philosophical paradigm can be doubted, so can the result [44, 25].

In the thesis, we have followed the positivism paradigm. We concluded that the other methods does not suffice acceptable in the type of research we will conduct.

2.1.3 Strategies

A strategy is a systematic plan of how to go forward with research. When conducting research, it is important to have a strategy because it defines the overall approach to answering the research question [44]. Oates [44] has six strategies listed in Figure 2.1 that are described as following:

- Survey - obtaining and analyzing a set of data from a large group of people or events in a standardized and systematic way.

- Design and Creation - developing a new IT-product or artifacts. It could also be elements of the development process such as new construct, model or method.
- Experiment - investigating cause and effect relationships or seek to prove or disprove a hypothesis.
- Case study - is focusing on one instance that is to be investigated. e.g. a information system. The aim is to obtain a rich detailed insight into that instance.
- Action research - focuses on research into action. The researcher does something, and afterwards investigates the outcome.
- Ethnography - understanding the culture and ways of seeing something, e.g. particular group or people.

Each strategy has their advantages and disadvantages, and which strategy to choose depends on what being researched on and what data to collected.

In our research, we used the strategy of design and creation as the main strategy to create new knowledge. We also conducted an experiment to test the theory implemented in the tool. The experiment put to use an survey to collect information from the participants in the experiment.

We will hence give an brief rundown of the strategies in the next sub sections.

Design and Creation

The design and creation strategy are focusing on developing a new IT product, also called artifacts. The strategy involves analyzing, designing and developing a new product that run on computers, e.g. a web site, application, animation etc.

When using this strategy, it is equally important to contribute to knowledge in some way. How to contribute to knowledge, depends on what the product does. Oates [44] has some examples of products that by it self contributes to knowledge:

1. A product that automates something that previously was done manually. The researcher can argue that the product is better.
2. A product that incorporate a new theory. The researcher can argue for the relevance and utility of the theory.

When employing the strategy, it is important to explain and document the Software Development Life Cycle (SDLC) of the project. How the work has been preceded through the stages of the project; analysis, design, implementation and testing.

Experiment

In research, an experiement is a strategy that investigates the relationship between cause and effect. An experiement is trying to disprove or prove a hypothesis. A hypothesis is usually based on some theory the researcher want to investigate further empirically.

The result gathered in the experiement should be based on variables. The researcher should determine what variables could be used and how they can be used to determine a result. The researcher can control the variables, either all at once, or in sequence to compute important parts.

Variables can be divided into dependent and independent variables. Dependent variables are affected by other variables, while dependent variables are not. How variables are affected, is important for the outcome of the result to be valid.

Experimental design is also important to ensure that the independent variables are not affected by possible external factors. There are many different kinds of experimental designs. Some examples are the "One Group Test" [44], "Static Group Comparison" [44] and the "Latin Square Design" [10]. Which one to choose depends on what are research on, what variables that are included, what bias could be presented and so on [44].

To give a confirm conclusion in an experiement, the hypotheses must be repeated many times and get the same results, also by other researchers.

Survey

A survey is a strategy that is used to obtain and analyze a set of data from a large group of people or events in a standardized and systematic way. The strategy is mostly associated with the positivism explained in Section 2.1.2. The data can be gathered from interviews, observations, questionnaire or documents. When the data is obtained, we can look for patterns and clues in the data that might be of importance [44, 17].

Planning and conducting of surveys can be divided into the following activities: data requirements, data generation method, sampling frame, sampling

technique, response rate and non-responses and sample size. When planning and conducting the survey, it is important to go through each of the activities. The following is a brief outline of the information Oates [44] presents of each of them:

1. Data requirements – are the activity where we find out what data to collect. When doing a survey, we normally only get one opportunity to collect the data, so it is important to have a good understanding of what data we wish to generate.

2. Data generation method – is the strategy of how to gather the data. As previously noted, there are many ways to collect data. There are basically two main categories of data; qualitative data and quantitative data. Both will be investigated further in Section 2.1.4.

3. Sampling frame – is the list or collection of what we are gathering the data from – mostly people. The sampling frame needs to be appropriate and include the whole population of interest.

4. Sampling technique – is how we select and choose the people from the sampling frame. It is important to choose sampling technique that fit the research, and that can not be criticized for having an impact on the result, e.g. because the sample size is to small. There exist two branches of sampling techniques; *probability* and *non-probability* sampling. The probability sampling means that the sample is chosen on the background that it is representative for the overall population. *Non-probability sampling* on the other hand, is used when the researcher does not think it is feasible or necessary to have a representative sample size.

5. Response rate and non-responses – are the strategy of getting a better response rate when trying to get people to participate in the survey. A problem is that many people ignore such requests, and those that are willing to participate can in some cases only do it for self-gain etc. By having this in mind, it is possible to have a plan, e.g. to send the requests once more to the people who did not answer.

6. Sample size – It is important to have a sample size that is large enough, so that the result cannot be criticized for not being valid. Oates [44] argues that a good rule-of-thumb is to have a final sample size of at least 30.

2.1.4 Data Generation and Analysis

Research data can be divided into two groups; quantitative and qualitative data. The main activity to gather the data is by using a data generation method, e.g. interviews, observations, questionnaires or documents [44].

The data generation method is commonly associated with some research strategy. For example a survey is often associated with questionnaires, while experiments with observations. It is also possible to use more than one data generation method to enhance validity which is called *method triangulation* [44].

When the data have been collected, we need to analyze it. Data analysis is the process of looking for relations and themes in the data that could say something about the result. One way of doing this is by using mathematical approaches such as statistics to examine the data. Some people are arguing that we sometimes need to let everything go, and look at the data with a open mind without any pre-conceived idea or theories. This method is called *Grounded theory* and is trying to secure that we do not overlook any important findings [44].

A simple data analysis would consist of tables, charts and graphs, that could make the researcher see some patterns of interest. There exist also much more complex techniques with mathematical calculations and statistics. Some tools exist on the marked to make these complex techniques more feasible. Even tough tools exist to make the job easier, it is important that the researcher knows what he is doing and uses the right technique at the right time [44].

Research data is divided into two main categories; quantitative and qualitative data. What generation and analysis method we use, depends on the type of category the data is in. In the next sub sections, we will briefly go through each of the categories [44].

Quantitative Data

Quantitative data are data or evidence that is based on numbers. This is the main type of data that regularly being generated by experiments and surveys, but other types of strategies can also generate it [44].

Oates[44] has listed the following advantages and disadvantages with quantitative data:

- Provides scientific respectability. Some people believe quantitative data is the only form of valid research.
- Quantitative analysis has been time tested and based on established techniques, which adds confident to the findings.
- Because the data are not generated from subjective impressions, the data can be regenerated by others.
- Large volumes of data can be analyzed quickly using tools.
- Many people dislike working with numbers.
- Complex techniques can easily be misunderstood.
- The analysis depends heavily on the data being generated.
- Many variables are dependent when trying to make objective analysis. E.g. the size of the group and frequency count etc.

Qualitative Data

Qualitative data includes all non-numerical data like words, images, sound etc. [44]. It is the main type of data that regularly is being generated by interview, diaries, documents and web sites. As quantitative data is mainly used by the positivism, the qualitative data is mostly used by other types of paradigms, like the interpretive paradigm. It is possible to generate quantitative data from the qualitative. One way of doing this is by counting a specific occurrence of something, or giving some sort of score on the data. Qualitative analysis is not that straight forward. One problem is the lack of a standard way to do it, and the result can vary depending on the skills of the researcher [44].

[44] has listed the following advantages and disadvantages with qualitative data:

- It can be difficult to study things that cannot be reduced to numbers
- There can be more than one possible explanation, rather than the presumption that there will be only one correct explanation. This means that two researcher may come to different concussions which is equally valid.
- It is easy to be overwhelmed by the volume of data.
- Difficult to identify themes and patterns because the volume of data are huge.

- Interpretation of the data are more tied to the researcher.
- Non-textual data are not easy to present, e.g. inside a paper.

2.2 Our Research Process

The research process are the series of steps that make up the research from start to finish. In this section we will be giving detailed description of how we planned and conducted the research to satisfy the research goals(presented in Section 1.2) and came to the conclusion as we did.

Meetings with advisors		Milestones	Activities
11	August		
25	August		
22	September		
6	October		
20	October		
3	November	60% of Literate Review	
17	November	100% finding solution for further work	
1	December		
15	December		
(Christmas)			
12	January		
26	January		
9	February		
23	February		
9	March	Finished implementation and testing of prototype	
23	March	Finished testing the usefulness of the tool	
(Easter)			
20	April		
5	May		
18	May		
		Finished with the thesis	

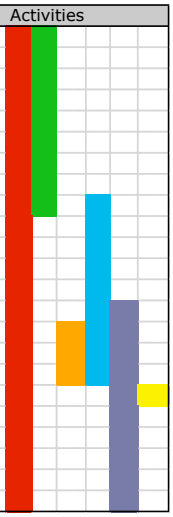


Figure 2.2: How we planned the research process.

In Figure 2.2 it is presented how we planned the research process. It shows when we planned to have meetings with the advisors, how much time we planned spending on each activity and what milestones we identified. The colors illustrates different activities, and have the following representation:

- Red - is the literature review, and from the figure, it is illustrated that we will never be finished with "learning" about software security.
- Green - illustrates the process of finding out what to do further with the research. In our case, this was the theory, and the idea of developing a tool to test it.
- Light blue - is the time developing the tool.
- Orange - is the time we allocated for testing and debugging of the tool.

- Yellow - is the time spent for testing the usefulness of the tool – data gathering.
- Dark blue - is the writing of the master thesis.

At the stage when we created this plan, not everything was planned down to the detail. When first starting the literature review, we only had one goal – to investigate the transition between security requirement and design. What guidelines, and how to utilize them was still unclear. We used the literature review phase (about 4 months) to investigate possible theories that we could work further on.

We managed to hold the time schedule(outlined in Figure 2.2) at most parts. The only problem we got was that the data analysis was not added to plan. Maybe, at the beginning of the project we under-estimated the workload or forgot it. This made a more or less hectic finish to the master thesis than we had foreseen.

In this section, we will outline how we planned and performed the main activities in the research process. First we will go through literature review, then we will be explain how we developed the tool and conducted the data collection. At last we will outline the data analysis.

2.2.1 Literature Review

In the literature review phase we investigated the transition between the requirement and design phase as this was one of the research goals. As further elaborated in Section 2.2.2, we wanted to develop a tool that would gain valuable new knowledge to the security field. The goal with a tool was still unclear. We used the literature review phase to investigate the possibilities.

While we were investigating, we noticed that the transition between the two phases, requirement and design, were not that incorporated. There was a tendency that security experts were doing the job in the requirement phase, and then later handed over the models, analysis and requirements to the design people. The design people, even though were security experts, had a problem utilizing the information from the requirement phase [15, 42].

We started to look more closely how the security requirement and design phase could be connected more naturally. The transition between requirement and design phase are regularly connected by using a software engineering methodology that has some idea about how to connect the phases.

As we were conducting the literature review, we also noticed that software security has the potential to be highly reusable in form of security patterns [20]. With that in mind, we found the possibility to use Security Design Patterns (SDPs) in the design phase(which is further elaborated in Section 3.4).

We started to investigate how patterns could be used more thoroughly. We came up with the idea that a connection between security requirements and design patterns could be made to create guidelines in the design phase.

To make the connection between security requirements and SDPs, we needed a way to define what security requirements to use. Investigating a bit further, we discovered that there have been some work on reusable security requirements(outlined in Section 3.2), and security patterns(outlined in Section 3.4. Also, Firesmith [20] states that requirements have a great potential to be highly reusable. Security Requirement Patterns (SRPs) were perfect for defining what security requirement was selected and could be used to map against design patterns.

This lead us to create the theory as described in Section 1.2, that drove us to create two more goals with the thesis. First, to study how security patterns could be used when eliciting security requirements. Second, to consider if a mapping between SRPs and SDPs could be used as guidelines in the design phase.

2.2.2 Development of the Tool

When starting to work on this thesis, we knew that we wanted to make a tool, or do some kind of software development. Much because of the practice and knowledge with software development, but also because it is fun.

The intention by developing a tool was to test the theory described in Section 1.2 and contribute to the goals with the thesis.

In Section 2.2.1 we described how we came up with the theory and goals with the thesis. Also, we felt that the goals by them selves were a bit broad. We created a list of hypotheses(listen in section 1.1) that we wanted to answer by developing the tool. Hopefully, the hypotheses could help us conclude the thesis at the end.

To create a tool that could test the theory, we came up with the following data mapping as described in Figure 2.3. At first, the theory was that "Category" in the current data model could be exchanged with "Attack/threat pattern". The theory was based on the idea that it is easier to know what

we want to be secure against, and difficult to explain it in form of a SRP. By using threats and attack patterns, the user could search or browse the information to find SRPs.

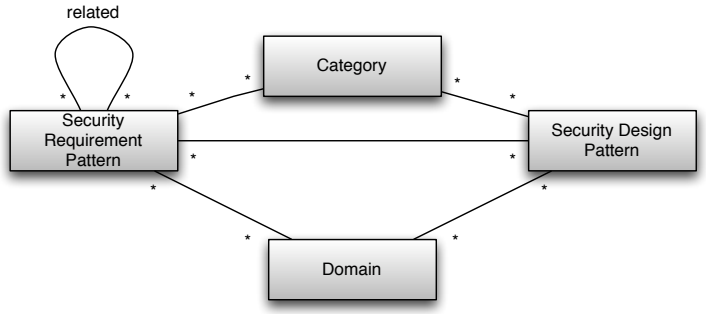


Figure 2.3: The data model. It shows how we connected the SRPs and SDPs

We have not rejected this theory. The fact is that by including threat and attack information, a lot more information would be added. In fare of overwhelming the user or make the tool unpractical because of to much information, we choose to withdraw the theory. More research on the subject is needed.

Figure 2.3 shows the data model that was implemented. As illustrated, the connection between SRPs and SDPs is connected as a many-to-many relationship with category, and is used to make the link between SRPs and SDPs. "Domain" in the figure, is to capture information about what area the pattern belongs to, e.g. health care or e-shop. As illustrated, a pattern can belong to many domains.

The domain information could be used to easier elicit requirements based on what domain we are finding requirements for – as a filter. This is based on the idea that a health care system has other requirements than an e-shop, and by providing the information about which domain we are gathering requirements for, we can remove unnecessary information from the list.

When deciding on development tools and technology, we gathered information about different technologies that we found interesting. The three most interesting technologies that we found and believed could be used were Adobe Flex, Microsoft Silverlight and different Ajax APIs. They are all modern technologies with focus on the User Interface (UI).

We think that the user experience depends heavily on the usability of the UI, and when choosing the technology, it was important that the UI were responsive and easy to use. Flexibility and development speed were also considered.

At last, we ended up with Silverlight 2.0 by Microsoft. The choice was not easy, all the technologies had their advantages and disadvantages. The final decision of selecting Silverlight, was that we could decrease development time, and still get a good product.

Silverlight has plenty of features that should make the life of a developer easier. It is cross-browser, cross-platform implementation of the .NET framework [40], which means we are actually running .NET code in the browser. This can be done because it is a browser plugin that is running on the client, just like Flash and Java applets. Even though Silverlight is a browser plugin, it is surprisingly easy for new users to install the client plugin and get started. Other reasons for choosing Silverlight are as follows:

- Silverlight is based on a subset of Windows Presentation Foundation (WPF), which is a markup language to build the UI [40].
- WPF makes a clear separation between the UI and business logic [40].
- Data binding enables linkage between the data layer and the UI automatically [40].
- WPF uses *Controls* to build the UI. This makes it easy to build the UI and reuse components [40].
- Silverlight uses the .NET framework which enables technologies like LINQ [40] and WCP [40].

As we had never used Microsoft technology before, we needed to learn everything from the ground up, which not only included Silverlight, but also LINQ [40], WCP [40], C-Sharp [40] and much more.

Silverlight.net [39] is a nice resource, and has a lot of information on how to start learning Silverlight. It has showcases, tutorials and links to other resources like e.g. WCP and LINQ. Also, the book "Windows Presentation Foundation Unleashed" [43] is a good resource on how to learn good UI development with WPF.

When the tool was finished, we needed to add data to it. From the literature review, we had found a repository of SDPs which is more described in Section 3.4.4. SRPs on the other hand, were more difficult to find. Some authors have recognized that security requirements can be highly reusable as stated in Section 3.2.3, but not much work has been done collecting the patterns. We recognized two alternatives; make the patterns our selves, or use the security requirements mechanism of Common Criteria (CC). None of the methods was perfect. Even though CC has been blamed for being a

complicated standard that is too technical for non-security experts[34], we went for the latter, and modified the security requirements mechanism from CC to become patterns.

Because CC groups the requirements into classes, families and so on, we applied classes as categories, families as a patterns, and the components as a requirement example within the pattern.

The data also needed to be mapped. The SRPs were mapped to SDPs, and both of them was mapped to categories. The mapping was based on the assumptions and knowledge and could be a reliability of the result. The data, with their mapping populated in the tool can be found in Appendix E.

2.2.3 Data Collection

When the tool was finished, it was time to test the theory and get data to support the goal with the thesis.

We decided to do an experiement with pilot users, as nothing could be more valuable than the feedback of others.

At first, the plan was to give people access over the internet, which the potential of getting a large sample size. With further thought, we decided it could be difficult to manage and secure the reliability of the result.

Because we wanted to test the hypotesis presented in the introduction we needed to test how the participants performed without a tool. An experiment like that could take time, we decided to have a controlled experiment in a controlled setting to make sure the reliability and validity of the data. We also decided to use questionnaires to gather feedback from the participants, and validate the result(method triangulation as described in Section 2.1.4).

The experiment had to be tested by participants with at least software development background. The goal of the tool is specific, and very technical. Not everyone can use it efficiently. The best scenario would be to get participants with only security background, because of their experience. On the other hand, getting so many people that are willing to be part of the experiment and have security background would be difficult. By getting participants that knew about software development, we would at lest get users with basic background.

There will always be the problem that some participants know more than

others in an experiment. This problem is very real in this case, because some users would have more experience within the security field than others. By collection background information of their experience, we could hopefully take it under consideration when analyzing the data.

The goal was to get at least eight participants that were willing to do the experiment. We were aware that this number was a bit small, and that [44] recommends at least 30 participants. We also estimated that the test could take about two hours. When a test takes that much time and we needed users with some experience with development, it could be difficult to get enough participants. We were very happy when we got informed that it was possible to get founding for the experiment. That way we could compensate the participants for their time and willingness. At the end, we got twelve volunteers by sending out requests on various e-mail lists on campus.

Getting participants by sending an e-mail like that, is reducing the sampling frame to only students at campus. Their motivation to participate should also be considered. If only students who have the interest, or knows something about software security is participating, we will get a narrow sampling frame [44].

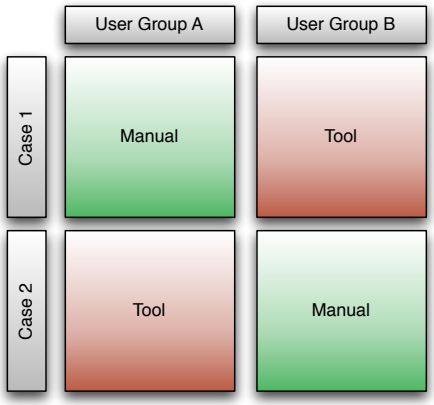


Figure 2.4: How we designed the layout of the experiment after the Latin Square Design.

In this experiment, we do not think the sample frame matter that much. Students are often curious, and want to participate in such experiments, as such their motivation should not be an issue. Also, we informed that software development experience had to be a minimum to participate, which would reduce the sample frame regardless of required method.

Because we wanted to test how the tool performed in contrast to the manual method, we added some extra insurance. We created two cases. For each case, we created two tasks. With each case, we also handed out some helping information to get them started. Cases, tasks and helping information can be found in Appendix C.

We could not find any similar research projects that we could compare the result against. We had to create every aspect of the test our selves(cases,

tasks, validation criteria and so on). This could be problem to validity and reliability as the participants might miss-understand the cases.

The first case was to find security requirements and the other one was to find SDPs. Each case was done with different methods. One with the tool, and the other with the manual method. The two cases were from two different domains, and had to some degree different security needs. Because there could be a learning effect between the two cases, we decided to use the Latin Square Design [10] as the layout of the experiment as illustrated in Figure 2.4.

The Latin Square Design could manage c number of cases, by creating c^2 number of experiments. This could be done by creating a square with c rows and c columns. The c number of experiments should be divided into the rows and columns so that none of the experiments is represented more than once in a row or column [10].

We had six participants do the first case with the tool and the second case with the manual method. Then we switched, so that six participants did it opposite(started with the manual method, and ended with the tool method). That way we could detect if there were any learning effect and how much impact it had on the data.

To find out if a tool could lessen the work finding security requirements and ease the transition between the requirement and design phase(goal two and three), we wanted to collect data to answer the hypotheses listed in Table 1.1.

We used questionnaires to gather data from the participants. They can be found in Appendix D. First, before the experiment started, we collected background information on the participants, as they may have an impact on the result. Then, after each case, we collected information about the experience the participants got using the different methods. From the information we collected, we elicited information of how they felt about the quality. Also, when using the tool, we collected information of their impression of the tool.

When creating questionnaires, it is important to ask the right questions, and that the questions can not be misunderstood. We did this in several iterations where we discussed what data we wanted to create, and how the questions should be formulated and presented with the supervisors.

Some of the problems we found with the questionnaires were that it was difficult to formulate the questions, and it required patience to do it right. Also, some of the questions we created, could depend on confidence, experience and so on.

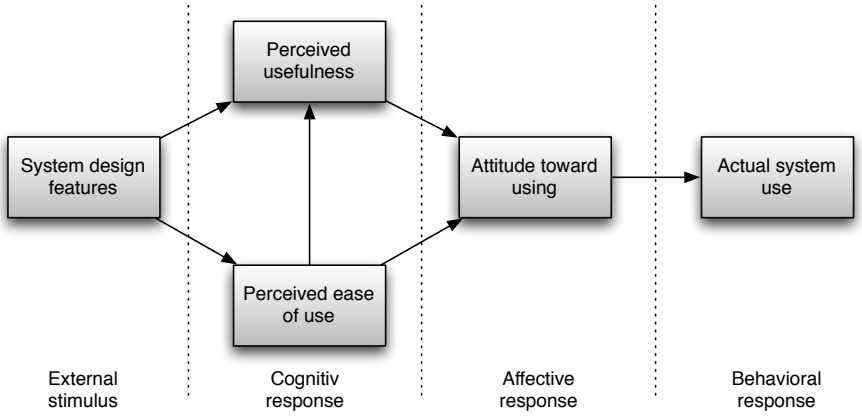


Figure 2.5: The figure outlines Technology Acceptance Model (TAM) [18].

Also, we tried gathering data that could support the Technology Acceptance Model (TAM). TAM specifies the relationship between perceived usefulness and perceived ease of use [18]. TAM is used to address why users accept or reject a information system, and how the user acceptance influences this. Figure 2.5 illustrates the TAM. From the figure, we can see how the cognitive- and affective-response is coupled with external stimulus and how the participants behave towards the system [18].

As this was a good idea, we lacked the knowledge and experience to do this in practice. We had misunderstood the concept, so instead of gathering data about their perceived usefulness and perceived ease of use *before* the users were introduced with the tool, we gave them the questions afterwards. The result was that instead of gathering data about the *perceived* usefulness, we gathered data about the *actual* usefulness.

Before the user test, we performed a pre-run on a couple of friend. By doing that, we got more experienced, validated the questionnaires and got a better idea on what sequence each activity should happen. What information to give to the user, before he get the case etc. We ended up with the following sequence of steps:

1. Hand out questionnaire about background.
2. Explain prerequisites, like who, how, and what we are testing.
3. Hand out first case.
4. After they had read and understood the case, they got the tools need to start working(e.g. pen and paper).

5. Hand out questionnaire about their experience.
6. Hand out second case.
7. After they had read and understood the case, they got the tools needed to start working(e.g. pen and paper).
8. Hand out questionnaire about their experience.

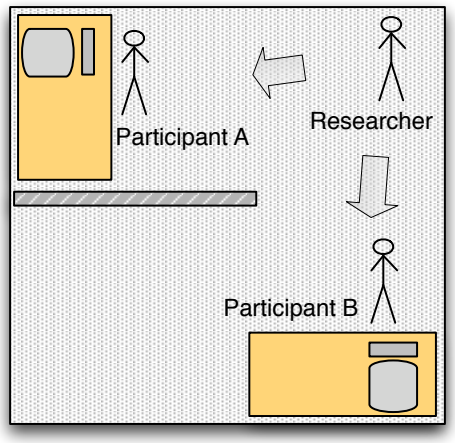


Figure 2.6: The research setting

Because this was a lot of information, and a lot of work, we gave them plenty of time. To not overwhelm the users, we choose to give them the case first, and then give them plenty of time to read and understand it before they got the tools needed to start working and had the possibility to ask questions. On each task, we gave them as much time they needed, but we tried to get them to finish within 30 minutes on task one and 15 minutes on task two in both cases.

The user tests were performed on the usability lab at Norwegian University of Science and Technology (NTNU) in the last week of March 2009. We decided to do the tests with two users at the same time. Each user got their own place to sit, with interior walls between them, so they could not see each other, but we could monitor and give information on both of them at the same time. Figure 2.6 illustrates the research setting. They also got explicit instruction to not talk or help each other. Both got a computer at each place with access to only the tool. When the tool was not needed, the computer was placed aside. When giving information, we gave information to both of them at the same time, and they followed the same experiment design. e.g. that both of them solved the same case with the same method.

2.2.4 Data Analysis

From the data collection phase, we knew what information to extract from the data. The hypotheses listed in Table 1.1 was used as a foundation.

First, we needed to obtain information about the security coverage – how good the security requirements and SDPs cover the security of the cases. Second, we need information about the quality – how good the data were described and what information it included. Third, we needed to extract valuable information from the questionnaires that could answer the hypotheses in Table 1.1, and get an overview of their background. Forth, as we thought there could be a learning effect between the cases, we needed to look at how much impact it had on the results.

As some security requirements can be described in different ways and denote the same, we used security aspects to describe what each case should be secure against. Security aspects is more a characteristic of consideration the system should be secure against(threats), than a security requirement, and we were hoping it would make the job of analyzing the security coverage easier.

When analyzing the security coverage of the data results, we assumed that by using the results from the participants, we could find security aspects that we had not thought of. We started by cycling through the results and look for security aspects that were new for us. When we found something new, we added it as one of the threats to the system if we thought it was needed. By using the results from the participants like this, we were hoping that the security aspects could cover as much as possible of the security needs in each case.

To calculate the security coverage, we went through each of the participants security requirements, and analyzed if they covered some of the security aspects that we had collected. When we were finished, we could calculate how many percentage of the security aspects that the security requirements had covered.

Looking at the coverage of SDPs, we recognized that it could be difficult to conclude if some of the SDPs were better than others. Event though if we were able to, it could be argued why something should be added and not. Also, we had realized that we to some extent helped the participants too much when eliciting SDPs using the manual method. Consequently, we concluded that looking at the popularity of the patterns could be more useful as it could tell us if the same patterns were selected with both methods.

When measuring the quality of the security requirements, we investigated

the literature on the subject and found different views of how a security requirement should be described and what information it should include. As denoted in Section 3.2.1, there is a lack of a defined way of describing requirements. This is way we used information from different sources to create variables that denotes a "good" security requirement. Using variables to define what to look for is a good idea as outlined in Section 2.1. The variables we found were as follows:

- Specific.
- Measurable.
- Traceable.
- Not be an architect or design decision[19].
- Express what is to happen in a given situation, as opposed to what is not ever to happen in any situation.[24].
- Describes why we need it[24].
- Describes what objects needs protecting[24].

Using variables, we went through all the security requirements, and gave them a score on how good quality they had.

When comparing the results, we compared how the participants used the tool method versus the manual method. The cases were from different domains, and had different security needs. Based on that, we concluded to represent the results based on both case and method. We could have summarized the results from each case, and only presented the data by method, but it is also valuable to see the difference between the cases. By looking at the result between the cases, we could pick up a possible learning effect or differences between the cases.

When analyzing the data from the SRPs, it was tempting to argue that SRPs should at least have a quality of 95%. A pattern is a well-understood solution to a recurring problem [16], and is literally capturing the experience from experts in a structured way that makes non-experts utilize the benefit from this information [51]. We argue that experts should at least manage to create patterns with a quality of 95%, based on the fact that they are experts and that a pattern is a fabrication of research and thorough consideration.

Analyzing the results we used a paired t-test to make a statistically claim if the results were valid. The t-test assesses whether the means of two groups are statistically different from each other taken variability into account [2].

When using the t-test, we looked at the p-value, because it represents the probability that the result was not a random mishap. This could indicate the significance in the results. When analyzing the results looking for something that was significant, we used a risk-level of 0.05. This is the mostly used risk-level and indicates that there is less than 5% chance that the result is significant [2].

Chapter 3

Literature Review

This chapter reviews the literature that we went through while we investigated the transition between the security requirement and design phase in more detail.

First Section 3.1 covers methodologies that clarify how to conduct secure software development. Then, in Section 3.2 we will give a thorough explanation of the most important aspects with security requirements. Afterwards, in Section 3.3 we will go through some popular security modeling methods and at last in Section 3.4 present the significant parts of security patterns.

3.1 Planning and Conducting Secure Software Development

When planning and conducting secure software development, it is important to have a well-defined plan on how to proceed with the activities at hand. Software engineering methodologies are structured plans on how to proceed with this process and describe how to approach tasks and activities that take place [21].

Secure software engineering methodologies are structured plans on how to conduct and plan a project in context of security. The methodologies presented in the next sections is not stand alone development methodologies, but are meant to be included into existing, standard software engineering methodologies.

In this section we will first go back to basic and describe what software engineering methodologies are and what they do in the planning of the

software development process. Afterwards we will be focusing on the most important secure software engineering methodologies and describe how they incorporate security into the planning of the software development process.

3.1.1 Software Engineering Methodologies

Software engineering methodologies are structured plans on how to proceed with the work in a project [21]. The plan generally consist of phases that emphasis on planning, time schedules, target dates, budgets and implementation details. Over the years new methodologies have evolved each with different advantages. What methodology to use depends on the project, management and team. Examples of some methodologies are[21]:

- Waterfall
- Prototyping
- Incremental
- Spiral
- Rapid Application Development(RAD)
- RUP
- Scrum

The waterfall methodology [48] is the oldest one, and was a popular in the 70s. Because new and more advanced methods are more or less based on the waterfall methodology, it is valuable to use it to explain the foundation behind software engineering methodologies. Figure 3.1 shows an overview over the waterfall methodology and how it uses *phases* to plan and structure the project. The phases are done sequentially and new phase cannot start before the first one was ended. In some versions of the methodology, it is possible to go backwards in the development cycle(or start over again) which indicate that the methodology was iterative. A phase can be seen as a state that the project has to go through in the development cycle. Other methods, e.g. the Agile, Spiral or the Spiral methodology are all going through the same phases, but they may do the phases over and over again or in parallel. Not only phases are used to distinguish the methodologies. All the methodologies have their own way of planning and conducting the project [21, 48].

Secure software engineering methodologies have emphasize on the importance of security. Mostly, these methods are not stand-alone methodologies,

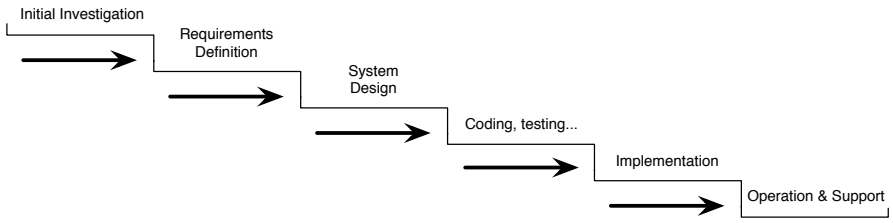


Figure 3.1: The phases in the waterfall method.

but are instead incorporated with the existing methodology used, e.g. the waterfall method.

3.1.2 Secure Software Engineering Methodologies

Secure software engineering methodologies are structured plans on how to conduct and plan a project in context of security. They are not stand alone development methodologies, but are meant to be included into existing, regular software engineering methodologies.

Secure software engineering methodologies are important because of the focus on security. One of the main problems that has been observed in application security is the lack of focus and knowledge on security. By implementing and using a secure software engineering methodology we are at least facing the problem and starting to do something about it.

SQUARE

Security Quality Requirements Engineering (SQUARE) is a software development method created by the Software Engineering Institute and provides the means for eliciting, categorizing, and prioritizing security requirements. The process is focusing on security from an early stage, and recognizes the need for documentation and analysis for later work [38, 37].

To get the most out of using the SQUARE process, it is best that it is conducted with a team of requirement engineers that have security expertise, together with the stakeholders of the project. Requirement engineering team and project stakeholders should first agree on technical definitions and start a baseline for all further communication. Business and security goal should be outlined and artifacts and documentation created to get the full understanding of the system [8].

SQUARE consists of a nine-step process that should end up with the final deliverable of categorized and prioritized security requirements [38, 37]:

1. Agree on definitions
2. Identify security goals
3. Develop supporting artifacts
4. Perform risk assessment
5. Select elicitation techniques
6. Elicit security requirements
7. Categorize requirements
8. Prioritize requirements
9. Requirements inspection

The steps should be conducted in a sequential order, so that the flow of information goes from one step to the other. After all the steps are finished, the information gathered should be merged into one deliverable; a security requirement document that the business should use in the rest of the process to build security into the system from an early stage.

Comprehensive Lightweight Application Security Process (CLASP)

CLASP is a technique that can be used to build security into existing or newly started software development life cycles. The method is structured, repeatable and measurable, which should make it easy to integrate into more agile development life cycles [52].

The CLASP process is divided into three sections; CLASP views, CLASP resources and vulnerability use cases. Further, the three sections are divided into activities that define the steps in the process [52].

CLASP recognizes security requirement elicitation as one of their "best practices", which means that it is included as one of the base activities. Nevertheless CLASP fails to give a thorough description of how the activities should be conducted [52].

Today, CLASP is part of the The Open Web Application Security Project (OWASP) standard, which is a worldwide free and open community focusing on software security. All their material is part of the open software license [4].

Trustworthy Computing Security Development Lifecycle (TCSDL)

TCSDL is the security development life cycle that Microsoft has adopted [29]. TCSDL objective is to modify the existing process to include security aspects to improve the software security. The modifications do not make drastic changes in the existing process, instead it defines checkpoints and security deliverables to be included [29].

In the requirements phase, TCSDL is focusing on inspection. The inspection regards defining the key security objectives that include finding a security advisor, making plans for security milestones and agree on security requirements [29].

TCSDL has also included threat modeling as one of the security deliverables. It is defined as one of the tasks that should be done in the design phase. The threat modeling process encompasses definition of assets, finding threats for the assets and estimation of risk. TCSDL uses treat modeling to identify needs for security features and to define where code review and security testing are required [29].

Three Pillars of Software Security

Mcgraw [33] presents three pillars of software security that could be cooperated with any software engineering methodology already used. The tree pillars of software security can better be described as guidelines and "best practices" than a security engineering methodology, but they serve as a good starting point and a good foundation for making secure software [33].

The three pillars of software security are applied risk management, software security touchpoints, and knowledge. Together they form a cost effective way a business could incorporate software security [33].

Risk management can be understood as a way to collect and analyze security information. Mcgraw [33] presents Risk Management Framework (RMF) as a continuous risk management process with the key idea to identify, rank, track and understand software security risks as they changes over time.

RMF has five fundamental stages:

- Understand the business context.
- Identify the business and technical risks.

- Synthesize and prioritize the risks, producing a ranked set.
- Define the risk mitigation strategy.
- Carry out required fixes and validate that they are correct.

Software security touchpoints have the intention to incorporate security best practices into the development life cycle. To develop more secure software, it is important that security is something that is incorporated in the whole development cycle, and that we have a good set of practices we can lean on. This is what software security touchpoints are trying to achieve by defining a set of good practices that has proven to be satisfying over years with experience.

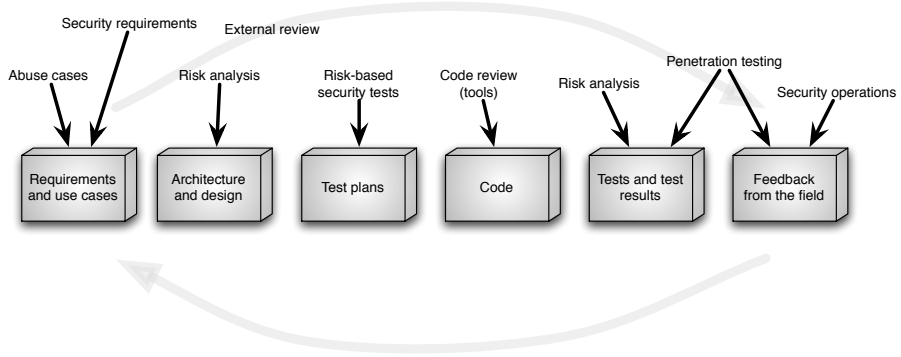


Figure 3.2: How the security touchpoints incorporate with the development life cycle

In Figure 3.2 the touchpoints is presented. The figure presents how the touchpoints could be used in the different stages of the development life cycle. Some touchpoints have proven to be better that others [33]. Mcgraw [33] has ordered the touchpoints by effectiveness, based on years of experience applying the touchpoints in real life projects:

1. Code review
2. Architectural risk analysis
3. Penetration testing
4. Risk-based security tests
5. Abuse cases

6. Security requirements
7. Security operations

Knowledge is the third pillar of software security. By knowledge [33] refere to the knowledge of "gathering, encapsulating and sharing of security knowledge". Compared with information, knowledge is how the information is set to use. Information could be books, reports, checklists etc. Knowledge is how this information is used, for instance in form of a tool. One significant category of tools in software security that uses important information are static analysis tools. Without the tools, the information would be difficult to use, understand and bring into context. By having a tool, the information transforms into knowledge that can be used when implementing a new system[33].

Knowledge is what drives us to better performance. By sharing knowledge we collaborate to the grater good and help the field innovate[33]. To day, there exist numerous databases and tools with the purpose of collect information about software security. One example is the Security Focus Vulnerability Database [7] which gathers information about security vulnerabilities. By using this information, we can help the software security field by innovating, decision-making and having a solid foundation for software security practices.

Mcgraw [33] also writes about what he claims to be the tree hurdles of knowledge. These are obstacles or bad practices in software security that knowledge would defeat the hurdles.

The three hurdles of knowledge are as follows:

1. Thinking of software security as a coding issue.
2. Thinking that software security is really about adopting various security features and/or conventions.
3. Overuse of checklists.

Common Criteria

Common Criteria (CC) is an international standard(ISO/IEC 15408) for evaluating IT products [14]. CC is basically a certification that IT products can get, but because of the guidelines on how to get the certification, CC is also valuable as a secure software development methodology[34, 14].

CC is an widely known and adopted standard, it is valuable because many knows what to expect from the certification, and it gives a measure of how good the IT product fulfills the security needs [34, 14].

CC consists of about 60 security functional requirements that are ordered in 11 classes. Together they provides a common set of requirements for the security functionality that should be used in the certification process. This makes it easier to find measures, and also makes certified IT products comparable [14].

Having a defined and international recognized standard like this is useful to gain confidence in the IT-system. It is difficult to assert that an application is secure because we do not have anything to support it. By getting certified, we can get that confidence. One problem however, is that the CC standard can often be too complicated and technical for non-security experts [34, 14].

3.2 Security Requirements

Requirements are the fundamental and critical part of every IT project. Without requirements, the developers would be lost and not know what to do, what to prioritize or what the outcome should be. Requirements are the bridge between the developers and the customers, where they agree on what the project should achieve [35, 12, 38].

In this section, we will first explains what difficulties exist when eliciting security requirements. Then we change focus to security requirement elicitation techniques and how they relate to the software engineering methodologies. Then we outline the importance of reusable security requirements and at last give some examples on reusable security repositories that exist to day.

3.2.1 Difficulties

Tøndel et al. [56] have done a survey on security requirements elicitation and their impression is that many are talking about it, but few are actually doing it. Security requirements are a fundamental task when trying to achieve better security because it values to begin early [35, 12, 38]. So why is not everyone doing it? Software security practices are still immature, and the approaches and techniques for eliciting security requirements may be too difficult or comprehensive for non-security experts. The steps involved in integrating the approach into existing software development life cycle may

be too cumbersome, e.g. in form of too narrow or wide to fit into the existing project.

There is lack of a universally accepted definition of security requirements [56]. From Section 3.1 we know that the different approaches uses different techniques, and the chosen approach will therefore have much to say regarding the quality of the security requirements. Hence it is important to have a clear definition of what security requirements we want, and what techniques we should use to get it before we choose an approach. The lack of a common definition of security requirements indicates that it would be easier for everyone to have concrete example on how to formulate security requirements [56].

It is also a lack of a defined way of what a security requirements should include and how they should be described. Haley et al. [24] has noticed that security requirements are described in different ways that may not give all necessary information. Haley et al. [24] points out that one way of formulating security requirements are by describing the security mechanisms to be used, e.g. "Use SHA256 encryption between server en client". The problem is that the requirement is telling what to do, not why. Another way of describing security requirements are by describing security functionality, e.g. "Use an Firewall to protect from outside attack". This requirement lack the description of what to protect and why. Haley et al. [24] points out that their framework expresses security requirements as what is going to happen, instead of describing what is not ever going to happen. This, they point out, has the ability to give information about when and why, and leaves the how up to the designer [24].

3.2.2 Approaches to Security Requirement Eliciting

Security requirement eliciting approaches are strategies to eliciting and analyzing information in the requirement phase. In Section 3.1 we gave a brief outline of the processes of conducting secure software engineering. In this section, we will describe the approaches, and how the secure software engineering methods employ them.

Tøndel et al. [56] lists the following approaches for doing secure requirement eliciting:

1. Definitions - agree on definitions that will make a common base of understanding.
2. Objectives - define objectives, the goal or achievements for the business.

3. Misuse/threats - eliciting misuse and threats to the system.

4. Assets - deduce the valuables of the business.

5. Coding standards - define what programming language to use and how to use it.

6. Categorize and prioritize - sorts the requirements to know what to prioritize.

7. Inspect and validate - the requirements may change over time.

These are security requirement eliciting approaches that the different processes have utilized. Some approaches are used more than others. Tøndel et al. [56] discovered with their research that Misuse/threat, Objectives and Assets approaches were mostly used. Because they are mostly used, it is reasonable to conclude that they also are the most important once [56].

In Table 3.1 the approach that was used in the different methodologies are listed. From the table, we can see that Security Quality Requirements Engineering (SQUARE) is the methodology that has put to use the most approaches. The other two, Comprehensive Lightweight Application Security Process (CLASP) and Trustworthy Computing Security Development Lifecycle (TCSDL) is only utilizing two of the approaches and have Objectives in common. Also SQUARE is using Objectives. The interesting part is that even though CLASP and TCSDL are only using two of the approaches, they both managed to have an approach that no other has.

Table 3.1: The relation between security methodologies and security requirement eliciting techniques [56].

	CLASP	SQUARE	TCSDL
Definitions		Yes	
Objectives	Yes	Yes	Yes
Misuse/threats		Yes	
Assets	Yes		
Coding standards			
Categorize and prioritize		Yes	
Inspect and validate		Yes	
Process planning			Yes

3.2.3 Reusable Security Requirements

Some research have shown that security requirements has the potential to be highly reusable [20, 58]. Studies have shown that while functional requirements tend to be very different, the same cannot be stated about security requirements. Firesmith [20] draws forward a good example; the functional requirements for an embedded avionics application and an ecommerce website may be very different, but the security requirements tend to exhibit far less variability. Also, Firesmith [20] has recognized that for every application, at the highest level of abstraction, they tend to have the same basic kinds of valuable and potentially vulnerable assets.

There are not much research on the subject, but from what has been conducted, reusable security requirements looks promising. Eliciting security requirements is a difficult process, which require both experience and security knowledge.

Firesmith [20] describes the following advantages by employing reusable security requirements:

- We will get suggestions on what information to include, general advice and common pitfalls.
- We do not have to write each requirement from scratch.
- We will get more consistent and well defined formulation of the security requirements.

3.2.4 Security Requirement Repository

Withall [58] has written a book "Software Requirements Patterns" which is using the idea behind reusable requirements. Not only do Withall [58] consider security requirements, but also functional requirements that he has discovered could be reusable. His catalogue includes 37 reusable requirements in the form of patterns, where six of them are security related in form of access control.

A pattern is one way of dealing with reusable items. Withall [58] has put to use the idea, and encapsulated the requirements into patterns with the following format:

1. Basic details - the pattern manifestation, owning domain, related patterns, anticipated frequency of use, pattern classifications and pattern author.

2. Applicability - what situations the pattern can be applied.
3. Discussion - how to write a requirement, and what to consider.
4. Content - what the requirement say.
5. Template(s) - starting point of writing requirements of this type.
6. Example(s) - one or more representative requirement written using this pattern.
7. Extra requirements - explains what sorts of requirements often follow on from this type.
8. Considerations for development - hints on how to implement it.
9. Considerations for testing - what to have in mind when testing.

Common Criteria (CC), which is more described under chapter 3.1.2, consist of about 60 security functional requirements that are ordered in 11 classes. Also, CC includes a set of security assurance requirements. In this context, they are not equally important because they are more focused on *assurance* of the measures instead of the measures. Together, all the requirements form the foundation of CC that are employed when evaluating IT products [14].

CC does not use patterns to represent the security requirements as Withall [58] does, but are organized into a hierarchical structure. The hierarchical structure consists of Classes, that consists of Families, that consist of Components that finally consist of Elements. This organization is provided to make it easier to find specific components.

Classes are used as the most general grouping. The members of the group share a common focus, e.g. the class FIA - "Identification and Authentication", is focusing on identification and authentication of users.

A family is a grouping of components that share a specific focus, but may differ in emphasis and how strict they are. An example is the FIA_UAU family, which is part of the FIA class. The FIA_UAU family concentrates on authentication of users.

A component is the smallest selectable unit in CC. The set of components are ordered by their emphasis and strictness, but they may also be ordered by their relations to other components. An example of a component is the FIA_UAU.3 – Unforgeable Authentication (which has focus on unforgeable authentication).

Elements are the smallest units, and together they form the construction of components. Example of an element is the FIA_UAU.3.2 which concerns the prevention of copied authentication data.

3.3 Security Modeling Methods

Security modeling methods are ways to gather, present and elicit information. They represent information that may have value e.g., when doing a security analysis or describing security information [47].

This is important because the security field is constantly changing, and the size of the systems is getting bigger and bigger. We need methods to represent the systems and their security related information. That way we can easily communicate and discuss information like expected functionality, potential threats and exploits [47].

There are methods that are elicitation the same information, but they may have different fulfillment, different name and emphasizes [23]. Some methods are focusing on eliciting of information and others on analyzing the information. As many of the methods have different views of the problem, they may complete each other. For example threat modeling can help develop realistic and meaningful security requirements [41].

Security modeling has evolved considerable and different modeling tools and methods exists for different areas within software security. The main areas that we have identified are threat, attack and vulnerability modeling, requirement eliciting modeling and architecture and design modeling [23].

What method to prefer over another may depend on the software engineering methodology used [23]. From Section 3.1 we know that the software engineering methodologies have different ways of conducting the development process. That the methodologies have different approaches which are favored as stated in Table 3.1, could have an impact on the preferred modeling method.

In this section we will be giving an outline of some of the most important security modeling techniques. First we will go through misuse and abuse cases, then we will describe attack trees. Afterwards we will outline Secure Tropos and describe UMLsec and SecureUML.

3.3.1 Misuse and Abuse Cases

Misuse and abuse cases are modeling techniques that are used to describe and elicit the threats of a system. Both systems are derived from use cases, with is one of the Unified Modeling Language (UML) diagrams that are used to describe the behavior of a system. Use cases captures the functional requirements while misuse and abuse cases captures the malicious behavior that the functional requirements can cause [55, 32]

Use cases are part of the widely used standard – UML. Because most developers already know UML and use cases, the threshold of understanding misuse and abuse cases are small. It is also easier to discuss the system and threats with the customer because of the low threshold of understanding [55, 32].

Misuse Cases

The approach of misuse cases is to extend the positive(regular) use case diagrams with negative use cases – misuse cases. Misuse cases specify the behavior that is not wanted with the purpose to eliciting the security requirements for a given system [54, 55].

The notation of misuse cases includes the *misuse case* and a *misuser* together with relationships. The relationship notation includes *threatens*, *mitigates* and *aggravates*, and they are used to draw the relationship between the use cases and the misuse cases, and the misusers and the misuse cases. The concept of this notation is that an use case mitigates a misuse case, and the misuse case threatens the use case [54, 55].

The notation uses inverted graphics that incorporate with regular use case diagram without confusion [54, 55]. Figure 3.3 illustrates an example of a misuse case of an e-shop. From the example, we can see how the transition between the behavioral and malicious behavior in the system is illustrated.

A misuse case can also be represented as a more detailed misuse case description [54, 55]. There are different templates suggested for this; some examples of such templates are the *heavyweight* and the *lightweight* template witch is described in more detail by Sindre and Opdahl [55]. Both of them describe the misuse case as textual representation but they differ in how much information that is included.

Sindre and Opdahl [54] has listed the following as proposed steps for eliciting security requirements with misuse cases:

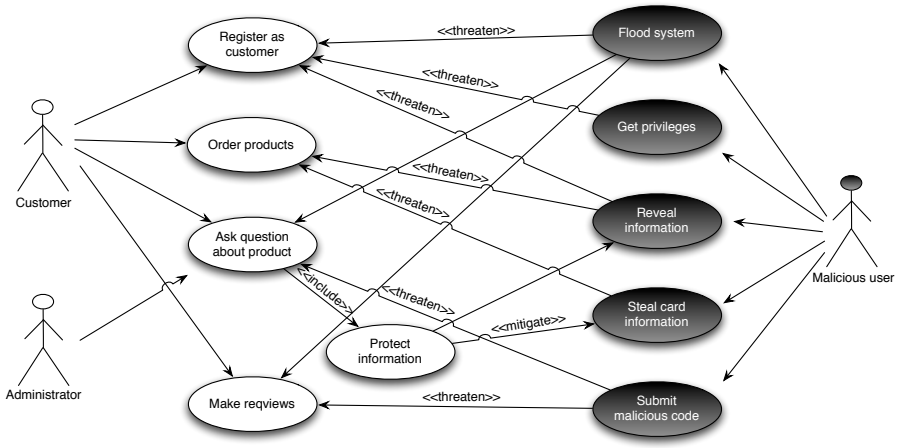


Figure 3.3: Example of a misuse case diagram for an e-shop [54].

- Identify critical assets.
- Define security goals.
- Identify threats.
- Identify and analyze risks.
- Define security requirements.

Røstad [47] proposes to extend the misuse case notation to include the ability to represent vulnerabilities and insider threats. The extended notation enables expressing a richer and more complete picture of security threats. The extension helps visualize the problem with insider threats, which have much more potential of misusing the system than an external actor. Considering this, the information will help making a more correct picture of the problem and might have an impact on the risk analysis and requirement prioritizing [47].

Abuse Cases

Abuse cases resemble misuse cases, but uses another approach to the problem. They use the same notation from UML, but differ in the philosophy that abuse cases should be made on different diagrams. Abuse cases are not shown on use case diagrams and use cases are not shown on abuse case diagrams [32].

McDermott and Fox [32] has listed the following steps to make abuse cases:

- Identify the actors.
- Identify the abuse cases.
- Define abuse cases.
- Check granularity.
- Check completeness and minimality.

Both approaches extend traditional use cases to also cover abuse and misuse, and are potentially useful for several other types of extra-functional requirements beyond security [54]. Sindre and Opdahl [55] have some examples of misuse cases in domains like availability, reliability and robustness, but points out that further research are needed to provide useful guidance on how to utilize the techniques.

3.3.2 Attack Trees

Attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks. An attack tree is a formal and methodical way to describe the security of systems based on varying attacks. The attacks are presented as nodes in a tree structure, with the goal of an attack as the root node. The different leaf nodes, will then be individual ways of achieving an attack on the system [49].

In Figure 3.4, an example of a simple attack tree created by Schneier [49] is illustrated. In the figure, we can see that the goal of an attack is to open the bank safe, which is the root node. All the child nodes are possible attacks to succeed the goal. Some child nodes also have children, e.g., the "Learn combo" node. This illustrates that there are many ways to get to the state where you have "Learned the combination". So actually, the "Learn combo" is a sub goal, on the way up to the root node.

When an attack tree is completed, we can assign values on the nodes to get more information about the attack. All type of values can be used, boolean and continuous, and they can be combined to learn even more about the vulnerabilities of the system. One example of such use, is to go through all the nodes and assign the values impossible and possible. Afterwards, we can easily see what paths we need to look at more closely. Other examples of such use are e.g. to find the probability of success of a given attack, or the likelihood that an attack will be tried and so on [49].

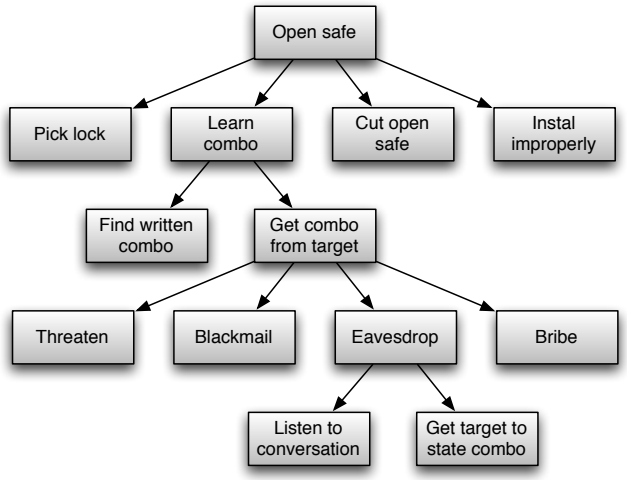


Figure 3.4: Example of an attack tree of a bank safe [49]

An attack tree can also be presented textually. It can be done by for example using indentation to represent each level in the tree. For some, making the tree textually may seem faster, and could be useful when e.g. doing brainstorming [49].

A really useful property about attack trees are their reusability. A complete attack tree can also be used in other situations that have the same needs [49].

A problem with attack trees are that they depend heavily on the security knowledge of the people creating them. When making an attack tree, it is important to think like an attacker and have comprehensive knowledge of types of attack and their severity. On the other hand, the reusability of attack trees can lessen this problem [23].

3.3.3 Secure Tropos

Secure Tropos is a formal framework for modeling and analyzing security requirements. Secure Tropos is an extension to the existing software development methodology Tropos [22].

Tropos is based on Agent Oriented Programing (AOP). AOP is a programming paradigm just like Object Oriented Programing (OOP) with focus on design and implementation of software architecture. AOP is based on agents, that should make the software more reliable when changes occur and

new components are added [45].

Tropos is a software development methodology that uses AOP and is used at different phases throughout the development cycle, from early requirements to detailed design [22]. The phases that Tropos is founded on are the following: early requirements, late requirements, architectural design, detailed design and Implementation where the last four is well-established in the software industry [22], and can be related to the phases described in Section 3.1.1.

The emphasis on requirements in the early phases makes Tropos get a deeper understanding of the environment the software should operate in, e.g., what kind of interactions the software will have with the external environment [45].

Tropos adopts the i* modeling framework which offers the notation of actor, goal and dependency. The strategy of i* is to model and describing the network of relationship among the actors. i* also has support for describing and supporting the reasoning of relationship with other actors[31].

It is argued that the Tropos framework lacks the ability to capture aspects of security, and therefore we need the extension Secure Tropos. Secure Tropos has the following extensions to the notion already defined in Tropos [22]:

- Ownership - is the state between the actor and a service. It represents that the actor is the legitimate owner of the service.
- Trust - is the trust between two actors and a service. Indicates the belief of one actor that another actor will not misuse the service that he has been granted.
- Delegation - is the delegation of trust between two actors and a service. Marks that permission has been granted.

The extended notions are intended for modeling and analysis of functional and security requirements [22].

Giorgini et al. [22] has the following example of a requirement that describes the delegation of trust between three actors using Secure Tropos:

> "Alice is interested in gathering data on student performance, for which she depends on Sam. Bob owns his personal data, such as his academic record. Bob delegates permission to provide information about his academic record to Sam, on condition that his privacy is protected (i.e., his identity is not revealed)."

In this example, there is a difference in the relationship between Alice-Sam and Sam-Bob. This can be graphically modeled with i* as in Figure 3.5. By using the information from the example and the model, we can also make formal arguments of the validity of the requirement [22].

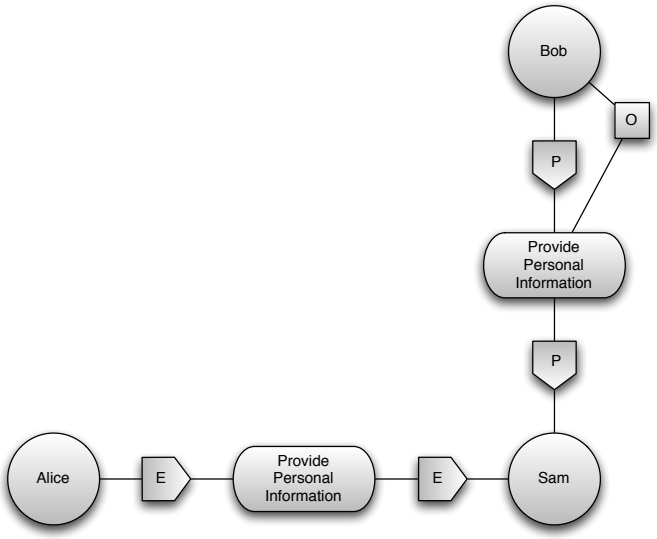


Figure 3.5: Example of modeling delegation of trust with i* using Secure Tropos[22]

3.3.4 UMLsec and SecureUML

UMLsec and SecureUML are techniques to describe the design of a system with security in mind. Both are extensions to the already well-defined UML standard. UML is the de-facto standard for OOP, which has the following advantages [30, 28]:

- Many modeling tools support UML.
- Developers are already familiar with the language.
- UML is relatively precisely defined.

When designing and implementing a system, it is widely used, and known, that a system need to be planned down to the very detail. When designing a system with security in mind, the current UML method is lacking functionality to describe security needs. This is what UMLsec and SecureUML are trying to do – add security information into the already widely known and used UML standard [28].

UMLsec extends the UML standard with the following mechanisms: stereotypes, tagged values and constraints [28].

Stereotypes defines new types of modeling elements and are represented in double angle brackets. Tagged values are a named-value pair that is associating data with model elements that are tagged with curly brackets. Constraints are restrictions that are placed on the stereotypes [28].

UMLsec manages to encapsulate the following security information in their modeling elements; security requirements, threat scenarios, security concepts and security mechanisms. These are information that UMLsec encapsulates in their models, so that security-experts and developers can read the information from the diagrams [28].

SecureUML only extends the UML standard with information relevant to access control. The information that SecureUML adds are based on the Role Based Access Control (RBAC) model. The RBAC model is a well-established access control model that has widely recognized advantages as well as being supported by a large number of software platforms[30]. SecureUML defines a vocabulary for expressing different aspects of access control like roles, permissions and user-role assignment.

The SecureUML method is only at the first step. Further work will involve focus on modeling security requirements and design information [30].

3.4 Security Patterns

Patterns are a approach of dealing with reusable solutions to common problem, which means that novices can benefit from know-how and skills of experts [26][51]. Patterns exist in many different fields. In software, the success started with the Gang of Four (GoF), which had focus on software design [26].

Security pattern is a type of pattern that deals with the security domain Schumacher and Roedig [51] defines a security pattern as follows:

> "A security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution"

Software security is a huge area, and there are many situations security patterns can be used. The success of patterns(in the software industry)

started with design patterns, and this is generally what most people think of when talking about patterns. There exist more variations that are worth mentioning [16]:

- Architectural patterns - describes a more high level abstraction than design patterns.
- Anti Patterns - describes something that is initially beneficial, but ultimately will have bad consequences[42]
- Pattern Languages - Families, groups or collections of interrelated patterns which are popular for object-oriented frameworks.

Also, security patterns can further be divided into more specific and formal identification.Darrell M. Kienzle [16] have identified two broad categories of security patterns:

- Structural Patterns - these are patterns that can be implemented in the final product. They encompass design patterns such as those used by the GoF. They usually include diagrams of structure and dynamic interaction.
- Procedural Patterns - these are patterns that can be used to improve the process for development of security-critical software. They often impact the organization or reporting structure of the project.

When we look at the problems the software industry are facing when regarding security, we can se that there are often the same mistakes over and over again [51]. With that in mind, security patterns should have a great potential to solve this. At present, there is a huge gap between theory and the practice, which indicates that we need to come up with tools and practices that make the theory more usable for the industry [51].

In this section we will first outline what state security patterns is in, and what benefits we can get from using them. Then we will describe how to use patterns, and what structure they have. At last we will give some example of existing pattern catalogs, and where to find them.

3.4.1 Future of Security Patterns

The patterns approach have been greatly utilized in the software industry. Java APIs, libraries and Microsoft Foundation Classes are examples that use

patterns extensively [16]. Also, there are comprehensive workshops, books and web sites that are devoted to further study and development of the patterns [16]. These are examples of design patterns. Security patterns have yet to see the same popularity as the design patterns [26]. Heyman et al. [26] and many others [3, 1] believe that security patterns have a great potential.

There exists user groups like SecurityPatterns.org [3] and the Yahoo user group [1] that are gathering and discussing the future of security patterns. Their work of collecting material, indicates that new security patterns and research on the subject is still evolving. In Figure 3.6, from Heyman et al. [26] work, we can see how the security pattern landscape has evolved from 1996 to 2006. From the dark line in the figure, we can clearly see that the number of security patterns are growing each year. Their work does not only incline that the number of security patterns are growing for each year, it also states that the quality of security patterns are getting better for each year.

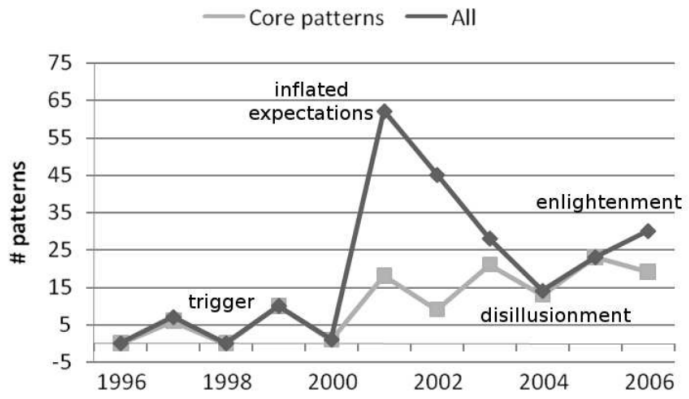


Figure 3.6: Number of security patterns by publication year [26]

Heyman et al. [26] has identified tree main advantages when using patterns:

- The solution should be sound, because it has been time-tested.
- Benefits and drawbacks are known and can be taken into account when designing a solution.
- Ease of communication between stakeholders.

With the advantages listed above, together with the security problems we are facing, security patterns have the potential to bridge the gap between

the design phase and secure code [26]. Also, a pattern is following a well-known structured way of dealing with information. Most developers already knows what a pattern is, and can relatively easily get into the details [50].

Because security is a complex topic and software development is mostly done by developers with little or no security knowledge. Make use of security patterns to transfer the skills and knowledge from security experts to software developers could be an advantage [51, 50, 16].

On the other hand, not everything is great with patterns. Many think of patterns as a silver bullet, but it is crucial to have an understanding of the pattern as well [13]. Also, as the number of patterns grown, it will get harder to maintain and find the correct pattern for the right task [26].

Security patterns have a huge potential and a lot of people are believing and working for it. The future of security patterns looks bright, and we can already see some of the effects it will have on the software industry by looking at the communities and their engagement.

3.4.2 How to Use Security Patterns

Given a good pattern description, it should be obvious to determine whether a pattern is applicable to a particular situation and how the actual instantiation of a pattern should be done [51]. From looking at the description of a pattern, e.g. from a pattern catalog, you should easily have the possibility to determine whether the pattern is usable or not in the case at hand. An example is the security pattern repository by Darrell M. Kienzle [15]. The Repository includes about 17 structural patterns, and 13 procedural patterns. At the beginning of the document there are a nice outline that summarizes all of the patterns by listing the name of the pattern and the abstract description. By skimming through this list, we can get an idea of what the pattern does, and if it could be valuable in the case [51].

Tools could have a huge impact on the simplicity and effectiveness of finding the right security pattern [51]. Tool support for security patterns is not an objective the pattern community has [51], but Schumacher and Roedig [51] points out the following benefits with a toolkit:

- Maintenance - creating, editing, publishing, and reading patterns is a huge effort. A tool could lessen the work enormously.
- Classification - with a classification we can get taxonomies on both problems and solutions. Similarly a context hierarchy could be developed.

- Modeling - a tool could help integrating the pattern when modeling the system.
- Reasoning - patterns can overlap and subsume each other, some cannot work with others etc. A tool can use this information to make reasoned decisions.

Schumacher and Roedig [51] have an example of practical, but still fictionally(because the tool does not exist) use of security patterns. The example is a real case of a security flaw that Microsoft introduce in their version of the Point-to-Point tunneling Protocol(PPTP). The problem was that the Point-to-Point protocol used a hash algorithm that was very easy to brute-force because of an implementation weakness. Also, both session key and authentication token were based on the same hash value. So an attacker that broke the hash could both have access to the system and read the traffic over PPTP.

A tool utilizing security patterns could have prevented this from happening by visualizing the information. The tool could show where patterns were implemented and their relations to part of the system. Also, the tool could have detected that both security mechanisms were based on the same hash and give a visual feedback on the security issue. Also, the tool could have given implementation details, like what type of data should be used to generate the hash. By having all those features into a tool, security patterns would definitely show their value [51].

3.4.3 Pattern Structure

A pattern consists of a set of attributes that constitutes the structure of the pattern. Design patterns generally stick to more restrictive format than other type of patterns. They include Unified Modeling Language (UML) diagrams that explain the structure and the interactions between objects. They also require examples of how they are used and relation to other patterns together with sample code. Darrell M. Kienzle [16] have cross referenced their template with others and found that the following consist of the most important elements in a security pattern:

- Pattern Name - the name of the pattern.
- Abstract - a short summary of the pattern.
- Aliases - other names that the pattern may be known as.

- Context - is the situation the patterns may apply.
- Problem - describes what conditions that motivate the usage of the pattern.
- Solution - describes how the patterns solves the problem.
- Static Structure - what elements that are involved with the usage of the pattern.
- Dynamic Structure - outlines the interaction between the components in the static structure.
- Implementation Issues - hints and details on how to implement the pattern.
- Common Attacks - which attacks that has been identified interacting with this pattern.
- Known Uses - actual examples of use.
- Sample Code - some sample code for the developer to more easily grasp the pattern.
- Consequences - what benefits the pattern provide and potential liabilities.
- Related Patterns - list of patterns that have something in common.
- References - citations related to the pattern in the literature.

Not all elements are equally important, and may be left out in some security patterns.

3.4.4 Security Pattern Repository

Pattern repositories are archives that collect and present patterns. The lists known are gathered either by communities or researchers.

The most known security pattern repository is the SecurityPattern.org [3] web site. The web page is collecting and presenting security patterns sorted by the date they came out. The list consists articles and books that constitute the receptive patterns, and does not contain any well-defined way of handling the pattern data.

Darrell M. Kienzle [15] have another example of a security patterns repository that consist of 16 structural patterns and 13 procedural patterns that focus on web application security.

Example of other pattern repositories(not security specific) are the Portland Pattern Repository [9] witch includes all sort of patterns. Quince [5] that features Graphical User Interface (GUI) patterns. Wikipattern [6] web-page which includes a huge collection of all sorts of patterns.

Chapter 4

Description of the Tool

This chapter describes the tool that we created to test the hypotheses in Table 1.1. First Section 4.1 outline the requirements that were defined for the tool. Then in Section 4.2 an presentation of the functionality of the tool is given. At last, in Section 4.3 we conclude if the tool is complete.

4.1 Requirements

The tool was first and foremost developed as a prototype that could test the hypotheses listed in Table 1.1. Goals like maintenance and interoperability were not prioritized. We focused mainly on usability, as this could have an impact on the result. We used an agile development method, with focus on prototyping. By agile, we mean that nothing was final before the very end and adapted new improvements and ideas as we went on. Regular meetings and discussions of the progress kept us at a steady course. We began the process by making low-fidelity prototypes with suggestions on how the tool could look and behave. By presenting them to the supervisors, we got feedback, made some changes and presented them again.

We made a list of requirements for the tool. At first, the list contained only functional requirements. After further iterations, we made some changes to the list and added priority, non-functional requirements and security requirements. The requirements were used as a guide when developing and planning the tool.

The list of requirements for the tool presented in this section are not intended as an exhaustive list. More work should have been put into keeping it up to date as the iterations went on. For what it is worth, it represents an

early draft of what we had in mind, and can to some degree validate the completeness of the tool from an early point.

The requirements were ordered by priority and were prioritized as follows:

- Essential - states functionality that needs to be in place for the tool to function properly.
- Beneficial - enhances the product and may have impact on the research, but are not unacceptable to omit.
- Optional - functionality that would make the tool better, but has no benefits to the research.

The following is the list of requirements that we elicited for the tool:

Functional requirements

Essential

1. Find Security Requirement Patterns (SRPs) by searching.
2. Find SRP by browsing categories and security measures.
3. Find mitigation techniques for the selected security requirements(in form of Security Design Patterns (SDPs)).
4. The possibility to create a private list of SRPs for easy access.
5. Clear the private list of SRPs.
6. Possible to create/delete/modify a SRP.
7. When changing a SRP, the changes will only be private for the user in question.
8. Possible to use examples in the SRP as a starting point of creating/-modifying security requirements.
9. Find mitigation techniques based on the security requirements in the private list.
10. Possible to sort the security requirements elicited.
11. Possible to have a private list of security requirements for more than one project.

12. The application should give feedback on functionality(when mouse-over) on parts that could be misunderstood.

Beneficial

13. Possibility to rank the security requirements.

14. Possible to submit suggestion for new data (SDP, SRP).

Optional

15. Context sensitive data. E.g., give suggestion of related patterns (using data from previous users).

Non-functional requirements

Essential

1. The tool should be implemented as a web-application for easy access.

Security requirements

Essential

1. Users should be divided into two privilege groups: administrator and regular users.

2. Only users with administrator privilege should have access to grant new data to the tool (suggestions, new pattern, etc.).

3. Only authenticated users should have access to use the tool.

4. Data created by the users should be considered personal and other users should not get access to it if not otherwise explicitly set or the user has administrator privileges.

The requirements were elicited using a mind intensive planning stage where we identified what the tool needed to support the hypotheses and goals listed in Section 1.2 and Table 1.1.

4.2 Design and Implementation

In this section we will outline how the functionality of the tool is implemented. We have roughly divided the tool into five areas; authentication, security requirement eliciting, private security requirements, project administration and tool administration. In this section, we will outline each of them in turn.

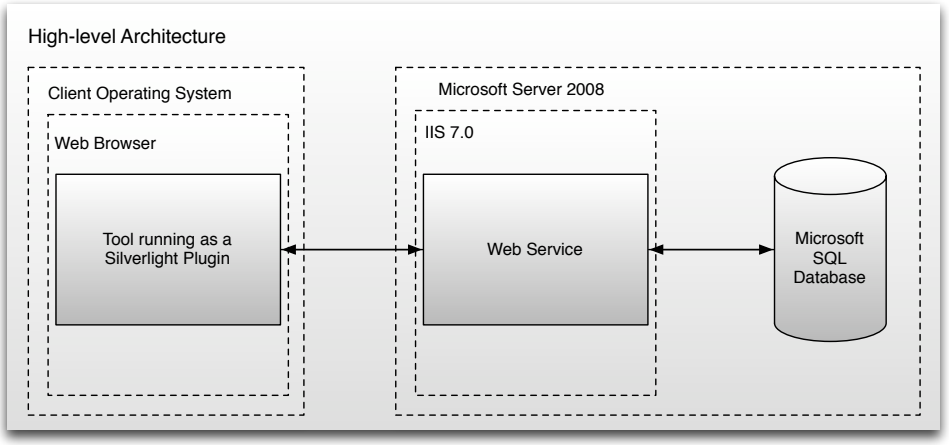


Figure 4.1: A high-level architecture of the tool.

A high-level architecture of the tool is presented in Figure 4.1. From the figure, we can see that the tool is a Silverlight plugin running in the web browser at the client. Because the application is running on the client, we also needed a web service. We could have managed only with the database, and do the data transfer straight from the client to the database. Doing it that way would not been a good solution as it could have caused problems for some users. By utilizing port 80 for data transfer(as a web service does), we circumvented any problems that e.g., a firewall could have caused.

A web service also has the advantage that it does not depend on the client. Other developers and researchers could access the content on the web service and use it to their likings. E.g., create a new an improved version of the client. The web service supports all the functionality as the client does today; creating new users, authentication, retrieval and populating of pattern data and so on.

4.2.1 Authentication

The tool need a way to handle users and their projects. One way of handling this was using authentication. We created two access levels, user and administrator as specified by the first security requirement in Section 4.1.

In Figure 4.2, the login window is presented. This was the first page all users would see when they entered the page, and worked like a wall between the users and the functionality in the tool and fulfills the third security requirement in Section 4.1. As you can see, authentication also involves creating new users(point 3 in Figure 4.2), and if they forgot their password, they needed a way to reset it(point 2 in Figure 4.2).

How users could reset their password is presented in Figure 4.4. This functionality was not added as a requirement, but we developed it anyway because it is a basic functionality when using authentication. Users forget their password all the time and this could be a useful.

In Figure 4.3 we can see how users could be registered in the tool. The registration was open for everyone, and gave each user registering a profile the "user" privilege. This indicated that the user had access to basic functionality within the tool, but not administrate features like other users or data population. The figures presented in this section is from a user with administrator privileges. The only difference is that the administrator also has access to the "AdminControl" tab as shown in the figure.

4.2.2 Security Requirement Eliciting

When a user is authenticated by logging in, the first page that is presented is the "Search" tab as shown in Figure 4.5. This is where users can search or browse for information.

The tool is populated with Security Requirement Patterns (SRPs), Security Design Patterns (SDPs) and categories. The patterns are added under a category, but it can also be a connection between the patterns as illustrated in the data model presented in Figure 2.3.

The left list represents the content in the tool. By selecting some of the content, the data about it will be listed on the right side. In Figure 4.5, the category at the top(by default) is selected. The information about the category is presented on the right side. Figure 4.10, 4.11 and 4.12 outlines how the different content is presented within the tool.

As the tool contains different types of data, we choose to present this by using

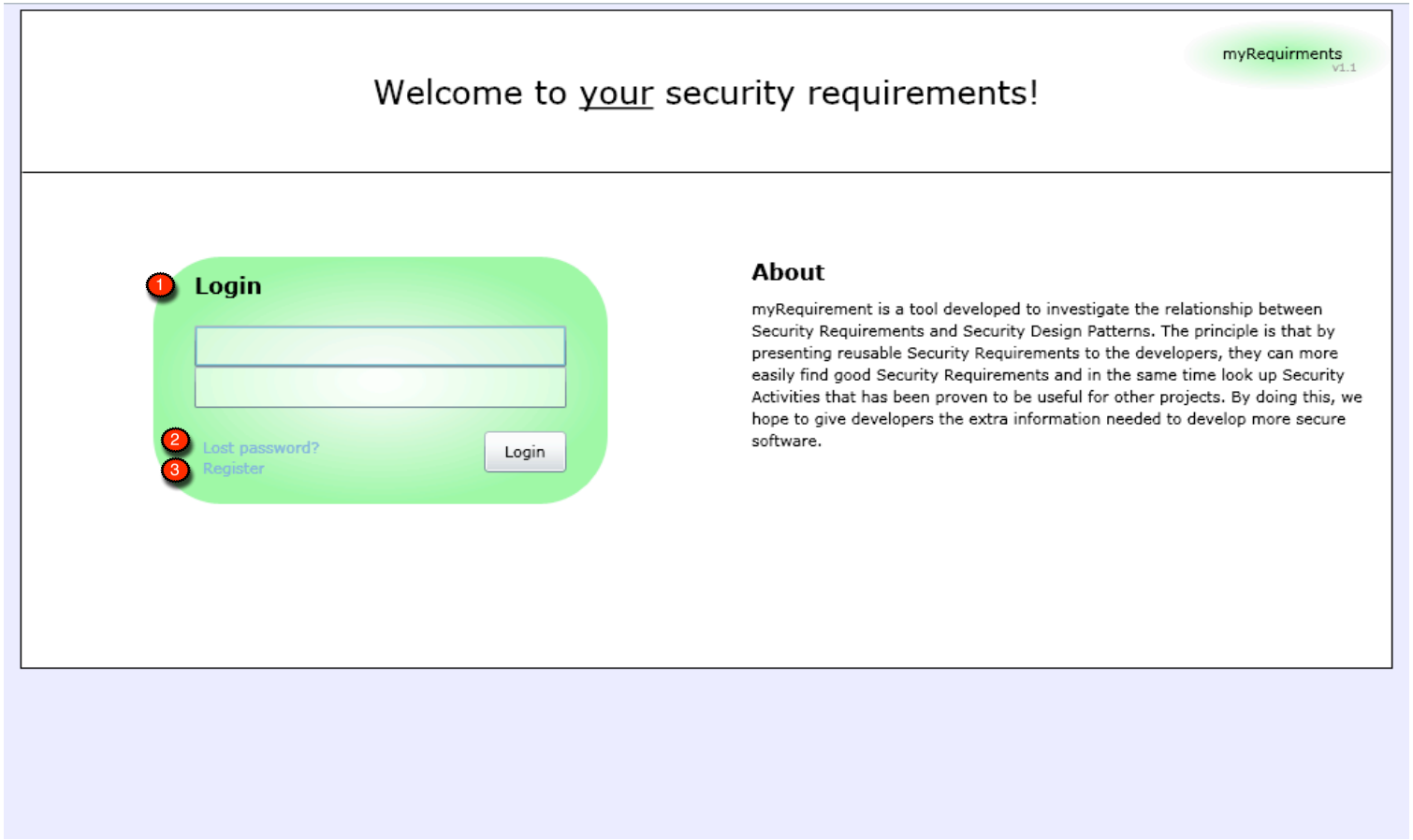


Figure 4.2: The figure shows how the login page looks. This is the first page a user will se when entering the tool, and all functionality is hidden until the user is authenticated.

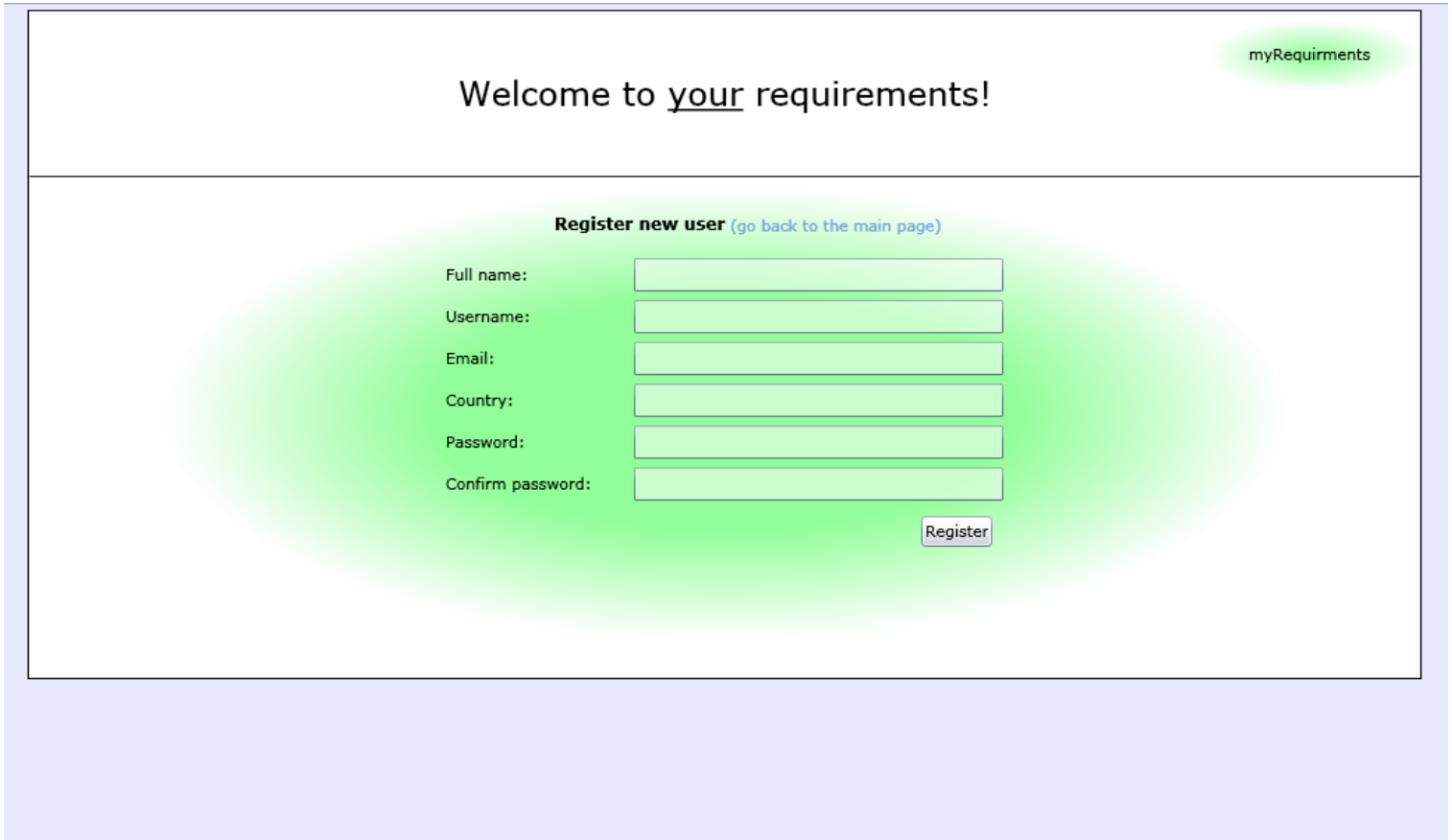


Figure 4.3: The figure shows how new users can register a user account. The tool is publicly available, and anyone can register and test the tool.

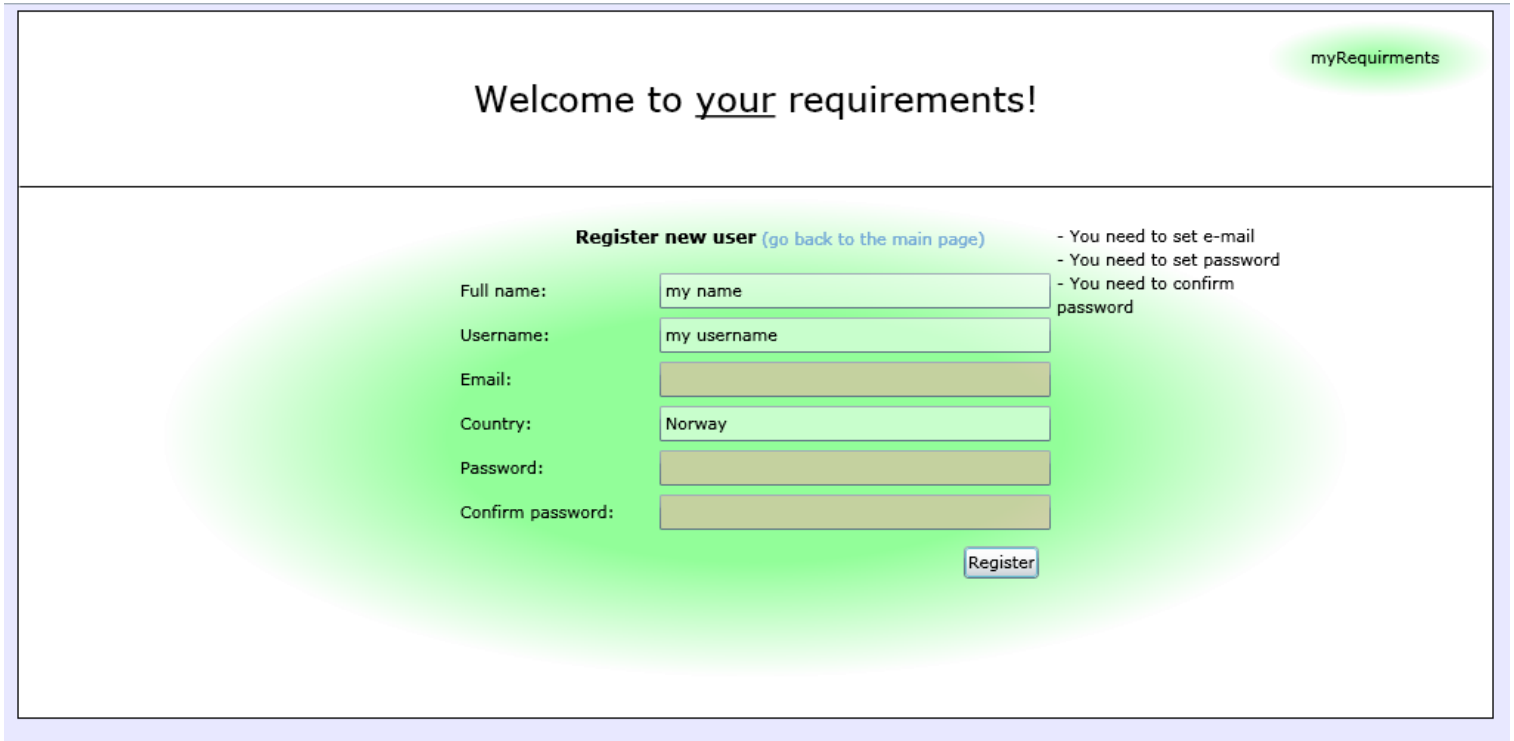


Figure 4.4: If the users forgets their password, it can be reset with the built in reseting mechanism.

different colors. The search filter presented in Figure 4.7 is one way that indicate what color that represent each content. The blue color represents categories, green the SRPs, while the red color presents the SDPs. Also, users can create their own SRPs, presented by a darker green color.

In Figure 4.6, an example of a search(point 1 in Figure 4.6) is demonstrated. This is the first functional requirement of the tool. From the search result(point 3 in Figure 4.6) we can see how the content is presented. More filters have been enabled, and explains why some content has another color. The default event, if the user has not done otherwise, is that the top result is selected. In this case, a SDP is selected, and the information it contains is presented on the right side(point 4 in Figure 4.6).

It is also possible to browse for content which is the second requirement for the tool. Figure 4.8 is an example of how it looks when browsing a category. In the category, two SRPs are revealed. A SRP has a checkbox beside the browse button. This is an indication if the SRP is added to the private list. The private list is configured in the settings tab and is presented as a "project". A user can have many projects, but only one project can be active at one time. The active project is the one the pattern will be added to, if we would select a pattern.

Sometimes it could be useful to know the relation between the data. For example if we knew that we needed to use a design pattern, what SRPs is related to it? That way, we could find SDPs that we had not thought of. Also, it could be valuable to see what SDPs are suggested knowing what security requirement we want. This would be the mitigation techniques for that requirement pattern(the third functional requirement for the tool). Both cases is possible, and is presented in Figure 4.9.

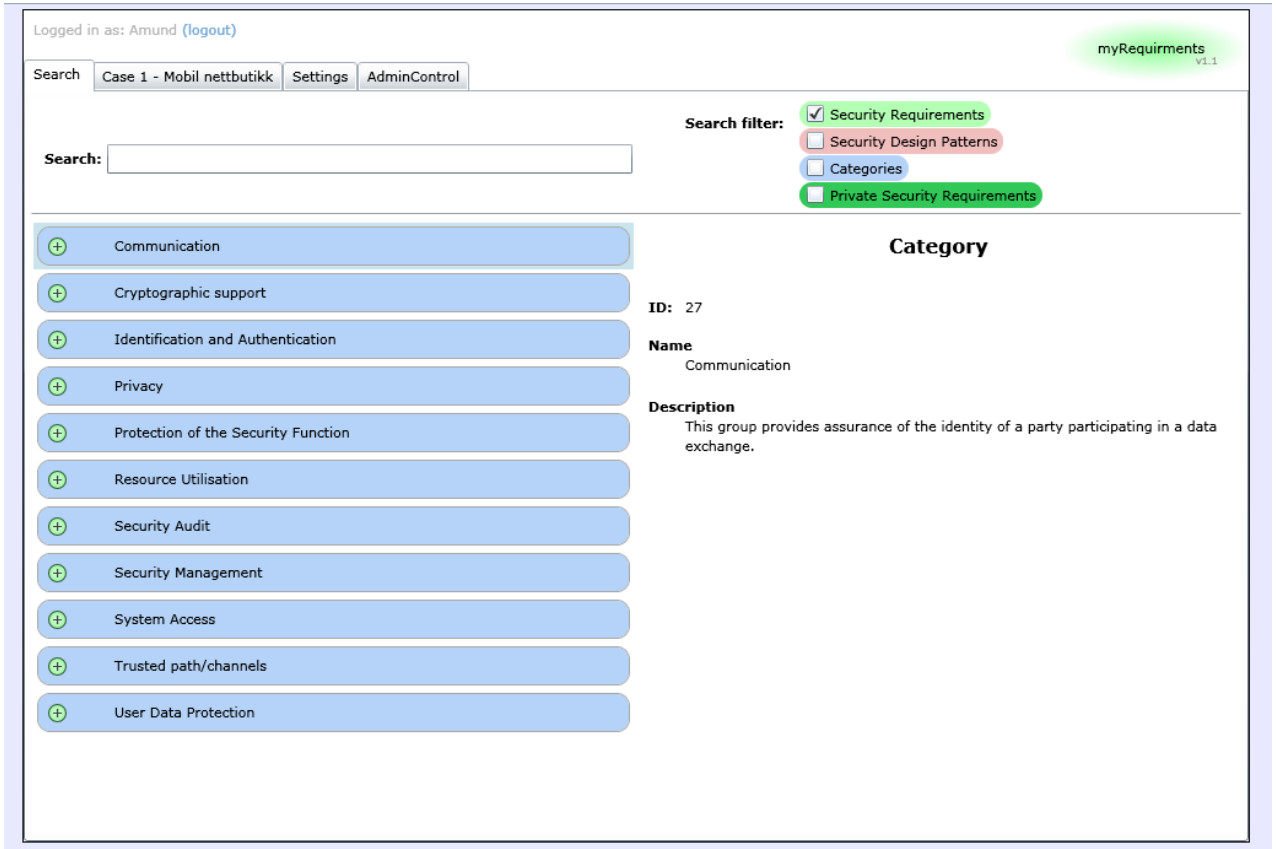


Figure 4.5: The figure presents how the first page looks when a user has logged in. The left list represents categories which are browsable by clicking the "plus" button on them. Information about the content is presented on the right side.

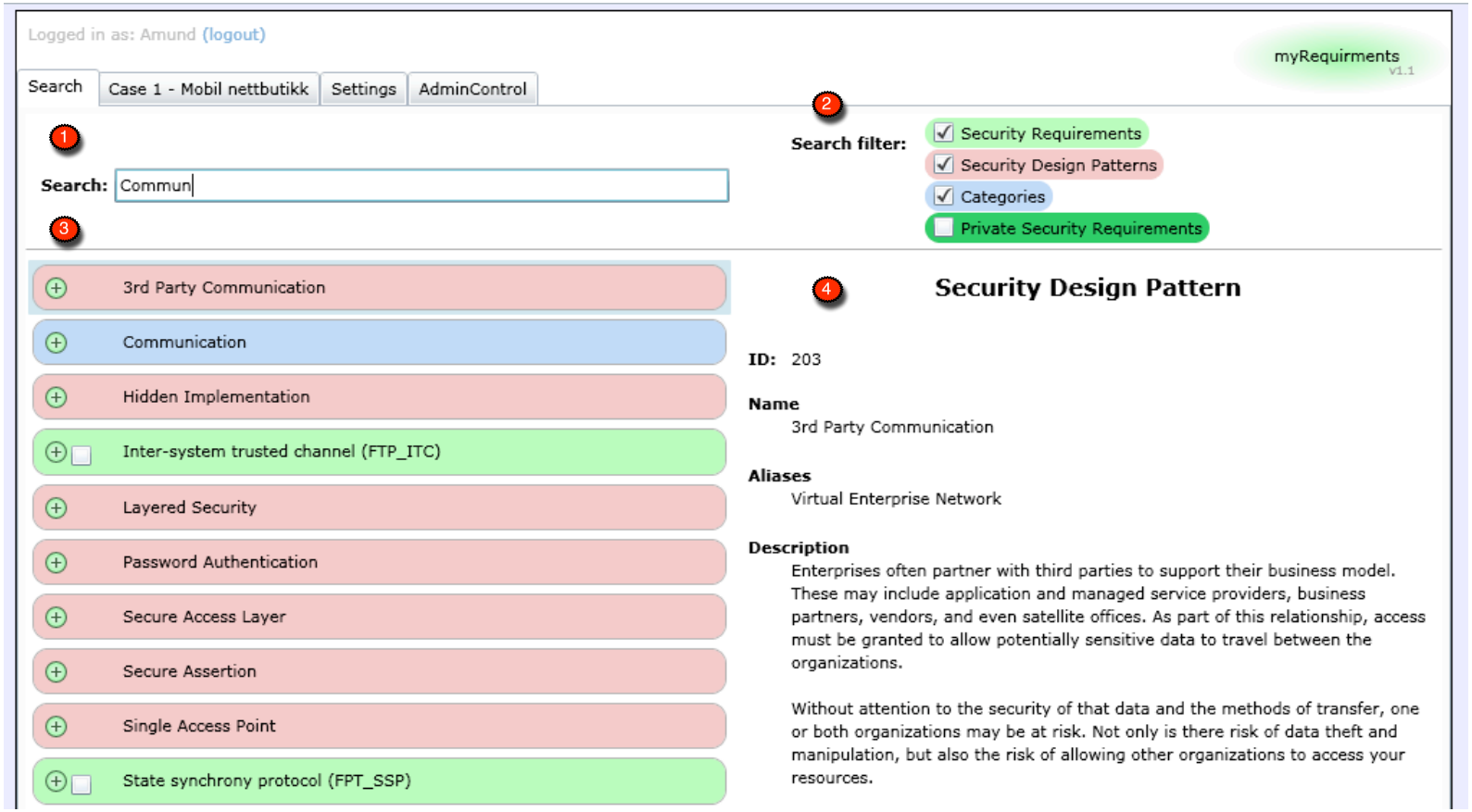


Figure 4.6: The figure shows a search in progress. As the user is typing, the result list will automatically update. The result contains different types of content because we have enabled more filters.

Figure 4.7: The search filter is used to filter the content that is presented by the tool. The filter is located at point 2 in Figure 4.5. Note that the colors, also represent the type of content in the search result list. By default, only the security requirement filter is checked to make sure the users are not overwhelmed by all the content.

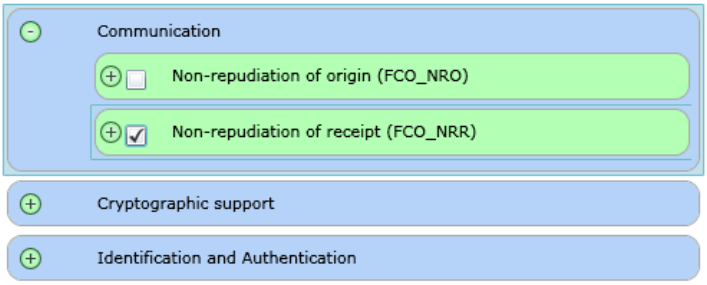
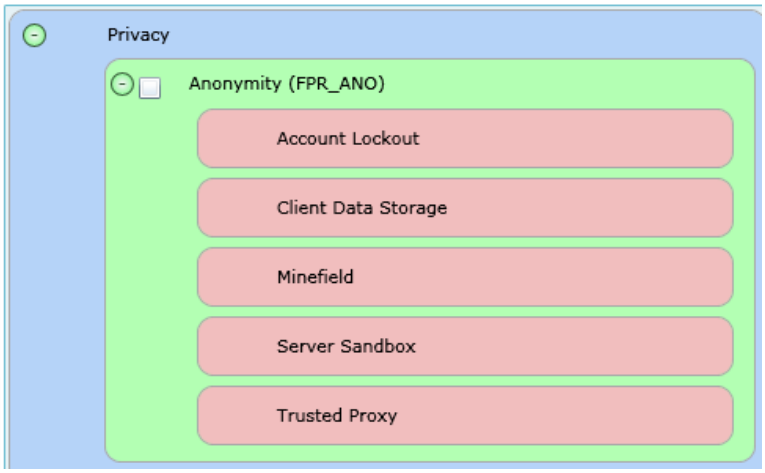
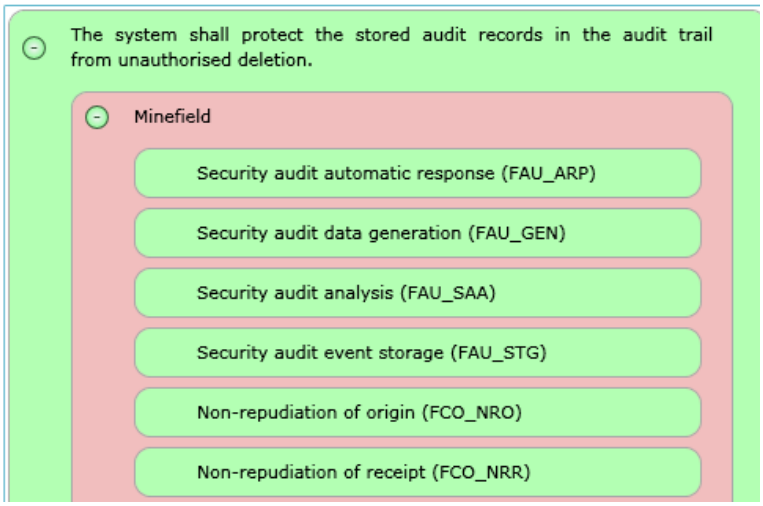


Figure 4.8: The figure shows how it is possible to browse for content. By using the browse button("plus" sign)in the top left of the content, new content will emerge based on what filters that are activated. The checkbox next to the browse button, represent that a pattern can be added to the private list of requirements.



(a) Case 1



(b) Case 2

Figure 4.9: By browsing the content, it is possible to find other patterns that resemble, or have a relation to the current one. In Figure (a), the design patterns that are mapped to the security requirement "Anonymity" are shown. Last, in Figure (b), how other requirement patterns can be found by using design patterns are shown. For this to be possible, it is important that the corresponding filters are enabled.

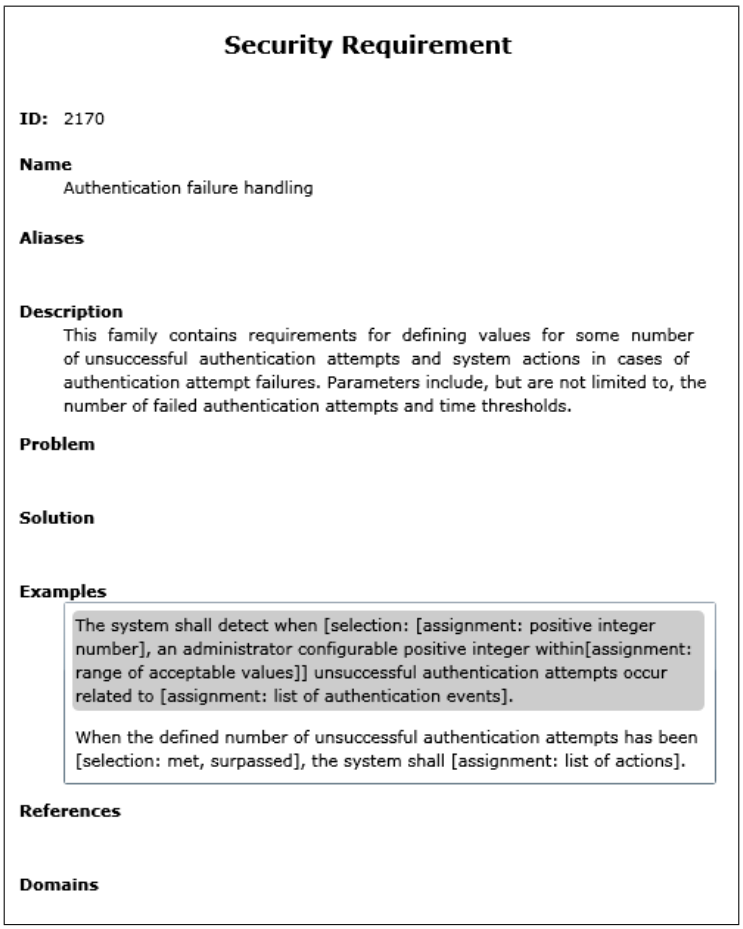


Figure 4.10: An example of how a Security Requirement Pattern is presented when it is selected in the list(point 4 in Figure 4.5).

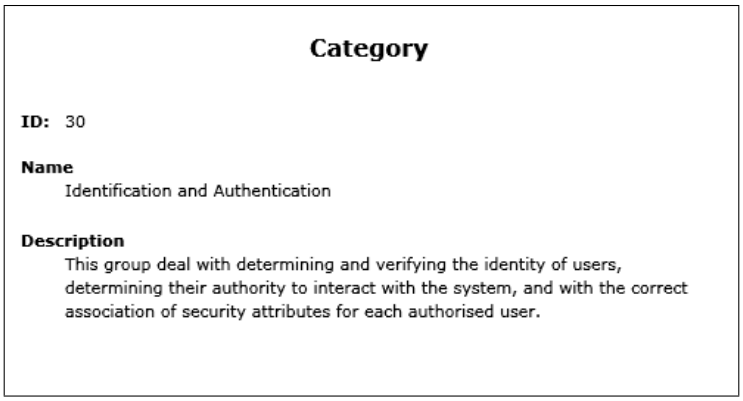


Figure 4.11: An example of how a category is presented when it is selected in the list(point 4 in Figure 4.5).

Figure 4.12: An example of how an SDP is presented when it is selected in the list(point 4 in Figure 4.5).

4.2.3 Private Security Requirements

The private security requirement list is located between the "Search" and "Settings" tab. The name of the tab varies with the name of the active project. We will thorough this section call it the project tab.

Before a user begins using the tool, it is important to create a new project. Before this is done, the project tab is hidden and it is not possible to add SRPs.

In Figure 4.13 an example of how it looks when we got a project called "Case 1 - Mobil nettbuikk" is shown. From the figure, we can see how the content is presented. On the left side the security requirements that the user has elicited is presented. On the right side, the information about the selected pattern is outlined. This has the same analogy as the content in the "Search" page. The list on the left side, is a presentation of the private SRPs for the user, and is the fourth functional security requirement for the tool.

The security requirement list is created using SRPs. This list is populated either by using patterns from the "Search" page or by creating new patterns by using the "(add new)" link.

A fundamental philosophy in the tool is that SRPs have examples of how a requirement can be described. These examples should be changed to fit into the case and used to populate the requirement list.

In Figure 4.13, the selected pattern includes examples of requirements(point 3 in Figure 4.13). When the user selects one of these examples, the text in the requirement list on the left will change to that example. If the user created a new SRP, the text in the requirement list on left side would be empty because the lack of examples. To make an example that can be selected, the user has to "edit" the pattern(point 4 in Figure 4.13), and create a new example that could be selected.

The tool also gives guidelines for choosing an appropriate design of the system(point 9 in Figure 4.13). The SRPs are mapped to SDPs that can help the user in the design phase. By clicking the browse button on the requirement, suggestions on SDPs will emerge(point 2 in Figure 4.13). The SDPs that emerges, are of course clickable to get more information.

It is possible to create, delete and modify a SRP which is the sixth functional requirement of the tool. Point 4 and 5 in Figure 4.13 shows how it is possible to change and delete a pattern. Figure 4.14, gives an overview of how it is possible to change an SRP. When changing a pattern, it is possible to change

the priority of the pattern as well(point 3 in Figure 4.14). The requirement list on the left, is sorted by this priority, as shown in point 6 in 4.13 and point 3 in Figure 4.14. This functionality covers the tenth functional requirement of the tool. Also, it is possible to create new examples(point 1), suggest a pattern as public (point 3), and save the pattern(point 4).

The possibility to suggest new data were one of the functional requirements with the priority "beneficial". For now, the suggestion of a new pattern is only saved in the database, and do not have any administrator capabilities to review, decline or accept a pattern.

The eighth functional requirement is not directly covered by the tool, but can be managed by a "copy and paste" - "create new requirement" method. The user has to copy the example text that he wants, create a new security requirement, change the requirement, add a new example, and paste in the text. A slow and cumbersome method, but it works. For further work, it should be possible to copy or duplicate an example into a new SRP.

All data that the user creates or changes will be saved as a private SRP. The evident reason is that all changes by a user should be private.

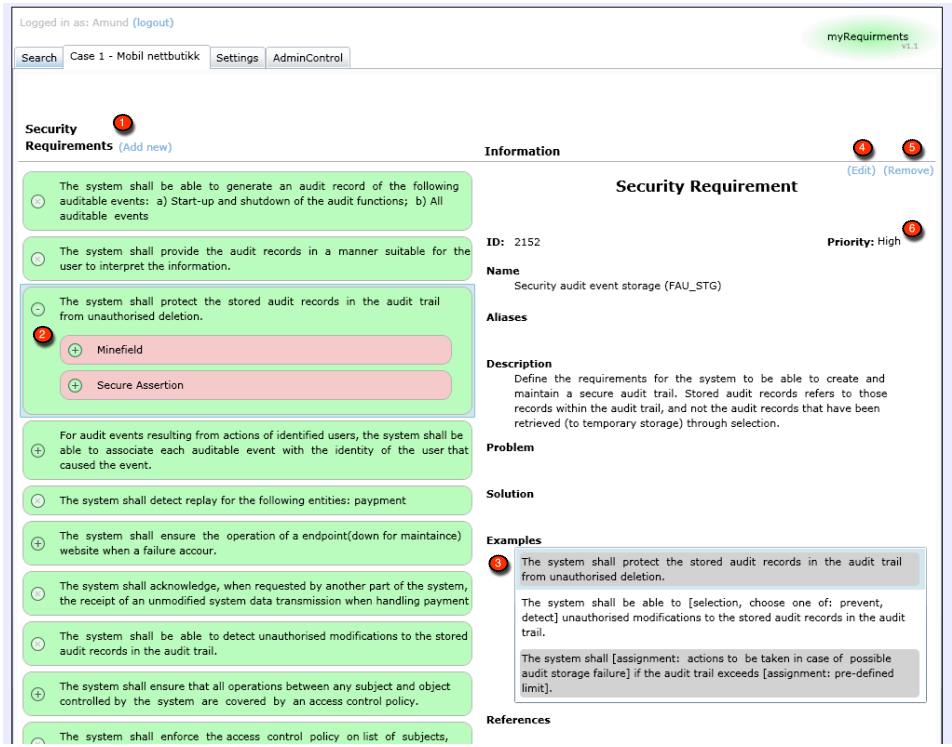


Figure 4.13: The figure displays the page that represents the active project. The left list is the security requirements the user has selected. The list represents the patterns as well. A requirement is actually a pattern. By selecting one of the requirements, the pattern information will be revealed on the right side of the list. The text in each requirement is based on the examples in a pattern. By selecting one of the examples, the requirement will be changed to that example. New examples can be created or changed to the users liking.

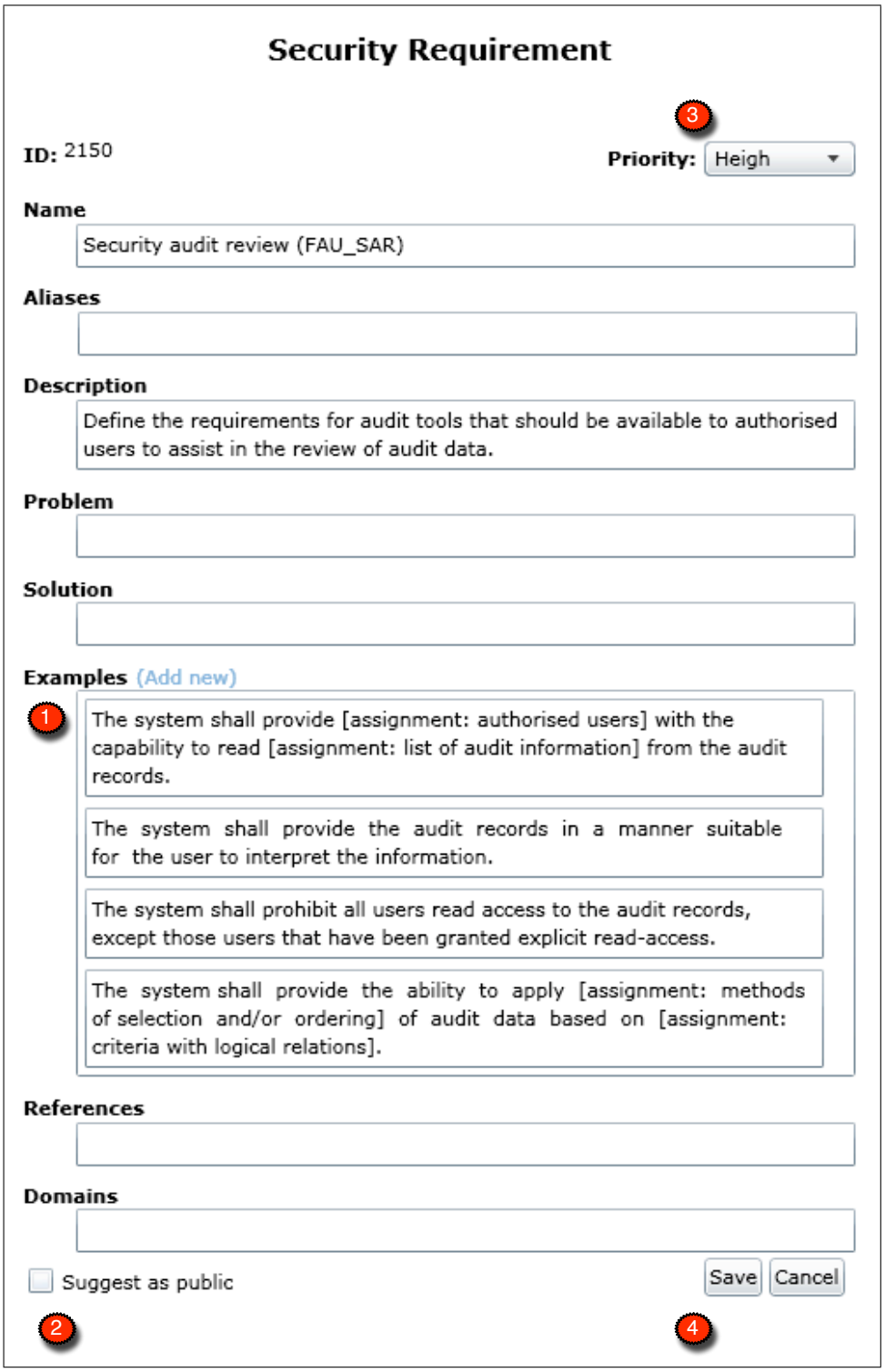


Figure 4.14: The figure shows how changes can be done with a SRP. When the pattern is saved, the pattern will become a private SRP that only the user has access to. The requirement will be searchable and could be used in other projects.

4.2.4 Project Administration

Some users are working on more than one system at a time, e.g., in the user test, we were testing two different cases which involved that each user had to gather security requirements for both cases. When such events occur it is useful to store each set of security requirements separately. In the tool, we have created the possibility to have projects, which could be used to separate the systems, which compensate for the eleventh functional requirement.

On the left side in Figure 4.15, the projects are listed. On the right side, information about the selected project is displayed.

On the left side, it is possible to create a new project(point 3 in Figure 4.15), by clicking the "(add new)" link. Also, it is possible to delete a project by clicking the delete button(point 4 in Figure 4.15). Private SRPs will not be deleted, as they also could have value for some of the other projects the user has. For the same reason, private SRPs that are deleted from the project, is not entirely gone. Only from the project. The fact is that the tool needs a better way of administrating these requirements. Point 1 in Figure 4.15 is the radio button which indicate what project is active.

On the right side, information about the selected project is listed. As described by Figure 4.15, it is possible to save information about a project(name and description), and we can see some statistics, like how many security requirements that are found, and how many security mechanisms that are suggested.

That the project has statistics of both security requirements and *private* security requirements are to distinguish between requirements that the user has changed in some way. Also, when talking about security requirements in this context, we are actually talking about patterns. In other words, a SRP that is changed, will automatically become a *private* SRP.

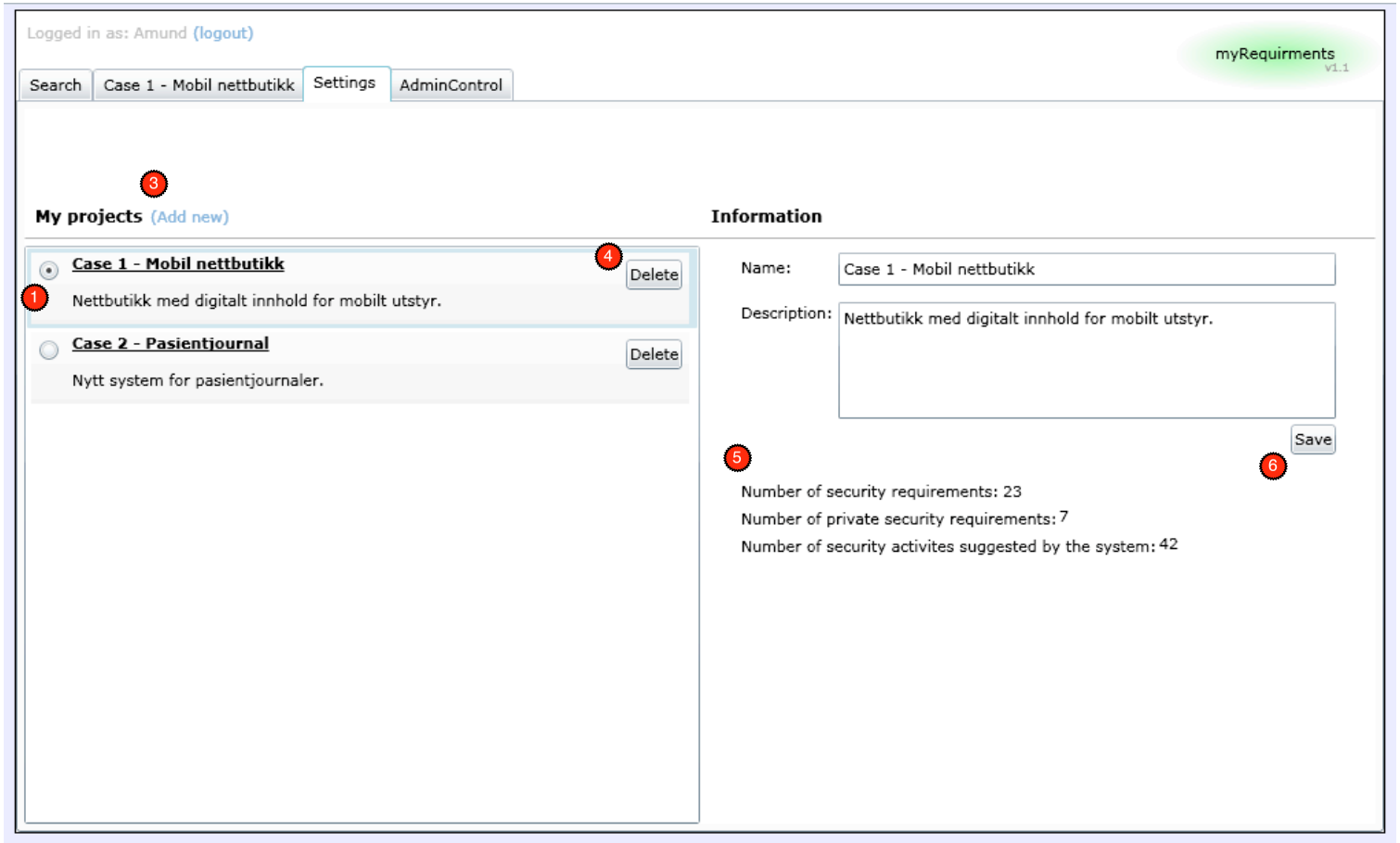


Figure 4.15: The figure shows how projects are managed. A user can have many projects, and a project can include many requirements. The active project(point 1 in the figure), is the project which is enabled in the top pane. New SRPs will be added to the active project.

4.2.5 Tool Administration

As the tool contains data and mapping between data, it also includes an administrator part where the data can be managed.

In Figure 4.16 how the data are administrated is shown. The figure displays the SRPs that the tool has been populated with on the left side. On the right side, the information about the selected patterns(point 1 in Figure 4.16) are shown. By clicking the "Add new" button(point 2 in Figure 4.16), a new pattern can be created. A pattern can be both deleted and changed with the buttons on the right side(point 3 in Figure 4.16). Point 4 shows how the mapping between the content is created. By having the identification number of some content, in this case a SDP, a mapping can be created by providing the number in the input field and clicking the "ok" button. The pattern that corresponds to that identification number will then emerge in the list with mapped patterns.

User administration is also an important part of the tool. In Figure 4.17 we can see how this was managed. We can see that the tool support a "userlevel" which corresponds to the access level the user has been granted. Also, an important part was the possibility to disable accounts when the user test was finished, and get access to the participants data. By clicking the "GoTo" button, we could get access to the information that the participants had created.

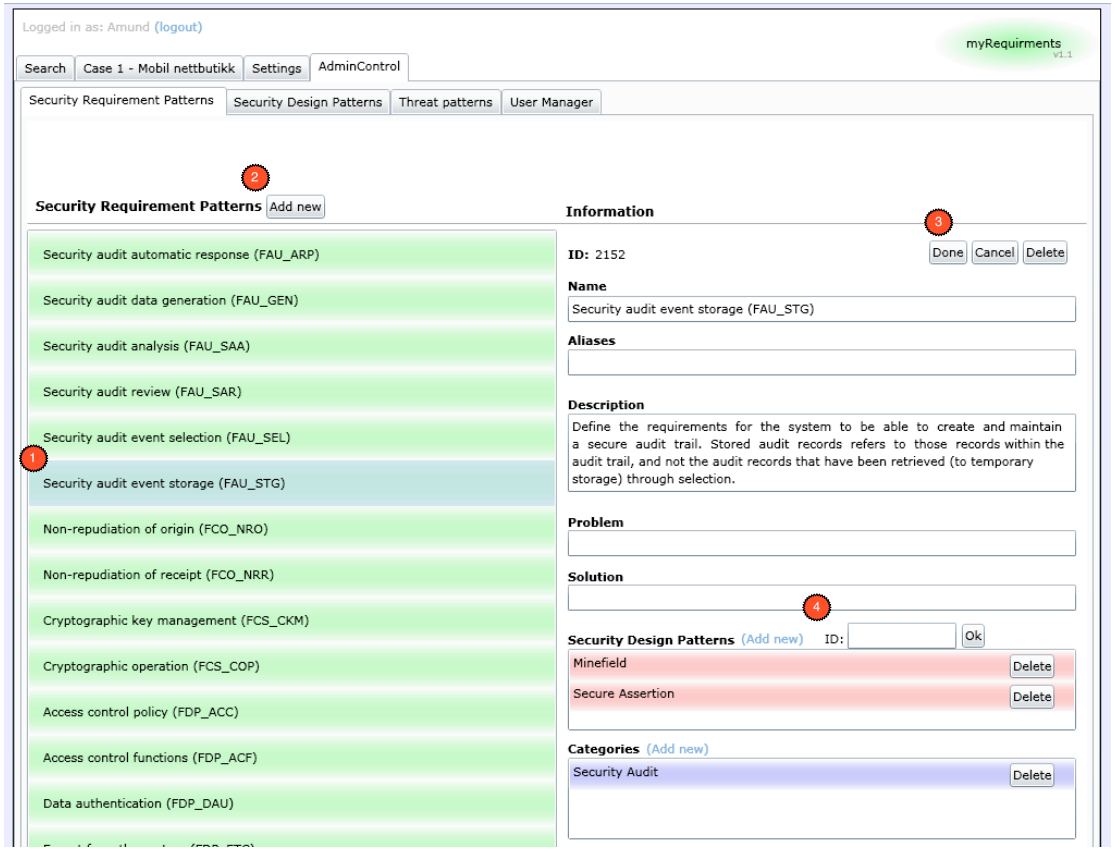


Figure 4.16: The figure shows how patterns and their mapping are administrated. Point 4 in the figure, shows how a new mapping can be added by suppling a pattern identification number.

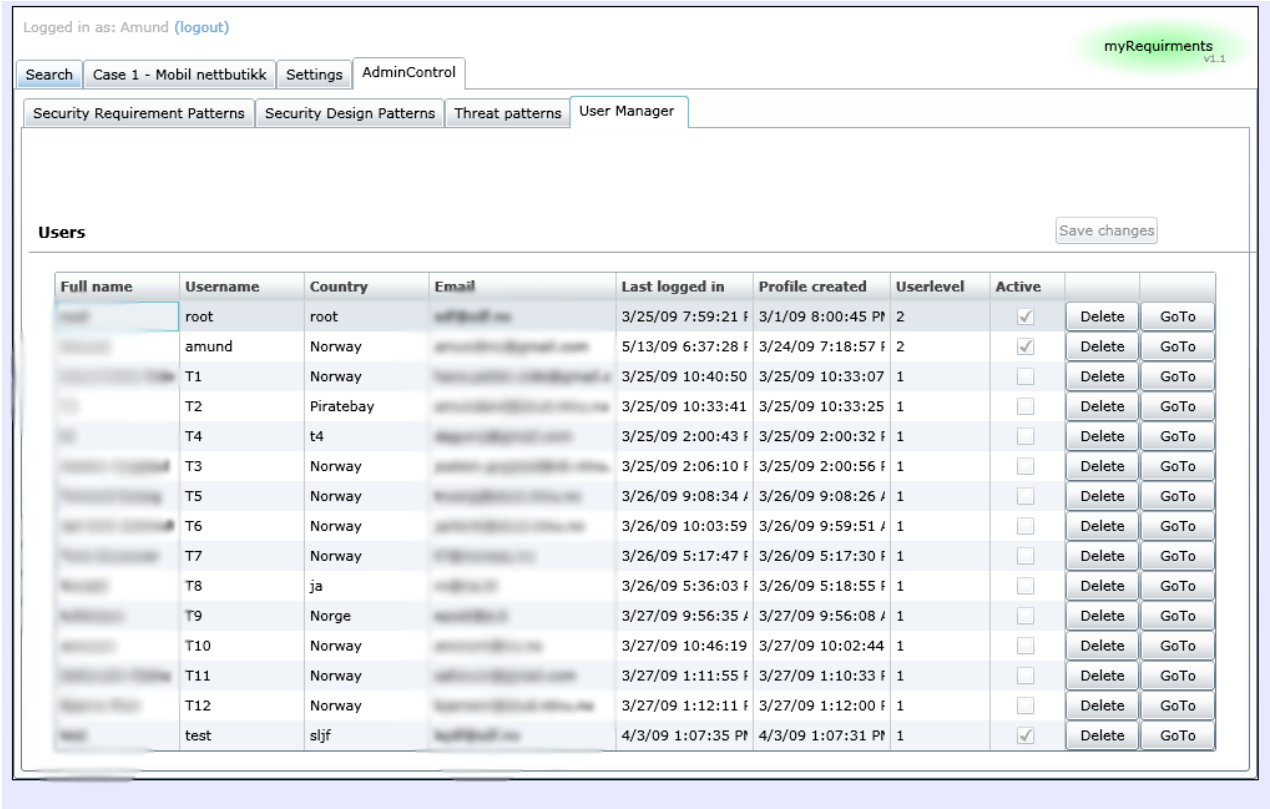


Figure 4.17: The figure shows how administrators can administrate users. In the figure, users from the user test are listed, and all the users are deactivated because the test is finished and the participants should not have the possibility to login. If they want, they could create another user to test the live demonstration of the tool.

4.3 Validation of Completeness

In this section we list the status of the requirements that were presented in Section 4.1. Table 4.1 lists the status of the functional requirements, non-functional requirements, and the security requirements.

Table 4.1: Status of the functional requirements.

Functional requirement	Priority	Status
1	Essential	Implemented
2	Essential	Implemented
3	Essential	Implemented
4	Essential	Implemented
5	Essential	50%, it is possible by deleting a project, or deleting each requirement one by one, but a specific functional is missing.
6	Essential	Implemented
7	Essential	Implemented
8	Essential	Implemented, but it is cumbersome. Further work should emphasize on making this functionality easier.
9	Essential	Implemented
10	Essential	Implemented, the list is ranked by priority.
11	Essential	Implemented
12	Beneficial	Implemented
13	Beneficial	Implemented
14	Optional	40%, client have the option to suggest a Security Requirement Pattern (SRP) as public. Missing administrator functionality and suggestion for SDP.
Non-functional requirement	**Priority**	**Status**
1	Essential	Implemented
Security requirements	**Priority**	**Status**
1	Essential	Implemented
2	Essential	Implemented
3	Essential	Implemented
4	Essential	Implemented

From the tables, we can see that all the requirements have more or less

been completed. The functional requirement five, can be argued to satisfy 100%, because we have not implemented a distinct feature that does the functionality.

Chapter 5

Results

In this chapter we present a summary of the results gathered when testing the tool. We used the data collected to find results related to the hypotheses listed in Table 1.1.

In Section 5.1 the participants background is listed. Then, in Section 5.2 the results from the security coverage. Later, in Section 5.3, the quality of the security requirements that was elicited will be presented, and at last, in Section 5.4 outline the participants experienced using the different methods.

5.1 Participants Background

The result from the user tests depend on the background and experience of the participants. If one of the users have much experience doing security related activities compared with the others, it could have an impact on the result.

From the questionnaire in Appendix D.1 we gathered information from the participants that we could use to consider if their background could have an impact on the data.

In Figure 5.2 the education level of the participants are presented. The blue color(y-axis) represents the number of years in higher education, while the red, green, purple and light blue color represents security related subjects they have taken.

The education level of the participants were quite equal, but they differed in security related education. From the figure, the number of years of higher

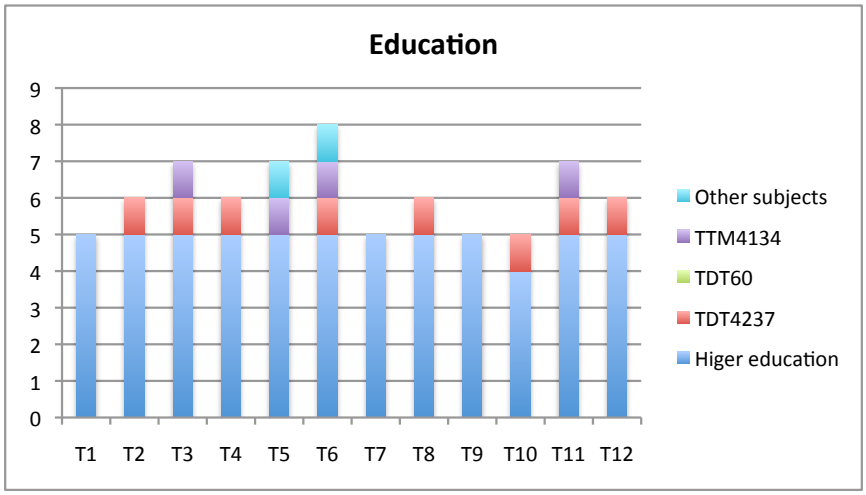


Figure 5.1: Education level of the participants.

education is stable, with a standard deviation of 0.9. All participants had five years of education, except one, who had four. Eight of the twelve participants had at least one security related course while four of them had at least two.

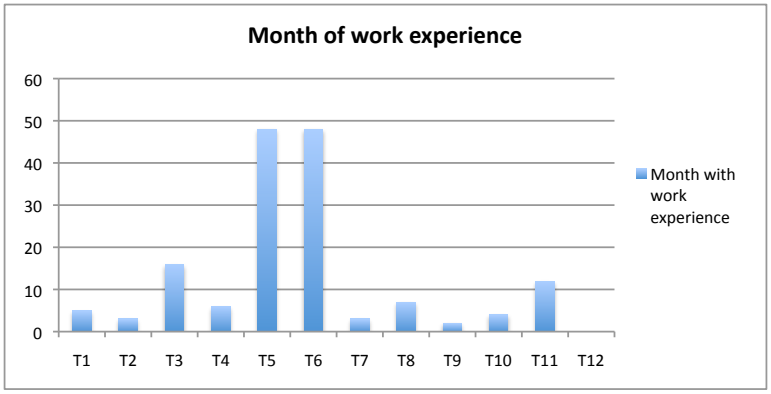
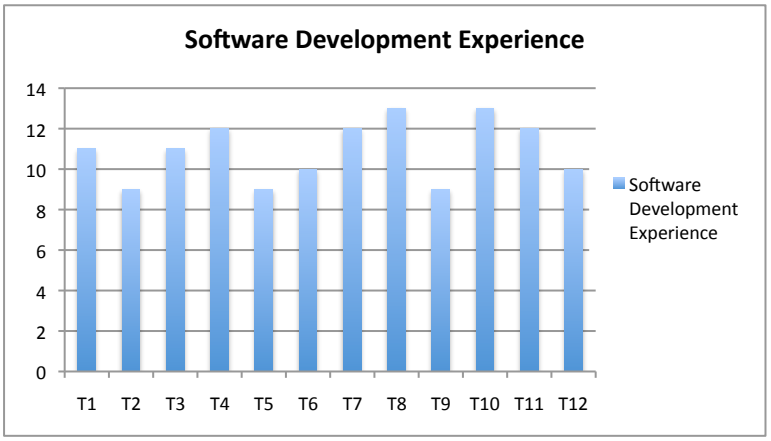


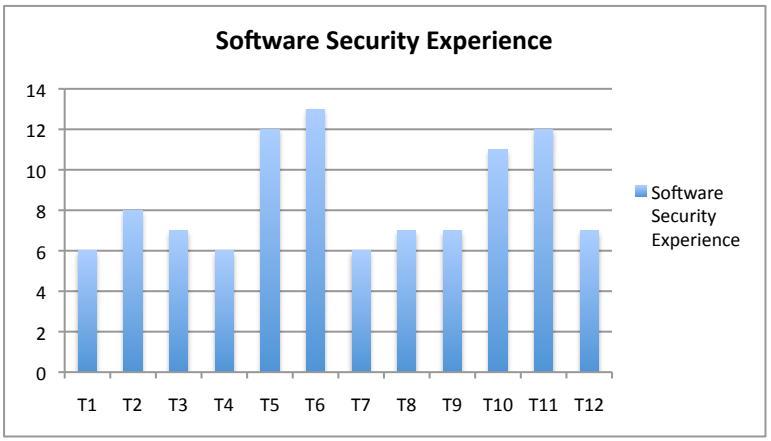
Figure 5.2: Months of work experience with Information Technology.

Figure 5.2 gives an idea of how many months of work experience each of the participants have. The y-value represents number of months with relevant work experience. From the figure, we can see that the experience vary with a standard deviation of 17. Two of the participants had much more experience than the other which explains the high deviation.

In Figure 5.3 the results from the questionnaire about experience is presented. The y-axis represents the score from the questionnaires about experience. The figure gives an indication on how the participants feel about



(a) Software development



(b) Software security

Figure 5.3: How the participants see their own experience in software development and software security.

their own experience in software development and software security.

From the figure, the participants have a relatively equal perception of their own software development experience, with a standard deviation of 1.5. Software security on the other hand is more variable, and has a standard deviation of 2.6.

5.2 Security Coverage

Security coverage is a measure of how well the security requirements and Security Design Pattern (SDP) will protect the system. The numbers are

based on the assumptions and classifications which is discussed in more detail in Section 2.2.4.

In this section we presents the security coverage that the participants managed to get. The security requirements is then presented and at last the SDPs.

5.2.1 Security Requirements

The security requirement coverage was calculated using a list of security aspects that we thought were valuable to include in the requirement phase for each case. This process is more thoroughly described in Section 2.2.4.

Table 5.1: Statistics of coverage with security requirements in Case 1.

	N	Mean	St.dev.	p
Manual	6	35%	0,274	0,358
Tool	6	43%	0,333	

Table 5.2: Statistics of coverage with security requirements in Case 2.

	N	Mean	St.dev.	p
Manual	6	30%	0,321	0,018
Tool	6	52%	0,242	

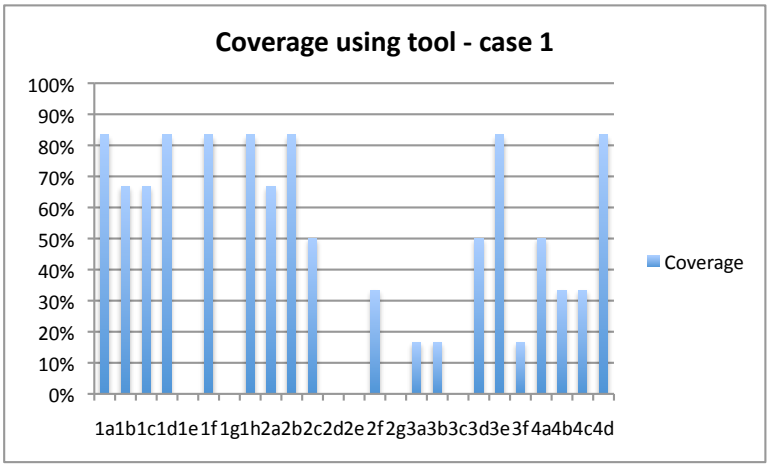
Table 5.1 and 5.2 presents the statistics of how well the security requirements covered the security of the system with each method.

In Case 1, the manual method covered 35%, while using the tool covered 43%. The values were not statistically significantly different with a p-value of 0.358 using the two paired t-test.

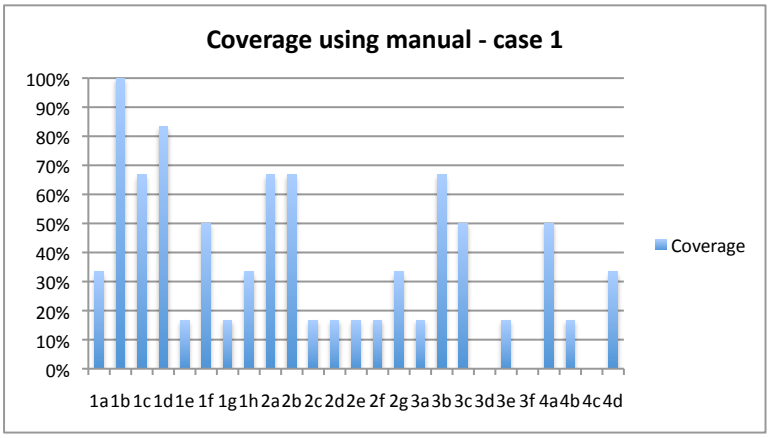
In Case 2, the manual method covered 30%, and the tool method 52%. The values were statistically significantly different with a p-value of 0.018, $p \leq 0.05$ using the two paired t-test.

Security Coverage by Security Aspect

Figure 5.4 gives an overview over which security aspect that was covered in Case 1. From the figure, with the tool method there were some security aspects that got covered a much more than others. With the manual method, more security aspects were covered, but in a less degree. The tool method had a 33% standard deviation, while the manual method had 27%.



(a) With the tool method



(b) With the manual method

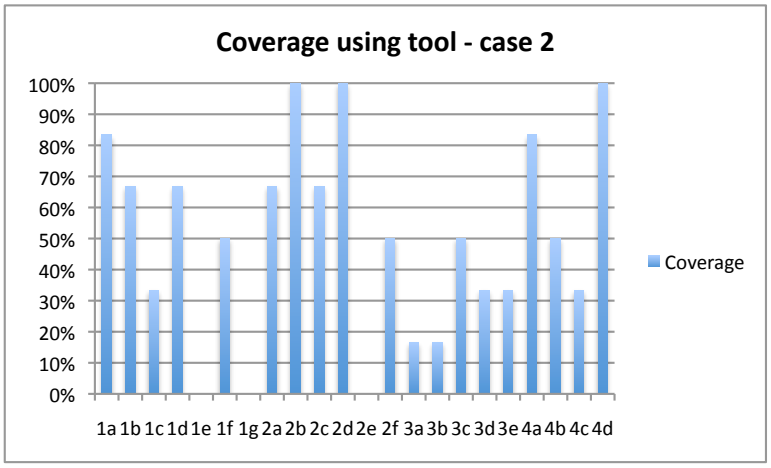
Figure 5.4: What security aspects each method covered in Case 1

Figure 5.5 gives the same overview of security aspects that were covered, but with Case 2. In this case, security aspects that are covered looks more even. The tool method had a 32% standard deviation, while the manual method had 24%.

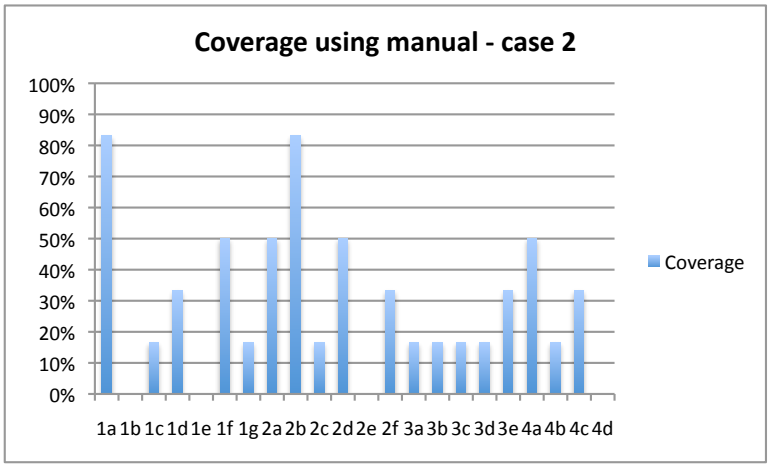
Security Coverage by Participants

Coverage by participants are how good each of the participant covered the security of the system.

In Figure 5.6 the result from each participant using the manual and the tool method are presented. User Group A is the participants who used the



(a) With the tool method



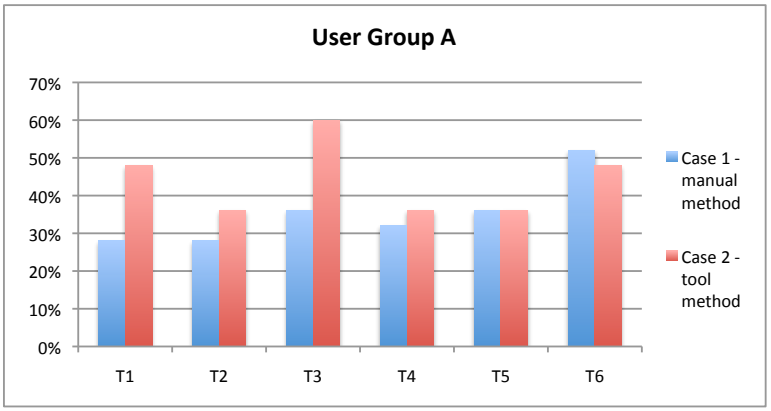
(b) With the manual method

Figure 5.5: What security aspects each method covered in Case 2

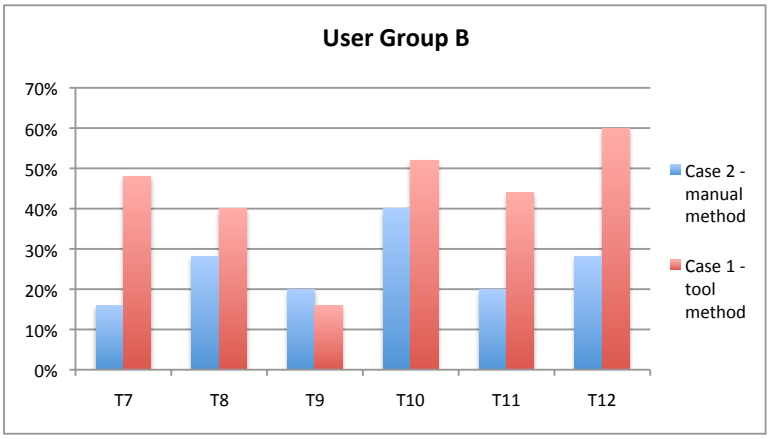
manual method first, while User Group B started with the tool method.

From the result, there is a variance between the results. In Case 1, using the tool method had a standard deviation of 15%, while using the manual method had 8.9%. In Case 2, using the tool method had a deviation of 9.8%, and the manual method 8.6%.

Also, some scores were unusually high or low in User Group B. With the tool method, T9 had a very low score compared with others. The score was actually 27% lower than the average score, and the only one in User Group B who scored higher using the manual method. Considering the background of T9, the participant did not have any software security education. With



(a) User Group A



(b) User Group B

Figure 5.6: The coverage result in percentage from each participant.

the tool method, T12 had a good score that was 18% better than the average score even though the participant did not have any remarkable background. In User Group A the scores were more even.

5.2.2 Security Design Patterns

The Security Design Pattern (SDP) coverage is how good the participants covered the security of the system design with different patterns. In Section 2.2.4 it is argued that it would be difficult to calculate the coverage, that it could be more valuable to look at the rate the patterns have been selected instead.

Table 5.3 list the different patterns and their abbreviation that are used

thorough this section.

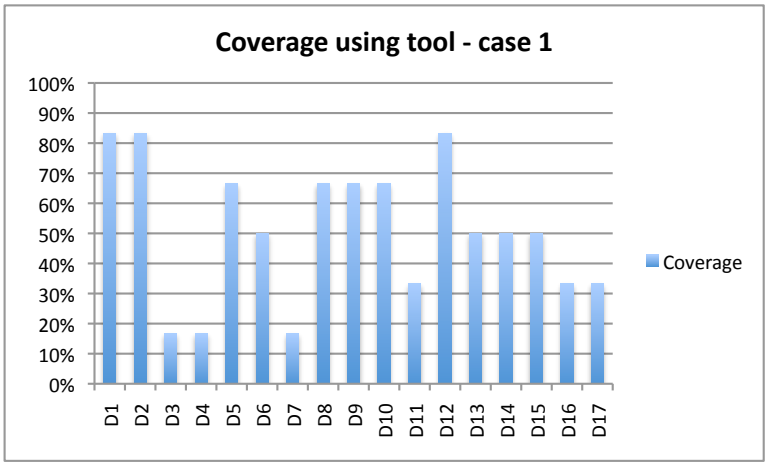
Table 5.3: SDPs and their corresponding abbreviation.

Abbreviation	SDP
D1	Account Lockout
D2	Authenticated Session
D3	Choose the Right Stuff
D4	Client Data Storage
D5	Client Input Filters
D6	Directed Session (mini-pattern)
D7	Hidden Implementation (mini-pattern)
D8	Encrypted Storage
D9	Minefield
D10	Network Address Blacklist
D11	Partitioned Application
D12	Password Authentication
D13	Password Propagation
D14	Secure Assertion
D15	Server Sandbox
D16	Trusted Proxy
D17	Validated Transaction (mini-pattern)

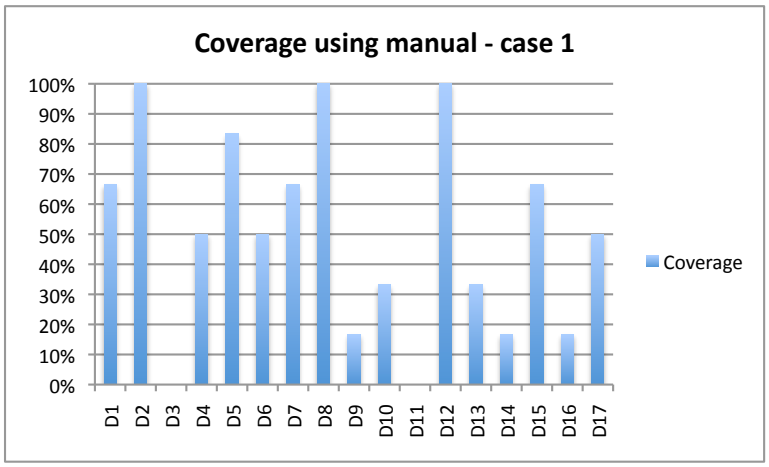
Table 5.4: The popularity of the SDPs using the tool method in Case 1

Tool method – Case 1			
Abbreviation	Most popular patterns	Abbreviation	Lest popular patterns
D1	Account Lockout	D3	Authenticated Session
D2	Authenticated Session	D4	Choose the Right Stuff
D12	Password Authentication	D7	Hidden Implementation

Figure 5.7 lists SDPs that have been selected with the manual and the tool method in Case 1. When using the manual method, pattern D1, D2 and D12 were the most popular once, but none of them were selected by all participants. Pattern D3, D4 and D7 were the least chosen once. When using the manual method, pattern D2, D8 and D12 were the most popular, while pattern D3 and D11 were the least. Also, when using the tool method, none of the patters were selected by all. It is interesting to note that two of the top three patterns were equal.



(a) With the tool method



(b) With the manual method

Figure 5.7: SDPs that were chosen in Case 1

In Figure 5.8 lists which SDPs that were selected in Case 2 using the tool and the manual method. With the tool method, D2, D8 and D12 were the most popular ones, while D3 and D7 were selected the least. When using the manual method, D2, D8 and D12 were the most popular, exactly like in Case 1. D3 was the only pattern that no one selected and was the least popular one, while the second least popular pattern was shared by four other patterns.

All the patterns and what abbreviation they belong to, can be found in Table 5.3. Also, the most and least popular patterns are listed in Table 5.4 and 5.5 for Case 1, and in Table 5.6 and 5.7 for Case 2. A description of the patterns can be found in the security patterns repository [15].

Table 5.5: The popularity of the SDPs using the manual method in Case 1

Manual method – Case 1			
Abbreviation	Most popular patterns	Abbreviation	Lest popular patterns
D2	Authenticated Session	D3	Choose the Right Stuff
D8	Encrypted Storage	D11	Partitioned Application
D12	Password Authentication		

Table 5.6: The popularity of the SDPs using the tool method in Case 2

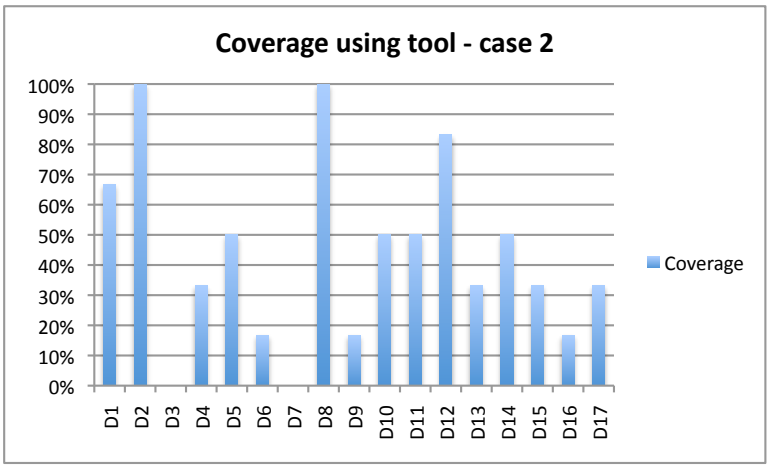
Tool method – Case 2			
Abbreviation	Most popular patterns	Abbreviation	Lest popular patterns
D2	Authenticated Session	D3	Choose the Right Stuff
D8	Encrypted Storage	D7	Hidden Implementation
D12	Password Authentication		

Table 5.7: The popularity of the SDPs using the manual method in Case 2

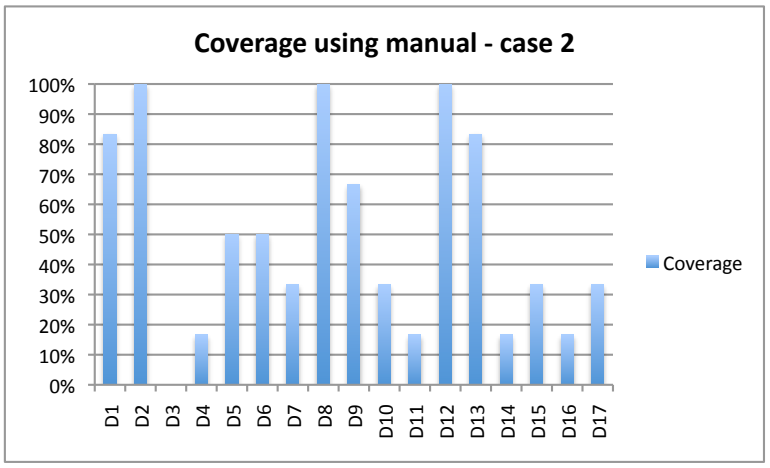
Manual method – Case 2			
Abbreviation	Most popular patterns	Abbreviation	Lest popular patterns
D2	Authenticated Session	D3	Choose the Right Stuff
D8	Encrypted Storage		
D12	Password Authentication		

5.3 Data Quality

The data quality is the quality of the information that the users generated. We examined the security requirements that the participants generated with



(a) With the tool method



(b) With the manual method

Figure 5.8: SDPs that were chosen in Case 2

and without the tool and gave each security requirement a score based on the assumptions described in Section 2.2.4.

Figure 5.9 presents the average quality of the security requirements generated for each case using the manual method. The score of the tool method is left out because we argue that the potential of a security requirement *pattern* should at least be 95%.

It can be seen from Figure 5.9 that the average quality for each case is not that different. In Case 1, the average quality is 71%, with a standard deviation of 13.3%. In Case 2, the average quality is 67% with a standard deviation of 6.4%.

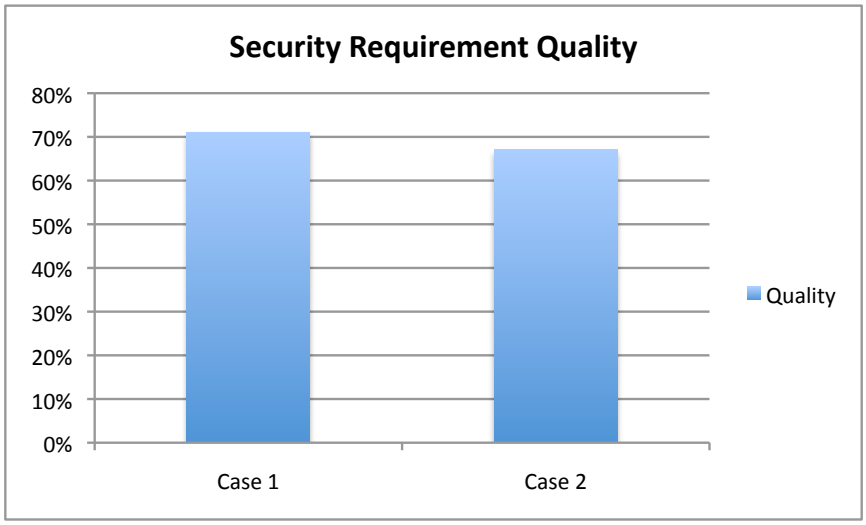


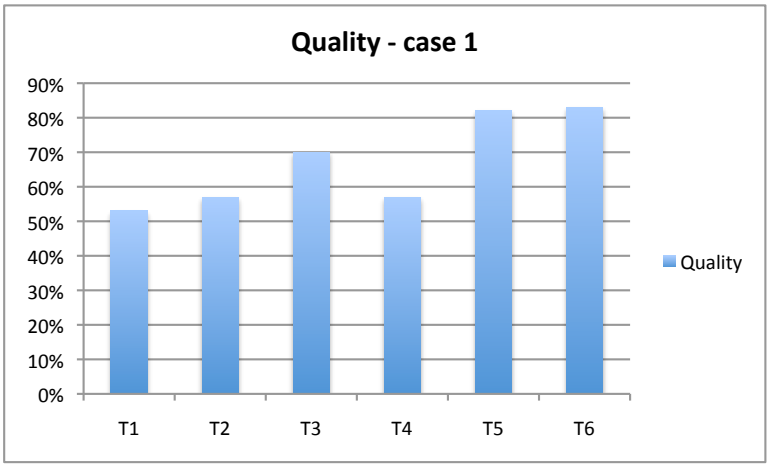
Figure 5.9: The quality of the security requirements for each case using the manual method.

If the quality score would be 95% using the tool(which we argue should be possible in time), we will get a p-value of 0.000 in Case 1, and 0.003 in Case 2 using the two paired t-test . Which means the values are statistically significantly different in both cases because $p \leq 0.05$.

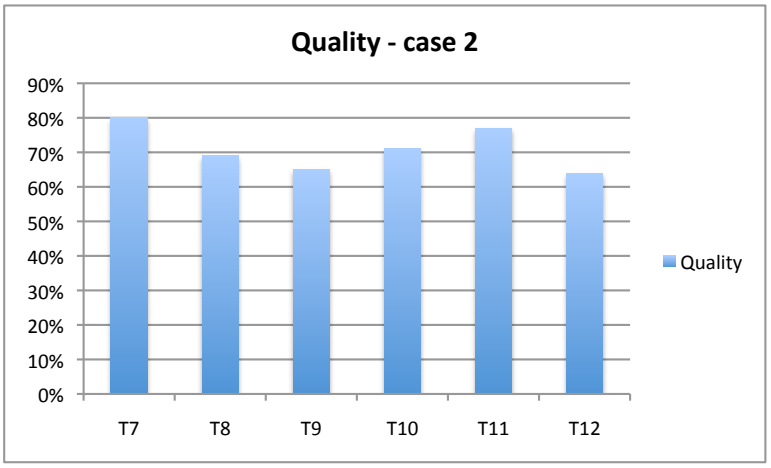
In this case however, the average score on the security requirements added with the tool were about 72%(the overall quality score of the repository). The result has a p-value of 0.71 in Case 1 and 0.39 in Case 2 using the two paired t-test – which is not statistically significantly different at all.

In Figure 5.10 the average quality of the security requirements for each case based on the participants is listed.

From the figure, participant T5 and T6 have a higher quality score than the others with a score of 82% and 83% respectively. The score relates to their high education, work experience and security knowledge. In Case 2, the participant T7 and T11 also have a good quality score of 80% and 77% respectively. T11 has a high education and security knowledge, however T7 does not have any significant background that explains the good score. None of the scores were noticeable low.



(a) Case 1



(b) Case 2

Figure 5.10: The security requirement quality for each case listed by participant.

5.4 Participants Experience From the Test

Participants experience are the participants impression of the method used in each case. By handing out a questionnaire, we collected information to answer the hypotheses in Table 1.1 with the questions from the questionnaire in Appendix D.2. Q1 correspond to question one, Q2 to question two, and so on.

Table 5.8 and 5.9 lists the average results from the questionnaires. The tables shows the participants satisfaction in percentage, the standard deviation between the methods and if the results are statistically significantly different.

Table 5.8: Data from the participants experience after Case 1.

Case 1					
Question	Manual Method	Tool Method	St.dev. Tool	St.dev. Manual	p
Q1	63 %	53 %	0,983	0,816	0,361
Q2	73 %	73 %	1,033	0,516	1,000
Q3	77 %	60 %	0,983	1,265	0,233
Q4	63 %	73 %	1,169	0,816	0,413
Q5	63 %	40 %	1,169	0,632	0,065
Q6	67 %	77 %	1,033	1,329	0,485

Table 5.9: Data from the participants experience after Case 2.

Case 2					
Question	Manual Method	Tool Method	St.dev. Tool	St.dev. Manual	p
Q1	57 %	57 %	0,983	0,408	1,000
Q2	77 %	60 %	0,753	0,632	0,065
Q3	57 %	63 %	1,472	0,753	0,636
Q4	73 %	70 %	1,033	0,837	0,765
Q5	47 %	53 %	0,816	1,366	0,622
Q6	83 %	80 %	0,753	0,632	0,687

In the following subsections we will present the findings.

5.4.1 Quality of Data

Question Q1 and Q2 gathers information about the participants experience of the quality considering the security requirements and Security Design Patterns (SDPs) that they collected. Q1 is the security requirements, and Q2 is the SDPs.

From the data, there is none statistically significantly difference in any of the questions, but the tendency is that the manual method is preferred. Q1 in Case 1 is the manual method and are 10% more desired, with a p-value of 0.36 using the two paired t-test. Q2 in Case 2, the manual method is 17% more desired, which is almost statistically significantly different with a p-value of 0.065 using the two paired t-test.

5.4.2 Confidence in Method

Question Q3 and Q4 gather information about the confidence in the method used. Q3 is about the security requirements, while Q4 is about the SDPs.

From the data, we can not read any statistically significantly difference. In Case 1, the manual method is 17% more desired, with a p-value of 0.23 using the two paired t-test. In Q4, the tool method is desired with 10%, with a p-value of 0.41 using the two paired t-test.

5.4.3 Eliciting Speed

Question Q5 and Q6 gather information about how the participant felt about the speed of the elicitation process. Q5 is the security requirements, while Q6 is the SDPs.

From the data in the table, there were no statistically significantly difference in either case. In Q5, the manual method was 23% more desired, and almost a significantly difference with p-value of 0.065. In Case 2, it was the opposite that the tool method was the most desired one.

5.4.4 Impression of Tool

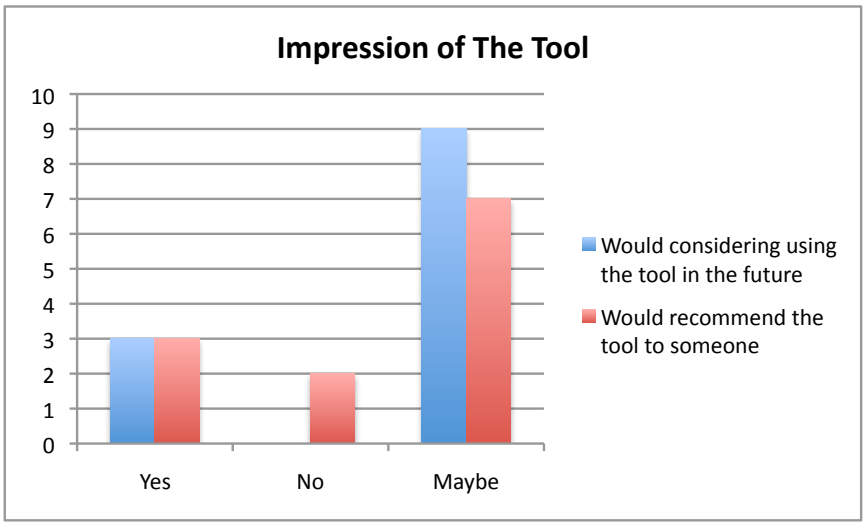


Figure 5.11: What impression the participants got from the tool.

We asked the participants if they would consider using the tool in the future, and if they would recommend the tool to someone. That way we got a

glimpse of their impression of it.

Figure 5.11 lists the results. There were many who answered maybe to both questions, but three, of the total 12 participants, answered that they would consider using and recommend the tool to someone.

Chapter 6

Evaluation and Discussion

In this chapter an evaluation and discussion of the findings in Chapter 5 are presented. As the results are aimed at answering the hypotheses listed in Table 1.1, we discuss the findings in relation to them.

First we evaluate and discuss the results in Section 6.1. Then we summaries how the results suffice to the hypotheses in Section 6.2. At last, in Section 6.3, we discuss the learning effect between the cases.

6.1 Discussion of The Results

The results presented in this section is directly aimed at answering the hypotheses in Table 1.1. The hypotheses is recapitulate in Table 6.1

6.1.1 Participants Background

The participants had an equal education level, but differed much looking at the software security education. This difference could have an impact on the results and make it unreliable if the difference is to large.

The results about the participants experience in software security, varied a great deal. Because user background could have an impact, we expect to see some variability on the other results as well in coherence with the participants background. Also, the results from the questionnaire could depend on their confidence, and should be treated with caution.

Some participants had more work experience than others. We mainly asked

Table 6.1: Recapitulation of the hypotheses listed in Table 1.1

	Discussed in Section	Using data from the questionnaire
H1	6.1.4	It seems easier and quicker to find security requirements and Security Design Patterns (SDPs) with a tool.
H2	6.1.4	The quality seems better when finding security requirements and SDPs with a tool.
H3	6.1.4	Users have a higher confident when using a tool to find security requirements and SDPs.
		Using data from the tasks
H4	6.1.3	Security requirements found with the tool has better quality.
H5	6.1.2	Security requirements found with the tool has better security coverage.
H6	6.1.2	SDPs found with the tool have better security coverage.

students to participate, but we also got participants that were finished and had worked couple of years. This difference could have an impact on the deviation of the results.

By comparing the results from the participants work experience and their confidence level in software development and security, we can clearly see an relation that those with more work experience, also feel more confident.

Before the user test, we knew that the participants could have different security knowledge. By giving the participants some help in form of hints and suggestion of approach(listen in Appendix C), we tried to compensate for the problem.

Also, to eliminate these kinds of threats to validity and reliability, it is important to have a large sample size [44]. In the test, we only got twelve participants, which could have an impact on the reliability of the result.

6.1.2 Security Coverage Test

The results from the security coverage test tended that the tool method had a better score than the manual method. In Case 2, the result was significantly different with a p-value of 0.018, $p \leq 0.05$ using the two paired t-test.

As Case 1 did not have a significant difference, while Case 2 did – some difference between the cases must have caused this.

The theme of the cases could be one cause. Case 2 was about creating a patient journal, which could be less known to the users in contrast to Case 1 that was about a shopping application. This could have made the participants more insecure. The patterns suggested by the tool could have been more directed or better at cover the security in one of the cases. However, we did not notice anything to make such a claim.

The participants background could also be a cause. From their background, we can not see any significant differences between the groups that could indicate the difference. One problem however, is that there could be a difference that we could not pick up with the questionnaire.

Bias when analyzing the requirements could also be a factor. Having a third party to double check the results could have made the process more reliable. This was not possible with the time frame we had.

The variance in the result was rather large, and in some cases this could be mapped against the users background. In some of the results, participants with the best score also had a good foundation of software security education. This correlation could be seen in most cases, but there were also some cases that could not be explained.

Eliciting security requirements are a difficult and mind intensive task, and that the results listed by participants varied a lot were not surprising. The t-test calculates the statistical significantly difference using the variation as consideration [2]. When the data varies, it is difficult to get a significant result without a larger sample size.

When considering the security coverage aspects that we used to evaluate against, the results were weak. The best average result we got, was only 52% coverage. This indicated that either it is very difficult to come up with security requirements, or the security aspects we used to evaluate against were not that good. Considering the damage potential in practice of a threat not being covered, the numbers is not optimistic. Only one exploit is necessary to have huge damage potential.

In Section 2.2.4 we explained the gathering process of security aspects, that was used to analyze the result. The security aspects could be a reason for the varied security coverage. Some security aspects were more valuable than others, and might have been more difficult to come up with. Also, because we tried to cover *all* the threats to the system, some aspects could not be covered by the tool (if not explicitly creating a new security requirement).

One example of an aspect that the tool did not cover is aspect 2e in Case 1, that says: "Card information(Expiry date and CVC-code should not be saved)". The aspect is specific, and could have been covered by a security requirement considering sensitive information.

It could be argued that all the security aspects should have the possibility to be covered by the tool. Doing this in practice is difficult, because the process of discovering aspects are difficult. By adding aspects after the test was finished, we argue that the result will reflect the real world in more extent. It will be difficult, at least for now, to make a tool that covers all possible security requirements in all possible cases.

Security Requirement Patterns (SRPs) populated in the tool could be another reason for the varied result. Some of the security requirement examples could have been more difficult to understand than others (e.g. because of difficult English). Also, some patterns were populated with more examples than others. If the requirement that covered an aspect was at the bottom(or in the middle), the participant might have overlooked it.

The result could also depend on how good we interpreted and evaluated each case. This were especially difficult when evaluating the security requirements from the tool. Not all security requirements were properly filled out, and needed to be interpreted to what aspect it was meant to fulfill.

We also got feedback that some of the SRP were difficult to understand. Not only was the English in some cases difficult, but some users were not sure what the security related expressions actually meant. One of the participants suggested that the tool should have a built in dictionary, which could have made it easier for people that have English as a second language or lack the security vocabulary.

The coverage of SDPs of the system could not be measured in the same way as the security requirements. The repository size populated in the tool only consisted of 17 patterns. We gave the participants too much help by handing out the catalogue of patterns. The catalogue had a good summary in one of the first sections, so the participants could easily get an overview and select the one they thought was right. Because we argue that the process of finding patterns in a real setting would consist of much more investigation, we needed to look at the data from another angle.

By looking at the popularity of the SDPs, we can see a tendency that the same patterns are favored with both methods. This indicates that the tool is not any worse at finding SDPs than the manual method. That the tool is not any worse, also indicates that a mapping between SRPs and SDPs could be used as guidelines in the design phase(goal two of the thesis).

6.1.3 Data Quality Test

Even though we had more or less harsh way of defining a security requirement with "good" quality(as described in Section 2.2.4), the quality of the security requirements were surprisingly good. Looking at the results, we can see that the average quality score is about 70% in each case. When analyzing the security requirement, not all were equally well described(e.g. using security terminology etc.), but they did fulfill the demands. Another reason for the good result could be that every participant had a background with software development, and the quality of functional requirements resembles that of security requirements.

As we have argued in Section 2.2.4, the quality score of SRPs should at least be 95%, we got a significant difference between the methods. A 95% quality score is high and could take some time before we have established a repository with that quality. Even though the result is in the future, we see the potential of using such patterns.

However, we can not draw any conclusion without doing more research and substantiate that the quality of the SRPs can actually get that good.

Fact of the matter is that in the SRPs did not have good enough quality to make the result significantly different. With an overall quality score of 72%(the whole security requirement repository), the tool was at least not performing any worse than the manual method.

That the tool did not perform any worse than using the manual method, indicates that security patterns could be used when eliciting security requirements (goal two with the thesis).

The SRPs populated in the tool were collected from the Common Criteria (CC)(described in Section 3.1.2 and 3.2.4), which has gotten some criticism for being difficult [34]. CC is first and foremost a certification standard where the security requirements are more meant as a common diagnostic assurance than patterns [14]. More research has to be done creating a SRP repository.

As we discovered with the security coverage result, we can also see from the average quality listed by participants, that the result is varying in some degree with the participants background. At some extent this shows that the quality of the security requirements have a relation to the participants experience and background.

As discussed earlier, the result depends on how good we are to interpret and give each security requirement a score with a mutual mind. This is a crucial

part, and to validate the result further, we could e.g include a third party that could analyze the results again and look for deviations.

Looking at the quality of the SDPs had no meaning as we used the same collection of patterns for both methods.

6.1.4 Participants Experience Test

The result from the questionnaires about the participants experience using the methods, had a tendency to favor the manual method.

None of the results were significantly different, but in some cases it was close. The manual method was favored with 17% to find SRPs. Likewise, the perceived speed of finding security requirements with the manual method were favored with 23%, with a p-value of 0.065 between the methods using the two paired t-test.

The high rate favoring the manual method to find SRPs, could be caused by different things. For one, the help the participants got by handing out the catalogue of patterns(with the summary at the beginning), made the task rather easy. Also, the tool could be interpreted as hiding information, because it only gave suggestion on patterns that were mapped with the security requirements. The low number of patterns that were suggested compared with the catalogue could be misunderstood as a restriction.

The tool had some shortcomings. There were several comments about the usability as well as suggestions on improvements. One comment was that the moving between the search and project tab were slow and could maybe be avoided. Another problem that we knew of beforehand, was that there should have been a method to "copy" a SRP because they could contain more than one desired example. To compensate for that, we introduced all the participants to the "Copy and Paste" – "Create New Security Requirement" method as described in Section 4.2.3, before we started the user test. This "method" could have a negative impression of the speed.

When collecting information with a questionnaire, some considerations have to be made. The questions could be misunderstood or the participants could be influenced by external factors, e.g., our presence. That we knew some of the participants could also have an influence as some could for instance feel obligated to answer the questions in some way.

Anonymity is also an issue. If the participants feel that they can somehow be related to the answers, they might be influenced by that. Even though we told the participants that both the questionnaire and the results from

the tasks were anonymous, many of the participant used their full name and email address when they created a new user account in the tool. This indicate that the participants did not feel that the user test was anonymous, or they did not see any reason for being anonymous. As a side note, we should have created user accounts before we presented the tool for the participants.

Some of the data that we collected from the questionnaires were about their own perception of knowledge. The problem is that some people are more self-confident than others, and might answer accordingly. In our results, we could se a mapping between the results and the participants background, but in some cases not.

The data that we gathered about the usefulness of the tool, there were a great deal of participants that *maybe* would consider using the tool in the future and recommend it to someone. This indicate that with further improvements, the participants would be more satisfied. Also, considering the tendency that the manual method was favored and still people consider using the tool in the future, is positive.

Even though the result from the tests had a tendency to favor the tool, it was the opposite when asking the participants. There could be many reasons for this, but we think the most evident reason is that the sample size is to small, and none of the results were actually significant.

One reason for the participants impression could be caused by the illusion of waiting time. When the participants could write down the security requirements manually, the user could write all the time and thus feel more effective. Using the tool, there was a "waiting time" because the participants had to find the requirement, but the tool compensates for this by giving some text "for free".

6.2 Hypotheses

Oates [44] states that if there is insufficient information to answer some questions, we should not completely reject the research report, but use the findings with caution and be wary about treating the report as evidence.

Looking at the hypotheses that we presented in Table 1.1, we found that none of the hypotheses can be confirmed or falsified and remain provisional. From the results, we only got one statistically significant result. The result was from testing the security coverage of the requirements gathered in Case 2, which suggests that H5 – "Security requirements found with the tool has better security coverage." is true. On the other hand, we did not get the

same result in Case 1, which means that no firm conclusion can be drawn.

From the fact that the results did not indicate any significance, aside from the one case about security coverage, it could be argued that we need to do more tests and not use the report as any evidence as Oates [44] explains. Also, this was the first release of the tool, and with more iterations we could most likely make the tool better and maybe get a more positive result.

A tool has both advantages and disadvantages testing the theory. Usability and the data foundation that were populated in the tool could have an influence on the result.

Usability is a huge area of interest and just like software security, has got a widespread recognition the recent years [53]. We had focus on the design and usability of the tool in the design phase. If we had more time, we could have had more focus on actually testing usability instead of only getting comments and suggestions.

The data populated in the tool could have been better. Not much research has been done on SRPs. The number of Security Design Patterns (SDPs) should have been higher. That we populated the tool and created the mapping(presented in Appendix E), could also be a problem as it is not validated by a third party.

A tool has some advantages as well. The results from the experiment could be argued to be more linked with a real world scenario, because it has been tested with users and a real tool. Also, if the theory is successful, the transition to a live version could go faster and with less hassle.

6.3 Learning Effect

As the two cases are not that different, we believe that there could be a learning effect, and possibly an influence between the two cases – that it matter which type of method a participant begins with, and what impressions they get from that.

Looking at the data, we can not see any learning effect. The results are so even, that even though we found an indication of a learning effect, we could not have drawn any conclusion because the data are not significant enough.

On the other hand, some results have the tendency to lean towards no learning effect, at least without any effect on the result. When looking at the result from security requirement coverage. The results from Case 2(that was done last) were worse than in Case 1. If we had significant result in

both cases, we could conclude that if there was a learning effect, it did not matter, because the tool was better anyway.

Of course, with more significant result, it could be valuable to dig even deeper into this assumption about the learning effect, but for now, there is no point of doing so.

Chapter 7

Conclusion

In this chapter we conclude the master thesis, outline the contribution and discuss further work.

The goals by conducting this research was based on the assumptions that patterns could be used when eliciting security requirements and also have a mapping to Security Design Patterns (SDPs) to be used as guidelines in the design phase.

From the theory, we constructed the following research goals(recapitulated from Section 1.2):

- Investigate the transition between security requirements and design in more detail.
- Study how security patterns could be used when eliciting security requirements.
- Consider if a mapping between Security Requirement Patterns (SRPs) and SDP could be used as guidelines in the design phase.

To answer the goals, we created the hypotheses introduced in Table 1.1. We conducted an experiment to test the hypotheses, and came to the conclusions listed in Section 6.2, that none of the hypotheses can be confirmed or falsified and remain provisional. On basis of these conclusions, we conclude the master thesis in this chapter.

All of the research goals have been conducted and described thorough this master thesis. The first point; investigating the transition between requirement and design in more detail was conducted and the important parts

are summarized in chapter 3. The second and third point were studied by developing a tool and investigate the hypotheses and usefulness of it by conducting an experiment with twelve participants.

From our discussion regarding the results, we conclude that none of the results were significant enough to tell if the tool was better or worse than doing it manually. The tendency was that the tool outperformed the manual method when considering the results from the tests. On the other hand, the participants tended to prefer the manual method.

Considering the shortcoming of the tool(discussed in Chapter 6) the potential after some improvements are there. With more iterations focusing on the shortcomings like usability, data quality and the number of patterns we are positive that the result could favor the tool.

That the results did not favor the tool or the manual method gives a clear indication that the theory is plausible, considering that the tool did at least not perform any worse. That the tool performed equally good as the manual method, indicated that security patterns could be used when eliciting security requirements(goal two), and a mapping to the SDPs could be used as guidelines in the design phase(goal three).

On the other hand, from the results, we did not get any picture of how good the theory is compared with other techniques. We did not come across other research that could be used to compare the results. Further research has to be conducted regarding the usefulness and advantage of the theory. We conclude that the theory performed good enough to deserve more research.

7.1 Contribution

The research has contributed to new knowledge on how security requirements can be encapsulated into patterns and mapped to Security Design Patterns (SDPs) to further be used as guidelines in the design phase. Also, our work contributes valuable insight in the usefulness of the method and how it can be utilized in practice.

We also leave behind a fully working prototype that demonstrates the theory in practice at “http://myrequirements.idi.ntnu.no”. Also, the source code can be found under the GPL license at “http://amundmo.github.com/myRequirements/” for further research and investigation.

As the web service is a stand alone implementation that supports both authentication and handling pattern data, it could be used for further research

on the subject. Other clients can easily manipulate the service, and make use of the functionality like authentication, retrieval of patterns and saving data(if they have the authorization). As the source code is publicly available under the GPL license, it would be an easy task to e.g., deploy another copy of the web service to create a repository of security patterns. Documentation of functionality that the web service supports can be found at http://myrequirements.idi.ntnu.no/WebService.svc?wsdl.

7.2 Further work

The data that were populated in the tool should be of better quality. There exists many Security Design Patterns (SDPs) out there, but lack of a common repository makes them difficult to employ and collect. The thought of using Security Requirement Patterns (SRPs) are kind of new, and further work creating, collecting and manage them should be conducted as well as investigating the potential they have.

From the experience with the tool and feedback from the user tests, we uncovered shortcomings that were left as further work. One is that the usage of colors to represent different content is inadequate as some people are color blind. A solution could be to use some sort of icons in addtion to the colors. That way, the colorblinds could also distinguish the content.

Another problem was that a pattern could include more examples, or requirements that the user wanted to include. A solution to duplicate, or selecting more examples should be implemented. We got feedback that there were much navigating between the tabs("Search" and "Project tab"), so a different navigation system should be investigated.

The tool also lack the support to export and share data with others. For example if a user created a list of security requirements for a system, it could be valuable to export them into another tool or have a more printer friendly outline.

We also have some radical thoughts of how the tool could evolve;

Having a more user centric approach where users could share, contribute and discuss data could be beneficial to make users participate in populating the data foundation of the tool. The elicitation of security requirements could be more directed to what domain the system is under. Some requirements could e.g. be pre-filled based on the percentage of the popularity and the system domain.

References

[1] Yahoo usergroup - security patterns. http://tech.groups.yahoo.com/group/securitypatterns/, Last visited: 02/02/09.

[2] Research methods knowledge base. http://www.socialresearchmethods.net/, Last visited: 03/05/09.

[3] Securitypatterns.org. http://securitypatterns.org/, Last visited: 04/04/09.

[4] The open web application security project. http://www.owasp.org/, Last visited: 04/05/2009.

[5] Quince. http://quince.infragistics.com, Last visited: 20/03/2009.

[6] Wikipatterns. http://www.wikipatterns.com/, Last visited: 20/04/2009.

[7] Security focus. http://www.securityfocus.com/, Last visited: 20/04/2009.

[8] Cert. http://www.cert.org/nav/index_purple.html/square.html, Last visited: 20/04/2009.

[9] Portland pattern repository. http://c2.com/ppr/, Last visited: 22/04/2009.

[10] Anderson. *Design of Experiments A Realistic Approach*. Marcel Dekker., NY, 1974.

[11] Sean Barnum and Amit Sethi. Attack patterns - knowing your enemies in order to defeat them. *OMG Software Assurance Workshop: Cigital*, 2007.

[12] Barry W. Boehm and Philip N. Papaccio. Understanding and controlling software costs. *IEEE Trans. Software Eng.*, 14(10): 1462–1477, 1988.

[13] Betty H. C. Cheng, Sascha Konrad, Laura A. Campbell, and Ronald Wassermann. Using security patterns to model and analyze security requirements. In *In IEEE Workshop on Requirements for High Assurance Systems*, pages 13–22, 2003.

[14] CSE, SCSSI, BSI, NLNCSA, CESG, NIST, and NSA. *Common Criteria for Information Technology Security Evaluation*, v3.1 edition.

[15] David S. Tyree James Edwards-Hewitt Darrell M. Kienzle, Matthew C. Elder. Security patterns repository. http://www.scrypt.net/ celer/securitypatterns/, Last visited: 07/05/2009.

[16] David S. Tyree James Edwards-Hewitt Darrell M. Kienzle, Matthew C. Elder. Security patterns security patterns template and tutorial. http://www.scrypt.net/ celer/securitypatterns/, Last visited: 22/05/2009.

[17] Duane L. Davis. *Business Research for Decision Making (with CD-ROM and InfoTrac)*. South-Western College Pub, 6 edition, August 2004. ISBN 0534404820.

[18] Fred D. Davis. User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. *International Journal of Man-Machine Studies*, 38(3):475–487, 1993.

[19] D G Firesmith. Engineering security requirements. *Journal of Object Technology*, (2):53–68, 2003.

[20] Donald Firesmith. Specifying reusable security requirements. *Journal of Object Technology*, 3(1):61–75, 2004.

[21] Centers for Medicare and Medicaid Services. *Selecting a Development Approach*, 2005, Revalidated 2008.

[22] Paolo Giorgini, Fabio Massacci, and John Mylopoulos. Modeling security requirements through ownership, permission and delegation. In *In Proc. of RE'05*, pages 167–176. IEEE Press, 2005.

[23] Winograd T. McKinley H. L. Oh L. Colon-M. McGibbon T. Fedchak E. Goertzel, K. M. and R. Vienneau. Software security assurance : State of the art report (soar). *Information Assurance Technology Analysis Centre (IATAC) and Data and Analysis Center for Software*, 2007.

[24] Charles Haley, Robin Laney, Jonathan Moffett, and Bashar Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1): 133–153, 2008.

[25] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *Management Information Systems Quarterly*, 28(1):75–106, 2004.

[26] Thomas Heyman, Koen Yskout, Riccardo Scandariato, and Wouter Joosen. An analysis of the security patterns landscape. In *SESS '07: Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, page 3, 2007.

[27] Michael Howard. Building more secure software with improved development processes. *IEEE Security and Privacy*, 2(6):63–65, 2004.

[28] Jan Jürjens. Umlsec: Extending uml for secure systems development. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 412–425, 2002. ISBN 3540442545.

[29] Steve Lipner. The trustworthy computing security development lifecycle. In *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference*, pages 2–13, 2004. ISBN 0-7695-2252-1.

[30] Torsten Lodderstedt, David Basin, and Jürgen Doser. *SecureUML: A UML-Based Modeling Language for Model-Driven Security*, pages 426–441. 2002.

[31] Université Catholique De Louvain, Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements-driven information systems engineering: The tropos project, 2002.

[32] John McDermott and Chris Fox. Using abuse case models for security requirements analysis. *acsac*, page 55, 1999.

[33] Gary Mcgraw. *Software Security: Building Security In (Addison-Wesley Software Security Series)*. Addison-Wesley Professional, 2006.

[34] Nancy R. Mead. The common criteria. https://buildsecurityin.us-cert.gov/, Last visited: 02/05/2009.

[35] Nancy R. Mead. Security requirements engineering. https://buildsecurityin.us-cert.gov/, Last visited: 06/05/2009.

[36] Nancy R. Mead. Security requirements engineering. https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/243-BSI.html, Last visited: 10/03/2009.

[37] Nancy R. Mead. Square process. https://buildsecurityin.us-cert.gov/, Last visited: 30/01/2009.

[38] Nancy R. Mead and Ted Stehney. Security quality requirements engineering (square) methodology. *SIGSOFT Softw. Eng. Notes*, 30 (4):1–7, 2005.

[39] Microsoft. Microsoft silverlight. http://silverlight.net/, Last visited: 01/05/2009.

[40] Microsoft. Microsoft development network. http://msdn.microsoft.com/, Last visited: 01/05/2009.

[41] Suvda Myagmar, Adam J. Lee, and William Yurcik. Threat modeling as a basis for security requirements. In *In Symposium on Requirements Engineering for Information Security (SREIS)*, 2005.

[42] K. Maruyama N. Yoshioka, H. Washizaki. A survey on security patterns. *Progress in Informatics, No. 5 pp. 35-47*, 2008.

[43] Adam Nathan. *Windows Presentation Foundation Unleashed*. Sams, 1 edition, December 2006. ISBN 0672328917.

[44] Briony Oates. *Researching Information Systems and Computing*. Sage Publications Ltd, 1 edition, November 2005. ISBN 141290224X.

[45] F. Giunchiglia P. Bresciani, P. Giorgini and J. Mylopoulos. Tropos: An agent-oriented software development methodology. JAAMAS, vol.8, no.3, pp.203-236, 2004.

[46] Robert Richardson. Csi computer crime and security survey. http://www.gocsi.com/, 2008.

[47] Lillian Røstad. An extended misuse case notation: Including vulnerabilities and the insider threat. *Proc. 12th Working Conf. Requirements Eng.: Foundation for Software Quality (REFSQ)*, 2006.

[48] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, pages 328–338, 1987.

[49] Bruce Schneier. Attack trees. *Dr. Dobb's Journal*, 1999.

[50] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security patterns: Integrating security and systems engineering*. John Wiley and Sons, October 2005.

[51] Markus Schumacher and Utz Roedig. Security engineering with patterns. In *Lecture Notes in Computer Science, LNCS 2754*. Springer, 2001.

[52] *CLASP - Comprehensive Lightweight Application Security Process.* Secure Software, Inc., 2006.

[53] Helen Sharp, Yvonne Rogers, and Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction.* Wiley, 2 edition, March 2007. ISBN 0470018666.

[54] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.

[55] Guttorm Sindre and Andreas L. Opdahl. Misuse cases for identifying system dependability threats. *Journal of Information Privacy and Security, 4(2):3-22, 2:3-22*, 2008.

[56] Inger Anne Tøndel, Martin Gilje Jaatun, and Per Håkon Meland. Security requirements for the rest of us: A survey. *IEEE Software*, 25 (1):20–27, 2008.

[57] R. S. Wahono and J. Cheng. Extensible requirements patterns of web application for efficient web application development. In *CW '02: Proceedings of the First International Symposium on Cyber Worlds (CW'02)*, page 0412, 2002.

[58] Stephen Withall. *Software Requirement Patterns.* Microsoft Press, June 2007. ISBN 0735623988.

Appendices

Appendix A

Terminology

Asset something of value that should be secured.

Threat is a potential breach of security and potential danger posed to the system [33].

Vulnerability is a defect or weakness in the systems security procedures, design, implementation, or internal controls that could be used to violate the system. Bugs and flaws collectively form the basis of most software vulnerabilities. [33] [11] .

Attack An attack is the act of carrying out an exploit [11].

Weakness An underlying condition existing in a software system that has the potential for negatively impacting the security of the system [11].

Countermeasure an action taken to counteract a danger or threat.

Pattern is a solution to a reoccurring problem encapsulated in a structured format.

Requirement is a description of something the system must fulfill.

Requirement Pattern is a description of a reoccurring behavior of the system that is encapsulated into a reusable format

Security Pattern is a solution to a reoccurring problem in the security domain that is encapsulated into a reusable format indended to show software developers how to design and implement solutions to common security problems [11].

Design Pattern is a solution to a reoccurring design problem encapsulated

in a reusable format [11].

Exploit is a action to the system executed against a vulerability, deading to security compromise [33].

Encryption is a transformation of data to a format so that other cannot conceal the original meaning of the data.

Access control is an assurance so that only the users who are entitled to the resources get access.

Authentication is the confirmation of the users identity.

Authorization is the approval or permission to perform something.

Availability is the accessibility of the system.

Confidentiality is the insurance that information is only disclosed by the those who has authorization to do so.

Integrity is the assurance that information has not been changed.

Appendix B

Acronyms

LGPL Lesser General Public License

RMF Risk Management Framework

CLASP Comprehensive Lightweight Application Security Process

SQUARE Security Quality Requirements Engineering

TCSDL Trustworthy Computing Security Development Lifecycle

CC Common Criteria

UML Unified Modeling Language

GoF Gang of Four

AOP Agent Oriented Programing

OOP Object Oriented Programing

RBAC Role Based Access Control

SDL Software Development Lifecycle

SDP Security Design Pattern

SRP Security Requirement Pattern

WPF Windows Presentation Foundation

UI User Interface

TAM Technology Acceptance Model

NTNU Norwegian University of Science and Technology

GPL GNU General Public License

OWASP The Open Web Application Security Project

SDLC Software Development Life Cycle

GUI Graphical User Interface

Appendix C

User Tests

C.1 Description of Case 1

Case 1 – Online store for use with mobile equipment

(Translated to English)

We are going to develop an online store with digital content like music, videos, movies, ring tones, software and other products for mobile equipment like e.g. 3G mobile phones. The online store should be accessible by using a mobile and have they same functionality as any normal online store.

Functional requirements

1. A customer can create a profile.
2. Payment information can be saved for later use.
3. The store should have a shopping cart.
4. A customer should search or look for products.
5. A product can be added to the shopping cart.
6. A customer must have the possibility to pay for the content in the basket cart.
7. There should be a list with content the customer owns
8. Paid content should be possible to download to the mobile.

Assets (Something of value)

Customer
Product
Profile
Payment information
Shopping cart
Paid items
Mobile equipment

Assignment

1. Find security requirements to this case.
2. Find security design patterns that you think suits the case and security requirements you found.

Hint for the assignment

- What information should be saved about the customer?
- Who should have access to the customer information?
- How should the customer be authenticated?
- Who has access to what kind of information?

C.2 Description of Case 2

Case 2 – Patient Care Journal

(Translated to English)

We are going to develop a new system for patient care journals. The information in the journals is highly sensitive, and the information should only be accessible from the hospital internal network.

Functional requirements

1. A patient care journal should automatically exist for Norwegian citizens.
2. It should be possible to create a patient care journal for foreigners.
3. It should be possible to add new information about the patient into the journal.
4. Information with connection to other information should be branched.
5. People with the right access should have permission to change the patient care journal
6. Those with access should have to possibility to read the patient care journal.

NB. A patient care journal has in practice many laws and regulations to deal with. In this case, you do not need to take this under consideration. Make assumptions where you think it fits!

Assets (Something of value)

Patient journal
Doctor
Nurse
Information

Assignment

1. Find security requirements to this case.
2. Find security design patterns that you think suits the case and security requirements you found.

Hint for the assignment

- Who should have access to the patient journal?
- What level of access should exist?
- How should the access be divided?
- How should the authentication work?
- What kind of information should be saved?

C.3 Security Aspects for Case 1

Case 1 - Mobile netshop

1. Identification and Authentication
   (a) What information is available for non-authenticated users
   (b) What information has higher authorization privileges(e.g. log)
   (c) Customer identification
   (d) Customer authentication (at least before they either pay or download content)
   (e) Authentication strength (e.g. password length and variability)
   (f) Authentication failure mechanisms
   (g) Authentication of the server
   (h) Session security(limitation of concurrent sessions, session lifetime, system access history etc.)
2. Information disclosure, Privacy and Integrity
   (a) Communication between client and server
   (b) Customer information(Integrity and confidentiality)
   (c) Validity of the information presented to the customer.(Integrity of database)
   (d) Mobility of the client(Not save any information locally)
   (e) Card information(Expiry date and CVC-code should not be saved)
   (f) Secure payment
   (g) Securing the downloadable content
3. Immunity
   (a) Protecting the system against itself(e.g. from infection by an unauthorized program, virus scan on content, etc.)
   (b) Client should not be trusted
   (c) Intrusion detection handling(e.g. block users that triggers the detection handling)
   (d) Notification (e.g. notify client or administrators about events, e.g. content has been downloaded)
   (e) Failure handling

   (f) Quota on downloaded content

4. Review and Non-repudiation

   (a) Logging of usage and system notifications
   (b) Availability of the log
   (c) Securing the log(tamper-proof, read only, encrypted etc)
   (d) Not possible to deny an activity(e.g. a purchase) from taking place

C.4 Security Aspects for Case 2

Case 2 - Patient journal

1. Identification and Authentication

   (a) Authentication
   (b) Identification
   (c) Authentication strength (e.g. password length and variability)
   (d) Authentication failure mechanisms
   (e) Authentication of the server
   (f) Session security(limitation of concurrent sessions, session lifetime, system access history etc.)
   (g) Physical identification

2. Information disclosure, Privacy and Integrity

   (a) Communication between client and server
   (b) Secure patient information from disclosure to unauthorized users
   (c) Mechanisms to secure the exported patient data from disclosure
   (d) Patient information integrity
   (e) Mobility and exposure of the client(Not save any information locally)
   (f) Communication between servers

3. Immunity

   (a) Protecting the system against itself(e.g. from infection by an unauthorized program, virus scan on content, etc.)

(b) Clients should not be trusted

(c) Intrusion detection handling(e.g. block users that triggers the detection handling)

(d) Notification (e.g. notify client or administrators about events, e.g. content has been added)

(e) Failure handling

4. Review and Non-repudiation

(a) Logging of usage and system notifications

(b) Availability of the log

(c) Securing the log(tamper-proof, read only, encrypted etc)

(d) Not possible to deny an activity(e.g. reading a patient journal) from taking place

Appendix D

Questionnaires

D.1 Background Information

Questionnaire – background information

(Translated to English)

How many years have you studied higher education?

Number of years: _____

Have you taken any of the following subjects?

- ☐ TDT4237 - Programvaresikkerhet
- ☐ TDT60 - Informasjonsikkerhet i datasystemer
- ☐ TTM4134 - Informasjonsikkerhet

Have you taken any other security related subjects?

Subjects: _____

Approximately how many months have you worked within the IT-business (summer job, etc)?

Number of labour months: _____

Have you any experience with IT-security?

- ☐ Never heard of it
- ☐ Heard of it, but never done any practical work
- ☐ Done some work with it in project etc in school
- ☐ Done some work with it both in school and work
- ☐ Working with it all the time

What is you knowledge on Design Patterns in software development?

- ☐ Never heard of it
- ☐ Heard of it, but never used it
- ☐ Know what it is, and have done some work with it in project etc.
- ☐ Know very well what it is, and have used it a lot
- ☐ Using it all the time

What is your experience with the requirement phase?

- ☐ Never heard of it
- ☐ Heard of it, but never done any practical work in this phase
- ☐ Done some work with it in project etc in school
- ☐ Done some work with it both in school and work
- ☐ Working with it daily

What is your experience with security requirements?

- ☐ Never heard of it
- ☐ Heard of it, but never done any practical work in this phase
- ☐ Done some work with it in project etc in school
- ☐ Done some work with it both in school and work
- ☐ Working with it daily

What is your experience with design of a system?

- ☐ Never heard of it
- ☐ Heard of it, but never done any practical work in this phase
- ☐ Done some work with it in project etc in school
- ☐ Done some work with it both in school and work
- ☐ Working with it daily

What is your experience with design of a system with security in mind?

- ☐ Never heard of it
- ☐ Heard of it, but never done any practical work in this phase
- ☐ Done some work with it in project etc in school
- ☐ Done some work with it both in school and work
- ☐ Working with it daily

D.2 Experience

NTNU
Det skapende universitet

Questionnaire – experience

(Translated to English)

What do you think about the quality of the security requirements you found (measurable, formulated, verifiable, etc)?

- ☐ Not happy at all
- ☐ Slightly happy
- ☐ It was OK
- ☐ Very happy
- ☐ Could not be better

What do you think about the quality of the security design patterns that you found? (Fit with the setting, will make the system more secure, etc)?

- ☐ Not happy at all
- ☐ Slightly happy
- ☐ It was OK
- ☐ Very happy
- ☐ Can not be better

How confident are you that you have done a good job finding security requirements that fit to this case?

- ☐ Not confident at all
- ☐ Slightly confident
- ☐ They are OK
- ☐ Very confident
- ☐ Could not be better

How confident are you that you have done a good job finding security design patterns that fit to this case?

- ☐ Not confident at all
- ☐ Slightly confident
- ☐ They are OK
- ☐ Very confident
- ☐ Could not be better

Do you think it went quickly to find security requirements?

- ☐ Not happy at all
- ☐ Slightly happy
- ☐ It was OK
- ☐ Very happy
- ☐ Could not be better

Do you think it went quickly to find security design patterns?

- ☐ Not happy at all
- ☐ Slightly happy
- ☐ It was OK
- ☐ Very happy
- ☐ Could not be better

Please fill out the following questions if you did the case with a tool.

Was the tool easy to learn?

- ☐ No, not at all
- ☐ Slightly difficult
- ☐ It was OK
- ☐ It was easy
- ☐ No problem at all

Was the tool easy to understand?

- ☐ No, not at all
- ☐ Slightly difficult
- ☐ It was OK
- ☐ It was easy
- ☐ No problem at all

Was the tool easy to use?

- ☐ No, not at all
- ☐ Slightly difficult
- ☐ It was OK
- ☐ It was easy
- ☐ No problem at all

If you in the future were going to find security requirements, would you consider using this tool?

- ☐ No, the solution is not good enough
- ☐ Maybe
- ☐ Yes I would

If you get the opportunity, would to recommend someone about this tool?

- ☐ No, the solution is not good enough
- ☐ Maybe
- ☐ Yes I would

What is your general impression of the tool?

Appendix E

Application Data

This is the data that the application presented to the users. The list contains information about what data was connect; Which groups the security requirements was ordered by and what Security Design Pattern was suggested for each group. Both Groups and security requirements are taken from the Common Criteria, while the Security Design Patterns are mostly from the Security Pattern Repository by Darrell M. Kienzle et al.

E.1 Communication

This group provides assurance of the identity of a party participating in a data exchange.

Non-repudiation of origin (FCO_NRO)

Non-repudiation of origin ensures that the originator of information cannot successfully deny having sent the information. This requires that the system provide a method to ensure that a subject that receives information during a data exchange is provided with evidence of the origin of the information. This evidence can then be verified by either this subject or other subjects.

SR:7738 The system shall be able to generate evidence of origin for transmitted [assignment: list of information types] at the request of the [selection: originator, recipient, [assignment: list of third parties]].

SR:7739 The system shall be able to relate the [assignment: list of attributes] of the originator of the information, and the [assignment: list of information fields] of the information to which the evidence applies.

SR:7740 The system shall provide a capability to verify the evidence of origin of information to [selection: originator, recipient, [assignment: list of third parties]] given [assignment: limitations on the evidence of origin].

Suggested Security Design Patterns

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

Non-repudiation of receipt (FCO_NRR)

Non-repudiation of receipt ensures that the recipient of information cannot successfully deny receiving the information. This requires that the system provide a method to ensure that a subject that transmits information during a data exchange is provided with evidence of receipt of the information. This evidence can then be verified by either this subject or other subjects.

SR:7742 The system shall be able to generate evidence of receipt for received [assignment: list of information types] at the request of the [selection: originator, recipient, [assignment: list of third parties]].

SR:7743 The system shall be able to relate the [assignment: list of attributes] of the recipient of the information, and the [assignment: list of information fields] of the information to which the evidence applies.

SR:7744 The system shall provide a capability to verify the evidence of receipt of information to [selection: originator, recipient, [assignment: list of third parties]] given [assignment: limitations on the evidence of receipt].

Suggested Security Design Patterns

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

E.2 Graphic Support

This group has elements about cryptographic functions, the implementation of which could be in hardware, firmware and/or software.

Cryptographic key management (FCS_CKM)

Cryptographic keys must be managed throughout their life cycle. This family is intended to support that lifecycle and consequently defines requirements for the following activities: cryptographic key generation, cryptographic key distribution, cryptographic key access and cryptographic key destruction. This family should be included whenever there are functional requirements for the management of cryptographic keys.

SR:7746 The system shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm [assignment: cryptographic key generation algorithm] and specified cryptographic key sizes [assignment: cryptographic key sizes] that meet the following: [assignment: list of standards].

SR:7747 The system shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method [assignment: cryptographic key distribution method] that meets the following: [assignment: list of standards].

SR:7748 The system shall perform [assignment: type of cryptographic key access] in accordance with a specified cryptographic key access method [assignment: cryptographic key access method] that meets the following: [assignment: list of standards].

SR:7749 The system shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method [assignment: cryptographic key destruction method] that meets the following: [assignment: list of standards].

Suggested Security Design Patterns

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

Cryptographic operation (FCS_COP)

In order for a cryptographic operation to function correctly, the operation must be performed in accordance with a specified algorithm and with a cryptographic key of a specified size. This family should be included whenever there are requirements for cryptographic operations to be performed.

SR:7750 The system shall perform [assignment: list of cryptographic operations] in accordance with a specified cryptographic algorithm [assignment: cryptographic algorithm] and cryptographic key sizes [assignment: cryptographic key sizes] that meet the following: [assignment: list of standards].

Suggested Security Design Patterns

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

E.3 User Data Protection

This group contains requirements related to protecting user data. This includes import, export, and storage as well as security attributes directly related to user data.

Access control policy (FDP_ACC)

Identifies access control policies.

SR:7751 The system shall enforce the [assignment: access control policy] on [assignment: list of subjects, objects, and operations among subjects and objects covered by the policy].

SR:7753 The system shall ensure that all operations between any subject and object controlled by the system are covered by an access control policy.

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The

Encrypted Storage pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Access control functions (FDP_ACF)

This family describes the rules for the specific functions that can implement an access control policy named in Access control policy (FDP_ACC). Access control policy (FDP_ACC) specifies the scope of control of the policy.

SR:7754 The system shall enforce the [assignment: access control policy] to objects based on the following: [assignment: list of subjects and objects controlled under the indicated policy, and for each, the policy-relevant security attributes, or named groups of policy-relevant security attributes].

SR:7755 The system shall enforce the following rules to determine if an

operation among controlled subjects and controlled objects is allowed: [assignment: rules governing access among controlled subjects and controlled objects using controlled operations on controlled objects].

SR:7756 The system shall explicitly authorise access of subjects to objects based on the following additional rules: [assignment: rules, based on security attributes, that explicitly authorise access of subjects to objects].

SR:7757 The system shall explicitly deny access of subjects to objects based on the [assignment: rules, based on security attributes, that explicitly deny access of subjects to objects].

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The

Encrypted Storage pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The

Password Propagation pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Data authentication (FDP_DAU)

Data authentication permits an entity to accept responsibility for the au-

thenticity of information (e.g., by digitally signing it). This family provides a method of providing a guarantee of the validity of a specific unit of data that can be subsequently used to verify that the information content has not been forged or fraudulently modified. In contrast to FAU: Security audit, this family is intended to be applied to "static" data rather than data that is being transferred.

SR:7758 The system shall provide a capability to generate evidence that can be used as a guarantee of the validity of [assignment: list of objects or information types].

SR:7759 The system shall provide [assignment: list of subjects] with the ability to verify evidence of the validity of the indicated information.

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The

Encrypted Storage pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The

Password Propagation pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Export from the system (FDP_ETC)

Define functions for system-mediated exporting of user data from the TOE such that its security attributes and protection either can be explicitly preserved or can be ignored once it has been exported. It is concerned with limitations on export and with the association of security attributes with the exported user data.

SR:7760 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] when exporting user data, controlled under the SFP(s), outside of the TOE.

SR:7761 The system shall export the user data without the user data's associated security attributes

SR:7762 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] when exporting user data, controlled under the SFP(s), outside of the TOE.

SR:7763 The system shall export the user data with the user data's associated security attributes.

SR:7764 The system shall ensure that the security attributes, when exported outside the TOE, are unambiguously associated with the exported user data.

SR:7765 The system shall enforce the following rules when user data is exported from the TOE: [assignment: additional exportation control rules].

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The

Encrypted Storage pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might

have complete access to every user's data. The

Password Propagation pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Information flow control policy (FDP_IFC)

The system mechanism controls the flow of information in accordance with the information flow control SFP. Operations that would change the security attributes of information are not generally permitted as this would be in violation of an information flow control SFP. However, such operations may be permitted as exceptions to the information flow control SFP if explicitly specified.

SR:7766 The system shall enforce the [assignment: information flow control SFP] on [assignment: list of subjects, information, and operations that cause controlled information to flow to and from controlled subjects covered by the SFP].

SR:7768 The system shall ensure that all operations that cause any information in the TOE to flow to and from any subject in the TOE are covered by an information flow control SFP.

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The

Encrypted Storage pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The

Password Propagation pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Information flow control functions (FDP_IFF)

This family describes the rules for the specific functions that can implement the information flow control SFPs named in Information flow control policy (FDP_IFC), which also specifies the scope of control of the policy. It consists of two kinds of requirements: one addressing the common information flow function issues, and a second addressing illicit information flows (i.e. covert channels). This division arises because the issues concerning illicit information flows are, in some sense, orthogonal to the rest of an information flow control SFP. By their nature they circumvent the information flow control SFP resulting in a violation of the policy. As such, they require special functions to either limit or prevent their occurrence.

SR:7769 The system shall enforce the [assignment: information flow control SFP] based on the following types of subject and information security attributes: [assignment: list of subjects and information controlled under the indicated SFP, and for each, the security attributes].

SR:7770 The system shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold: [assignment: for each operation, the security attribute-based relationship that must hold between subject and information security attributes].

SR:7771 The system shall enforce the [assignment: additional information flow control SFP rules].

SR:7772 The system shall explicitly authorise an information flow based on the following rules: [assignment: rules, based on security attributes, that explicitly authorise information flows].

SR:7773 The system shall explicitly deny an information flow based on the following rules: [assignment: rules, based on security attributes, that explicitly deny information flows].

SR:7774 The system shall enforce the [assignment: information flow control SFP] to limit the capacity of [assignment: types of illicit information flows] to a [assignment: maximum capacity].

SR:7776 The system shall enforce the [assignment: information flow control SFP] to monitor [assignment: types of illicit information flows] when it exceeds the [assignment: maximum capacity].

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The *Encrypted Storage* pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Import from outside of the system (FDP_ITC)

Define the mechanisms for system-mediated importing of user data into the system such that it has appropriate security attributes and is appropriately protected. It is concerned with limitations on importation, determination of desired security attributes, and interpretation of security attributes associated with the user data.

SR:7777 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] when importing user data, controlled under the SFP, from outside of the TOE.

SR:7778 The system shall ignore any security attributes associated with the user data when imported from outside the TOE.

SR:7779 The system shall enforce the following rules when importing user data controlled under the SFP from outside the TOE: [assignment: additional importation control rules].

SR:7780 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] when importing user data, controlled under the SFP, from outside of the TOE.

SR:7781 The system shall use the security attributes associated with the imported user data.

SR:7782 The system shall ensure that the protocol used provides for the unambiguous association between the security attributes and the user data received.

SR:7783 The system shall ensure that interpretation of the security attributes of the imported user data is as intended by the source of the user data.

SR:7784 The system shall enforce the following rules when importing user data controlled under the SFP from outside the TOE: [assignment: additional importation control rules].

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The *Encrypted Storage* pattern ensures that even if it is stolen, the most

sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Internal system transfer (FDP_ITT)

This family provides requirements that address protection of user data when it is transferred between separated parts of a system across an internal channel. This may be contrasted with the Inter-TSF user data confidentiality transfer protection (FDP_UCT) and Inter-TSF user data integrity transfer protection (FDP_UIT) families, which provide protection for user data when it is transferred between distinct TSFs across an external channel, and Export from the TOE (FDP_ETC) and Import from outside of the TOE (FDP_ITC), which address TSF-mediated transfer of data to or from outside the TOE.

SR:7785 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] to prevent the [selection: disclosure, modification, loss of use] of user data when it is transmitted between physically-separated parts of the TOE.

SR:7786 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] to monitor user data transmitted between physically-separated parts of the TOE for the following errors: [assignment: integrity errors].

SR:7787 Upon detection of a data integrity error, the system shall [assignment: specify the action to be taken upon integrity error].

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The *Encrypted Storage* pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Residual information protection (FDP_RIP)

This family addresses the need to ensure that any data contained in a resource is not available when the resource is de-allocated from one object and reallocated to a different object. This family requires protection for any data contained in a resource that has been logically deleted or released, but may still be present within the system-controlled resource which in turn may be re-allocated to another object.

SR:7789 The system shall ensure that any previous information content of a resource is made unavailable upon the [selection: allocation of the resource to, deallocation of the resource from] the following objects: [assignment: list of objects].

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the

theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The *Encrypted Storage* pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Rollback (FDP_ROL)

The rollback operation involves undoing the last operation or a series of operations, bounded by some limit, such as a period of time, and return to a previous known state. Rollback provides the ability to undo the effects of an operation or series of operations to preserve the integrity of the user data.

SR:7791 The system shall enforce [assignment: access control SFP(s) and/or information flow control SFP(s)] to permit the rollback of the [assignment: list of operations] on the [assignment: information and/or list of objects].

SR:7792 The system shall permit operations to be rolled back within the [assignment: boundary limit to which rollback may be performed].

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized ex-

amples of hackers stealing databases containing sensitive user information. The *Encrypted Storage* pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Stored data integrity (FDP_SDI)

This family provides requirements that address protection of user data while it is stored within containers controlled by the system. Integrity errors may affect user data stored in memory, or in a storage device. This family differs from Internal system transfer (FDP_ITT) which protects the user data from integrity errors while being transferred within the system.

SR:7794 The system shall monitor user data stored in containers controlled by the system for [assignment: integrity errors] on all objects, based on the following attributes: [assignment: user data attributes].

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The *Encrypted Storage* pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Inter-system user data confidentiality transfer protection (FDP_UCT)

Define the requirements for ensuring the confidentiality of user data when it is transferred using an external channel between the system and another trusted IT product.

SR:7795 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] to be able to [selection: transmit, receive] user data in a manner protected from unauthorised disclosure.

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The *Encrypted Storage* pattern ensures that even if it is stolen, the most

sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Inter-system user data integrity transfer protection (FDP_UIT)

Define the requirements for providing integrity for user data in transit between the system and another trusted IT product and recovering from detectable errors. At a minimum, this family monitors the integrity of user data for modifications. Furthermore, this family supports different ways of correcting detected integrity errors.

SR:7796 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] to be able to [selection: transmit, receive] user data in a manner protected from [selection: modification, deletion, insertion, replay] errors.

SR:7797 The system shall be able to determine on receipt of user data, whether [selection: modification, deletion, insertion, replay] has occurred.

SR:7798 The system shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] to be able to recover from [assignment: list of recoverable errors] with the help of the source trusted IT product.

Suggested Security Design Patterns

Encrypted Storage

The *Encrypted Storage* pattern provides a second line of defense against the

theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The *Encrypted Storage* pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

E.4 Identification and Authentication

This group deal with determining and verifying the identity of users, determining their authority to interact with the system, and with the correct association of security attributes for each authorised user.

Authentication failure handling

This family contains requirements for defining values for some number of unsuccessful authentication attempts and system actions in cases of authentication attempt failures. Parameters include, but are not limited to, the number of failed authentication attempts and time thresholds.

SR:7800 The system shall detect when [selection: [assignment: positive integer number], an administrator configurable positive integer within[assignment: range of acceptable values]] unsuccessful authentication attempts occur related to [assignment: list of authentication events].

SR:7813 When the defined number of unsuccessful authentication attempts has been [selection: met, surpassed], the system shall [assignment: list of

actions].

Suggested Security Design Patterns

Account Lockout

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. However, password-guessing attacks have proven to be very successful at discovering poorly chosen, weak passwords. Worse, the Web environment lends itself to high- speed, anonymous guessing attacks. Account lockout protects customer accounts from automated password-guessing attacks, by implementing a limit on incorrect password attempts before further attempts are disallowed.

Network Address Blacklist

A network address blacklist is used to keep track of network addresses (IP addresses) that are the sources of hacking attempts and other mischief. Any requests originating from an address on the blacklist are simply ignored. Ideally, breaking attempts should be investigated and prosecuted, but there are simply too many such events to address them all. The *Network Address Blacklist* pattern represents a pragmatic alternative.

User attribute definition (FIA_ATD)

All authorised users may have a set of security attributes, other than the user's identity, that is used to enforce the security requirements. Define the requirements for associating user security attributes with users as needed to support the system in making security decisions.

SR:7814 The system shall maintain the following list of security attributes belonging to individual users: [assignment: list of security attributes].

Suggested Security Design Patterns

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The *Client Data Storage* pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.

Specification of secrets (FIA_SOS)

Define requirements for mechanisms that enforce defined quality metrics on provided secrets and generate secrets to satisfy the defined metric.

SR:7815 The system shall provide a mechanism to verify that secrets meet [assignment: a defined quality metric].

SR:7816 The system shall provide a mechanism to generate secrets that meet [assignment: a defined quality metric].

SR:7817 The system shall be able to enforce the use of system generated secrets for [assignment: list of system functions].

Suggested Security Design Patterns

Password Authentication

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. Any site that needs to reliably identify its users will almost certainly use passwords. The *Password Authentication* pattern protects against weak passwords, automated password- guessing attacks, and mishandling of passwords.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

User authentication (FIA_UAU)

Define the types of user authentication mechanisms supported by the system. This family also defines the required attributes on which the user authentication mechanisms must be based.

SR:7818 The system shall allow [assignment: list of system mediated actions] on behalf of the user to be performed before the user is authenticated.

SR:7819 The system shall require each user to be successfully authenticated before allowing any other system-mediated actions on behalf of that user.

SR:7820 The system shall [selection: detect, prevent] use of authentication data that has been forged by any user of the system.

SR:7821 The system shall [selection: detect, prevent] use of authentication data that has been copied from any other user of the system.

SR:7822 The system shall prevent reuse of authentication data related to [assignment: identified authentication mechanism(s)].

SR:7823 The system shall provide [assignment: list of multiple authentication mechanisms] to support user authentication.

SR:7824 The system shall authenticate any user's claimed identity according to the [assignment: rules describing how the multiple authentication mechanisms provide authentication].

SR:7825 The system shall re-authenticate the user under the conditions [assignment: list of conditions under which re-authentication is required].

SR:7826 The system shall provide only [assignment: list of feedback] to the user while the authentication is in progress.

Suggested Security Design Patterns

Account Lockout

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. However, password-guessing attacks have proven to be very successful at discovering poorly chosen, weak passwords. Worse, the Web environment lends itself to high- speed, anonymous guessing attacks. Account lockout protects customer accounts from automated password-guessing attacks, by implementing a limit on incorrect password attempts before further attempts are disallowed.

Authenticated Session

An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.

Network Address Blacklist

A network address blacklist is used to keep track of network addresses (IP addresses) that are the sources of hacking attempts and other mischief. Any requests originating from an address on the blacklist are simply ignored. Ideally, breaking attempts should be investigated and prosecuted, but there are simply too many such events to address them all. The *Network Address*

Blacklist pattern represents a pragmatic alternative.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Password Authentication

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. Any site that needs to reliably identify its users will almost certainly use passwords. The *Password Authentication* pattern protects against weak passwords, automated password- guessing attacks, and mishandling of passwords.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

User identification (FIA_UID)

Define the conditions under which users shall be required to identify themselves before performing any other actions that are to be mediated by the system and which require user identification.

SR:7827 The system shall allow [assignment: list of system-mediated actions] on behalf of the user to be performed before the user is identified.

SR:7828 The system shall require each user to be successfully identified before allowing any other system-mediated actions on behalf of that user.

Suggested Security Design Patterns

Account Lockout

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. However, password-guessing attacks have proven to be very successful at discovering poorly chosen, weak pass-

words. Worse, the Web environment lends itself to high- speed, anonymous guessing attacks. Account lockout protects customer accounts from automated password-guessing attacks, by implementing a limit on incorrect password attempts before further attempts are disallowed.

Authenticated Session

An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.

Network Address Blacklist

A network address blacklist is used to keep track of network addresses (IP addresses) that are the sources of hacking attempts and other mischief. Any requests originating from an address on the blacklist are simply ignored. Ideally, breaking attempts should be investigated and prosecuted, but there are simply too many such events to address them all. The *Network Address Blacklist* pattern represents a pragmatic alternative.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Password Authentication

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. Any site that needs to reliably identify its users will almost certainly use passwords. The *Password Authentication* pattern protects against weak passwords, automated password- guessing attacks, and mishandling of passwords.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

User-subject binding (FIA_USB)

An authenticated user, in order to use the system, typically activates a subject. The user's security attributes are associated (totally or partially) with this subject. Define requirements to create and maintain the association of the user's security attributes to a subject acting on the user's behalf.

SR:7829 The system shall associate the following user security attributes with subjects acting on the behalf of that user: [assignment: list of user security attributes].

SR:7830 The system shall enforce the following rules on the initial association of user security attributes with subjects acting on the behalf of users: [assignment: rules for the initial association of attributes].

SR:7831 The system shall enforce the following rules governing changes to the user security attributes associated with subjects acting on the behalf of users: [assignment: rules for the changing of attributes].

Suggested Security Design Patterns

Account Lockout

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. However, password-guessing attacks have proven to be very successful at discovering poorly chosen, weak passwords. Worse, the Web environment lends itself to high- speed, anonymous guessing attacks. Account lockout protects customer accounts from automated password-guessing attacks, by implementing a limit on incorrect password attempts before further attempts are disallowed.

Authenticated Session

An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.

Network Address Blacklist

A network address blacklist is used to keep track of network addresses (IP addresses) that are the sources of hacking attempts and other mischief. Any requests originating from an address on the blacklist are simply ignored. Ideally, breaking attempts should be investigated and prosecuted, but there

are simply too many such events to address them all. The *Network Address Blacklist* pattern represents a pragmatic alternative.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Password Authentication

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. Any site that needs to reliably identify its users will almost certainly use passwords. The *Password Authentication* pattern protects against weak passwords, automated password- guessing attacks, and mishandling of passwords.

Password Propagation

Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The *Password Propagation* pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.

E.5 Security Management

Specify the management of several aspects of the system: security attributes, system data and functions. The different management roles and their interaction, such as separation of capability, can be specified.

Management of functions in system (FMT_MOF)

This family allows authorised users control over the management of functions in the system. Examples of functions in the system include the audit functions and the multiple authentication functions.

SR:7832 The system shall restrict the ability to [selection: determine the behaviour of, disable, enable, modify the behaviour of] the functions [assignment: list of functions] to [assignment: the authorised identified roles].

Suggested Security Design Patterns

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The *Client Data Storage* pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.

Client Input Filters

Client input filters protect the application from data tampering performed on untrusted clients. Developers tend to assume that the components executing on the client system will behave as they were originally programmed. This pattern protects against subverted clients that might cause the application to behave in an unexpected and insecure fashion.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

Management of security attributes (FMT_MSA)

This family allows authorised users control over the management of security attributes. This management might include capabilities for viewing and modifying of security attributes.

SR:7833 The system shall enforce the [assignment: access control SFP(s), information flow control SFP(s)] to restrict the ability to [selection: change_default, query, modify, delete, [assignment: other operations]] the security attributes [assignment: list of security attributes] to [assignment: the authorised identified roles].

SR:7834 The system shall ensure that only secure values are accepted for [assignment: list of security attributes].

SR:7835 The system shall enforce the [assignment: access control SFP, information flow control SFP] to provide [selection, choose one of: restrictive, permissive, [assignment: other property]] default values for security attributes that are used to enforce the SFP.

SR:7836 The system shall allow the [assignment: the authorised identified roles] to specify alternative initial values to override the default values when an object or information is created.

SR:7837 The system shall use the following rules to set the value of security attributes: [assignment: rules for setting the values of security attributes]

Suggested Security Design Patterns

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The *Client Data Storage* pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.

Client Input Filters

Client input filters protect the application from data tampering performed on untrusted clients. Developers tend to assume that the components executing on the client system will behave as they were originally programmed. This pattern protects against subverted clients that might cause the application to behave in an unexpected and insecure fashion.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

Management of system data (FMT_MTD)

This family allows authorised users (roles) control over the management of system data. Examples of system data include audit information, clock and other system configuration parameters.

SR:7838 The system shall restrict the ability to [selection: change_default, query, modify, delete, clear, [assignment: other operations]] the [assignment: list of system data] to [assignment: the authorised identified roles].

SR:7917 The system shall restrict the specification of the limits for [assignment: list of system data] to [assignment: the authorised identified roles].

SR:7918 The system shall take the following actions, if the system data are at, or exceed, the indicated limits: [assignment: actions to be taken].

SR:7919 The system shall ensure that only secure values are accepted for [assignment: list of system data].

Suggested Security Design Patterns

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The *Client Data Storage* pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.

Client Input Filters

Client input filters protect the application from data tampering performed on untrusted clients. Developers tend to assume that the components executing on the client system will behave as they were originally programmed. This pattern protects against subverted clients that might cause the application to behave in an unexpected and insecure fashion.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation

for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

Revocation (FMT_REV)

This family addresses revocation of security attributes for a variety of entities within a system.

SR:7842 The system shall restrict the ability to revoke [assignment: list of security attributes] associated with the [selection: users, subjects, objects, [assignment: other additional resources]] under the control of the system to [assignment: the authorised identified roles].

SR:7843 The system shall enforce the rules [assignment: specification of revocation rules].

Suggested Security Design Patterns

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The *Client Data Storage* pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.

Client Input Filters

Client input filters protect the application from data tampering performed on untrusted clients. Developers tend to assume that the components executing on the client system will behave as they were originally programmed. This pattern protects against subverted clients that might cause the application to behave in an unexpected and insecure fashion.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

Security attribute expiration (FMT_SAE)

Addresses the capability to enforce time limits for the validity of security attributes.

SR:7844 The system shall restrict the capability to specify an expiration time for [assignment: list of security attributes for which expiration is to be supported] to [assignment: the authorised identified roles].

SR:7845 For each of these security attributes, the system shall be able to [assignment: list of actions to be taken for each security attribute] after the expiration time for the indicated security attribute has passed.

Suggested Security Design Patterns

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this

data. The *Client Data Storage* pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.

Client Input Filters

Client input filters protect the application from data tampering performed on untrusted clients. Developers tend to assume that the components executing on the client system will behave as they were originally programmed. This pattern protects against subverted clients that might cause the application to behave in an unexpected and insecure fashion.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

Specification of Management Functions (FMT_SMF)

This family allows the specification of the management functions to be provided by the TOE. Management functions provide TSFI that allow administrators to define the parameters that control the operation of security-related aspects of the TOE, such as data protection attributes, TOE protection attributes, audit attributes, and identification and authentication attributes. Management functions also include those functions performed by an operator to ensure continued operation of the TOE, such as backup and recovery. This family works in conjunction with the other components in the FMT: Security management class: the component in this family calls out the management functions, and other families in FMT: Security management restrict the ability to use these management functions.

SR:7846 The system shall be capable of performing the following management functions: [assignment: list of management functions to be provided by the system].

Suggested Security Design Patterns

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The *Client Data Storage* pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.

Client Input Filters

Client input filters protect the application from data tampering performed on untrusted clients. Developers tend to assume that the components executing on the client system will behave as they were originally programmed. This pattern protects against subverted clients that might cause the application to behave in an unexpected and insecure fashion.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular

technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

Security management roles (FMT_SMR)

This family is intended to control the assignment of different roles to users. The capabilities of these roles with respect to security management are described in the other families in this class.

SR:7847 The system shall maintain the roles [assignment: the authorised identified roles].

SR:7848 The system shall be able to associate users with roles.

SR:7850 The system shall require an explicit request to assume the following roles: [assignment: the roles].

Suggested Security Design Patterns

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The *Client Data Storage* pattern uses encryption to allow sensitive or

otherwise security-critical data to be securely stored on the client.

Client Input Filters

Client input filters protect the application from data tampering performed on untrusted clients. Developers tend to assume that the components executing on the client system will behave as they were originally programmed. This pattern protects against subverted clients that might cause the application to behave in an unexpected and insecure fashion.

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

E.6 Privacy

This group contains privacy elements. These elements provide a user protection against discovery and misuse of identity by other users.

Anonymity (FPR_ANO)

This family ensures that a user may use a resource or service without disclosing the user's identity. The requirements for Anonymity provide protection of the user identity. Anonymity is not intended to protect the subject identity.

SR:7851 The system shall ensure that [assignment: set of users and/or subjects] are unable to determine the real user name bound to [assignment: list of subjects and/or operations and/or objects].

Suggested Security Design Patterns

Account Lockout

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. However, password-guessing attacks have proven to be very successful at discovering poorly chosen, weak passwords. Worse, the Web environment lends itself to high- speed, anonymous guessing attacks. Account lockout protects customer accounts from automated password-guessing attacks, by implementing a limit on incorrect password attempts before further attempts are disallowed.

Client Data Storage

It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The *Client Data Storage* pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Trusted Proxy

A trusted proxy acts on behalf of the user to perform specific actions requiring more privileges than the user possesses. It provides a safe interface by constraining access to the protected resources, limiting the operations that can be performed, or limiting the user's view to a subset of the data.

Pseudonymity (FPR_PSE)

This family ensures that a user may use a resource or service without disclosing its user identity, but can still be accountable for that use.

SR:7852 The system shall ensure that [assignment: set of users and/or subjects] are unable to determine the real user name bound to [assignment: list of subjects and/or operations and/or objects].

SR:7853 The system shall be able to provide [assignment: number of aliases] aliases of the real user name to [assignment: list of subjects].

SR:7854 The system shall [selection, choose one of: determine an alias for a user, accept the alias from the user] and verify that it conforms to the [assignment: alias metric].

Suggested Security Design Patterns

Trusted Proxy

A trusted proxy acts on behalf of the user to perform specific actions requiring more privileges than the user possesses. It provides a safe interface by constraining access to the protected resources, limiting the operations that can be performed, or limiting the user's view to a subset of the data.

Unlinkability (FPR_UNL)

This family ensures that a user may make multiple uses of resources or services without others being able to link these uses together.

SR:7855 The system shall ensure that [assignment: set of users and/or subjects] are unable to determine whether [assignment: list of operations][selection: were caused by the same user, are related as follows[assignment: list of relations]].

Suggested Security Design Patterns

Directed Session

The *Directed Session* pattern ensures that users will not be able to skip around within a series of Web pages. The system will not expose multiple URLs but instead will maintain the current page on the server. By guaranteeing the order in which pages are visited, the developer can have confidence that users will not undermine or circumvent security checkpoints.

Hidden Implementation

The *Hidden Implementation* pattern limits an attacker's ability to discern the internal workings of an application–information that might later be used to compromise the application. It does not replace other defenses, but it supplements them by making an attacker's job more difficult.

Unobservability (FPR_UNO)

This family ensures that a user may use a resource or service without others, especially third parties, being able to observe that the resource or service is being used.

SR:7856 The system shall ensure that [assignment: list of users and/or subjects] are unable to observe the operation [assignment: list of operations] on [assignment: list of objects] by [assignment: list of protected users and/or subjects].

SR:7857 The system shall provide [assignment: list of services] to [assignment: list of subjects] without soliciting any reference to [assignment: privacy related information].

SR:7858 The system shall provide [assignment: set of authorised users] with the capability to observe the usage of [assignment: list of resources and/or services].

Suggested Security Design Patterns

Trusted Proxy

A trusted proxy acts on behalf of the user to perform specific actions requiring more privileges than the user possesses. It provides a safe interface by constraining access to the protected resources, limiting the operations that can be performed, or limiting the user's view to a subset of the data.

E.7 Protection of the Security Function

This group contains elements that relate to the integrity and management of the mechanisms that constitute the security functions and to the integrity of the data.

Fail secure (FPT_FLS)

The requirements of this family ensure that the system will always enforce its requirements in the event of identified categories of failures in the system.

SR:7859 The system shall preserve a secure state when the following types of failures occur: [assignment: list of types of failures in the system].

Suggested Security Design Patterns

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

Availability of exported system data (FPT_ITA)

Define the rules for the prevention of loss of availability of system data

moving between the system and another trusted IT product. This data could, for example, be system critical data such as passwords, keys, audit data, or system executable code.

SR:7860 The system shall ensure the availability of [assignment: list of types of system data] provided to another trusted IT product within [assignment: a defined availability metric] given the following conditions [assignment: conditions to ensure availability].

Suggested Security Design Patterns

Confidentiality of exported system data (FPT_ITC)

Define the rules for the protection from unauthorised disclosure of system data during transmission between the system and another trusted IT product. This data could, for example, be system critical data such as passwords, keys, audit data, or system executable code.

SR:7861 The system shall protect all system data transmitted from the system to another trusted IT product from unauthorised disclosure during transmission.

Suggested Security Design Patterns

Integrity of exported system data (FPT_ITI)

Define the rules for the protection, from unauthorised modification, of system data during transmission between the system and another trusted IT product. This data could, for example, be system critical data such as passwords, keys, audit data, or system executable code.

SR:7862 The system shall provide the capability to detect modification of all system data during transmission between the system and another trusted IT product within the following metric: [assignment: a defined modification metric].

SR:7863 The system shall provide the capability to verify the integrity of all system data transmitted between the system and another trusted IT product and perform [assignment: action to be taken] if modifications are detected.

SR:7864 Inter-system detection and correction of modification

Suggested Security Design Patterns

Authenticated Session

An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.

Internal system data transfer (FPT_ITT)

This family provides requirements that address protection of system data when it is transferred between separate parts of a system across an internal channel.

SR:7865 The system shall protect system data from [selection: disclosure, modification] when it is transmitted between separate parts of the system.

SR:7866 The system shall be able to detect [selection: modification of data, substitution of data, re-ordering of data, deletion of data, [assignment: other integrity errors]] for system data transmitted between separate parts of the system.

SR:7867 Upon detection of a data integrity error, the system shall take the following actions: [assignment: specify the action to be taken].

Suggested Security Design Patterns

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

System physical protection (FPT_PHP)

system physical protection components refer to restrictions on unauthorised physical access to the system, and to the deterrence of, and resistance to, unauthorised physical modification, or substitution of the system.

SR:7868 The system shall provide unambiguous detection of physical tampering that might compromise the system.

SR:7869 The system shall provide the capability to determine whether physical tampering with the system's devices or system's elements has occurred.

SR:7870 The system shall resist [assignment: physical tampering scenarios] to the [assignment: list of system devices/elements] by responding automatically such that the SFRs are always enforced.

Suggested Security Design Patterns

Trusted recovery (FPT_RCV)

The requirements of this family ensure that the system can determine that the system is started up without protection compromise and can recover without protection compromise after discontinuity of operations. This family is important because the start-up state of the system determines the protection of subsequent states.

SR:7871 After [assignment: list of failures/service discontinuities] the system shall enter a maintenance mode where the ability to return to a secure state is provided.

SR:7872 When automated recovery from [assignment: list of failures/service discontinuities] is not possible,

SR:7873 For [assignment: list of failures/service discontinuities], the system shall ensure the return of the TOE to a secure state using automated procedures.

SR:7874 The system shall ensure that [assignment: list of functions and failure scenarios] have the property that the function either completes successfully, or for the indicated failure scenarios, recovers to a consistent and secure state.

Suggested Security Design Patterns

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Replay detection (FPT_RPL)

This family addresses detection of replay for various types of entities (e.g. messages, service requests, service responses) and subsequent actions to correct. In the case where replay may be detected, this effectively prevents it.

SR:7875 The system shall detect replay for the following entities: [assignment: list of identified entities].

SR:7876 The system shall perform [assignment: list of specific actions] when replay is detected.

Suggested Security Design Patterns

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

State synchrony protocol (FPT_SSP)

Distributed systems may give rise to greater complexity than monolithic systems through the potential for differences in state between parts of the system, and through delays in communication. In most cases synchronisation of state between distributed functions involves an exchange protocol, not a simple action. When malice exists in the distributed environment of these protocols, more complex defensive protocols are required.

SR:7877 The system shall acknowledge, when requested by another part of the system, the receipt of an unmodified system data transmission.

Suggested Security Design Patterns

Time stamps (FPT_STM)

This family addresses requirements for a reliable time stamp function within a system.

SR:7878 The system shall be able to provide reliable time stamps.

Suggested Security Design Patterns

Inter-system data consistency (FPT_TDC)

In a distributed environment, a system may need to exchange system data (e.g. the security requirement attributes associated with data, audit information, identification information) with another trusted IT product, Define the requirements for sharing and consistent interpretation of these attributes between the system of the TOE and a different trusted IT product.

SR:7879 The system shall provide the capability to consistently interpret [assignment: list of system data types] when shared between the system and another trusted IT product.

SR:7880 The system shall use [assignment: list of interpretation rules to be applied by the system] when interpreting the system data from another trusted IT product.

Suggested Security Design Patterns

Testing of external entities (FPT_TEE)

Define requirements for the system to perform tests on one or more external entities.

SR:7881 The system shall run a suite of tests [selection: during initial start-up, periodically during normal operation, at the request of an authorised user, [assignment: other conditions]] to check the fulfillment of [assignment: list of properties of the external entities] .

SR:7882 If the test fails, the system shall [assignment: action(s)] .

Suggested Security Design Patterns

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Internal system security function data replication consistency (FPT_TRC)

The requirements of this family are needed to ensure the consistency of system data when such data is replicated internal to the system. Such data may become inconsistent if the internal channel between parts of the system becomes inoperative. If the system is internally structured as a network and parts of the TOE network connections are broken, this may occur when parts become disabled.

SR:7883 The system shall ensure that system data is consistent when replicated between parts of the system.

SR:7884 When parts of the system containing replicated system data are disconnected, the system shall ensure the consistency of the replicated system data upon reconnection before processing any requests for [assignment: list of functions dependent on system data replication consistency].

Suggested Security Design Patterns

Partitioned Application

The *Partitioned Application* pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.

System self test (FPT_TST)

The family defines the requirements for the self-testing of the system with respect to some expected correct operation. Examples are interfaces to enforcement functions, and sample arithmetical operations on critical parts of the system. These tests can be carried out at start-up, periodically, at the request of the authorised user, or when other conditions are met. The actions to be taken by the system as the result of self testing are defined in other families.

SR:7885 The system shall run a suite of self tests [selection: during initial start-up, periodically during normal operation, at the request of the authorised user, at the conditions[assignment: conditions under which self test should occur]] to demonstrate the correct operation of [selection: [assignment: parts of system], the system].

SR:7886 The system shall provide authorised users with the capability to verify the integrity of [selection: [assignment: parts of system], system data].

SR:7887 The system shall provide authorised users with the capability to

verify the integrity of stored system executable code.

Suggested Security Design Patterns

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

E.8 Resource Utilisation

This group provides elements that support the availability of required resources such as processing capability and/or storage capacity.

Fault tolerance (FRU_FLT)

The requirements of this family ensure that the system will maintain correct operation even in the event of failures.

SR:7888 The system shall ensure the operation of [assignment: list of system capabilities] when the following failures occur: [assignment: list of type of failures].

Suggested Security Design Patterns

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Server Sandbox

Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The *Server Sandbox* pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.

Priority of service (FRU_PRS)

The requirements of this family allow the system to control the use of resources under the control of the system by users and subjects such that high priority activities under the control of the system will always be accomplished without undue interference or delay caused by low priority activities.

SR:7890 The system shall assign a priority to each subject in the system.

SR:7891 The system shall ensure that each access to [assignment: controlled resources] shall be mediated on the basis of the subjects assigned priority.

Suggested Security Design Patterns

Resource allocation (FRU_RSA)

The requirements of this family allow the system to control the use of resources by users and subjects such that denial of service will not occur because of unauthorised monopolisation of resources.

SR:7892 The system shall enforce maximum quotas of the following resources: [assignment: controlled resources] that [selection: individual user, defined group of users, subjects] can use [selection: simultaneously, over a specified period of time].

Suggested Security Design Patterns

E.9 System Access

This group specifies elements for controlling the establishment of a user's session.

Limitation on scope of selectable attributes (FTA_LSA)

Define requirements to limit the scope of session security attributes that a user may select for a session.

SR:7893 The system shall restrict the scope of the session security attributes [assignment: session security attributes], based on [assignment: attributes].

Suggested Security Design Patterns

Authenticated Session

An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.

Directed Session

The *Directed Session* pattern ensures that users will not be able to skip around within a series of Web pages. The system will not expose multiple URLs but instead will maintain the current page on the server. By guaranteeing the order in which pages are visited, the developer can have confidence that users will not undermine or circumvent security checkpoints.

Limitation on multiple concurrent sessions (FTA_MCS)

Define requirements to place limits on the number of concurrent sessions that belong to the same user.

SR:7894 The system shall restrict the maximum number of concurrent sessions that belong to the same user.

SR:7895 The system shall enforce, by default, a limit of [assignment: default number] sessions per user.

Suggested Security Design Patterns

Authenticated Session

An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.

Directed Session

The *Directed Session* pattern ensures that users will not be able to skip

around within a series of Web pages. The system will not expose multiple URLs but instead will maintain the current page on the server. By guaranteeing the order in which pages are visited, the developer can have confidence that users will not undermine or circumvent security checkpoints.

Validated Transaction

The *Validated Transaction* pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

Session locking and termination (FTA_SSL)

Define requirements for the system to provide the capability for system-initiated and user-initiated locking, unlocking, and termination of interactive sessions.

SR:7897 The system shall lock an interactive session after [assignment: time interval of user inactivity] by: a) clearing or overwriting display devices, making the current contents unreadable; b) disabling any activity of the user's data access/display devices other than unlocking the session.

SR:7898 The system shall require the following events to occur prior to unlocking the session: [assignment: events to occur].

SR:7899 The system shall allow user-initiated locking of the user's own interactive session, by: a) clearing or overwriting display devices, making the current contents unreadable; b) disabling any activity of the user's data access/display devices other than unlocking the session.

SR:7900 The system shall require the following events to occur prior to unlocking the session: [assignment: events to occur].

SR:7901 The system shall terminate an interactive session after a [assignment: time interval of user inactivity].

SR:7902 The system shall allow user-initiated termination of the user's own interactive session.

Suggested Security Design Patterns

Authenticated Session

An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.

Directed Session

The *Directed Session* pattern ensures that users will not be able to skip around within a series of Web pages. The system will not expose multiple URLs but instead will maintain the current page on the server. By guaranteeing the order in which pages are visited, the developer can have confidence that users will not undermine or circumvent security checkpoints.

Network Address Blacklist

A network address blacklist is used to keep track of network addresses (IP addresses) that are the sources of hacking attempts and other mischief. Any requests originating from an address on the blacklist are simply ignored. Ideally, breaking attempts should be investigated and prosecuted, but there are simply too many such events to address them all. The *Network Address Blacklist* pattern represents a pragmatic alternative.

System access banners (FTA_TAB)

Define requirements to display a configurable advisory warning message to users regarding the appropriate use of the system.

SR:7903 Before establishing a user session, the system shall display an advisory warning message regarding unauthorised use of the system.

Suggested Security Design Patterns

System access history (FTA_TAH)

Define requirements for the system to display to a user, upon successful session establishment, a history of successful and unsuccessful attempts to access the user's account.

SR:7904 Upon successful session establishment, the system shall display the [selection: date, time, method, location] of the last successful session establishment to the user.

SR:7905 Upon successful session establishment, the system shall display the

[selection: date, time, method, location] of the last unsuccessful attempt to session establishment and the number of unsuccessful attempts since the last successful session establishment.

SR:7906 The system shall not erase the access history information from the user interface without giving the user an opportunity to review the information.

Suggested Security Design Patterns

System session establishment (FTA_TSE)

Define requirements to deny a user permission to establish a session with the system.

SR:7907 The system shall be able to deny session establishment based on [assignment: attributes].

Suggested Security Design Patterns

Account Lockout

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. However, password-guessing attacks have proven to be very successful at discovering poorly chosen, weak passwords. Worse, the Web environment lends itself to high- speed, anonymous guessing attacks. Account lockout protects customer accounts from automated password-guessing attacks, by implementing a limit on incorrect password attempts before further attempts are disallowed.

Authenticated Session

An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.

Directed Session

The *Directed Session* pattern ensures that users will not be able to skip around within a series of Web pages. The system will not expose multiple URLs but instead will maintain the current page on the server. By guaranteeing the order in which pages are visited, the developer can have confidence

that users will not undermine or circumvent security checkpoints.

E.10 Security Audit

Security auditing involves recognising, recording, storing, and analysing information related to security relevant activities (i.e. activities controlled by the system).

Security audit automatic response (FAU_ARP)

The response to be taken in case of detected events indicative of a potential security violation.

SR:7715 The system shall take [assignment: list of actions] upon detection of a potential security violation.

Suggested Security Design Patterns

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Security audit data generation (FAU_GEN)

Define requirements for recording the occurrence of security relevant events that take place under system control. This family identifies the level of auditing, enumerates the types of events that shall be auditable by the system, and identifies the minimum set of audit-related information that should be provided within various audit record types.

SR:7716 The system shall be able to generate an audit record of the following auditable events: a) Start-up and shutdown of the audit functions; b) All auditable events for the [selection, choose one of: minimum, basic, detailed, not specified] level of audit; and c) [assignment: other specifically defined auditable events].

SR:7718 For audit events resulting from actions of identified users, the system shall be able to associate each auditable event with the identity of the user that caused the event.

Suggested Security Design Patterns

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Security audit analysis (FAU_SAA)

Define requirements for automated means that analyse system activity and audit data looking for possible or real security violations. This analysis may work in support of intrusion detection, or automatic response to a potential security violation.

SR:7719 The system shall be able to apply a set of rules in monitoring the audited events and based upon these rules indicate a potential violation of the enforcement of the security requirement.

SR:7720 The system shall be able to maintain profiles of system usage, where an individual profile represents the historical patterns of usage performed by the member(s) of [assignment: the profile target group].

SR:7721 The system shall be able to maintain a suspicion rating associated with each user whose activity is recorded in a profile, where the suspicion rating represents the degree to which the user's current activity is found inconsistent with the established patterns of usage represented in the profile.

SR:7722 The system shall be able to indicate a possible violation of the enforcement of the SFRs when a user's suspicion rating exceeds the following threshold conditions [assignment: conditions under which anomalous activity is reported by the system].

SR:7723 The system shall be able to maintain an internal representation of the following signature events [assignment: a subset of system events] that may indicate a violation of the enforcement of the SFRs.

SR:7724 The system shall be able to compare the signature events against the record of system activity discernible from an examination of [assignment: the information to be used to determine system activity].

SR:7725 The system shall be able to indicate a potential violation of the enforcement of the SFRs when a system event is found to match a signature event that indicates a potential violation of the enforcement of the SFRs.

Suggested Security Design Patterns

Account Lockout

Passwords are the only approach to remote user authentication that has gained widespread user acceptance. However, password-guessing attacks have proven to be very successful at discovering poorly chosen, weak passwords. Worse, the Web environment lends itself to high- speed, anonymous guessing attacks. Account lockout protects customer accounts from automated password-guessing attacks, by implementing a limit on incorrect password attempts before further attempts are disallowed.

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

Network Address Blacklist

A network address blacklist is used to keep track of network addresses (IP addresses) that are the sources of hacking attempts and other mischief. Any

requests originating from an address on the blacklist are simply ignored. Ideally, breaking attempts should be investigated and prosecuted, but there are simply too many such events to address them all. The *Network Address Blacklist* pattern represents a pragmatic alternative.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

Security audit review (FAU_SAR)

Define the requirements for audit tools that should be available to authorised users to assist in the review of audit data.

SR:7729 The system shall provide [assignment: authorised users] with the capability to read [assignment: list of audit information] from the audit records.

SR:7730 The system shall provide the audit records in a manner suitable for the user to interpret the information.

SR:7731 The system shall prohibit all users read access to the audit records, except those users that have been granted explicit read-access.

SR:7732 The system shall provide the ability to apply [assignment: methods of selection and/or ordering] of audit data based on [assignment: criteria with logical relations].

Suggested Security Design Patterns

Security audit event selection (FAU_SEL)

Define requirements to select the set of events to be audited during operation from the set of all auditable events.

SR:7733 The system shall be able to select the set of audited events from the set of all auditable events based on the following attributes: a) [selection: object identity, user identity, subject identity, host identity, event type] b)

[assignment: list of additional attributes that audit selectivity is based upon]

Suggested Security Design Patterns

Security audit event storage (FAU_STG)

Define the requirements for the system to be able to create and maintain a secure audit trail. Stored audit records refers to those records within the audit trail, and not the audit records that have been retrieved (to temporary storage) through selection.

SR:7734 The system shall protect the stored audit records in the audit trail from unauthorised deletion.

SR:7735 The system shall be able to [selection, choose one of: prevent, detect] unauthorised modifications to the stored audit records in the audit trail.

SR:7736 The system shall [assignment: actions to be taken in case of possible audit storage failure] if the audit trail exceeds [assignment: pre-defined limit].

Suggested Security Design Patterns

Minefield

The *Minefield* pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.

Secure Assertion

The *Secure Assertion* pattern sprinkles application-specific sanity checks throughout the system. These take the form of *assertions* ? a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.

E.11 Trusted path/channels

This group provides elements for a trusted communication path between users and the system, and for a trusted communication channel between the system and other trusted IT products.

Inter-system trusted channel (FTP_ITC)

Define requirements for the creation of a trusted channel between the system and other trusted IT products for the performance of security critical operations. This family should be included whenever there are requirements for the secure communication of user or system data between the system and other trusted IT products.

SR:7908 The system shall provide a communication channel between itself and another trusted IT product that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from modification or disclosure.

SR:7909 The system shall permit [selection: the system, another trusted IT product] to initiate communication via the trusted channel.

SR:7910 The system shall initiate communication via the trusted channel for [assignment: list of functions for which a trusted channel is required].

Suggested Security Design Patterns

Trusted path (FTP_TRP)

Define the requirements to establish and maintain trusted communication to or from users and the system. A trusted path may be required for any security-relevant interaction. Trusted path exchanges may be initiated by a user during an interaction with the system, or the system may establish communication with the user via a trusted path.

SR:7911 The system shall provide a communication path between itself and [selection: remote, local] users that is logically distinct from other communication paths and provides assured identification of its end points and protection of the communicated data from [selection: modification, disclosure, [assignment: other types of integrity or confidentiality violation]].

SR:7912 The system shall permit [selection: the system, local users, remote users] to initiate communication via the trusted path.

SR:7913 The system shall require the use of the trusted path for [selection: initial user authentication, [assignment: other services for which trusted path is required]].